

# CONFERENCE PROCEEDINGS

20 – 23 SEPTEMBER 2016

PETROS M. NOMIKOS  
CONFERENCE CENTRE  
SANTORINI, GREECE

<http://cig16.image.ece.ntua.gr>

#ieeecig16

#cig16

Sponsored by



IEEE  
Computational  
Intelligence  
Society



## Preface

On behalf of the organizing committee of the 2016 IEEE Conference on Computational Intelligence and Games (CIG2016), we welcome you to Santorini for our 12<sup>th</sup> meeting. The conference program includes three keynote talks from the industry and academia: Anton Nijholt (NL) from the University of Twente, Tom Schaul (LU) from Google DeepMind, and Innes McKendrick (UK) from Hello Games. In addition the program includes two tutorials which will be presented during the first day of the conference: Antonios Liapis and Michael Cook on Designing Games around AI and Christoph Salge, Tobias Mahlmann and Christian Guckelsberger on Intrinsic motivation in general game-playing and NPCs.

Moreover, we have eight exciting competitions: StarCraft, Fighting Game, Geometry Friends, The General Video Game, Angry Birds, Ms. Pacman Vs Ghosts, Artificial Text Adventurer, and Visual Doom. In addition, a special session on Game Analytics, organized by Anders Drachen, Rafet Sifa, and Julian Runge is included in the program.

In total we received 113 papers from 30 countries. All papers were peer reviewed by at least three domain experts and 43/92 (acceptance rate: 46.7%) were accepted for oral presentation and publication in the proceedings. The program also includes 6 vision papers, 4 competition papers, 4 demo papers and 11 posters.

We hope that you will enjoy the conference and your stay in Santorini. We would also like to thank ICCS for its sponsorship and the program committee members for their contribution to making this happen.

September 15, 2016

Kostas Karpouzis,  
General Chair  
Gillian Smith and Georgios N. Yannakakis,  
Program chairs  
Tommy Thompson,  
Proceedings chair



## Keyword Index

2016	419
adaptive user modelling	31
affective computing	233
AI	294
AI-assisted game design tools	467
alpha beta algorithm	294
altruistic punishment	9
Analytics	497
Angry Birds	178
applied games	485
Apriori	102
archetype analysis	262
artificial player	294
artificial life	302
artificial player	278
assessment	485
Asymmetric video games	333
Automated game balancing	333
Automated play testing	372
autonomous agent	302
Bad Move Detection	395
balancing	278
Behavioral Analytics	142
behavioral profiling	262
believability	134, 247
Blackjack	158
car racing games	31
case-based reasoning	247
CCG	217
churn prediction	325
CIG	419
classification	94
cluster analysis	262
clustering	39
collectible card games	286
combat modeling	118
Communication	419
Competition	419
Computational Creativity	372, 427
computer go	186
Constrained Optimization	166

constructive primitive	86
Contextual bandit	387
Convolutional neural network	493
cortical learning algorithm	302
CPPNs	349
customised track design	31
Deathmatch	166
Deep Convolutional Neural Networks	380
deep learning	201
Deep learning application	493
deep reinforcement learning	341, 483
Destiny	325
digital games	325
Divergent Search	166
Dungeon	78
Dynamic Environments	241
emotion modeling	193
empowerment	150
evolutionary algorithms	286
Evolutionary Computation	349, 357
evolutionary optimization	278
Exergaming	16
Expressivity Analysis	78
First Person Shooter	166
first-person environment	483
first-person perspective games	341
first-person shooter	341
forward pruning	294
Free to Play	497
free-to-play	39
game AI	247
Game Analytics	254
game analytics	39, 62, 262
game data science	39
game design	270
game generation	467
Game of Go	395
Games	102
games	94
gaussian mixed models	262
General AI	150
general artificial intelligence	467
General game playing	150
General Game Playing	309

general gameplaying	467
general video game AI	47
General Video Game Playing	435, 443
general video game playing	47, 94, 317, 459
Generalized Rapid Action Value Estimation	309
Generative Grammars	78
Generic Player Modelling	209
Genetic Algorithm	23
Genetic Algorithms	417
Genetic programming	158
Ghost	419
Greedy search	158
GVG-AI	435
GVG-AI competition	402
GVGAI	47
heal	118
Hearthstone	217
hearthstone	286
Heuristic Search	380
Heuristics	158
hidden Markov model	325
Hierarchical Task Network Planning	1
hierarchical temporal memory	302
Human Decision Modeling	409
Human gesture classification	16
hybrid algorithms	459
hyper-heuristics	94, 459
Image conversion	493
Influence Map	364
information theory	150
Informed Sampling	70
Interactive Evolution	349, 357
Intrinsic motivation	451
Intrinsic Motivation	150
k-maxoids	262
k-means	262
keepaway soccer	201
Kinect	16
Learning	158
level design	134
Level design	372
level generation	467, 495
LinUCB	387
Longitudinal data	254

Machine Learning	70
Machine learning	16
machine learning	233
Machine-Learning	395
macro-actions	47
massively multi-player online game	325
MCTS	47, 402
Micro	364
minimax search	294
MMOG	325
Mobile	497
mobile games	39, 62, 126
Mobile gaming	372
Monte Carlo	55
Monte Carlo Tree Search	47, 70, 380, 387
monte carlo tree search	94, 186, 459
Monte-Carlo Tree Search	309, 435, 443
Motion Planning	241
Ms. Pac-Man	419
multi-agent learning	201
Multi-agent systems	174
Multi-armed Bandit problem	387
Multi-Objective Optimization	23
Nash	55
NEAT	349, 357
neocortex layers	302
Netrunner	102
Neural Networks	349, 357
neural networks	201, 341
Non-player character	150
non-player characters	193, 225
NPC behaviors	357
online games	325
online level generation	86
open loop control	317
Open Loop Search	443
Opponent based learning	174
options	47
Pac-Man	419
Partial Observability	419
Partially Observable Game	55
pathfinding	225
Perception Simulation	110
personalised gaming	31

personality modeling	193
Phantom Go	55
physiological signals	233
Plan Reuse	1
planning	225
Planning	110
player behavioral analysis	39
Player Education	395
player experience modeling	467
player modeling	134, 485
Player Modeling	409
Player Modelling	209
POMDP	110
Potential Fields	23, 364
Preference Learning	209
preference learning	134
Procedural Content Generation	78, 166, 349, 417
Procedural content generation	178, 451, 493
procedural content generation	86, 270, 475, 495
process model	278
public goods game	9
Pursuit-Evasion	241
Pysics-based	178
q-learning	201
Random Forests	16
rankings	317
Rapid Action Value Estimation	309
Real-Time Strategy Games	70, 364, 380
Recommendation System	409
reinforcement learning	201, 483
Reinforcement learning	451
Reinforcement Learning	443
Representation Learning	142
retention	126, 325
Retention Prediction	142
Roguelike	78
rolling horizon evolutionary algorithms	459
Rule-Mining	102
scaling	402
Search	427
search-based pcg	495
Search-based PCG	166
Security	217
semantics in games	475
simulation balancing	186

Skeletal data	16
sleep	118
social dilemmas	9
Social learning	174
Sonification	417
Spatio-temporal Data Mining	142
StarCraft	23
State Evaluation	380
Stealth game	16
Story Generation	110
Storytelling	110
strong AI	294
Super Mario	357
Super Mario Bros.	86
Supply Chain Game	409
support vector machines	134
Surprise	166
survival	126
survival analysis	62
Survival Analysis	254
Survival Horror	417
synthetic agents	174
targeting	118
Tensor Factorization	142
time series	39
tools	270
Tower defense games	333
tragedy of the commons	9
Transfer Learning	209
Turn based strategy game	294
Tutorials	158
Unreal Tournament 3	166
unsupervised learning	39
user activity	126
user retention	62
Video Games	1, 497
Video games	178
video games	341
Visibility	241
Vision paper	427
visual learning	341
visual-based reinforcement learning	341
vizdoom	483
Weapon	166

web of data

475

## Author Index

Abdelkader, Ahmed	241
Abou-Zleikha, Mohamed	209
Agureikin, Aleksandr	278
Airola, Antti	62, 126
Allart, Thibault	254
Allebosch, Gianni	16
Anokhin, Alexander	278
Aversa, Davide	225
Bakkes, Sander	333
Bakkes, Sander C.J.	47
Barriga, Nicolas A.	380
Bauckhage, Christian	142
Beau, Philipp	333
Beyer, Marlene	278
Buro, Michael	380
Bursztein, Elie	217
Camilleri, Elizabeth	134
Carvalho, David B.	110
Cazenave, Tristan	55
Chamberland, Cindy	233
Chen, Ke	86
Chu, Chun-Yin	443
Clerico, Andrea	233
Clua, Esteban	110
Colton, Simon	150, 270, 372
Cook, Michael	270
Cowling, Peter	102
De Mesentier Silva, Fernando	158
de Waard, Maarten	47
Deboeverie, Francis	16
Demiris, Yiannis	31
Devlin, Sam	102
Dewitt, Tyler	364
Dingli, Alexiei	134
Drachen, Anders	142, 262, 325, 497
Falk, Tiago H.	233
Gagnon, Jean-Christophe	233
García Sánchez, Pablo	286
Gaudl, Swen	372



Georgiou, Theodosis	31
Gow, Jeremy	270
Graf, Tobias	186
Gravina, Daniele	166
Gray, Chester	262
Green, James	262
Greenwood, Garrison	9
Guckelsberger, Christian	150
Guilloux, Agathe	254
Guitart, Anna	39
Harada, Tomohiro	443
Harik, Elie	262
Harteveld, Casper	409
Heikkonen, Jukka	62, 126
Henriksen, Lars	485
Hess, Andy	380
Hitchens, Michael	325
Hodge, Victoria	102
Holmgård, Christoffer	485
Horn, Hendrik	459
Ikeda, Kokolo	294, 395
Isaksen, Aaron	158
Ito, Suguru	443
Jackson, Philip L.	233
Jaśkowski, Wojciech	201, 341
Kaeli, David	409
Kalles, Dimitris	174
Kaneko, Tomoyuki	387
Karavolos, Daniel	78, 495
Kempka, Michał	341
Kersjes, Hanneke	193
Kim, Kyung-Joong	483, 493
Kiourt, Chairi	174
Klabjan, Diego	262
Kurek, Mateusz	201
Köstler, Harald	23
Laenger, Christoph	278
Levieux, Guillaume	254
Liang, Chisheng	409
Liapis, Antonios	78, 166, 417, 475, 495
Liu, Jialin	55
Liu, Siming	364
Lopes, Phil	417

Louis, Sushil	364
Lu, Patty	262
Lucas, Simon	317, 419
Mandai, Yusaku	387
Mendes Da Silva, Andre	94
Merelo, Jj	286
Michon, Pierre-Emmanuel	233
Mora, Antonio	286
Mozgovoy, Maxim	247
Natkin, Stephane	254
Nealen, Andy	94, 158
Nelson, Mark J.	372, 402
Nolte, Felix	278
Ojeda, Cesar	142
Olsen, Jeppeh	357
Ontañón, Santiago	70
Paes, Aline	110
Pahikkala, Tapio	62, 126
Parent, Mark	233
Park, Hyunsoo	483
Patrascu, Cristinel	349
Perez, Diego	317, 419, 459
Perianez, Africa	39
Pflanzl, Nicolas	278
Philips, Wilfried	16
Pierfitte, Michel	254
Platzner, Marco	186
Powley, Edward J.	372
Preuss, Mike	278, 459
Purgina, Marina	247
Raffe, William	325
Renka, Marcel	278
Renz, Jochen	178
Rieger, Martin	278
Risi, Sebastian	349, 357
Roegiers, Sanne	16
Rojers, Diederik M.	47
Ross, Nicholas	497
Runc, Grzegorz	341
Runge, Julian	497
Saas, Alain	39
Sacco, Owen	475

Salge, Christoph	150
Samothrakis, Spyridon	317
Sardina, Sebastian	225
Sato, Naoyuki	294, 395
Saunders, Rob	372
Schaul, Tom	317
Schmitt, Jonas	23
Schuster, Torsten	435
Sephton, Nick	102
Shaker, Noor	209, 451
Shi, Peizhi	86
Sifa, Rafet	142, 262, 325, 497
Sironi, Chiara F.	309, 435
Slaven, Nicholas	102
Soemers, Dennis J. N. J.	1, 435
Spronck, Pieter	193
Squillero, Giovanni	286
Srikanth, Sridev	142
Stanescu, Marius	380
Stephenson, Matthew	178
Sun, Yifan	409
Sungur, Ali Kaan	302
Surer, Elif	302
Sutherland, Steven	409
Sørensen, Patrikk	357
Tamassia, Marco	325
Teytaud, Fabien	55
Teytaud, Olivier	55
Thawonmas, Ruck	443
Toczek, Jakub	341
Togelius, Julian	94, 158, 317, 467, 485
Tonda, Alberto	286
Tremblay, Sébastien	233
Umarov, Iskander	247
Vassos, Stavros	225
Veelaert, Peter	16
Ventura, Dan	427
Verbrugge, Clark	118
Viennot, Simon	395
Viljanen, Markus	62, 126
Volz, Vanessa	278, 459
Williams, Piers	419
Winands, Mark H. M.	1, 309, 435
Winterberg, Jonas	278

Wydmuch, Marek	341
Xu, Shuo	118
Yannakakis, Georgios	475
Yannakakis, Georgios N.	78, 134, 166, 417, 467, 495
Yoo, Byungho	493
Zambetta, Fabio	325

## Program Committee

Samad Ahmadi  
Peter Andras  
Daniel Ashlock  
Stylianos Asteriadis

Byung-Chull Bae  
Sander Bakkes  
Panagiotis Bamidis

Christian Bauckhage  
Manuel Bedia  
Amit Benbassat  
Rafael Bidarra  
Alan Blair  
Bruno Bouzy  
Joost Broekens  
Joseph Alexander Brown  
Cameron Browne  
Paolo Burelli  
Michal Bída

Alessandro Canossa  
Tristan Cazenave  
Darryl Charles  
Yun-Gyung Cheong  
Sung-Bae Cho  
Simon Colton

Michael Cook  
Vincent Corruble  
Peter Cowling  
Steve Dahlskog  
José Valente De Oliveira  
Joerg Denzinger  
Sam Devlin  
Anders Drachen  
Mirjam Palosaari Eladhari  
Antonio J. Fernández Leiva

Antonio Fernández-Ares  
Allan Fowler  
Marcus Gallagher  
Jeremy Gow  
Kazjon Grace  
Daniele Gravina  
Garry Greenwood

De Montfort University  
Keele University  
University of Guelph  
Department of Knowledge Engineering, University of Maastricht  
Hongik University  
University of Amsterdam  
Lab of Medical Physics, Medical School, Aristotle University of Thessaloniki  
Fraunhofer IAIS  
University of Zaragoza  
Ben-Gurion University of the Negev  
Delft University of Technology  
University of New South Wales  
Paris Descartes University  
TU Delft  
Innopolis University  
QUT  
Aalborg University Copenhagen  
Charles University in Prague, Faculty of Mathematics and Physics  
Northeastern University  
LAMSADE Université Paris Dauphine  
University of Ulster  
SKKU  
Yonsei University  
Department of Computing, Goldsmiths College, University of London  
Goldsmiths College, University of London  
LIP6, Université Pierre et Marie Curie (Paris 6)  
University of York  
Malmö University  
Universidade do Algarve  
Department of Computer Science, University of Calgary  
University of York  
Game Analytics  
University of Malta  
Departamento de Lenguajes y Ciencias de la Computación.  
Universidad de Málaga

Kennesaw State University  
University of Queensland  
Goldsmiths, University of London  
University of North Carolina at Charlotte  
Politecnico di Milano  
Portland State University

Hisashi Handa	Kindai University
Tomonori Hashiyama	University of Electro-communications
Erin Hastings	University of Central Florida
Rania Hodhod	Columbus State University
Christoffer Holmgård	IT-University of Copenhagen
Amy K. Hoover	University of Central Florida
Ian Horswill	Northwestern University
Eva Hudlicka	Psychometrix Associates
Hiroyuki Iida	JAIST
Aaron Isaksen	New York University
Hisao Ishibuchi	Osaka Prefecture University
Mikhail Jacob	Georgia Institute of Technology
Wojciech Jaskowski	Poznan University of Technology
Anna Jordanous	University of Kent
Daniel Karavolos	University of Malta
Konstantinos Karpouzis	ICCS
Graham Kendall	University of Nottingham
Kyung-Joong Kim	Sejong University
Stefanos Kollias	Professor
Jan Koutnik	IDSIA
Pier-Luca Lanzi	Politecnico di Milano
John Levine	University of Strathclyde
Antonios Liapis	University of Malta
Chong-U Lim	Massachusetts Institute of Technology
Siming Liu	University of Nevada, Reno
Daniele Loiacono	Politecnico di Milano
Phil Lopes	Institute of Digital Games, University of Malta
Sushil Louis	University of Nevada, Reno
Simon Lucas	University of Essex
Ioanna Lykourantzou	Luxembourg Institute of Science and Technology
Dario Maggiorini	University of Milano
Ilias Maglogiannis	University of Piraeus
Tobias Mahlmann	IT University of Copenhagen
Manolis Maragoudakis	University of the Aegean
Carlos Martinho	INESC-ID and Instituto Superior Técnico, Technical University of Lisbon
Helmut Mayer	University of Salzburg
Jacek Mańdziuk	Faculty of Mathematics and Information Science Warsaw University of Technology Plac Politechniki 1, 00-661 Warsaw, Poland
Antonio Mora	University of Granada
Phivos Mylonas	National Technical University of Athens
Mark J. Nelson	Anadrome Research
Santiago Ontañón	Drexel University
Jose-Maria Pena	Universidad Politécnica de Madrid
Diego Perez	University of Essex
Rui Prada	INESC-ID and Instituto Superior Técnico, Universidade de Lisboa

Mike Preuss	University of Dortmund
Robert Reynolds	Wayne State University
Sebastian Risi	IT University of Copenhagen
Guenter Rudolph	TU Dortmund University
Thomas Runarsson	University of Iceland
Owen Sacco	Digital Enterprise Research Institute (DERI)
Yago Saez	University Carlos III of Madrid
Christoph Salge	University of Hertfordshire
Spyridon Samothrakis	University of Essex
Tom Schaul	New York University
Björn Schuller	University of Passau / Imperial College London
Noor Shaker	Center for Computer Games Research, IT University of Copenhagen
Moshe Sipper	Ben-Gurion University of the Negev
Adam M. Smith	University of California Santa Cruz
Gillian Smith	Northeastern University
Pieter Spronck	Tilburg University
Giovanni Squillero	Politecnico di Torino
Nathan Sturtevant	University of Denver
Gabriel Synnaeve	Grenoble University (LIG)
Anastasios Tefas	Aristotle University of Thessaloniki
Ruck Thawonmas	Ritsumeikan University
Tommy Thompson	Anglia Ruskin University
Julian Togelius	New York University
Mike Treanor	American University
Stavros Vassos	Sapienza University of Rome
Manolis Wallace	University of Peloponnese
Ben Weber	University of California, Santa Cruz
Mark H. M. Winands	Maastricht University
Kevin Wong	Murdoch University
I-Chen Wu	National Chiao Tung Univ.
Georgios Yannakakis	Institute of Digital Games, University of Malta
Stefanos Zafeiriou	Imperial College London
Fabio Zambetta	RMIT University
Alexander Zook	Georgia Institute of Technology

## Additional Reviewers

Athanasiadis, Christos  
Azaria, Itay  
Cerny, Martin  
Dann, Michael  
Fernández Ares, Antonio  
García Sánchez, Pablo  
Gaudl, Swen  
Ghaleb, Esam  
Guckelsberger, Christian  
Harada, Tomohiro  
Lara-Cabrera, Raul  
Moerland, Thomas  
Nogueira-Collazo, Mariela  
Pepels, Tom  
Plch, Tomas  
Powley, Edward J.  
Prada, Rui  
Raffe, William  
Spanakis, Gerasimos  
Spyrou, Evaggelos  
Sánchez-Ruiz, Antonio A.



## Table of Contents

Hierarchical Task Network Plan Reuse for Video Games .....	1
<i>Dennis J. N. J. Soemers and Mark H. M. Winands</i>	
Altruistic Punishment Can Help Resolve Tragedy of the Commons Social Dilemmas .....	9
<i>Garrison Greenwood</i>	
Human Gesture Classification by Brute-Force Machine Learning for Exergaming in Physiotherapy .....	16
<i>Francis Deboeverie, Sanne Roegiers, Gianni Allebosch, Peter Veelaert and Wilfried Philips</i>	
A Multi-Objective Genetic Algorithm for Simulating Optimal Fights in StarCraft II .....	23
<i>Jonas Schmitt and Harald Köstler</i>	
Personalised Track Design in Car Racing Games .....	31
<i>Theodosios Georgiou and Yiannis Demiris</i>	
Discovering Playing Patterns: Time Series Clustering of Free-To-Play Game Data .....	39
<i>Alain Saas, Anna Guitart and Africa Perianez</i>	
Monte Carlo Tree Search with Options for General Video Game Playing .....	47
<i>Maarten de Waard, Diederik M. Roijers and Sander C.J. Bakkes</i>	
Learning opening books in partially observable games: using random seeds in Phantom Go	55
<i>Tristan Cazenave, Jialin Liu, Fabien Teytaud and Olivier Teytaud</i>	
Modelling User Retention in Mobile Games .....	62
<i>Markus Viljanen, Antti Airola, Tapio Pahikkala and Jukka Heikkonen</i>	
Informed Monte Carlo Tree Search for Real-Time Strategy Games .....	70
<i>Santiago Ontañón</i>	
Evolving Missions to Create Game Spaces .....	78
<i>Daniel Karavolos, Antonios Liapis and Georgios N. Yannakakis</i>	
Online Level Generation in Super Mario Bros via Learning Constructive Primitives .....	86
<i>Peizhi Shi and Ke Chen</i>	
Hyper-heuristic general video game playing .....	94
<i>Andre Mendes Da Silva, Julian Togelius and Andy Nealen</i>	
Using Association Rule Mining to Predict Opponent Deck Content in Android: Netrunner	102
<i>Nick Sephton, Peter Cowling, Sam Devlin, Victoria Hodge and Nicholas Slaven</i>	
Planning Social Actions Through the Others' Eyes for Emergent Storytelling .....	110
<i>David B. Carvalho, Esteban Chua and Aline Paes</i>	
Heuristics for Sleep and Heal in Combat .....	118
<i>Shuo Xu and Clark Verbrugge</i>	
User Activity Decay in Mobile Games Determined by Simple Differential Equations? .....	126
<i>Markus Viljanen, Antti Airola, Tapio Pahikkala and Jukka Heikkonen</i>	

Platformer Level Design for Player Believability .....	134
<i>Elizabeth Camilleri, Georgios N. Yannakakis and Alexiei Dingli</i>	
Predicting Retention in Sandbox Games with Tensor Factorization-based Representation Learning .....	142
<i>Rafet Sifa, Sridev Srikanth, Anders Drachen, Cesar Ojeda and Christian Bauckhage</i>	
Intrinsically Motivated General Companion NPCs via Coupled Empowerment Maximisation .....	150
<i>Christian Guckelsberger, Christoph Salge and Simon Colton</i>	
Generating Heuristics for Novice Players .....	158
<i>Fernando De Mesentier Silva, Aaron Isaksen, Julian Togelius and Andy Nealen</i>	
Constrained Surprise Search for Content Generation.....	166
<i>Daniele Gravina, Antonios Liapis and Georgios N. Yannakakis</i>	
Using Opponent Models to Train Inexperienced Synthetic Agents in Social Environments ..	174
<i>Chairi Kiourt and Dimitris Kalles</i>	
Procedural Generation of Complex Stable Structures for Angry Birds Levels.....	178
<i>Matthew Stephenson and Jochen Renz</i>	
Monte-Carlo Simulation Balancing Revisited .....	186
<i>Tobias Graf and Marco Platzner</i>	
Modeling Believable Game Characters.....	193
<i>Hanneke Kersjes and Pieter Spronck</i>	
Heterogeneous Team Deep Q-Learning in Low-Dimensional Multi-Agent Environments ...	201
<i>Mateusz Kurek and Wojciech Jaśkowski</i>	
Transfer Learning for Cross-Game Prediction of Player Experience .....	209
<i>Noor Shaker and Mohamed Abou-Zleikha</i>	
I am a legend: hacking Hearthstone using statistical learning methods.....	217
<i>Elie Bursztein</i>	
Pruning and Preprocessing Methods for Inventory-Aware Pathfinding .....	225
<i>Davide Aversa, Sebastian Sardina and Stavros Vassos</i>	
Biometrics and classifier fusion to predict the fun-factor in video gaming .....	233
<i>Andrea Clerico, Cindy Chamberland, Mark Parent, Pierre-Emmanuel Michon, Sébastien Tremblay, Tiago H. Falk, Jean-Christophe Gagnon and Philip L. Jackson</i>	
Recovering Visibility and Dodging Obstacles in Pursuit-Evasion Games.....	241
<i>Ahmed Abdelkader</i>	
Believable Self-Learning AI for World of Tennis .....	247
<i>Maxim Mozgovoy, Iskander Umarov and Marina Purgina</i>	
Design Influence on Player Retention : A Method Based on Time Varying Survival Analysis .....	254
<i>Thibault Allart, Guillaume Levieux, Michel Pierfitte, Agathe Guilloux and Stephane Natkin</i>	

Guns and Guardians: Comparative Cluster Analysis and Behavioral Profiling in Destiny ..	262
<i>Anders Drachen, James Green, Chester Gray, Elie Harik, Patty Lu, Rafet Sifa and Diego Klabjan</i>	
Towards The Automatic Optimisation Of Procedural Content Generators.....	270
<i>Michael Cook, Jeremy Gow and Simon Colton</i>	
An Integrated Process for Game Balancing.....	278
<i>Marlene Beyer, Aleksandr Agureikin, Alexander Anokhin, Christoph Laenger, Felix Nolte, Jonas Winterberg, Marcel Renka, Martin Rieger, Nicolas Pflanzl, Mike Preuss and Vanessa Volz</i>	
Evolutionary Deckbuilding in the Collectible Card Game HearthStone.....	286
<i>Pablo García Sánchez, Alberto Tonda, Giovanni Squillero, Antonio Mora and Jj Merelo</i>	
Three Types of Forward Pruning Techniques to Apply Alpha Beta Algorithm to Turn-Based Strategy Game .....	294
<i>Naoyuki Sato and Kokoro Ikeda</i>	
Voluntary Behavior on Cortical Learning Algorithm Based Agents.....	302
<i>Ali Kaan Sungur and Elif Surer</i>	
Comparison of Rapid Action Value Estimation Variants for General Game Playing .....	309
<i>Chiara F. Sironi and Mark H. M. Winands</i>	
Analyzing the Robustness of General Video Game Playing Agents.....	317
<i>Diego Perez, Spyridon Samothrakis, Julian Togelius, Tom Schaul and Simon Lucas</i>	
Predicting Player Churn in Destiny: A Hidden Markov Models Approach to Predicting Player Departure in a Major Online Game .....	325
<i>Marco Tamassia, William Raffae, Rafet Sifa, Anders Drachen, Fabio Zambetta and Michael Hitchens</i>	
Automated Game Balancing of Asymmetric Video Games.....	333
<i>Philipp Beau and Sander Bakkes</i>	
ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning .....	341
<i>Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek and Wojciech Jaśkowski</i>	
Artefacts: Minecraft meets Collaborative Interactive Evolution .....	349
<i>Cristinel Patrascu and Sebastian Risi</i>	
Breeding a Diversity of Super Mario Behaviors Through Interactive Evolution .....	357
<i>Patrik Sørensen, Jeppe Olsen and Sebastian Risi</i>	
Evolving Micro for 3D Real-Time Strategy Games.....	364
<i>Tyler Dewitt, Sushil Louis and Siming Liu</i>	
Semi-automated Level Design via Auto-Playtesting for Handheld Casual Game Creation..	372
<i>Edward J. Powley, Simon Colton, Swen Gaudl, Rob Saunders and Mark J. Nelson</i>	
Evaluating Real-Time Strategy Game States Using Convolutional Neural Networks .....	380
<i>Marius Stanescu, Nicolas A. Barriga, Andy Hess and Michael Buro</i>	

Improved LinUCT and Its Evaluation on Incremental Random-Feature Tree .....	387
<i>Yusaku Mandai and Tomoyuki Kaneko</i>	
Detection and Labeling of Bad Moves for Coaching Go .....	395
<i>Kokolo Ikeda, Simon Viennot and Naoyuki Sato</i>	
Investigating Vanilla MCTS Scaling on the GVG-AI Game Corpus .....	402
<i>Mark J. Nelson</i>	
Modeling Player Decisions in a Supply Chain Game .....	409
<i>Yifan Sun, Chisheng Liang, Steven Sutherland, Casper Hartevelde and David Kaeli</i>	
Sonancia: a Multi-Faceted Generator for Horror .....	417
<i>Phil Lopes, Antonios Liapis and Georgios N. Yannakakis</i>	
Ms. Pac-Man Versus Ghost Team CIG 2016 Competition .....	419
<i>Piers Williams, Diego Perez and Simon Lucas</i>	
Beyond Computational Intelligence to Computational Creativity in Games .....	427
<i>Dan Ventura</i>	
Enhancements for Real-Time Monte-Carlo Tree Search in General Video Game Playing...	435
<i>Dennis J. N. J. Soemers, Chiara F. Sironi, Torsten Schuster and Mark H. M. Winands</i>	
Position-based Reinforcement Learning Biased MCTS for General Video Game Playing...	443
<i>Chun-Yin Chu, Suguru Ito, Tomohiro Harada and Ruck Thawonmas</i>	
Intrinsically Motivated Reinforcement Learning: A Promising Framework for Procedural Content Generation .....	451
<i>Noor Shaker</i>	
MCTS/EA Hybrid GVGAI Players and Game Difficulty Estimation .....	459
<i>Hendrik Horn, Vanessa Volz, Diego Perez and Mike Preuss</i>	
General General Game AI .....	467
<i>Julian Togelius and Georgios N. Yannakakis</i>	
A Holistic Approach for Semantic-Based Game Generation .....	475
<i>Owen Sacco, Antonios Liapis and Georgios Yannakakis</i>	
Deep Q-Learning using Redundant Outputs in Visual Doom .....	483
<i>Hyunsoo Park and Kyung-Joong Kim</i>	
Computational Intelligence and Cognitive Performance Assessment Games .....	485
<i>Christoffer Holmgård, Julian Togelius and Lars Henriksen</i>	
Changing Video Game Graphic Styles Using Neural Algorithms .....	493
<i>Byungho Yoo and Kyung-Joong Kim</i>	
Evolving Missions for Dwarf Quest Dungeons .....	495
<i>Daniel Karavolos, Antonios Liapis and Georgios N. Yannakakis</i>	
Stylized facts for Mobile Game Analytics .....	497
<i>Anders Drachen, Nicholas Ross, Julian Runge and Rafet Sifa</i>	

# Hierarchical Task Network Plan Reuse for Video Games

Dennis J. N. J. Soemers and Mark H. M. Winands

Department of Data Science and Knowledge Engineering, Maastricht University

d.soemers@gmail.com, m.winands@maastrichtuniversity.nl

**Abstract**—Hierarchical Task Network Planning is an Automated Planning technique. It is, among other domains, used in Artificial Intelligence for video games. Generated plans cannot always be fully executed, for example due to nondeterminism or imperfect information. In such cases, it is often desirable to re-plan. This is typically done completely from scratch, or done using techniques that require conditions and effects of tasks to be defined in a specific format (typically based on First-Order Logic). In this paper, an approach for Plan Reuse is proposed that manipulates the order in which the search tree is traversed by using a similarity function. It is tested in the *SimpleFPS* domain, which simulates a First-Person Shooter game, and shown to be capable of finding (optimal) plans with a decreased amount of search effort on average when re-planning for variations of previously solved problems.

## I. INTRODUCTION

The problem of deciding what tasks should be executed by an agent in a real-time video game can be addressed in a number of different ways. Commonly used techniques [1], [2] are Finite State Machines, Behavior Trees, Utility-based decision making systems, and Automated Planning. One of the first well-known applications of Automated Planning in video games is in *F.E.A.R.* [3].

Hierarchical Task Network (HTN) Planning [4], [5] is an automated planning technique that has seen some use in video games. HTN Planning has been used to control a team of bots in the game *Unreal Tournament 2004* [6], to generate scripts offline for the game *The Elder Scrolls IV: Oblivion* [7], in serious gaming [8], [9], and in the adversarial real-time strategy game  $\mu$ RTS [10]. Examples of commercial video games that are known to use HTN Planners are *Killzone 3* and *Transformers 3: Fall of Cybertron* [11].

Generated plans cannot always be successfully executed, because HTN Planners are not able to predict the actions of other agents or deal with imperfect information and nondeterminism in the environment without incorporating extensions [12], [13]. Existing systems using HTN Planning in video games typically construct new plans from scratch whenever a plan fails during execution [8]. Planning systems outside the context of video games do the same in some cases [14], but there is also work describing more sophisticated approaches [15]–[20]. These approaches require conditions and effects of tasks on the environment to be explicitly defined in a predefined format (typically based on First-Order Logic).

This paper proposes an approach for reusing old plans of an HTN Planner that does not require conditions and effects

to be explicitly defined, but allows for them to be defined in functions that are essentially black boxes from the point of view of the Planner. The main purpose of reusing plans is to speed up the process of finding a new plan. Another motivation for reusing plans is to increase the likelihood of finding a plan that is similar to a partially executed previous plan. In video games this can reduce the number of times that an agent abruptly changes behavior, and therefore increase the believability of the agent's behavior. It is tested on problems from the *SimpleFPS* [21] domain, using the *Unreal Engine 4 (UE4)* game engine as test environment. *UE4* is the latest version of a commercial and widely-used game engine. Experiments have been carried out to measure the effect of Plan Reuse on the search effort required to find optimal plans. The effect of Plan Reuse on the quality of plans returned when terminating a search process early has also been measured.

The remainder of the paper is structured as follows. Section II provides background information on HTN Planning. In Section III, the implementation of the HTN Planning algorithm is described. The approach used for reusing old plans is described in Section IV. A description of the experiments can be found in Section V. Finally, Section VI concludes the paper and provides ideas for future research.

## II. HIERARCHICAL TASK NETWORK PLANNING

A Hierarchical Task Network (HTN) Planning Problem [4], [5], [22] can be defined as a tuple  $\mathcal{P} = (S, T, \mathcal{O}, \mathcal{M})$ , where  $S$  is the current *World State*,  $T$  is the current *Task Network*,  $\mathcal{O}$  is the set of *Operators*, and  $\mathcal{M}$  is the set of *Methods*. More specifically,  $S$  is a description of the environment in which the agent is located, and should contain all the information that is relevant for the planning process.  $T$  is a collection of *tasks* that need to be accomplished by the agent, where tasks can be constrained to require accomplishment before certain other tasks in the network. A task can be either *primitive*, meaning that it directly corresponds to an action that the agent can execute, or *compound*, meaning that it represents a higher-level plan that needs to be decomposed into a Task Network.  $S$  and  $T$  change during the planning process.

Every Operator  $o \in \mathcal{O}$  represents the execution of a single primitive task  $t_p$ . Given  $S$ ,  $o$  defines conditions that must hold in  $S$  for  $t_p$  to be applicable in  $S$ , defines how  $S$  changes if  $t_p$  is applied (executed by the agent) in  $S$ , and defines a nonnegative cost for applying  $t_p$ . Similarly, every Method  $m \in \mathcal{M}$  represents the execution of a single compound task  $t_c$ .

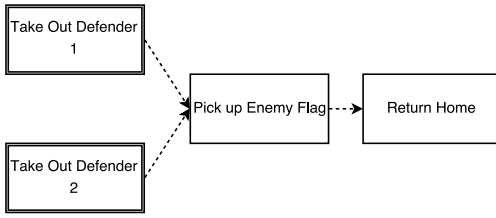


Fig. 1. Example Task Network. An arrow pointing from one task to another task means that the first task is constrained to require execution before the second task (the first task is a predecessor of the second task). Boxes with a double line are compound tasks, boxes with a single line are primitive tasks.

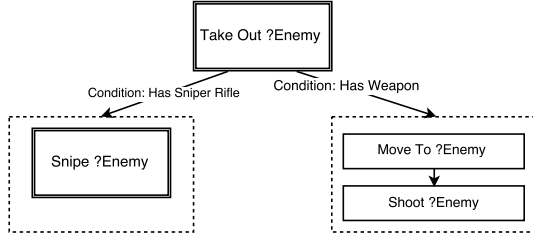


Fig. 2. Example Method. “Take Out ?Enemy” is the compound task for which this method is defined, where “?Enemy” is a variable. The boxes with dashed lines denote task networks. The two arrows pointing away from this compound task point to task networks that the original compound task can be decomposed into, under certain conditions. The left decomposition consists of only a single compound task, whereas the right decomposition consists of an ordered sequence of two primitive tasks.

Given  $S$ ,  $m$  defines all the different Task Networks that  $t_c$  can be decomposed into.  $\mathcal{O}$  and  $\mathcal{M}$  remain constant during the planning process.

An HTN Planning system is expected to take a Problem  $\mathcal{P}$  as input, and produce a valid plan  $\Pi$  as output. A plan  $\Pi$  is a valid plan for  $\mathcal{P}$  if it is an ordered list of primitive tasks that can be obtained by consecutively applying Methods and Operators from  $\mathcal{M}$  and  $\mathcal{O}$  to the tasks in  $T$  in an order that satisfies the constraints of  $T$ , until  $T$  is empty. Applying an Operator  $o$  removes the corresponding primitive task  $t_p$  from  $T$ , appends it to  $\Pi$ , and changes  $S$  as defined by  $o$ . Applying a Method  $m$  replaces the corresponding compound task  $t_c$  in  $T$  with a subnetwork that is a valid *decomposition* according to  $m$  in  $S$  ( $m$  can have more than one valid decomposition in any given state  $S$ ). A decomposition is a Task Network that must be fully executed for the original compound task to be considered executed, and can be viewed as a lower-level description of the more abstract compound task.

An example of a Task Network is depicted in Figure 1. An agent can execute this Task Network by first taking out two defenders, then picking up an enemy flag, and then returning home (to his own base). It does not matter in which order the two enemy defenders are taken out, but they both need to be taken out before the agent can pick up the enemy flag. The two tasks to take out defenders are compound tasks, meaning that they cannot be executed directly but need to be refined further. An example method to do so is depicted in Figure 2. This method defines that the compound task to take out an

enemy can be decomposed into a single compound task to snipe the enemy under the condition that the agent has a Sniper Rifle, or it can be decomposed into an ordered sequence of two primitive tasks under the condition that the agent has a weapon. If neither condition is satisfied, the compound task cannot be decomposed and therefore cannot be executed.

In this description of the HTN Planning formalism, it has only been specified what information needs to be defined by the various structures (such as Operators and Methods), and not how this should be specified. Many existing planners, such as SHOP2, define world states, conditions and effects in (a subset of) First-Order Logic. The framework described in this paper makes no assumptions about the form in which this information is defined, and allows for it to be implemented directly in C++ functions and variables, as described in more detail in Section III. It means that there are few restrictions on what can be specified in an HTN Planning problem. It is possible to define planning problems where the tasks are not totally ordered, and variables are allowed. Such problems can become undecidable [23].

### III. HTN PLANNER

This section describes the implementation of the HTN Planner plugin that has been developed for *Unreal Engine 4*.

#### A. Planning Problem Definition

Many HTN Planners, like SHOP2 [22], define the structures of an HTN Planning problem, such as Operators, using formalisms based on logical expressions. The HTN Planner described in this paper does not use such a logic-based formalism, but instead uses a similar approach as SHPE [24]. Instead of defining all the relevant information of a planning problem using logical expressions, it is defined directly using C++ functions and variables. The main motivation for this approach is that it does not require an inference engine, and is expected to require less processing time [24].

For primitive and compound tasks, two base classes are provided with a number of virtual functions that can be implemented in subclasses to define domain-specific tasks. A primitive task  $t_p$  has the following functions that can be overridden; *ApplyTo(S)*, which should be implemented to apply any effects of  $t_p$  to a world state  $S$ ; *ExecuteTask()*, which should be implemented to execute  $t_p$  during real gameplay (as opposed to during the planning process); *GetCost(S)*, which can be implemented to return the cost of applying  $t_p$  in the world state  $S$ ; and *IsApplicable(S)*, which should be implemented to return a boolean value indicating whether or not  $t_p$  is applicable in a world state  $S$ . A compound task  $t_c$  only has a single function to override; *FindDecompositions(S)*, which should be implemented to return a list  $\mathcal{T}$ , where every  $T \in \mathcal{T}$  is a Task Network that is a valid decomposition of  $t_c$  in the world state  $S$ . Note that, in this implementation, the concepts of *Operators* and *Methods* as mentioned in Section II are no longer used, and any information that these structures contained is instead located directly in the corresponding primitive and compound tasks.

---

**Algorithm 1** The HTN Planning algorithm

---

```
1: function INITIALIZE( $T_0, S_0$ )
2:    $Fringe \leftarrow [(T_0, S_0, \emptyset, 0)]$ 
3:    $BestCost \leftarrow \infty$ 

4: function FINDPLAN
5:   while  $Fringe \neq \emptyset$  and TIMEAVAILABLE() do
6:      $(T_i, S_i, \Pi_i, C_i) \leftarrow Fringe.NEXT()$ 
7:     if  $T_i$  is empty then
8:        $BestPlan \leftarrow \Pi_i$ 
9:        $BestCost \leftarrow C_i$ 
10:    continue
11:    for all  $t \in T_i$  without predecessors do
12:      if  $t$  is primitive then
13:        if  $t.ISAPPLICABLE(S_i)$  then
14:           $T' \leftarrow T_i.REMOVE(t)$ 
15:           $S' \leftarrow t.APPLYTO(S_i)$ 
16:           $\Pi' \leftarrow [\Pi_i, t]$ 
17:           $C' \leftarrow C_i + t.GETCOST(S_i)$ 
18:           $Fringe.ADD((T', S', \Pi', C'))$ 
19:        else
20:           $\mathcal{D} \leftarrow t.FINDDECOMPOSITIONS(S_i)$ 
21:          for all  $D \in \mathcal{D}$  do
22:             $T' \leftarrow T_i.REPLACE(t, D)$ 
23:             $Fringe.ADD((T', S_i, \Pi_i, C_i))$ 
```

---

### B. Finding a Plan

The HTN Planner is expected to take a *Task Network*  $T_0$  and an initial *World State*  $S_0$  as input, and produce a valid plan  $\Pi$  as output, as described in Section II. The algorithm that has been implemented to do this is similar to the algorithm used by *SHPE* [24] and *SHOP2* [22]. It performs a search through the space of all (partially) decomposed networks, starting from  $T_0$ . The intuition behind it is that  $T_0$  is a highly abstract Task Network, containing a relatively large number of compound tasks, and it is gradually simplified by decomposing compound tasks and moving primitive tasks from the network into the plan. Primitive actions are inserted into the plan in the same order in which they are intended to be executed after planning.

Pseudocode for this algorithm can be found in Algorithm 1. The algorithm is initialized with a single tuple  $(T_0, S_0, \emptyset, 0)$  in the *Fringe*. The *Fringe* is the collection of nodes in the search tree that have not been processed yet. Every element in this collection contains the Task Network of tasks that have not yet been completed, the current world state, the (partial) plan constructed so far, and the execution cost so far.

In each iteration, one node  $(T_i, S_i, \Pi_i, C_i)$  is removed from the *Fringe*, and every task  $t \in T_i$  that does not have any predecessors is processed. A predecessor of  $t$  is a task that is constrained by  $T_i$  to require processing before  $t$ . All tasks that are allowed to be executed directly according to  $T_i$  are processed, and all other tasks are not yet processed.

If  $t$  is a primitive task that is applicable in  $S_i$ ,  $t$  is applied to  $S_i$ , appended to  $\Pi_i$ , and removed from  $T_i$ . This results in a single new tuple that is placed in the *Fringe*. If  $t$  is a compound task, one new tuple is placed in the *Fringe* for every valid decomposition  $D$  of  $t$  in  $S_i$ . In this case,  $T_i$  is modified in every new tuple by replacing  $t$  in  $T_i$  with  $D$ .

In the planner described in this paper, the *Fringe* has been implemented as a stack. This means that the algorithm acts as a depth-first search. Many other planners, such as *SHOP2* and *SHPE*, are also implemented in this way.

### C. Branch-and-bound and Heuristic Cost Estimation

A branch-and-bound optimization can speed up the search algorithm described above when searching for an optimal solution. Immediately after taking a tuple  $(T_i, S_i, \Pi_i, C_i)$  from the *Fringe* in line 6 of Algorithm 1, the cost  $C_i$  for executing the partial solution  $\Pi_i$  is compared to the cost of the best solution found so far ( $BestCost$ ). If at this stage  $C_i \geq BestCost$ ,  $\Pi_i$  cannot lead to an improvement on the best solution found so far, and the algorithm can immediately continue with the next element of the *Fringe*. This is the same branch-and-bound optimization as described in [22], [24].

An admissible heuristic function  $h(T_i, S_i)$  that estimates the future cost of executing the remaining Task Network  $T_i$  given a current world state  $S_i$  can be used to improve the branch-and-bound optimization. Given such a function, the algorithm can prune partial solutions where  $C_i + h(T_i, S_i) \geq BestCost$ .

Such a heuristic function can incorporate domain-specific knowledge, but if two extra restrictions are placed on the problem definition it is also possible to define a domain-independent heuristic function. The first of these restrictions is that the cost of executing a primitive task cannot depend on the world state in which it is executed. The second restriction is that compound tasks cannot be defined in such a way that they can result in an infinitely long sequence of decompositions (which is possible when using recursion in the definition of compound tasks). Under these restrictions, the following domain-independent heuristic function  $h(T_i)$  is well-defined:

$$h(T_i) = \sum_{t \in T_i} h(t) \quad (1)$$

$$h(t_p) = Cost(t_p) \quad (2)$$

$$h(t_c) = \min_{D \in \mathcal{D}} h(D) \quad (3)$$

In Equation 1, the world state  $S_i$  has been omitted as an argument because it cannot provide any information for a domain-independent heuristic. The heuristic function  $h(t)$  for a single task  $t$  used in Equation 1 is defined by Equation 2 for the case where  $t$  is primitive, or Equation 3 for the case where  $t$  is compound. In Equation 3,  $\mathcal{D}$  denotes the set of all possible decompositions of  $t_c$  in any possible world state.

## IV. PLAN REUSE

In this section, an approach for reusing old plans to more efficiently find new plans for similar problems is described. This approach does not require effects and conditions of tasks to be explicitly defined in a predefined format.

When an HTN Planner has previously found a plan for some planning problem, and is later required to find a plan for a new planning problem that is similar to the previous problem, it

is expected to be possible to make use of the old solution to speed up the planning process. Existing approaches for reusing or repairing plans in an HTN Planner [15]–[20] require effects and conditions of tasks to be defined in a predefined format (typically using First-Order Logic). In most cases, this is because they analyze dependencies between the conditions and effects of tasks and store this information in graphs or other structures. These approaches are not compatible with the implementation of the planner as described in Section III, where the conditions of tasks and the effects of tasks on the world state are implemented in functions that are black boxes from the point of view of the Planner. The following approach does not have this problem.

Let  $\mathcal{P}^{old} = (S^{old}, T^{old}, \mathcal{O}, \mathcal{M})$  be an old planning problem for which an optimal plan  $\Pi^{old}$  was generated using the HTN Planning algorithm as described in the previous section. Let  $\mathcal{P}^{new} = (S^{new}, T^{new}, \mathcal{O}, \mathcal{M})$  be a new planning problem for which the HTN Planner needs to find a solution  $\Pi^{new}$ .  $\mathcal{O}$  and  $\mathcal{M}$  are equivalent for the two problems, so the same sets of primitive and compound tasks are defined. The assumption is made that  $S^{old}$  and  $S^{new}$  are in some sense similar, and that  $T^{old}$  and  $T^{new}$  are also in some sense similar. Finally, the assumption is made that an optimal solution  $\Pi^{new}$  will, due to the previous assumptions, also be similar to  $\Pi^{old}$ .

The intuition behind the approach is that it is likely to find higher quality solutions first if branches of the search tree that led to  $\Pi^{old}$  are prioritized when traversing the new search tree to look for  $\Pi^{new}$ . Similar ideas have also previously been used in game-tree search algorithms for abstract games [25], [26]. There are two benefits to finding high quality solutions as soon as possible. The first benefit is that, if in a real-time setting such as a video game the planning process is terminated early, a higher quality plan will be available. The second benefit is that the upper bound on the cost of the optimal plan is lowered more quickly, and therefore the branch-and-bound optimization can prune larger parts of the search space.

#### A. Search Tree Structure

Because the approach for plan reuse relies on manipulating the order in which the planning algorithm traverses branches of the search tree, it is useful to first take a closer look at the structure of this search tree.

An example of a search tree for a simple planning problem, with only a single compound task in the initial Task Network, is depicted in Figure 3. A node  $N_i$  in the search tree encapsulates a tuple  $(T_i, S_i, \Pi_i, C_i)$  as found in Algorithm 1. When  $N_i$  is visited (returned by *Fringe.Next()* and processed as seen in the pseudocode), a set of successor nodes  $Successors(N_i)$  can be generated. For example, in Figure 3,  $Successors(\mathbf{A}) = \{\mathbf{B}, \mathbf{E}\}$ .  $Successors(N_i)$  is empty if  $N_i$  gets pruned by the branch-and-bound optimization, or if  $N_i$  is a leaf node. A leaf node is either a *solution* node if  $T_i$  is empty, or a *failed* node if  $T_i$  is non-empty and does not contain any tasks without predecessors that can be executed in  $S_i$ . In the figure, **D** and **G** are solution nodes.

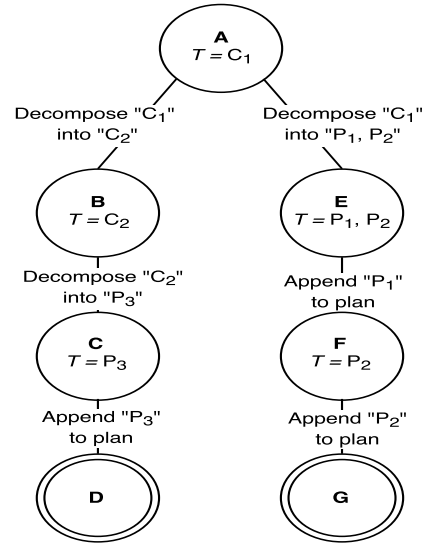


Fig. 3. Example search tree. Every circle represents a node in the search tree. The boldface letters are used to refer to specific nodes in the text. The Task Network of tasks that still need to be solved is shown under this identifier for every node.  $C_i$  are compound tasks, and  $P_i$  are primitive tasks. The text on every branch describes the change that is applied on that transition between two nodes. Nodes **D** and **G** have empty Task Networks.

If  $Successors(N_i)$  is non-empty,  $N_i$  has at least one successor node. If  $T_i$  is *totally ordered*, meaning that there is *exactly one* task  $t \in T_i$  that does not have any predecessors, there will be exactly one successor node if  $t$  is primitive, and there can be more than one successor node if  $t$  is compound. Traversing a branch in this situation can be viewed as committing to solve  $t$  in a certain, concrete way. If  $T_i$  is not totally ordered, there is one set of branches for every task  $t \in T_i$  that does not have any predecessors, and within these sets it is again true that there is exactly one branch if  $t$  is primitive, and there can be more branches if  $t$  is compound. In this case, traversing a branch corresponds to selecting a task  $t$  and then choosing a concrete way to solve that task  $t$ . All Task Networks that appear in Figure 3 are totally ordered.

#### B. Similarity-Based Branch Reordering

The proposed approach for Plan Reuse requires a similarity function  $Sim(\Pi^{old}, N_i)$ , which computes a measure of similarity between  $\Pi^{old}$  and a current node  $N_i$  in the search tree. The information available in  $N_i$  is the tuple  $(T_i, S_i, \Pi_i, C_i)$ . The approach described in this paper only makes use of  $\Pi_i$ , which simplifies the similarity function to  $Sim(\Pi^{old}, \Pi_i)$ .

The example search tree in Figure 3 depicts how the Task Network  $T$  changes in every node, and a description of the processing that is done to generate successor nodes is placed on every branch. When a compound task is processed to generate a successor, only the contents of  $T$  change. This means that, in Figure 3, nodes **A**, **B**, **C** and **E** all share the same plan  $\Pi$ . When a primitive task is processed, the description on the branch indicates how  $\Pi$  changes. This means that nodes **D**, **F** and **G** all have different plans from the other nodes.

Suppose that, at some point in the planning process that generated  $\Pi^{old}$ , the compound task  $C_1$  was solved using the



path in the right-hand side of the figure. Without any changes to the definition of a plan  $\Pi$ , any function  $Sim(\Pi^{old}, \Pi_i)$  returns the same results for the nodes **A**, **B**, **C** and **E** because they share the same plan. This means that **F** is the first node in which it can be recognized that a path is being continued that was a part of the optimal solution of  $\Pi^{old}$ . The first change made to the planning algorithm is to redefine the concept of a plan  $\Pi$  to also append compound tasks to  $\Pi$  as they are processed. With this change, it is already possible to recognize in nodes **B** and **E** that they have some similarity with  $\Pi^{old}$  (they all contain the compound task  $C_1$ ). In node **C**, it can then be recognized that an “incorrect” path is traversed ( $C_2$  has been added which does not occur in  $\Pi^{old}$ ), and in node **F** it can be recognized that the “correct” path is continued, leading to different levels of similarity.

The similarity function proposed in this paper is the function that computes the longest *Currently Matching Streak* (CMS). Intuitively, it is the function that finds the length of the longest sequence of consecutive tasks in  $\Pi^{old}$  that also occurs *at the end* of the current (partial) plan  $\Pi$ . More formally, let  $\Pi[i]$  denote the  $i^{th}$  task in a plan  $\Pi$ . Let  $m$  denote the number of tasks in a plan  $\Pi$ . The similarity measure  $CMS(\Pi^{old}, \Pi)$  is then defined as the maximum possible value  $n$  such that, for some index  $x$ ,  $\Pi^{old}[x] = \Pi_i[m]$ , and  $\Pi^{old}[x - n + j] = \Pi_i[m - n + j]$  for all  $j$  where  $1 \leq j < n$ . A score of 0 is assigned if  $\Pi_i[m]$  does not occur anywhere in  $\Pi^{old}$ .

For example, let  $\Pi^{old} = [A, B, C, D, E]$ ,  $\Pi_i = [A, B, C]$  and  $\Pi_j = [A, B, C, X, D, E]$ . Then  $CMS(\Pi^{old}, \Pi_i) = 3$ , because the entire sequence of tasks of  $\Pi_i$  also occurs as a consecutive sequence in  $\Pi^{old}$ .  $\Pi_j$  also contains the same sequence, but in  $\Pi_j$  the sequence is followed by a non-matching task  $X$ , and then followed by another streak of length 2 that occurs in  $\Pi^{old}$ . Therefore,  $CMS(\Pi^{old}, \Pi_j) = 2$ .

This similarity measure “rewards” streaks of consecutive tasks that also occurred in the same order in  $\Pi^{old}$ , and also instantly punishes appending a non-matching task to an existing matching streak by resetting the score to 0. It is used to sort nodes for processing as follows. A node  $N_i$  is processed before a node  $N_j$  if  $CMS(\Pi^{old}, \Pi_i) > CMS(\Pi^{old}, \Pi_j)$ . If  $CMS(\Pi^{old}, \Pi_i) = CMS(\Pi^{old}, \Pi_j) = 0$ , the  $CMS$  scores of the closest ancestor nodes with non-zero  $CMS$  scores are used instead of the  $CMS$  scores of the nodes themselves. Finally, ties are broken by using the same ordering as a regular DFS. An example search tree is depicted in Figure 4. The numbers in this figure indicate the order in which parts of the subtree are processed.

### C. Domain-Specific Ordering

The approach for reordering branches based on a similarity measure as described above is expected to be better than an arbitrary ordering of branches in cases where the new planning problem is related to the old planning problem. In reality, however, the branches typically are not ordered arbitrarily but are already ordered more efficiently based on domain-specific knowledge. For example, if a compound task is processed to find an item of a specific type somewhere in a map, and

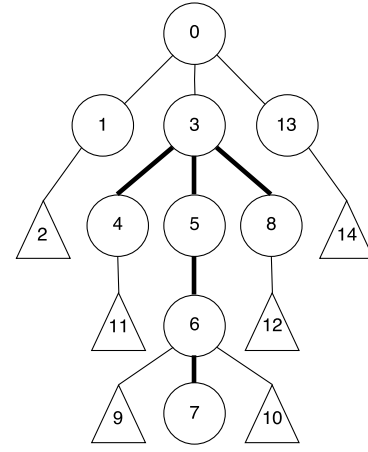


Fig. 4. Example search tree with plan reuse. Circles represent individual nodes, and triangles represent arbitrarily large subtrees. The numbers indicate the order in which nodes or subtrees are processed. Thick lines represent branches where a “correct” choice was made, meaning a task was added that continued a current matching streak or started a new streak.

one valid decomposition is created for every item of that type, these branches can be ordered according to the distance between the agent and those items. Branches for items that are already nearby are then explored first.

When there is already a good ordering of branches based on domain-specific knowledge, Plan Reuse can be detrimental, especially if it is also uncertain if a new planning problem is really similar to an old problem. Two approaches are proposed to reduce the likelihood of Plan Reuse having detrimental effects in the presence of domain-specific ordering, at the cost of also reducing the potential gains of Plan Reuse.

The first approach is the introduction of a parameter  $M$  denoting the *Minimum Streak Length* required for branches to be reordered according to their  $CMS$  score. Any  $CMS$  score that is less than  $M$  is simply set to 0. This makes the plan reuse less aggressive, which means there are fewer potential gains, but it is also more likely that a matching streak of tasks can actually be continued when it already has a sufficient length.

The second approach is to make the search probabilistic. This idea is inspired by a probabilistic approach for resuing plans in classical planning [27]. A parameter  $p$  is introduced, where  $0 \leq p \leq 1$ , which defines the probability with which the planner temporarily ignores parts of the search tree that are prioritized according to Plan Reuse, and instead searches parts that are not prioritized in a DFS manner. This idea has been implemented as follows. Whenever the planning algorithm processes a leaf node, the algorithm is set in a mode where it ignores prioritized nodes with probability  $p$ , and it is set in a mode where it does not ignore prioritized nodes with probability  $1 - p$ . Nodes are considered to be prioritized if and only if they are on a path that contains some node with  $CMS > 0$ . The reason for continuing to run in the same mode until a leaf node is processed is to avoid switching modes too often. Nodes that are pruned by the branch-and-bound optimization are not considered to be leaf nodes. The parameter value  $p = 0$  means that the ordering of Plan Reuse

always overrides the ordering of domain-specific heuristics, and  $p = 1$  means that Plan Reuse is not used. With  $0 < p < 1$ , lower values for  $p$  are more suitable for cases where Plan Reuse is expected to be more reliable than domain-specific heuristics, and higher values are more suitable otherwise.

## V. EXPERIMENTS

This section describes the setup and the results of the experiments that have been carried out to evaluate the performance of the approach for Plan Reuse.<sup>1</sup>

### A. Experimental Setup

*SimpleFPS* [21] is a planning domain that has been designed to simulate planning problems in FPS games. Originally it was defined as a classic planning domain, but it has also been translated into an HTN Planning domain and used for the evaluation of the HTN Planner *SHPE* [24]. Even though *SimpleFPS* is only a simulation of an FPS game, and not a real game, the generated planning problems are not necessarily less complex. With an average optimal plan length of 32 in the experiments described below, the problems can be estimated to be an order of magnitude more complex than those observed in real games [28].

Problems of this planning domain have been randomly generated and used to evaluate the performance of Plan Reuse in comparison to the same planning algorithm without Plan Reuse. The experiments have been carried out inside *UE4*. This means that any overhead involved in implementing and running a planner inside a game engine, as opposed to running it in isolation, is included in the results. The results were obtained using an Intel Core i5 CPU (2.67GHz), running on Windows 7. During the planning processes in these experiments, the memory usage of the entire plugin (including the *SimpleFPS* map data and the constant memory usage of the planner when idle) was at most in the order of 1 MB.

The original version of *SimpleFPS* is deterministic (after random problem generation), and assumes that the agent has access to perfect information. This means that these problems do not require any re-planning. For these experiments, the problems have been changed such that all doors in the maps are assumed to be unlocked initially, and the agent only obtains the information that a door is locked if the agent attempts to move through it. This means that the planner typically finds invalid solutions first, and problems often require re-planning when new information is obtained. To evaluate the performance of Plan Reuse, these “re-planning episodes” have been performed both with and without Plan Reuse. The previous plan is used as  $\Pi^{old}$  for Plan Reuse, but first pre-processed to remove all tasks that have already been executed.

Furthermore, the *SimpleFPS* domain was changed to punish the agent with an extra cost (equivalent to 50 “normal” tasks) for plans in which it chose for a different attacking approach from the original plan, where the three possible attacking

<sup>1</sup>The implementation of the planner used in these experiments, and a more detailed description of the implementation, are available at [https://github.com/DennisSoemers/HTN\\_Plan\\_Reuse](https://github.com/DennisSoemers/HTN_Plan_Reuse).

TABLE I  
EFFECTS OF PLAN REUSE IN TOTAL

Parameter Values	$\Delta$ Nodes Processed	$\Delta$ Time
$M = 10, p = 0$	<b>-7.04%</b>	<b>-11.11%</b>
$M = 10, p = 0.25$	<b>-11.51%</b>	<b>-18.60%</b>
$M = 20, p = 0$	<b>-1.70%</b>	+1.86%
$M = 20, p = 0.25$	<b>-1.18%</b>	+3.53%
$M = 30, p = 0$	<b>-2.81%</b>	<b>-2.41%</b>
$M = 30, p = 0.25$	<b>-2.43%</b>	<b>-0.98%</b>

approaches are melee, ranged and stealth. This means that if, for example, the old plan involved picking up a knife that turns out to be behind a closed door, it is unlikely that a new optimal plan will instead involve picking up a gun somewhere else. With this change, the likelihood of parts of  $\Pi^{old}$  still being useful for a new optimal solution is increased. It is still possible that there also is a second knife somewhere in a more convenient location, so there also still are problems where Plan Reuse can be detrimental.

Six different variants of Plan Reuse have been tested based on the approach described in Section IV, with different values for the parameters  $M$  and  $p$ . For  $M$ , the values 10, 20 and 30 have been tested. The optimal value for this parameter is domain-specific though, and different values may be more suitable for different problems. The value  $M = 1$  has also shortly been tested, but was found to be too aggressive, and has not been included in the results. For  $p$ , the values 0 and 0.25 have been tested, where  $p = 0$  means the use of Plan Reuse is not probabilistic. The value of  $p = 0.25$  was chosen after a smaller number of tests, but is also close to 0.3, which is one of the values used for a similar parameter in [27].

### B. Results

A total of 209 problems were completely processed by all variants of Plan Reuse, of which 10 problems were proven not to have any solutions. There were 22 problems that were not solved by any of the variants because they were terminated due to taking too much time. Planning processes were terminated early and declared a failure if no solution was found at all within 75 seconds, or no optimal solution within 150 seconds.

Table I shows the total change in the number of nodes processed and the amount of time spent planning of all the planning problems added together. It shows that especially the two most aggressive variants of Plan Reuse, with  $M = 10$ , perform well. The variants with  $M = 20$  have a weak performance. This is largely caused by one specific problem, which is one of the largest problems in the set, where Plan Reuse with  $M = 20$  turned out to be highly detrimental. The mean change in the absolute number of nodes processed by the variant with  $M = 10$  and  $p = 0.25$  is significant according to a paired, two-tailed Student’s  $t$ -test with a significance level of 0.05 ( $p$ -value  $\approx 0.037$ ).

Figure 5 shows the difference in plan quality that Plan Reuse makes as a function of the search effort. In this figure, all planning problems have been mapped to a single measure of search effort and a single measure of plan quality. Informally, if a plot has a point  $y$  at  $x = 0.5$ , that variant of Plan Reuse

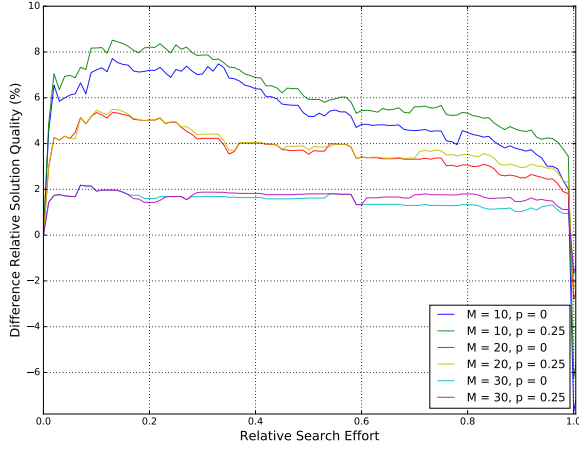


Fig. 5. The change in relative solution quality obtained by adding Plan Reuse as a function of relative search effort.

changes the plan quality by  $y\%$  on average if a planning process is interrupted after half the search effort that planning without Plan Reuse would require to find the optimal solution. A point  $x$  on the  $x$ -axis denotes that  $x \times n$  number of nodes have been processed, where  $n$  is the number of nodes that were required to find (but not necessarily prove) the optimal solution when planning without Plan Reuse. A point  $y$  on the  $y$ -axis denotes the difference in the average quality of the best solution found so far between planning with and without Plan Reuse. The quality of a solution is defined as  $\frac{C^*}{C} \times 100\%$ , where  $C$  is the cost of that solution and  $C^*$  is the cost of an optimal solution for that problem.

All the plots show a decrease close to  $x = 1$ . This is simply because all variants of Plan Reuse have at least some problems where Plan Reuse is detrimental, and  $x = 1$  denotes exactly the amount of search effort that planning without Plan Reuse requires for all problems. So,  $x = 1$  denotes the point in time where it is no longer possible to do any better than planning without Plan Reuse, and it is only possible to do worse. The figure shows that all the variants of Plan Reuse are above the  $x$ -axis (indicating an improvement in average plan quality of up to 8%) until  $x$  approaches 1. The variants with lower values for  $M$  show larger improvements on average. All variants peak with low amounts of search effort, indicating that Plan Reuse is especially beneficial if solutions are required in a short amount of time (for instance, in real-time). For the variants with  $M < 30$ , the changes in quality are significant ( $p$ -value  $< 0.05$ ). There appears to be less variance in the changes in quality for the variants with  $p = 0.25$  ( $p$ -value  $< 0.01$ ).

### C. Unchanged Problems Removed

The results for the problems as described above include all the problems on which none of the variants of Plan Reuse made any difference at all in the number of nodes processed compared to re-planning without Plan Reuse. On some of these problems, Plan Reuse cannot have any effect because the value for  $M$  is too conservative. This is a side effect of avoiding detrimental cases, and therefore these problems have not been excluded from the results above.

TABLE II  
EFFECTS OF PLAN REUSE IN TOTAL - NO UNCHANGED PROBLEMS

Parameter Values	$\Delta$ Nodes Processed	$\Delta$ Time
$M = 10, p = 0$	<b>-11.57%</b>	<b>-18.26%</b>
$M = 10, p = 0.25$	<b>-18.92%</b>	<b>-29.89%</b>
$M = 20, p = 0$	<b>-2.79%</b>	+1.68%
$M = 20, p = 0.25$	<b>-1.93%</b>	+4.26%
$M = 30, p = 0$	<b>-4.62%</b>	<b>-4.81%</b>
$M = 30, p = 0.25$	<b>-3.99%</b>	<b>-2.75%</b>

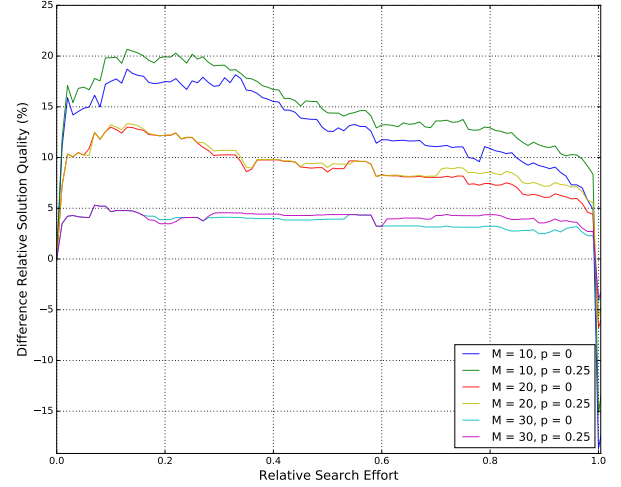


Fig. 6. The change in relative solution quality obtained by adding Plan Reuse as a function of relative search effort. (No unchanged problems)

However, these unchanged problems can also include problems where  $M$  was not set too high, but Plan Reuse simply did not result in any significant changes in the ordering, such as trivial problems where the domain-specific ordering is (nearly) optimal. For planning domains where such planning problems are not expected to occur, it can be interesting to look at the results obtained by removing the problems that remained unchanged by all variants of Plan Reuse from the sets.

The results with these unchanged problems (117 problems) removed can be found in Table II and Figure 6. On this set of problems, Plan Reuse reduces the total number of nodes processed by up to 18.92% and the processing time by up to 29.89% (both for  $M = 10, p = 0.25$ ). The peak increase in average plan quality is up to 20% at a relative search effort of 20% ( $M = 20, p = 0.25$ ). The  $p$ -values of these results are nearly identical to those of the corresponding results including unchanged problems.

## VI. CONCLUSION AND FUTURE WORK

In this paper, an approach has been proposed for reusing previously found plans in an HTN Planner when presented with new planning problems that are similar to one that was previously solved. Unlike existing approaches, it does not require conditions and effects of the domain to be specified in a pre-determined form, but allows for them to be implemented in black box functions. The main idea behind the approach is to manipulate the order in which the search tree is traversed by using a similarity function for plans.

Plan Reuse has been shown to be capable of reducing the average number of nodes required to find optimal solutions for *SimpleFPS* planning problems [21] that are likely to have similar solutions to previously solved problems, and also reduce the computation time. It has also been shown to improve the average quality of plans when using the planning algorithm as an anytime algorithm.

For future work, it would be interesting to investigate whether it is possible to estimate whether Plan Reuse will be likely to be beneficial or detrimental for a specific planning problem before the planning process is started. A direction of future research is to do this automatically by, for instance, computing a similarity measure between the old and the new planning problem. If this likelihood can be estimated accurately, Plan Reuse can be turned off in problems where it is expected not to be beneficial, and it can be turned on in problems where it is expected to be beneficial.

Furthermore, it could be interesting to investigate if Plan Reuse finds plans that are more similar to the old plan in cases where there are multiple plans that are all optimal with respect to the cost function. This has been mentioned as a motivation for Plan Reuse in the paper, because it can reduce the likelihood of an agent abruptly changing behavior in a video game and therefore increase the believability of the behavior. It has not been investigated in this paper's experiments because the *SimpleFPS* problems were found to typically have a low number of different optimal solutions. The planner and approach for Plan Reuse have also briefly been tested in an alpha version of the game of *Unreal Tournament*, which uses Unreal Engine 4. These tests are not described in more detail because the planning problems were too simple for Plan Reuse to make a noticeable difference. Experiments with more complex planning problems in later versions of the game, or in other games, could be done in future research.

Finally, a direction for future work would be to look into different variants of similarity measures. For instance, the *CMS* measure could be changed to have a lower value when appending a non-matching task to an existing streak, instead of resetting the score entirely to 0.

## REFERENCES

- [1] I. Millington and J. Funge, *Artificial Intelligence for Games*, 2nd ed. Morgan Kaufmann, 2009.
- [2] D. Isla, "Managing Complexity in the Halo 2 AI System," in *Proceedings of the Game Developers Conference*, 2005.
- [3] J. Orkin, "Three States and a Plan: The A.I. of F.E.A.R.," in *Game Developers Conference*, 2006.
- [4] E. D. Sacerdoti, "The Nonlinear Nature of Plans," in *Proc. 4th Int. Joint Conf. Artif. Intell.*, vol. 1. Stanford, CA: Morgan Kaufmann Publishers Inc., 1975, pp. 206–214.
- [5] K. Erol, J. Hendler, and D. S. Nau, "UMCP: A Sound and Complete Procedure for Hierarchical Task-Network Planning," in *Proc. 2nd Int. Conf. on Artif. Intell. Planning Syst.*, K. Hammond, Ed. Menlo Park, CA: AAAI Press, 1994, pp. 249–254.
- [6] H. Hoang, S. Lee-Urban, and H. Muñoz-Avila, "Hierarchical Plan Representations for Encoding Strategic Game AI," in *Proc. AIIIDE*. AAAI Press, 2005, pp. 63–68.
- [7] J. P. Kelly, A. Botea, and S. Koenig, "Offline Planning with Hierarchical Task Networks in Video Games," in *Proc. 4th AIIIDE Conf.*, C. Darken and M. Mateas, Eds. Menlo Park, CA: AAAI Press, 2008, pp. 60–65.
- [8] A. Menif, C. Guettier, and T. Cazenave, "Planning and Execution Control Architecture for Infantry Serious Gaming," in *Proc. of the Planning in Games Workshop of ICAPS 2013*, M. Buro, E. Jacopin, and S. Vassos, Eds., 2013, pp. 31–34.
- [9] I. M. Mahmoud, L. Li, D. Wloka, and M. Z. Ali, "Believable NPCs in Serious Games: HTN Planning Approach Based on Visual Perception," in *Proc. IEEE Conf. Comput. Intell. Games*, 2014, pp. 248–255.
- [10] S. Ontañón and M. Buro, "Adversarial Hierarchical-Task Network Planning for Complex Real-Time Games," in *Proc. 24th Int. Joint Conf. Artif. Intell.*, Q. Yang and M. Wooldridge, Eds. AAAI Press, 2015, pp. 1652–1658.
- [11] T. Humphreys, "Exploring HTN Planners through Examples," in *Game AI Pro: Collected Wisdom of Game AI Professionals*, 1st ed., S. Rabin, Ed. CRC Press, 2013, ch. 12, pp. 149–167.
- [12] U. Kuter and D. Nau, "Forward-Chaining Planning in Nondeterministic Domains," in *Proc. 19th Nat. Conf. Artif. Intell.*, A. G. Cohn, Ed. Menlo Park, CA: AAAI Press, 2004, pp. 513–518.
- [13] U. Kuter, D. Nau, M. Pistore, and P. Traverso, "Task Decomposition on Abstract States, for Planning under Nondeterminism," *Artificial Intelligence*, vol. 173, no. 5–3, pp. 669–695, 2009.
- [14] S. Yoon, A. Fern, and R. Givan, "FF-Replan: A Baseline for Probabilistic Planning," in *Proc. 17th ICAPS*, M. Boddy, M. Fox, and S. Thiébaux, Eds. Menlo Park, CA: AAAI Press, 2007, pp. 352–359.
- [15] S. Kambhampati and J. A. Hendler, "A Validation-Structure-Based Theory of Plan Modification and Reuse," *Artificial Intelligence*, vol. 55, no. 2–3, pp. 193–258, 1992.
- [16] B. Drabble, J. Dalton, and A. Tate, "Repairing Plans On-the-fly," in *Proc. NASA Workshop on Planning and Scheduling for Space*, 1997.
- [17] R. P. J. van der Krogt and M. M. de Weerd, "Plan Repair as an Extension of Planning," in *Proc. 15th ICAPS*, S. Biundo, K. Myers, and K. Rajan, Eds. Menlo Park, CA: AAAI Press, 2005, pp. 161–170.
- [18] N. F. Ayan, U. Kuter, F. Yaman, and R. P. Goldman, "HOTRIDE: Hierarchical Ordered Task Replanning in Dynamic Environments," in *Planning and Plan Execution for Real-World Systems—Principles and Practices for Planning in Execution: Papers from the ICAPS Workshop*, F. Ingrand and K. Rajan, Eds., 2007.
- [19] I. Warfield, C. Hogg, S. Lee-Urban, and H. Muñoz-Avila, "Adaptation of Hierarchical Task Network Plans," in *FLAIRS Conference*, 2007, pp. 429–434.
- [20] J. Bidot, B. Schattenberg, and S. Biundo, "Plan Repair in Hybrid Planning," in *KI 2008: Advances in Artificial Intelligence*, ser. LNCS, A. R. Dengel, K. Berns, T. M. Breuel, F. Bomarius, and T. R. Roth-Berghofer, Eds. Springer, 2008, vol. 5243, pp. 169–176.
- [21] S. Vassos and M. Papakonstantinou, "The SimpleFPS Planning Domain: A PDDL Benchmark for Proactive NPCs," in *AIIIDE Workshop: Intelligent Narrative Technologies*, E. Tomai, D. Elson, and J. Rowe, Eds. AAAI Press, 2011, pp. 92–97.
- [22] D. Nau, T. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman, "SHOP2: An HTN Planning System," in *JAIR*, M. E. Pollack, Ed. AAAI Press, 2003, vol. 20, pp. 379–404.
- [23] K. Erol, J. Hendler, and D. S. Nau, "HTN Planning: Complexity and Expressivity," in *Proc. 12th Nat. Conf. on Artif. Intell.* Menlo Park, CA: AAAI Press, 1994, pp. 1123–1128.
- [24] A. Menif, E. Jacopin, and T. Cazenave, "SHPE: HTN Planning for Video Games," in *Computer Games*, ser. Communications in Computer and Information Science, T. Cazenave, M. H. M. Winands, and Y. Björnsson, Eds. Springer, 2014, vol. 504, pp. 119–132.
- [25] Y. Kawano, "Using Similar Positions to Search Game Trees," in *Games of No Chance*, ser. MSRI Book Series, R. Nowakowski, Ed. Cambridge: Cambridge University Press, 1996, vol. 29, pp. 193–202.
- [26] M. Sakuta, T. Hashimoto, J. Nagashima, J. W. H. M. Uiterwijk, and H. Iida, "Application of the Killer-Tree Heuristic and the Lambda-Search Method to Lines of Action," *Information Sciences*, vol. 154, no. 3, pp. 141–155, 2003.
- [27] D. Borrajo and M. Veloso, "Probabilistically Reusing Plans in Deterministic Planning," in *Proc. ICAPS'12 Workshop on Heuristics and Search for Domain-Independent Planning*, P. Haslum, M. Helmert, E. Karpas, C. L. López, G. Röger, J. Thayer, and R. Zhou, Eds. AAAI Press, 2012, pp. 17–25.
- [28] E. Jacopin, "Game AI Planning Analytics: The Case of Three First-Person Shooters," in *Proc. 10th AIIIDE Conf.* AAAI Press, 2014, pp. 119–124.

# Altruistic Punishment Can Help Resolve Tragedy of the Commons Social Dilemmas

Garrison W. Greenwood  
Department of Electrical & Computer Engineering  
Portland State University  
Portland, OR 97207-0751 USA  
Email: greenwd@pdx.edu

**Abstract**—Social dilemmas force individuals to choose between cooperation, which benefits a group, and defection which benefits the individual. The unfortunate outcome in most social dilemmas is mutual defection where nobody benefits. Researchers frequently use mathematical games such as public goods games to help identify circumstances that might improve cooperation levels within a population. Altruistic punishment has shown promise in these games. Many real-world social dilemmas are expressed via a tragedy of the commons metaphor. This paper describes an investigation designed to see if altruistic punishment might work in tragedy of the commons social dilemmas. Simulation results indicate not only does it help resolve a tragedy of the commons but it also effectively deals with the associated first-order and second-order free rider problems.

## I. INTRODUCTION

Social dilemmas arise whenever individuals must choose between self-interests and collective interests. The study of social dilemmas has dramatically increased over recent years because they describe many real-world pressing problems such as overfishing, use of public lands and energy consumption. In such social dilemmas individuals can choose to “cooperate” by putting their self-interests aside for the collective good or “defect” by putting their self-interests first even if it means the collective benefits are reduced. Social dilemmas have two conflicting characteristics: (1) individuals who defect get higher rewards regardless of what others decide, but (2) mutual cooperation yields greater returns for all individuals than if everyone defects. The unfortunate outcome of most social dilemmas is everyone defects and the group suffers.

Mathematical games provide an ideal framework for studying social dilemmas. These  $N$ -player games ( $N > 2$ ) start with a random allocation of individuals in a population choosing to cooperate or defect and then observing how those population choices evolve over time. The objective is to gain insight into why humans make particular choices. The most widely studied such game is a *Public Goods Game* (PGG). Another less studied game (but the author would argue a more realistic social dilemma model) is the *Tragedy of the Commons* (TOC). The difference between these two social dilemmas will be discussed in the next section.

Nowak [1] posited five rules that could promote cooperation in populations: direct reciprocity, indirect reciprocity, network reciprocity, kin selection, and group selection. These rules possibly explain the emergence of cooperation in pairwise

interactions but they are practically too simplistic to explain it in large groups. A more holistic approach is needed. For example, indirect reciprocity, which is based on reputation, probably has little effect after 2 or 3 interaction rounds. Direct reciprocity, where individuals make decisions in reaction to other players’ previous interactions, are unlikely to explain how an individual makes decisions in a group where say half of the other individuals cooperated while the other half defected. Moreover, Nowak’s rules only deal with personal perspectives. In other words, they are responses to past decisions other players made; they do not consider deliberate actions players might take to influence the future choices other players make. One such deliberate action is *altruistic punishment* where players punish others who did not cooperate in previous rounds, while incurring a personal cost to inflict this punishment.

Altruistic punishment has been studied in PGGs, but little work has explored how it affects TOC social dilemmas. TOC is fundamentally different than a PGG so it is interesting to see how altruistic punishment might help resolve a TOC. This issue was investigated and our results are reported in this paper. These results indicate that altruistic punishment can help resolve a TOC.

The paper is organized as follows, In Section II the difference between a PGG and TOC are explained and some past work on altruistic punishment in PGGs is presented. Section III describes our model while Section IV presents experimental results. Section V discusses how altruistic punishment should be implemented to help resolve TOC social dilemmas and provides recommendations for future work.

## II. BACKGROUND

Altruistic punishment has been applied to PGGs but not in a TOC. These two social dilemmas have a prisoner’s dilemma schema, but they are not the same so it is important to explain the difference. The evolution of player strategies in a finite population is governed by discrete replicator equations which are also described in this section.

### A. PGG and TOC

In each round of a PGG  $N$  players independently decide whether to cooperate by contributing a fixed amount  $y$  to a common pool or defect by contributing nothing. An external benefactor multiplies the pool by a factor  $r < N$  and

then equally distributes this increased amount to every player whether or not they contributed. Defectors are therefore free riders because they benefit from the contributions of others.

In each round of a TOC  $N$  players use or consume a fixed resource. This resource is periodically renewed. Cooperators limit their consumption to help preserve the resource whereas defectors maximize their consumption regardless of how the resource is affected. But by limiting their consumption cooperators make more of the resource available for the defectors. Thus defectors free-ride by exploiting the goodwill of others. A TOC is considered “resolved” if the resource remains viable—i.e., it is preserved for future use. A population of all cooperators resolves a TOC.

**Definition 1.** A good is non-excludable if everyone can use it.

An example of non-excludability is national defense. Everyone benefits from national defense whether or not they pay the taxes needed to finance it. Cooperators pay taxes whereas defectors are free riders who pay little or no taxes.

**Definition 2.** Diminishable goods are finite and can be completely depleted unless replenished.

One characteristic of diminishable goods is use by one individual denies use by another individual. A good example is fisherman in the Tasman Sea. The amount of fish is finite so any fish caught by one fisherman cannot be caught by another fisherman. Cooperators voluntarily limit the amount they catch to keep the fish population viable. Defectors are free riders because they exploit the benevolence of cooperators because there are now more fish available to take.

The inevitable outcome for a PGG or a TOC is everybody defects. Both a PGG and a TOC are non-excludable because everyone is allowed to participate. However, a PGG is not diminishable because regardless of the number of rounds and regardless of the amount contributed, it is assumed the external benefactor always has sufficient funds to increase the pool amount and distribute it. Conversely, in a TOC the resource is diminishable because it is finite and, unless replenished, will eventually become depleted. That difference may appear subtle, but the distinction is important: players are rivals only in the TOC. In a PGG a payoff given to one player does not reduce the payoff to another player because the pool is distributed equally. However, in a TOC the shared resource is finite so any amount consumed by one player is not available to another player. TOC is thus a zero-sum game while a PGG is not.

### B. Altruistic Punishment

Diminishable goods and the associated rivalry are important distinctions between a PGG and a TOC social dilemma. Those distinctions lead to different motivations for punishment.

**Definition 3.** Altruistic punishment is punishment inflicted on free riders even if costly to the punisher and even if the punisher receives no material benefits from it.

The motivation behind punishing defectors is to convince them it is better to cooperate. In a PGG the motivation is purely self-serving: the more others cooperate the larger the pool and the higher the payoff to existing cooperators. Conversely, the motivation for punishment in a TOC is for the collective good: the more others cooperate the higher the likelihood the public good will be preserved. This raises an important question. If the motivation behind the punishment is different in the two social dilemmas, will punishment have a similar affect on cooperation levels in both social dilemmas?

Altruistic punishment is easily added to a PGG. There are now three strategies: *cooperators* ( $C$ ) who contribute an amount  $y$  to a common pool; *defectors* ( $D$ ) who contribute nothing; and *punishers* ( $P$ ) who also contribute  $y$  but, at some small cost  $\alpha > 0$  to impose a fine on defectors. Let  $\pi(z)$  be the return to a player  $z \in \{C, D, P\}$ . Then with a population size  $N$  and multiplication factor  $r$  the payoffs each round are

$$\pi(z) = \begin{cases} rky/N - m\beta & \text{defector} \\ rky/N & \text{cooperator} \\ rky/N - \alpha & \text{punisher} \end{cases} \quad (1)$$

where  $k$  is the number of cooperators,  $m$  is the number of punishers,  $\beta$  is the imposed punishment (e.g., a fine) per punisher and  $\alpha$  is the fixed cost paid for inflicting a punishment.. Notice the punisher acts like a cooperator, but receives a slighter lower return because of the cost paid to inflict punishment on the defectors. The returns are identical if there are no defectors in the population because the cost is incurred if and only if punishment is administered.

There are a number possible PGG variations. For example, in Eq. (1) the punishment for a defector depends on the number of punishers but the cost  $\alpha$  to the punisher is the same regardless of how many defectors are punished. This cost could be changed to say  $\ell\alpha$  where  $\ell$  is the number of defectors. Also, punishment could be added to cooperators to address the so-called 2nd order free rider problem with an associated additional cost to the punisher. (These additional punishment forms and costs are incorporated into our TOC experiments.)

Human experiments have shown altruistic punishment has a positive affect on cooperation levels in PGGs. Fehr and Gächter [2] had 240 university students participate in two sets of 6-round PGGs: one set with punishment and one set without. They found that 72% of above average contributors punished below average contributors. They found cooperation levels were considerably higher ( $\approx 40\%$ ) with punishment regardless of whether the set of games with punishment preceded the set without punishment or vice versa. Furthermore, they found contribution levels either remained constant or slightly increased over time with punishment but tended to decrease over time with no punishment<sup>1</sup>.

<sup>1</sup>Fehr and Gächter also believed Nowak’s 5 rules were too simplistic to explain group behavior.

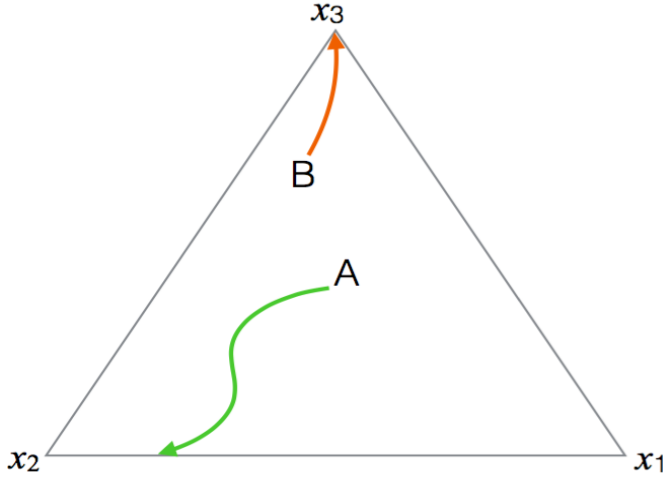


Fig. 1. A 2-simplex showing two trajectories representing the evolution of an infinite population of three strategies. The green trajectory starts at point A and terminates a fixed point on the  $x_1$ - $x_2$  boundary. The final population mixture contains only type 1 and type 2 strategies with more type 2 strategies. The red trajectory starts at point B and ends at the  $x_3$  vertex. This final population has been taken over by type 3 strategies.

### C. Replicator Equations

How do we represent the evolution of the population over time? A convenient representation is a simplex, Figure 1 shows a 2-D simplex, which is suitable for three strategies. Let  $x_i \in [0, 1]$  be the frequency of strategy  $i$  where  $\sum_i x_i = 1.0$ . Each point in the simplex represents a mixture of three strategies in an infinite size population—i.e., every point has coordinates  $[x_1 \ x_2 \ x_3]$ . Trajectories in the simplex reflect how the population evolves over time. This evolution is governed by *replicator equations* which are 1st-order differential equations of the form

$$\dot{x}_i = x_i (F_i - \bar{F}) \quad (2)$$

where  $F_i$  is the fitness of strategy  $i$  and  $\bar{F}$  is the mean fitness of the population. If the term in parenthesis is positive, then strategy  $i$  increases; if negative it decreases; and if zero it does not change. The population size is infinite so a trajectory can pass through any point in the simplex.

Figure 1 shows two trajectories. Trajectories are smooth because their path is described by a differential equation. The green trajectory begins at point A and terminates on the  $x_1$ - $x_2$  boundary. This is a fixed point where the final population consists of (mostly)  $x_2$  strategies but some  $x_1$  strategies. The red trajectory begins at point B and terminates at the  $x_3$  vertex. That final population consists solely of  $x_3$  strategies.

Human populations, however, are always finite. Let  $k_i$ ,  $i \in \{C, P, D\}$  be the number of players choosing strategy  $i$ . In a finite population of size  $N$  the frequency of strategy  $i$  at time  $t$  is  $p_i^t = k_i/N$ . Then the population evolution over time is now given by the *discrete replicator equation*

$$p_i^{t+1} = p_i^t \left( \frac{\pi_i^t}{\bar{\pi}^t} \right) \quad (3)$$

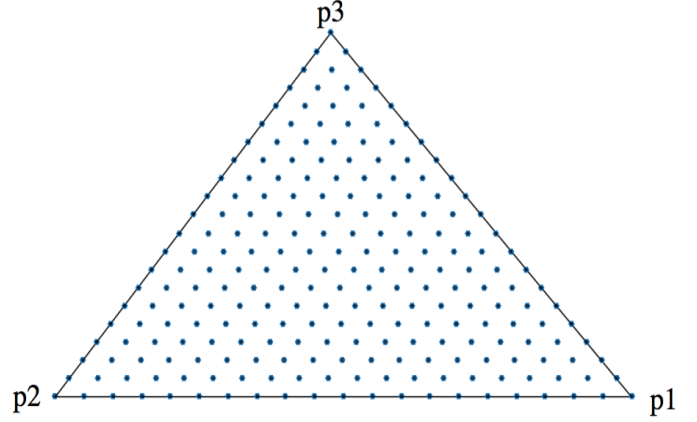


Fig. 2. A 2-simplex for a finite population with  $N = 20$ . Only the points shown represent an integer number of strategies where  $\sum_i k_i = N$ . A trajectory can only move between these points.

where  $\pi_i^t$  is the return for strategy  $i$  at time  $t$ . The term inside the parenthesis is the ratio of the return of strategy  $i$  to the average population return  $\bar{\pi}^t$ . If the term is greater than 1.0 then strategy  $i$  grows in the population; if equal to 1.0 it remains the same; and if less than 1.0 it decreases.

But there is a problem. Eq. (3) can be rewritten (after multiplying both sides by  $N$ ) as

$$k_i^{t+1} = k_i^t \left( \frac{\pi_i^t}{\bar{\pi}^t} \right) \quad (4)$$

The problem is there is no guarantee the term in parenthesis is an integer which means the left hand side of Eq. (4) may not be an integer. Clearly  $k_1 + k_2 + k_3 = N$  is required. To overcome this problem the quantization algorithm below (from [3]) was used. The algorithm takes  $\{p_i^{t+1}\}$  and  $N$  and returns  $k'_i$  where  $\sum k'_i = N$ .

1) Compute

$$k'_i = \lfloor N p_i + \frac{1}{2} \rfloor, \quad N' = \sum_i k'_i$$

2) Let  $d = N' - N$ . If  $d = 0$ , then go to step 4. Otherwise, compute the errors  $\delta_i = k'_i - N p_i$ .

3) If  $d > 0$ , decrement the  $d$   $k'_i$ 's with the largest  $\delta_i$  values. If  $d < 0$ , increment the  $|d|$   $k'_i$ 's with the smallest  $\delta_i$  values.

4) Return  $[k'_1 \ k'_2 \ k'_3]$  and exit.

For this investigation we use a 2-D simplex to represent the population evolution. But in finite populations only certain points are valid because there are only a finite number of points where  $\sum_i k_i = N$ . Figure 2 shows a 2-D simplex for a finite population with  $N = 20$ . The trajectories showing the population evolution are piecewise linear since only transitions between these points are allowed.

### III. THE TOC MODEL

The model has  $N = 20$  players. We adopt the same notion used in the previous section where  $k_i$  is the number of players



in the population using strategy  $i \in \{C, P, D\}$  and  $p_i^t = k_i/N$  is the corresponding frequency at time  $t$ . (For convenience the  $t$  will be dropped when there is no ambiguity.) Each round each player consumes an amount of a finite resource, which is assumed to have an initial capacity 50000 “units” (an arbitrary unit of measure).  $C$  and  $D$  players consume 40 units while  $P$  players consume 50 units. The slight higher consumption rate for  $D$  players reflects a greedier, self-serving approach. After 10 rounds the remaining resource capacity is increase by 20%. If the population is composed entirely of  $C$  or  $P$  players the total consumption over 10 rounds would be  $20 \cdot 10 \cdot 40 = 8000$  units. A 20% increase boosts the capacity to slightly over 50000 before starting the next 10 rounds. This slight increase over 50000 keeps the resource capacity viable even if a small number of defectors are present. The game continues for a fixed number of rounds or until the resource is depleted, whichever comes first.

The return is 39 units per round for  $C$  or  $P$  players and 82 units per round for  $D$  players. There is a strong incentive to consume more since the return for defecting is more than twice that of not defecting.

#### IV. EXPERIMENTAL RESULTS

All simulations were conducted with  $N = 20$ . The rewards for each strategy  $i$  were

$$\pi(i) = \begin{cases} 82 - k_2\beta & \text{defectors} \\ 39 & \text{cooperators} \\ 39 - ck_3\alpha & \text{punishers} \end{cases} \quad (5)$$

where  $\beta$  is the defector’s punishment and  $\alpha = 1.0$  is the cost a punishers pays. The constant  $c$  equals 1 if there are defectors in the population or 0 otherwise. This constant ensures punishers pay costs only when defectors are present. Notice punishments and costs are additive since each punisher imposes a punishment  $\beta$  on a defector and pays a cost  $\alpha$  for every defector punished.

The returns in Eq. (5) are slightly different from Eq. (1) because the former is for a TOC game and the latter for a PGG. The returns in a PGG are based on a contribution amount  $y$ , a multiplication factor  $r$  and the frequency of cooperators. In the TOC the constants 82 and 39 represent player consumption rates which only depend on the strategy played. The punishment for defection is the same in both games. Eq. (5) has an additional penalty for cooperator free riding and a corresponding additional cost for the punisher.

Figure 3 shows the population evolution for an initial strategy distribution of  $[p_1 \ p_2 \ p_3] = [0.35 \ 0.35 \ 0.30]$  corresponding to  $k_1 = k_2 = 7$  and  $k_3 = 6$ . The replicator equations show for low  $\beta$  values the punishment isn’t high enough to prevent defectors from taking over the population. The punishment becomes high enough to coerce defectors to switch strategy as  $\beta$  approaches 7. The trajectory with  $\beta = 7.1$  terminates at an interior fixed point. (Fixed points are discussed in the next section.) The trajectory with  $\beta = 10.0$  terminates at the  $p_1$ - $p_2$  boundary which means there are no defectors left

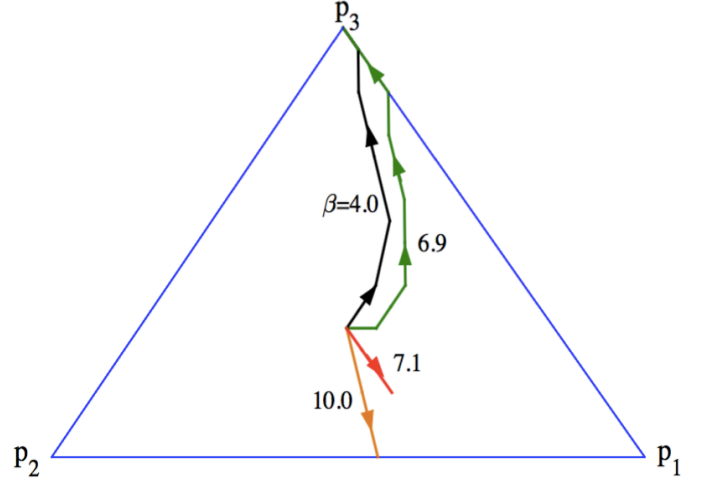


Fig. 3. Evolution of a finite populations ( $N = 20$ ) for various punishment values ( $\beta$ ). Punisher cost is  $\alpha = 1$ . Initial distribution for all trajectories is  $[p_1 \ p_2 \ p_3] = [0.35 \ 0.35 \ 0.30]$  which corresponds to  $k_1 = k_2 = 7$  and  $k_3 = 6$ . Payoffs are 39, 39 and 82 for  $C$ ,  $P$  and  $D$  players respectively. For  $\beta \approx 7$  or less defectors eventually take over the population.

in the population. Consequently,  $c = 0$  yielding the same payoffs for a punisher and a cooperator. In that case there is no distinction between the two players and the TOC was completely resolved.

Figure 4 shows just two trajectories from Figure 3. The black dashed line is parallel to the  $p_2$ - $p_3$  boundary. Every point on this dashed line has  $p_1 = 0.35$  ( $k_1 = 7$ ). Similarly the red dashed line is parallel to the  $p_1$ - $p_3$  boundary and every point on it has  $p_2 = 0.35$ . Notice the trajectory with  $\beta = 7.1$  exactly overlaps the red dashed line. This means the number of punishers did not change from the initial value of  $k_2 = 7$ . Thus all defectors who switched strategies became cooperators. The trajectory with  $\beta = 10.0$  intersects the  $p_1$ - $p_2$  boundary to the right of where the dashed black line intersects that boundary indicating the cooperator frequency increased. But it intersects to the left of where the red dashed line intersects indicating the punisher frequency also increased. Thus in this case some defectors switched to cooperators while others switched to punishers<sup>2</sup>.

There are actually two ways of inflicting more punishment on defectors: fix  $k_2$  and increase  $\beta$  or fix  $\beta$  and increase  $k_2$ . To explore this latter case more thoroughly we conducted an experiment with various initial values of  $k_2$  while fixing the initial value of  $k_3$  at 7. For all runs  $\beta = 6.9$  which Figure 3 indicates was too small to prevent the population from being taken over by defectors. Figure 5 shows 7 and even 8 punishers were not enough to prevent the take over by defectors. However, for  $k_2 \geq 9$  the population contains enough punishers to coerce defectors to switch strategies.

<sup>2</sup>The initial population had  $k_1 = 7, k_2 = 7$  and  $k_3 = 6$  whereas the final population had  $k_1 = 11, k_2 = 9$  and  $k_3 = 0$ .



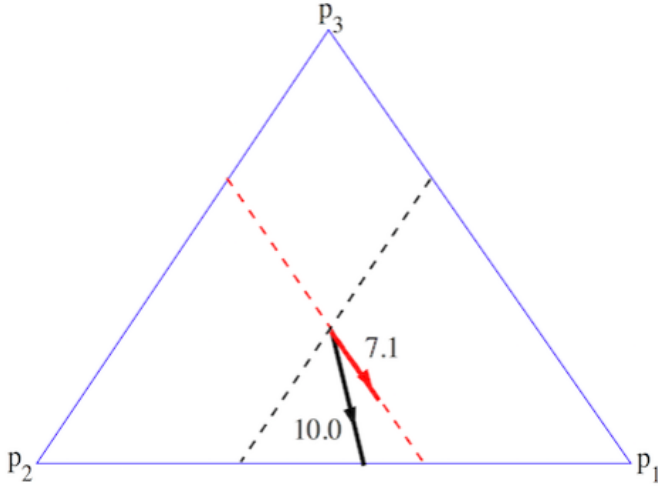


Fig. 4. Evolution of a finite populations ( $N = 20$ ) for  $\beta = 7.1$  and  $10.0$ . Dashed lines represent constant strategy frequencies for  $p_1$  (black) and  $p_2$  (red). The initial distribution is  $[p_1 \ p_2 \ p_3] = [0.35 \ 0.35 \ 0.30]$ . These trajectories are the same as those in Figure 3.

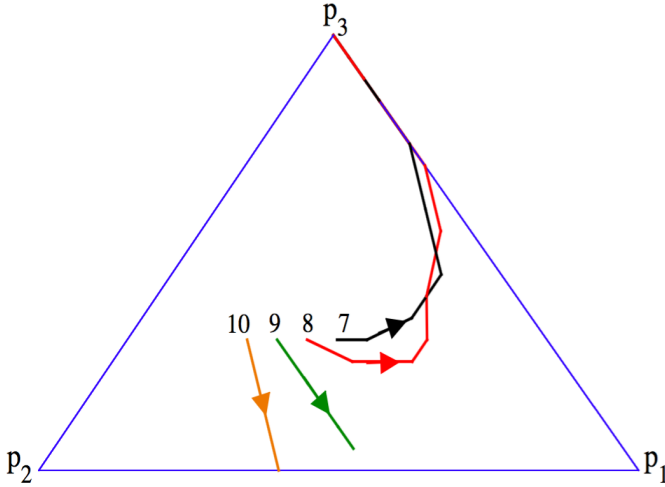


Fig. 5. Effect of varying the frequency of punishers with fixed punishment  $\beta = 6.9$  and  $k_3 = 6$  (cf., Figure 3). Numbers indicate the initial number of punishers. Punisher cost is  $\alpha = 2$ .

## V. DISCUSSION

The 2-D simplex was used for trajectories showing how strategies evolve in the population. These trajectories were generated from the discrete replicator equations. Strategy frequencies will not change any further if the trajectory hits a *fixed point*. This simplex contains several fixed points. For instance, it is easy to prove every vertex is a fixed point. All interior fixed points are caused by the quantization process. Consider the red trajectory in Fig 3. That trajectory hits a fixed point at  $p_i = [0.528 \ 0.341 \ 0.131]$  which corresponds to  $k_1 = 10$ ,  $k_2 = 7$  and  $k_3 = 3$ . Plugging those  $p_i$  values into the quantization algorithm returns the same population mixture. The location of these interior fixed points will change as  $N$  increases and completely disappear as  $N \rightarrow \infty$ . However, there are some natural fixed points on one simplex boundary

(other than at the vertices) which do not disappear and the exact number grows with  $N$ . For infinite population sizes every point on this boundary is a fixed point.

**Theorem.** Every point on the  $p_1$ - $p_2$  boundary of the 2-D simplex is a fixed point.

*Proof.* Suppose at time  $\tau$  a trajectory intersects a point on the  $p_1$ - $p_2$  boundary. Then  $k_3 = 0$  and by Eq. (4) remains so for all  $t \geq \tau$ . This makes  $c = 0$  in Eq. (5) so the return to a punisher or a cooperator is the same. The proof follows because with identical returns there is no incentive to change strategies.  $\square$

Referring back to Figure 3, the initial population mixture was  $[p_1 \ p_2 \ p_3] = [0.35 \ 0.35 \ 0.30]$ .  $\beta = 6.9$  wasn't a sufficient punishment to prevent defectors from eventually taking over the population. This poses an interesting question: if the punishment were subsequently increased would this induce defectors to switch strategies?

To investigate this issue further an experiment was conducted where punishers doubled the punishment if  $p_3$  exceeded 0.35. The results are shown in Figure 6. The replicators equations predict after an initial decrease in defectors their number increased again until they take over the population (green trajectory). A second experiment (red trajectory) investigated what would happen if the punishers immediately doubled the punishment if there was *any* increase in defector frequency. In this case, after an initial increase in defectors, they rapidly decreased until a fixed point was reached. This suggests if punishers want to prevent defectors from taking over a population, then they shouldn't wait too long before increasing the punishment.

In this last experiment  $\beta$  was doubled to stop the growth of defectors. That may seem excessive since Figure 3 shows  $\beta = 6.9$  wasn't sufficient to stop defector growth but  $\beta = 7.1$  was sufficient. Notice the trajectories where defectors prevail move towards the  $p_2$ - $p_3$  simplex boundary and then up to the  $p_3$  vertex. In other words, the number of punishers is decreasing. Consequently, the punishment *per punisher* must increase sharply to stop defector growth because there are fewer punishers in the population.

Cooperators limit their consumption of the shared resource for the good of the group. This allows the resource to replenish itself, thereby keeping it viable. Defectors are free riders. They may consume more of the shared resource by free riding on cooperators who want to preserve the resource. The experimental results show punishment can effectively reduce the number of defectors in a finite population, but if and only if the punishment is sufficiently large enough. This can be achieved two ways: either keep the same number of punishers in the population and increase the punishment level or keep the punishment level the same but increase the number of punishers. In both cases the defectors are reduced.

But there is another type of free rider. Altruistic punishment comes with a cost to the punisher, which reduces his return. Cooperators who do not punish free ride on punishers. That is, they reap the benefits of fewer defectors in the population,

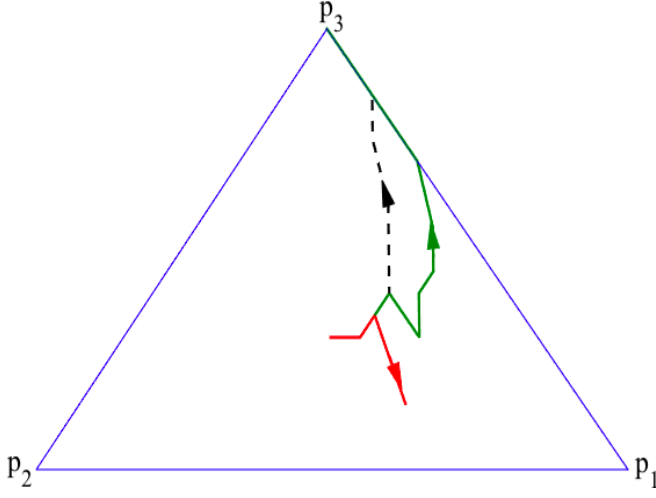


Fig. 6. Effect of adapting  $\beta$ . The dashed trajectory is the same trajectory for  $\beta = 6.9$  from Figure 3 included for reference. Initial distribution is  $[p_1 \ p_2 \ p_3] = [0.35 \ 0.35 \ 0.30]$ .  $\beta$  is doubled when  $p_3 \geq 0.35$  (green trajectory) or  $p_3 > 0.30$  (red trajectory).

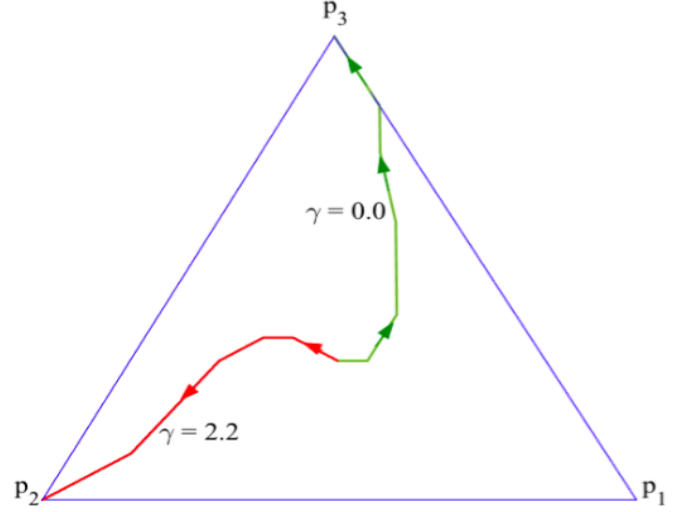


Fig. 7. Effect of punishing cooperators with punisher cost  $\eta = 0.1$ .  $\gamma$  is punishment to cooperator. Defector punishment is  $\beta = 6.9$ .  $\gamma = 0.0$  trajectory is same trajectory as the green trajectory from Figure 3.

but they let punishers pay the associated cost. This is referred to as the *second-order free rider problem*.

Punishment effectively handles the first-order free rider problem—i.e., defector free riding—so it is reasonable to assume it would also handle the second-order free rider problem. Now cooperators are punished for not punishing defectors. Each punisher pays a cost  $\eta$  to reduce a cooperator's return by  $\gamma$ . The new returns for a strategy  $i$  are

$$\pi(i) = \begin{cases} 82 - k_2\beta & \text{defectors} \\ 39 - ck_2\gamma & \text{cooperators} \\ 39 - c[(k_3\alpha) + (k_1\eta)] & \text{punishers} \end{cases} \quad (6)$$

The punishments and costs can be summarized as follows:

- 1) defectors get punished  $\beta$  by each punisher
- 2) cooperators get punished by  $\gamma$  by each punisher
- 3) punishers pay a cost  $\alpha$  for each defector punished plus a cost  $\eta$  for each cooperator punished.
- 4) if there are no defectors  $c = 0$  to remove all costs and punishments

Figure 7 shows how punishing cooperators who don't punish defectors helps resolve the second-order free rider problem. The defector punishment was  $\beta = 6.9$  with a cost  $\alpha = 1.0$ . The green trajectory has  $\gamma = 0.0$  so there is no cooperator punishment. (This is the same as the green trajectory depicted in Figure 3). The red trajectory shows a cooperator punishment of  $\gamma = 2.2$ , with a cost to the punisher of  $\eta = 0.1$ . The defectors are rapidly switching to punishers who eventually take over the population—i.e.,  $p_2 = 1.0$  and  $p_1 = p_3 = 0.0$ . But a population of all punishers has the same affect on the shared resource as does a population of all cooperators; no defectors, no punishment costs so the returns are identical. Punishing defectors resolves the first-order free rider problem and punishing cooperators who don't punish resolves the

second-order free rider problem. More importantly, this dual punishment approach resolves the TOC at the same time.

In finite populations discrete replicator equations predict strategy evolution but some form of quantization is necessary. Quantization can produce fixed points in the simplex interior. Interior fixed points always have  $k_3 \geq 1$ . The TOC is only partially resolved in this case, which can prove problematic. The defectors that remain will continue to over-consume the shared resource and, unless the replenishment rate (20% after 10 rounds in these experiments) is sufficient, the resource will eventually become depleted. Under those circumstances  $\beta$  should be increased to force a complete resolving. An example is shown in Figure 3.  $\beta = 7.1$  resulted in the population reaching an interior fixed point whereas  $\beta = 10.0$  completely purged defectors from the population.

Defectors always do better in social dilemmas regardless of what others do. The simulation results show altruistic punishment can help resolve a TOC. These results are consistent with previous studies where it was shown punishment is most effective in iterated scenarios where group memberships don't change [4]. But there still is an open question: why do individuals choose particular strategies in a TOC? Defectors may be motivated purely by self interest. Other individuals may choose cooperation because they feel morally obligated to preserve a finite resource. But why would individuals choose to be a punisher, especially if punishment is costly?

Unfortunately replicator dynamics will probably *not* be very helpful in getting an answer. Replicator equations only describe how strategies in a population evolve over time. There is no mutation involved so new strategies cannot emerge and lost strategies cannot reappear. Evolution is dictated strictly in terms of fitness relative to the average population fitness. Put another way, replicator equations only provide proximate causes, not underlying reasons. For instance, Figure 7 shows

punisher growth. But to get this growth punishers had to address both the first-order and the second-order free rider problem and with high enough punishment levels. Punisher growth occurs if and only if the cost incurred is less than the punishment inflicted—i.e.,  $\beta/\alpha > 1$  and  $\gamma/\eta > 1$ . Note these punishment-to-cost ratios are necessary but not sufficient conditions. Recall the punishment inflicted depends on the number of available punishers (see Figure 5). Natural selection will still favor defectors or cooperators if punishers are rare. Replicator equations also do not explain why an individual decided to be a punisher in the first place.

The human experiments conducted by Fehr and Gächter [2] may provide some proximate explanations. Subjects participated in a PGG and afterwards recorded their anger and annoyance at free riders. They found subjects who contributed more were far more likely to be angry at free riders. They were also more likely to retaliate by inflicting punishment. Selp et al. [5] found that just witnessing non-cooperative behaviors provided the “extra fuel” needed for people to engage in costly punishment. Prior work by this author [6] showed that both anger and guilt are present in TOC social dilemmas.

These results regarding the role emotions play in social dilemmas have profound implications. In the past researchers have focused on Nowak’s five rules as the genesis of cooperative behavior—e.g., kin selection or reciprocity are essential mechanisms. Those mechanisms may provide insight for pairwise interactions, but not group dynamics. For example, directory reciprocity is the underlying mechanism in the tit-for-tat strategy in a 2-player iterated prisoner’s dilemma game. But now consider a 5-player PGG where 2 players cooperate and the other 2 defect. It is not obvious in this case how direct reciprocity helps the 5th player develop an effective strategy for the next round.

Rational individuals weigh the costs and benefits of new strategies before making any change but emotions are the triggering events that cause individuals to reassess their current strategy. Under what circumstances would a cooperator decide to become a punisher? A cooperator who is merely irritated by defectors may decide no strategy change is warranted; an outraged cooperator may decide differently. Emotions may also provide some insight into how high punishment levels are set and what costs punishers are willing to pay. Future efforts should focus on explaining the origins of altruistic punishment rather than mechanisms.

## REFERENCES

- [1] M. Nowak, “Five rules for the evolution of cooperation,” *Science*, vol. 314, no. 5805, pp. 1560–1563, 2006.
- [2] E. Fehr and S. Gächter, “Altruistic punishment in humans,” *Nature*, vol. 413, pp. 137–140, 2002.
- [3] V. Chandrasekhar, Y. Reznik, G. Takacs, D. Chen, S. Tsai, R. Grzeszczuk, and R. Giró, “Quantization schemes for low bitrate compressed histogram of gradients descriptors,” in *2010 IEEE Computer Vision and Pattern Recognition Workshops*, 2010, pp. 33–40.
- [4] D. Baillet, L. Mulder, and P. V. Lange, “Reward, punishment and cooperation: a meta-analysis,” *Psych. Bull.*, vol. 137, no. 4, pp. 594–615, 2011.
- [5] E. Selp, W. van Dijk, and M. Rotteveel, “On hotheads and dirty harries,” *Annals N. Y. Acad. Sci.*, vol. 1167, pp. 190–196, 2009.

- [6] G. Greenwood, “Evolving strategies to help resolve tragedy of the commons social dilemmas,” in *Proc. 2015 IEEE Conf. Comput. Intell. & Games*, 2015, pp. 383–390.

# Human Gesture Classification by Brute-Force Machine Learning for Exergaming in Physiotherapy

Francis Deboeverie, Sanne Roegiers, Gianni Allebosch, Peter Veelaert and Wilfried Philips

Department of Telecommunications and Information Processing,

Image Processing and Interpretation, UGent/iMinds,

St-Pietersnieuwstraat 41, 9000 Ghent, Belgium

Email: Francis.Deboeverie@ugent.be

**Abstract**—In this paper, a novel approach for human gesture classification on skeletal data is proposed for the application of exergaming in physiotherapy. Unlike existing methods, we propose to use a general classifier like Random Forests to recognize dynamic gestures. The temporal dimension is handled afterwards by majority voting in a sliding window over the consecutive predictions of the classifier. The gestures can have partially similar postures, such that the classifier will decide on the dissimilar postures. This brute-force classification strategy is permitted, because dynamic human gestures show sufficient dissimilar postures. Online continuous human gesture recognition can classify dynamic gestures in an early stage, which is a crucial advantage when controlling a game by automatic gesture recognition. Also, ground truth can be easily obtained, since all postures in a gesture get the same label, without any discretization into consecutive postures. This way, new gestures can be easily added, which is advantageous in adaptive game development. We evaluate our strategy by a leave-one-subject-out cross-validation on a self-captured stealth game gesture dataset and the publicly available Microsoft Research Cambridge-12 Kinect (MSRC-12) dataset. On the first dataset we achieve an excellent accuracy rate of 96.72%. Furthermore, we show that Random Forests perform better than Support Vector Machines. On the second dataset we achieve an accuracy rate of 98.37%, which is on average 3.57% better than existing methods.

## I. INTRODUCTION

Human gesture recognition [1], [2], [3], [4] is defined as automatically identifying and interpreting human body movements using a set of sensors. Human body movements may be performed with the hands, arms, body, head, etc. Human gestures may include for instance standing, lying, bending, sitting, walking, jumping, etc. Human gesture recognition has been heavily studied because it plays an important role in human computer interaction applications [5], [6], [7], [8] such as health monitoring systems, surveillance systems, motion analysis in sports, and human behavior analysis.

In this paper we perform human gesture recognition for the application of exergaming in physiotherapy. It is not always easy for children in a rehabilitation or fitness program to sustain their efforts. Exergaming, which combines exercise and gaming, can motivate children (and adults) to keep moving. Exergaming also offers the possibility for remote monitoring and coaching in an e-environment. Coaches and therapists can select the games with the desired level of difficulty and remotely monitor the children's progress. The project wE-

MOVE<sup>1</sup> is an innovative solution for exergaming that remotely supports the childrens rehabilitation and prompts them to move. The software consists of a gross motoric exergame and a platform allowing both the child and the coach to monitor progress. In this framework, automatic human gesture classification is needed to control the game.

Human gesture recognition is mainly performed on RGB-D (Red, Green, Blue and Depth) data [9], [10], [11], [12], [6], [13] or on skeleton data [14], [15], [16], [17], [18], [13], [19], [20], [21], where skeletal data can be extracted from RGB-D data. To recognize static gestures (i.e. postures, such as sitting, standing or lying), a general classifier [22] or a template-matcher is generally used. Dynamic gestures (i.e. consecutive postures, such as running, jumping) have a temporal dimension, which is traditionally handled by Hidden Markov Models (HMM) or motion based models. When classifying many dynamic human gestures, constructing these models is complex and time consuming. Also, the models are usually not generally applicable, so that it is difficult to extend the classifier with new gestures. Furthermore, building ground truth requires a discretization into consecutive postures of the dynamic gesture, which is again complex and time consuming.

In this work, we propose a novel approach which uses a general classifier [22], such as Random Forests (RF) [23], to recognize dynamic gestures in skeletal data. The gestures can have partially similar postures, such that the classifier will decide on the dissimilar postures. The temporal dimension is handled afterwards by majority voting in a sliding window over the consecutive predictions of the classifier. This way of online continuous human gesture recognition can recognize dynamic gestures in an early stage, since we build up reliability when sliding the window. This is a crucial advantage when controlling a game by automatic gesture recognition, since the feedback to the user should be given in real time. Also, ground truth can be easily obtained, since all postures in a dynamic gesture get the same label, without any discretization into consecutive postures. Furthermore, the classifier is general and can be easily extended with new gestures, which is advantageous in adaptive game development.

We elaborate RF combined with majority voting in a sliding

<sup>1</sup>More details can be found at <http://www.iminds.be/en/projects/2015/03/11/we-move>

window for human gesture recognition. RF are considered amongst the most robust classifiers currently available, and have been shown to perform as well as or better than Support Vector Machines (SVM), while being much less computationally expensive to train or execute. We consider normalized skeleton data provided by the Microsoft Kinect v2 [24]. The Kinect device is a motion sensing device which was originally designed for the Microsoft Xbox 360 video game console where the user is the controller. The device is composed of multiple sensors: an RGB camera to capture a colored video stream, a depth camera to compute the 3D environment and an infrared light sensor. Skeletal data is extracted from RGB-D images. Kinect v2 can detect up to six users at the same time and compute their skeletons in 3D with 25 joints representing body junctions like the feet, knees, hips, shoulders, elbows, wrists, head, etc. For each pose of a skeleton the position numbers and the angle numbers of the joints form a feature vector. These feature vectors are used to train a RF and classify gross motoric movements.

In our experiments, we evaluate the above-mentioned strategy on two datasets. The first dataset is a self-captured stealth game gesture dataset, including 5 human subjects performing 23 specific movements for a gross motor stealth game. The gestures have partial similarity, such as walking and running, or jumping low and jumping high. Our method is evaluated by leave-one-subject-out cross-validation (LOSubO CV) [25]. In the results we will show that RF and majority voting in a sliding window achieves an accuracy rate of 96.72%. Furthermore, we will show that RF perform better than SVM. The second dataset is the publicly available Microsoft Research Cambridge-12 Kinect (MSRC-12) gesture dataset [26]. The dataset includes 30 people performing 12 gestures. Among all publicly available datasets [27], [28], the MSRC-12 dataset is best suited for our application, since the ground truth annotation for each sequence marks the action point of the gesture as a single time instance at which the presence of the action is clear and that can be uniquely determined for all instances of the action. For a real-time application, such as a game, this is the point at which a recognition module is required to detect the presence of the gesture. Using LOSubO CV We achieve an accuracy rate of 98.37%, which is on average 3.57% better than existing methods [29], [30], [31].

The remainder of the paper is as follows. In Section II, we give an overview of the RF classifier. In Section III, we explain the strategy of majority voting in a sliding window. In Section IV, we evaluate and compare the proposed method on two datasets. Finally, in Section V we conclude the paper.

## II. RANDOM FORESTS

In this section we shortly review some basic work on RF for classification problems. A RF [23], [22] is an ensemble classifier composed of several binary decision trees, each trying to solve the same task.

Like any classifier, a decision tree takes a set of features as input, and returns a class label as its output. A decision tree consists of a set of nodes that are connected by branches.

Non-leaf-nodes are called decision-nodes. In binary decision trees, each decision-node has exactly two child-nodes. Each branch that connects a decision-node with its two child-nodes, corresponds to a binary decision value. During classification, each decision-node compares a specific feature value with a threshold value, and then follows one of the two branches that corresponds to binary outcome of this test. This process is repeated until a leaf-node is reached. Leaf-nodes in a decision tree correspond to class labels, and thus represent the final decision.

Although decision tree classifiers are easy to use and can be implemented extremely efficiently, training a decision tree is a difficult problem. During tree construction, one of the several available features have to be chosen for the binary test at each decision-node. Common methods to train decision trees are the ID3 algorithm and its successor, C4.5 [32], which resort to a greedy heuristic approach to determine the splitting criteria. At each decision-node, the information gain (also known as mutual information) for each possible splitting criterion is calculated, and the criterion yielding the highest information gain is chosen.

When choosing a classifier, we consider the bias-variance trade-off. The bias of a classifier represents the number of samples that would be consistently misclassified, if the classifier would be trained on different subsets of the complete training population. The variance of a classifier measures the variability of the number of misclassifications when different subsets of the training population are used. In general, classifiers that are able to fit the data well, exhibit low bias but high variance, while classifiers that result in more general decision boundaries yield high bias but low variance. While a low-bias, low-variance classifier is desirable, lowering the variance of a classifier, often implicitly increases the bias, and vice versa.

A decision tree is a low-bias high-variance classifier. An obvious way to lower the variance is to train multiple decision trees on different subsets of the population, and then use the average decision (i.e. regression) or a voting scheme (i.e. classification). However, in practice only a limited amount of training data is available, instead of the whole population. A well known method to approximate the distribution of the complete population if only a limited number of observations are available, is to construct multiple training samples by bootstrapping the original training dataset. Bootstrapping is a resampling method that is known by statisticians as sampling with replacement. Since sampling with replacement means that samples can be selected multiple times, this means that each bootstrapped sample contains duplicated data. As a result, each classifier that is trained on a different bootstrapped sample, will have slightly different decision boundaries. By aggregating the resulting decisions of each of these possibly high-variance classifiers, by means of averaging or voting, a low-variance classifier is obtained. This concept of bootstrap aggregating is called bagging. For bagging in RF, the number of trees in the forest is an important parameter to choose. The larger the better, but also the longer it will take to compute. In addition, the results will stop getting significantly better



Figure 1. In the training phase of the classifier, all postures in a dynamic gesture (grey zone) get the same label.

beyond a critical number of trees.

RF use bagging to reduce the variance of the final ensemble classifier, compared to a single decision tree. However, bagged trees exhibit a high correlation because of the duplicates in their training data and the similarity in their training method. Highly correlated trees would therefore make the same errors in similar regions of the feature space. This means that reducing the variance by means of bagging, increases the bias of the resulting classifier. To decrease the bias of the ensemble classifier, RF ensure diversity of the tree classifiers by introducing randomness into the splitting criterion: each time the training set is split, only a randomly selected subset of all features is considered for selecting the feature for the next decision-node.

Thus, a RF, is a low-bias, low-variance ensemble classifier, trained by bootstrapping and random feature selection. RF have been shown to be almost invariant to overfitting and robust to noise. Finally, classification by means of RF can be implemented extremely efficient, since each decision tree can simply be represented by a set of conditional statements.

As feature input for the classifier, we consider normalized skeleton data provided by the Microsoft Kinect v2 [24]. The skeletons are computed in 3D with 25 joints representing body junctions, where each joint consist of a position encoded in three numbers and an angle encoded as four quaternion numbers, which is a common encoding method in robotics. For each pose of a skeleton the position numbers and the quaternion numbers of 25 joints form a 175-dimensional feature vector. These feature vectors are used to train and classify gross motoric movements.

In the training phase of the classifier, all postures in a dynamic gesture get the same label, as illustrated in Figure 1, where all different postures in the grey zone get the same label. Different gestures can have partially similar postures. In this case, the classifier will decide on the dissimilar postures. In the results we will show that the selection of human gestures in many applications shows sufficient dissimilarity in the postures. Using this brute-force strategy, ground truth can be easily obtained, because only the beginnings and the endings of the gestures have to be indicated, without any discretization into consecutive postures. Furthermore, the classifier can be easily extended with new gestures, which is advantageous in adaptive game development.

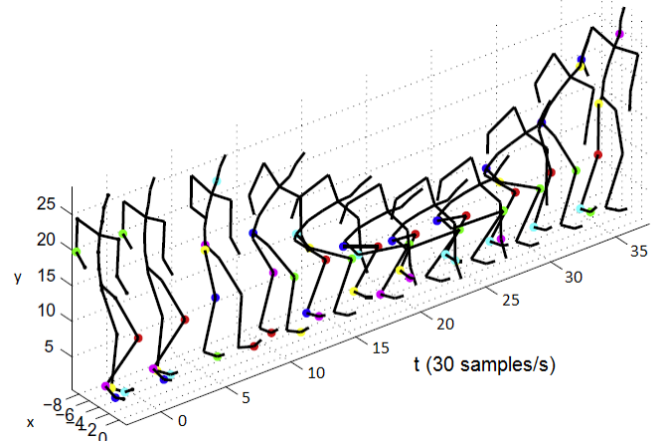


Figure 2. The temporal dimension of a human gesture in the skeletal data.

### III. MAJORITY VOTING IN A SLIDING WINDOW

In an online continuous human gesture recognition mode, the temporal dimension in human gestures is handled by majority voting in a sliding window over the consecutive predictions of the classifier. An example of the temporal dimension of a human gesture, i.e. bowing, in the skeletal data is illustrated in Figure 2.

In a sliding window, we compute the observation probability of a human gesture using a number of continuing observations within the sliding window. The final gesture type is decided by a majority vote of all recognition results that are obtained between the start and end point of the window. For optimal classification, the length of the time window is dependent on the duration of the gestures, and thus the selection of gestures and the subjects performing the gestures. In this work, the size of the sliding window  $w_s$  is determined empirically; in our work we found that one second was a good value, which means a buffer of 30 classifier predictions at a skeleton rate of 30Hz.

Using a sliding window technique in human gesture recognition introduces many advantages. The first advantage is that it improves the classification performance of gesture recognition greatly, as we will show in the results. A second advantage is that it reduces the undesirable effect of an abrupt change of observations within a short interval that can be caused by erroneous and incomplete skeletons. The third advantage is that human dynamic gestures can be recognized in an early stage, since we build up reliability when sliding the window. This is a crucial advantage when controlling a game with automatic gesture recognition, since the feedback to the user should be given in real time.

### IV. RESULTS

In this section, we evaluate the proposed strategy on two datasets: a self-captured stealth game gesture dataset and the Microsoft Research Cambridge-12 Kinect (MSRC-12) dataset.



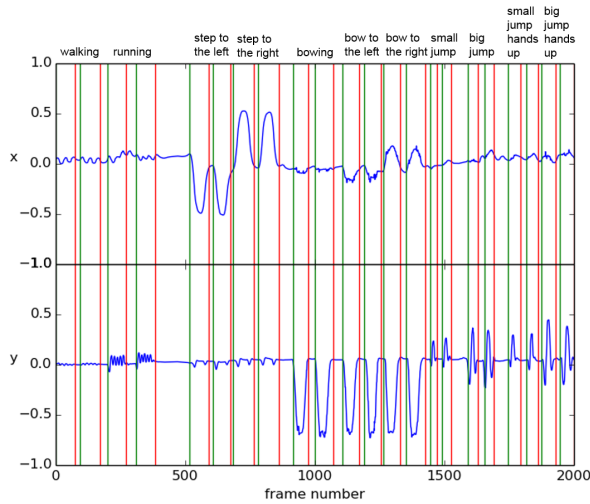


Figure 3. The normalized  $x$  and the  $y$  coordinates of the *spine\_mid* joint over time for 11 gestures. The green and the red lines indicate the beginnings and the endings of the gestures (ground truth), respectively.

#### A. Stealth game gesture dataset

In the first dataset, we recorded human subjects performing 23 specific movements for a gross motor stealth game, recorded with the Kinect v2 at the Sportlab of the department of Movement and Sport Sciences at Ghent University in Belgium. The dataset includes five subjects that repeat 23 exercises of a stealth game three times (26534 samples at 30Hz). Between every exercise the subject takes the neutral posture, which is standing up with the arms along the body.

This is the list of 23 movements with their corresponding label: 0: neutral, 1: walking, 2: running, 3: step to the left, 4: step to the right, 5: bowing, 6: bow to the left, 7: bow to the right, 8: little jump, 9: big jump, 10: little jump with the hands up, 11: big jump with the hands up, 12: climbing, 13: flying like a hummingbird, 14: flying with small arm movements, 15: flying with big arm movements, 16: punch to the left, 17: punch to the right, 18: pushing forward, 19: high kick to the left, 20: high kick to the right, 21: low kick to the left, 22: low kick to the right.

The graph in Figure 3 plots the normalized  $x$  and the  $y$  coordinates of the *spine\_mid* joint over time for 11 gestures. The green and the red lines indicate the beginnings and the endings of the gestures (ground truth), respectively. In the coordinates we can clearly distinguish the different expected patterns of the gestures.

Figure 4 plots the  $y$  coordinates over the  $x$  coordinates of the *spine\_mid* joint indicated in different colors for all gestures. This graph shows that the feature vectors of all postures in each dynamic gesture form a continuous cluster which is separable from the clusters of the other gestures.

Figure 5 shows the percentages of overlapping postures between the different gestures. The highest percentages of overlap are noticed between the gestures walking and running (up to 80%), and between the flying gestures (up to 60%).

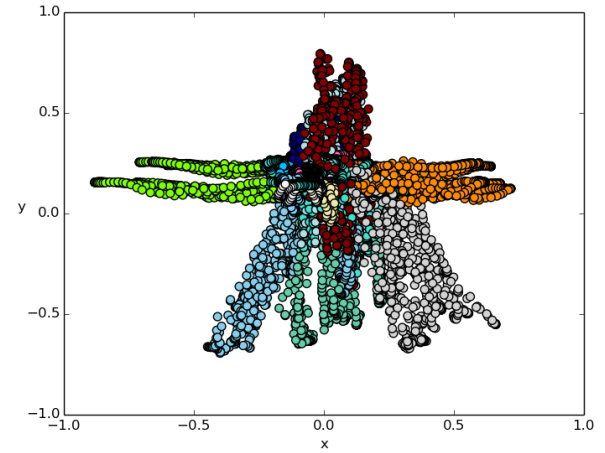


Figure 4. The  $y$  coordinates over the  $x$  coordinates of the *spine\_mid* joint indicated in different colors for all gestures.

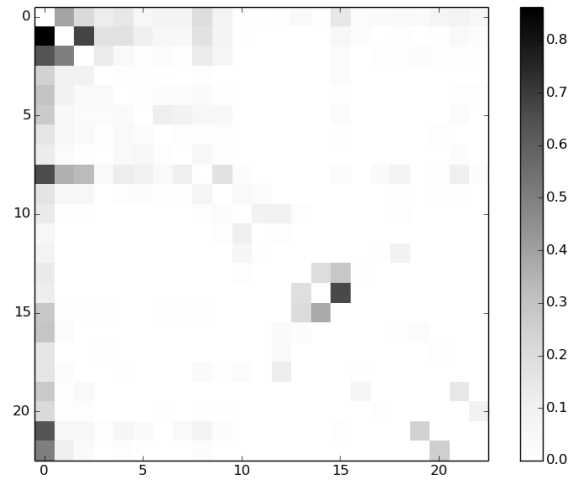


Figure 5. The percentages of overlapping postures between the different gestures.

Also, all gestures have a high overlap with the neutral gesture. This is because the neutral gesture occurs between every two other gestures. During annotation, a few samples of the neutral gesture may be included in another gesture. In our approach, despite the overlap, the remaining non-overlapping postures are sufficient to classify gestures.

We made an annotation of the dataset by indicating the beginnings and the ending of the gestures. The number of annotations we had to make is  $5 \text{ persons} \times 23 \text{ gestures} \times 3 \text{ repeats} \times 2 \text{ annotations} = 690 \text{ annotations}$ , instead of 26534 annotations in the case of a discretization of the dynamic gesture.

We evaluate our method by leave-one-subject-out cross-validation (LOSubO CV) [25], which is the most widely adopted evaluation protocol in action recognition algorithms towards maturity and robustness for real-world applications.

Table I

THE CLASSIFICATION ACCURACY RATES ON THE STEALTH GAME GESTURE DATASET WHEN EVALUATING AFTER 50% AND 100% FINISHING THE EXERCISE USING RF, SVM LINEAR AND SVM POLYNOMIAL COMBINED WITH MAJORITY VOTING IN A SLIDING WINDOW, RESPECTIVELY.

Method	Accuracy(%) 50% finished	Accuracy(%) 100% finished
SVM linear ( $C = 4.7$ ) + SW	$83.34 \pm 7.32$	$94.12 \pm 0.09$
SVM polynomial ( $D = 13$ ) + SW	$83.71 \pm 4.92$	$94.80 \pm 0.03$
RF ( $N = 27$ ) + SW	$88.26 \pm 4.23$	$96.72 \pm 0.02$

LOSubO CV means that the classifier is trained with all but one subject and tested with the unseen data. This is repeated for all subjects and the average of the outcomes as the final result is reported. Thus, in our case we perform a 5-fold cross-validation.

For the RF classifier, we choose the number of decision trees  $N$  in the forest equal to 27, which we determined empirically by a parameter sweep. Beyond this number of trees, the results stop getting significantly better. The size of the sliding window (SW)  $w_s$  is equal to 30. The window is initialized with labels of class 0 (neutral). As a comparison, we also test SVM in a one-against-one approach for multi-class classification using a linear and polynomial kernel, respectively. For the SVM with a linear kernel, we choose the penalty parameter  $C$  of the error term equal to 4.7. For the SVM with a polynomial kernel, we choose the polynomial degree  $D$  equal to 13. These values have been determined empirically for an optimal accuracy rate on the dataset. Table I presents the accuracy rates when evaluating the classification after 50% and 100% finishing the exercise, respectively. The accuracy is measured as the set of labels predicted for a sample that exactly match the corresponding set of labels in the ground truth. The observational latency is an important evaluation criterion in our game application. After 50% finishing the exercise, the RF +SW already classifies 88.26% of the gestures correctly. The classification accuracy increases to 96.72% after 100% finishing the exercise. Furthermore, these numbers also shows that the RF classifier performs better and faster than the SVM classifiers. The training times of the RF classifier, the linear SVM and the polynomial SVM are 12.65 seconds, 23.59 seconds and 27.78 seconds, respectively.

The precision, recall and f1-scores per human gesture are presented in the bar chart in Figure 6. Overall, the average precision, recall and f1-score, weighted with the number of class labels, are all equal to 0.97. Our method has a little less performance in recall on jumping and walking gestures, because these gestures have many similar postures with the neutral posture. This is further illustrated in the confusion matrix in Figure 7. The confusion matrix is a matrix which shows the accuracy of a classification algorithm, where each column of the matrix represents the instances in a predicted class, while each row represents the instances in an actual class. We can clearly see that our method has the biggest confusion in the classification of non-neutral gestures as neutral gestures, which is due to the accuracy of the ground truth and the occurrence

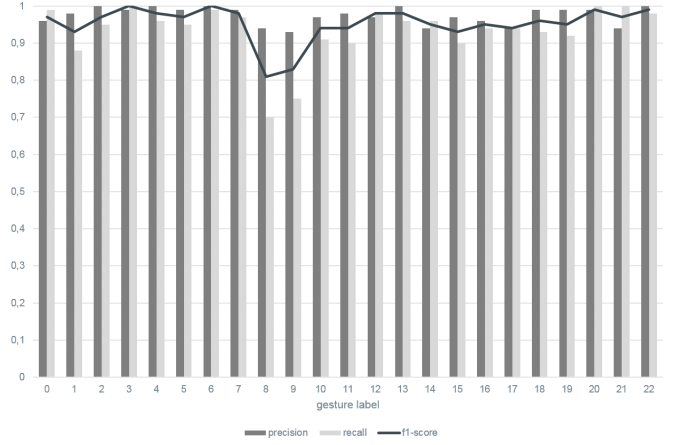


Figure 6. The precision, recall and f1-scores per human gesture.

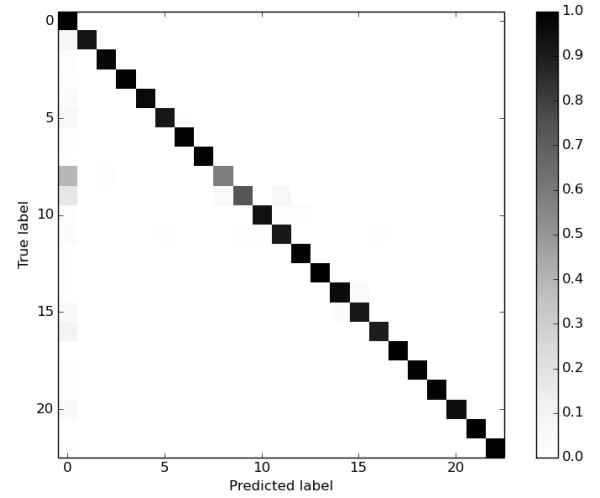


Figure 7. The confusion matrix showing the accuracy of the proposed classification algorithm.

of neutral postures in the non-neutral gestures. Regarding the accuracy of the ground truth, the beginnings and endings of the gestures can include a few overlapping neutral postures, causing the classifier to classify non-neutral gestures as neutral gesture.

Figure 8 presents a visualization of the application output. On the right hand side the skeleton of the posture of a subject performing gesture 15 (flying with big arm movements) is shown. On the left hand side we see the observational probability per gesture class in the sliding window. The probability of the class 0 (neutral) is decreasing to zero, the probability of the class 15 would increase to one in case of a perfect classification. However, in this case, the RF classifier makes a few mistakes by predicting class 14 (flying with small arm movements), due to the similar postures with class 15. These errors are handled by majority voting for the final class decision, which is printed in red color on the right hand side. Even though class 14 and 15 have similar postures, the classifier is still able to decide on the dissimilar postures. This



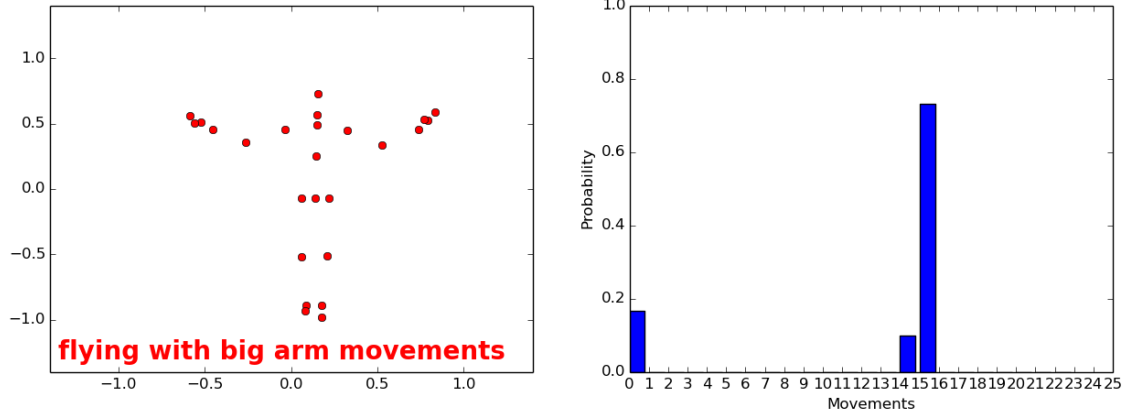


Figure 8. Visualization of the application output.

figure also demonstrates the advantage that human dynamic gestures can be recognized in an early stage, since we build up reliability in the sliding window when performing the exercise.

#### B. MSRC-12 dataset

The second dataset is the Microsoft Research Cambridge-12 Kinect (MSRC-12) gesture dataset [26], which consists of sequences of human movements, represented as body-part locations, and the associated gesture to be recognized by the system. The dataset includes 594 sequences and 719359 frames, approximately six hours and 40 minutes, collected from 30 people performing 12 gestures. In total, there are 6244 gesture instances. The motion files contain tracks of 20 joints estimated using the Kinect Pose Estimation pipeline. The body poses are captured at a sample rate of 30Hz with an accuracy of about two centimeters in joint positions. The list of movements with their corresponding label: 0:lift outstretched arms, 1:duck, 2:push right, 3:goggles, 4:wind it up, 5:shoot, 6:bow, 7:throw, 8:had enough, 9:change weapon, 10:beat both, 11: kick. Among all publicly available datasets [27], [28], the MSRC-12 dataset is best suited for our application, since the ground truth annotation for each sequence marks the action point of the gesture as a single time instance at which the presence of the action is clear and that can be uniquely determined for all instances of the action. For a real-time application, such as a game, this is the point at which a recognition module is required to detect the presence of the gesture.

For the RF classifier, we choose the number of decision trees  $N$  in the forest equal to 27, which gives an optimal classification rate on this dataset. The size of the sliding window is again  $w_s = 30$ . We compare our method to the methods in [29], [30], [31], which are the highest performing methods on this dataset as also reported in [27]. The methods implement human gesture recognition by decision forest based feature selection, a temporal hierarchy of covariance descriptors and sequence matching, respectively. Table II presents the accuracy rates when evaluating at the time instance marked in

Table II  
THE ACCURACY RATES WHEN EVALUATING AT THE TIME INSTANCE MARKED IN THE GROUND TRUTH OF THE DATASET USING THE PROPOSED RF +SW METHOD AND THE METHODS IN [29], [30], [31].

Method	Accuracy(%)
RDF-selected features [29]	94.03
Cov3DJ [30]	93.60
ESM [31]	96.76
RF ( $N = 23$ ) + SW	98.37

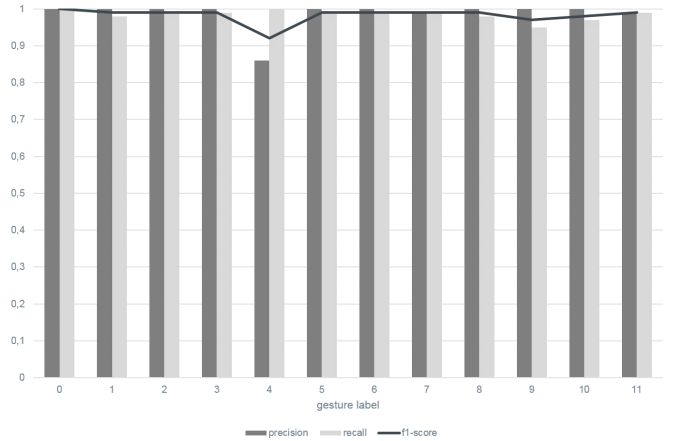


Figure 9. The precision, recall and f1-scores per human gesture on the MSRC-12 dataset.

the ground truth of the MSRC-12 dataset using LOSubO CV as evaluation protocol (30-fold cross-validation).

Our method achieves an accuracy rate of 98.37%, which is on average 3.57% better than the methods to which we compare.

The precision, recall and f1-scores per human gesture are presented in the bar chart in Figure 9. Overall, the average precision, recall and f1-score, weighted with the number of class labels, are all equal to 0.98, which is among the highest in literature.

## V. CONCLUSION

In this work, we proposed a novel approach for human gesture classification on skeletal data provided by Microsoft Kinect. We use Random Forests to recognize dynamic human gestures, where the temporal dimension is handled afterwards by majority voting in a sliding window over the consecutive predictions of the classifier. The gestures to be recognized have partially similar postures, such that the classifier decides on the dissimilar postures. We showed that this brute-force classification strategy is permitted because the selection of human gestures in many applications shows sufficient dissimilar postures. This way, ground truth can be easily obtained, because only the beginnings and the endings of the gestures have to be indicated, without any discretization into consecutive postures. Furthermore, the classifier can be easily extended with new gestures, which is advantageous in adaptive game development. Additionally, online continuous human gesture recognition can recognize dynamic gestures in an early stage, which is a crucial advantage when controlling a game by automatic gesture recognition. We evaluated our strategy by a leave-one-subject-out cross-validation on a self-captured stealth game gesture dataset and the Microsoft Research Cambridge-12 Kinect (MSRC-12) dataset. On the first dataset we achieved an accuracy rate of 96.72%. Moreover, we showed that in this application Random Forests perform better than Support Vector Machines. On the second dataset we achieved an accuracy rate of 98.37%, which is on average 3.57% better than existing methods. In this work, we proved that the proposed simple brute-force strategy of using a general classifier in combination with majority voting in a sliding window provides excellent classification results, while the annotation process went very fast.

## REFERENCES

- [1] J. K. Aggarwal and L. Xia, "Human activity recognition from 3d data: A review," *Pattern Recognition Letters*, vol. 48, pp. 70–80, 2014.
- [2] R. Lun and W. Zhao, "A survey of applications and human motion recognition with microsoft kinect," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 29, no. 05, 2015.
- [3] S. S. Rautaray and A. Agrawal, "Vision based hand gesture recognition for human computer interaction: a survey," *Artificial Intelligence Review*, vol. 43, no. 1, pp. 1–54, 2015.
- [4] L. L. Presti and M. La Cascia, "3d skeleton-based human action classification: A survey," *Pattern Recognition*, 2015.
- [5] P. Paliyawan, K. Sookhanaphibarn, W. Choensawat, and R. Thawonmas, "Body motion design and analysis for fighting game interface," in *Computational Intelligence and Games (CIG), 2015 IEEE Conference on*, 2015, pp. 360–367.
- [6] N. Li, Y. Dai, R. Wang, and Y. Shao, "Study on action recognition based on kinect and its application in rehabilitation training," in *Big Data and Cloud Computing (BDCloud), 2015 IEEE Fifth International Conference on*, 2015, pp. 265–269.
- [7] P. K. Pisharady and M. Saerbeck, "Recent methods and databases in vision-based hand gesture recognition: A review," *Computer Vision and Image Understanding*, vol. 141, pp. 152–165, 2015.
- [8] M. Devi, S. Saharia, and D. Bhattacharyya, "Dance gesture recognition: A survey," *International Journal of Computer Applications*, vol. 122, no. 5, 2015.
- [9] H. Kim, S. Lee, Y. Kim, S. Lee, D. Lee, J. Ju, and H. Myung, "Weighted joint-based human behavior recognition algorithm using only depth information for low-cost intelligent video-surveillance system," *Expert Systems with Applications*, vol. 45, pp. 131–141, 2016.
- [10] M. Liu and H. Liu, "Depth context: a new descriptor for human activity recognition by using sole depth sequences," *Neurocomputing*, vol. 175, pp. 747–758, 2016.
- [11] A.-A. Liu, N. Xu, Y.-T. Su, H. Lin, T. Hao, and Z.-X. Yang, "Single/multi-view human action recognition via regularized multi-task learning," *Neurocomputing*, vol. 151, pp. 544–553, 2015.
- [12] H. Eum, C. Yoon, H. Lee, and M. Park, "Continuous human action recognition using depth-mhi-hog and a spotter model," *Sensors*, vol. 15, no. 3, pp. 5197–5227, 2015.
- [13] F. Jiang, S. Zhang, S. Wu, Y. Gao, and D. Zhao, "Multi-layered gesture recognition with kinect," *The Journal of Machine Learning Research*, vol. 16, no. 1, pp. 227–254, 2015.
- [14] W. Ding, K. Liu, X. Fu, and F. Cheng, "Profile hmms for skeleton-based human action recognition," *Signal Processing: Image Communication*, 2016.
- [15] G. Zhu, L. Zhang, P. Shen, and J. Song, "An online continuous human action recognition algorithm based on the kinect sensor," *Sensors*, vol. 16, no. 2, p. 161, 2016.
- [16] —, "Human action recognition using multi-layer codebooks of key poses and atomic motions," *Signal Processing: Image Communication*, 2016.
- [17] G. Lu, Y. Zhou, X. Li, and M. Kudo, "Efficient action recognition via local position offset of 3d skeletal body joints," *Multimedia Tools and Applications*, pp. 1–16, 2015.
- [18] S. Boubou and E. Suzuki, "Classifying actions based on histogram of oriented velocity vectors," *Journal of Intelligent Information Systems*, vol. 44, no. 1, pp. 49–65, 2015.
- [19] J. Ding and C.-W. Chang, "Feature design scheme for kinect-based dtw human gesture recognition," *Multimedia Tools and Applications*, pp. 1–16, 2015.
- [20] —, "An eigenspace-based method with a user adaptation scheme for human gesture recognition by using kinect 3d data," *Applied Mathematical Modelling*, vol. 39, no. 19, pp. 5769–5777, 2015.
- [21] X. Chen and M. Koskela, "Skeleton-based action recognition with extreme learning machines," *Neurocomputing*, vol. 149, pp. 387–396, 2015.
- [22] Y. Kodratoff, *Introduction to machine learning*. Morgan Kaufmann, 2014.
- [23] L. Breiman, "Random forests," *Journal of Machine Learning*, vol. 45, no. 1, pp. 5–32, oct 2001.
- [24] Z. Zhang, "Microsoft kinect sensor and its effect," *MultiMedia, IEEE*, vol. 19, no. 2, pp. 4–10, 2012.
- [25] S. Arlot, A. Celisse *et al.*, "A survey of cross-validation procedures for model selection," *Statistics surveys*, vol. 4, pp. 40–79, 2010.
- [26] S. Fothergill, H. M. Mentis, P. Kohli, and S. Nowozin, "Instructing people for training gestural interactive systems," in *CHI*, J. A. Konstan, E. H. Chi, and K. Höök, Eds. ACM, 2012, pp. 1737–1746.
- [27] J. Zhang, W. Li, P. O. Ogunbona, P. Wang, and C. Tang, "Rgb-d-based action recognition datasets: A survey," *arXiv preprint arXiv:1601.05511*, 2016.
- [28] S. Ruffieux, D. Lalanne, E. Mugellini, and O. A. Khaled, "A survey of datasets for human gesture recognition," in *Human-Computer Interaction. Advanced Interaction Modalities and Techniques*, 2014, pp. 337–348.
- [29] F. Negin, F. Özdemir, C. B. Akgül, K. A. Yüksel, and A. Erçil, "A decision forest based feature selection framework for action recognition from rgb-depth cameras," in *Image Analysis and Recognition*, 2013, pp. 648–657.
- [30] M. E. Hussein, M. Torki, M. A. Gowayyed, and M. El-Saban, "Human action recognition using a temporal hierarchy of covariance descriptors on 3d joint locations," in *IJCAI*, vol. 13, 2013, pp. 2466–2472.
- [31] H.-J. Jung and K.-S. Hong, "Enhanced sequence matching for action recognition from 3d skeletal data," in *Computer Vision—ACCV 2014*, 2014, pp. 226–240.
- [32] J. R. Quinlan, *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., 1993.

# A Multi-Objective Genetic Algorithm for Simulating Optimal Fights in StarCraft II

Jonas Schmitt

Friedrich-Alexander-Universität Erlangen-Nürnberg

Email: jonas.schmitt@fau.de

Harald Köstler

Friedrich-Alexander-Universität Erlangen-Nürnberg

Email: harald.koestler@fau.de

**Abstract**—The goal of this work is to develop a multi-objective genetic algorithm for simulating optimal fights between arbitrary units in the real-time strategy game StarCraft II. As there is no freely available application programming interface for controlling units in the game directly, this first requires an accurate simulation of the actual game mechanics. Next, based on the concept of artificial potential fields a general behavior model is developed which allows controlling units in an optimal way based on a number of real-valued parameters. The goal of each individual unit is to maximize their damage output while minimizing the amount of received damage. Finding parameter values that control the units of two opposing players in an optimal way with respect to these objectives can be formulated as a multi-objective continuous optimization problem. This problem is then solved by applying a genetic algorithm that optimizes the behavior of each unit of two opposing players in a competitive way. To evaluate the quality of a solution, only a finite number of solutions of the opponent can be used. Therefore, the current optima are repeatedly exchanged between both players and serve as input for the simulated encounter. By comparing the solutions of both players at the end of the optimization, it can be estimated if one of the two players has an advantage. Finally, in order to evaluate the effectiveness of the presented approach, a number of sample build orders, which correspond to the amount of units that have been produced until a certain point of time, serve as input for several optimization runs.

## I. INTRODUCTION

Electronic sports (abbreviated eSports), which is a term for organized video game competitions, continues to gain in popularity. With prize pools comparable to and viewer numbers even surpassing those of traditional sport events<sup>1</sup>, an increasing number of video games is designed with an adaption as professional gaming playground in mind. A prominent example is StarCraft® II: Heart of the Swarm®, a real-time strategy game from Blizzard Entertainment<sup>2</sup>. It is not only present on most important international eSport tournaments currently held, but has even evolved into a professional sport within South Korea. In a typical game two players face each other. Both opponents can choose among three races (Terran, Zerg and Protoss), which differ in their strengths and weaknesses. Starting under the same conditions, both players' goal is to produce the right composition of units, defeat the opponent's units in combat

and ultimately win by destroying all of his structures. In general, a game can be divided into three different phases: Early, mid and late game. The early game mostly focuses on developing the economy and building up a decent army while occasionally fending off early aggression. In the mid game the majority of fights take place with both players continuously increasing the size of their armies. In case the game continues further without a clear winner, the end game phase is reached and both players are able to utilize their races' full potential, including the most powerful units and upgrades. In all three phases a player has to choose among numerous possibilities and strategies. The outcome of a game mainly depends on two aspects of a player's decision making: Macro- and micromanagement. The former describes the production order of structures and units and the latter how units are controlled in combat situations. To ensure that both opponents have equal winning chances, all three races must be balanced with respect to the aforementioned aspects in all phases of the game.

An overview about common techniques for optimizing AI behavior in real-time strategy games can be found in [1]. In [2] and [3] an approach for optimizing build orders, i.e., the order in which structures and units are produced by a player, is presented. These build orders then represent the potentially best macromanagement strategy a certain player can execute within a fixed amount of time. By simulating a battle between the units created within the respective build order, it can be predicted which of two opponents has an advantage at the current point of time in the game, assuming that all units are controlled optimally. Optimizing unit behavior in real-time strategy games based on parametrized artificial potential fields is a common approach (see for example [4] [5] [6] [7] [8]) that originally stems from robotics [9] [10]. In [11] it was shown that this approach is also suitable for StarCraft II, producing reasonable results for different types of units. However, only encounters between units of a single type were considered and the units' special abilities were completely ignored. Moreover, the simulation of unit behavior was based on simplified game mechanics and did not fully cover the actual in-game complexity of micromanagement. In this paper the shortcomings of this work are addressed and it is further extended to produce optimal combat behavior for heterogeneous unit groups in a reasonable accurate simulation of the StarCraft II games mechanics.

<sup>1</sup>According to the "Global Growth of eSports Report" by newzoo games market research from 2015

<sup>2</sup>©2013 Blizzard Entertainment, Inc. All rights reserved. StarCraft and Heart of the Swarm are trademarks or registered trademarks of Blizzard Entertainment, Inc., in the U.S. and/or other countries.

## II. SIMULATION

Until now there exists no freely available API (Application Programming Interface) for controlling units in StarCraft II. To ensure the relevance of eventual results, it is therefore necessary to simulate the game by accurately emulating the actual game mechanics. As this work focuses on the simulation and optimization of micromanagement, the following assumptions are made:

- Units can move and interact on a two-dimensional rectangular area that possesses neither obstacles nor height differences.
- Structures are ignored.
- No new units are created during the simulation. Only those occurring in a predefined build order are considered.

In StarCraft II each unit is characterized by a number of attributes each of which possesses a distinctive value. Additionally to these attributes units can have special characteristics and abilities, which are either unit or race specific. For example, all Zerg units are able to regenerate their health at a slow rate. Also, some attacks exhibit special characteristics, for instance, the attack of the Baneling, a Zerg unit, causes damage to all ground units within a circular area of influence, and after execution the Baneling is removed from the game. While characteristics are always active, special abilities either cost an additional resource called energy or can only be applied consecutively after a certain cooldown period has passed. An example is the Stalker's blink ability. If researched by the player, this unit is able to instantly teleport over a certain distance with respect to a cooldown between consecutive applications.

Despite these details, in principle a unit is able to perform the following actions at each arbitrary point of time in the game:

- **Attacking.** If the time span from the unit's last attack is smaller than its cooldown, it is able to attack exactly one unit within its range. The damage is computed according to the enemy's type and the applying bonuses. If present, special attack characteristics must be considered.
- **Moving.** If the unit decides not to or is not able to attack, it can move in each direction according to its speed.
- **Special Abilities.** If the unit possesses special abilities, these can be applied either additionally or instead of the aforementioned actions depending on the characteristics of the ability.

### A. Potential Fields

To control units such that they behave optimally in combat, artificial potential fields are employed. In the context of a real-time strategy game bodies correspond to units and forces denote the direction each unit moves. Thus, by artificially creating different potential fields around a unit's position other units' interactions with it can be defined. The goal of each individual unit is to deal a maximum amount of damage while receiving a minimum amount at the same time. As a unit can only deal damage as long as an enemy stays within its attack

range, an attractive potential is required that forces the unit to approach enemy units until this condition is fulfilled. On the contrary, a unit can only take damage as long as it stays within the enemy's range, therefore a repulsive potential is required that forces the unit to retreat. Finally, friendly units should not act on their own, but as a group instead, both approaching and retreating collectively. This is realized by adding a third potential which creates an artificial attraction between friendly units and to a certain degree forces them to follow their companions' paths instead. A unit's movement direction is then computed by accumulating the gradients of all potential fields evaluated at the respective position.

While the overall behavior is clear, each potential field comes with certain degrees of freedom, as it is unclear how the various unit attributes should contribute in an arbitrary situation. It is especially not obvious how the magnitudes of the resulting forces should compare to each other in case that multiple fields overlap at a position. In order to achieve optimal combat behavior it is necessary to parametrize the described potential fields, such that all relevant attributes can be weighted arbitrarily. Therefore, they are modeled as functions of the distance between both involved units which contain 14 additional parameters with a range of values of  $[0, 1]$ , as described in Figure 1, 2 and 3. Additional factors that occur within their computation are summarized in I. As the values of the coefficients that occur within the computation of the gradients of these functions differ significantly, they are mapped to the same uniform range.

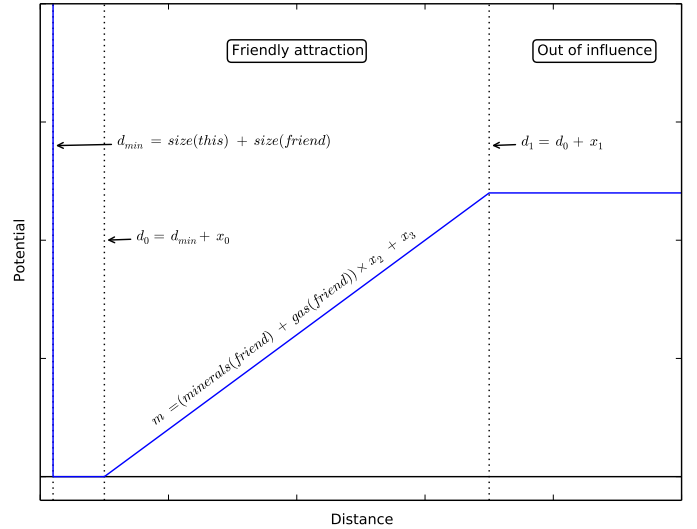


Fig. 1: Function describing the attractive potential of friendly units.

Figure 1 denotes the attractive potential of friendly units with a range that can be arbitrarily chosen according to two parameters. The magnitude of the gradient and thus the strength of the resulting attraction mainly depends on the amount of resources, i.e., minerals and vespene gas, required to produce the unit. The costlier a unit, the more important it usually is, making it necessary to accompany and protect it

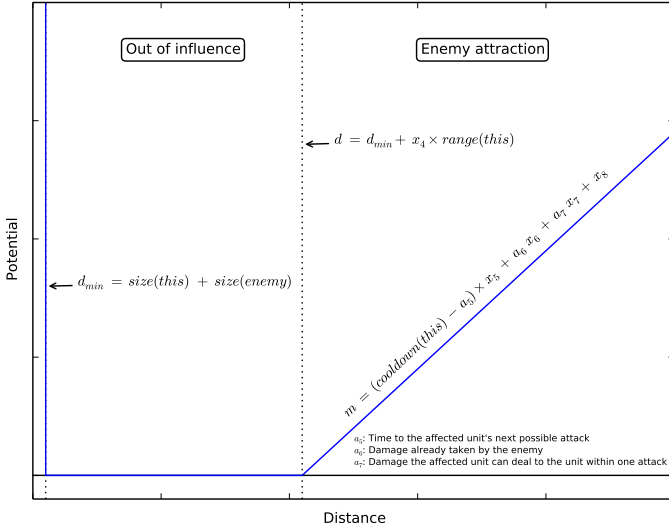


Fig. 2: Function describing the attractive potential of enemy units.

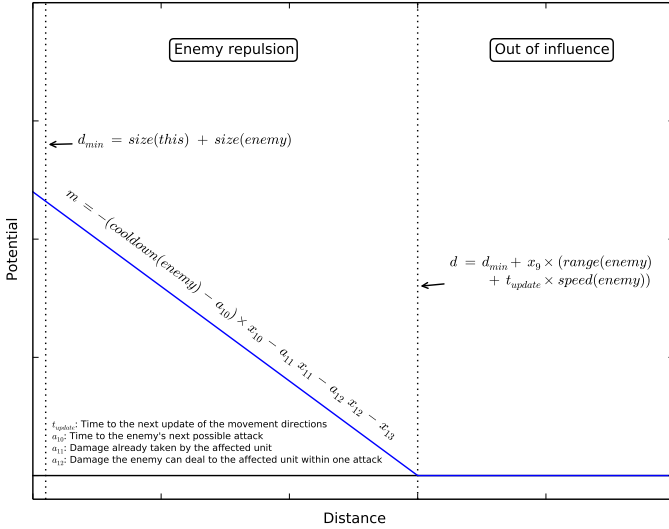


Fig. 3: Function describing the repulsive potential of enemy units.

TABLE I: Additional factors that occur within the potential field computation.

$a_5$	Time to the affected unit's next possible attack
$a_6$	Damage already taken by the enemy unit during the simulation
$a_7$	Damage the affected unit can deal to the enemy unit within one attack
$t_{update}$	Time to the next update of the movement directions
$a_{10}$	Time to the enemy unit's next possible attack
$a_{11}$	Damage already taken by the affected unit during the simulation
$a_{12}$	Damage the enemy can deal to the affected unit within one attack

with a number of cheaper units. Additionally, collisions, which happen when friendly units approach each other at a distance smaller than the sum of their two sizes, need to be avoided. This is achieved by setting the magnitude of the potential to a value near infinity, such that the two units are forced to

move apart until the condition is fulfilled. Figure 2 denotes the attractive potential of enemy units. Its range depends on the attack range of the affected unit while the magnitude of the attraction is controlled by the following parameters:

- The less time is left until the affected unit is able to attack the higher the attraction, as approaching the enemy only makes sense when an attack is possible.
- The more damage the enemy has already taken the easier it is to kill and thus the higher is the attraction.
- The more damage can be applied to the enemy within one attack the higher the attraction.

Collisions between opposing units also need to be avoided which is accomplished in the same way as described above.

Finally, Figure 3 denotes the repulsive potential of enemy units. Its range depends on the attack range of the enemy unit, but as an additional safety buffer, the movement speed of the enemy is also taken into account. The magnitude of the repulsion is controlled by the same parameters applying to Figure 2, but from the perspective of the enemy unit. Besides all relevant unit attributes, the concrete behavior of the described potential fields entirely depends on the 14 parameters  $x_0, \dots, x_{13}$  that occur within their calculation.

### B. Workflow

As a human player is only able to perform a limited number of actions per second, it is reasonable to allow making decisions only at discrete time steps. After fixing all relevant parameters, the outcome of an encounter between two groups of units is simulated in the following way. At the beginning the units of both players are placed at equidistant positions on opposite sides of the field. During a time step the following actions are performed by each unit:

- If attacking is possible, a target is chosen among all enemies within attack range by favoring units that can be defeated if the wasted damage<sup>3</sup> does not exceed a certain threshold, which is set by an additional parameter  $x_{14}$ . These are then prioritized by the amount of applicable damage.
- Else, the unit moves and its position at the next time step is computed with the following equation:

$$p_{i+1} = p_i + \frac{\vec{F}}{\|\vec{F}\|} \times s \quad (1)$$

where  $p_i$  is the position at time step  $i$ ,  $\vec{F}$  the current force and  $s$  the movement range within one time step<sup>4</sup>.

The force of each unit is recomputed after a fixed number of time steps by accumulating the gradients of all potential fields applying to it:

$$\vec{F} = \sum_{j=1}^n \vec{F}_j \quad (2)$$

<sup>3</sup>The difference between the remaining health of the target and the damage the attack could theoretically deal to a unit with the same attributes.

<sup>4</sup>The movement range equals the product of the length of a time step and the speed of the unit.

where  $n$  is the number of potential fields affecting the unit and  $\vec{F}_j$  the gradient of the  $j$ th potential field. In case a unit exhibits special abilities, these need to be modeled and considered during each time step, too. If their application incurs any kind of decision making, additional parameters are introduced<sup>5</sup>. The simulation finishes when either all units of a player have been defeated or its duration exceeds a certain limit. As described above, the behavior of each individual unit is controlled by the values of 15 real-valued parameters, where  $x_0 \dots x_{13}$  describe the movement behavior and  $x_{14}$  sets the attack threshold in case of multiple targets, plus possibly additional parameters controlling the use of special abilities. In consequence, the values of these parameters completely describe the behavior of each individual unit.

What is left to be answered is how many sets of parameters should be assigned to the units of a build order. Compared to the overall amount of units in a build order the number of different units is small. Units of the same kind share attributes and special abilities, such as for example the same range and movement speed, which facilitates the synchronization of their behavior. Moreover, it can be expected that acting as a group increases their combat effectiveness because it enables them to combine their attacks and kill single enemy units faster. Therefore, only one set of parameter values is assigned to all units of the same kind<sup>6</sup>. This does not only prevent a linear growth of the number of parameters with the length of a build order, but also enables the optimization of the behavior of similar units in a consistent way, which is covered in the following section.

### III. OPTIMIZATION

#### A. Problem Formulation

The goal of this work is to predict which of two players wins in a combat based on the units produced and assuming optimal control. According to the last section, the behavior of a subgroup of units of the same kind is controlled by at least 15 parameters with a range of values of  $[0, 1]$ . The combat effectiveness of each individual unit is represented by two objectives: The overall damage the unit is able to cause to enemy units ( $f_d$ ) and how much of its own health it is able to preserve ( $f_h$ ). For the sake of simplicity, from now on the set of parameters that describes the combined behavior of all units of a player is denoted as his *strategy*. Under the assumption that the optimal strategy of the opponent is known, its effectiveness according to both objectives can be evaluated by simulating the respective encounter with both strategies as input. Calculating the overall success of a strategy with respect to an objective can be simply achieved by adding up the corresponding measure at the end of the simulated encounter. Not all units that possess the same amount health

are also equally important. For example the sum of health of three Zerglings is slightly larger than that of one Infestor, although the latter unit is much more valuable, possessing powerful special abilities, which needs to be incorporated when evaluating both objectives. The cost (in minerals and vespene gas) of a unit represents a good approximation of its importance and is therefore used as weighting factor when evaluating the respective objective. Furthermore, to prevent units from being completely passive, an additional penalty is introduced: The health of units that have not attacked at least once during the simulation is not taken into account when evaluating the outcome. The effectiveness of a strategy with respect to the two objectives,  $f_d$  and  $f_h$ , is then calculated by the following formulas from the outcome of a simulated encounter with  $U$  as the set of units of the player whose strategy is evaluated and  $V$  as the set of units of his opponent:

$$f_d(U, V) = \sum_{v \in V} (\text{minerals}(v) + \text{gas}(v)) \times \text{damage}(v) \quad (3)$$

where  $\text{damage}(v)$  is the sum of health and shield the unit  $v$  has lost, measured at the end of the simulation.

$$f_h(U, V) = \sum_{u \in U} (w(u) \times (\text{minerals}(u) + \text{gas}(u)) \times (\text{health}(u) + \text{shield}(u))) \quad (4)$$

where  $\text{health}(u)$  is the remaining health and  $\text{shield}(u)$  the remaining shield<sup>7</sup> of unit  $u$ , measured at the end of the simulation. The weighting function  $w(u)$  is defined as follows:

$$w(u) = \begin{cases} 1 & \text{if } u \text{ has attacked} \\ 0 & \text{otherwise} \end{cases}$$

Maximizing  $f_d$  and  $f_h$  as output of a simulation with the strategy of the first player as input leads to a multi-objective continuous optimization problem. Because of the untraceable relationship between in- and output of a simulation, gradient-based methods are not applicable. As a remedy search heuristics which do not require such knowledge can be applied, such as genetic algorithm. To solve the problem of competitively optimizing the strategies of two players, a multi-objective genetic algorithm based on NSGA-II [12] is employed.

#### B. Genetic Algorithm

In the following the different parts of the algorithm are described in detail.

1) *Fitness Evaluation*: Finding optimal strategies for both players would theoretically require to evaluate all possible solutions against each other. Because the search space is continuous the number of possible solutions is uncountable. To avoid this dilemma and still get a reasonable fitness approximation, each solution is only evaluated against a limited number of the best solutions of the opponent. At the beginning,  $n$  additional solutions are generated randomly and exchanged

<sup>5</sup>Describing them all goes beyond the scope of this work but it is assured that all units occurring within the tested build orders are modeled completely, including all special abilities, parameterizing relevant behavioral aspects.

<sup>6</sup>For example in case of a build order containing four marines and two marauders, one set of parameters is assigned to the four marines and another one to the two marauders.

<sup>7</sup>For Terran and Zerg units the shield is always zero.

between both populations. To evaluate an individual, these solutions act as second input for  $n$  simulations and objective function evaluations whose average then forms its fitness in both objectives:

$$f_d(X) = \frac{1}{n} \sum_{i=1}^n f_d(U_X, U_{Y_i}) \quad (5)$$

$$f_h(X) = \frac{1}{n} \sum_{i=1}^n f_h(U_X, U_{Y_i}) \quad (6)$$

$f_d(X)$  denotes the fitness of individual  $X$  in the first and  $f_h(X)$  in the second objective.  $Y_1, \dots, Y_n$  are the  $n$  individuals currently used for fitness evaluation.  $U_X$  and  $U_{Y_i}$  denote the corresponding sets of units at the end of a simulation with  $X$  as input for the first and  $Y_i$  as input for the second player.

After a fixed number of generations both players exchange their  $n$  best solutions, which then replace the previous ones used for fitness evaluation. This scheme results in a competitive optimization of both populations. By repeatedly exchanging optima, both populations are forced to adapt to changing requirements, which prevents them from overfitting with respect to a subset of possible solutions of the opponent. After passing the last generation, the individuals of both population can finally be evaluated against each other to examine the effectiveness of the corresponding build orders in a direct comparison.

2) *Crossover*: The purpose of crossover is to produce improved solutions by combining the chromosome of existing ones, selected from the current population. In order to find the best crossover operator for the existing case, their performance is tested by competitively optimizing two sample build orders that include units with divers characteristics and abilities.

- **Player 1:** 4 Marines with Combat Shield and Stimpack, 2 Marauders with Stimpack
- **Player 2:** 2 Zealots with Charge, 2 Stalkers with Blink

The operators are tested in combination with binary tournament selection and the three different mutation operators mentioned in Table III, in two optimization runs over 500 iterations and with population sizes of 1600 for both opponents. Each individuals is evaluated against ten strategies. Every tenth iteration, the objective function is adapted by exchanging optima between both populations. As self-adaptive simulated binary crossover requires additional objective function evaluations to achieve comparable runtimes, the number of generations is reduced by 30 %. The results of the experiments are summarized in Table II. According to this comparison, self-adaptive simulated binary crossover provides superior performance, surpassing the other tested operators in all relevant metrics. It is therefore exclusively considered for the implementation of crossover within the presented algorithm, and its functioning is briefly described in the following.

<sup>8</sup>Online performance denotes the average of all objective function evaluations.

<sup>9</sup>Offline performance denotes the average of the optima of all generation.

<sup>10</sup>The number of crossover points is one less than the number of different units in the build order, but at least one.

TABLE II: Results for different crossover operators. Online and offline performance are averaged over both populations and all runs.

	Online Performance <sup>8</sup> (Damage, Health)	Offline Performance <sup>9</sup> (Damage, Health)	Number of Evaluations	Runtime
N-Point Crossover [13] <sup>10</sup>	66.85, 41.61	90.64, 62.60	60048000	68 min
Uniform Crossover [14]	69.04, 37.67	89.88, 57.49	60048000	64 min
Intermediate Crossover [15]	51.53, 51.94	76.70, 73.21	60048000	61 min
Line Crossover [15]	52.24, 52.78	82.14, 72.50	60048000	60 min
SBX [16]	53.25, 51.27	84.02, 70.94	60048000	66 min
Self-Adaptive SBX [17]	76.90, 54.49	98.59, 80.58	61602960	62 min

3) *Mutation*: Mutation in the context of genetic algorithms means altering single parts of an individual's chromosome (genes) with the purpose of maintaining genetic diversity in the population. The chance of mutation is determined by the mutation probability. To decide which genes to mutate theoretically requires the creation of  $n$  random numbers for each individual where  $n$  is the length of the chromosome. In [18] different alternative schemes have been investigated. According to this comparison, the suggested mutation clock scheme possesses superior performance. In the context of real-coded genetic algorithms, different mutation operators for altering single genes have been proposed. To determine which one leads to the best performance in combination with adaptive SBX, three common mutation operators are tested in ten optimization runs with the same setting used in the crossover evaluation. The results can be seen in Table III. Overall,

TABLE III: Results for different mutation operators. Online and offline performance are averaged over both populations and all runs.

	Online Performance (Damage, Health)	Offline Performance (Damage, Health)	Number of Evaluations	Runtime
Uniform Mutation <sup>11</sup>	84.81, 52.51	99.17, 79.69	170890850	190 min
Polynomial Mutation [18]	79.28, 50.31	98.60, 77.50	166659150	184 min
Gaussian Mutation [18]	62.48, 62.68	94.66, 90.81	165756100	203 min

uniform mutation performs slightly better than polynomial mutation, but possesses similar characteristics. Gaussian mutation on the other hand leads to more balanced results with respect to both objectives while sacrificing online performance in the first objective. It is although the only mutation operator that is able to guide the search towards finding nearly optimal solutions in both objectives, with equally good offline performance and should therefore be preferred.

4) *Main Loop*: By combining the aforementioned components, the general structure of the genetic algorithm is summarized in Algorithm 1.

<sup>11</sup>The old value of the parameter is replaced by a uniformly distributed random value.



**Algorithm 1** Main Loop

---

```

1: procedure MAIN( $p, m, n, s$ )
2:   ▷ Initialize both players
3:   INITIALIZE( $P^{(1)}, p, S^{(1)}, s$ )
4:   INITIALIZE( $P^{(2)}, p, S^{(2)}, s$ )
5:   ▷ Run the genetic algorithm on both populations for
      $m \times n$  generations
6:   for  $i = 0 \dots m$  do
7:     OPTIMIZE( $P^{(1)}, p, n, S^{(1)}, s, \hat{S}^{(2)}$ )
8:     OPTIMIZE( $P^{(2)}, p, n, S^{(2)}, s, \hat{S}^{(1)}$ )
9:     ▷ Exchange the  $s$  best individuals
10:     $S^{(1)} = \hat{S}^{(1)}$ 
11:     $S^{(2)} = \hat{S}^{(2)}$ 
12:    ▷ Reevaluate both populations
13:    EVALUATE-AND-SORT( $P^{(1)}, S^{(1)}$ )
14:    EVALUATE-AND-SORT( $P^{(2)}, S^{(2)}$ )

```

---



---

```

1: procedure INITIALIZE( $P, p, S, s$ )
2:   ▷ randomly generate an initial population  $P$  with  $p$ 
     individuals
3:   GENERATE-POPULATION( $P, p$ )
4:   CHOOSE-STRATEGIES( $S, s$ ) ▷ randomly choose  $s$ 
     evaluation strategies  $S = \{S_1, \dots, S_s\}$ 
5:   EVALUATE-AND-SORT( $P, S$ )

```

---



---

```

1: procedure EVALUATE-AND-SORT( $P, S$ )
2:   EVALUATE( $P, S$ ) ▷ evaluate the individuals con-
     tained in  $P$  using  $S$ 
3:   NON-DOMINATED-SORTING( $P$ )
4:   CROWDING-DISTANCE-ASSIGNMENT( $P$ )

```

---



---

```

1: procedure OPTIMIZE( $P, p, n, S, s, \hat{S}$ )
2:   for  $i = 1 \dots n$  do
3:      $P' = \text{SELECT}(P)$  ▷ select  $p$  individuals from  $P$ 
     and save them in  $P'$ 
4:      $Q = \text{CROSSOVER}(P')$  ▷ create  $p$  new individuals
      $Q$  from  $P'$  using crossover
5:     MUTATE( $Q$ ) ▷ mutate a subset of the individuals
     contained in  $Q$ 
6:     EVALUATE( $Q, S$ )
7:      $P = Q \cup P$ 
8:     NON-DOMINATED-SORTING( $P$ )
9:     CROWDING-DISTANCE-ASSIGNMENT( $P$ )
10:     $P = P \setminus \{P_{p+1}, \dots, P_{2p}\}$  ▷ remove the worst
     individuals from  $P$  such that  $|P| = p$ 
11:     $\hat{S} = P \setminus \{P_{s+1}, \dots, P_p\}$  ▷ store the  $s$  best individuals
     of  $P$  in  $\hat{S}$ 

```

---

## IV. RESULTS AND DISCUSSION

To evaluate the presented approach, one sample build order is chosen for all three races which contains units that in sum

cost the same amount of resources<sup>12</sup>:

- Zerg: 2 Zerglings, 5 Roaches.
- Terran: 5 Marines, 2 Marauders.
- Protoss: 2 Zealots, 2 Stalkers.

These build orders are then pairwise optimized in five test runs using the following settings: For both players a total population of 5000 is maintained, which is then optimized over 200 generations using ten individuals for fitness evaluation which are exchanged every tenth generation. For selection a tournament size of two is chosen and the mutation probability is set to  $\frac{1}{n}$  where  $n$  is the length of the chromosome. To simulate the game with a reasonable accuracy, time steps with a length of 10 milliseconds are used, and the movement direction of each unit is updated every tenth time step. Zerg units possess an increased movement speed when they operate on creep. It is a special ground layer that is automatically created by Zerg buildings and some units and can be used tactically. To estimate the importance of creep placement, the Zerg build order is evaluated once with and once without assuming creep on the ground. After an optimization run is finished, the 100 best individuals of both players are evaluated in a direct competition, such that each individual is evaluated against all 100 of the other player. By computing the average of all test runs an estimate can be obtained if the build orders are balanced with respect to each other. If one build order is able to outperform the other one in all runs, it probably has an advantage, at least under the assumptions that have been made to simulate the game.

Table IV shows the results of all encounters.  $\bar{f}_d$  is the average damage in percent that the units of the denoted player were able to cause to their enemies in the final comparisons and  $\sigma_d$  the standard deviation of all test runs. As the average health of the units of one player is correlated to the damage caused by the units of the other player, it is omitted in the table. Without creep the Zerg build order is inferior to the

TABLE IV: Results of the final comparison of the investigated encounters.

Encounter	$\bar{f}_d$ (Player 1)	$\sigma_d$ (Player 1)	$\bar{f}_d$ (Player 2)	$\sigma_d$ (Player 2)	Advantage
Zerg vs. Terran	28.90	3.96	92.48	4.68	Terran
Zerg vs. Terran (Creep)	67.42	6.17	79.86	11.87	None
Zerg vs. Protoss	29.47	7.13	71.40	1.97	Protoss
Zerg vs. Protoss (Creep)	64.64	14.28	61.64	7.38	None
Terran vs. Protoss	59.57	4.97	86.60	5.38	Protoss

other two, as in all test runs its average damage output is lower than its enemy's. Although, the picture changes when creep is assumed. Now, in both encounters the difference between the averages of both players is relatively small and no clear winner can be determined. Thus, it can not be assumed that any of both opponents has a significant advantage. In general, it can be concluded that the closer the outcome of an encounter is, which corresponds to a small difference between the averages of both players, the lower is the standard deviation. In three of the investigated encounters a clear winner, who prevails in

<sup>12</sup>Minerals and vespene gas are weighted equally.



all test runs over his opponent, could be determined. In the other two cases there are a number of factors that could have prevented the algorithm from reaching the same final state in all test runs. First of all, it is possible that one or multiple parameters of the algorithm have been chosen in an insufficient way. Increasing the population size, number of generations, fitness evaluations per individual or including more individuals into the final comparison could possibly lead to the desired stability. If none of the above factors leads to an improvement, an adaption of the algorithm must to be considered.

#### A. Evaluation of Unit Behavior

Finally, it is examined if the algorithm is able to produce unit behavior that can also be considered as reasonable from an outer perspective. For this purpose, a fight between the two solutions of each encounter that perform best in the final comparison is simulated and the paths of all units are tracked. The resulting plots can be seen in Figure 4 to 6.

First of all, compared to the preceding work [11] the resulting paths are significantly more fine-grained. The improved accuracy of the simulation allows a more precise control of the individual units, which leads to a higher number of changes in the movement directions because each unit tries to outmaneuver its enemies. In general, the resulting paths are of complicated nature and can not be easily described. All units have in common that they continuously try to stay in the sinks of the accumulated potentials, which corresponds to the area where they are able to attack while remaining unharmed by their enemies. This results in a competitive back and forth movement between opposing units, and is represented by the zigzag patterns occurring in all of the visualized encounters.

Moreover, there are a number of recurring patterns that evidently correspond to behavior that can be considered as intelligent. As it was explained at the end of Section II, the formation of collective behavior is facilitated by parameter sharing. Therefore, as expected, the units do not act as sole individuals but instead as a group which often results in the connection of their paths. Especially units of the same kind usually form subgroups, for example, the blue paths of the Marines in Figure 4. As they are able to move with the same speed and attack at the same range, it is easy for them to synchronize their behavior in an effective way. For example melee units such as Zealots all share the goal of approaching the enemy as close as possible, because otherwise they are unable to attack. On the other hand, ranged units try to position themselves out of their enemies range, such that they are able to attack but stay unharmed at the same time. By acting as a group units are able to exploit these similarities and combine their combat strength in a synchronous way. Another evident behavior is that faster units effectively utilize their speed advantage. For example in the encounter between Protoss and Terran, denoted by Figure 6, the Stalkers (dark blue) constantly avoid the lower ranged Marines (red), which is reflected in their large area of movement.

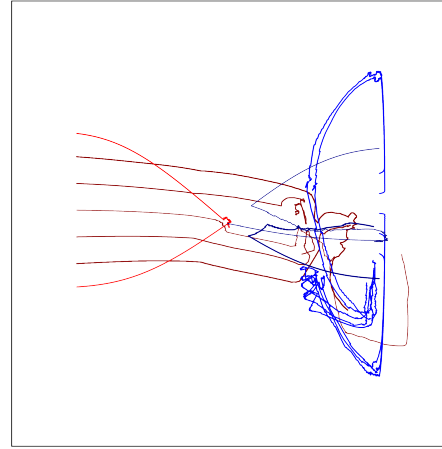


Fig. 4: Unit paths of an encounter between Zerg with 2 Zerglings (red), 5 Roaches (dark red) and Terran with 5 Marines (blue), 2 Marauders (dark blue). Winner: Terran.

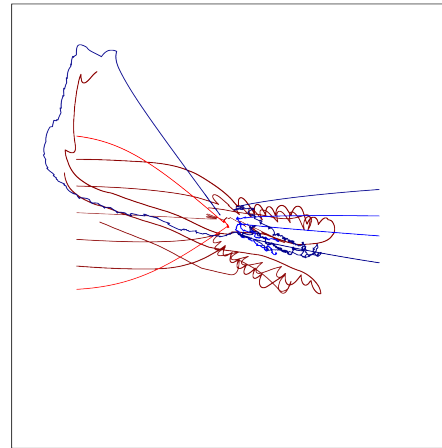


Fig. 5: Unit paths of an encounter between Zerg with 2 Zerglings (red), 5 Roaches (dark red) and Protoss with 2 Zealots (blue), 2 Stalkers (dark blue). Winner: Protoss.

## V. CONCLUSION AND FUTURE WORK

In this work a multi-objective genetic algorithm for simulating optimal fighting behavior in StarCraft II was presented. Its application aims to investigate the balancing of the game. By evaluating optimal build orders generated by the approach presented in [3] possible imbalances between the three races could be detected. However, as stable results could not be achieved in all cases, the approach needs further assessment, possibly requiring algorithmic improvements to achieve the mentioned goal. With the recent release of the second StarCraft II expansion pack Legacy of the Void<sup>13</sup>, an adaption of the simulation can be considered, which would

<sup>13</sup>Legacy of the Void is a trademark and StarCraft is a trademark or registered trademark of Blizzard Entertainment, Inc., in the U.S. and/or other countries.

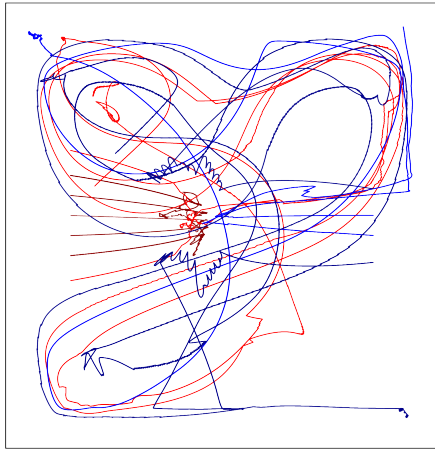


Fig. 6: Unit paths of an encounter between Terran with 5 Marines (red), 2 Marauders (dark red) and Protoss with 2 Zealots (blue), 2 Stalkers (dark blue). Winner: None (Time limit exceeded).

give eventual results more significance because the introduced changes could potentially lead to new imbalances. Although the presented approach was specifically designed to be applied to StarCraft II, which acts as a show case, its generality allows an easy adaption to similar games. As balancing in real time strategy games is generally hard to achieve, including time consuming testing phases involving human test players, its application can possibly shorten the game development cycle, allowing earlier releases. Moreover, because of the growing popularity of the eSports scene, to ensure equal winning chances for all participants of a competition, detecting and eliminating imbalances in games is of increasing importance.

## VI. REFERENCES

- [1] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, “A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft,” *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 5, no. 4, pp. 293–311, 2013.
- [2] B. Gmeiner, G. Donnert, and H. Köstler, “Optimizing Opening Strategies in a Real-time Strategy Game by a Multi-objective Genetic Algorithm,” in *Research and Development in Intelligent Systems XXIX*, pp. 361–374, Springer, 2012.
- [3] H. Köstler and B. Gmeiner, “A Multi-Objective Genetic Algorithm for Build Order Optimization in StarCraft II,” *KI-Künstliche Intelligenz*, vol. 27, no. 3, pp. 221–233, 2013.
- [4] J. Hagelbäck and S. J. Johansson, “Using multi-agent potential fields in real-time strategy games,” in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pp. 631–638, International Foundation for Autonomous Agents and Multiagent Systems, 2008.
- [5] G. Synnaeve and P. Bessiere, “A Bayesian model for RTS units control applied to StarCraft,” in *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pp. 190–196, IEEE, 2011.
- [6] T. W. Sandberg and J. Togelius, *Evolutionary Multi-Agent potential field based AI approach for SSC scenarios in RTS games*. PhD thesis, 2011.
- [7] J. Hagelbäck, “Potential-field based navigation in starcraft,” in *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pp. 388–393, IEEE, 2012.
- [8] E. A. Rathe and J. B. Svendsen, “Micromanagement in Starcraft using potential fields tuned with a multi-objective genetic algorithm,” 2012.
- [9] O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” *The international journal of robotics research*, vol. 5, no. 1, pp. 90–98, 1986.
- [10] J. Barraquand, B. Langlois, and J.-C. Latombe, “Numerical potential field techniques for robot path planning,” *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 22, no. 2, pp. 224–241, 1992.
- [11] J. Schmitt, S. Seufert, C. Zoubek, and H. Köstler, “Potential Field Based Unit Behavior Optimization for Balancing in StarCraft II,” in *GECCO Companion '15: Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference*, (New York, NY, USA), pp. 1481–1482, ACM, 2015.
- [12] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 2, pp. 182–197, 2002.
- [13] K. A. De Jong and W. M. Spears, “A formal analysis of the role of multi-point crossover in genetic algorithms,” *Annals of mathematics and Artificial intelligence*, vol. 5, no. 1, pp. 1–26, 1992.
- [14] G. Syswerda, “Uniform Crossover in Genetic Algorithms,” in *Proceedings of the 3rd International Conference on Genetic Algorithms*, (San Francisco, CA, USA), pp. 2–9, Morgan Kaufmann Publishers Inc., 1989.
- [15] H. Mühlenbein and D. Schlierkamp-Voosen, “Predictive models for the breeder genetic algorithm i. continuous parameter optimization,” *Evolutionary computation*, vol. 1, no. 1, pp. 25–49, 1993.
- [16] K. Deb and R. B. Agrawal, “Simulated binary crossover for continuous search space,” *Complex Systems*, vol. 9, no. 3, pp. 1–15, 1994.
- [17] K. Deb, K. Sindhya, and T. Okabe, “Self-adaptive simulated binary crossover for real-parameter optimization,” in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2007)*, UCL London, pp. 1187–1194, Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2007), UCL London, 2007.
- [18] K. Deb and D. Deb, “Analysing mutation schemes for real-parameter genetic algorithms,” *International Journal of Artificial Intelligence and Soft Computing*, vol. 4, no. 1, pp. 1–28, 2014.

# Personalised Track Design in Car Racing Games

Theodosios Georgiou

Personal Robotics Laboratory

Department of Electrical and Electronic Engineering

Imperial College London

Email: theodosios.georgiou08@imperial.ac.uk

Yiannis Demiris

Personal Robotics Laboratory

Department of Electrical and Electronic Engineering

Imperial College London

Email: y.demiris@imperial.ac.uk

**Abstract**—Real-time adaptation of computer games’ content to the users’ skills and abilities can enhance the player’s engagement and immersion. Understanding of the user’s potential while playing is of high importance in order to allow the successful procedural generation of user-tailored content. We investigate how player models can be created in car racing games. Our user model uses a combination of data from unobtrusive sensors, while the user is playing a car racing simulator. It extracts features through machine learning techniques, which are then used to comprehend the user’s gameplay, by utilising the educational theoretical frameworks of the Concept of Flow and Zone of Proximal Development. The end result is to provide at a next stage a new track that fits to the user needs, which aids both the training of the driver and their engagement in the game. In order to validate that the system is designing personalised tracks, we associated the average performance from 41 users that played the game, with the difficulty factor of the generated track. In addition, the variation in paths of the implemented tracks between users provides a good indicator for the suitability of the system.

## I. INTRODUCTION

“Future games are expected to have less manual and more user-generated or procedurally-generated content” [2]. This notion has the opportunity to elicit personalised and novel game content that can provide everlasting levels. In order to personalise you must first understand the player’s skills and abilities in a particular game. This involves knowing the game *Mechanics* – the components and rules of the game – that give rise to game *Dynamics* – how mechanics behave on user inputs – and fuses to game *Aesthetics* – user experiences invoked by the game (MDA framework) [3].

The main characteristics of an enjoyable game have been decoded by Malone [4] to be: challenge, fantasy and curiosity. In our research we are targeting both the training and engagement of the user in car racing games by stimulating the factors of game enjoyment through the user-adaptive procedural-generation of a track’s path. Pushing the car to the limits and handling tight turns at high speeds is what engages the users in that category of games. However, according to Steels [5], frustration and anger intensifies if the skills of the user are not sufficient enough to handle the difficulty of a given track.

We are proposing a framework where a combination of raw data from the game and sensors provided (e.g. eye tracking and head pose) are used to extract relevant features through machine learning techniques to implement a user model that is able to explain the current user’s performance during

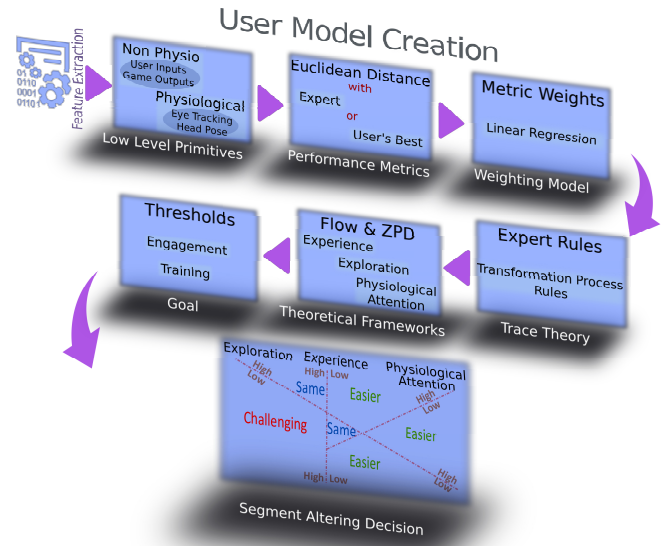


Fig. 1. Personalised user modelling approach for evaluating a driver in car racing game. Low level inputs are being converted to performance metrics where each metric defines the level of expertise of the user at a particular domain. For each domain a significance weight is assigned according to the individual and the task performed. Then game related rules transform metric-weight pairs to notions from the Concept of Flow [1] which are in turn exploited to evaluate the performance of the user and provide instructions on how the track’s path should be altered.

gameplay. Performance in games is a continuous function of the user’s skill and challenges, as well as the attention of the player. The aim of the framework is to: (a) monitor these properties, (b) update the user model, (c) provide decision adjustments for the alteration of the racing track according to the user, and (d) generate new tracks that suit the user profile. We propose a technique that modifies the game experience in real-time with the purpose of keeping the player’s satisfaction high and enhancing the learning process.

## II. RELATED WORK

User-oriented track generation has been approached before by Togelius et al. [6], [7], [8]. Their evolutionary algorithm (*Cascading Elitism*) generated a number of different tracks either by changing the control points of a basic track segment or by constraining their angular position. Then a



neural-network based controller [7], that was trained on human driver behaviour, was testing if a generated track is challenging enough for a particular driver. Fitness metrics (e.g. varying challenge, fast driving regions) were used to evaluate the suitability of a new track for the controller. However, the research was focused on the methodology and creativity of the generated tracks instead of their evaluation with human drivers.

Taking the research to the next step, Loiacono et al. [9] derived an algorithm for generating new tracks in a car simulator (TORCS) using single and multi-objective genetic algorithms. By maximising the entropy of certain criteria (e.g. path curvature distributions along the track, achievable speeds distributions) and under the condition that the track has to be closed, their algorithm fills the path through particular “control” points that the road needs to pass through. Their initial aim was to provide tracks with an adequate amount of challenge and a large degree of diversity across their path. A further improvement, for embedding a human oriented decision to the algorithm, was proposed by Cardamone et al. [10] where the framework for advancing the algorithm to a next generation of tracks was also influenced by human assistance. Subjects voted for each generated track using scoring interfaces (5 Likert scale or boolean type) that were influencing the algorithm over the next generations of tracks. They showed that there was an improvement of user satisfaction in early generations. However, when the evolved tracks were tested by human subjects, they concluded that the tracks were only appealing to the players with some experience in racing games.

Apart for just being entertaining and stimulating, games can be used for training and educating as well. Those games are referred in literature as “Serious Games” [11]; a medium to enhance education in a more entertaining way. Based on the notion of teaching through games, Backlund et al. [12] conducted research on driver behaviour between racing, action, sport (RAS) gamers and non-gamers. People were categorised into two groups through a questionnaire, whereas their driving skills (attention, decision making, risk assessment, etc.) and attitude (respect speed limits, speed margins and fellow drivers) were rated by driving school instructors, using 7-point Likert scale. Their findings show a positive correlation between gaming and skill oriented aspects of driving. An important concluding statement suggested that games, and more specifically driving simulators, are able to provide positive effects on driving behaviour and user skill enhancement thus motivating further research.

### III. SYSTEM OVERVIEW

#### A. The Simulator

In order to test the functionality of our user model and the performance of the track design algorithm, actual game-play data were collected from 41 users through high-end commercial racing car simulator game, called *rFactor 2*. The game provides an Application Program Interface (API) that allows the output of real-time data from the game environment

as well as user inputs. It also allows the introduction of new tracks into the game. Our custom made car simulator (see Fig. 2) is equipped with Vision Racer VR3 seat, Logitech G27 Force Feedback Steering Wheel and a combination of three monitors to enhance the user’s immersion to the game. Also, a Tobii EyeX eye tracker is installed in order to monitor the gaze of the user and 2 RGB-D cameras (Kinect) are used for capturing the player’s face, head pose, actions and the output from the monitors. Each user was asked to complete 20 laps on a particular track that we developed and answer a few questions before and after playing the game in order to assess their performance through the model.

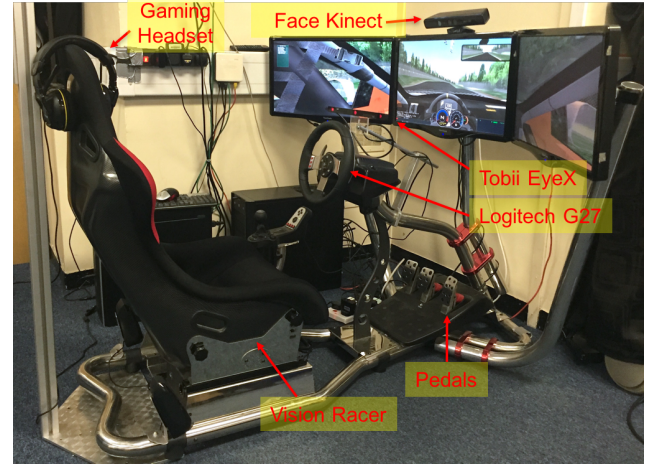


Fig. 2. Our custom car simulator setup consisting of Vision Racer Seat with a force feedback steering wheel, pedals and three monitors to enhance the user’s immersion to the game. Two RGB-D cameras as well as an eye tracker was installed to capture user’s gaze and pose positions.

#### B. The User Model

The aim of our user model is to capture the user’s current state through the available data and provide decisions for the future path of the track that suits the particular profile of the player. The user model structure, shown in Fig. 1, performs a series of analysis on the incoming user data to reach the decision state.

1) *Feature Extraction*: Features from user inputs, game outputs, eye tracker and head pose are extracted from the raw data provided through the game API and sensors and are grouped according to the location of the user in predefined paths of the selected track (*segments*). As seen in Fig. 3, segments are defined in such a way that they will consist only of a single path type (e.g. straight, turn, chicane, etc.). The selected track is short ( $\approx 1.93\text{Km}$ ) but still challenging track with various types of segments.

Table I lists the features collected, categorised by their sources. It is important to mention that head pose is established by passing the depth data from the RGB-D sensor, facing the user, through pre-trained Random Forests (RFs), that estimate the 3D coordinates of the nose tip and the angles of the head’s rotation [13]. Also, the clustering process to obtain the center

TABLE I  
FEATURES OF THE LOWEST LEVEL OF THE USER MODEL

User Inputs	Game Outputs	Eye Tracker	Head Pose
1. Average Braking	6. No. of collisions	10. No. of Blinks/s	17. CN of HP
2. Average Throttle	7. Car XYZ Position	11. Screen Concentration	18. CC of HP
3. Average Steering	8. Position and Speed	12. CN of EG	19. CN of HP and VO
4. CN of User Inputs	9. (Segment Time)	13. CC of EG	20. CC of HP and VO
5. CC of User Inputs		14. CN of EG and VO	
		15. CC of EG and VO	
		16. Eye Fixations	

— CN: Cluster Number, CC: Cluster Centres, VO: Virtual Orientation, EG: Eye Gaze, HP: Head Pose

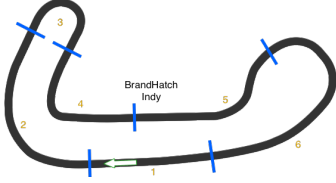


Fig. 3. Track used for the experiments split manually into segments

locations and cluster number of eye gaze and head poses features, is performed using the Affinity Propagation (AP) algorithm [14]. Features with Virtual Orientation (VO) are those whose raw data are paired with the respective sequences of virtual world angles of the car in the game.

2) *Performance Metrics*: Every time a user completes a segment, the features (from Table I) are calculated and compared for proximity to user's previous "player\_best" features to form the Performance Metrics. The "player\_best" are determined as these with the fastest segment time feature. The "player\_best" set is re-initialised every time the user does a better segment time and all of the previous metrics of that segment are re-computed. In order to reduce the outliers, each of the metrics, in the feature vector is converted to a percentage value using the exponential function shown in (1). The  $X$  is the highest value a metric can obtain. It is determined when a "player\_best" feature is compared with itself to obtain a metric. The Constant,  $C$ , for each metric was determined empirically off-line through all the data collected so that their median value lies around 50%.

$$\%Metric = 100 * \exp\left(-\frac{(Metric - X)}{C}\right) \quad (1)$$

3) *Weighting Model*: Metrics are classified according to two groups: the *Physio* group, which includes metrics associated with user's physiological data, such as those obtained from eye tracker and head pose and the *Non-Physio* group, which consists of metrics obtained from user inputs and game outputs. The user model is based on how much each of the metric created is correlated with the performance of the user on a particular segment. Good performance means low segment time. As we will explain later, the user model is providing decisions according to three notions: the *Experience*, the *Challenge* and the *Attention* of the user to the game. For each of these notions there is a principle that uses a subset of

the available metrics. The subsets are: *All* – which contains all metrics – *Physio* ( $p$ ) and *Non-Physio* ( $np$ ). For each metric in each subset we determine a weight according to the proportionality shown in (2).

$$t_s^l \propto \sum_{k=1}^m z W_k^s M_k^{sl} \quad (2)$$

where:

- $t_s^l$  is the time performance metric of a segment  $s$  and user lap  $l$  (No. 9 in Table I)
- $z W_k^s$  is the weight for a particular segment  $s$  of a metric  $k$  in the subset  $z$ .
- $m$  is the number of metrics defined in each of their group: All, Non-Physio, Physio.
- $M_k^{sl}$  is the Performance Metric  $k$  of segment  $s$  and lap  $l$

In order to find the weights  $z W_k^s$  we are treating (2) as a linear regression (LR) model. This is solved using the least squares (LS) problem method by minimising the sum of the squared error ( $\min \|Ax - b\|_2^2$ ) where in our case  $A$  represents a matrix of our metrics observations and  $b$  is their corresponding time metrics. The algorithm is forced to provide either positive or zero valued coefficients which are then normalised to find the weights  $z W_k^s$ . The reasoning for the positive bounds is because metrics were designed to have positive correlations to time and also the conversion of the metrics to percentages was one sided with 100% being the user's "player\_best". Weights are re-evaluated every-time a new set of features are created for a particular segment.

4) *Transformation Rules*: Following the theoretical frameworks of behavioural analysis like the Concept of Flow [1], [15], [5], the Zone of Proximal Development [16] and the Trace-Based System theory [17], [18] we further analyse the subsets of metrics using game-specific assumptions into certain rules and principles in order to form three classes that make up the high level in the user model: Experience, Exploration and Physiological Attention.

a) *Experience (E)*: (or the skills of the user) is determined by the proximity of the user's metrics to the "player\_best" ones. Since the metrics are expressed as percentages, the higher the value the better the user is performing on that particular metric. The significance of this metric to the user's skill is determined by the weight calculated in the *All* subset. Also, skill cannot be determined by a single

group of values. It has to be an overall value of several trials. Therefore, it is calculated by the weighted sum of the mean of each performance metric over a certain number of laps for a particular segment, as shown by (3). In this paper we considered 10 laps since they were enough to show the skills and avoid the overtraining of the particular track.

$$Experience_n^{l,s} = \sum_{k=1}^m all W_k^s * \frac{\sum_{i=l+(1-n)}^l M_k^{i,s}}{n}, \quad (3)$$

$l > n, n > 0, s \in S$

b) *Exploration (C)*: In the racing game, exploration can be explained when the user tries different approaches and techniques to complete a segment. This can be either different racing lines, unusual eye fixations, new input averages, etc. Through the metrics we define the Experience for each segment as the weighted sum of the mean of the consecutive “jump” ( $all J_k^s$ ) of each of the performance metrics in the *All* subset. If the difference between two consecutive metrics passes a fixed percentage value of the current experience then the value is positive, otherwise it is negative. This is shown by (4).

$$Exploration_n^{l,s} = \sum_{k=1}^m all W_k^s * \left( (|M_k^{l,s} - M_k^{l-1,s}|) - (all J_k^s * \frac{\sum_{i=l+(1-n)}^l M_k^{i,s}}{n}) \right),$$

$1 > J_k > 0, l > n, n > 0, l > 1, s \in S$

(4)

c) *Physiological Attention (PA)*: is designed to keep record of the continuous attention of the user along consecutive segments. This is calculated by first evaluating the Experience (*E*) of the user only from non-physiological (*np* subset) data through consecutive segment data. If the value is above a threshold ( $^{np}T$ ) then the assumption is that the user’s attention is high, since (s)he performs well, and the value is kept. Otherwise, we calculate the Exploration (*C*) value only from the physiological data (*p* subset) and use that for physiological attention. This is expressed by (5).

$$I = f(l, s),$$

$$^{np}E_n^{l,s} = \sum_{k=1}^{^{np}m} ^{np}W_k^s * \frac{\sum_{i=I+(1-n)}^I M_k^{i,s}}{n},$$

$$^pC_n^{l,s} = \sum_{k=1}^p W_k^s * \left( (|M_k^{l,s} - M_k^{l-1,s}|) - (J_k^s * \frac{\sum_{i=I+(1-n)}^I M_k^{i,s}}{n}) \right),$$

$$Attention_n^{l,s} = \begin{cases} ^{np}E_n^{l,s} - ^{np}T & \text{if } ^{np}E_n^{l,s} \geq ^{np}T \\ ^pC_n^{l,s} & \text{if } ^{np}E_n^{l,s} < ^{np}T \end{cases}$$

$1 > J_k^s > 0, l > n, n > 0, l > 1, s \in S$

(5)

where: *I* is the index number representing segment *s* in lap *l*. Function *f* transforms *s* and *l* to *I* so that *I* – 1 defines

the previous consecutive segment of index *I*. *n* defines the number of laps from which the value is calculated.

5) *Concept of Flow and ZPD*: By defining these three notions we are able to follow the Theory of Flow [1] which is defined to be the feeling of being completely immersed and engaged in an activity and also experiencing high levels of enjoyment and fulfilment [15]. Steels [5] states that for a person to remain engaged with a task and in the “flow” there has to be balance between the level of challenge (Exploration) and user’s skills (Experience). However, when the Attention of the user is low these values are more sensitive to each other. By using thresholds, the user model provides instructions for each segment (keep same, easier, more challenging) to the track design algorithm (described in Section III-C) so as to provide new relevant segments customised to the user.

The main idea of the model is also to utilise the theoretical framework of Zone of Proximal Development (ZPD) [16] which defines the difference between what a person can achieve with (Potential Development (PD)) or without help (Actual Developmental Level (ADL)). Therefore, if the user is becoming skilled enough (high Experience) on particular segments, then the segment should become more challenging in order to increase the user’s PD and encourage more training. On the other hand, if (s)he cannot cope with the current path then it should become easier since the user’s ADL is not enough for the particular segment. The bottom part of Fig. 1 (Segment Altering Decision) summarises the output of the user model according to predefined thresholds for each notion.

### C. Track Design Algorithm

The proposed idea of this paper is to alter the segments of a track according to the model of the user. Therefore, we start from a track that already exists from which we evolve and generate new ones by changing its segments. For each segment of the track, as shown, via a single segment, in Fig. 4, we keep 3 kinds of data:

- 1) Detailed (*XYZ*) point representation of the center path of the segment.
- 2) The orthogonal distance of each center point to the edges of the road so that we know where the right and left sides are located.
- 3) The point representation of the optimal path on that segment paired with the car speed (if the data are available from an expert).

In order to be able to compare the paths created by the user to the optimal or center paths of the segment, we parameterise each path by fitting its points into an *n*<sup>th</sup> order Bezier Curve. Through experimentation we found that an 9<sup>th</sup> order Bezier is accurate and efficient enough to describe a segment path. A sequence of Bezier curves (b-splines) were also used by Togelius et al. [6], [7], [8] to represent the path of the track.

A Bezier curve equation is described by (6). The number of control points *P<sub>i</sub>* describing the curve depend on the order of the degree chosen and in our algorithm their locations are determined by finding the best fit through the data points using the least squares (LS) method. The objective of the LS method

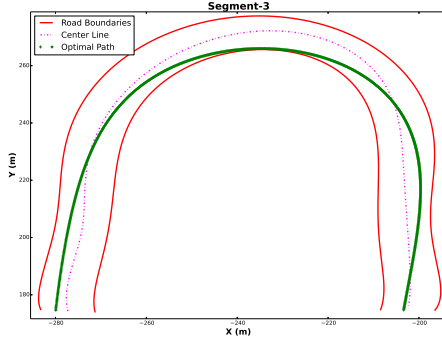


Fig. 4. Red lines define the boundary of the road. Dotted line is the center path and the green line defines the optimal path created by an expert.

is to minimise the sum of the squared errors (7). Each error (8) is defined as the difference between a path's data point ( $d_k$ ) on one of the axes and the value obtained by the best fit equation of the curve (6). Calculating the partial derivatives of (7) and setting the result to zero as shown by (9) we end up with  $n$  parameters ( $P_i$ ) and  $n$  gradient equations that can be arranged in a matrix form (10) and solve by using matrix inversion (11). By having the Bezier equation of a path enables us to extract a fixed number of  $XYZ$  points (by adjusting  $t_k$  divisions) and to compare it with other paths. In order to increase efficiency and also make sure that each equation starts and ends on the location we need them to,  $P_0$  and  $P_n$  are fixed to the starting point and the ending point of the relative segment.

$$B(t_k) = \sum_{i=0}^n \binom{n}{i} (1-t_k)^{n-i} t_k^i P_i, \quad t_k \in [0, 1] \quad (6)$$

$$M = \sum_{k=0}^n e_k^2 \quad (7)$$

$$e_k = d_k - B(t_k) \quad (8)$$

$$\frac{\partial M}{\partial P_i} = -2 \sum_{k=0}^n e_k \frac{\partial e_k}{\partial P_i} = 0 \quad (9)$$

$$AX = C \quad (10)$$

$$X = A^{-1}C, \quad X^T = (P_0 \quad \dots \quad P_n) \quad (11)$$

The track design algorithm keeps track of the user's paths along each segment in the form of a Bezier equation and stays on hold until a "segment change decision" is received from the model. On the instruction to keep the *Same* segment, then no action is performed. When an *Easier* segment is needed then a mean path from previous user paths is calculated. This mean path serves as the optimal path to the center of the new segment that is created. There is one assumption here that the user will find a path easier if the optimal route (s)he has to perform is already experienced in previous trials. An example of a user adapted easy segment is shown in Fig. 5.

If there is a need of a more *Challenging* segment then a different procedure is performed. Here we assume that a path can be converted to a more difficult one by either (a)

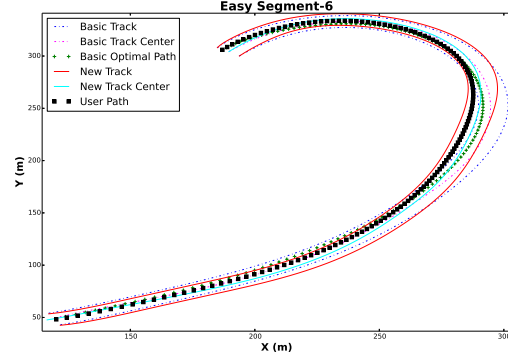


Fig. 5. The new center has been shifted in such a way that the average of the paths of the user is the optimal path of the new segment.

increasing the angle of its curve, (b) increasing the number of turns or (c) both. We are using a back-tracking algorithm to perform path planning between two points by setting a number of constraints. The algorithm proceeds from a starting point towards an end point by randomly checking for valid points along the circumference of a fixed radius. The constraints are set such as:

- There are no loops in the path, so that the path can be constructed in the game.
- The lines created by 3 points should not have an angle less than  $145^\circ$ , so that to keep the path smooth.
- A new point should be at a fixed distance ( $d$ ) from any orthogonal point on the current path, so that to avoid any intersections between other segments (the value  $d$  depends on the basic track selected).
- The new path should be at least 80% different from any of the previous paths of the user.
- The new path should start and end where the previous path was, in order to be able to plug and play the new created segment.

A challenging example is shown in Fig. 6. When a new path is created then the points are fit into a Bezier curve with the additional constraint that the end points gradient should stay the same as the path they are replacing. This refines the points and preserves the seamless link between the segments.

#### D. Graphics Generation

In order to be able to generate a track model that can be imported into the game, we need to create a track in a more specialised environment that is also supported by the game. We are using a computer graphics program called *Autodesk 3ds Max 2012* for which the particular game has a graphics model export plugin. The basic track was manually designed in the 3D software in a way that every object in the scene is virtually attached to the center path line of the track. The advantage of this method is that by altering the points of the center line to a new location then all the graphics objects are modified as well to adapt to the new change. Therefore, when creating a new segment we only need to alter the points specified in the



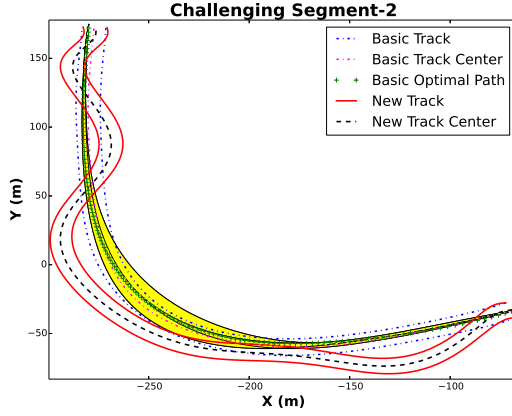


Fig. 6. Red boundaries show the path constructed by the backtracking algorithm. The previous segment path is displayed in dotted blue lines and the yellow shaded area defines the variance of the paths covered by the particular user in her previous attempts.

software for the particular segment. The basic track's graphic model and its rendering by the game used in our experiments are shown in Fig. 7.

One of the possible ways to describe a line between two points in the software is by using sequences of *cubic bezier splines*. This is using two points for defining the starting and ending location of the curve and two control points that determine the path curvature between the two points. Consecutive lines share location points (named as *knots*). Therefore, the track design algorithm has to convert the  $n^{\text{th}}$  Bezier spline segment of a new path into multiple cubic bezier splines under two conditions:

- It should have the same starting and ending point to the basic track segment, so that there is a smooth continuation between the segments.
- The total number of knots of each segment should be the same as the basic track's so that their indexing in the software is always the same. This restriction is not compulsory but if the number of knots change then the index configurations of all segments should update as well. Also, the knots are quite dense so there is no problem expanding the length of the path.

Evaluation of user model, decision taking and track design are all happening online. The track design algorithm is communicating with the graphics software through a custom TCP client-server program. The *client* is controlled by the track design algorithm and is responsible for sending (a) the appropriate segment identification and also (b) the software ready locations of knots and control points. The *server* is responsible for (a) communicating with the graphics software and (b) calling the software's scripts for altering the knots and their control points, (c) creating the graphics model and (d) sending the new track model files to the game.

## IV. RESULTS

The user model outputs three instructions on how each segment should change in the evolved track (easier, same, challenging). Our basic track consists of six segments from which five are supposed to change through the framework (start line – segment 1 – is omitted due to game related constraints). Therefore, this gives  $3^5 - 1 = 242$  new permutations, whereas each one is unique to the user as well as the randomness of the backtracking algorithm. In order to validate for the diversity and personalisation of our framework, we used the data collected from 41 users to run offline track generation at the point where the users performed 10 out of 20 laps. The generated tracks of 12 randomly selected users are shown in Fig. 9.

The users' profile was broad with ages between 19-35 and various expertise in racing games (31% non-gamers, 69% gamers) and driving (0–18 years). Because the purpose of the framework is set to provide training to the user, it is not supposed to deviate greatly from the basic track, since every segment starts and ends at the same location. From the shape of each track we can see that even with similar change instructions (defined by the change id: C - Challenging, S - Same, E - Easier) the segments are not the same, except if no change is performed. In order to test for user profile customisation we correlated the overall difficulty of the track generated to the average time taken for the user to complete the 10 laps. Difficulty was determined by assigning a score to the instructions (e.g.  $C = 2$ ,  $S = 1$ ,  $E = 0$ ) and summing them over all segments. In Fig. 9, difficulty is shown by the number in the brackets. Therefore, the basic track has a difficulty score of 5 since  $ChangeID = SSSSS$ , whereas the range varies between 0 – 10.

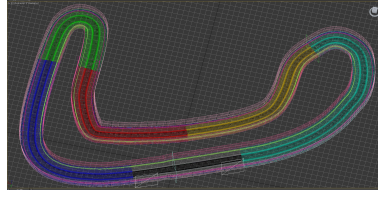
The Spearman correlation test between *average time* and *track difficulty* from all user data gave a statistically significant value of  $-0.82$  ( $p < 0.01$ ). This shows that there is a strong negative correlation between the average lap time and the track difficulty value generated by the model. It is important to mention that the user model is not directly using the time taken for the user to complete each segment in order calculate the performance of the user.

By assigning each user to a group using their average time over the 10 laps, according to Table II, we were able to utilise *Kruskal-Wallis* statistical test to investigate if the track difficulty assignment to the different groups is coming from the same distribution (random assignment). The test returned  $\chi^2 = 29.12$ ,  $df = 3$ ,  $p - value < 0.01$  which gives us enough evidence to reject the null hypothesis. The time thresholds (specified in Table II) for grouping the users were set according to the largest gaps that were formed between subjects' average time.

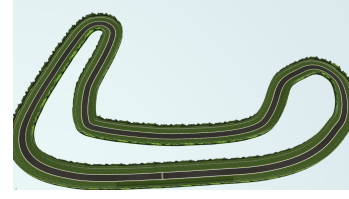
### A. Discussion

We can notice a clear distinction between Advanced-Intermediates groups and Average-Beginners groups (Fig. 8). The former typically gets difficulty values above 6 whereas the latter generates values around and below 6. Therefore, the





(a) Wireframe from Graphics software



(b) Game Rendered Track

Fig. 7. The basic track's path is similar to a real racing track called "BrandHatch Indy" which is located in London (UK). It is considered to be a short( $\approx 1.93\text{Km}$ ) but still challenging track with various types of segments.

general idea that more experienced users tend to get more challenging tracks than inexperienced ones is inferred by the framework.

The user model is designed to take account of the real-time attention of the user, in addition to the user's skills and challenges. Consequently, if the user is self-motivated to learn, boost their skills and achieve better lap times during longer training sessions ( $>10$  laps), the user model instructs most of the segments to stay the same and as a result track difficulty is closer to 5 (baseline). On the contrary, if the user lacks attention then, depending on their skill level, the model tries to accommodate either more challenging or easier segments. This increases the engagement of the user in the game and avoids the build up of boredom (when high experienced) or frustration (when low experienced). Every person is unique in their way of learning and training, this reason explains the variability between each group in Fig. 8.

TABLE II  
AVERAGE TIME VERSUS EXPERIENCE ASSIGNMENT FROM DATA TAKEN FROM THE BASIC TRACK.

Experience	Average Time(%)	Players
1. Advanced	$t \leq 60$	5
2. Intermediate	$60 < t \leq 70$	14
3. Average	$70 < t \leq 83$	15
4. Beginner	$t > 83$	7

The proposed framework relates to the general Experience-Driven Procedural Content Generation (EDPCG) framework proposed by [19]. The low level of the model follows a *hybrid* player experience modelling (PEM). It extracts information from unobtrusive *objective* PEM combined with *gameplay-based* inputs and outputs of the racing game. The extraction of data and weights follows a *model-free* approach whereas the higher levels follow a *model-based* using theoretical frameworks. Through this paper, the evaluation of the game content is performed using a *data-driven direct* functions, as the expertise of the user is correlated with the difficulty of the implemented track. In an ongoing future work our aim is to perform evaluation via *interactive implicit* and *explicit* functions by assessing the overall training of the user and by analysing the user feedback. Content representation, generation and optimisation is the main focus of this paper and according to the EDPCG framework it is a form of a

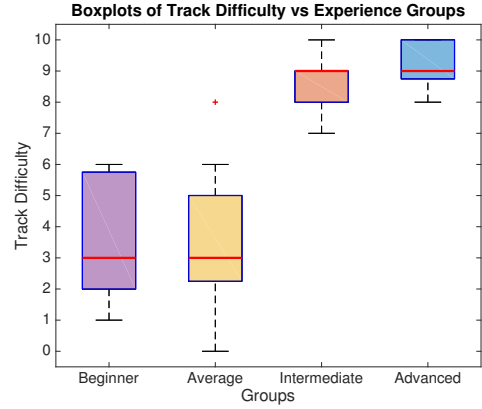


Fig. 8. Box plots of the track difficulty against user's experience group. *Kruskal-Wallis* test shows statistically significant results:  $\chi^2 = 29.12$ ,  $df = 3$ ,  $p\text{-value} < 0.01$  and Spearman test shows a strong negative correlation:  $-0.82$  ( $p < 0.01$ )

*search based approach.*

## V. CONCLUSION

We have implemented a framework for altering the path of the track online according to the skills, challenges and attention of the user, using physiological and non-physiological features through machine learning techniques, game-related rules and theoretical frameworks. To evolve the track we are employing the past history of the user, the decisions of the framework and the functionality of the Bezier curves with the aim to train and also keep the user engaged.

In our results we showed that the framework follows the general trend regarding the skills of the user and provides diversity between and among the subjects. To the extent of our knowledge, this is the first attempt to evolve the track of a 3D-game in situ, using real-time input from the user while is playing the game. User adaptive generation of tracks is a very interesting domain which can alter the way racing games are being designed and increase the market of consumers due to their flexible and unlimited content. In the future we would like to extensively test the evolved tracks with human subjects.

## ACKNOWLEDGMENT

The authors would like to thank Loizos Shianios for the help provided on designing the track in the specialised software.

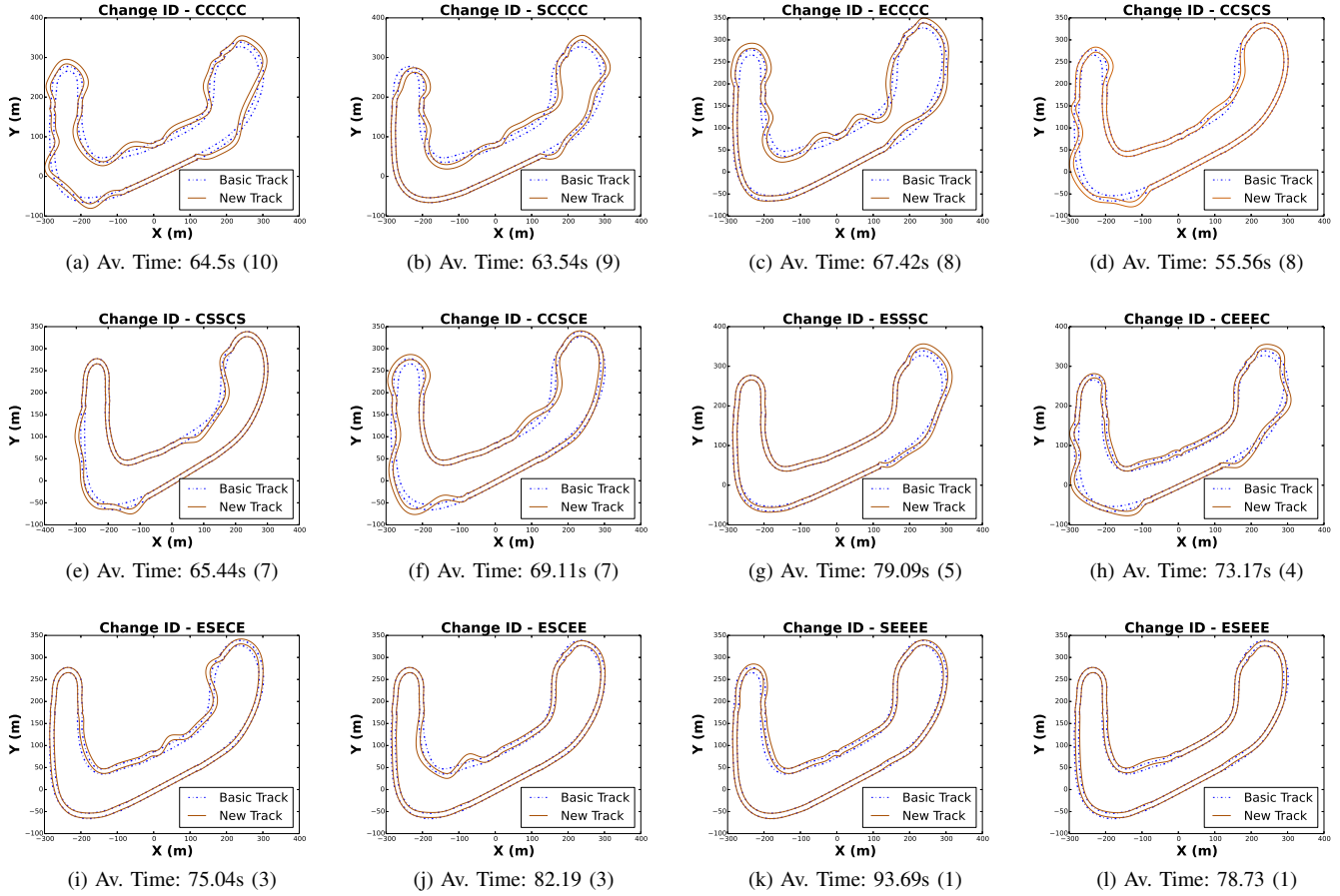


Fig. 9. Tracks generated through different users after the model collected data over 10 laps. The number in the brackets indicates the difficulty level of the new track according to the changes performed on the basic one, whereas the “Average Time” is an indication of the skill of the particular user (Difficulty range varies between 0 – 10)

## REFERENCES

- [1] J. Nakamura and M. Csikszentmihalyi, “The concept of flow,” in *Flow and the foundations of positive psychology*. Springer, 2014, pp. 239–263.
- [2] G. N. Yannakakis, P. Spronck, D. Loiacono, and E. André, “Player Modeling,” in *Artificial and Computational Intelligence in Games*. Schloss Dagstuhl Series, 2013, vol. 6, pp. 45–59.
- [3] R. Hunnicke, M. LeBlanc, and R. Zubeck, “MDA: A Formal Approach to Game Design and Game Research,” in *Proceedings of the AAAI Workshop on Challenges in Game AI*, vol. 4, 2004, pp. 1–4.
- [4] T. W. Malone, “What makes things fun to learn? Heuristics for designing instructional computer games,” in *Proceedings of the 3rd ACM SIGSMALL symposium*, 1980, pp. 162–169.
- [5] L. Steels, “The Architecture of Flow,” in *A Learning Zone of One’s Own*. IOS Press, 2004, pp. 135–150.
- [6] J. Togelius, R. De Nardi, and S. M. Lucas, “Making Racing Fun Through Player Modeling and Track Evolution,” in *Proceedings of the SAB Workshop on Adaptive Approaches to Optimizing Player Satisfaction*, 2006, pp. 61–70.
- [7] J. Togelius and S. M. Lucas, “Evolving robust and specialized car racing skills,” in *IEEE International Conference on Evolutionary Computation*, 2006, pp. 1187–1194.
- [8] J. Togelius, R. De Nardi, and S. M. Lucas, “Towards automatic personalised content creation for racing games,” in *IEEE Symposium on Computational Intelligence and Games*, 2007, pp. 252–259.
- [9] D. Loiacono, L. Cardamone, and P. L. Lanzi, “Automatic track generation for high-end racing games using evolutionary computation,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 245–259, 2011.
- [10] L. Cardamone, D. Loiacono, and P. L. Lanzi, “Interactive evolution for the procedural generation of tracks in a high-end racing game,” in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. ACM, 2011, pp. 395–402.
- [11] M. Zyda, “From visual simulation to virtual reality to games,” *IEEE Computer*, vol. 38, no. 9, pp. 25–32, 2005.
- [12] P. Backlund, H. Engström, and M. Johannesson, “Computer gaming and driving education,” in *Proceedings of the ICCE 2006 workshop Pedagogical Design of Educational Games*, 2006, pp. 9–16.
- [13] G. Fanelli, J. Gall, and L. Van Gool, “Real time head pose estimation with random regression forests,” in *Computer Vision and Pattern Recognition*, 2011, pp. 617–624.
- [14] B. J. Frey and D. Dueck, “Clustering by passing messages between data points,” *Science*, vol. 315, no. 5814, pp. 972–976, 2007.
- [15] M. Csikszentmihalyi, *Flow: The psychology of optimal experience*. New York: Harper and Row, 1990.
- [16] L. Vygotsky, “Interaction between learning and development,” *Readings on the development of children*, vol. 23, no. 3, pp. 34–41, 1978.
- [17] L. S. Settouti, Y. Prie, J.-C. Marty, and A. Mille, “A trace-based system for technology-enhanced learning systems personalisation,” in *Ninth IEEE International Conference on Advanced Learning Technologies*, 2009, pp. 93–97.
- [18] P. Bouvier, K. Sehaba, and E. Lavoué, “A trace-based approach to identifying users engagement and qualifying their engaged-behaviours in interactive systems: application to a social game,” *User Modeling and User-Adapted Interaction*, vol. 24, no. 5, pp. 413–451, 2014.
- [19] G. N. Yannakakis and J. Togelius, “Experience-driven procedural content generation,” *IEEE Transactions on Affective Computing*, vol. 2, no. 3, pp. 147–161, 2011.

# Discovering Playing Patterns: Time Series Clustering of Free-To-Play Game Data

Alain Saas, Anna Guitart and África Periañez  
Silicon Studio

1-21-3 Ebisu Shibuya-ku, Tokyo, Japan

Email: alain.saas@siliconstudio.co.jp, anna.guitart@siliconstudio.co.jp  
africa.perianez@siliconstudio.co.jp

**Abstract**—The classification of time series data is a challenge common to all data-driven fields. However, there is no agreement about which are the most efficient techniques to group unlabeled time-ordered data. This is because a successful classification of time series patterns depends on the goal and the domain of interest, i.e. it is application-dependent.

In this article, we study free-to-play game data. In this domain, clustering similar time series information is increasingly important due to the large amount of data collected by current mobile and web applications. We evaluate which methods cluster accurately time series of mobile games, focusing on player behavior data. We identify and validate several aspects of the clustering: the similarity measures and the representation techniques to reduce the high dimensionality of time series. As a robustness test, we compare various temporal datasets of player activity from two free-to-play video-games.

With these techniques we extract temporal patterns of player behavior relevant for the evaluation of game events and game-business diagnosis. Our experiments provide intuitive visualizations to validate the results of the clustering and to determine the optimal number of clusters. Additionally, we assess the common characteristics of the players belonging to the same group. This study allows us to improve the understanding of player dynamics and churn behavior.

## I. INTRODUCTION

In the past years, free-to-play (F2P) has emerged as the dominant monetization model of games on mobile platforms [1], [2]. F2P games are offered for free, and monetized by charging for in-game content through in-app purchases, with player retention being key to a successful monetization. The always-connected nature of mobile devices allows to constantly collect data about player behavior in the game. These data are used to guide design decisions for updates and release of additional content to maintain players' interest, sometimes in the form of periodic events giving access to new game content for a limited period of time [3].

This study is motivated by the idea that the automatic clustering of time series of player behavior can lead to a better understanding of player engagement. With daily active user bases ranging from thousands to millions of players, a game developer cannot know how every player reacts to a game or content update. At best, she can visualize averages of manually defined segments [4]. In this paper, we show that we can automatically cluster and visualize the main trends in player behavior and that we can determine differentiating characteristics of players belonging to different clusters. We

also consider the evolution of players after the end of the time series studied, and we investigate the use of this clustering as a feature addressing temporal dynamics for further supervised learning applications, e.g. a churn prediction model.

Previous efforts on clustering game data appear in [5], [6], [7], [8], [9], with the common goal of extracting player pattern behavior. However, the focus of these studies is non-time-oriented data. On the other hand, in the work presented by [10], a clustering of time series is performed, but the measurements are obtained from PC games, not from F2P game data which allow a robust behavioral analysis.

The aim of the present paper is to identify similar patterns in unlabeled temporal datasets of player activity in F2P games. In order to discover natural groups of players, based on their behavior and interaction with the game, we apply diverse clustering techniques which focus on maximizing the dissimilarity between different clusters and maximizing the homogeneity within the groups.

To the best of our knowledge this is the first article that applies unsupervised learning techniques to cluster time series of player behavior from F2P games. We have successfully extracted relevant user patterns from two F2P games: *Age of Ishtaria* and *Grand Sphere*, which helps us to examine quickly the player activity, allowing a visual game diagnosis and an intuitive evaluation of the weekly-based game events.

The games chosen for this study are representative of the most played F2P mobile social role-playing games in Japan and they have also been successful worldwide, reaching several millions of players.

## II. CLUSTERING TIME SERIES OF GAME DATA

Time series consist of sequential observations collected and ordered over time. Nowadays, almost every application, web or mobile based, produces a massive amount of time series data. The goal of unsupervised time series learning, e.g. clustering methods, is to discover hidden patterns in time ordered data.

Clustering time series data has received high attention over the last two decades [11], starting with the seminal work of [12] in 1993. It has faced many challenges [13], among which one of the most important is probably the high dimensionality level that time series contains and therefore the difficulty of defining a similarity measure, i.e. the *distance* between series, in order to classify close patterns in the same group. Working

with *raw* time series is computationally expensive and technically complicated. So as to cluster them efficiently, their complexity must be reduced through *representation* techniques [14], trying to maintain the characteristic features of the data. Both the dimensionality reduction and the similarity distance definition are obviously application-dependent.

Furthermore, with the fast increase of digital data, the clustering algorithms must be ready to deal with *Big Data* challenges [15], e.g. a vast volume of data to be processed with high efficiency and speed, sometimes even in real time. The literature on time series clustering is very extensive. For a comprehensive review about time series similarity search methods, check [16], [17], [18], [19].

In this Section we review the methods applied in the present paper to cluster the time series of player behavior. We briefly explain separately the representation methods and similarity measures used to evaluate the clustering results.

#### A. Similarity Measures

Given a time series, defined as a sequence such as

$$X_n = (x_1, x_2, \dots, x_N), \quad (1)$$

where  $x$  are the observations measured at different times  $n$ , we need to determine the level of *similarity/dissimilarity* (i.e. agreement/discrepancy) between a pair of them in order to cluster a sample of  $K$  time series.

Traditional distance computations, such as the Euclidean distance, can produce interesting results. However, in the case of *time series*, notions of distance need not to be confined to this simple geometric paradigm.

There are several ways to measure the dissimilarity between pairs of time series. This paper aims to cluster player profiles from *Age of Ishtaria* and *Grand Sphere* games based on their in-game behavior, hence dissimilarity measures were chosen according to this business target.

We are interested in the so-called *model-free* measures [20]. A naive *model-free* approach is to treat each series as an  $n$ -dimensional vector, and to calculate a shape-based geometric distance measure (without taking into account the absolute value of the time series selected). Such measures are the focus of our work, as we are interested in the *shape* pattern behavior (geometric comparison) rather than the magnitude of the time series.

Among the dissimilarity methods tested, those which provide the most robust results to classify time series are: Euclidean distance, Correlation (COR), Raw Values and Temporal Correlation (CORT) and Dynamic Time Warping (DTW). In addition, a *complexity-based* approach [20] called Complexity Invariant Distance (CID) [21] is applied. With this method, instead of focusing on the shape of the series, we expect to group profiles from a different perspective, taking into account the degree of variability over time.

Some other measures were also evaluated, e.g. autocorrelation-based dissimilarity, Frechet distance measure, periodogram-based dissimilarity, among others (a review about these techniques can be found in [20]). However, these

methods did not output as satisfactory results as the ones mentioned in the previous paragraph. The selected measures of interest are defined below, considering two time series  $X, Y$  of size  $T$  (which is the temporal dimension).

1) *Dynamic Time Warping (DTW)*: DTW is a *non-linear* similarity measure obtained by minimizing the distance between two time series [22]. This method permits to group together series that have similar shape but out of phase [23]. Figure 3 shows in the right upper panel how DTW aligns series with delayed but similar patterns.

DTW distance can be expressed as

$$DTW(X, Y) = \min_{r \in M} \left( \sum_{m=1}^M |x_{im} - y_{jm}| \right), \quad (2)$$

where the path element  $r = (i, j)$  represents the relationship between the two series. The goal is to minimize the time warping path  $r$  so that summing its  $M$  components gives the lowest measure of minimum cumulative distance between the time series. DTW searches for the best alignment between  $X$  and  $Y$ , computing the minimum distance between the points  $x_i$  and  $y_j$ .

2) *Correlation-based measure (COR)*: It performs dissimilarities based on the estimated Pearson's correlation of two given time series. The COR computation can be expressed as

$$COR(X, Y) = \frac{\sum_{n=1}^N (x_n - \bar{X})(y_n - \bar{Y})}{\sqrt{\sum_{n=1}^N (x_n - \bar{X})^2} \sqrt{\sum_{n=1}^N (y_n - \bar{Y})^2}}. \quad (3)$$

3) *Temporal Correlation and Raw Values Behaviors measure (CORT)*: It computes an adaptive index between two time series that covers both dissimilarity on raw values and dissimilarity on temporal correlation behaviors. It can be written as

$$CORT(X, Y) = \frac{\sum_{n=1}^{N-1} (x_{n+1} - x_n)(y_{n+1} - y_n)}{\sqrt{\sum_{n=1}^{N-1} (x_{n+1} - x_n)^2} \sqrt{\sum_{n=1}^{N-1} (y_{n+1} - y_n)^2}}. \quad (4)$$

4) *Complexity-Invariant Distance measure (CID)*: CID computes the similarity measure based on the Euclidean distance but corrected by the complexity estimation of the series [21]. CID is written as

$$CID(X, Y) = dist(X, Y) \cdot CF(X, Y), \quad (5)$$

with  $CF$  being the complexity correction factor defined by

$$CF(X, Y) = \frac{\max(CE(X), CE(Y))}{\min(CE(X), CE(Y))}. \quad (6)$$

And  $CE(\cdot)$  corresponds to the complexity estimations of a time series of length  $N$ , given by

$$CE(X) = \sqrt{\sum_{n=1}^{N-1} (x_n - x_{n+1})^2}. \quad (7)$$

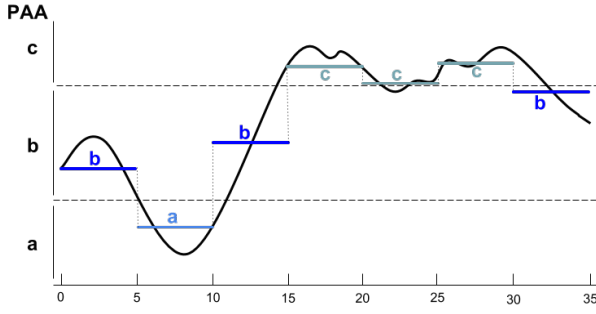


Fig. 1: Illustration of SAX representation method (dimensionality reduction of time series), performed with the following parameter values:  $w = 7$  and  $\alpha = 3$ .

### B. Representation Methods

Time series data collected from F2P games are high dimensional objects. In order to reduce their complexity and make the comparison feasible, it is convenient to perform a dimensional reduction transformation beforehand. There are several ways to reduce the data from  $n$ -dimensions to  $N$ -dimensions, and depending on the application domain, there are techniques more suitable than others. We focus on function approximation procedures to simplify the time series objects we aim to cluster. In the following subsections, we briefly summarize the most successful procedures for the video-game data tested in the present study.

1) *Discrete Wavelet Transform (DWT)*: This method uses a wavelet decomposition to approximate the actual series. A wavelet is a function used to approximate the target time series by means of superposition of several (wavelet) functions. A *wavelet* object provides information about variations of the time series locally, as it can be shown in Figure 3 in the right lower panel. DWT assigns a coefficient to each *wavelet* component and the distance is computed between the wavelet-approximated time series.

2) *Symbolic Aggregate Approximation related functions measure (SAX)*: SAX is a symbolic representation to simplify continuous time series [24]. The series is discretized and divided into sequential frames of equal size. Firstly, the series is divided in  $w$  set intervals and it is represented by its corresponding mean Piecewise Aggregate Approximation (PAA) dimensional reduction. Afterwards SAX is represented by a subset of alphabet letters of size  $\alpha$  where  $\alpha = (l_1, \dots, l_\alpha)$  and the transformed series  $\hat{X} = (\hat{X}_1, \dots, \hat{X}_\alpha)$  is computed by determining equal-sized zones under a Gaussian distribution. The distance is then computed between the approximated time series. Figure 1 depicts a schematic view of the SAX dimensionality reduction method.

3) *Trend Extraction*: A time series is composed of different elements such as seasonal components, medium or long-term trend, cyclical movement (repeated pattern but non-periodic) or irregular fluctuations (also known as residuals). Depending on the clustering application, some of these components can be representative of the characteristics of the *raw* time series.

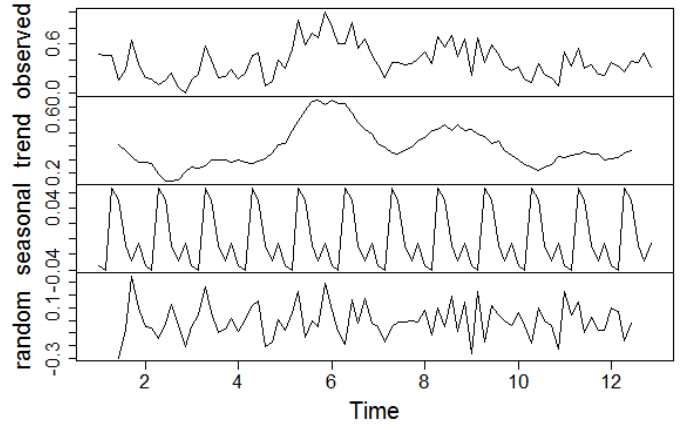


Fig. 2: Time Series Decomposition used for dimension reduction. Original series in the upper panel, trend extraction in the second plot, the seasonal and the random (residual fluctuations) components shown in the third and fourth panel, respectively.

As we are interested in characterizing players by their behavior, we focus on the trend component as it provides essential information for this purpose. Seasonality behavior or irregular data are not the center of our attention, the trend rather reflects significant information about player's interests. As it is mentioned in [25] "The trend of a time series is considered as a smooth additive component that contains information about global change". The method used in this work to extract the trend is the moving average filtering.

### C. Hierarchical clustering

In this subsection, we describe the methods to create the nested partitions in order to classify the total number of time series.

Hierarchical clustering creates homogeneous partitions of data according to their level of dissimilarity, maximizing the difference between clusters [26], [27]. The clustering growing method can be increasing (agglomerative clustering or bottom-up) or decreasing (divisive clustering or top-down) at each step. It is normally represented by dendrograms that show the clustering levels in a tree-based graph. Figure 4 illustrates with a dendrogram the hierarchical clustering performed to classify player behavior of *Age of Ishtaria*.

The method selected to cluster the datasets studied in the current paper is *agglomerative clustering*. There are different methods of agglomerative clustering [26], but the one used for our analysis is the so-called *Ward method* which is a minimum variance technique. In Ward, the distance between two clusters is defined as the deviance between them. The clusters that are merged in the same group are the ones that lead to a minimum increase in the total within-cluster variance (calculated from the dissimilarity measure selected between the time series) [27]. This method is used to obtain the results presented in the Section V, as our goal is to obtain a low variance within the clusters.



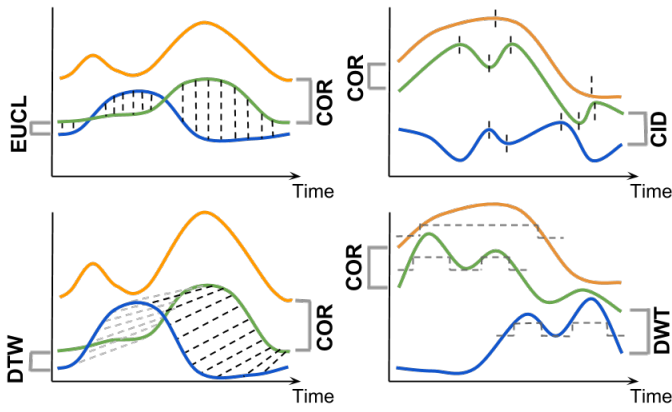


Fig. 3: Illustrations of the difference between time series clustering results obtained using Euclidean (EUCL) and Correlation dissimilarity measure (COR) (left upper panel), Dynamic Time Warping (DTW) in the upper right panel, Complexity Invariant Distance (CID) in the left lower panel, and Discrete Wavelet Transform (DWT) in the right lower panel.

### III. COMPARISON OF CLUSTERING METHODS

The selection of an adequate technique to cluster time series depends on the application and business interest. All the methods reviewed in Section II were tested to cluster the time series of game data. We want to classify players by pattern-shape, without giving too much attention to small fluctuations or to the total magnitude of the time series. Based on this aim we can conclude:

- DTW works particularly well to group similar player profiles with a shift on the time axis. DTW also groups together similar patterns but at different scale. However, as we are interested in evaluating the impact of game events on player activity, we rather focus on clustering synchronized profiles. Therefore, this is not the most suitable tool to measure the distance between time series for the purpose of the current study.
- In the DWT method of dimensionality reduction, the wavelets define the frequency of the series, which sometimes does not fit with the weekly seasonality we want to study.
- SAX representation method could be a useful tool for our problem as we are interested in identifying the pattern behavior, and not detailed aspects of the time series. However, the manual tuning of the two parameters  $w$  and  $a$  can be a drawback, although it is easily done once we introduce apriori information about the seasonality of the time series (the weekly events in our case). We hoped that the dimensionality reduction offered by SAX would allow us to cluster longer time series (2 months or more) but we did not obtain conclusive results.
- COR is a promising method for our goal. It groups similar geometric and synchronous profiles. As a drawback, COR seems to be sensitive to noise data and outliers (which are present in our datasets).

- CORT is similar to COR, but we ultimately obtained the most convincing results with the second.
- COR+trend is the combination of COR and trend extraction, which addresses COR's sensitivity to noise. This method allows us to obtain the best results for non-sparse time series (such as the time series of time played). Indeed, the trend extraction does not work well with time series containing many zero values (such as the time series of in-app purchases).
- CID groups series that have similar complexity patterns. This method performs poorly in classifying similar geometric profiles, which is what we do successfully with COR+trend applied to the time series of time played. However, it is the method that provides the best results when it comes to classifying time series containing large amount of zero values (such as the time series of purchases).

Figure 3 shows an intuitive comparison between similarity measures and representation methods to help to understand the difference between different techniques. The similarity methods and representation techniques described in this paper were tested to obtain the results presented in Section V.

#### A. Evaluation Metrics of Clustering results

The validation of the clustering methods is a challenging task, as we do not have any *truth* we can rely on to compare the accuracy of the classification, contrary to the supervised learning models. Several techniques to evaluate the adequacy of the similarity measures and representation methods to cluster time series were tested, among them: Dunn and average of Silhouette width [28], Normalized Hubert's statistic [29] and Entropy [30]. However, due to the difficulty of the task and the high complexity of time series objects, the results were not satisfactory. We used several kinds of visualization techniques to validate the clustering results and to determine the optimal number of clusters.

### IV. DATASETS

#### A. Data Source

Our data come from the games *Age of Ishtaria* and *Grand Sphere* by Silicon Studio.

We worked with time series of the following variables that are game independent and can be measured in all free-to-play games. These variables are measured per user and per day.

- *Time*: The amount of time spent in the game
- *Sessions*: The total number of playing sessions
- *Actions*: The total number of actions performed
- *Purchase*: The total amount of in-app purchases

Time, sessions and actions time series are highly correlated and produce very similar results. Thus, for the purpose of our study, we focus on the Time variable, which has a lower measurement error.

Purchases time series are different from the others because they are sparse (they contain many zero values), as the majority of the paying users do not complete an in-app purchase every day.

TABLE I: Summary description of the clustering results with *Age of Ishtaria* and *Grand Sphere* data.

Clustering result name	Game	Data	Technique	Clusters	Start date	End date
Age of Ishtaria time clustering	Age of Ishtaria	Time played per day	COR+trend	8	11-Jun-2015	1-Jul-2015
Age of Ishtaria spending clustering	Age of Ishtaria	Spending per day	CID	5	30-Jan-2015	19-Feb-2015
Grand Sphere time clustering	Grand Sphere	Time played per day	COR+trend	8	11-Sept-2015	1-Oct-2015

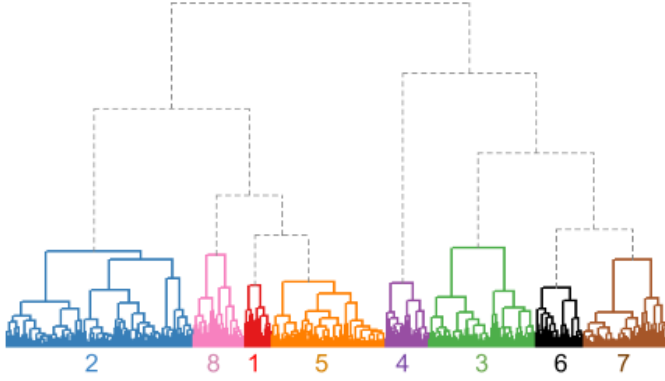


Fig. 4: Hierarchical Clustering represented by a dendrogram of *Age of Ishtaria* time-played data. The Age of Ishtaria time clustering is performed with COR similarity measure and trend extraction as representation method.

#### B. Time Series Studied

The frequency of our time series is daily. We study them on a weekly basis since there are weekly game events influencing player behavior. The studied period  $P$ , therefore contains  $N_{week} \times 7$  values, with  $N_{week} = 3$  being the number of weeks selected for this study. We synchronize the starting date  $P_{start}$  and the ending date  $P_{end}$  of our time series with the starting date of the game events.

In order to avoid a bias due to the partial absence of data from players who join or leave the game during  $P$ , we only consider data from players who installed the game before  $P_{start}$  and who are still active after  $P_{end}$ .

For the final results of clustering the time series of Time played, we consider only data from the users who played at least 6 days per week. There are two reasons for this choice. Firstly, from a free-to-play game developer perspective, we are interested in the most active players. Secondly, the clustering technique that allowed us to obtain the best results for this clustering performs poorly with sparse time series.

For the final results of clustering the times series of Purchases, which are mostly sparse, except for the very top spenders, we considered the players who did at least one purchase during  $P$ . Due to the sparse nature of these time series, we then obtain the best results using a different clustering technique.

Finally, we take random samples of 1000 time series respecting the conditions above for our experiments.

#### V. RESULTS

In this section, we present the results of the clustering experiments summarized in Table I. For each experiment, we

call event A, B and C the game events released respectively on week 1, 2 and 3. This is a naming convention independent of the content of the events.

#### A. Clustering and Visualization

1) *Clustering time series of time played*: Figure 5 visualizes the classification obtained by clustering time series of time played per day by Age of Ishtaria players, using the correlation dissimilarity measure on the trend extracted from the raw time series. We call this method *COR+trend*.

For each cluster, we plot the mean of the time series and a heatmap containing all the time series (one time series for each player included in the sample).

Visualizing the mean allows to see the top trends in player behavior. For example, the activity of class 3 plunged for event B but spiked for event C, while class 4 followed an opposite pattern. Since the game events are usually designed based on predefined game templates (i.e. they are reused throughout the lifetime of the game), this analysis helps the game designer to better understand the interest of several groups of players in different kinds of game events. This supports future game event planning and improves the knowledge about the impact of the game events on the player activity.

Visualizing the time series of each cluster on a heatmap allows to quickly validate the quality of the clustering. Figure 5 shows that the time series follow the same patterns within each cluster.

Heatmap along with a dendrogram visualization, represented in Figures 4 and 5, proves to be a better tool than the statistical measures tested, mentioned in Section III-A, for choosing the optimal number of clusters. Using these tools we determine that the most optimal clustering is obtained with 8 clusters.

We apply the same clustering technique on time series of time played by the players of Grand Sphere, and obtain similar results, as it can be checked in Figure 6. This is a promising fact towards obtaining an adequate technique ready to cluster data from other F2P games.

2) *Clustering time series of purchases*: Figure 7 depicts the clusters obtained by clustering time series of purchases per day by Age of Ishtaria players, using the Complexity Invariant Distance (CID) on the raw time series.

For each cluster, we represent the distribution of the data separated per week in a box-plot, and a heatmap containing all the time series.

Visualizing the time series of each cluster on a heatmap allows to distinguish different purchase patterns. For example, players from class 1 and class 3 purchase sparsely while players from class 2 purchase nearly every day.

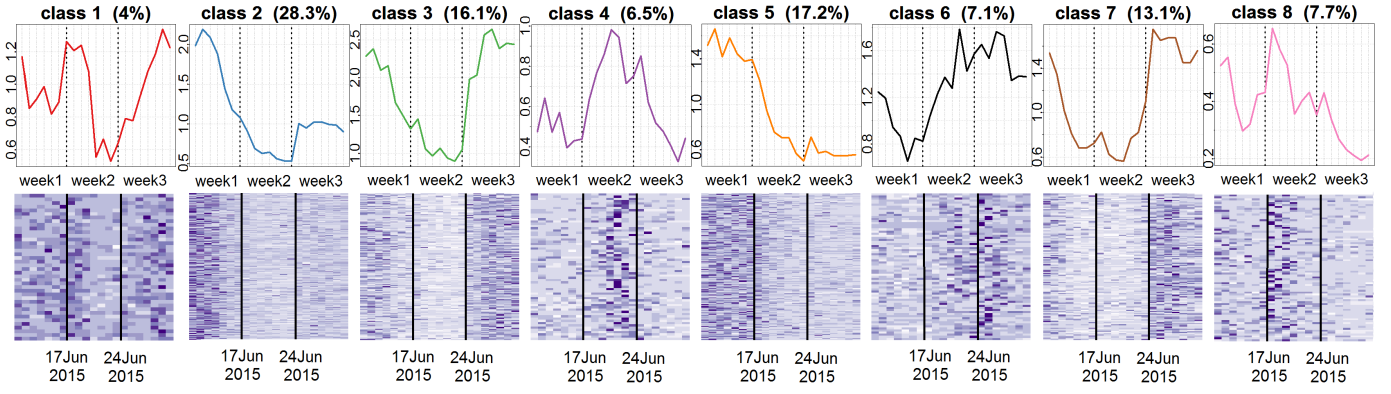


Fig. 5: Mean of the time series and heatmap for each cluster from Age of Ishtaria time clustering (time played per day). Vertical lines delimiting the game events. Clustering performed with COR similarity measure and trend extraction.

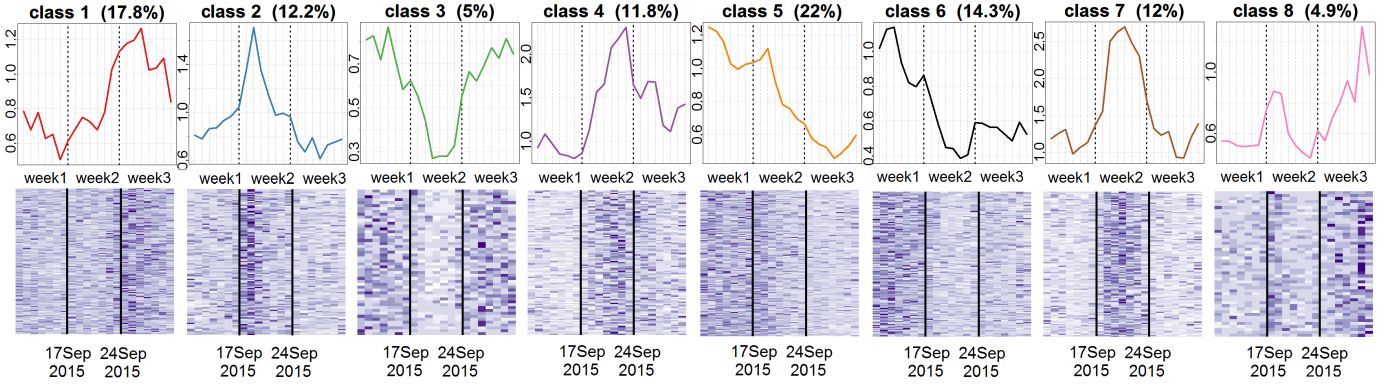


Fig. 6: Mean of the time series and heatmap for each cluster from Grand Sphere time clustering (time played per day). Vertical lines delimiting the game events. Clustering performed with COR similarity measure and trend extraction.

TABLE II: Characteristics of the players at the starting date of the studied period (Age of Ishtaria time clustering)

variables	class 1	class 2	class 3	class 4	class 5	class 6	class 7	class 8
number of players	40	283	161	65	172	71	131	77
ratio PU	30.0%	33.2%	44.7%	20.0%	33.1%	33.8%	35.1%	14.3%
average level	47	53	75	35	51	49	58	31

TABLE III: Cumulative churn ratio in the months following the clustering, after period  $P$  (Age of Ishtaria time clustering)

churners ratio	class 1	class 2	class 3	class 4	class 5	class 6	class 7	class 8
July	15.0%	11.7%	4.3%	15.4%	19.2%	18.3%	5.3%	22.1%
August	27.5%	19.8%	13.0%	26.2%	30.8%	31.0%	14.5%	28.6%
November	45.0%	48.4%	29.2%	47.7%	51.7%	20.7%	32.8%	58.4%

Since the scale of each heatmap is normalized separately to be able to visualize properly the full range of purchases on each heatmap, we can not compare the amount of the purchases between the clusters using only this visualization. And, contrary to the clustering of the time series of time played described above, the time series of purchases are mostly sparse, which makes it irrelevant to plot the mean of these time series. That is why we use the box-plot representation of the spending for each week, in order to visualize the difference of scale between the different groups. This additional plot allows us, for example, to see that class 5 contains very high spenders

even if they have a relatively sparse purchase behavior like class 1 and 3.

As for the time series of time played, we used the heatmap visualizations and the dendrogram to choose to use  $k = 5$  clusters.

### B. Extraction of Players Characteristics

We are not only interested in clustering game users and discovering hidden patterns, but also want to analyze the characteristics they have in common.



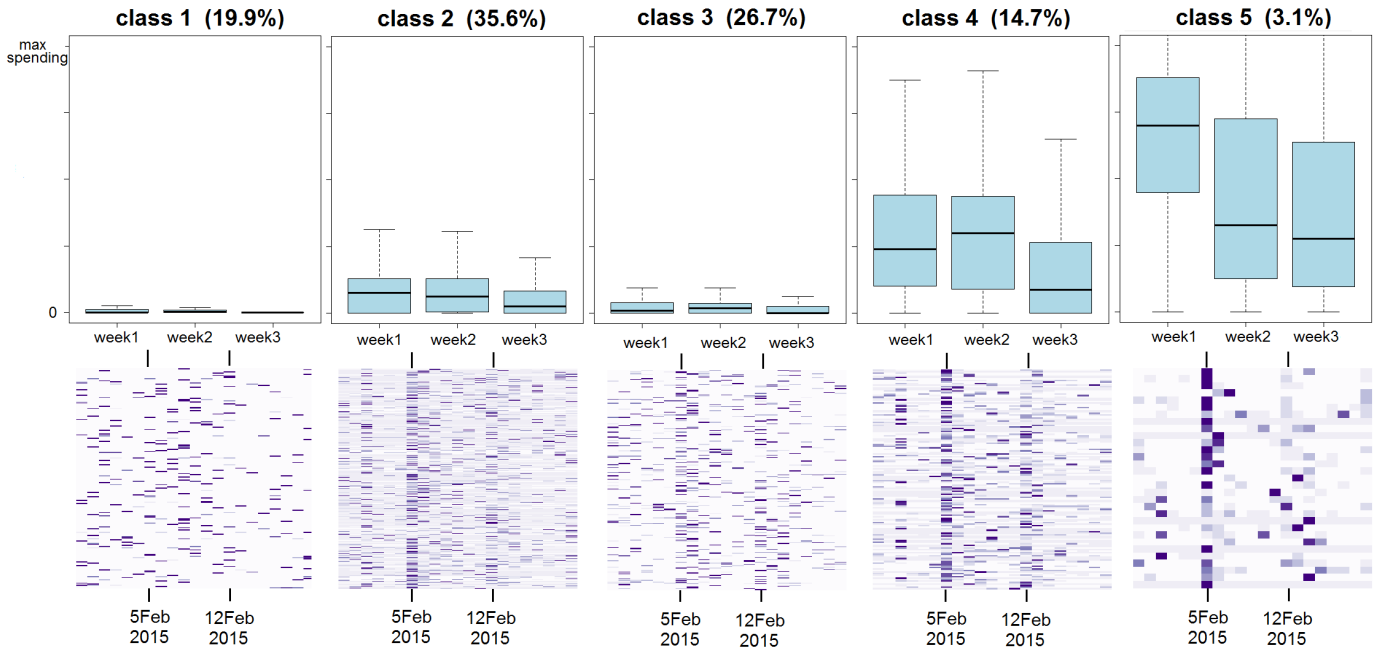


Fig. 7: Clustering results from the visualization of purchase's time series from Age of Ishtaria data using CID similarity measure (Age of Ishtaria spending clustering). Box plots of the spending per player and per week in the upper panel. Corresponding heatmap for each cluster below. The dates on the  $x$ -axis delimiting the weekly game events.

After performing the clustering, we measure how players behave during the period  $P$  by analyzing their characteristics on the start date  $P_{start}$  of the time series.

Table III reflects that class 3 contains players with the highest playing levels and also the highest ratio of paying users, while classes 4 and 8 contain the players with the lowest levels and the lowest ratios of paying users.

It is interesting to note that these clusters coincide with the ones already discussed earlier as reflecting an opposite interest in certain game events. With these two observations, we can conclude that event B was unpopular for advanced players and more popular for less advanced players, and that event C was more popular for advanced players and unpopular for less advanced players. A game planner visualizing this could conclude that she had better avoid triggering an event of event C's type soon after a user acquisition campaign, as it would likely be unpopular for the new coming less advanced players just acquired.

This example shows that it is possible to extract differentiating player characteristics from the clustering we obtained.

### C. Churn behavior

Player retention is of crucial importance in F2P games. Several models have been proposed to help to understand and predict the churn of players [31], [32], [33].

Based on the results obtained in Table II and Figure 5, we study the evolution of the players after the period of time  $P$  covered by the time series, in order to see if there is a relation between their behavior during  $P$  and after  $P$ .

Table II shows the churning rate 1, 2 and 5 months after the period  $P$  for each cluster. We observe that class 3 and 7 have

a significantly lower churning rate than class 4 and 8, being 3 to 4 times lower after 1 month and 1.5 to 2 times lower after 5 months.

According to this result, players have a different churn behavior following their profile classification performed during the period  $P$ .

Therefore, the use of the unsupervised classification of player profiles suggested in this article could be an interesting feature to address the temporal dynamics of players data for a churn supervised learning model. In [32] an alternative approach was proposed using a Hidden Markov Model.

However, in order to use this predictor in a supervised model some changes need to be performed in the definition of the problem as we discussed in Section IV. This comprehensive analysis is beyond the scope of this paper. For example, this would involve to cluster players based on their last weeks behavior, e.g. the time series starting date would be 3 weeks before the last day the players connected to the game instead of taking fixed dates as in the present work.

This time series classification would allow us to improve the understanding about the churn of players but, on the other hand, it would not provide information about game events reaction, which is a principal target of the current analysis.

## VI. SUMMARY AND CONCLUSION

In the present article, we have conducted a research about unsupervised clustering of time series data from two free-to-play games. We evaluate several similarity measures and representation methods to extract meaningful behavioral patterns of players. This allows us to assess the impact of

weekly game events and discover hidden playing dynamics regarding purchases and time played per day. An appropriate characterization of time series allows us to find significant attributes in common among players belonging to the same group. Ongoing and future work involve the application of the time series clustering results to churn prediction models and further analysis of the player profiles.

## VII. SOFTWARE

The analyses presented in Section V were performed with the R version 3.2.3 for Windows, using the following packages from CRAN: *TSclust* 1.2.3 [20], *timeSeries* 3022.101.2 [34], *fpc* 2.1-10 [35], *Rmisc* 1.5 [36], *reshape* 0.8.5 [37], *ggplot2* 2.0.0 [38].

## ACKNOWLEDGMENTS

We thank our colleagues Sovannrith Lay, Hiroshi Okuno, Takeshi Kimura, Tomomi Hamamura, Kotaro Narizawa and Yumi Kida for their help to collect the data and their support during this study. We also thank Thanh Tra Phan for the careful review of the article.

## REFERENCES

- [1] T. Fields, "Mobile & social game design: Monetization methods and mechanics," *CRC Press*, 2014.
- [2] A. Annie, "IDC. 2014. Mobile App Advertising and Monetization Trends 2012-2017: The Economics of Free."
- [3] W. Xing, "Unleash the Power of Events to Drive Revenue in Games," Mar. 2014. [Online]. Available: [http://www.gamasutra.com/blogs/XingWang/20140319/213429/Unleash\\_the\\_Power\\_of\\_Events\\_to\\_Drive\\_Revenue\\_in\\_Games.php](http://www.gamasutra.com/blogs/XingWang/20140319/213429/Unleash_the_Power_of_Events_to_Drive_Revenue_in_Games.php)
- [4] E. Fradley-Pereira, "A Beginners Guide to Player Segmentation," May 2015. [Online]. Available: <https://www.fusepowered.com/blog/2015/05/a-beginners-guide-to-player-segmentation/>
- [5] C. Bauckhage, A. Drachen, and R. Sifa, "Clustering game behavior data," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 7, no. 3, pp. 266–278, 2015.
- [6] A. Drachen, R. Sifa, C. Bauckhage, and C. Thureau, "Guns, swords and data: Clustering of player behavior in computer games in the wild," in *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*. IEEE, 2012, pp. 163–170.
- [7] A. Drachen, C. Thureau, R. Sifa, and C. Bauckhage, "A comparison of methods for player clustering via behavioral telemetry," *CoRR*, vol. abs/1407.3950, 2014. [Online]. Available: <http://arxiv.org/abs/1407.3950>
- [8] R. Sifa, C. Bauckhage, and A. Drachen, "The Playtime Principle: Large-scale cross-games interest modeling," in *2014 IEEE Conference on Computational Intelligence and Games, CIG 2014, Dortmund, Germany, August 26-29, 2014*, 2014, pp. 1–8. [Online]. Available: <http://dx.doi.org/10.1109/CIG.2014.6932906>
- [9] A. Drachen, R. Sifa, C. Bauckhage, and C. Thureau, "Guns, swords and data: Clustering of player behavior in computer games in the wild," in *CIG*. IEEE, 2012, pp. 163–170.
- [10] H. D. Menéndez, R. Vindel, and D. Camacho, "Combining time series and clustering to extract gamer profile evolution," in *Computational Collective Intelligence. Technologies and Applications*. Springer, 2014, pp. 262–271.
- [11] T. Fu, "A Review on Time Series Data Mining," *Eng. Appl. Artif. Intell.*, vol. 24, no. 1, pp. 164–181, Feb. 2011. [Online]. Available: <http://dx.doi.org/10.1016/j.engappai.2010.09.007>
- [12] R. Agrawal, C. Faloutsos, and A. N. Swami, "Efficient similarity search in sequence databases," in *Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms*, ser. FODO '93. London, UK, UK: Springer-Verlag, 1993, pp. 69–84. [Online]. Available: <http://dl.acm.org/citation.cfm?id=645415.652239>
- [13] P. Esling and C. Agon, "Time-series Data Mining," *ACM Comput. Surv.*, vol. 45, no. 1, pp. 12:1–12:34, Dec. 2012. [Online]. Available: <http://doi.acm.org/10.1145/2379776.2379788>
- [14] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh, "Querying and Mining of Time series Data: Experimental Comparison of Representations and Distance Measures," *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1542–1552, Aug. 2008. [Online]. Available: <http://dx.doi.org/10.14778/1454159.1454226>
- [15] X. Xi, E. Keogh, C. Shelton, L. Wei, and C. A. Ratanamahatana, "Fast Time Series Classification using Numerosity Reduction," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06. New York, NY, USA: ACM, 2006, pp. 1033–1040. [Online]. Available: <http://doi.acm.org/10.1145/1143844.1143974>
- [16] T. W. Liao, "Clustering of time series data - a survey," *Pattern recognition*, vol. 38, no. 11, pp. 1857–1874, 2005.
- [17] X. Wang, K. Smith, and R. Hyndman, "Characteristic-Based Clustering for Time Series Data," *Data Mining and Knowledge Discovery*, vol. 13, no. 3, pp. 335–364, 2006. [Online]. Available: <http://dx.doi.org/10.1007/s10618-005-0039-x>
- [18] E. Keogh and S. Kasetty, "On the need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration," *Data Min. Knowl. Discov.*, vol. 7, no. 4, pp. 349–371, Oct. 2003. [Online]. Available: <http://dx.doi.org/10.1023/A:1024988512476>
- [19] E. Keogh, "A Decade of Progress in Indexing and Mining Large Time Series Databases," in *Proceedings of the 32nd International Conference on Very Large Data Bases*, ser. VLDB '06. VLDB Endowment, 2006, pp. 1268–1268. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1182635.1164262>
- [20] P. Montero and J. A. Vilar, "TSclust: An R package for time series clustering," *Journal of Statistical Software*, vol. 62, no. 1, pp. 1–43, 2014. [Online]. Available: <http://www.jstatsoft.org/v62/i01/>
- [21] G. E. Batista, E. J. Keogh, O. M. Tataw, and V. M. de Souza, "CID: an efficient complexity-invariant distance for time series," *Data Mining and Knowledge Discovery*, vol. 28, no. 3, pp. 634–669, 2014.
- [22] D. J. Bemdt and J. Clifford, "Using Dynamic Time Warping to Find Patterns in Time Series," 1994.
- [23] C. A. Ralanamahatana, J. Lin, D. Gunopulos, E. Keogh, M. Vlachos, and G. Das, "Mining time series data," in *Data Mining and Knowledge Discovery Handbook*. Springer, 2005, pp. 1069–1103.
- [24] J. Lin, E. Keogh, L. Wei, and S. Lonardi, "Experiencing SAX: a novel symbolic representation of time series," *Data Mining and knowledge discovery*, vol. 15, no. 2, pp. 107–144, 2007.
- [25] T. Alexandrov, S. Bianconcini, E. Dagum, P. Maass, and T. McElroy, "A review of some modern approaches to the problem of trend extraction," *Econometric Reviews*, vol. 31, no. 6, pp. 593–624, 2012.
- [26] F. Murtagh and P. Contreras, "Algorithms for hierarchical clustering: an overview," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 1, pp. 86–97, 2012.
- [27] F. Murtagh and P. Legendre, "Ward's Hierarchical Agglomerative Clustering Method: Which algorithms implement Ward's Criterion?" *Journal of Classification*, vol. 31, no. 3, pp. 274–295, 2014. [Online]. Available: <http://dx.doi.org/10.1007/s00357-014-9161-z>
- [28] B. Desgraupes, "Clustering Indices," *University of Paris Ouest-Lab ModalX*, vol. 1, p. 34, 2013.
- [29] M. Halkidi, Y. Batistakis, and M. Vazirgiannis, "On clustering validation techniques," *Journal of intelligent information systems*, vol. 17, no. 2, pp. 107–145, 2001.
- [30] M. Meilă, "Comparing clusterings - an information based distance," *Journal of multivariate analysis*, vol. 98, no. 5, pp. 873–895, 2007.
- [31] F. Hadji, R. Sifa, A. Drachen, C. Thureau, K. Kersting, and C. Bauckhage, "Predicting player churn in the wild," in *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*. IEEE, 2014, pp. 1–8.
- [32] J. Runge, P. Gao, F. Garcin, and B. Faltings, "Churn prediction for high-value players in casual social games," in *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*. IEEE, 2014, pp. 1–8.
- [33] A. Periañez, A. Saas, A. Guitart, and C. Magne, "Churn prediction in mobile social games: towards a complete assessment using survival ensembles," *Submitted to DSAA*, 2016.
- [34] D. Wuertz and Y. Chalabi, "timeSeries: Rmetrics-financial time series objects," *R package version 2100*, 2009.
- [35] "fpc: Flexible Procedures for Clustering."
- [36] R. Hope, "Rmisc: Ryan miscellaneous. R package version 1.5," 2013.
- [37] H. Wickham, "reshape: Flexibly reshape data, 2007," *R package version 0.8.0*.
- [38] H. Wickham and W. Chang, "ggplot: An implementation of the Grammar of Graphics," *R package version 2.0.0*, 2013.

# Monte Carlo Tree Search with Options for General Video Game Playing

Maarten de Waard  
University of Amsterdam  
Science Park 904  
Amsterdam, Netherlands  
Email: mrtndwrd@gmail.com

Diederik M. Roijers  
University of Oxford  
Wolfson Building, Parks Road  
Oxford, United Kingdom  
Email: Diederik.Roijers@cs.ox.ac.uk

Sander C.J. Bakkes  
Tilburg University  
Warandelaan 2, Dante Building  
Tilburg, Netherlands  
Email: S.C.J.Bakkes@uvt.nl

**Abstract**—General video game playing is a challenging research area in which the goal is to find one algorithm that can play many games successfully. “Monte Carlo Tree Search” (MCTS) is a popular algorithm that has often been used for this purpose. It incrementally builds a search tree based on observed states after applying actions. However, the MCTS algorithm always plans over actions and does not incorporate any higher level planning, as one would expect from a human player. Furthermore, although many games have similar game dynamics, often no prior knowledge is available to general video game playing algorithms. In this paper, we introduce a new algorithm called “Option Monte Carlo Tree Search” (O-MCTS). It offers general video game knowledge and high level planning in the form of “options”, which are action sequences aimed at achieving a specific subgoal. Additionally, we introduce “Option Learning MCTS” (OL-MCTS), which applies a progressive widening technique to the expected returns of options in order to focus exploration on fruitful parts of the search tree. Our new algorithms are compared to MCTS on a diverse set of twenty-eight games from the general video game AI competition. Our results indicate that by using MCTS’s efficient tree searching technique on options, O-MCTS outperforms MCTS on most of the games, especially those in which a certain subgoal has to be reached before the game can be won. Lastly, we show that OL-MCTS improves its performance on specific games by learning expected values for options and moving a bias to higher valued options.

## I. INTRODUCTION

Recent game programming research focusses on algorithms capable of solving several games with different types of objectives. A common approach is to use a tree search in order to select the best action for any given game state. In every new game state, the tree search is restarted until the game ends. A popular example is *Monte Carlo tree search* (MCTS).

A method to test the performance of a general video game playing algorithm is by using the framework of the *general video game AI* (GVGAI) competition [1]. In this competition, algorithm designers can test their algorithms on a set of diverse games. When submitted to the competition, the algorithms are applied to an unknown set of games in the same framework to test their general applicability. Many of the algorithms submitted to this contest rely on a tree search method.

A limitation in tree search algorithms is that since many games are too complex to plan far ahead in a limited time frame, many of these algorithms incorporate a maximum

search depth. As a result, tree search based methods often only consider short-term score differences and do not incorporate long-term plans. Moreover, many algorithms lack common video game knowledge and do not use any of the knowledge gained from the previous games.

In contrast, when humans play a game we expect them to make assumptions about its mechanics, e.g., pressing the left button often results in the player’s avatar moving to the left on the screen. Furthermore, we expect human players to define specific subgoals for themselves, e.g., when there is a portal on screen, a player is likely to try to find out what the portal does by walking towards it. The player will remember the effect of this and use that information for the rest of the game.

In certain situations it is clear how such a subgoal can be achieved and a *policy*, which defines which actions to take in which state, can be defined to achieve it. A policy to achieve a specific subgoal is called an *option* [2]. Thus, an option selects an action, given a game state, that aims at satisfying its subgoal. In this paper, options are game-independent. The options are expected to guide the exploration of a game’s search space to feasible areas.

We propose a new algorithm called *option Monte Carlo tree search* (O-MCTS) that extends MCTS to use options. Because O-MCTS chooses between options rather than actions when playing a game, we expect it to be able to plan more efficiently, at a higher level of abstraction. Furthermore, we introduce *option learning MCTS* (OL-MCTS), an extension of O-MCTS that approximates which of the available options work well for the game it is playing. This can be used to shift the focus of the tree search exploration to more promising options. This information can be transferred to subsequent levels in order to increase performance.

We compare our algorithms to MCTS on games from the GVGAI competition. Our results indicate that the O-MCTS and OL-MCTS algorithms outperform MCTS in games that require a high level of action planning, e.g., games in which something has to be picked up before a door can be opened. In most other games, O-MCTS and OL-MCTS perform at least as well as MCTS.

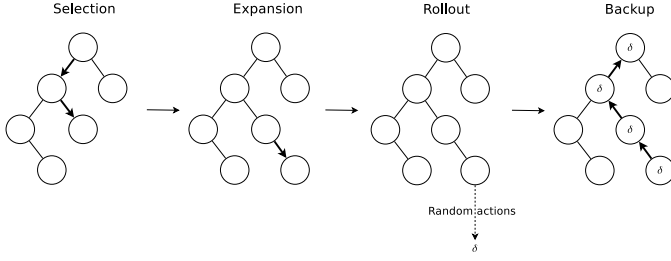


Fig. 1. One MCTS iteration. This process is repeated in order to improve the estimates of action values. Circles represent states, edges represent actions.

## II. BACKGROUND

We first explain the most important concepts needed to understand the algorithms that are proposed in this paper. We first describe *Markov decision processes (MDPs)*, then MCTS, then options and finally the *video game description language (VGDL)*.

### A. Markov Decision Processes

We treat games as MDPs, which provide a mathematical framework for use in decision making problems. An MDP is a tuple  $\langle S, A, T, R \rangle$ , where  $S$  denotes the set of states,  $A$  is the set of possible actions,  $T$  is the transition function and  $R$  is the reward function. Since an MDP is fully observable, a state in  $S$  contains all the information of the game's current condition: locations of sprites like monsters and portals; the location, direction and speed of the avatar; which resources the avatar has picked up; etcetera.  $A$  is a finite set of actions, the input an agent can deliver to the game.  $T$  is a transition function defined as  $T : S \times A \times S \rightarrow [0, 1]$ ; it specifies the probabilities over the possible next states, when taking an action in a state.  $R$  is a reward function defined as  $R : S \times A \times S \rightarrow \mathbb{R}$ . When the game score changes, the difference is viewed as the reward. Algorithms maximize the cumulative reward. In the scope of this paper algorithms only observe state transitions and rewards when they happen and do not have access to  $T$  and  $R$ .

### B. Monte Carlo Tree Search

The success of MCTS started in 2006, when the tree search method and UCT formula were introduced, yielding good results in Computer Go [3]. Since 2006, the algorithm has been extended with many variations. It is still being used for other computer games [4], including the GVGAI competition [5]. In this paper, we use MCTS as the basis for the new algorithms.

This section explains how MCTS approximates action values for states. A tree is built incrementally from the states and actions that are visited in a game. Each node in the tree represents a state and each edge represents an action taken in that state. MCTS consists of four phases that are constantly repeated, as depicted in Figure 1. The root node of the tree represents the current game state. Then, the first action is chosen by an *expansion* strategy and subsequently simulated. This results in a new game state, for which a node is created. After expansion, a *rollout* is done from the new

node, which means that a simulation is run from that node, applying random actions until a predefined stop criterion is met. Finally, the score difference resulting from the rollout is *backed up* to the root node, which means that the reward is saved to all visited nodes, after which a new iteration starts. When all actions are expanded in a node, that node is deemed *fully expanded*. This means that MCTS will use its *selection* strategy to select child nodes until a node is selected that is not fully expanded. Then, the expansion strategy is used to create a new node, after which a rollout takes place and the results are backed up.

The selection strategy selects optimal actions in internal tree nodes by analyzing the values of their child nodes. An effective selection strategy is UCT, which employs an exploration bonus to balance the choice between poorly explored actions with a high uncertainty about their value and actions that have been explored extensively, but have a higher value [6]. A child node  $j$  is selected to maximize

$$UCT = v_{s'} + C_p \sqrt{\frac{2 \ln n_s}{n_{s'}}} \quad (1)$$

Where  $v_{s'}$  is the value of child  $s'$  as calculated by the backup function,  $n_s$  and  $n_{s'}$  are the number of times nodes  $s$  and child  $s'$  have been visited and  $C_p > 0$  is a constant that shifts priority from exploration to exploitation.

The traditional expansion strategy is to explore each action at least once in each node. After all actions have been expanded, the node applies the selection strategy. Some variants of MCTS reduce the branching factor of the tree by only expanding the nodes selected by a special expansion strategy. A specific example is the *crazy stone* algorithm [7], which is an expansion strategy that was originally designed for Go. We will use an adaptation of this strategy in the algorithm proposed in Section V. When using crazy stone, an action  $i$  is selected with a probability proportional to  $u_i$

$$u_i = \exp \left( K \frac{\mu_0 - \mu_i}{\sqrt{2(\sigma_0^2 + \sigma_i^2)}} \right) + \varepsilon_i \quad (2)$$

Each action has an estimated value  $\mu_i$  ordered in such a way that  $\mu_0 > \mu_1 > \dots > \mu_N$ , and a variance  $\sigma_i^2$ .  $K$  is a constant that influences the exploration — exploitation trade off.  $\varepsilon_i$  prevents the probability of selecting a move to reach zero. Its value is proportional to the ordering of the expected values of the possible actions:  $\varepsilon_i = \frac{0.1 + 2^{-i} + a_i}{N}$ . Here,  $a_i$  is 1 when an action is an *atari move*, a go-specific move that can otherwise easily be underestimated by MCTS, and otherwise 0.

After a rollout, the reward is backed up, which means that the estimated value for every node that has been visited in this iteration is updated with the reward of this simulation.

### C. Options

In order to mimic human game playing strategies, such as defining subgoals and subtasks, we use options. Options, or macro-actions, have been proposed by Sutton et al. [2] as a method to incorporate temporal abstraction in reinforcement learning. The majority of the research seems to focus on

learning algorithms, little work has been done on combining options with tree search methods [8], although most learning algorithms are time and memory heavy and tree search methods have shown more promising results on complex games.

An option is a predefined method of reaching a specific subgoal. Formally, it is a triple  $\langle I, \pi, \beta \rangle$  in which  $I \subseteq S$  is an initiation set,  $\pi : S \times A \rightarrow [0, 1]$  is a policy and  $\beta : S^+ \rightarrow [0, 1]$  is a termination condition.

When an agent starts in state  $s$ , it can choose from all of the options  $o \in O$  that have  $s$  in its initiation set  $I_o$ . Then the option's policy  $\pi$  is followed, possibly for several time steps. The agent stops following the policy as soon as it reaches a state that satisfies a termination condition in  $\beta$ . This means that the option has reached its subgoal, or a criterion is met that renders the option obsolete (e.g., its goal does not exist anymore). Afterwards, the agent chooses a new option.

A popular algorithm that uses options instead of actions is SMDP Q-learning [2]. In general, it estimates the expected rewards for using an option in a certain state, in order to find an optimal policy over the option set.

#### D. General Video Game Playing

We use the general video game playing problem as a benchmark for our algorithms. Recent developments in this area include VGDL [9], a framework in which a large number of games can be defined and accessed in a similar manner. Using VGDL, algorithms can access all the games similarly, resulting in a method to compare their performances on several games.

The GVGAI competition provides games written in VGDL. The games function as a black box from which algorithms can only observe the game state. In each game tick, algorithms have limited time to plan their action, during which they can access a *forward model*, which simulates new states and rewards for actions. The actions that are used on the forward model do not influence the real game score. Algorithms should return actions before their simulation time runs out.

The algorithms proposed in this paper will be benchmarked on the GVGAI game sets, using the rules of that competition. This means that the algorithms do not have any access to the game and level descriptions. When an algorithm starts playing a game, it typically knows nothing of the game except for the observations described above.

### III. RELATED WORK

This section covers some popular alternative methods for general video game playing and prior work on tree search with options.

*Deep Q networks (DQN)* [10] is a general video game playing algorithm that trains a convolutional neural network that has the last four pixel frames of a game as input and tries to predict the return of each action. A good policy can then be created by selecting the action with the highest return. In this case it was not desirable to implement DQN because of the limitations proposed by our testing framework. The GVGAI competition framework currently works best for

planning algorithms that use the forward model to quickly find good policies. Learning over the course of several games is difficult. In contrast, DQN typically trains on one game for several days before a good policy is found and does not utilize the forward model, but always applies actions directly to the game in order to learn.

Another alternative is the algorithm *Planning under uncertainty with Macro-Actions (PUMA)*, which applies forward search to options and works on *Partially Observable MDPs (POMDPs)* [11]. PUMA automatically generates goal-oriented MDPs for specific subgoals, the advantage of which is that effective options can be created without requiring any prior knowledge of the POMDP. The disadvantage is that this takes a lot of computation time and thus would not work in the GVGAI framework, where only 40 milliseconds of computation time is allowed between actions. Furthermore PUMA has to find out the optimal length per macro-action, our algorithm can use options of variable length with starting and stopping conditions.

Another algorithm that uses MCTS with macro-actions is called *Purofvio*. Purofvio plans over simple macro-actions which are defined as repeating one action several times [12]. No more complex options are defined. All the options are of exactly the same size. Purofvio is created solely for the physical travelling salesperson problem. Although Purofvio could also work on other games, we decided to create a different algorithm, that is capable of using more complex options.

### IV. O-MCTS

We propose O-MCTS, a novel algorithm that simulates the use of subgoals by planning over options using MCTS, enabling the otherwise infeasible use of options in complex MDPs. The resulting algorithm achieves higher scores than MCTS on complex games that have several subgoals. It works as follows: like in MCTS, a tree of states is built by simulating game plays. The algorithm chooses options instead of actions. When an option is chosen, each next node in the search tree represents an action chosen by that option. The search tree can only branch if an option is finished, i.e., its subgoal is reached. Since traditional MCTS branches on each action, whereas O-MCTS only branches when an option is finished, deeper search trees can be built in the same amount of time. This section describes how the process works in more detail.

The tree representation of an O-MCTS tree is the same as in MCTS: a node represents a state, a connection represents an action. An option spans several actions and therefore several nodes in the search tree, as shown in Figure 2. We introduce a change in the expansion and selection strategies, which select options rather than actions. When a node has an unfinished option, the next node will be created using an action selected by that option. When a node contains a finished option (the current state satisfies its termination condition  $\beta$ ), a new option can be chosen by the expansion or selection strategy. The search tree can only branch when an option is finished.

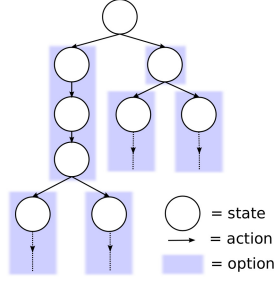


Fig. 2. The search tree constructed by O-MCTS. In each blue box, one option is followed. The arrows represent actions chosen by the option. An arrow leading to a blue box is an action chosen by the option represented by that box.

---

**Algorithm 1** O – MCTS( $O, r, t, d$ )

---

```

1:  $C_{s \in S} \leftarrow \emptyset$   $\triangleright c_s$  is the set of children nodes of  $s$ 
2:  $o \leftarrow \emptyset$   $\triangleright o_s$  will hold the option followed in  $s$ 
3: while  $time\_taken < t$  do
4:    $s \leftarrow r$   $\triangleright$  start from root node
5:   while  $\neg stop(s, d)$  do
6:     if  $s \in \beta(o_s)$  then  $\triangleright$  if option stops in state  $s$ 
7:        $p_s \leftarrow \cup_o (s \in I_o)$   $\triangleright p_s$  = available options
8:     else
9:        $p_s \leftarrow \{o_s\}$   $\triangleright$  continue with current option
10:    end if
11:     $m \leftarrow \cup_o (o_s \in c_s)$   $\triangleright$  set  $m$  to expanded options
12:    if  $p_s = m$  then  $\triangleright$  if all options are expanded
13:       $s' \leftarrow \max_{c \in c_s} uct(s, c)$   $\triangleright$  Eq. 1
14:       $s \leftarrow s'$   $\triangleright$  continue loop with child
15:    else
16:       $\omega \leftarrow \text{random\_element}(p_s - m)$ 
17:       $a \leftarrow \text{get\_action}(\omega, s)$ 
18:       $s' \leftarrow \text{expand}(s, a)$   $\triangleright$  create child  $s'$  using  $a$ 
19:       $c_s \leftarrow c_s \cup \{s'\}$   $\triangleright$  add to set of children
20:       $o_{s'} \leftarrow \omega$ 
21:      break
22:    end if
23:  end while
24:   $\delta \leftarrow \text{rollout}(s')$   $\triangleright$  simulate until stop
25:   $\text{back\_up}(s', \delta)$   $\triangleright$  save reward to parent nodes (Eq. 3)
26: end while
27: return  $\text{get\_action}(\max_{o \in c_r} \text{value}(o), r)$ 

```

---

We describe O-MCTS in Algorithm 1. It is invoked with a set of options  $O$ , a root node  $r$ , a maximum runtime  $t$  in milliseconds and a maximum search depth  $d$ . The set of options used for our experiments is described in Section VI. Two variables are instantiated.  $C_s$  is a set of sets, containing the set of child nodes for each node. The set  $o$  contains which option is followed for each node. The main loop starts at line 3, which keeps the algorithm running until time runs out. The inner loop runs until a node  $s$  is reached that meets a stop criterion defined by the function *stop*, or a node is expanded into a new node. In lines 6 until 10,  $p_s$  is set to all options that are available in  $s$ . If an option has not finished,  $p_s$  contains

only the current option. Otherwise, it contains all the options  $o$  that have state  $s$  in their initiation set  $I_o$ . For example, the agent is playing *zelda* and the current state  $s$  shows no NPCs on screen. If  $o$  is an option for avoiding NPCs,  $I_o$  will not contain state  $s$ , because there are no NPCs on screen, rendering  $o$  useless in state  $s$ .  $p_s$  will thus not contain option  $o$ .

O-MCTS consists of the same four phases as MCTS. In line 11,  $m$  is set to the set of options chosen in the children of state  $s$ . If  $p_s$  is the same set as  $m$ , i.e., all possible options have been explored at least once in node  $s$ , a new node  $s'$  is *selected* by UCT. In line 14,  $s$  is instantiated with the new node  $s'$ , continuing the inner loop using this node. Else, some options are apparently unexplored in node  $s$ . It is *expanded* with a random, currently unexplored option by lines 15 to 22. After expansion or when the stop criterion is met, the inner loop is stopped and a *rollout* is done, resulting in score difference  $\delta$ . This score difference is *backed up* to the parent nodes of  $s$  using the backup function, after which the tree traversal restarts with the root node  $r$ .

A number of functions is used by Algorithm 1. The function *stop* returns true when either the game ends in state  $s$  or the maximum depth is reached in  $s$ . The function *get\_action* lets option  $\omega$  choose the best action for the state in node  $s$ . The function *expand* creates a new child node  $s'$  for node  $s$ .  $s'$  contains the state that is reached when action  $a$  is applied to the state in node  $s$ . Typically, the *rollout* function chooses random actions until *stop* returns true, after which the difference in score achieved by the rollout is returned. In O-MCTS however, *rollout* always applies actions chosen by option  $o$  first and applies random actions after  $o$  is finished. The *back\_up* function traverses the tree through all parents of  $s$ , updating their expected value. In contrast to traditional MCTS, which backs up the mean value of the reward to all parent nodes, a discounted value is backed up. The backup function for updating the value of ancestor node  $s$  when a reward is reached in node  $s'$  looks like this:

$$v_s \leftarrow v_s + \delta \gamma^{d_{s'} - d_s}, \quad (3)$$

where  $\delta$  is the reward that is being backed up,  $v_s$  is the value of node  $s$ .  $d_s$  and  $d_{s'}$  are the node depths of tree nodes  $s$  and  $s'$ . Thus, a node that is a further ancestor of node  $s'$  will be updated with a smaller value.

When the time limit is reached, the algorithm chooses an option from the children of the root node,  $c_r$ , corresponding to the child node with the highest expected value. Subsequently, the algorithm returns the action that is selected by this option for the state in the root node. This action is applied to the game. In the next state, the algorithm restarts by creating a new root node from this state.

We expect that since this implementation of MCTS with options reduces the branching factor of the tree, the algorithm can do a deeper tree search. This is illustrated in Figure 2, where the tree can not branch inside blue boxes. Furthermore, we expect that the algorithm will be able to identify and meet a game's subgoals by using options. In the experiments section we show results that support our expectations.

## V. OL-MCTS: LEARNING OPTION VALUES

Although we expect O-MCTS to be an improvement over MCTS, we also expect the branching factor of O-MCTS's search tree to increase as the number of options increases. When many options are defined, exploring all the options becomes infeasible. In this section, we will define *option values*: the expected mean and variance of an option. We adjust O-MCTS to learn the option values and focus more on the options with higher option values. Especially when a level is played several times, we expect this to be advantageous. We call the new algorithm *Option Learning MCTS (OL-MCTS)*. We expect that OL-MCTS can create deeper search trees than O-MCTS in the same amount of time, which results in more accurate node values and an increased performance. Furthermore, we expect that this effect is the greatest in games where the set of possible options is large, or where only a small subset of the option set is needed in order to win.

In general, OL-MCTS saves the return of each option after it is finished, which is then used to calculate global option values. During the expansion phase of OL-MCTS, options that have a higher mean or variance in return are prioritized. Contrary to O-MCTS not all options are expanded, but only those with a high variance or mean return. The information learned in a game can be transferred if the same game is played again by supplying OL-MCTS with the option values of the previous game.

The algorithm learns the option values,  $\mu$  and  $\sigma$ . The expected mean return of an option  $o$  is denoted by  $\mu_o$ . This state-independent number represents the returns that were achieved in the past by an option for a game. Similarly, the variance of all the returns of an option  $o$  is saved to  $\sigma_o$ .

For the purpose of generalisation, we divide the set of options into *types* and *subtypes*. The option for going to a movable sprite has type *GoToMovableOption*. An instance of this option exists for each movable sprite in the game. A subtype is made for each sprite type (i.e., each different looking sprite). The option values are saved and calculated per subtype. Each time an option  $o$  is finished, its subtype's values  $\mu_o$  and  $\sigma_o$  are updated by respectively taking the mean and variance of all the returns of this subtype. This enables the algorithm to generalize over subtypes.

Using option values, we can incorporate the progressive widening algorithm from Equation 2, crazy stone, to shift the focus of exploration to promising regions of the tree. The crazy stone algorithm is applied in the expansion phase of OL-MCTS. As a result, not all children of a node will be expanded, but only the ones selected based on crazy stone. When using crazy stone, we can select the same option several times, this enables deeper exploration of promising subtrees, even during the expansion phase. After a predefined number of visits  $v$  to a node, the selection strategy UCT is followed in that node to tweak the option selection. When it starts using UCT, no new expansions will be done in this node.

The new algorithm (Algorithm 2) has two major modifications. The updates of the option values are done in line 7.

---

### Algorithm 2 OL – MCTS( $O, r, t, d, v, \mu, \sigma$ )

---

```

1:  $C_{s \in S} \leftarrow \emptyset$ 
2:  $\mathbf{o} \leftarrow \emptyset$ 
3: while  $time\_taken < t$  do
4:    $s \leftarrow r$ 
5:   while  $\neg stop(s, d)$  do
6:     if  $s \in \beta(o_s)$  then
7:        $update\_values(s, o_s, \mu, \sigma)$   $\triangleright$  update  $\mu$  and  $\sigma$ 
8:        $\mathbf{p}_s \leftarrow \cup_o(s \in I_{o \in O})$ 
9:     else
10:       $\mathbf{p}_s \leftarrow \{o_s\}$ 
11:    end if
12:     $\mathbf{m} \leftarrow \cup_o(o_s \in \mathbf{c}_s)$ 
13:    if  $n_s < v$  then  $\triangleright$  if state is visited  $< v$  times
14:       $\mathbf{u}_s \leftarrow crazy\_stone(\mu, \sigma, \mathbf{p}_s)$   $\triangleright$  Eq. 2
15:       $\omega \leftarrow weighted\_random(\mathbf{u}_s, \mathbf{p}_s)$ 
16:      if  $\omega \notin \mathbf{m}$  then  $\triangleright$  option  $\omega$  not expanded
17:         $a \leftarrow get\_action(\omega, s)$ 
18:         $s' \leftarrow expand(s, a)$ 
19:         $\mathbf{c}_s \leftarrow \mathbf{c}_s \cup \{s'\}$ 
20:         $o_{s'} \leftarrow \omega$ 
21:      break
22:    else  $\triangleright$  option  $\omega$  already expanded
23:       $s' \leftarrow s \in \mathbf{c}_s : o_s = \omega$   $\triangleright$  child that uses  $\omega$ 
24:    end if
25:    else  $\triangleright$  apply UCT
26:       $s' \leftarrow uct(s)$ 
27:    end if
28:     $s \leftarrow s'$ 
29:  end while
30:   $\delta \leftarrow rollout(s')$ 
31:   $back\_up(s', \delta)$ 
32: end while
33: return  $get\_action(\max_{o \in \mathbf{c}_r} value(o), r)$ 

```

---

The function *update\_values* takes the return of the option  $o$  and updates its mean  $\mu_o$  and variance  $\sigma_o$  by calculating the new mean and variance of all returns of that option subtype. The second modification starts on line 13, where the algorithm applies crazy stone if the current node has been visited less than  $v$  times. If the node is visited more than  $v$  times, it applies UCT similarly to O-MCTS. The *crazy\_stone* function returns a set of weights over the set of possible options  $\mathbf{p}_s$ . A weighted random then chooses a new option  $\omega$  by using these weights. If  $\omega$  has not been explored yet, i.e., there is no child node of  $s$  in  $\mathbf{c}_s$  that uses this option, the algorithm chooses and applies an action and breaks to rollout in lines 17 to 27. This is similar to the expansion steps in O-MCTS. If  $\omega$  has been explored in this node before the corresponding child node  $s'$  is selected from  $\mathbf{c}_s$  and the loop continues like when UCT selects a child.

We expect that by learning option values and applying crazy stone, the algorithm can create deeper search trees than O-MCTS. These trees are focused more on promising areas of the search space, resulting in improved performance. Furthermore,



we expect that by transferring option values to the next game, the algorithm can improve after replaying games.

## VI. EXPERIMENTS

In this section we describe our experiments on O-MCTS and OL-MCTS. The algorithms are compared to the MCTS algorithm, as described in Section II-B. All algorithms are run on a set of twenty-eight different games in the VGDL framework. The set consists of all the games from the first four training sets of the GVGAI competition, excluding puzzle games that can be solved by an exhaustive search and have no random component (e.g. NPCs). Each game has five levels.

Firstly, we compare O-MCTS to MCTS by showing the win ratio and mean score of both algorithms on all the games. Secondly we show the improvement that OL-MCTS makes compared to O-MCTS when it is allowed 4 games of learning time. Lastly we compare the three algorithms by summing up all the victories of all the levels of each game.

For these experiments we construct an option set which is aimed at providing action sequences for any type of game, since the aim here is general video game playing. Note that since the option set is a variable of the algorithm, either a more specific or automatically generated set of options can be used by the algorithm as well.

The set of option types consists of one option that executes a specific action once, an option that avoids the nearest NPC by moving away from it, an option that moves to a movable sprite until it is close to it (but not on it). There are options that go to a movable sprite and options to go to a certain position in the game. Lastly we create an option that waits until an NPC is at a certain distance and then fires the weapon. The specifics about the option set can be found in the original thesis [13].

For each option type, a subtype per visible sprite type is created during the game. For each sprite, an option instance of its corresponding subtype is created. For example, the game *zelda* contains three different sprite types (excluding the avatar and walls); monsters, a key and a portal. The first level contains three monsters, one key and one portal. The aim of the game is to collect the key and walk towards the portal without being killed by the monsters. The score is increased by 1 if a monster is killed, i.e., its sprite is on the same location as the sword sprite, if the key is picked up, or when the game is won. *go to movable* and *go near movable* options are created for each of the three monsters and for the key. A *go to position* option is created for the portal. One *go to nearest sprite of type* option is created per sprite type. One *wait and shoot* option is created for the monsters and one *avoid nearest NPC* option is created. This set of options is  $O$ , as defined in Section II-C. In a state where, for example, all the monsters are dead, the possible option set  $\mathbf{p}_s$  does not contain the *avoid nearest NPC*, *go to movable* and *go near movable* options for the monsters.

The *go to ...* and *go near ...* options utilize an adaptation of the  $A^*$  algorithm to plan their routes [14]. An adaptation is needed, because at the beginning of the game there is no knowledge of which sprites are traversable by the avatar and which are not. Therefore, during every move that is simulated

by the agent, the  $A^*$  module has to update its beliefs about the location of walls and other blocking objects. This is accomplished by comparing the movement the avatar wanted to make to the movement that was actually made in game. If the avatar did not move, it is assumed that all the sprites on the location the avatar should have arrived in are blocking sprites. Our  $A^*$  keeps a *wall score* for each sprite type. When a sprite blocks the avatar, its wall score is increased by one. Additionally, to prevent the avatar from walking into deadly sprites, when a sprite kills the avatar, its wall score is increased by 100. Traditionally the  $A^*$ 's heuristic uses the distance between two points. Our  $A^*$  adaptation adds the wall score of the goal location to this heuristic, encouraging the algorithm to take paths with a low wall score. This method enables  $A^*$  to try to traverse paths that were unavailable earlier, while preferring safe and easily traversable paths. For example in *zelda*, a door is closed until a key is picked up. Our  $A^*$  implementation will still be able to plan a path to the door once the key is picked up, to win the game. Note that because the games can be stochastic,  $A^*$  has to be recalculated for each simulation.

We empirically optimize the parameters of the algorithms for the experiments. We use discount factor  $\gamma = 0.9$ , maximum action time  $t = 40$  milliseconds. The maximum search depth  $d$  is set to 70, which is higher than most alternative tree search algorithms, for example in the GVGAI competition, use. This is possible because the search tree has a relatively low branching factor. The number of node visits after which uct is used,  $v$ , is set to 40. Crazy stone parameter  $K$  is set to 0.5. For comparison, we use the MCTS algorithm provided with the Java implementation of VGDL which employs a maximum tree depth of 10, because the branching factor is higher. Both algorithms have uct constant  $C_p = \sqrt{2}$ . Unfortunately, comparing to Q-learning with options was impossible, because the state space of these games is too big for the algorithm to learn any reasonable behavior. All the experiments are run on an Intel i7-2600, 3.40GHz quad core processor with 6 GB of DDR3, 1333 MHz RAM memory. In all the following experiments on this game set, each algorithm plays each of the 5 levels of every game 20 times.

First, we will describe the results of the O-MCTS algorithm in comparison with MCTS. This demonstrates the improvement that can be achieved by using our algorithm. The games in this and the following experiments are ordered from left to right by the performance of an algorithm that always chooses a random action, indicating the complexity of the games. Figure 3 shows the win ratio and normalized score of the algorithms for each game. In short, the O-MCTS algorithms performs at least as good as MCTS in almost all games, and better in more than half.

O-MCTS outperforms MCTS in the games *missile command*, *bait*, *camel race*, *survive zombies*, *firestorms*, *lemmings*, *firecaster*, *overload*, *zelda*, *chase*, *boulderchase* and *eggomania* winning more games or achieving a higher mean score. By looking at the algorithm's actions for these games, we can see that O-MCTS succeeds in efficiently planning paths in a dangerous environment, enabling it to do a further forward



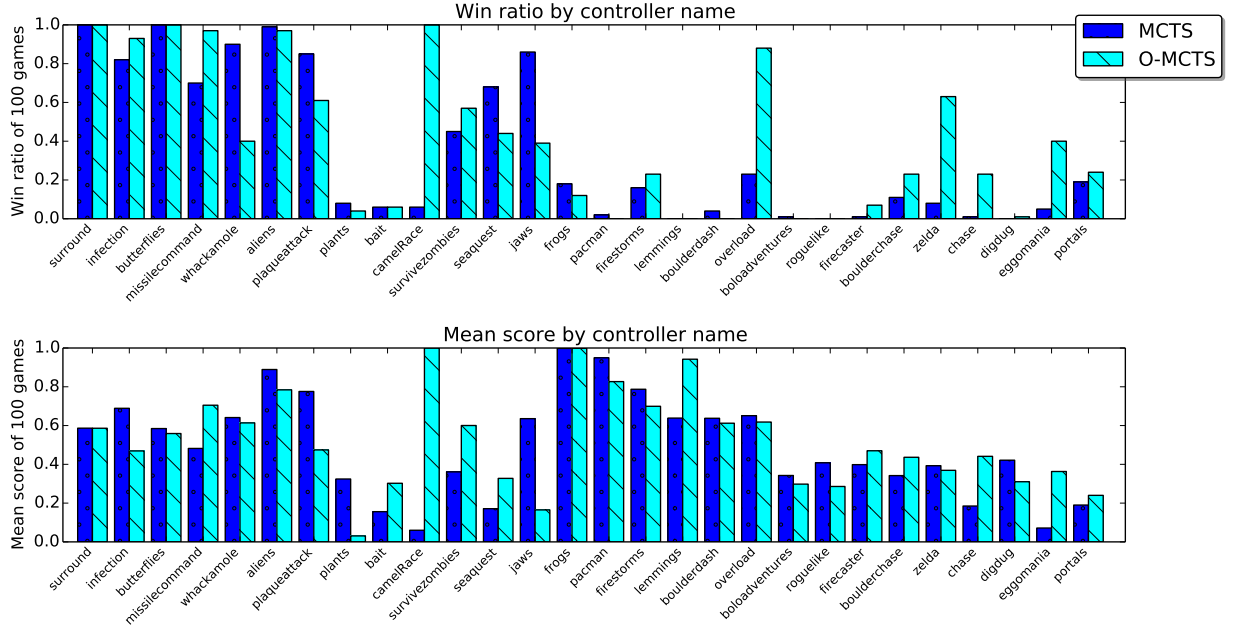


Fig. 3. Win ratio and mean normalized score of the algorithms per game. O-MCTS outperforms MCTS.

search than MCTS. *Camel race* requires the player to move to the right for 80 consecutive turns to reach the finish line. No intermediate rewards are given to indicate that the agent is walking in the right direction. This is hard for MCTS, since it only looks 10 turns ahead. O-MCTS always wins this game, since it can plan forward a lot further. Furthermore, the rollouts of the option for walking towards the finish line have a bigger chance of reaching the finish line than the random rollouts executed by MCTS. In *zelda* we can see that the MCTS algorithm achieves roughly the same score as O-MCTS, but does not win the game, since picking up the key and walking towards the door is a difficult action sequence. We assume that the mean score achieved by MCTS is because it succeeds in killing the monsters, whereas O-MCTS achieves its score by picking up the key and walking to the door. These results indicate that O-MCTS performs better than MCTS in games where a sequence subgoals have to be reached.

The MCTS algorithm performs better than O-MCTS in *pacman*, *whackamole*, *jaws*, *sequest* and *plaque attack* (note that for *sequest*, O-MCTS has a higher mean score, but wins less than MCTS). A parallel between these games is that they have a big number of sprites, for each of which several options have to be created by O-MCTS. When the number of options becomes too big, constructing the set of possible options  $p_s$  for every state  $s$  becomes so time-consuming that the algorithm has too little time to build a tree and find the best possible action. To test this hypothesis we ran the same test with an increased computation time of 120ms and found that the win ratio of O-MCTS increases to around 0.8 for *sequest* and *plaque attack*, whereas the win ratio for MCTS increased to 0.9 and 0.7 respectively. This means that with more action time, the difference between O-MCTS and MCTS is reduced for *sequest* and O-MCTS outperforms MCTS on

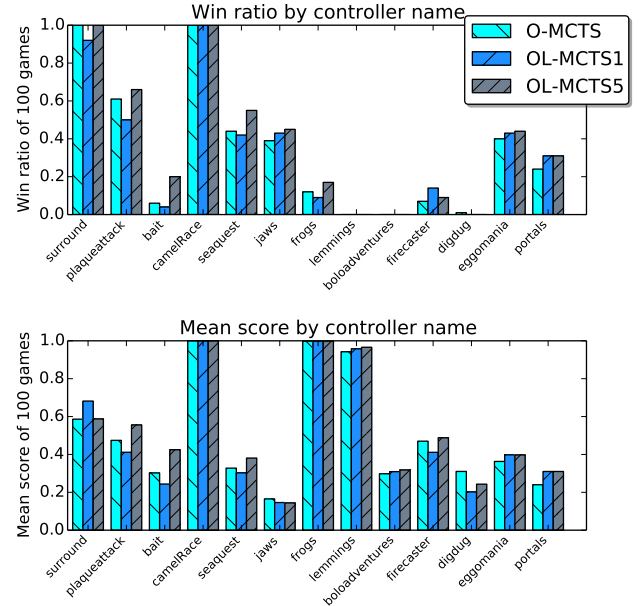


Fig. 4. Normalized win ratio and score comparison of OL-MCTS and O-MCTS. OL-MCTS outperforms O-MCTS by a small margin in some games. In the games that are not shown both algorithms perform equally.

*plaque attack*.

Secondly, we compare OL-MCTS to O-MCTS by running it on the same set of games. The option learning algorithm is allowed four learning games, after which the fifth is used for the comparisons. Figure 4 shows the performance difference between O-MCTS and OL-MCTS on some games. For the other games, the performance was approximately the same. Here OL-MCTS1 shows the performance of OL-MCTS on the first game. OL-MCTS5 shows the performance of the

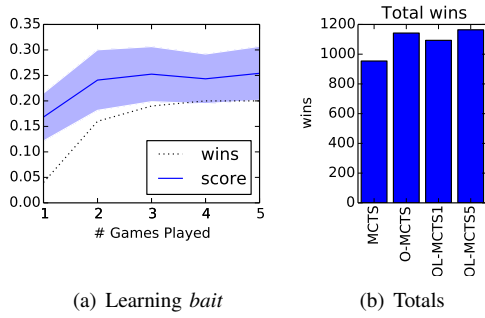


Fig. 5. Learning improvement on game *bait*, it shows win ratio and normalized score. Total number of wins of the algorithms on all games.

algorithm after learning for four games.

We can see that, although the first iteration of OL-MCTS sometimes performs a bit worse than O-MCTS, the fifth iteration often scores at least as high, or higher than O-MCTS. We expect that the loss of performance in OL-MCTS1 is a result of the extra overhead that is added by the crazy stone algorithm: a sorting of all the option values has to take place in each tree node. The learning algorithm significantly improves score and win ratio for the game *bait*. Figure 5(a) shows the improvement in score and win ratio for this game. There are two likely explanations for this improvement: 1. There are sprites that kill the player, which the algorithm learns to avoid 2. The algorithm learns that it should pick up the key.

Furthermore, we can see small improvements on the games *sequest*, *plague attack* and *jaws*, on which O-MCTS performs worse than MCTS. Although OL-MCTS does not exceed the score of the MCTS algorithm, this improvement suggests that OL-MCTS is on the right path of improving O-MCTS.

## A. Results

Summarizing, our tests indicate that on complex games O-MCTS outperforms MCTS. For other games it performs at least as well, as long as the number of game sprites is not too high. The OL-MCTS algorithm can increase performance for some of the games, such as *bait* and *plague attack*. On other games, little to no increased performance can be found.

An overview of the results is depicted in Figure 5(b), which shows the sum of wins over all games, all levels. It shows a significant ( $p < 0.05$ ) improvement of O-MCTS and OL-MCTS over MCTS. There is no significant difference between performance of OL-MCTS over O-MCTS, although our results suggest that it does improve for a subset of the games.

## VII. CONCLUSIONS AND FUTURE WORK

From the experimental results we may conclude that the O-MCTS algorithm almost always performs at least as well as MCTS. It excels in games with both a small level grid or a small amount of sprites and high complexity, such as *zelda*, *overload* and *eggomania*. Furthermore, O-MCTS can look further ahead than most tree searching alternatives, resulting in a high performance on games like *camel race*, in which reinforcement is sparse. An inherent advantage of having deep

search trees is that the probability of an promising option not finishing reduces. We confirm our hypothesis that by using options O-MCTS can win more games than MCTS. The algorithm performs worse than expected in games with a high amount of sprites, since the size of the option set becomes so large that maintaining it takes a lot of time, leaving too little time for tree building. Over all twenty-eight games, O-MCTS wins more games than MCTS.

The results of OL-MCTS indicate that it is possible to learn about which options work better, meaning that in the future it should be possible to completely remove infeasible options that have low expected rewards from the option set. We expect that this could reduce the computation time O-MCTS needs to construct and check all the options. However, the algorithm can be further improved.

Furthermore, more research should be done in the influence of the option set. The  $A^*$  algorithm could be replaced by a simpler algorithm, such as Enforced Hill Climbing [15]. The learning algorithm could be improved by calculating the option values differently. An alternative method can use discounting in order to prioritize more recent observations.

## REFERENCES

- [1] D. Perez, S. Samothrakis, J. Togelius, T. Schaul, S. Lucas, A. Couëtoux, J. Lee, C.-U. Lim, and T. Thompson, "The 2014 general video game playing competition."
- [2] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1, pp. 181–211, 1999.
- [3] S. Gelly, Y. Wang, R. Munos, and O. Teytaud, "Modification of uct with patterns in monte-carlo go," 2006.
- [4] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton *et al.*, "A survey of monte carlo tree search methods," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 1, pp. 1–43, 2012.
- [5] D. Perez, S. Samothrakis, and S. Lucas, "Knowledge-based fast evolutionary mcts for general video game playing," in *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*. IEEE, 2014, pp. 1–8.
- [6] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *Machine Learning: ECML 2006*. Springer, 2006, pp. 282–293.
- [7] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," in *Computers and games*. Springer, 2007, pp. 72–83.
- [8] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems*, vol. 13, no. 1-2, pp. 41–77, 2003.
- [9] T. Schaul, "A video game description language for model-based or interactive learning," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013, pp. 1–8.
- [10] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.
- [11] R. He, E. Brunskill, and N. Roy, "Puma: Planning under uncertainty with macro-actions," in *AAAI*, 2010.
- [12] E. J. Powley, D. Whitehouse, P. Cowling *et al.*, "Monte carlo tree search with macro-actions and heuristic route planning for the physical travelling salesman problem," in *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*. IEEE, 2012, pp. 234–241.
- [13] M. de Waard, "Monte carlo tree search with options for general video game playing," Ph.D. dissertation, University of Amsterdam, February 2016.
- [14] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 100–107, 1968.
- [15] B. Ross, "General video game playing with goal orientation," September 2014.

# Learning opening books in partially observable games: using random seeds in Phantom Go

Tristan Cazenave<sup>1</sup>, Jialin Liu<sup>2,3</sup>, Fabien Teytaud<sup>4</sup>, and Olivier Teytaud<sup>2</sup>

<sup>1</sup>Lamsade, Univ. Paris Dauphine, Paris, France

Email: cazenave@lamsade.dauphine.fr

<sup>2</sup>Inria, CNRS UMR 8623, Univ. Paris-Sud, Gif-sur-Yvette, France

Email: {lastname.firstname}@inria.fr

<sup>3</sup>CSEE, Univ. of Essex, Colchester, UK

Email: jialin.liu@essex.ac.uk

<sup>4</sup>Lisic, Univ. Littoral, France, Calais, France

Email: fabien.teytaud@lisic.univ-littoral.fr

**Abstract**—Many artificial intelligences (AIs) are randomized. One can be lucky or unlucky with the random seed; we quantify this effect and show that, maybe contrarily to intuition, this is far from being negligible. Then, we apply two different existing algorithms for selecting good seeds and good probability distributions over seeds. This mainly leads to learning an opening book. We apply this to Phantom Go, which, as all phantom games, is hard for opening book learning. We improve the winning rate from 50% to 70% in 5x5 against the same AI, and from approximately 0% to 40% in 5x5, 7x7 and 9x9 against a stronger (learning) opponent.

## I. INTRODUCTION

### A. Offline learning in games

Offline learning in games can be e.g. endgame table building [1], opening book construction by self-play [2], or parameter estimation [3]. We propose the use of Random-Seed-portfolios, which consists in optimizing the probability distribution on random seeds, for offline learning in games for which randomized AIs perform well. This approach will essentially, though not only and not explicitly, learn at the level of the opening book. Learning opening books is particularly hard in partially observable games, due to the difficult belief state estimation; therefore, this recent approach by random seeds is particularly suitable in this case.

The random seeds approach has already been proposed for the game of Go [4], [5], but the present paper is, to the best of our knowledge, the first application to *partially observable games*, and our resulting algorithm outperforms by far the original algorithm, which is at the current top level in Phantom Go, including its traditional board sizes. This is mainly obtained through opening book learning - which is a hard task in partially observable games.

### B. Randomized artificial intelligences

1) *Why randomizing AIs.*: There are games in which optimal policies are randomized and, beyond that, in many cases

the state of the art is made of randomized algorithms, in particular since the advent of Monte Carlo Tree Search [6], [7]. Randomized AIs are also required when the AI should be robust to “overfitting” by an opponent - i.e. when we do not want an opponent to be able to learn, by repeated games, a simple winning strategy. A deterministic AI is certainly not suitable in such a case, e.g. for playing on a server or for the pleasure/education of a human opponent. Still, we point out that our approach makes sense in terms of pure performance against the baseline algorithms.

2) *The original Monte Carlo approach in games.*: The Monte Carlo approach in games goes back to [8]. The basic idea is to evaluate a position using random simulations. The value at a state  $s$  is obtained by averaging the result of hundreds of games played randomly from this state  $s$ . This is compliant with partially observable games by randomly sampling the hidden parts of the state. With ad hoc randomization, this approach is the state of the art in Phantom Go [9].

3) *Improvements of the original Monte Carlo approach.*: The original Monte Carlo method for games has been vastly improved [10], [11]. For fully observable games it was outperformed by Monte Carlo Tree Search [6], which adds a tree search to the Monte Carlo evaluation principle. For fully observable puzzles (one player games), nested Monte Carlo often outperforms Monte Carlo [12], [13]. In partially observable games with large number of hidden states, Monte Carlo remains at the top of game programming [14], [15].

### C. Boosting randomized artificial intelligences and learning opening books

Randomized AIs can be seen as random samplers of deterministic policies. A random seed is randomly drawn, and then a deterministic AI, depending on this seed, is applied. The choice of the random seed is usually considered of negligible importance. However, a recent work [16] has shown that

random seeds have an impact, and that the bias inherent to the use of a given randomized AI, which has an implicit probability distribution on random seeds, can be significantly reduced by analyzing the impact of random seeds. We here extend this work to a more challenging case, namely Phantom Go.

Section II describes Phantom Go, our testbed for experiments. Section III describes our approach for boosting random seeds. Section IV presents experimental results.

## II. PHANTOM GO

The game of Phantom Go is a two-player game with hidden information. It consists in playing Go without seeing the other player's moves. Each player does not see the board of the other player. In addition, there is a reference board that is managed by a referee and that the players do not see either. On each player's turn, the player proposes a move to the referee. If the move is legal on the reference board, it is played on the reference board and it is the other player's turn. If the move is illegal on the reference board, the referee tells the player that the move is illegal and the player is asked to play another move. The referee is in charge of maintaining the reference board and of telling illegal moves to the players. The game is over when the two players pass.

Monte Carlo methods have been used in Phantom Go since 2005. The resulting program plays at the level of strong human Go players. Monte Carlo Phantom Go was one of the early success of Monte Carlo methods in Go and related games. The principle of Monte Carlo Phantom Go is to randomly choose a "determinization" (i.e. a filling of the unknown parts of the state space) consistent with the previous illegal moves before each payout. For each possible move, the move is simulated, followed by a determinization and a random payout. Thousands of such determinizations and payout sequences are played for each move and the move with the highest resulting mean is played.

This simple method has defeated more elaborate methods using Monte Carlo Tree Search in the former computer Olympiads. Using a parallelization of the algorithm on a cluster our program won five gold medals and one silver medal during the last six computer Olympiads. When winning the silver medal, the program lost to another program using the same method. The program also played three strong Go players in exhibition matches during the 2011 European Go Congress and won all of its three games.

## III. RANDOM SEEDS AND THEIR BOOSTING

### A. Seeds in games

We consider a randomized artificial intelligence (AI), equipped with random seeds. Our experiments will be performed on a Monte Carlo approach for Phantom Go, though the method is generic and could be applied to any randomized algorithm such as those cited in Section I-B.

We can decide the seed - and when the seed is fixed, the AI becomes deterministic. The original (randomized) algorithm

**Algorithm 1** The BestSeed algorithm for boosting a randomized AI. There is a parameter  $K$ ;  $K$  greater leads to better performance but slower computations. The resulting AI is deterministic, but it can be made stochastic by random permutations of the 8 symmetries of the board.

**Require:**  $K$ , and a randomized AI.

```

1: for  $i \in \{1, \dots, K\}$  do
2:   for  $j \in \{1, \dots, K\}$  do
3:     Play a game between
       • an AI playing with seed  $i$  as Black;
       • an AI playing with seed  $j$  as White.
4:    $M_{i,j} \leftarrow 1$  if Black wins, 0 otherwise
5:   end for
6: end for
7:  $i_0 \leftarrow \operatorname{argmax}_{i \in \{1, \dots, K\}} \sum_{j=1}^K M_{i,j}$ 
8:  $j_0 \leftarrow \operatorname{argmin}_{j \in \{1, \dots, K\}} \sum_{i=1}^K M_{i,j}$ 
9: return The (deterministic) AI using seed  $i_0$  when playing Black and  $j_0$  when playing White.
```

can be seen as a probability distribution over these deterministic AIs.

A Random-Seed-portfolio (RS-portfolio) consists in optimizing the probability distribution on random seeds. Such an algorithm has been proposed in [16]. We recall below the two algorithms they propose, namely Nash and BestSeed. In both cases, the learning of the probability distribution is based on the construction of a  $K \times K$  binary matrix  $M$ , where  $M_{i,j} = 1$  if Black with random seed  $i$  wins against White with random seed  $j$ , and  $M_{i,j} = 0$  otherwise. This matrix is the learning set; for validating our approach in terms of performance against the original randomized algorithm, we use random seeds which are not in this matrix, and distributed as in the original randomized algorithm.

### B. Strategies for choosing seeds

We describe here two methods for choosing a probability distribution on rows  $i \in \{1, 2, \dots, K\}$  and a probability distribution on columns  $j \in \{1, 2, \dots, K\}$ . These probability distributions are then used as better probability distributions on random seeds at the beginning of later games.

1) *BestSeed approach.*: BestSeed is quite simple; the probability distribution for Black has mass 1 on some  $i$  such that  $\sum_{j \in \{1, \dots, K\}} M_{i,j}$  is maximal. We randomly break ties. For White, we have probability 1 for some  $j$  such that  $\sum_{i \in \{1, \dots, K\}} M_{i,j}$  is minimum. The BestSeed approach is described in Algorithm 1. This method is quite simple, and works because

$$\lim_{K \rightarrow \infty} \frac{1}{K} \sum_{j=1}^K M_{i,j}$$

$$(\text{resp. } \lim_{K \rightarrow \infty} \frac{1}{K} \sum_{j=1}^K M_{j,i})$$

is far from being a constant when  $i$  varies.

2) *Nash approach.*: This section describes the Nash approach. It is more complicated than the BestSeed approach, but it is harder to *exploit*, as detailed in the experimental section. First, we introduce constant-sum matrix games, and then we explain how we use them for building portfolios of random seeds.

a) *Constant-sum matrix games*: We consider constant-sum matrix games; by normalizing matrices, we work without loss of generality on games such that the sum of the rewards for player 1 and for player 2 is one. Consider the following game, parametrized by a  $K \times K$  matrix  $M$ . Black plays  $i$ . White is not informed of Black's choice, and plays  $j$ . The reward for Black is  $M_{i,j}$  and the reward for White is  $1 - M_{i,j}$ .

It is known [17], [18] that there exists at least one Nash equilibrium  $(x, y)$  such that if Black plays  $i$  with probability  $x_i$  and White plays  $j$  with probability  $y_j$ , then neither of the players can improve its expected reward by changing unilaterally his policy. More formally:

$$\exists(x, y), \forall(x', y'), x'^t M y \leq x^t M y \leq x^t M y',$$

where  $x, y, x'$  and  $y'$  are non-negative vectors summing to one. Moreover, the value  $v = x^t M y$  is unique - but the pair  $(x, y)$  is not necessarily unique.

It is possible to compute  $x$  and  $y$  in polynomial time, using linear programming [19]. Some faster methods provide approximate results in sublinear time [20], [21]. Importantly, these fast approximation algorithms are mathematically proved and do not require all the elements of the matrix to be available - only  $O(K \log(K)/\epsilon)$  elements in the matrix have to be computed for a fixed precision  $\epsilon > 0$  on the Nash equilibrium.

b) *Nash portfolio of random seeds*: Consider  $(x, y)$  the Nash equilibrium of the matrix game  $M$ , obtained by e.g. linear programming. Then the Nash method uses  $x$  as a probability distribution over random seeds for Black and uses  $y$  as a probability distribution over random seeds for White. The algorithm is detailed in Algorithm 2. We also tested a sparse version, which gets rid of pure strategies with low values. The algorithm depends on a parameter  $\alpha$ , and it is detailed in Algorithm 3.

### C. Criteria

We now give two performance criteria, namely performance against the baseline (which is the original randomized algorithm) and performance against an agent which can choose its seed with perfect rationality among a finite randomly drawn set of a given cardinal - the second criterion is harder, and simulates an opponent who has "learnt" how to play against us by optimizing his seed.

1) *Performance against the baseline.*: The first criterion is the success rate against the original AI, with its randomized seed. This is precisely the criterion that is optimized in BestSeed; but we perform experiments in cross-validation, i.e. the performance obtained by BestSeed and displayed in Section IV is the performance against seeds which were not used in the learning phase.

---

**Algorithm 2** The Nash method for boosting a randomized AI. There is a parameter  $K$ ;  $K$  greater leads to better performance but slower computations. The resulting AI is stochastic. It is outperformed by BestSeed in terms of winning rate against the original (randomized) algorithm, but harder to overfit.

---

**Require:**  $K$  and a randomized AI.

```

1: for  $i \in \{1, \dots, K\}$  do
2:   for  $j \in \{1, \dots, K\}$  do
3:     Play a game between
      • an AI playing with seed  $i$  as Black;
      • an AI playing with seed  $j$  as White.
4:    $M_{i,j} \leftarrow 1$  if Black wins, 0 otherwise
5:   end for
6: end for
7: Let  $(x, y)$  be a pair of probability distributions over  $\{1, \dots, K\}$ , forming a Nash equilibrium of  $M$ .
8: return The (stochastic) AI using seed
      •  $i_0$  randomly drawn with probability distribution  $x$  when playing Black
      • and  $j_0$  randomly drawn with probability distribution  $y$  when playing White.
```

---



---

**Algorithm 3** The SparseNash method for boosting a randomized AI. Compared to Algorithm 2, there is an additional parameter  $\alpha$ .

---

**Require:**  $K$ ,  $\alpha$  and a randomized AI.

```

1: for  $i \in \{1, \dots, K\}$  do
2:   for  $j \in \{1, \dots, K\}$  do
3:     Play a game between
      • an AI playing with seed  $i$  as Black;
      • an AI playing with seed  $j$  as White.
4:    $M_{i,j} \leftarrow 1$  if Black wins, 0 otherwise
5:   end for
6: end for
7: Let  $(x, y)$  be a pair of probability distributions over  $\{1, \dots, K\}$ , forming a Nash equilibrium of  $M$ .
8:  $x_{max} \leftarrow \max_{1 \leq i \leq K} x_i$ 
9:  $y_{max} \leftarrow \max_{1 \leq i \leq K} y_i$ 
10: For all  $i \in \{1, \dots, K\}$ , if  $x_i < \alpha x_{max}$ , then  $x_i \leftarrow 0$ .
11: For all  $i \in \{1, \dots, K\}$ , if  $y_i < \alpha y_{max}$ , then  $y_i \leftarrow 0$ .
12:  $x \leftarrow x / \sum_{i=1}^K x_i$ 
13:  $y \leftarrow y / \sum_{i=1}^K y_i$ 
14: return The (stochastic) AI using seed
      •  $i_0$  randomly drawn with probability distribution  $x$  when playing Black
      • and  $j_0$  randomly drawn with probability distribution  $y$  when playing White.
```

---

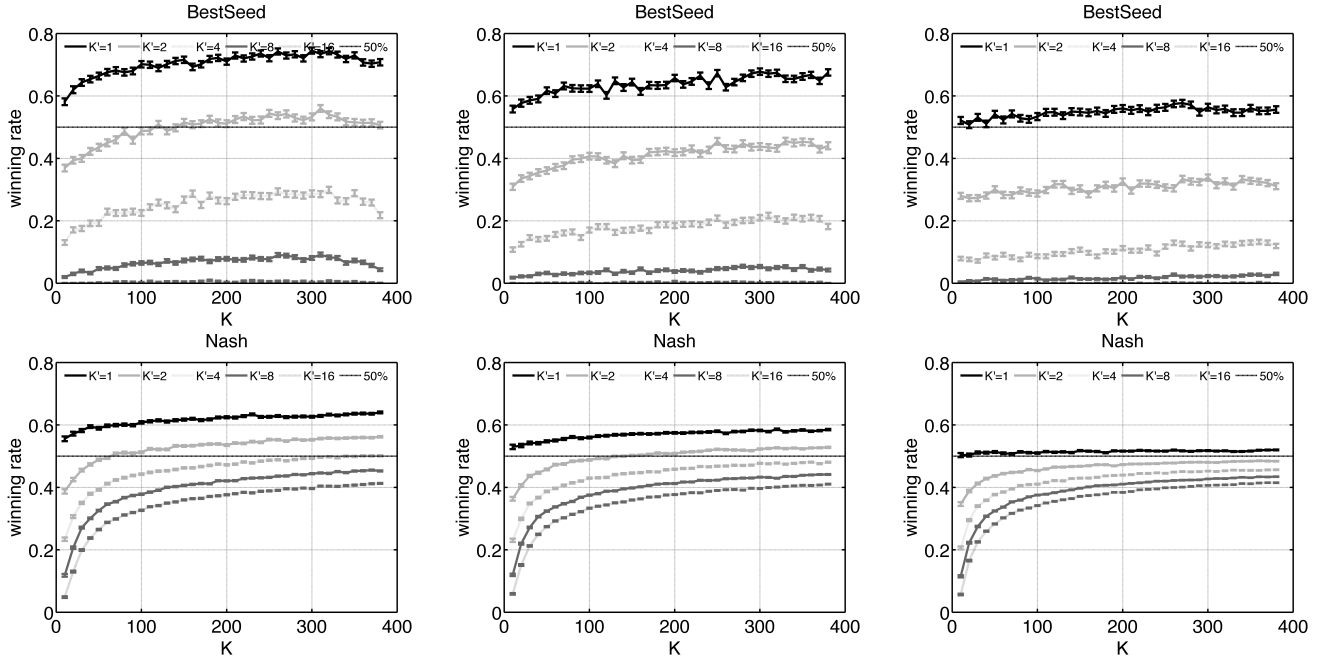


Fig. 1: Winning rate for Phantom Go 5x5 (left), 7x7 (middle) and 9x9 (right). Top: BestSeed; bottom: Nash. X-axis:  $K$  such that we learn on a  $K \times K$  matrix  $M$ . Y-axis: winning rate. The training is performed on a  $K \times K$  matrix. The testing is performed on  $K' \times K'$  strategies, i.e.  $K'$  strategies for Black and  $K'$  strategies for White.  $K' = 1$  corresponds to a randomized seed - this is therefore the original randomized AI, and performance greater than 50% for  $K' = 1$  means that we outperformed the original randomized AI.  $K' > 1$  corresponds to the best performing AI, for each color, among  $K'$  randomly drawn seeds; this is a very difficult opponent, who can try  $K'^2$  strategies and keep only the best of their results. The black dashed curve refers to a winning rate= 50%. The experiments are repeated 1000 times. The standard deviations are shown in the plots.

2) *Performance against an opponent who learns.*: The second criterion is the success rate against an opponent who plays with the original randomized AI, but can test  $K'$  randomly drawn random seeds and can select the best of these seeds. This modelizes the case in which the opponent can choose (perfect choice) one policy among  $K'$  policies. We here consider  $K'$  policies, each of them obtained by fixing the random seed to some random value. We do this choice among  $K'$  policies for Black, and  $K'$  policies for White as well, so that we have indeed the worst performance against  $K' \times K'$  policies. This becomes a very tough criterion when  $K'$  increases - our opponent can basically test  $(K')^2$  openings and choose the one for which we are the weakest.

Obviously, all experiments in the present paper are performed with separate seeds for the learning and the validation experiments, so that no overfitting can explain the positive results in Section IV.

#### IV. EXPERIMENTAL RESULTS

We perform experiments on the Phantom Go testbed. Our randomized AI is Golois [14], [15]. In all our results, we use cross-validation; we test performance against seeds which were not used during the learning phase. We consider values of  $K \leq 400$ . We use the two criteria described in Section III-C, i.e. winning rate against the original randomized algorithm and worst of the winning rates against  $K' \times K'$  deterministic

policies obtained as described in Section III-C2. All presented winning rates are the average between the winning rate as Black and the winning rate as White.

We observe in Figure 1 (also Table I):

- The BestSeed approach clearly outperforms the original method in 5x5, 7x7 and 9x9. The performance is excellent in 5x5, greater than 71%; around 67% in 7x7; it is still good in 9x9 (54%).
- The Nash approach reaches 64% in 5x5, 58% in 7x7. This is already reasonably good for a very randomized game such as Phantom Go; in partially observable games like Phantom Go, Poker, or many card games, several games are usually required for knowing the best among two players. In 9x9, we got only 52% - not very impressive.
- The SparseNash approach outperforms BestSeed in terms of success rate against the original randomized AI, in 5x5 (Figure 2 (top), summarized in Table I). Results are however disappointing on larger board sizes (Figure 2 (middle and bottom); also presented in Table I).

These two methods were tested directly on the original algorithm, without using the symmetries of the game or any prior knowledge. All results are obtained with proper cross-validation. Standard deviations are shown on figures and are negligible compared to deviations from 50%. The approach has a significant offline computational cost; but the online

TABLE I: Winning rate for Phantom Go 5x5, 7x7 and 9x9 with  $K = 380$  (cf. Figure 1).  $\alpha$  is the sparsity parameter (cf. Algorithm 3). The experiments are repeated 1000 times. The standard deviations are shown after  $\pm$ .  $K' = 1$  corresponds to the original algorithm with randomized seed;  $K' = 2$  corresponds to the original algorithm but choosing optimally (after checking their performance against its opponent) between 2 possible seeds, i.e. it is guessing, in an omniscient manner, between 2 seeds, each time an opponent is provided.  $K' = 4$ ,  $K' = 8$ ,  $K' = 16$  are similar with 4, 8, 16 seeds respectively;  $K' = 16$  is a very strong opponent for our original algorithm (our winning rate is initially close to 0), but after Nash seed learning we get results above 40% in 5x5, 7x7 and 9x9.

Board	Method	Winning rate (%)				
		$K' = 1$	$K' = 2$	$K' = 4$	$K' = 8$	$K' = 16$
5x5	Baseline	50	30.5 $\pm$ 0.9	12.5 $\pm$ 0.7	0.5 $\pm$ 0.2	0.0 $\pm$ 0.0
	BestSeed	70.7 $\pm$ 1.0	49.8 $\pm$ 1.1	23.4 $\pm$ 0.9	4.8 $\pm$ 0.4	0.2 $\pm$ 0.1
	Nash	63.8 $\pm$ 0.3	56.1 $\pm$ 0.2	<b>50.3 <math>\pm</math> 0.2</b>	<b>45.4 <math>\pm</math> 0.2</b>	<b>41.3 <math>\pm</math> 0.2</b>
	Sparse	$\alpha = 0.500$	68.3 $\pm$ 0.6	43.9 $\pm$ 0.5	32.8 $\pm$ 0.4	24.4 $\pm$ 0.3
		$\alpha = 0.750$	<b>74.7 <math>\pm</math> 0.8</b>	36.7 $\pm$ 0.9	20.2 $\pm$ 0.6	9.0 $\pm$ 0.4
		$\alpha = 1.000$	76.2 $\pm$ 0.9	31.9 $\pm$ 1.0	8.7 $\pm$ 0.6	0.9 $\pm$ 0.2
7x7	Baseline	50	23.0 $\pm$ 0.9	8.5 $\pm$ 0.6	0.5 $\pm$ 0.2	0.5 $\pm$ 0.2
	BestSeed	<b>66.5 <math>\pm</math> 1.0</b>	44.1 $\pm$ 1.1	19.9 $\pm$ 0.9	3.9 $\pm$ 0.4	0.1 $\pm$ 0.1
	Nash	58.3 $\pm$ 0.2	<b>52.8 <math>\pm</math> 0.2</b>	<b>48.1 <math>\pm</math> 0.2</b>	<b>44.2 <math>\pm</math> 0.1</b>	<b>41.1 <math>\pm</math> 0.1</b>
	Sparse	$\alpha = 0.500$	59.6 $\pm$ 0.3	44.0 $\pm$ 0.2	38.7 $\pm$ 0.2	33.2 $\pm$ 0.2
		$\alpha = 0.750$	58.7 $\pm$ 0.6	44.1 $\pm$ 0.6	21.4 $\pm$ 0.4	14.5 $\pm$ 0.3
		$\alpha = 1.000$	56.4 $\pm$ 1.1	33.0 $\pm$ 1.0	1.7 $\pm$ 0.3	0.0 $\pm$ 0.0
9x9	Baseline	50	27.0 $\pm$ 1.0	4.0 $\pm$ 0.4	1.0 $\pm$ 0.2	0.0 $\pm$ 0.0
	BestSeed	<b>54.4 <math>\pm</math> 1.1</b>	32.8 $\pm$ 1.0	12.2 $\pm$ 0.7	2.8 $\pm$ 0.4	0.1 $\pm$ 0.0
	Nash	51.9 $\pm$ 0.1	<b>48.4 <math>\pm</math> 0.1</b>	<b>45.6 <math>\pm</math> 0.1</b>	<b>43.5 <math>\pm</math> 0.1</b>	<b>41.6 <math>\pm</math> 0.1</b>
	Sparse	$\alpha = 0.500$	52.2 $\pm$ 0.3	45.3 $\pm$ 0.2	35.3 $\pm$ 0.2	31.1 $\pm$ 0.2
		$\alpha = 0.750$	52.4 $\pm$ 0.6	38.6 $\pm$ 0.5	18.4 $\pm$ 0.4	12.5 $\pm$ 0.3
		$\alpha = 1.000$	52.9 $\pm$ 1.1	27.3 $\pm$ 1.0	8.2 $\pm$ 0.6	1.2 $\pm$ 0.2
						0.1 $\pm$ 0.1

computational overhead is zero. The offline computational overhead is  $K^2$  times the cost of one game, plus the Nash solving. The Nash solving by linear programming is negligible in our experiments. For large scale matrices, methods such as [20] should provide much faster results as the number of games would be  $O(K \log(K)/\epsilon^2)$  instead of  $K^2$  for a fixed precision  $\epsilon$ .

## V. CONCLUSIONS

We tested various methods for enhancing randomized AIs by optimizing the probability distribution on random seeds. Some of our methods are not new, but up to now, they were only tested on a fully observable game, without opening book, whereas in fully observable games building an opening book is far less a challenge. We work on Phantom Go, a very challenging problem, with the program which won most competitions in recent years. The three tested methods provide results as follows:

- With BestSeed, we get 71%, 67%, 54% of success rate against the baseline in 5x5, 7x7 and 9x9, just by “managing” the seeds.
- The Nash approach provides interesting results as well, in particular strongly boosting the performance against stronger opponent such as  $K' = 2$ ,  $K' = 4$ ,  $K' = 8$ ,  $K' = 16$ , reaching 40% (in 5x5, 7x7 and 9x9) whereas our original algorithm was close to 0% winning rate for  $K' = 16$ . This means that the opening book we have learnt is robust against stronger opponents than the ones used for the self-play involved in our learning.
- Using the Nash approach with sparsity, with the exponent  $\alpha = .75$  recommended in earlier papers on sparsity [22], maybe not the best for each case separately, but outperforming the baseline in all cases.

The method has no computational overhead online - all the computational cost is an offline learning. As a consequence, the method looks like a free bonus: when your randomized AI is ready, apply Algorithm 2 and get a better AI. The BestSeed method is the best performing one, but it can be overfitted. The Nash approach is less efficient against the original AI, but more robust, i.e. more difficult to overfit.

## Further work

We propose the following further works:

- The approach is quite generic, and could be tested on many games in which randomized AIs are available. For the BestSeed approach, the game does not have to be a two-player game.
- Our work does not use any of the natural symmetries of the game; this should be a very simple solution for greatly improving the results; in particular, it would be much harder to overfit BestSeed if it was randomized by the 8 classical board symmetries.
- Mathematically analyzing the approach is difficult, because we have no assumption on the probability distribution of  $\mathbb{E}_j M_{i,j}$  for a randomly drawn seed  $i$  - how many  $i$  should we test before we have a good probability of having a really good one? Bernstein inequalities [23], [24], [25] for the BestSeed approach, and classical properties of Nash equilibria for the Nash approach, provide only preliminary elements.
- Computing approximate Nash equilibria (using [20] or [21]) should strongly reduce the offline computational cost. The computational cost was not a big deal for the results presented in the present paper, but performance might be much better with  $K$  larger. Approximate Nash equilibria do not need the entire  $K \times K$  matrix; they

only sample  $O(K \log(K)/\epsilon^2)$  elements of the matrix for a precision  $\epsilon$ .

- This last further work opens some problems also in the algorithmic theory of Nash equilibria. We have done the present work in a not anytime manner; we know  $K$  a priori, and we do not have any approximate results until the  $K^2$  games are played. However, we might prefer not to choose a priori a number  $K$  of games, and get anytime approximate results. To the best of our knowledge, [20], [21] have never been adapted to an infinite set of arms. Also, adversarial bandit approaches such as Exp3 [21] have never been parallelized. [20] is parallel, but possibly harder to adapt in an anytime setting.

## REFERENCES

- [1] E. V. Nalimov, C. Wirth, G. M. Haworth *et al.*, “Kqkqk and the kasparov-world game,” *ICGA Journal*, vol. 22, no. 4, pp. 195–212, 1999.
- [2] R. Gaudel, J.-B. Hoock, J. Pérez, N. Sokolovska, and O. Teytaud, “A principled method for exploiting opening books,” in *Computers and Games*. Springer, 2010, pp. 136–144.
- [3] G. Chaslot, M. Winands, I. Szita, and H. van den Herik, “Parameter tuning by cross entropy method,” in *European Workshop on Reinforcement Learning*, 2008. [Online]. Available: <http://www.cs.unimaas.nl/g.chaslot/papers/ewrl.pdf>
- [4] T. Cazenave, J. Liu, and O. Teytaud, “The rectangular seeds of domineering,” in *Computational Intelligence and Games (CIG), 2015 IEEE Conference on*. IEEE, 2015, pp. 530–531.
- [5] D. L. St-Pierre, J. Liu, and O. Teytaud, “Nash reweighting of monte carlo simulations: Tsumego,” in *Evolutionary Computation (CEC), 2015 IEEE Congress on*. IEEE, 2015, pp. 1458–1465.
- [6] R. Coulom, “Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search,” In P. Ciancarini and H. J. van den Herik, editors, *Proceedings of the 5th International Conference on Computers and Games, Turin, Italy*, pp. 72–83, 2006.
- [7] L. Kocsis and C. Szepesvari, “Bandit based Monte-Carlo planning,” in *15th European Conference on Machine Learning (ECML)*, 2006, pp. 282–293.
- [8] B. Bruegmann, “Monte-carlo Go (unpublished draft <http://www.althofer.de/bruegmann-montecarlo.pdf>),” 1993.
- [9] T. Cazenave, “A phantom-go program,” in *Advances in Computer Games*. Springer, 2005, pp. 120–125.
- [10] B. Bouzy, “Associating domain-dependent knowledge and monte carlo approaches within a go program,” *Information Sciences, Heuristic Search and Computer Game Playing IV, Edited by K. Chen*, no. 4, pp. 247–257, 2005.
- [11] B. Bouzy and G. Chaslot, “Bayesian generation and integration of k-nearest-neighbor patterns for 19x19 go,” In G. Kendall and Simon Lucas, editors, *IEEE 2005 Symposium on Computational Intelligence in Games, Colchester, UK*, pp. 176–181, 2005.
- [12] T. Cazenave, “Nested monte-carlo search,” in *IJCAI*, C. Boutilier, Ed., 2009, pp. 456–461.
- [13] J. Méhat and T. Cazenave, “Combining uct and nested monte carlo search for single-player general game playing,” *IEEE Trans. Comput. Intellig. and AI in Games*, vol. 2, no. 4, pp. 271–277, 2010.
- [14] T. Cazenave, “A phantom-go program,” in *Proceedings of Advances in Computer Games*, ser. Lecture Notes in Computer Science, H. J. van den Herik, S.-C. Hsu, T.-S. Hsu, and H. H. L. M. Donkers, Eds., vol. 4250. Springer, 2006, pp. 120–125.
- [15] T. Cazenave and J. Borsboom, “Golois wins phantom go tournament,” *ICGA Journal*, vol. 30, no. 3, pp. 165–166, 2007.
- [16] D. L. Saint-Pierre and O. Teytaud, “Nash and the Bandit Approach for Adversarial Portfolios,” in *CIG 2014 - Computational Intelligence in Games*, ser. Computational Intelligence in Games, IEEE. Dortmund, Germany: IEEE, Aug. 2014, pp. 1–7. [Online]. Available: <https://hal.inria.fr/hal-01077628>
- [17] J. V. Neumann and O. Morgenstern, *Theory of Games and Economic Behavior*. Princeton University Press, 1944. [Online]. Available: <http://jmvivald.cse.sc.edu/library/neumann44a.pdf>
- [18] J. Nash, “Some games and machines for playing them,” Rand Corporation, Tech. Rep. D-1164, 1952.
- [19] B. von Stengel, “Computing equilibria for two-person games,” in *Handbook of Game Theory*, R. Aumann and S. Hart, Eds. Amsterdam: Elsevier, 2002, vol. 3, pp. 1723 – 1759.
- [20] M. D. Grigoriadis and L. G. Khachiyan, “A sublinear-time randomized approximation algorithm for matrix games,” *Operations Research Letters*, vol. 18, no. 2, pp. 53–58, Sep 1995.
- [21] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, “Gambling in a rigged casino: the adversarial multi-armed bandit problem,” in *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 322–331.
- [22] D. Auger, J. Liu, S. Ruetz, D. L. St-Pierre, and O. Teytaud, “Sparse binary zero-sum games,” in *ACML*, 2014.
- [23] S. Bernstein, *The Theory of Probabilities*. Gastehizdat Publishing House, Moscow, 1946.
- [24] W. Hoeffding, “Probability inequalities for sums of bounded random variables,” *Journal of the American Statistical Association*, vol. 58, pp. 13–30, 1963.
- [25] H. Chernoff, “A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations,” *Annals of Math. Stat.*, vol. 23, pp. 493–509, 1952.



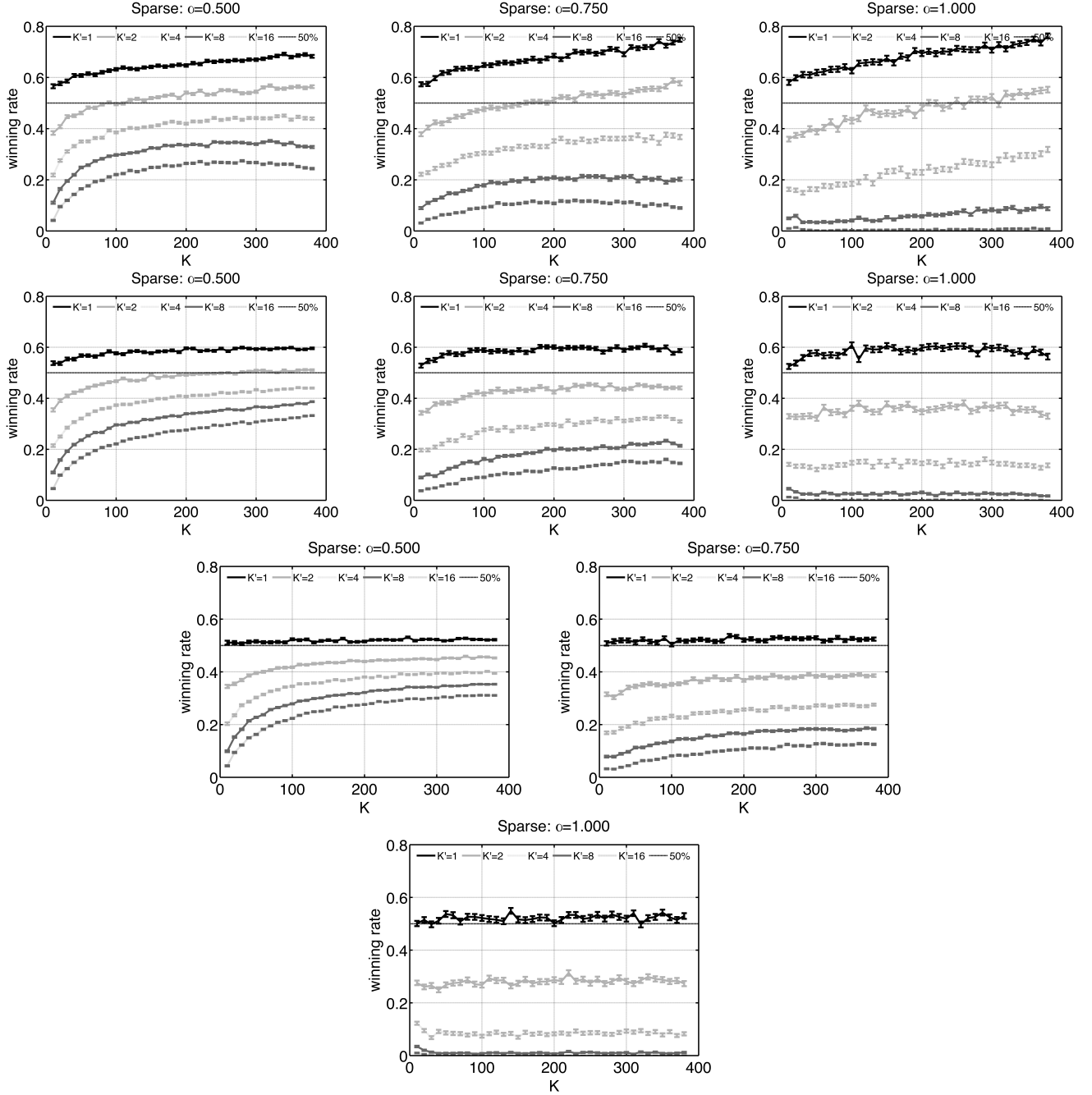


Fig. 2: Winning rate for Phantom Go 5x5 (top), 7x7 (middle) and 9x9 (bottom) using sparse strategy with different sparsity parameter  $\alpha$ . X-axis: parameter  $K$  (size of the learning set). Y-axis: performance in generalization against the original algorithm ( $K' = 1$ ) and against the learning opponent (see Section III-C2;  $K' = 2$  to  $K' = 16$ ).

# Modelling User Retention in Mobile Games

Markus Viljanen\*, Antti Airola, Tapio Pahikkala, Jukka Heikkonen

Department of Information Technology

University of Turku, Finland

\*majuvi@utu.fi

**Abstract**— User activity in five mobile games is found to be accurately described by stochastic processes related to recurrent event models in survival analysis. We specify four simple parametric models and methods to fit them to data which specify this process within day accuracy in the individual user level. This model implies commonly used population level retention metrics: retention, rolling retention and lifetime retention. Furthermore, modelling aids in understanding the underlying phenomena generating these metrics, which is verified visually in five diverse mobile games. The model assists in obtaining analytical insight into frequency and longevity of product use and precipitates predictive modelling by forecasting their evolvement over time.

## I. INTRODUCTION

### A. User Statistics in the Industry

Recent developments in the gaming industry have made analytics a core component of the business model. The past decade has seen the emergence of ubiquitous mobile devices, digital delivery platforms and multiple revenue models. One of the revenue models has become predominant in mobile games: games that are available for free now represent the majority of titles and revenues in all major mobile digital delivery platforms. Majority of these games make money by displaying advertisements, offering in-app purchases or having additional premium versions. This economy of zero cost bundled with paid extras has also been titled Freemium, a portmanteau of free and premium. The goal in this economy is to obtain massive scale with the goal of converting some of the fickle users into paying customers [1]. Analytical insight into the user base is central in achieving this goal, and one of its most important applications is optimizing for both longevity and frequency of user engagement.

The gaming industry uses quantitative measurements called metrics to describe the average use case and provide high level aggregate statistics for business and development decisions. There are several established metrics in the gaming industry, with retention metrics considered the principal metric [1] since without frequency and longevity of product use there is no potential for successful monetization. Retention is often tracked with the aim of following certain rules-of-thumb in the development stage, unsatisfactory retention is thought to be a signal of low quality and predict insufficient monetization in the freemium model. Retention measurements are therefore used in the development stage to refine the game by guiding successive iterations of develop-release-measure cycles.

In the academia, various statistics have been derived and analyzed from past gaming data. There have been studies where the focus has been total time played [2-3], with Weibull

distribution providing a good fit. Wide applicability was reported in [2], where the authors fitted the Weibull distribution to over 3000 games in Steam, and used kernel archetype analysis to distinguish four prototypical total time played distributions. Session length and inter arrival times are another such quantities, often analyzed in combination with multiple other game statistics and fitted with the Weibull distribution [4-6], with one of the earliest studies using Exponential distribution for session lengths and heavy-tailed distribution for inter arrival times [7]. Player churn has been analyzed previously by training general purpose machine learning methods on game features and aggregate session statistics [8-9] or multiple arbitrary event frequencies [10] to predict churn or first purchase. Player engagement has been treated as a regression problem on a vector of game features to dynamically adjust the game to optimize session level retention [11] or improve game design [12]. Empirical distributions of various engagement metrics were studied in [13] and short term engagement metrics were found to correlate to other short term but less so to long term engagement metrics. In contrast to game measurements, life circumstances and personal motivations have also been investigated as a predictor of engagement [14-15].

Tracking user retention is both important and difficult. Because majority users in mobile games exhibit very short lifetimes and the highly important but small engaged minority very long lifetimes, gathering enough data to make reliable deductions over long timespans is a challenge. Interest in long-term retention may require spending thousands of dollars at the acquisition cost of 1\$ per user to obtain a small cohort of long term users and then waiting an extended period of time to be able to calculate retention metrics. Furthermore, the data is based on an environment of incomplete knowledge since one cannot observe the event of user quitting, but merely lack of activity on the user's part. This makes deductions about the user base inherently approximative and uncertain.

To our knowledge, previous research has not investigated the user process as a whole. In contrast, popular retention metrics for example are based on the population of user processes over their lifetimes. Furthermore, previous research has not been conducted in the setting where the user process is still ongoing and one cannot assume last session was the final one. In data gathered from users with a long history the data may be processed with an assumption that the users have quit, which is analytically convenient but does not reflect reality in the industry. One can afford to wait months to gather data for research purposes, but in the industry one needs to know what effect changes had before starting the next development cycle.

## B. Retention Metrics

There is no generally settled upon terminology for different retention metrics. We use the terms retention, rolling retention and lifetime retention for the most popular metrics. They are defined as follows:

- X'th day retention is defined as the number of users who were playing X calendar days from the day they started, divided by the total number of users who started that day.
- X'th day rolling retention is the percentage of users who are active X calendar days from the day they started, where active user is defined as a user who is playing either on that day or any day after it.
- X'th day lifetime retention is the percentage of users who played X or more days in total given the day they started.

Each retention metric is defined for a cohort of users starting the same day and consists of one measure for each day. For example, if 1000 users start on Monday and of those 143 users are playing on Sunday, 6-day retention is then 14.3%. If we look at data available for the entire month to see that 267 users were playing at or after that Sunday, rolling retention for that day is 26.7%. Retention is commonly used in iterative development cycles to evaluate the product quality and identify weak points in game content. Rolling retention aims to quantify the number of users who have not quit the game, but it has the weakness that values inherently depend on the number of days we can observe up to the current day. Some players will return in the future but they are interpreted as having quit in the calculation of this metric. Same problem applies to lifetime retention.

## C. Research Goal

This motivates mathematical modelling of user engagement with three-fold aims:

1. Understanding user engagement through the underlying model components and parameters.
2. Accuracy and extrapolation of metrics based on few samples and limited observation window.
3. Robust comparison of game versions, user cohorts within the game and variables affecting engagement.

We present a parametric model which is able to describe and predict user activity data binned to day accuracy at the level of a single user, incorporating ambiguity inherent in user disengagement. Having been fitted to observed data, this model implies long-term player behavior and can be used in analytical and predictive modelling of population level retention metrics.

The model developed is to our knowledge novel in the gaming industry and was found to categorize certain stochastic processes and be closely related to models used in survival analysis with a peculiar censoring setup. To foster collaboration we utilize the problem setting from that field. To evaluate the applicability of the model, we consider five mobile games with both millions of players and in-development games using periodic user acquisition tests.

## II. DATA SET

### A. Mobile Games

We consider the user data produced by the following five mobile games developed by Tribeflame, Ltd.:

TABLE I. MOBILE GAMES USED IN THE STUDY

Game	Description	Genre
Benji Bananas	"Exciting and fun physics based adventure!"	Endless Runner
Benji Bananas Adventures	"Everyone's favorite swinging monkey returns for more action in the jungle."	Arcade Platformer
Mad-Croc	"Mad Croc is an endless runner where you control the fierce Mad Croc crocodile!"	Endless Runner
Hipster Maze <sup>a</sup>	"Help this unique sheep in his quest of finding the next big thing!"	Brain Puzzle
Dragon Fortress <sup>a</sup>	"Build your epic fortress and make it invincible!"	Combat City Builder

<sup>a</sup> In Development

The last two titles are under development whereas Mad-Croc was launched in December 2015. Both games in the Benji franchise are well-established; the sequel Benji Bananas Adventures launched May 2014 and the original Benji Bananas in 2013 varying by platform. We do not have exact data on every single unique player over all versions and days for the Apple App Store and Window store; the number of players is estimated at the time of collection 19.2.2015 as follows:

TABLE II. NUMBER OF PLAYERS

Game	Google Play	Apple Store	Windows Store	Total
Benji Bananas	53 000 000	8 000 000	1 100 000	62 000 000
Benji Bananas Adventures	9 000 000	2 000 000	200 000	11 000 000
Mad-Croc	100 000	40 000		140 000
Hipster Maze <sup>a</sup>	3600			3 600
Dragon Fortress <sup>a</sup>	11 000			11 000

### B. Data Set and Limitations

The data we use is based on Tribeflame's in-house logging framework which sends custom events at pre-defined actions in the game. It stores these events locally as they occur and sends them in batches. However, the data suffers from various issues which can lead to systematic biases as abnormally long user processes or idling periods:

1. The user id is not unique: since user device generates the user id two ids may collide. In later versions long user id leads to this being moderately rare.
2. Events may not be sent: if user device is offline long periods the local message buffer may overflow, leading to missing events in-between. This again is quite rare.
3. Final events are not sent: when user quits the game for good through 'home'-button, the logging framework has no chance to send accumulated events. This censors the ending time of the very last session.

Because of the non-unique id problem caused by short id in early versions combined with immense popularity of Benji Bananas and a hard drive failure affecting Benji Bananas Adventures, their data is constrained to begin at 2015-09-01. Mad-Croc data begins at release 2015-12-09, Hipster Maze in-development data since 2015-06 and Dragon Fortress in-development data since 2015-09. The data for Dragon Fortress is recorded up to 2015-12-30 and the others to 2016-02-19. Data is filtered to exclude the users (~3-7%) where it can be verified first or second issue occurred. These problems can be mitigated by changes to the logging framework, and are irrelevant to the applicability of the model.

In our study, we focus on two events sent by the framework: session\_start and session\_end. The session\_start message is sent when user brings the game to foreground by starting the game or resuming the running game. The session\_end message is sent when the game is put to background by pressing the 'home'-button. Both events contain the running session number and timestamp. Our data in stacked format consists of the following tuples:

user id	session number	timestamp start	timestamp end
gp_0e1d32bd-c388-47d5-a093-bc143c334ea3			
gp_0e68ab5c-4962-41d2-a070-ce7a40cc8076			
gp_0e9c0489-4ca7-4890-b4ba-8a5456c8dda0			
gp_0ea61525-0309-4430-b823-0207d2c718e7			
gp_0f31f62a-4236-4d0d-8fa1-a9ed1933d150			

Fig. 1. Continuous timeline

This data is visualized in Fig. 1. as a timeline for five players, consisting of intervals of activity and inactivity with vertical ticks denoting day limits in the picture.

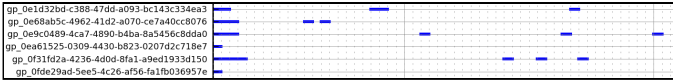


Fig. 2. Discrete timeline

Because we are interested in retention, we may bin the data to days as in Fig. 2. by setting each day active or inactive based on whether user played that day. In the picture each day is denoted by an interval of activity between midnights when there was a session within that day. Dashed vertical ticks denote day limits and solid vertical ticks months. Retention is then computed by summing vertically over all players to obtain number of players playing each day or horizontally to obtain total lifetimes.

Note that the data set is right-censored. Given a cohort of users who started playing on the same day  $0$  we have data from days  $\{0, 1, 2, \dots, N\}$  only up to the current day  $N$ . For each subject we have only  $N+1$  days of data for a given observation window. Indexing starts at  $0$  for the convenience of speaking of  $N$ 'th day retention, which is 100% for day  $0$  by definition. Since retention is based on number of days passed, it is also possible to combine users from different days into a single cohort. One use case is pooling together users from a single acquisition spanning one week. In this case the number of days observed up to the censoring limit  $N_i$  varies for each user. A practical model needs to be able to robustly deal with censoring

where users continue playing after  $N+1$  or  $N_i+1$  days without having data to describe the behavior thereon.

### C. Data Set Modelling

Intuitive understanding of the user process is assisted by considering a theoretical player who probabilistically produces the observed data by being in one of three states: playing, idling and gone. The continuous process is fully specified, given number of sessions  $n$ , by an alternating sequence of durations  $P_i$  in playing state and durations  $I_{i+1}$  in idling state with a sequence  $r=(I_1, P_1, I_2, P_2, \dots, P_n)$ :



Fig. 3. Continuous state transition process

Since the continuous model in Fig. 3. has additional components without implications to retention, such as number of sessions on a given day or the length of a session, we discretize the data functionally performing a type of filtering. The process  $s \in 2^N$  is specified by a sequence of 1s or 0s corresponding to playing and not playing. In the playing state, the player is active and engaged with the game ( $s_i=1$ ) whereas in the idling state the player is active but is not playing the game ( $s_i=0$ ) at that time. In the gone state, the player has quit the game for good and will never return ( $s_{\geq i}=00\dots$ ):

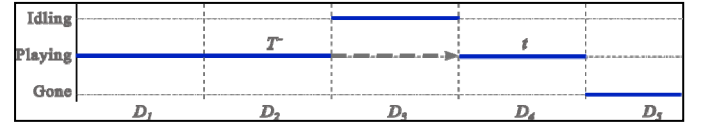


Fig. 4. Discrete state transition process

Notice that while there are three states in Fig. 4, in reality the state producing the final interval is latent variable since both gone and idling produce sequence of 0s. This is demonstrated in the conceptual diagram in Fig. 5. One then wishes to find a probabilistic model that correctly specifies a probability  $P$  for every discrete activity sequence  $s$  in the sample space  $2^N$  and which can be robustly fit to data because this would imply aggregate statistics.

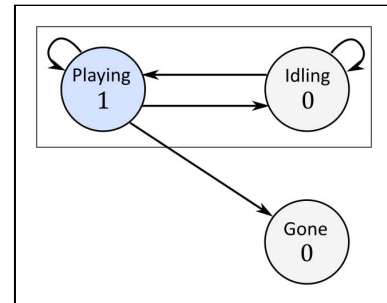


Fig. 5. Conceptual state transition diagram with a latent state

In the continuous case, a natural approach is to model the distribution of duration each state lasts, or the survival of the state until time  $t$  when a transition occurs. The duration of each session, here duration  $P_i$  of the playing state, and the total time played, i.e. total time  $\sum_i P_i$  in playing state, is known follow Weibull distribution [2-6]. We verified this to be the case, and in our data even the session inter arrival times, here duration  $I_i$  of the idling state, follow Weibull distribution in the aggregate for most of their domain. However, we found that there is a strong and complicated dependence of idling times on previous player history and time. Based on preliminary experiments, a simple process generating independent and identically distributed (i.i.d.) Weibull idling durations  $I_i$  alternating with i.i.d. Weibull session durations  $P_i$  does not describe the player process. This is because modelling underlying dependence is needed for such process to aggregate to correct statistics.

The observation that duration of each state and total time in playing state is Weibull distributed is reasonable, since time in a state is akin to survival and Exponential and Weibull distributions are known to be the most widely applicable distributions in survival analysis [16]. User playing a mobile game over the length his or her engagement could be expected to diverge from a regular process, so complete independence of idles would have been remarkable.

The discrete process measures essentially the same distributions on a coarser scale with fewer dependence structures present. We investigated the assumption that total number of days played is generated by a process similar to total time played since they are quite closely related. Furthermore, one may assume each user goes to idling state by sleeping at midnight by the latest and then idles thereon to play at some time in the future. This again would resemble the inter session idling process and be approximately Weibull underneath. We formed the discrete model by specifying the following two component distributions where  $H(t)$  denotes the player history up to the previous day played at time  $t$ :

- $P(\text{Days Played} = k \mid H(t))$
- $P(\text{Days to Next} = k \mid H(t))$

First distribution specifies the total number of days played over  $k=1,2,\dots$  and the second the number of days to next day played over  $k=1,2,\dots$ . In general these distributions are not i.i.d., but may have a complex dependence on previous history  $H(t)$ . With these components it is possible to give a generative model specifying the probability of any user activity sequence up to any future date. To model the data, next challenge is then two-fold: specify distributional form for these components and specify the simplest dependence structure possible. While the continuous process is complicated, the discrete formulation is very close to the kind of data simple processes in survival analysis would discretize into [17]. We therefore utilize a survival analysis formulation, popular distributions therein and widely used dependence structures in recurrent event models to investigate their suitability for user engagement in gaming data.

#### A. Time to Event

Assume we have a data set with subjects, variables associated to the subjects and one or more events for each subject. Survival analysis is concerned with modelling the time to event(s) and quantifying the effect of variables to this time [18]. Usually the time is measured as a continuous variable, and it may be censored if the event is not included in the observation window. We present the formulation given in [16].

Denoting time to event random variable  $T$  with probability density function  $f(t)$  (PDF) and cumulative density function  $F(t)$  (CDF), the survival function  $S(t)$  (SF) measures the probability of an event occurring after time  $t$ :

$$S(t) = P[T > t] = 1 - F(t) \quad (2)$$

The hazard function  $h(t)$  measures the instantaneous event rate at time  $t$  given that the event has not occurred yet. It is formally defined:

$$h(t) = \lim_{\Delta t \rightarrow 0} \frac{P[t < T \leq t + \Delta t \mid T > t]}{\Delta t} \quad (3)$$

The cumulative hazard  $\Lambda(t)$  describes the accumulated risk up to time  $t$ , defined as the integral of the hazard function:

$$\Lambda(t) = \int_0^t h(t) dt \quad (4)$$

It is straightforward to derive the following results which relate these functions and are often used in formulation of survival models:

$$h(t) = \frac{f(t)}{S(t)} \text{ and } S(t) = \exp(-\Lambda(t)) \quad (5)$$

Since we are dealing with censored data where event times occurring after  $W$  are not observed, instead of event time  $T_i$  for subject we observe  $T_i = \min(T_i, W)$  with a censoring indicator  $C_i = I(T_i > W)$ .

#### B. Recurrent Events and Time Dependence

In general,  $i$ 'th subject may experience multiple events  $T_{i1}, \dots, T_{in(i)}$ . Survival Analysis approaches to such data have been developed relatively recently [17]. There are three popular approaches to specifying the hazard depending on which time interval the subject is considered to be at risk for events over: calendar time, gap time and marginal time [19-20].

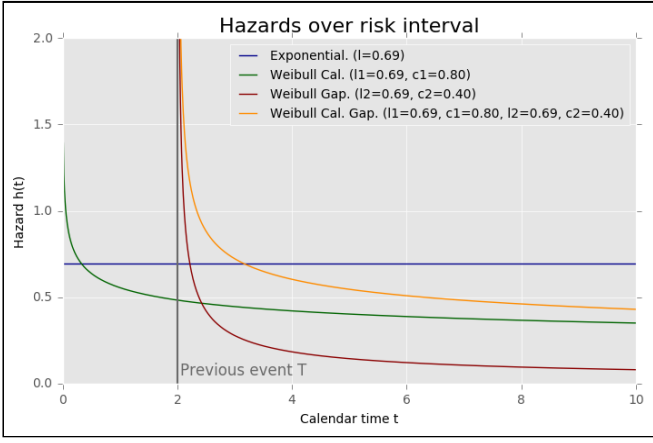


Fig. 6. Example of Weibull hazard over different risk intervals

In calendar time interval at risk is defined over calendar time passed. If user starts at calendar time  $0$  and hazard is defined as a function of calendar time  $t$ , the cumulative hazard up to first event  $t_1$  is  $\Lambda(t_1)$ . The cumulative hazard to next event  $t_2$  given first event  $t_1$  is obtained by integrating the hazard over the corresponding calendar time at risk:  $\Lambda(t_2, t_1) = \Lambda(t_2) - \Lambda(t_1)$ . Event occurrence process is independent of previous event history is known as a Poisson process. If such process is independent of time as well it is homogeneous, otherwise inhomogeneous.

In gap time the interval at risk is defined as time passed since previous event. Denoting calendar time  $t$  and time of last event  $T$  with gap time hazard  $h'(w)$  defined as a function of time passed  $w$ , the cumulative hazard is  $\Lambda(t-T)$ . This formulation is also known as a renewal process, named accordingly since an event in a sense resets the subject making history preceding the renewing event irrelevant to further events.

Marginal time approach is the simplest, since it assumes events are independent and the subject is at risk for each event since the beginning. Common hazard function for each event makes them identically distributed, which is clearly an oversimplification in this case. This approach is most suited for limited number of events where each event can be considered a separate process and their order irrelevant.

The calendar time and gap time models can be combined so that hazard depends on both the calendar time and gap time [21]. Assuming additive hazards we specify the effect of calendar time  $h_1$  and gap time  $h_2$  hazards to the total hazard as  $h(t|T) = h_1(t) + h_2(t-T)$ . These risk formulations are visualized in Fig.6. In general the hazard is constrained to be positive and may arbitrarily depend on event history.

### C. Subject Dependence

It is common for time to event measurements to exhibit greater variation than predicted by a theoretical model [18]. One source of variation is a subject specific propensity to events, commonly called frailty in the survival analysis literature. This effect is important in recurrent event models because this implies a dependence structure between events

[23] where within subject event times are correlated. Popular formulation of frailty assumes a subject specific latent variable  $u_i$  with multiplicative effect to hazard  $h_i(t) = u_i h(t)$  [18].

Since variable  $u_i$  is unknown, it is assumed to be distributed according to a population level prior distribution of unknown hyperparameters. Satisfactory results are often achieved using the Gamma distribution [22] with unknown variance  $\theta$  and mean equal to one when the base hazard  $h(t)$  includes a constant. This makes the hazard a random variable with  $u_i \sim \text{Gamma}(1/\theta, \theta)$ .

### D. Time to Event Distributions

In survival analysis time to event is most commonly evaluated for fitting the exponential distribution or Weibull distribution [18]. Exponential distribution is defined simply by assuming a constant hazard  $h(t) = \lambda$ , which implies the following:

$$\Lambda(t) = \lambda t \text{ and } S(t) = \exp(-\lambda t) \quad (8)$$

The Weibull distribution specifies a hazard  $h(t) = \lambda c t^{c-1}$  and thus generalizes the Exponential distribution ( $c=1$ ), allowing for both decreasing hazard ( $c < 1$ ) and increasing hazard ( $c > 1$ ):

$$\Lambda(t) = \lambda t^c \text{ and } S(t) = \exp(-\lambda t^c) \quad (9)$$

In modelling discrete phenomena distributions with discrete support  $k=1, 2, \dots$  should be used. In binned data one may hypothesize that the underlying phenomena is continuous and aggregates to densities predicted by the Exponential and the Weibull distributions. In this case the CDFs are binned versions of their continuous counterparts. Such motivation leads to the Geometric and Discrete Weibull distributions [24], which are defined such that their CDF is equal to the corresponding continuous CDF at the discrete support as in Fig.7. Discrete probability mass function (PMF) can be derived from the CDF  $F(x)$  through a simple relation  $P(X=k) = F(k) - F(k-1) = S(k-1) - S(k)$ .

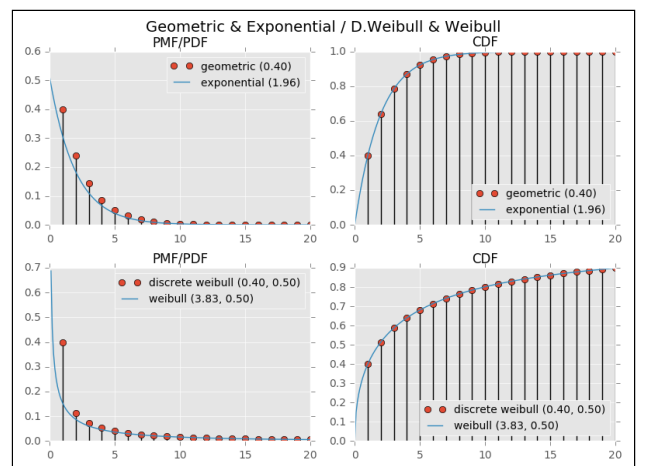


Fig. 7. Geometric and Discrete Weibull distributions



The Geometric distribution, informally Discrete Exponential, is defined:

$$h(k) = p \text{ and } S(k) = (1 - p)^k \quad (10)$$

The Discrete Weibull distribution is defined:

$$h(k) = 1 - (1 - p)^{k^c - (k-1)^c} \text{ and } S(k) = (1 - p)^{k^c} \quad (11)$$

In these formulas  $h(k)$  denotes a discrete hazard. It is defined  $h(k) = f(k)/S(k-1)$  reminiscent of the continuous hazard, instead giving the proportion experiencing event at  $k$  of population surviving without event to  $k$ .

#### E. Hidden Quit

Unlike most survival analysis settings, a subject in our study may randomly quit without us having information the subject has done so. In settings where this information is available it is common to consider such subjects censored or incorporate a second event terminating the event process [18]. We acquire user data as long as the user is playing the game and stop acquiring data once he or she has quit, making termination of an event process in our case indistinguishable from continuation of the event process without events.

One solution is to assume that the probability of the event process remaining active and the user remaining at risk of events follows some law. For example, number of events may be assumed to follow a specified distribution and we then consider both possibilities. The lack of activity following last event is caused by being active and remaining at risk for an event without experiencing subsequent event before censoring or the event process may have become inactive and stops producing events. For subject not experiencing events for gap time  $t$  after the last event, tail probability is conceptually:

$$P[\text{Active}]P[\text{Event} > t \mid \text{Active}] + P[\text{Inactive}] \quad (12)$$

We found an analogous result from medical literature with cure models, an old concept [25] recently gaining recognition [26-28]. In cure models, a proportion of subjects are assumed to be ‘cured’, i.e. immune to events. Such immunity may be either a feature of the population or caused by an intervention in response to events or other features of the process. In our case if every player is observed over a long time span with a long period of inactivity the quit assumption is justified. As stated previously, this is not feasible in game analytics if we seek to make decisions to guide development cycles. Our model is able to fit extremely time-limited data by incorporating tail term to allow for ambiguity between quitting and idling.

#### F. Model Specification

Many survival models are relatively simple, often assuming Poisson or Renewal process and considering frailty as a possible complication with the calendar, gap or marginal time risk formulations [20-22]. We implicitly consider these models

through their discrete counterparts. Denote the calendar day  $t$  of next day played where  $T^-$  is the calendar day of previous day played and we have the following correspondence:

TABLE III. MODELS

Name	C. Survival	D. Survival
Exp.	$\exp(-\lambda_0(t - T^-))$	$p_0^{t-T^-}$
Weibull Cal.	$\exp(-\lambda_1(t^{c_1} - T^{-c_1}))$	$p_1^{t^{c_1} - T^{-c_1}}$
Weibull Gap.	$\exp(-\lambda_2(t - T^-)^{c_2})$	$p_2^{(t-T^-)^{c_2}}$
Weibull Cal.Gap.	$\exp\left(\begin{matrix} -\lambda_1(t^{c_1} - T^{-c_1}) \\ -\lambda_2(t - T^-)^{c_2} \end{matrix}\right)$	$p_1^{t^{c_1} - T^{-c_1}} p_2^{(t-T^-)^{c_2}}$

Since the exponential distribution has a constant hazard the process corresponds to a homogeneous Poisson process regardless of risk formulation [17]. Weibull hazard over calendar time is an inhomogeneous Poisson process, Weibull hazard over gap time is a Renewal process and calendar&gap time cannot be categorized as either. The distributions we fit to days to next day played are discretized Exponential, Gap Weibull, Calendar Weibull and Gap&Calendar Weibull with and without frailties. We fit the Weibull distribution to the number of days played to model surviving playing or quitting. We found that the effect of the cure model is non-negligible even when we are operating with a long observation window.

### IV. MODEL FIT AND VERIFICATION

#### A. Parameters and Likelihood

Denote  $N$  total players and for  $i$ 'th player: total days played  $n_i$ , days played  $t_{ij}$  and censoring time  $t_i$ . Given the PMF and SF of days to next day played and total days played parameterized in terms of  $\theta_1$  and  $\theta_2$  respectively, the distributions are fit by maximizing the likelihood:

$$L(\mathcal{D}|\theta_1, \theta_2) = \prod_{i=1}^m \left[ \prod_{j=1}^{n_i} \mathbb{P}_{\text{ToNext}}[t_{ij}|t_{i(j-1)}, \theta_1] \right] \{ \mathbb{P}_{\text{Days}}[n_i|\theta_2] + \mathbb{S}_{\text{Days}}[n_i|\theta_2] \mathbb{S}_{\text{ToNext}}[t_i|t_{in_i}, \theta_1] \} \quad (13)$$

Fitting the model produced following table of parameters and negative log-likelihood score (n.l.l.) for total days played *Weibull*( $k|p_d, c_d$ ) and days to next day played *Exponential*( $k|p_0$ ), *Weibull.Cal*( $k|p_1, c_1$ ), *Weibull.Gap*( $k|p_2, c_2$ ), *Weibull.Cal.Gap*( $k|p_1, c_1, p_2, c_2$ ):

TABLE IV. MODEL PARAMETERS AND LIKELIHOOD

Game	$p_d$	$c_d$	$p_0$	$p_1$	$c_1$	$p_2$	$c_2$	n.l.l.
<b>Benji Adventures</b>	0.55	0.59	0.22					48656
	0.55	0.58		0.78	0.53			43492
	0.55	0.59				0.57	0.44	41500
	0.55	0.59		0.55	0.40	0.45	0.39	40999
<b>Benji Bananas</b>	0.44	0.63	0.28					309189
	0.44	0.62		0.83	0.55			279141
	0.44	0.63				0.58	0.48	268894

	0.44	0.62		0.61	0.44	0.45	0.41	265026
Hipster Maze	0.42	0.64	0.24					8674
	0.42	0.62		0.74	0.58			8024
	0.42	0.63				0.54	0.48	7579
	0.42	0.62		0.45	0.44	0.45	0.43	7521
Mad-Croc	0.86	0.36	0.26					9484
	0.86	0.36		0.66	0.60			8981
	0.86	0.35				0.53	0.51	8695
	0.86	0.35		0.40	0.46	0.42	0.47	8646
Dragon Fortress	0.67	0.68	0.34					1368
	0.67	0.67		0.72	0.58			1282
	0.66	0.68				0.67	0.39	1201
	0.66	0.67		0.38	0.45	0.57	0.31	1196

There are limitations to interpreting model fits in terms of the likelihood. Since likelihood is decreasing for increasing amount of data regardless of goodness-of-fit, these models can only be compared within a given dataset. Smaller n.l.1 implies increasing likelihood and better distributional fits. Results are aligned with expectations: Since the calendar and gap time Weibull distributions generalize the Exponential distribution and the calendar&gap time Weibull generalizes both, the likelihood is increasing according to this hierarchy. We see that the gap time formulation has consistently larger likelihood than the calendar time formulation.

The frailty model which included a gamma frailty term in all cases produced a small variance  $\theta \approx 0.01$ . This translated to almost no effect in aggregate statistics so we refrain from reproducing the table with it. This would imply a random variable  $p_0/p_1/p_2$  resembling a normal distribution with 3 s.d. region bounded by  $\pm 0.1$ .

### B. Model Interpretation

One way to interpret the model is to plot discrete hazards which gives the proportion of users remaining without events at  $t$  experiencing event at  $t$ . We used Hipster Maze whose parameters have similarities with the Benji franchise. Perhaps the component of most game development interest is quitting as a function of days played  $Weibull(k|p_d|c_d)$ . The hazard over the number of days played demonstrates churning of the remaining population as a function of total days played.

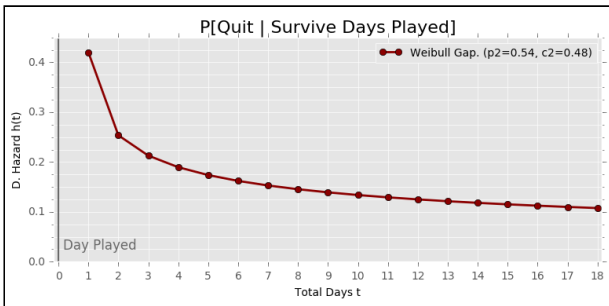


Fig. 8. Discrete hazard of Weibull distributed days played

In Fig. 8, we see that a large 43% fraction of players quit after the first day played and the next few days have an

elevated risk of  $\sim 10\%$  decreasing slowly towards zero. One could claim that the quit rate stabilizes to 10% after churning out uninterested users with a slight tendency to commit users.

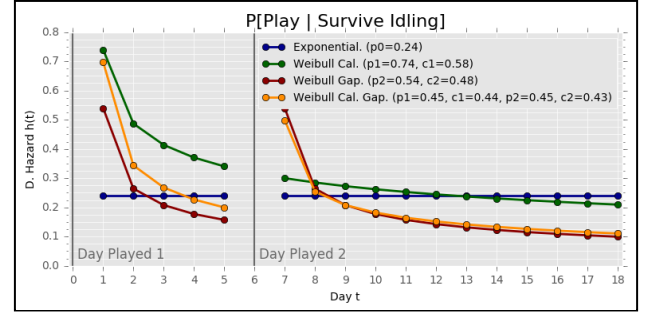


Fig. 9. Discrete hazard of Weibull distributed days to next day played

In Fig. 9, we observe that in exponential model 24% of idlers re-engage, whereas in Weibull Cal. players idle little when the game is novel and quickly become habituated to an approximately 20% play rate. The gap time denoted in red is impervious to calendar time passed and has a high re-engagement rate in the day following a day played, slowly decreasing. One can say playing state is self-exciting. The calendar&gap time formulation combines novelty and self-excitative effects to produce initially short idles and thereon resembles the gap time model.

### C. Visual Verification

Retention statistics can be computed from this model by generating sequences according to the fitted probabilities. Note that the model has not been fitted to any particular aggregate statistic, and the result in Fig. 10. is faithful only to the extent to which the underlying process is being modelled.

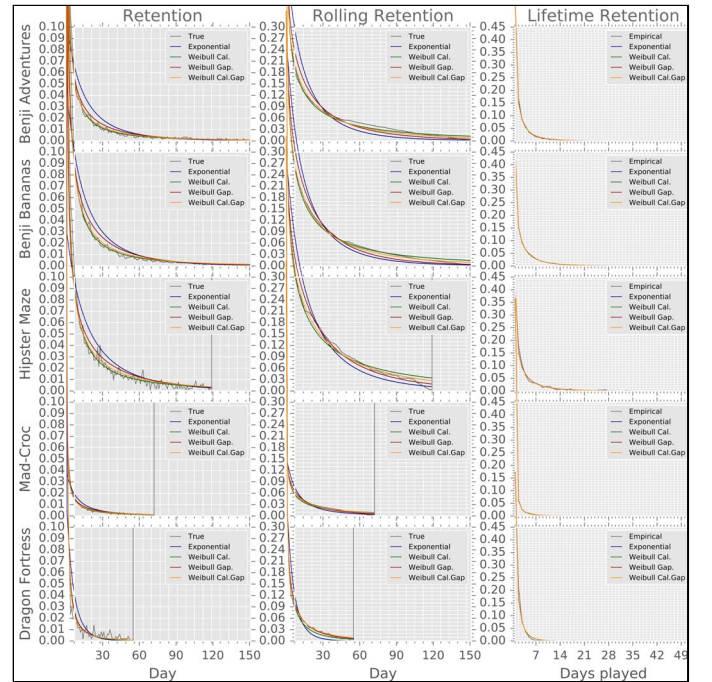




Fig. 10. True and model fits of retention metrics

Due to a rapid initial decrease obfuscating the tail, we plot retention and rolling retention in Fig. 10. from day 7 onwards where they are better visible. We see that Weibull Cal. and Weibull Cal.Gap. fit retention in this region in an unbiased way up to the maximum observation limit of 150 days. Smaller observation limits are denoted with a vertical dash. Days 2-6 excluded from these plots have a small systematic 1-3% absolute percentage point pessimistic bias relative to 10-50% absolute values.

Indeed it is quite difficult to distinguish the smooth shape of the rolling retention from the Weibull Cal. and Weibull Cal. Gap fits. Upon closer inspection Weibull Cal.Gap is slightly better, suggesting with likelihood that gap time does model an aspect in addition to the novelty effect. The Exponential model has an unacceptably large bias which Weibull Gap. shares with roughly half the magnitude suggesting data is not modelled by a homogeneous Poisson process and neither appreciably so by a simple renewal process. Lifetime retention in Fig.10. has very good fits for every model, confirming that total days played is Discrete Weibull distributed.

#### D. Conclusion and Future Work

We demonstrated that discretization of the dataset and discretized versions of popular survival analysis methods lead to a problem formulation which makes modelling user engagement possible. The entire user process at the individual level is construed as a simple generative model having two components: 2 parameter D. Weibull distribution for total days played and 1-3 parameter D. Weibull distribution for days to next day played with an optional 1 parameter frailty term. This model reveals interesting connections to survival analysis and certain classes of stochastic processes. Retention, rolling retention and lifetime retention computed from the model suggest that the gap&calendar and to an extent the calendar time formulation accurately describe the underlying phenomena in mobile games. In addition to insight obtained it is possible to apply this model to analytical and predictive modelling of any day activity based aggregate statistics such as retention.

#### ACKNOWLEDGMENT

Tribeflame, Ltd. generously participated in the funding of this research, granted access to wonderful games data and indulged our academic predilection.

#### REFERENCES

- [1] E. Seufert, "Freemium Economics", Morgan Kaufmann, 2014.
- [2] R. Sifa, C. Bauckhage and A. Drachen, "The Playtime Principle: Large-scale cross-games interest modeling," *Proc. IEEE CIG*, 2014.
- [3] C. Bauckhage, K. Kersting, R. Sifa, C. Thureau, A. Drachen, and A. Canossa, "How Players Lose Interest in Playing a Game: An Empirical Study Based on Distributions of Total Playing Times," *Proc. IEEE CIG*, 2012.
- [4] D. Pittman and C. Chris GauthierDickey, "Characterizing Virtual Populations in Massively Multiplayer Online Role-Playing Games" *Advances in Multimedia Modeling Volume 5916 of the series Lecture Notes in Computer Science*, 2010.
- [5] C. Chambers, W. Feng, S. Sahu and D. Saha "Measurement-based characterization of a collection of on-line games" *IMC*, 2005.
- [6] W. Feng, "A Long-term Study of a Popular MMORPG", *NetGames 2007*, 2007.
- [7] T. Henderson, S. Bhatti: Modelling user behaviour in networked games, *Multimedia 2001*, 2001.
- [8] F. Hadiji, R. Sifa, A. Drachen, C. Thureau, K. Kersting and C. Bauckhage, "Predicting player churn in the wild," *Proc. IEEE CIG*, 2014.
- [9] P. Tarnag, K. Chen and P. Huang, "On prophesying online gamer departure" in *Proc. 8th Ann. Workshop Netw. Syst. Support Games*, 2009.
- [10] H. Xie, S. Devlin, D. Kudenko and P. Cowling, "Predicting player disengagement and first purchase with event-frequency based data representation," *Proc. IEEE CIG*, 2015.
- [11] B. Harrison and D. Roberts, "An Analytic and Psychometric Evaluation of Dynamic Game Adaption for Increasing Session-Level Retention in Casual Games," in *Proc. IEEE CIG*, 2015.
- [12] B. Weber, M. John, M. Mateas and A. Jhala, "Modeling player retention in Madden NFL 11," in *Proc. 23rd IAAI Conf.*, 2011.
- [13] P. Tarnag, K. Chen and P. Huang, "An analysis of wow players' game hours" in *Proc. 7th ACM SIGCOMM Workshop Netw. Syst. Support Games*, 2008.
- [14] T. Debeauvais, C. V. Lopes, N. Yee and N. Ducheneaut, "Retention and progression: Seven months in World of Warcraft," in *Proc. 9th Int. Conf. Found. Digit. Games*, 2014.
- [15] Z. Lin, C. Lewis, S. Kurniawan and J. Whitehead, "Why players start and stop playing a Chinese social network game," *J. Gam. Virtual Worlds*, vol. 5, no. 3, 2013.
- [16] F. Harrell. Regression Modeling Strategies: with Applications to Linear Models, Logistic Regression, and Survival Analysis. Springer, 2001.
- [17] J. Cook and J. Lawless. The Statistical Analysis of Recurrent Events. Springer, 2001.
- [18] D. Kleinbaum and M. Klein. Survival analysis: a self-learning text, 3rd edn. Springer, 2012.
- [19] P. Kelly, L. Lim. Survival analysis for recurrent event data: an application to childhood infectious diseases. *Stat Med.* 19(1):13-33. 2000.
- [20] S. Ullah, T. Gabbett, C. Finch. Statistical modelling for recurrent events: an application to sports injuries. *Br J Sports Med* 2012.
- [21] L. Duchateau, P. Janssen, I. Kezic and C. Fortpiet. Evolution of Recurrent Asthma Event Rate over Time in Frailty Models. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 52(3), 55-363, 2003.
- [22] J. Cook, E. Ng, J. Mukherjee and D. Vaughan, Two-state mixed renewal processes for chronic disease. *Statist. Med.*, 18: 175-188, 1999.
- [23] J. Box-Steffensmeier and S. De Boef. Repeated events survival models: the conditional frailty model. *Statist. Med.*, 25: 3518-3533, 2006.
- [24] H. Rinne. The Weibull distribution: a handbook. CRC Press, 2008.
- [25] J. Boag. Maximum likelihood estimates of the proportion of patients cured by cancer therapy. *J. R. Stat. Soc.*, 11: 15-53. 1949.
- [26] P. Lambert, J. Thompson, C. Weston, P. Dickman. Estimating and modeling the cure fraction in population-based cancer survival analysis. *Biostatistics*, 8:576-94, 2007.
- [27] V. Rondeau, E. Schaffner, F. Corbiere, J.R. Gonzalez and S. Mathoulin-Pelissier. Cure frailty models for survival data: application to recurrences for breast cancer and to hospital readmissions for colorectal cancer. *Stat Methods Med Res*, 22, pp. 243-260, 2013.
- [28] F. Louzada and J. Cobre, A multiple time scale survival model with a cure fraction, *Test* 21, pp. 355-368, 2012.

# Informed Monte Carlo Tree Search for Real-Time Strategy Games

Santiago Ontañón

Computer Science Department, Drexel University

Philadelphia, PA, USA 19104

Email: santi@cs.drexel.edu

**Abstract**—The recent success of *AlphaGO* has shown that it is possible to combine machine learning with Monte Carlo Tree Search (MCTS) in order to improve performance in games with large branching factors. This paper explores the question of whether similar ideas can be applied to a genre of games with an even larger branching factor: *Real-Time Strategy* games. Specifically, this paper studies (1) the use of Bayesian models to estimate the probability distribution of actions played by a strong player, (2) the incorporation of such models into NaiveMCTS, a MCTS algorithm designed for games with combinatorial branching factors. We call this approach *informed MCTS*, since it exploits prior information about the game in the form of a probability distribution of actions. We evaluate its performance in the  $\mu$ RTS game simulator, significantly outperforming the previous state of the art.

## I. INTRODUCTION

The recent success of *AlphaGO* [1] has shown that it is possible to combine machine learning with Monte Carlo Tree Search (MCTS) [2] in order to improve performance in games with large branching factors. This paper explores the question of whether similar ideas can be applied to games with an even larger branching factor: *Real-Time Strategy* (RTS) games.

RTS games are a videogame genre where players command large armies in real time in order to defeat the other players. RTS games pose a significant challenge to artificial intelligence due to two main reasons: they have huge branching factors, and they are real-time, leaving very little time for players to decide which actions to play [3], [4]. Several approaches have been proposed to handle these games, such as MCTS for games with combinatorial branching factors [5], [6], portfolio approaches [7], [8], abstraction (simplify the game state and search in a simplified space) [9], [10], [11], hierarchical search [12], adversarial HTN planning [13], or case-based reasoning [14].

In this paper, we build upon work on MCTS approaches for RTS games, and study how to bring ideas from AlphaGO into this game genre. Specifically, we present:

- A collection of Bayesian models to estimate the probability distribution of actions played by a strong player, evaluating them by attempting to model a collection of bots in the context of an RTS game. These models are analogous to the *policy network* in AlphaGO.
- We incorporate these models into NaiveMCTS, a MCTS algorithm designed for games with combinatorial branching factors to inform the search process, resulting in what

we call an *informed MCTS* approach. Our hypothesis is that the Bayesian models proposed in this paper (simpler than the deep neural networks used in AlphaGO) are enough to significantly improve MCTS.

We use the  $\mu$ RTS game simulator as our evaluation domain<sup>1</sup>, and report results significantly outperforming the previous state of the art, LSI [5] and NaiveMCTS [6].  $\mu$ RTS is deterministic, fully-observable and a forward model is available (necessary to implement game-tree search approaches).

The remainder of this paper is organized as follows. Section II introduces RTS games from an AI point of view. Section III presents our Bayesian probability distribution estimation models. Section IV describes how can they be incorporated into MCTS approaches, and finally Section V presents an empirical evaluation of the proposed approach.

## II. REAL-TIME STRATEGY GAMES

Real-time Strategy (RTS) games are complex adversarial domains, typically simulating battles between a large number of military units, that pose a significant challenge to both human and artificial intelligence [3]. Designing AI techniques for RTS games is challenging because:

- They have *huge decision spaces*: the branching factor of a typical RTS game, StarCraft, has been estimated to be on the order of  $10^{50}$  or higher [4] (for comparison, that of Chess is about 35, and that of Go about 180).
- They are *real-time*, which means that: 1) RTS games typically execute at 10 to 50 decision cycles per second, leaving players with just a fraction of a second to decide the next action, 2) players can issue actions simultaneously, and 3) actions are durative.

The reason for which the branching factor in RTS games is so large is that players controls many units, and players can issue multiple actions at the same time (one per unit). We will refer to those actions as *unit-actions*. A *player-action* is the set of unit-actions that one player issues simultaneously in a given game cycle. Thus players issue only one *player-action* at any given time (which will consist of zero or more unit-actions).

To illustrate this, consider the situation from the  $\mu$ RTS game shown in Figure 1. Two players, *max* (shown in blue)

<sup>1</sup>A fork of the  $\mu$ RTS project containing all the source code and datasets necessary to replicate the results reported in this paper can be downloaded from <https://sites.google.com/site/santiagoontanonvillar/code/CIG-2016-microRTS-source-code.zip>.

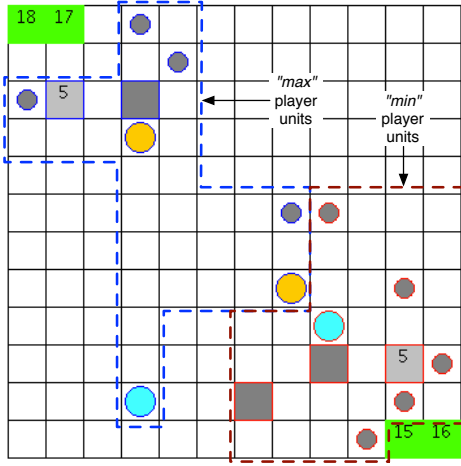


Fig. 1. A screenshot of the  $\mu$ RTS simulator. Square units correspond to “bases” (light grey, that can produce workers), “barracks” (dark grey, that can produce military units), and “resources mines” (green, from where workers can extract resources to produce more units), the circular units correspond to workers (small, dark grey) and military units (large, yellow or light blue).

and *min* (shown in red) control 9 units each. Consider the bottom-most circular unit in Figure 1 (a worker). This unit can execute 8 actions: stand still, move left or up, harvest the resource mine to the right, or build a barracks or a base in any of the two adjacent cells. In total, player *max* can issue 1,008,288 different player-actions, and player *min* can issue 1,680,550 different player-actions. Thus, even in relatively simple scenarios, the branching factor is very large.

In the remainder of this paper, we will use the following definition for an RTS game. An RTS game is a tuple  $G = (P, S, A, L_u, L_p, T, W, s_{init})$ , where:

- $P = \{max, min\}$  is the set of players.
- $S$  is the set of possible states. We will write  $units(p, s)$  as the set of units that belong to player  $p$  in state  $s$ .
- $A$  is the finite set of unit-actions that units can execute.
- $L_u(u, a, s) \rightarrow \{true, false\}$ , is a function that returns whether unit  $u$  can execute unit-action  $a$  in state  $s$ . For simplicity, we will write  $L_u(u, s) = \{a \in A | L_u(u, a, s) = true\}$ , and  $ready(p, s) = \{u \in units(p, s) | L_u(u, s) \neq \emptyset\}$ .
- $L_p(p, \alpha, s) \rightarrow \{true, false\}$ , is a function that returns whether player  $p$  can execute player-action  $\alpha$  in state  $s$ . Given the set of ready units  $ready(p, s) = \{u_1, \dots, u_n\}$ , a player-action  $\alpha$  is defined as  $alpha = \{(u_1, a_1), \dots, (u_n, a_n)\}$ , such that  $L_u(u_i, a_i, p) = true$  for  $1 \leq i \leq n$ . Thus, the *ready* function determines the set of units that can execute unit-actions,  $L_u$  determines which actions can each of those units execute, which determines the set of possible player-actions, and  $L_p$  determines which of those possible player-actions is legal.
- $T(s_t, \alpha_{min}, \alpha_{max}) \rightarrow S$  is the deterministic transition function, that given a state  $s_t \in S$  at time  $t$ , and the player-actions of each player ( $\alpha_{min}$  and  $\alpha_{max}$ ), returns the state that will be reached at time  $t + 1$  (i.e.,  $T$  is the

forward model of the game).

- $W : S \rightarrow \{maxwins, minwins, draw, ongoing\}$  is a function that determines the winner of the game, if the game is still ongoing, or if it is a draw.
- $s_{init} \in S$  is the initial state.

### III. UNIT-ACTION PROBABILITY ESTIMATION

*AlphaGO* used a deep neural network (DNN) to estimate the probability of each of the legal actions in a game state to be played by an expert player (the *policy network*). This network was trained from a large collection of expert games available on-line. A working hypotheses of this paper is that a simpler machine learning model than a DNN suffices to significantly improve the performance of MCTS. To assess such hypothesis, we propose simpler Bayesian models, based on the idea of the Naive Bayes classifier [15], and thus, with negligible training time. We present the proposed probability estimation models and how to generate training data below.

#### A. Bayesian Models

Given a game state  $s$ , a player  $p$ , and a unit  $u \in ready(p, s)$ , we would like to model the probability  $P(a|s, u)$  with which an expert would select each of the actions  $a \in L_u(u, s)$ .

In order to model such probability distribution, we assume that the game state  $s$  (from the perspective of  $u$ ) is represented by means of a feature vector  $\mathbf{x}(u, s) = \{x_1(u, s), \dots, x_n(u, s)\}$ , and that the distribution is estimated from a training set  $I = \{(s, u, a), \dots\}$ , where each training instance has a game state  $s$ , a unit  $u$ , and the action  $a$  that was chosen. The feature set used in our experiments is described in Section III-C. We experimented with two different models:

- **Calibrated Naive Bayes Model (CNB)**: while the Naive Bayes classifier [15] often works very well for classification purposes, it is well known that the probability distribution it estimates is not well “calibrated” [16], i.e., values tend to be very extremely close to either 0 or 1. Several approaches exist to better calibrate the posterior probability estimates of Naive Bayes classifiers, such as fitting them via a sigmoid function [16], or binning [17]. We propose a much simpler method, which achieved good results in our experiments: introducing a calibration parameter  $\kappa > 0$  into the standard Naive Bayes formulation, as follows:

$$P(a|u, s) = \frac{1}{Z} \left( P(a) \prod_{i=1 \dots n} P(x_i(u, s)|a) \right)^{\frac{1}{1+\kappa n}}$$

where  $n$  is the number of features used to represent the game state,  $P(x_i(u, s)|a)$  and  $P(a)$  are estimated from the training set<sup>2</sup>,  $Z$  is just a normalization constant to make all the probabilities add up to 1, and  $\kappa$  is a calibration parameter, whose effect is to make the probability

<sup>2</sup>All probability estimations from the training set were estimated using Laplace estimation. For example, when estimating  $P(a)$ , we add 1 to the numerator, and  $|A|$  to the denominator, resulting in  $P(a) = \frac{\text{number of times } a \text{ is selected} + 1}{\text{size of the training set} + |A|}$ .

values less extreme. Thus, notice the **CNB** model is identical to the standard Naive Bayes formulation, except for the addition of the  $\frac{1}{1+\kappa n}$  calibration exponent.

In our experiments,  $\kappa$  is determined via simple grid search using the training set, testing values between 0.0 to 1.0 at intervals of 0.05, and keeping the value that maximizes the likelihood of the training data given the model. Intuitively, when  $\kappa = 0$ , no correction is applied to the probabilities, and when  $\kappa$  grows, the probabilities grow less extreme. In the limit, when  $\kappa \rightarrow \infty$ , all probabilities would converge to the same number, making  $P(a|u, s)$  the uniform distribution. In the particular case when  $\kappa = 1$ , the model corresponds to a geometric mean of the different factors in the probability estimation.

- **Action-Type Interdependence Model (AIM):** the previous model does not consider that some actions might have a low or a high probability based on which other actions are legal. The AIM model captures the distribution given the actions that are legal in the current game state. Moreover, in order to reduce the number of parameters to estimate, we assume the existence of a function  $type(a)$ , which assigns a *type* to an action  $a$  from a predefined set of action types (e.g., move, attack, etc.). So, even if actions such as “move up” and “move down” are different actions, they both have the same type, “move”. Let us define  $legaltypes_u(u, s) = \{type(a) | a \in L_u(u, s)\}$  as the set of action types that unit  $u$  can perform in state  $s$ . The **AIM** model is defined as follows:

$$P(a|u, s) = \frac{1}{Z} (P(a) L(type(a), T) F(a, u, s))^c$$

where,  $c = \frac{1}{1+\kappa(n+|T|)}$ ,  $T = legaltypes_u(u, s)$ , and  $F(a, s)$  is the product of the factors contributed by the features in  $\mathbf{x}(s)$ , as in the **CNB** model:

$$F(a, u, s) = \prod_{i=1 \dots n} P(x_i(u, s) | a)$$

Finally,  $L(type(a), T)$  is the product of a collection of new factors that estimate the probability that a certain unit-action type is legal, given the type of the unit-action that was selected:

$$L(t, T) = \prod_{t' \in T} P(t' \text{ is legal} | t \text{ was selected})$$

Here,  $P(t' \text{ is legal} | t \text{ was selected})$  is the estimated probability that an action of type  $t'$  was legal in a game state where an action of type  $t$  was selected.

In practice, we observed that learning a different model for each different unit type in the game (*workers*, *bases*, *barracks*, etc. in  $\mu$ RTS), resulted in better estimation of the probabilities<sup>3</sup>. Additionally, we also observed that adding feature selection also slightly improved the estimation. So, for the experiments reported in the remainder of this paper, we generated the probability models in the following way:

<sup>3</sup>The **CNB** model has learnable 6141 parameters, and **AIM** has 10971.

- For each unit type: we train a model using the subset of the training data referring to such unit type. If this subset is empty, then just train with the whole training set (i.e., if we have no training data to model the way a specific unit is controlled, we just train a model with the whole training set for such unit, hoping it will reflect what the expert would have done).
- After training each model, we use a greedy additive wrapper feature selection method [18] to determine the subset of features from  $\mathbf{x}(s)$  that maximize the predictive accuracy in the training set (by doing cross validation).
- After feature selection is performed, for each model, we find the parameter  $\kappa$  that maximizes the likelihood of the training set given the model.

## B. Extending Unit-Action to Player-Action Distributions

When using the trained models to generate actions, it is necessary to actually generate player-actions, and not just unit-actions. When a player-action for player  $p$  needs to be generated in a game state  $s$  according to a unit-action distribution  $P$  (generated with either the **CNB** or the **AIM** models), we use the following procedure:

- 1) Push all the units that require unit-actions,  $ready(p, s)$ , to a queue  $Q$  in a random order. Initialize an empty player-action  $\alpha = \emptyset$ .
- 2) If  $Q$  is empty, return  $\alpha$ .
- 3) Otherwise, remove the first unit  $u$  from  $Q$ . Let  $l = \{a \in L_u(u, s) | L_p(p, \alpha \cup (u, a), s)\}$ , i.e., the set of legal unit-actions for  $u$  that when added to the player-action  $\alpha$  still keep  $\alpha$  being legal.
- 4) If  $l = \emptyset$ , restart the process from 1.
- 5) Otherwise, sample one action  $a$  from  $l$  according to  $P$ , add it to  $\alpha$  as:  $\alpha = \alpha \cup (u, a)$ , and go back to 2.

The previous process samples a player-action using the unit-action distributions, while respecting unit-action legality ( $L_u$ ) and player-action legality ( $L_p$ ). Moreover, notice that in practice,  $l$  will never be  $\emptyset$  in step 4, since units in RTS games can always execute the *idle* action, which does not conflict with any other action. So, in practice this algorithm can sample a player-action without ever requiring going back to step 1. Also notice that choosing a random order in step 1 is necessary to prevent any undesired biases that can be caused by the order in which units are processed in steps 2 and 3 (since some unit-actions might prevent some other unit-actions).

## C. Generating Training Data

Given that no available training data from expert players is available for  $\mu$ RTS, we generated training data in the following way. We selected the two current best reported Monte Carlo search-based bots in the literature (*LSI* [5], and *NaiveMCTS* [6]), and four hard-coded bots built into  $\mu$ RTS (*WorkerRush*, *LightRush*, *HeavyRush*, and *RangedRush*), and played a round-robin tournament (all 36 combinations of each of the 6 bots playing as player 1 and as player 2) in 8 different

maps<sup>4</sup>, resulting in a total of  $288 = 36 \times 8$  games. The configuration used for NaiveMCTS and LSI was the default one as implemented in  $\mu$ RTS, where playouts are limited to be at most 100 game frames long, after which an evaluation function is applied. We also experimented with the AHTN bot [13], but found that it only performed well in 2 of the 8 maps used in our evaluation, probably because the domain definition used was tailored to some specific type of maps.

Moreover, we repeated this round-robin tournament four times giving *NaiveMCTS* and *LSI* a computation budget of 500, 1000, 2000, and 5000 playouts per game frame respectively<sup>5</sup>, resulting in four sets of game logs: RR500, RR1000, RR2000 and RR5000. We then constructed twelve datasets:

- $I_{WR}$ : consisting of all the unit-actions of *WorkerRush* from RR500 (25739 instances).
- $I_{LR}$ : consisting of all the unit-actions of *LightRush* from RR500 (38844 instances).
- $I_{HR}$ : consisting of all the unit-actions of *HeavyRush* from RR500 (28321 instances).
- $I_{RR}$ : consisting of all the unit-actions of *RangedRush* from RR500 (32753 instances).
- $I_{LSI}^{500}, I_{LSI}^{1000}, I_{LSI}^{2000}, I_{LSI}^{5000}$ : consisting of all the unit-actions of *LSI* from RR500, RR1000, RR2000 and RR5000 respectively (89029, 74564, 70696, and 60000 instances).
- $I_{nmcts}^{500}, I_{nmcts}^{1000}, I_{nmcts}^{2000}, I_{nmcts}^{5000}$ : consisting of all the unit-actions of *NaiveMCTS* from RR500, RR1000, RR2000 and RR5000 respectively (84784, 77200, 75462, and 70071 instances).

The feature vector  $\mathbf{x}(u, s)$  used to represent each game state contains only eight features: the number of resources available to the player, the cardinal direction (north, east, south, west) toward where most friendly units are, the cardinal direction toward where most enemy units are, whether we have a barracks or not, and four features indicating the type of the unit in the cell two positions north, east, south or west (or whether these cells are empty or are a wall). Adding more features could certainly improve performance, which will be part of our future work.

#### D. Empirical Evaluation

We evaluated the probability distribution models presented above in two different ways. By measuring how accurately can they predict the behavior of the bots (Table I), and by using them directly to play against the same six bots used to generate the training data (Table II).

a) *Model Predictive Accuracy*: Table I shows the predictive accuracy of both proposed models in each of the 12 datasets using a 10-fold cross validation. There is a total of 69 different actions unit can perform in  $\mu$ RTS, but each individual

<sup>4</sup>Specifically, we used the maps *OneBaseWorker8x8*, *TwoBasesWorkers8x8*, *ThreeBasesWorkers8x8*, *FourBasesWorkers8x8*, *OneBaseWorker12x12*, *TwoBasesWorkers12x12*, *ThreeBasesWorkers12x12*, and *FourBasesWorkers12x12* included with  $\mu$ RTS.

<sup>5</sup>Notice that “500 playouts per frame” does not mean “500 playouts per decision”. Both LSI and NaiveMCTS spread computation across multiple frames when all units are busy in order to maximize playouts per decision.

TABLE I  
CLASSIFICATION ACCURACY, AND AVERAGE LOG-LIKELIHOOD ACHIEVED WITH THE PROPOSED MODELS IN THE 12 DATASETS.

Dataset	CNB		AIM	
	Acc.	Exp. l.l.	Acc.	Exp. l.l.
$I_{WR}$	0.651	-0.952	0.679	-0.892
$I_{LR}$	0.800	-0.509	0.836	-0.441
$I_{HR}$	0.860	-0.381	0.877	-0.345
$I_{RR}$	0.846	-0.409	0.892	-0.318
$I_{LSI}^{500}$	0.380	-1.262	0.381	-1.229
$I_{LSI}^{1000}$	0.392	-1.237	0.397	-1.211
$I_{LSI}^{2000}$	0.410	-1.223	0.413	-1.193
$I_{LSI}^{5000}$	0.419	-1.211	0.425	-1.175
$I_{nmcts}^{500}$	0.393	-1.242	0.396	-1.206
$I_{nmcts}^{1000}$	0.407	-1.228	0.412	-1.193
$I_{nmcts}^{2000}$	0.418	-1.222	0.423	-1.181
$I_{nmcts}^{5000}$	0.429	-1.203	0.440	-1.160

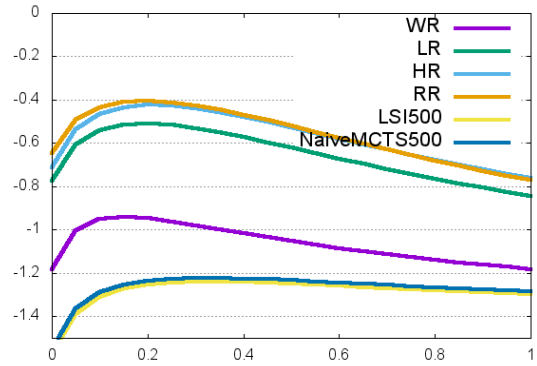


Fig. 2. Expected log-likelihood of the actions (vertical axis) in the training set for the AIM model using various training sets as a function of the  $\kappa$  calibration parameter (horizontal axis).

unit can perform only between 5 and 33 unit-actions. We measure both classification accuracy (1.0 corresponding to perfect predictions), and expected log-likelihood of the data. We can see that both models manage to predict the behavior of the four hard-coded bots very well (and specially the *Ranged Rush*, which the CNB model predicts with 0.846 accuracy and the AIM model with 0.892), but have a harder time predicting the behavior of the *LSI* and *NaiveMCTS* bots. Moreover, notice that the higher the computation budget, the easier *LSI* and *NaiveMCTS* are to predict, indicating that they might be converging to a more stable strategy. Moreover, the AIM model is systematically more accurate than the CNB model.

We also report the expected log-likelihood of the actions in the dataset given the model. This is a better metric to consider than classification accuracy given that we want to estimate the probability distribution of the actions, and not just predicting the most likely action. The best possible log-likelihood would be 0. As we can see, the trends follow exactly those for classification accuracy.

Figure 2 shows how the expected log-likelihood of the actions in the dataset change as a function of the calibration parameter  $\kappa$  for some of the datasets when a single AIM model is trained with all the data and without feature selection. As

we can see, introducing the calibration parameter improves the probability estimation significantly with respect to not having the calibration factor ( $\kappa = 0$ ). In particular, values around the 0.1 to 0.3 range tend to obtain better results, but the maximum is achieved at a different value of  $\kappa$  depending on the training set. Finally, notice that the  $\kappa$  parameter does not have any effect on classification accuracy, since the relative order of which action has higher probability is not affected.  $\kappa$  only affects the probability distribution estimation, making the probabilities less extreme (i.e., less close to either 0 or 1).

b) *Model Play Strength*: Table II shows the average results of playing (by sampling from the trained distribution) in all eight maps described above, and against all the eight bots used to generate the training data. Each model played 20 games against each bot (10 as player 1, and 10 as player 2). Thus, each number in the table is the average of 160 games. Games longer than 3000 game frames were considered a draw. Bots scored 1 point for winning, and 0.5 for reaching a draw. We compare these results against those obtained by two baseline distributions: *Rnd*, which picks actions randomly, and *RndBiased*, where the *attack*, *harvest* and *return* actions have 5 times more probability than the rest of actions. The first thing we can see is that the **CNB** models perform much worse than the **AIM** models. For example, when training from the  $I_{WR}$  dataset, they achieve a score of 0.196 and 0.518 respectively in average. Models trained with the  $I_{WR}$  dataset achieved the highest score (mostly because *WorkerRush* employs a very aggressive strategy that works well against other, less aggressive, strategies). We also see that models trained from *LSI* and *NaiveMCTS* achieved better scores when trained from datasets generated with a higher computation budget. No model managed to win any game against the *WorkerRush* bot or *NaiveMCTS*, and only one bot won one game against *LSI*. So, although none of the trained models are strong enough to defeat the MCTS bots, they are significantly stronger than the baselines. The fact that the **AIM** model trained with the  $I_{WR}$  dataset can defeat most of the hardcoded bots is remarkable since: (1) it plays purely reactively, (2) the time required to train these models is negligible.

#### IV. INFORMED MONTE CARLO TREE SEARCH

We incorporated the models described above into Monte Carlo Tree Search (MCTS), a family of planning algorithms based on sampling the decision space rather than exploring it systematically [2]. MCTS employs two different *policies* to guide the search: (1) a *tree policy* determines which nodes in the tree to explore (i.e., given a node in the tree, which of its children to consider next), and, (2) each time a new node is added to the tree, a simulation (a *playout* or *rollout*) of how the game would unfold from that state until the end of the game (or until a predefined maximum playout length) is executed by using a *default policy* to generate actions for both players.

Thus, the action probability models learned above can be used in MCTS in two ways: to define *tree policies* or *default policies*. While an action probability model can be used directly as a default policy, in order to be used as a *tree*

*policy*, it needs to be incorporated into a multi-armed bandit policy. Below, we describe *informed  $\epsilon$ -Greedy sampling* and *informed naive sampling*, two informed policies, required to then explain *INMCTS*, our proposed algorithm.

##### A. Informed $\epsilon$ -Greedy Sampling

The *tree policy* of MCTS algorithms is usually defined as a *multi-armed bandit* (MAB) policy. A MAB is a problem where, given a predefined set of actions, an agent needs to select which actions to play, and in which sequence, in order to maximize the sum of rewards obtained when performing those actions. The agent has no information of the expected reward of each action initially, and needs to discover them by iteratively trying different actions. MAB policies are algorithms that tell the agent which action to select next, by balancing *exploration* (when to select new actions) and *exploitation* (when to re-visit actions that had already been tried in the past and looked promising).

MAB sampling policies traditionally assume that no a priori knowledge about how good each of the actions are exists. For example, UCT [19], one of the most common MCTS variants, uses the UCB1 [20] sampling policy, which assumes no a priori knowledge about the actions. A key idea used in AlphaGO is to employ a MAB policy that incorporated a prior distribution over the actions into a UCB1-style policy. Here, we apply the same idea to  $\epsilon$ -greedy and then (in the next section) to Naive Sampling.

As any MAB policy, *informed  $\epsilon$ -greedy sampling* will be called many iterations in a row. At each iteration  $t$ , an action  $a_t \in A$  is selected, and a reward  $r_t$  is observed.

Given  $0 \leq \epsilon \leq 1$ , a finite set of actions  $A$  to choose from, and a probability distribution  $P$ , where  $P(a)$  is the a priori probability that  $a$  is the action an expert would choose, *informed  $\epsilon$ -greedy sampling* works as follows:

- Let us call  $\bar{r}_t(a)$  to the current estimation (at iteration  $t$ ) of the expected reward of  $a$  (i.e., the average of all the rewards obtained in the subset of iterations from 0 to  $t-1$  where  $a$  was selected). By convention, when an action has not been selected before  $t$  we will have  $\bar{r}_t(a) = 0$ .
- At each iteration  $t$ , action  $a_t$  is chosen as follows:
  - With probability  $\epsilon$ , choose  $a_t$  according to the probability distribution  $P$ .
  - With probability  $1 - \epsilon$ , choose the best action so far:  $a_t = \operatorname{argmax}_{a \in A} \bar{r}_t(a)$  (ties resolved randomly).

If  $P$  is uniform, this is equivalent to  $\epsilon$ -greedy.

##### B. Informed Naive Sampling

*Naive sampling* is based on the idea of combinatorial multi-armed bandits (CMABs) [6] and internally uses a collection of  $\epsilon$ -greedy sampling policies. Here we define *informed naive sampling*, as a result of replacing some of the internal  $\epsilon$ -greedy sampling policies by informed  $\epsilon$ -greedy.

Informed naive sampling takes four input parameters: a unit-action probability distribution  $P$ , and three constants  $\epsilon_0$ ,  $\epsilon_l$  and  $\epsilon_g$ , and determines which player-action to choose using a collection of MABs:



TABLE II  
GAMEPLAY STRENGTH OF THE MODELS TRAINED WITH DIFFERENT DATASETS, COMPARED AGAINST TWO BASELINES (*Rnd*, AND *RndBiased*).

Bot	<i>Rnd</i>	<i>RndBiased</i>	CNB					AIM				
			$I_{WR}$	$I_{LSI}^{500}$	$I_{LSI}^{5000}$	$I_{nmcts}^{500}$	$I_{nmcts}^{5000}$	$I_{WR}$	$I_{LSI}^{500}$	$I_{LSI}^{5000}$	$I_{nmcts}^{500}$	$I_{nmcts}^{5000}$
<i>Rnd</i>	0.500	0.944	0.263	0.206	0.344	0.231	0.331	<b>1.000</b>	0.881	0.906	0.844	0.938
<i>RndBiased</i>	0.025	0.469	0.019	0.006	0.025	0.019	0.044	<b>0.881</b>	0.163	0.400	0.288	0.463
<i>WorkerRush</i>	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
<i>LightRush</i>	0.013	0.050	0.375	0.013	0.063	0.000	0.038	<b>0.650</b>	0.075	0.200	0.075	0.113
<i>HeavyRush</i>	0.038	0.063	0.488	0.038	0.138	0.038	0.075	<b>0.825</b>	0.213	0.313	0.200	0.300
<i>RangedRush</i>	0.038	0.050	0.425	0.025	0.088	0.025	0.100	<b>0.775</b>	0.163	0.250	0.125	0.263
<i>LSI (500)</i>	0.000	0.000	0.000	0.000	0.000	0.000	0.000	<b>0.013</b>	0.000	0.000	0.000	0.000
<i>NaiveMCTS (500)</i>	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
<i>Average</i>	0.077	0.197	0.196	0.036	0.082	0.039	0.073	<b>0.518</b>	0.187	0.259	0.197	0.259

- A set of *Local MABs*: For each unit  $u_i \in ready(p, s)$ , we define  $MAB_i$  that only considers the unit-actions of  $u_i$ .
- *Global MAB*:  $MAB_g$  considers each player-action that has been sampled so far. This means that in the first iteration,  $t = 0$ ,  $MAB_g$  contains no player-actions at all. Specifically, at each iteration  $t$ :

- With probability  $\epsilon_0$  choose *explore* and with probability  $1 - \epsilon_0$  choose *exploit*:
  - If *explore* was selected: a legal player-action  $\alpha^t$  is selected by using an *informed  $\epsilon$ -greedy* policy (with  $\epsilon_l$ ) to select a unit-action for each unit  $u_i \in ready(p, s)$  independently (i.e., the policy is used  $n$  times, one per local MAB), while ensuring the resulting player-action is legal.  $\alpha^t$  is added to the global MAB.
  - If *exploit* was selected: a player-action  $\alpha^t$  is selected by using an  $\epsilon$ -greedy policy with  $\epsilon_g$  over the player-actions already present in the global MAB.

Intuitively, when exploring, informed  $\epsilon$ -greedy is used to select unit-actions that take into account both the prior probability distribution of unit-actions, and the current reward estimation of the unit-actions. When exploiting, the global MAB is used to find the player-action, among all the explored ones, with the maximum expected reward. After each iteration, the reward estimates of all the individual MABs are updated.

*INMCTS* (*informed NaiveMCTS*), is the result of replacing naive sampling from NaiveMCTS [6] by informed naive sampling.

## V. EMPIRICAL EVALUATION

In order to evaluate the play strength of INMCTS, we performed four sets of experiments, reported in the following three subsections. All the experiments presented in this section are performed under the same conditions as those reported in Table II. The computation budget used for experiments was 500 playouts per cycle.

### A. Experiment 1: Baselines

Table III shows the results of evaluating the gameplay strength of a collection of baseline algorithms against all the bots used for generating the training data, averaged over all the eight maps used in our experiments. The two left-most columns show the performance of NaiveMCTS, using two

TABLE III  
GAMEPLAY STRENGTH OF VARIOUS BASELINE CONFIGURATIONS AVERAGED OVER THE EIGHT MAPS USED IN OUR EXPERIMENTS (20 GAMES PER MATCH-UP PER MAP). <sup>1</sup> DEFAULT CONFIGURATION OF NAIVEMCTS AS ORIGINALLY REPORTED IN [6].

Algorithm	NaiveMCTS	NaiveMCTS <sup>1</sup>	LSI	INMCTS
<i>Tree Policy</i>	-	-	-	<i>RndBiased</i>
<i>Default Policy</i>	<i>Random</i>	<i>RndBiased</i>	<i>RndBiased</i>	<i>RndBiased</i>
<i>Random</i>	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>
<i>RndBiased</i>	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>
<i>WorkerRush</i>	0.600	<b>0.725</b>	<b>0.725</b>	0.700
<i>LightRush</i>	0.713	<b>0.900</b>	0.744	0.894
<i>HeavyRush</i>	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>
<i>RangedRush</i>	0.700	0.788	0.725	<b>0.862</b>
<i>LSI</i>	0.363	0.669	0.500	<b>0.688</b>
<i>NaiveMCTS</i>	0.300	0.500	0.325	<b>0.613</b>
<i>Average</i>	0.709	0.823	0.752	<b>0.845</b>

different *default policies*: *Random* and *RndBiased* (described in Section III-D). We can see that going from *Random* to *RndBiased* has a large effect in the gameplay strength, going from 0.709 for *Random* to 0.823 for *RndBiased*. The third column shows the performance of LSI, which is a bit lower than NaiveMCTS. Although previously reported results showed that LSI outperformed NaiveMCTS [5], in our experiments, that was the case for the smaller maps, while in the larger maps, when the branching factor grows drastically, NaiveMCTS outperforms LSI. This is analyzed later in Section V-D.

The right-most column shows the performance of INMCTS when using *RndBiased* to inform both the *tree policy* and the *default policy*. The table shows that using *RndBiased* to inform the *tree policy* provides an additional performance improvement, going up to 0.845.

### B. Experiment 2: Informed Sampling in the Tree Policy

Table IV shows results when we used *RndBiased* as the *default policy* in INMCTS, but we used different learned models to inform the *tree policy*. We do not show results with all the models shown in Table II due to space limitations, but we show some representative instances. The first thing we observe is that the **AIM** models outperform the **CNB** models. For example, when training models with the *WorkerRush* dataset ( $I_{RW}$ ), we observed a performance of 0.813 with **CNB** versus 0.883 with **AIM**. Moreover, comparing Table IV with

TABLE IV  
GAMEPLAY STRENGTH OF INMCTS USING DIFFERENT MODELS AS THE TREE POLICY (20 GAMES PER MATCH-UP PER MAP).

Algorithm	INMCTS	INMCTS	INMCTS	INMCTS
Tree Policy	CNB( $I_{WR}$ )	AIM( $I_{WR}$ )	AIM( $I_{nmcts}^{500}$ )	AIM( $I_{nmcts}^{5000}$ )
Default Pcly	RndBiased	RndBiased	RndBiased	RndBiased
Random	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>
RndBiased	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>
WorkerRush	0.700	<b>0.781</b>	0.713	0.748
LightRush	0.900	<b>0.975</b>	0.900	<b>0.975</b>
HeavyRush	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>
RangedRush	0.863	<b>0.906</b>	0.838	0.875
LSI	0.594	<b>0.763</b>	0.638	0.675
NaiveMCTS	0.450	<b>0.644</b>	0.575	0.625
Average	0.813	<b>0.883</b>	0.833	0.861

TABLE V  
GAMEPLAY STRENGTH OF INMCTS USING DIFFERENT PROBABILITY MODELS IN THE PLAYOUTS (20 GAMES PER MATCH-UP PER MAP).

Algorithm	INMCTS	INMCTS	INMCTS	INMCTS
Tree Policy	AIM( $I_{WR}$ )	AIM( $I_{nmcts}^{500}$ )	AIM( $I_{nmcts}^{5000}$ )	AIM( $I_{nmcts}^{5000}$ )
Default Pcly	AIM( $I_{WR}$ )	AIM( $I_{nmcts}^{500}$ )	AIM( $I_{nmcts}^{5000}$ )	AIM( $I_{WR}$ )
Random	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>
RndBiased	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>
WorkerRush	0.888	0.731	0.838	<b>0.950</b>
LightRush	<b>0.925</b>	0.675	0.763	<b>0.925</b>
HeavyRush	<b>1.000</b>	0.981	<b>1.000</b>	<b>1.000</b>
RangedRush	<b>0.963</b>	0.838	0.825	0.950
LSI	<b>0.844</b>	0.556	0.669	0.794
NaiveMCTS	<b>0.813</b>	0.450	0.513	0.650
Average	<b>0.929</b>	0.779	0.826	0.909

Table III we can see that informing the tree policy with the **AIM** model significantly outperforms NaiveMCTS.

The second effect we see is that when training models with the NaiveMCTS or LSI datasets, performance increases when training from datasets generated with a higher computation budget. For example, the performance of INMCTS when using an **AIM** model trained with  $I_{nmcts}^{500}$  is 0.833, and it goes up to 0.861 when trained with the  $I_{nmcts}^{5000}$  dataset.

Finally, we see that the best performance was achieved when using the **AIM**( $I_{WR}$ ) model (trained with the *WorkerRush* dataset), which outperformed models trained from NaiveMCTS or LSI. Our hypothesis is that this is because: 1) *WorkerRush* is a very aggressive early rush strategy, which works very well in the small maps used for testing; 2) *WorkerRush* is a deterministic strategy which is easy to model using our probability models; on the other hand, as can be seen from Table I, NaiveMCTS is harder to learn.

### C. Experiment 3: Probability models as Default Policies

Table V shows results from similar experiments to the previous subsection, but where we used trained **AIM** models as the *default policy*. The three left-most columns show results when using the same model for both informing the *tree policy* and as *default policies*, where we can see that when using the model trained with the *WorkerRush* dataset, results improve significantly (to 0.929), but when using the models trained with NaiveMCTS data, performance goes down with respect

TABLE VI  
MEDIAN, AVERAGE AND MAXIMUM BRANCHING FACTOR ENCOUNTERED IN EACH OF THE EIGHT MAPS.

Map	Branching		
	Median	Average	Max
1BW8x8	14.50	1466.35	$2.65 \times 10^5$
2BW8x8	84.00	$1.87 \times 10^6$	$2.49 \times 10^9$
3BW8x8	106.00	$4.53 \times 10^5$	$1.75 \times 10^8$
4BW8x8	342.50	$9.52 \times 10^5$	$3.74 \times 10^8$
1BW12x12	30.50	$1.23 \times 10^6$	$6.14 \times 10^8$
2BW12x12	112.00	$1.56 \times 10^{12}$	$2.90 \times 10^{15}$
3BW12x12	—	—	—
4BW12x12	—	—	—

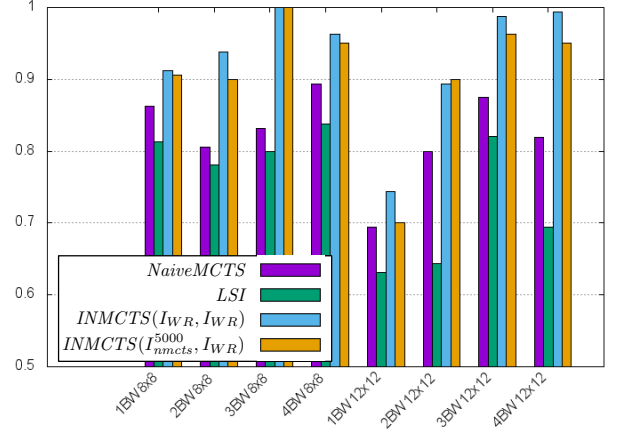


Fig. 3. Performance (vertical axis) in each of the 8 maps used in our experiments (horizontal axis) of two baselines, and the two INMCTS instantiations that performed better in our experiments, with computation budget 500.

to just using the *RndBiased* policy. The right-most column shows results when using a model trained with the *NaiveMCTS* dataset for informing the *tree policy*, and one trained with the *WorkerRush* dataset as the *default policy*, showing also very good performance (0.909).

In conclusion, we can see that the idea of using informed sampling has the potential to significantly improve the performance of MCTS-based bots. The best combination reported in this paper (left-most column on Table V) has a win rate of over 80% against NaiveMCTS, and even higher against LSI. This is remarkable, since the probability distribution model we are using (**AIM**) is much simpler than the complex deep neural network (DNN) approach employed by AlphaGO. This confirms the hypothesis that simpler probability distribution models are enough to significantly improve the performance of MCTS. However, we still believe that performance can be increased even more by using better probability distribution models. In particular, more complex models, such as DNNs might be required when training models from complex bots such as NaiveMCTS or LSI, at the cost of increased computational cost during training.

### D. Effect of the Branching Factor

Table VI shows the median, average and maximum branching factor encountered in each of the eight maps when making



some of the baseline bots play against each other. As can be seen, the 12x12 maps result in significantly higher branching factors, and specially in the last two maps (where players start with 3 and 4 bases respectively), where branching factors were beyond the range we could calculate them exactly (but we could estimate some cases to be over  $10^{18}$ ).

Figure 3 shows the score achieved by two baseline models (NaiveMCTS and LSI) and the two best instantiations of INMCTS in our experiments in each of the eight maps. As can be seen, performance in each of the maps varies greatly. For example, performance in the first four maps (8x8 in size) is higher than in the second four maps (12x12 in size). We can also see that while LSI performs similar to NaiveMCTS in the first 4 maps, its performance drops significantly in the larger four maps. Moreover, in experiments where we increase the computational budget (to 5000 playouts per game frame, not reported in this paper), LSI actually outperforms NaiveMCTS in the first 4 maps, but still performs worse in the larger four maps (consistent with results reported by [5]).

Finally, all bots struggled to perform well in the fifth map (12x12 in size, and players starting with only one base and one worker). This is the map where the distance between the starting positions of the players is the largest. It is so large in fact, that the maximum length of the playouts used in our experiments (100 game frames) is not long enough to simulate a unit traveling all the way to the enemy base. Thus, MCTS-based bots behave erratically at the beginning of the game.

## VI. CONCLUSIONS

This paper has explored the idea of informed sampling in the context of MCTS for games with combinatorial branching factors. The goal was to understand whether one of the key ideas that made AlphaGO surpass existing MCTS approaches can be also help in games with even larger branching factors. For that purpose, we proposed two probability distribution models, **CNB** and **AIM**, which play the role of the *policy network* of AlphaGO and were trained from datasets generated from existing bots. Our experimental results showed the proposed approach significantly outperforms the current state of the art.

Moreover, when modeling probability distributions from bots employing a simple strategy (such as *WorkerRush*), our probability estimation models are enough to significantly improve MCTS. In order to learn distributions from bots, or humans, employing more complex strategies (such as NaiveMCTS), a more complex probability model might be required.

As part of our future work, we would like to explore better probability distribution models, and compare also results against models trained using DNNs (which would likely require larger datasets to avoid overfitting). Moreover, in this paper we did not consider other components of AlphaGO, such as the value network, or learning by self-play, which we would like to study in the future. We would also like to study the scalability of the approach with respect to the game definition. For example, the number of parameters to be learned from data grows linearly with the number of possible unit-actions for the **CNB** model, but quadratically for the **AIM** model. Variables

such as “map size” affect the MCTS search process, but not the probability distribution model learning process. Finally, we believe the idea of informed sampling can be generalized to many other MCTS algorithms for other domains. We have already started work on these ideas on StarCraft, where the main new challenges are the fact that the game is partially observable, and that we do not have a forward model.

## REFERENCES

- [1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, “Mastering the game of go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
- [2] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, “A survey of monte carlo tree search methods,” *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 1, pp. 1–43, 2012.
- [3] M. Buro, “Real-time strategy games: a new AI research challenge,” in *Proceedings of IJCAI 2003*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003, pp. 1534–1535.
- [4] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, “A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft,” *IEEE Transactions on Computational Intelligence and AI in Games (TCIAIG)*, vol. 5, pp. 1–19, 2013.
- [5] A. Shleyfman, A. Komenda, and C. Domshlak, “On combinatorial actions and CMABs with linear side information,” in *ECAI*, 2014, pp. 825–830.
- [6] S. Ontanón, “The combinatorial multi-armed bandit problem and its application to real-time strategy games,” in *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2013.
- [7] D. Churchill and M. Buro, “Portfolio greedy search and simulation for large-scale combat in starcraft,” in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013, pp. 1–8.
- [8] N. Justesen, B. Tillman, J. Togelius, and S. Risi, “Script-and cluster-based UCT for starcraft,” in *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*. IEEE, 2014, pp. 1–8.
- [9] A. Uriarte and S. Ontanón, “Game-tree search over high-level game states in RTS games,” in *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2014)*, 2014.
- [10] D. Soemers, “Tactical planning using mcts in the game of starcraft,” Ph.D. dissertation, Masters thesis, Department of Knowledge Engineering, Maastricht University, 2014.
- [11] R.-K. Balla and A. Fern, “UCT for tactical assault planning in real-time strategy games,” in *Proceedings of IJCAI 2009*, 2009, pp. 40–45.
- [12] M. Stanescu, N. A. Barriga, and M. Buro, “Hierarchical adversarial search applied to real-time strategy games,” in *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2014)*, 2014.
- [13] S. Ontanón and M. Buro, “Adversarial hierarchical-task network planning for complex real-time games,” in *Proceedings of the 24th International Conference on Artificial Intelligence*. AAAI Press, 2015, pp. 1652–1658.
- [14] D. Aha, M. Molineaux, and M. Ponsen, “Learning to win: Case-based plan selection in a real-time strategy game,” in *ICCB'2005*, ser. LNCS, no. 3620. Springer-Verlag, 2005, pp. 5–20.
- [15] I. Rish, “An empirical study of the naive bayes classifier,” in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3. IBM New York, 2001, pp. 41–46.
- [16] P. N. Bennett, “Assessing the calibration of naive bayes’ posterior estimates,” Carnegie Mellon University, Tech. Rep. CMU-CS-00-155, 2000.
- [17] B. Zadrozny and C. Elkan, “Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers,” in *ICML*, vol. 1. Citeseer, 2001, pp. 609–616.
- [18] R. Kohavi and G. H. John, “Wrappers for feature subset selection,” *Artificial intelligence*, vol. 97, no. 1, pp. 273–324, 1997.
- [19] L. Kocsis and C. Szepesvri, “Bandit based monte-carlo planning,” in *Proceedings of ECML 2006*. Springer, 2006, pp. 282–293.
- [20] P. Auer, N. Cesa-Bianchi, and P. Fischer, “Finite-time analysis of the multiarmed bandit problem,” *Machine learning*, vol. 47, no. 2, pp. 235–256, 2002.

# Evolving Missions to Create Game Spaces

Daniel Karavolos

Institute of Digital Games

University of Malta

e-mail: daniel.karavolos@um.edu.mt

Antonios Liapis

Institute of Digital Games

University of Malta

e-mail: antonios.liapis@um.edu.mt

Georgios N. Yannakakis

Institute of Digital Games

University of Malta

e-mail: georgios.yannakakis@um.edu.mt

**Abstract**—This paper describes a search-based generative method which creates game levels by evolving the intended sequence of player actions rather than their spatial layout. The proposed approach evolves graphs where nodes representing player actions are linked to form one or more ways in which a mission can be completed. Initially simple graphs containing the mission's starting and ending nodes are evolved via mutation operators which expand and prune the graph topology. Evolution is guided by several objective functions which capture game design patterns such as exploration or balance; experiments in this paper explore how these objective functions and their combinations affect the quality and diversity of the evolved mission graphs.

## I. INTRODUCTION

Procedural content generation (PCG) in games has received considerable academic interest in the last decade, exploring different ways to represent, generate and evaluate game content such as rulesets, card decks, puzzles, weapons, terrain, etc. Among the most prominent generative techniques being explored are search-based techniques [1] which often use artificial evolution to explore a vast search space guided by an objective function, constraint-based techniques [2] which carefully define the space of viable solutions, and generative grammars [3] which define the creation and expansion rules of an artifact and can gradually increase its level of detail.

The vast majority of PCG research focuses on game level generation, following the trends of the game industry where PCG primarily creates game spaces such as the dungeons of *Diablo* (Blizzard 1996), the gameworlds of *Civilization V* (Firaxis 2010) or the mansions of *Daylight* (Zombie Studios 2014). While commercial games primarily use constructive generative techniques [1], academic interest in PCG has moved beyond this narrow focus and has tested a broad variety of techniques, representations, and types of game levels which can be generated. Most often, such generators create the level's layout and then evaluate its spatial characteristics such as its navigable regions [4] or functional characteristics derived from e.g. playtraces of artificial agents running through it [5]. In the case of [6], the generator creates a tile-based layout of a dungeon for a role-playing game adventure module, which is then used to derive a room connectivity graph for placing encounters to follow the progression of a player from the dungeon's entrance. However, an inverse generative process is also possible, where the structure of the player experience (with all its possible variations and branches) is generated first and is used to derive the spatial structure of the game level.

This paper presents a search-based approach for generating levels through an indirect representation, evaluating and evolving the player's sequence of possible actions rather than the explicit sequence of rooms they have to visit. While the level geometry and the action sequence are linked (i.e. the latter constrains the former), the action sequence is a more concise representation as it does not contain trivial information such as empty rooms or walls. Moreover, the action sequences are represented as a graph of nodes while game levels tend to be represented as some form of bit array [7]; this allows the design of genetic operators (for adding, removing, or connecting nodes) which have a better locality and result in non-trivial yet non-destructive changes to the phenotype. Finally, parsing the graph directly allows for fast and simple evaluations of the decision density of a player traversing a level from start to finish. The paper focuses on the generation of mission graphs for the dungeon crawl game *Dwarf Quest* (Wild Card Games 2013), with nodes representing the start and end of the mission, puzzles, rewards and combat sections. Results show that many different types of mission graphs can be generated, from simple, short playthroughs to complex structures with multiple paths to the goal. The *Dwarf Quest* levels created from these mission graphs similarly range from straightforward and short to maze-like and grueling.

## II. RELATED WORK

Procedural content generation has been used in the game industry, and primarily for the generation of game levels, since the 1980s with games such as *Rogue* (Toy and Wichman 1980) and *Elite* (Acornsoft 1984). Level generation has only increased in scale and commercial appeal in recent years with games such as *Minecraft* (Mojang 2011) and *No Man's Sky* (Hello Games 2016) embracing it as a major selling point. Academic interest in level generation is similarly extensive, with levels for first person shooters [4], puzzle games [2], side-scrolling platformers [8], strategy games [9] and many other game genres being generated using a diverse set of techniques.

Particularly relevant to the current work are search-based and grammar-based techniques for generating levels. The family of search-based PCG [1] methods attempt to gradually improve a level by applying local changes; most often, artificial evolution is used and the local changes take the form of mutation of tiles in a grid-based map or recombination of the layouts of two parents to create offspring that combine the features of both parents. In search-based PCG, it is common

to select the most promising parents to create the next batch of results (generation) based on a quantifiable objective function which evaluates how appropriate a game level is: examples include the length of its paths [9], the combat duration between artificial agents [4] or the distribution of its treasures [5].

As their name suggests, grammar-based techniques take advantage of generative grammars, which represent a set of rewrite rules which transform expressions. Although originally designed to analyze and classify language phrases, grammars can be used to transform any expression. For level generation, grammars have been described and used extensively by Dormans [10] while the most well-known commercial application of grammar-based level generation is *Spelunky* (Mossmouth 2008) [11]. At its core, a generative grammar is a set of rules which can be iteratively applied to increase the complexity of an expression (e.g. a game level). Such rules can frame the problem (e.g. *dungeon*  $\rightarrow$  *obstacle* + *treasure*) or can be recursive (e.g. *obstacle*  $\rightarrow$  *monster* + *obstacle*). If multiple rules can be applied to the expression, one is chosen randomly.

The dual representation for game levels (as a mission and as a space) was first introduced in [3] and expanded in [10], where the mission graph was created via a graph grammar while the architecture was built from shape grammars which rewrite mission nodes into rooms of various sizes. The paradigm was applied to the game *Dwarf Quest* in [12], where both mission graph and layout was created through grammars: the layout solver places rooms on a 2D grid based on the mission graph, obeying requirements on planarity and orthogonality and applying pre-processing steps as needed to repair non-conforming missions. In [13], the generation of missions and spaces in *Dwarf Quest* was enhanced through a human-computer interface that allowed a human designer to interject in (or replace) the generative grammars with her own intuitions. The tool allowed the designer to create missions in varying levels of detail, e.g. authoring a rough sketch of a mission and allowing the generative grammars to expand on that sketch automatically (or with some human curation).

### III. METHODOLOGY

This paper uses search-based techniques to evolve a mission graph representing the player's possible action sequences, which is then used to create a level architecture for *Dwarf Quest* (Wild Card Games 2013). The representation of the mission graph and the types of nodes it can contain is described in III-A, the details of the evolutionary approach and its mutation operators in III-B, the objectives which drive evolution in III-C, and finally the methods for converting the evolved mission graphs into game levels in III-D.

#### A. Mission Representation

The evolved artifacts consist of mission graphs represented as a list of nodes and edges. The nodes represent abstract player actions, such as solving a puzzle. This abstract action will later be transformed into a specific action by a grammar, which is then transformed by a layout solver into one or more rooms where gameplay will take place. A more detailed



Fig. 1. In-game screenshot of *Dwarf Quest* (Wild Card Games 2013), showing a fight node with doorways to two adjacent rooms.

TABLE I  
LIST OF POSSIBLE NODE TYPES SPLIT INTO CATEGORIES.

Neutral	Reward	Fight	Puzzle
Start	RandomItem	Enemy	DoorPuzzle
End	HealthPotion	Boss	BridgePuzzle
	BattleCard		ChestPuzzle
	Treasure		FloorTrap
	Altar		

description of this process is provided in [13]. There are 14 types of nodes described in Table I, split into four categories: *fight*, *puzzle*, *reward*, and *neutral*. Fight nodes involve active opposition from monsters, puzzle nodes involve passive opposition (e.g. locked doors), while reward nodes have no opposition but provide power-ups for future fights<sup>1</sup>. Neutral nodes are the *start* node, where the player is initially placed, and the *end* node where the player completes the level; the goal of the mission is to traverse the graph starting from the start node and reaching the end node. For evolution, each node is stored as an integer acting as the identifier of its node type.

Edges connect two nodes, and are represented by three parameters: the index of the starting node, the index of the ending node, and a flag on whether the edge is directed. For example, edge  $(0, 1, \text{false})$  represents a bidirectional edge between element 0 and element 1 in the node list. Since the corridors in *Dwarf Quest* are bidirectional the current work ignores the third parameter, but this representation supports other game modes involving e.g. one-way portals.

#### B. Mission Evolution

The generative approach followed in this paper evolves an initial population of individuals in order to maximize a fitness function consisting of one or more objectives (covered in III-C). The initial population consists of identical individuals representing the simplest possible mission: a start node, an end node and an edge between them. The following generations increase the topology of these initial individuals, and after the first generation the selection process favors individuals with a higher fitness. The algorithm uses an elitism of 10%,

<sup>1</sup>Among the reward nodes, battle cards act as one-time powerups while altars function like a shop in which potions and battle cards may be purchased.

TABLE II  
LIST OF MUTATION OPERATORS.

Name	Description
Insert Node	A randomly chosen edge is split and a random node is inserted between the edge's start and end nodes, connecting the inserted node via two edges to the initial start and end nodes. This creates longer action sequences.
Add Node	As the insert node operator except the chosen edge is not deleted, providing multiple paths between its start and end nodes (directly or indirectly via the new node).
Change Node	A randomly chosen non-neutral node changes into a random other non-neutral node type.
Delete Node	A randomly chosen non-neutral node is deleted with the following constraints: if the node has one edge, both the node and its edge is deleted; if the node has two edges, an edge is added linking the nodes connected to the deleted node; nodes with 3 or more edges are not deleted as it would be too destructive.
Add Edge	Two randomly chosen nodes are connected with a bidirectional edge. This can create duplicate edges, except when the individual only contains a start and an end node (in which case this mutation can not be applied).
Delete Edge	A randomly chosen edge is deleted, unless it is a node's last edge.

making copies of the fittest parents in the next generation; the remaining individuals in the next generation are mutations of parents chosen via fitness-proportionate roulette wheel selection. The same parent can be selected multiple times, thus generating multiple mutated offspring. Evolution is carried out via mutation alone, and each offspring is a copy of its parent to which multiple mutation operators can be applied based on a probability. Several mutation operators are designed in order to change the topology of the mission graph while obeying constraints to avoid undesirable results. The mutation operators are summarized in Table II. Mutation operators are not allowed to place more than one boss node and more than one altar node per level; other node types are chosen in those cases. The mutation probabilities are based on preliminary testing and favor adding nodes and edges over deleting them, as the latter is more disruptive in most fitness landscapes.

### C. Mission Objectives

There are several desirable patterns that evolved mission graphs should exhibit. Inspired in part by the general design patterns of [14] and their mathematical formulations in [15], five fitness dimensions are designed to drive evolution (alone or combined into a weighted sum). Steps have been taken to convert all the metrics into a [0,1] value range, with high scores representing more desirable content. Designer intuition was applied to specify the desirable value ranges of several of these metrics (e.g. a desired shortest path of 5 to 10 nodes).

- *Shortest Path.* The number of nodes along the shortest path between start and end nodes ( $d_{s,e}$ ) is normalized by eq. (1) to give optimal scores to paths with 5 to 10 nodes.

$$f_p = \min \{ (1 + e^{3-d_{s,e}})^{-1}, 1 - (1 + e^{13-d_{s,e}})^{-1} \} \quad (1)$$

- *Exploration.* Inspired by [15], this function uses flood fill from the start node to evaluate how much the player will need to explore the level before reaching the end

node. Eq. (2) normalizes this metric to give optimal scores to exploration covering three times as many nodes as the shortest path.

$$f_e = 1 - \frac{1}{3} |F_{s,e} - d_{s,e}| \quad (2)$$

where  $F_{s,e}$  is the number of nodes covered by a flood fill algorithm starting from the start node and stopping once the end node is covered.

- *Variation.* The percentage of edges that connect nodes of different categories, excluding start and end nodes.

$$f_v = \frac{E_d}{E} \quad (3)$$

where  $E_d$  is the number of edges connecting non-neutral nodes of different categories (e.g. a fight node and a reward node), and  $E$  is the total number of edges connecting non-neutral nodes.

- *Dispersed rewards.* Based on [15], eq. (4) evaluates the number of nodes considered safe to rewards (i.e. nodes which are much closer to one reward node versus all other reward nodes).

$$f_a = \frac{1}{N} \sum_{i=1}^R A_i \quad (4)$$

where  $N$  and  $R$  is the number of nodes and reward nodes in the mission, respectively, and  $A_i$  the number of nodes with a safety score for reward  $i$  above a threshold of 0.35. Details of how safety is calculated can be found in [15].

- *Balanced rewards.* Based on [15], eq. (5) evaluates whether every reward has an equal number of safe nodes around it as every other reward.

$$f_b = 1 - \frac{1}{R(R-1)} \sum_{i=1}^R \sum_{\substack{j=1 \\ j \neq i}}^R \frac{|A_i - A_j|}{\max\{A_i, A_j\}} \quad (5)$$

### D. From Mission Graphs to Levels

In order to create the game's final levels, evolved mission graphs are interpreted by the layout solver described in [12], which is in turn constrained by the map options of the *Dwarf Quest* game. Due to these constraints, three post-processing steps must be applied on the evolved mission graphs:

- 1) The room with the player's spawn point (start node) has only one corridor. If the start node has more than one edge, we create an empty node linked to the start node and move the start node's edges to the empty one.
- 2) If there are three nodes that are all pair-wise connected, the layout solver cannot decide which of the rooms to place first. To solve this, we insert an empty node between one of the edges.
- 3) *Dwarf Quest* rooms must have at least two corridors: non-neutral nodes with only one edge are omitted.

Furthermore, the layout solver considers the edges between nodes as directional edges, even though they are not implemented as such in *Dwarf Quest*, and uses them to determine the relative positions of the rooms. To achieve that, a flooding

TABLE III  
MEAN FITNESS SCORES (AND THEIR STANDARD DEVIATION) OF THE FITTEST INDIVIDUAL AFTER 100 GENERATIONS.

Fitness	$f_p$	$f_e$	$f_v$	$f_s$	$f_b$
Single Objective	0.99 (0.00)	0.67 (0.21)	1.00 (0.00)	0.68 (0.18)	0.99 (0.02)
$f_p+f_e$	0.90 (0.05)	0.84 (0.10)	—	—	—
$f_e+f_v$	—	0.70 (0.25)	0.99 (0.03)	—	—
$f_v+f_a$	—	—	1.00 (0.00)	0.67 (0.08)	—
$f_p+f_e+f_v$	0.87 (0.09)	0.64 (0.12)	0.98 (0.03)	—	—
$f_p+f_e+f_a$	0.91 (0.07)	0.83 (0.12)	—	0.73 (0.03)	—
All	0.89 (0.08)	0.67 (0.17)	0.95 (0.04)	0.67 (0.03)	0.71 (0.02)

algorithm turn the bidirectional edges of the mission graph into directed ones, based on each node's distance to the start. If the result has nodes with only incoming or outgoing edges, an edge is chosen (based on the distance of its linked node to the end node) and its direction is flipped.

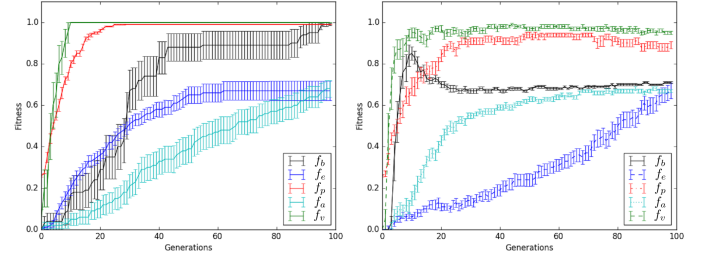
#### IV. RESULTS

The experiments in this paper assess the performance of evolution on mission graphs optimizing each objective individually, optimizing all objectives simultaneously, and a few sample combinations of objectives. Each experiment included 20 independent runs of 100 generations, on a population of 100 individuals. The reported values are averaged from these 20 runs, and the standard deviation is displayed in parentheses or error bars (in tables and figures respectively). Statistical significance tests are performed via two-tailed Student's  $t$ -tests (assuming unequal variance) with a significance threshold of 5%. Since post-processing only contributes to the interpretation of the mission and not to the mission itself, the results below are based on the graphs before post-processing.

##### A. Optimization Performance

Table III displays the average scores in several fitness dimensions of the fittest evolved individuals after 100 generations. Results are derived from optimization runs targeting a single objective (in the single objective row), all objectives and a sample of the possible combinations of objectives. In case of multiple objectives, the overall fittest individuals are considered (according to the summed fitness dimensions' scores). Observing Table III, it is surprising that missions evolved towards  $f_e$  and  $f_a$  individually have a high deviation and low scores while they often reach higher scores when combined with other objectives (significantly higher for  $f_p+f_e$  and  $f_p+f_e+f_a$ ). Other objectives exhibit a less surprising behavior, reaching high scores when evolution targets them individually. Among the objectives,  $f_v$  manages to achieve near-optimal values in all runs and in all combinations of objectives. This may point to the fact that this objective tends to dominate others during multi-objective evolution, although it is equally likely that its fitness score formulation in eq. (3) can reward optimal values to a broad range of mission graphs.

It should be noted that the efficiency of the GA was tested against a *baseline* which ran 20 evolutionary runs with the same parameters, but rewarding all individuals with a constant fitness score (i.e. random selection). The final maximum scores



(a) Single objective evolution for each fitness function. (b) Evolution towards a linear combination of all objectives.

Fig. 2. Progression of the best individuals' fitness scores during evolution, for the fitness functions in Table III and all fitness functions combined. The error bars show the standard error.

of individual fitnesses in the baseline was significantly lower than the respective single-objective optimization runs; while  $f_v$  was relatively close, fitness scores in  $f_a$  and  $f_e$  were 18 times and 6 times those of the baseline respectively. Comparing the best individuals for all objectives (summed) between the baseline and the optimization run targeting it, similar differences were found, with optimization runs creating individuals with 2.8 times the fitness scores of the baseline.

Figure 2a shows the optimization behavior of each fitness dimension when used as a single objective. It is obvious that  $f_p$  and  $f_v$  are quick to optimize, reaching optimal scores in the first 10 to 20 generations; by comparison,  $f_b$  reaches optimal scores much more slowly, with a high standard deviation in most generations (shown as error bars) indicating an unpredictable optimization behavior. On the other hand,  $f_e$  and  $f_a$  reach lower scores (as evidenced by Table III) and improve much slower than the other objectives:  $f_a$  in particular seems to be the slowest to reach even sub-optimal scores.

Figure 2b shows how the scores in individual fitness dimensions fluctuate in the overall fittest individual when evolution targets the sum of all five objectives. Comparing Fig. 2b with Fig. 2a, the differences are surprising. While  $f_v$  and  $f_p$  unsurprisingly reach optimal scores quickly and remain high throughout evolution,  $f_b$  also increases quickly (reaching far higher scores than when evolving individually) and then drops, stabilizing at lower final scores than in Fig. 2a. The optimization behavior of  $f_s$  and  $f_e$  is similarly affected: while they reach similar final scores as in Fig. 2a,  $f_a$  optimizes faster when combined with other objectives and  $f_e$  optimizes far slower. This is likely due to the way that  $f_e$  is computed:



TABLE IV  
MEAN AND STANDARD DEVIATION OF THE DUNGEON METRICS FOR A SAMPLE OF THE FITNESS FUNCTIONS.

Fitness	Graph Size	Shortest Path	Branching Factor	Fights Ratio	Puzzles Ratio	Rewards Ratio
$f_p$	9.25 (0.43)	9.00 (0.00)	1.85 (0.12)	0.14 (0.13)	0.39 (0.18)	0.47 (0.19)
$f_e$	8.60 (2.67)	2.75 (0.43)	2.75 (0.21)	0.20 (0.17)	0.45 (0.19)	0.35 (0.24)
$f_v$	4.10 (0.30)	2.90 (0.70)	1.99 (0.30)	0.23 (0.25)	0.38 (0.20)	0.39 (0.20)
$f_a$	12.15 (4.40)	6.55 (3.47)	2.24 (0.40)	0.05 (0.07)	0.21 (0.10)	0.69 (0.21)
$f_b$	5.90 (1.81)	3.10 (0.94)	2.38 (0.36)	0.09 (0.13)	0.32 (0.18)	0.59 (0.13)
$f_p+f_e$	22.35 (1.88)	6.36 (0.65)	2.68 (0.15)	0.14 (0.06)	0.50 (0.09)	0.36 (0.12)
$f_v+f_e$	8.75 (3.86)	2.55 (0.50)	2.87 (0.23)	0.21 (0.15)	0.37 (0.13)	0.42 (0.19)
$f_v+f_a$	12.76 (3.11)	6.03 (3.05)	2.35 (0.16)	0.19 (0.08)	0.35 (0.08)	0.45 (0.10)
$f_p+f_e+f_v$	17.85 (2.35)	6.15 (0.85)	2.57 (0.13)	0.21 (0.05)	0.40 (0.04)	0.39 (0.06)
$f_p+f_e+f_a$	22.75 (1.61)	6.55 (0.74)	2.58 (0.11)	0.10 (0.06)	0.35 (0.10)	0.55 (0.10)
All	19.00 (0.45)	6.35 (0.08)	2.56 (0.02)	0.18 (0.02)	0.39 (0.00)	0.43 (0.02)

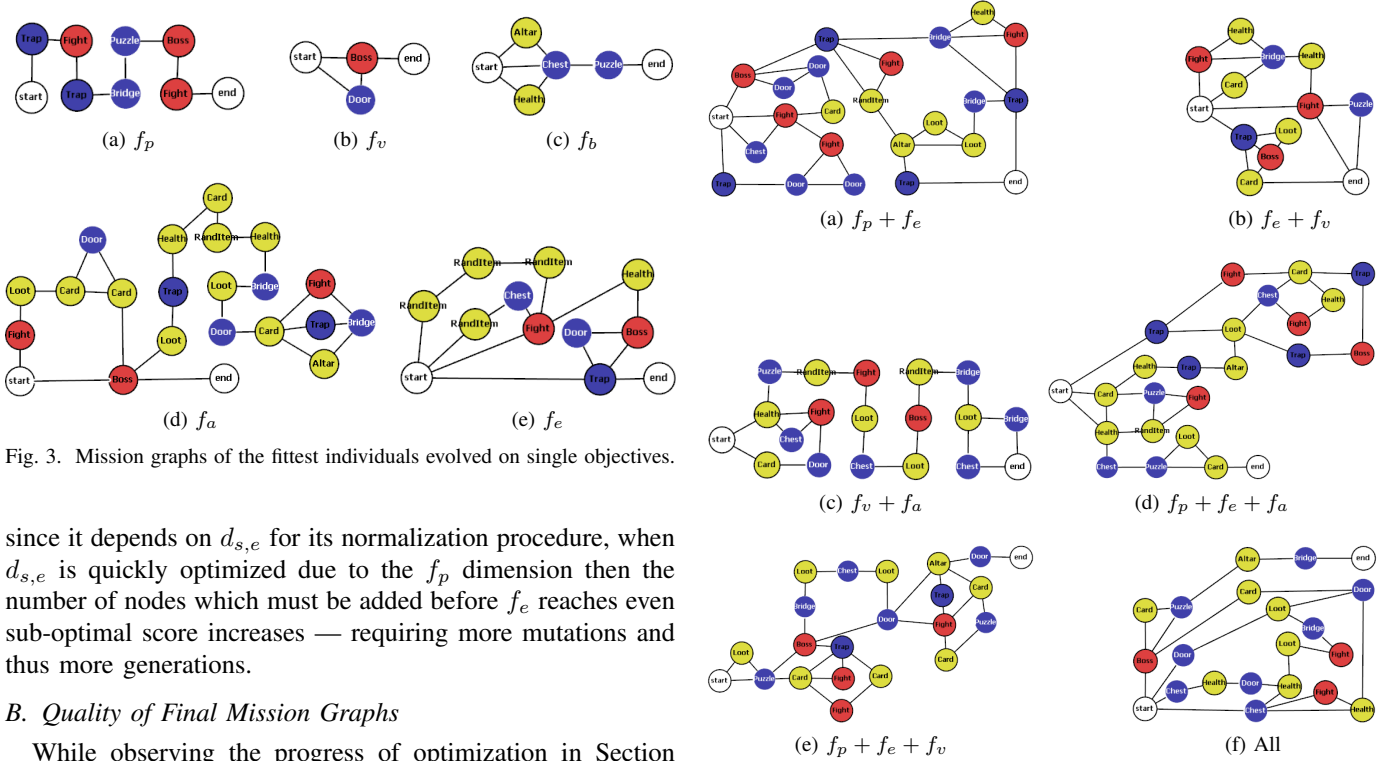


Fig. 3. Mission graphs of the fittest individuals evolved on single objectives.

since it depends on  $d_{s,e}$  for its normalization procedure, when  $d_{s,e}$  is quickly optimized due to the  $f_p$  dimension then the number of nodes which must be added before  $f_e$  reaches even sub-optimal score increases — requiring more mutations and thus more generations.

### B. Quality of Final Mission Graphs

While observing the progress of optimization in Section IV-A from a purely quantitative perspective provides insights on the fitness design, it is perhaps more worthwhile to observe the final mission graphs from the perspective of their potential in-game use. Towards that effect, this section evaluates the fittest final mission graphs (according to different objective functions) in terms of their size, shortest path length, branching factor and composition. Such metrics, which are shared by all mission graphs regardless of the objective function used to evolve or evaluate them, allow for a better comparison between the patterns favored by the different objectives.

Table IV contains the metrics' scores of the fittest individuals evolved towards different objectives; the level heuristics chosen evaluate the structure of the graph (e.g. its size and branching factor) and the composition of its nodes (i.e. how many of them belong to the reward, fight, or puzzle category). We observe that the ratio of puzzles, fights and rewards tends to fluctuate significantly (based on the standard deviation) between individuals, even when they are optimized towards the

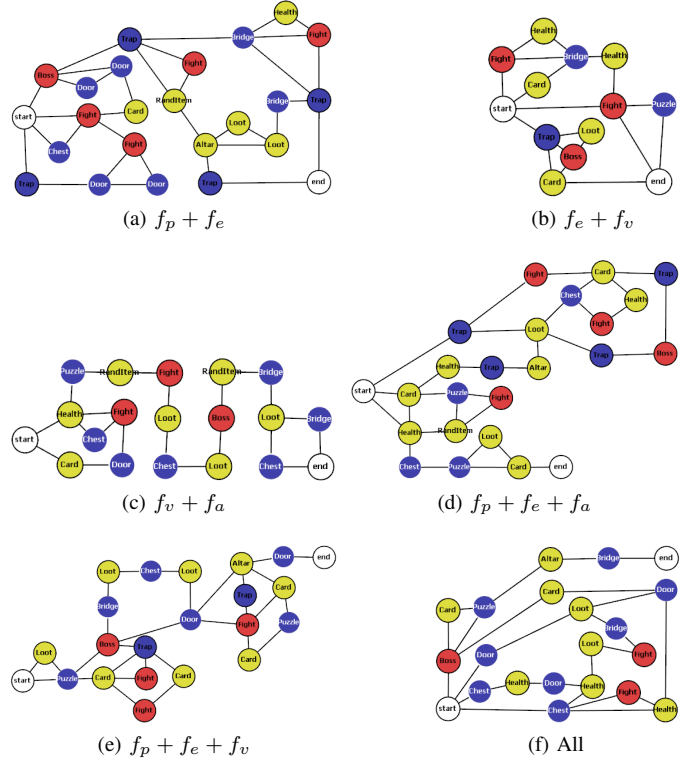


Fig. 4. Mission graphs of the fittest individuals evolved on multiple objectives.

same objective. This should not be surprising considering the fact that when adding new nodes or changing existing ones, the node type is picked randomly. Moreover, the objectives  $f_p$  and  $f_e$  do not differentiate between node types. In graphs evolved towards  $f_p$  or  $f_e$  or their combination, the number of puzzle and reward nodes is roughly equal, with fight nodes being roughly half the number of each other node category. In the case of  $f_v$ , the fitness of eq. (3) rewards changes in type between adjacent nodes, although this does not seem to affect the number of fight nodes in a significant way; therefore, variation likely alternates between reward and puzzle nodes rather than adding more fight nodes. Finally, since  $f_a$  and  $f_b$  specifically focus on reward nodes when evaluating their safety or balance, they create mission graphs with far more rewards than any other type. However, when combining  $f_v$  with  $f_a$  or

$f_b$  (e.g. for  $f_v + f_a$  or all objectives), the number of rewards stays close to that of puzzles due to the variation requirement.

Regarding the topology of the missions, from Table IV it is obvious that  $f_e$  and  $f_a$  create larger mission graphs (graph size) although that does not ensure that the end node is far from the start node. Meanwhile, the fittest mission graphs for  $f_p$  always have a shortest path between start and end node equal to 9 in all runs; this is not surprising as this fitness directly rewards mission graphs with 5 to 10 nodes and the highest value of eq. (1) is when  $d_{s,e}$  is around 9. Additionally, optimal graphs for  $f_p$  have only slightly larger graph size than shortest path length: all nodes of the mission graph are on the shortest path as evidenced by the low branching factor. Missions evolved towards  $f_e$  have the highest branching factor as  $f_e$  directly rewards a much larger flood filled area around the start node than the shortest path length to the end node. Mission graphs for  $f_v$  and  $f_b$  can reach optimal values without reaching a large graph size; this explains why in Fig. 2a these fitness dimensions are optimized so quickly as a few mutations which add nodes to the mission can yield optimal scores. However, these same fitness dimensions when combined with others ( $f_a$ ,  $f_p$  or  $f_e$ ) can create large graphs which still have high scores in that dimension (e.g. in  $f_p + f_e + f_v$ ). Finally, combining all fitness dimensions seems to create levels with the best traits of each objective: large graphs, with long paths from start to end node (although not as long as when  $f_p$  is optimized alone) and a high branching factor. It should be noted that when optimizing both  $f_e$  and  $f_p$  (e.g. when combining all fitness dimensions), the graph size is larger than when  $f_e$  is optimized by itself since  $f_p$  rewards longer paths, pushing  $f_e$  to add more nodes to the mission graph in order to increase the covered area between start and end node up to triple the length of the shortest path.

Figures 3 and 4 show the fittest mission graphs for each of the objectives when optimized alone or in combination, respectively. These graphs support the conclusions from Table IV: the graph for  $f_p$  has no side-passages outside the single path to the end node, the graph for  $f_e$  is large but only one node separates start and end node, the graph for  $f_v$  and  $f_a$  are very small while the graph for  $f_a$  mostly contains reward nodes. It is worthwhile to investigate why  $f_v$  and  $f_a$  are optimal despite their small size: the graph for  $f_v$  has only two non-neutral nodes, which are different and thus assign an optimal  $f_v$  score according to eq. (3). Indeed, having more than three non-neutral nodes (granted that there are three such categories) would be more difficult to optimize due to random node assignment, causing  $f_v$  to actively favor smaller graphs. On the other hand, the graph for  $f_b$  has two rewards placed symmetrically to all other nodes: due to the reward nodes' connections, all nodes are actually unsafe (i.e. equally close) to both rewards and thus the mission graph is “balanced” in terms of safe areas around rewards, with the caveat that there are no such safe areas for either reward.

Observing Figure 4, we observe that all graphs are much larger and complex when optimizing multiple objectives. The paths from start to end node also seem more ‘interesting’ from

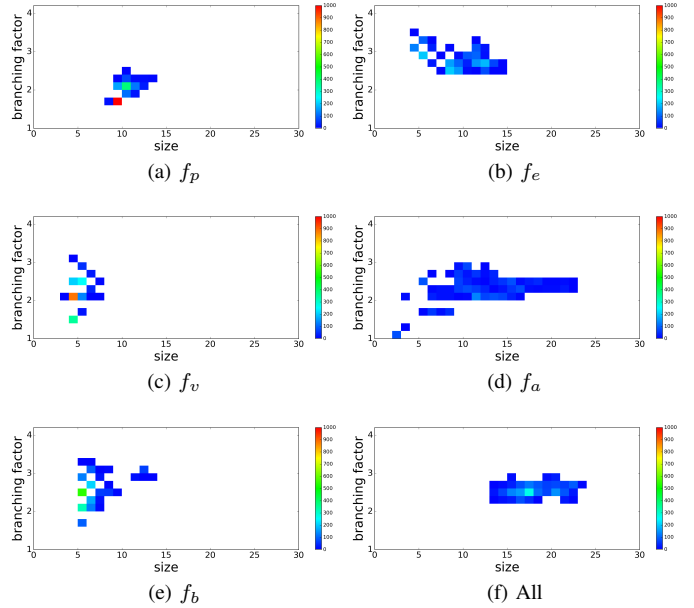


Fig. 5. Graph size versus average branching factor of all final populations evolved for different objectives.

the perspective of progression between node types ( $f_e + f_v$  is an exception, as the hero can reach the exit node by crossing one fight node). Of particular note is the graph for  $f_a + f_v$ , where the path to the end node (which lacks many side-passages) consists of shifts between reward nodes and fight or puzzle nodes, shaping a gameplay that oscillates between tension and relaxation. When all objectives are optimized in Fig. 4f, an interesting pattern emerges: there is extensive branching in the first steps between start and end node, so if the hero takes the right choice at the start then they can reach the exit quickly and without much decision-making later (no branching paths near the end node) or much challenge (one fight along that path). However, if the hero takes the wrong initial decision they can get lost in mazelike side-passages which can make them go in circles back to the start node.

### C. Expressivity Analysis

While observing the fittest mission graphs in Section IV-B provides vital insight into the patterns favored by these objectives, only the one fittest individual per run is assessed. On the other hand, the expressive range [16] of the generator can assess the variety of possible results when optimizing different objectives. The two dimensions explored in this paper are the graph size and branching factor: both of these metrics are not directly targeted by the objectives, as suggested by [16], and are indicative of the actions the hero has to make and the decisions they have to take respectively.

Figure 5 shows heatmaps of the branching factor and graph size values of the final populations of all runs, i.e. a total of 2000 individuals per objective. We observe that the  $f_p$ ,  $f_v$  and  $f_b$  have the most consistent results, with little spread and most individuals centered in specific areas of this expressivity space.

The vast majority of graphs evolved for  $f_p$  have a branching factor of less than 2 and a size of 9 nodes, although when the branching factor increases the graph size also increases (since the shortest path is likely still 9 nodes, the extra branching paths add to the graph size). Most graphs evolved for  $f_v$  are very small (4 or 5 nodes) and no mission graphs have more than 6 nodes; a similar expressivity is exhibited by  $f_b$  although the branching factor is higher. In contrast, graphs evolved for  $f_e$  or  $f_a$  exhibit more expressivity, being able to create very small mission graphs (e.g. with only start and end nodes in the case of  $f_a$  as shown by the bottom-left corner of its heatmap) but tending towards larger mission graphs. Graphs evolved towards  $f_e$  tend towards more branching paths than those evolved via  $f_a$ , which tend towards larger graphs. Finally, when combining all objectives, the expressivity of the results is interesting as it is not similar to that of any individual fitness dimension. Evolved graphs of Fig. 5f are larger with average branching factors, and the values are less dispersed on either metrics than for most of the dimensions. This points to an interesting consensus reached by the — sometimes conflicting — fitness dimensions being optimized.

#### D. Example Level

Since the player will experience the evolved mission graphs as a spatial layout of the dungeon of *Dwarf Quest*, it is worthwhile to investigate how such a level architecture would be. The evolved mission graphs are post-processed and then refined via the mixed-initiative grammar-based system of [13], which creates a larger and more detailed mission graph. This refined mission graph is converted into *Dwarf Quest* levels by the layout solver described in [12].

Figure 6 illustrates level architectures for *Dwarf Quest* based on the evolved mission graphs of Figures 3 and 4. The actual rooms which contain nodes in the mission graph are shown in circles of different colors. The level in Fig. 6a is created from the mission graph of Fig. 4f, which was evolved to maximize all objectives. It is immediately obvious that most rooms in the final level layout are empty and in many cases form long corridors to connect the nodes. This is due to the high branching factor of the graph in Fig. 4f, which forces the layout solver to connect areas far away spatially to their adjacent nodes in the mission graph. In contrast, the central part of the dungeon has fewer empty rooms, with only a couple of rooms between each pair of mission graph nodes.

It should be noted that simpler mission graphs with less branching, such as the graph evolved for  $f_p$  in Fig. 3a, result in far fewer empty rooms as the level is essentially a single path from start node to end node (see Fig. 6b). Similarly the small yet branching mission evolved for  $f_b$  in Fig. 3c creates a similarly simple level (see Fig. 6c) which contains several empty rooms without being exaggerated. The layout solver used for these conversions seems less suited for creating levels with high branching factors or complex topologies, which is also evidenced by the need for the post-processing steps described in Section III-D. By adjusting the layout solver to

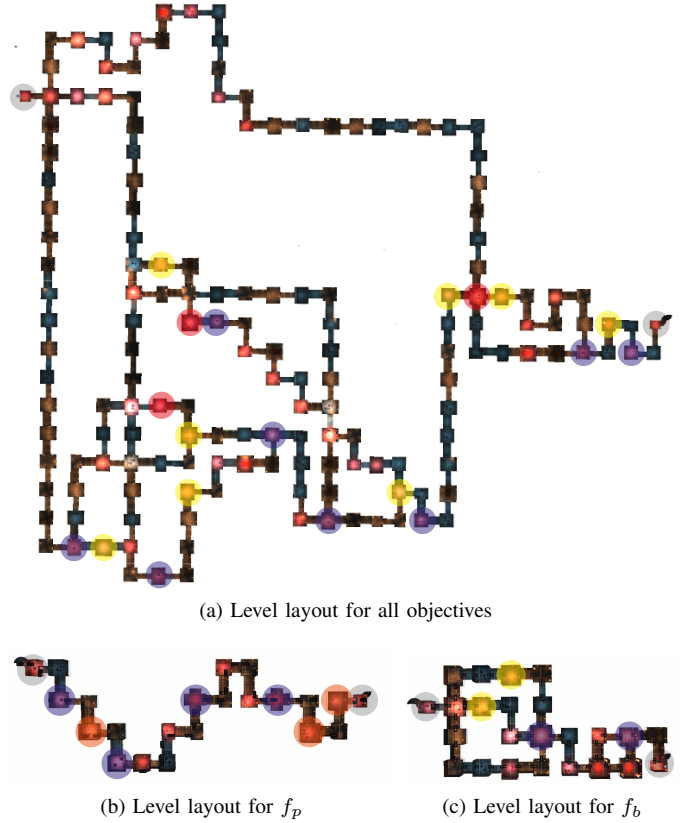


Fig. 6. Level layouts created from missions of Fig. 4f, 3a and 3c. Rooms included in the mission graph are highlighted as circles of different colors. Red, yellow, and blue circles indicate fights, rewards, and puzzles respectively. Gray circles are the start node (left-most) and end node (right-most). In the above illustrations, bright rooms were necessary to place this mission into space, while dark rooms were added as part of the variation process.

place graph nodes closer to one another, many of the issues of extraneous rooms could be avoided.

#### V. DISCUSSION

The results in this paper highlighted the strengths and weaknesses of search-based mission generation, as well as the patterns favored by different objectives of Section III-C. Overall, evolving towards a single objective tends to result in one-dimensional graphs which e.g. have no branching (and thus require no decision-making from the player) or have very trivial level traversals with a couple of non-neutral nodes. Meanwhile, aggregating the scores of multiple fitnesses into a simple sum results in more interesting mission graphs with emergent features such as a larger size or pacing between challenge and relaxation. Observing the way each fitness dimension is optimized when aggregating all objectives hints at the fact that some of the objectives are conflicting and thus a multi-objective optimization approach [17] would probably enhance the quality of the results. However, even with the admittedly naive aggregated approach the outcomes are useful: optimizing all objectives simultaneously creates the most interesting missions with long paths to the end, multiple side-passages and a variety of fight, reward and puzzle nodes.



When assessing the quality of the fittest individuals with respect to their topology and variety of nodes, it is obvious that there are far fewer fight nodes than other types. From a designer's perspective, fights are the most challenging and interesting encounters to be had in the dungeon as they involve the most varied game mechanics (including expending rewards found in the dungeon, such as battle cards). The lack of fight nodes was an artifact of the random node type selection in the different mutation operators: the two types of fight nodes were less often picked than the 4 or 5 node types in the other categories, especially since the boss node could be picked once per level. This could be countered by biasing the choice of fight nodes with a higher probability. More interestingly, designing objectives on the 'quality' of the fight node progression could also enhance the importance of fights in the generated missions. To a degree, the variation ( $f_v$ ) objective achieves that effect, and mission graphs that optimize it (such as in Fig. 4c) alternate between fight or puzzle nodes and reward nodes. However, putting explicit emphasis on fight nodes and e.g. the placement of the boss node towards the end of the mission could improve the current results.

As noted in Section IV-D, the level layouts created from the mission graphs often contain too many empty rooms. The mission generator for the most part creates graphs that adhere to the rules of the level generator, especially after post-processing. Post-processing steps may seem overbearing, such as omitting nodes with one edge: these steps are less destructive than it seems, however, since the mutation operators rarely result in single-edge nodes (none of the examples in Fig. 3 and 4 have non-neutral nodes with one edge). Nonetheless, the resulting spatial structure may be less suited for gameplay than the mission graph suggests. Apart from changes to the level layout solver in order to better handle the branching mission graphs created by some objectives, this limitation can be addressed by evaluating the final level instead of — or in conjunction to — the mission graph. An interesting approach could be to evaluate how much empty space (i.e. non-node rooms) are in the final level layouts of a certain mission graph, applying a penalty to its fitness (calculated as per Section III-C) proportionate to the amount of empty rooms.

## VI. CONCLUSION

This paper described an approach for generating game levels by evolving their indirect representation (a player's action sequence) rather than their direct representation (room layout). Mission graphs representing the possible paths of the player for reaching the goal (end node) were evolved towards different objectives inspired by general game design patterns such as exploration, balance and safety of resources [14]. Experiments in evolving mission graphs towards different objectives individually and in conjunction showed that while different objectives favor different patterns, combining multiple objectives (or even all objectives) results in more complex and more interesting mission graph structures. These more complex graph structures similarly result in quite complex level layouts, which may increase player fatigue when navigat-

ing them. How to address such limitations, and evaluate both the graph structure and the final level layout (i.e. the direct and indirect representation of a game level) is a promising direction for future research.

## VII. ACKNOWLEDGMENT

We would like to thank Dylan Nagel for giving us full access to *Dwarf Quest*'s source code, as well as Rafael Bidarra and Roland van der Linden for the layout solver of *Dwarf Quest*. This work was supported, in part, by the FP7 Marie Curie CIG project AutoGameDesign (project no: 630665) and the Horizon 2020 project CrossCult (project no: 693150).

## REFERENCES

- [1] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 3, no. 3, pp. 172–186, 2011.
- [2] A. M. Smith, E. Butler, and Z. Popovic, "Quantifying over play: Constraining undesirable solutions in puzzle design," in *Proceedings of the International Conference on the Foundations of Digital Games*, 2013.
- [3] J. Dormans, "Adventures in level design: generating missions and spaces for action adventure games," in *Proceedings of the FDG Workshop on Procedural Content Generation in Games*, 2010.
- [4] L. Cardamone, G. N. Yannakakis, J. Togelius, and P. L. Lanzi, "Evolving interesting maps for a first person shooter," in *EvoApplications (1)*, 2011, pp. 63–72.
- [5] A. Liapis, C. Holmgård, G. N. Yannakakis, and J. Togelius, "Procedural personas as critics for dungeon generation," in *Applications of Evolutionary Computation*. Springer, 2015, vol. 9028, LNCS.
- [6] D. Ashlock and C. McGuinness, "Automatic generation of fantasy role-playing modules," in *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2014.
- [7] D. Ashlock, S. Risi, and J. Togelius, "Representations for search-based methods," in *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2015, (In progress).
- [8] G. Smith, J. Whitehead, and M. Mateas, "Tanagra: Reactive planning and constraint solving for mixed-initiative level design," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 201–215, 2011.
- [9] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelbäck, G. N. Yannakakis, and C. Grappiolo, "Controllable procedural map generation via multiobjective evolution," *Genetic Programming and Evolvable Machines*, vol. 14, no. 2, pp. 245–277, 2013.
- [10] J. Dormans and S. C. J. Bakkes, "Generating missions and spaces for adaptable play experiences," *IEEE Transactions on Computational Intelligence and AI in Games. Special Issue on Procedural Content Generation*, vol. 3, no. 3, pp. 216–228, 2011.
- [11] D. Yu, "The full spelunky on spelunky," 2011, accessed 14 April 2016. [Online]. Available: <http://makegames.tumblr.com/post/4061040007/the-full-spelunky-on-spelunky>
- [12] R. van der Linden, "Designing procedurally generated levels," Master's thesis, TU Delft, 2013.
- [13] D. Karavolos, A. Bouwer, and R. Bidarra, "Mixed-initiative design of game levels: Integrating mission and space into level generation," in *Proceedings of the International Conference on the Foundations of Digital Games*, 2015.
- [14] S. Björk and J. Holopainen, *Patterns in Game Design*. Charles River Media, 2004.
- [15] A. Liapis, G. N. Yannakakis, and J. Togelius, "Towards a generic method of evaluating game levels," *Proceedings of the AAAI Artificial Intelligence for Interactive Digital Entertainment Conference*, 2013.
- [16] G. Smith and J. Whitehead, "Analyzing the expressive range of a level generator," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM, 2010.
- [17] C. A. C. Coello, "A comprehensive survey of evolutionary-based multi-objective optimization techniques," *Knowledge and Information systems*, vol. 1, no. 3, pp. 269–308, 1999.

# Online Level Generation in Super Mario Bros via Learning Constructive Primitives

Peizhi Shi and Ke Chen

School of Computer Science

University of Manchester

{shipa, chen}@cs.manchester.ac.uk

**Abstract**—In procedural content generation (PCG), how to assure the quality of procedural games and how to provide effective control for designers are two major challenges. To tackle these issues, this paper exploits the synergy between rule-based and learning-based methods to produce quality yet controllable game segments in *Super Mario Bros* (SMB), hereinafter named *constructive primitives* (CPs). Easy-to-design rules are employed for removal of apparently unappealing game segments, and subsequent data-driven quality evaluation function is learned based on designer’s annotations to deal with more complicated quality issues. The learned CPs provide not only quality game segments but also an effective control manner at a local level for designers. As a result, a complete quality game level can be generated online by integrating relevant constructive primitives via controllable parameters. Extensive simulation results demonstrate that the proposed approach efficiently generates controllable yet quality game levels in terms of different quality measures.

## I. INTRODUCTION

Procedural content generation (PCG) is of great interest to game design and development as it generates game content automatically. In PCG research, a number of methods have been proposed for game content generation in different game genres [1]. *Super Mario Bros* (SMB), a classic 2D platform game, has become a popular test bed for PCG-related research [2], [3]. In this platform game, a player runs from the left side of the screen to the right side, fights enemies, and rescues *Princess Peach*. SMB has a number of different game elements (e.g., enemies, tubes, and cannons) that can be used in PCG. In recent years, several SMB level generators have been developed for *level generation track of the Mario AI Championship* [4] as well as presented in publications [5]–[11].

Although wide varieties of PCG techniques have been proposed, there are still some open challenges faced by SMB level generation as well as generic PCG techniques. One challenge in PCG is how to assure the quality of procedural content. Procedural levels sometimes contain unplayable structures [1], aesthetically unappealing items [8], unexplainable difficulty spikes [12] and unreachable resources [13], which could result in negative gameplay experience. Another challenge is how to effectively control the geometrical features (e.g., coordinate of each enemy and tube) and some properties of procedural levels (e.g., linearity [14] and density [6]). In general, a game designer has to encode the desired properties in handcrafted rules (e.g., theory-driven evaluation functions) to control the procedural levels [12], that could slow down the level generation [13].

This paper presents an alternative approach to address the aforementioned issues. Motivated by the learning-based PCG (LBPCG) framework [16] and other existing works, we explore the content space in SMB from a different perspective by taking short game segments into account. To address the quality assurance issue, we exploit the synergy between rule-based and learning-based methods. Easy-to-design rules are employed for removal of apparently unappealing game segments, and then a data-driven evaluation function is constructed based on designer’s annotations about the quality of game segments. Once the data-driven evaluation function is constructed, we use it along with the aforementioned rules to produce high quality game segments, hereinafter named *constructive primitives* (CPs). Those CPs not only provide quality game segments but also enable to control the geometry and the level properties effectively. As a result, a complete quality game level can be generated online by integrating relevant CPs together via controllable parameters. Experimental results demonstrate that our data-driven evaluation function could implicitly encode multiple quality-related criteria, and improve the quality of procedural level with respect to different quality measures. Results also show that our generator is capable of generating procedural levels of desired properties online.

The main contributions of the paper are summarized as follows: a) a novel approach to producing quality yet controllable game segments or CPs in SMB; b) a controllable online level generator based on CPs; and c) a thorough evaluation of our proposed approach.

## II. LEARNING CONSTRUCTIVE PRIMITIVES

In this section, we first describe the motivation underlying our approach and then present our approach to producing constructive primitives (CPs) in SMB.

### A. Motivation

By a close look at existing SMB level generators, we observe that the content space on all the complete procedural levels is huge. As there are an enormous variety of combinations among game elements and structures at procedural levels, an approach working on such content space inevitably faces a greater challenge in managing quality assurance and generation efficiency in PCG. Nevertheless, a complete procedural level in SMB can be decomposed into a number of segments as evident in [7]–[9]. Partitioning levels into fixed-size game pieces permits us to decompose the level design problem [15]. As a result, all the possible segments form a new content space of lower complexity. We believe that it is less difficult to

understand the properties and quality of short game segments and hence the use of those segments as building blocks would facilitate tackling aforementioned non-trivial issues in SMB.

For quality assurance, there are generally two methodologies in developing such a mechanism in PCG [1], [16]: deductive vs. inductive. To adopt the deductive methodology, game developers have to understand the content space fully and know how to formulate/encode their knowledge into rules explicitly. In the presence of a huge content space, however, it would be extremely difficult to understand the entire content space. Thus, less accurate (even conflicted) rules might be used in PCG, which could generate low quality games. Nevertheless, we observe that some rules are easy to design/identify while a complete set of rules for evaluating the content quality are hard to handcraft. For example, overlapped tubes in SMB is unacceptable and can be easily detected with a simple rule. On the other hand, a learning-based PCG (LBPCG) framework [16] was recently proposed where an inductive methodology, i.e. learning from data, was advocated for quality assurance. As game content is observable but less explainable, it is easier for game developers to make a judgement on quality for a specific game by applying their knowledge implicitly than to encode their knowledge into rules or constraints [8]. Thus, the LBPCG suggests that a quality evaluation function should be learned from data annotated by game developers. Hence, a hybrid approach to quality assurance would allow us to exploit the synergy between rule-based and learning-based methods.

For controllability, game developers usually encode desired properties (e.g., linearity, leniency) into theory-driven evaluation functions (e.g., [5], [7]). Then, game level of desired properties can be generated via generate-and-test method, which is not efficient. We observe that it could provide more efficient and effective control for designers if they directly select game from desired region of content space instead of using theory-driven evaluation functions to explore the content space.

With the motivation described above, we propose a hybrid approach to producing CPs, quality yet controllable game segments, in SMB. Fig. 1 illustrates the main steps of our approach. First of all, game developers choose a region of interest from the entire content space via controllable parameters. Then game segments in the region of interest are evaluated by a set of easy-to-design handcrafted conflict resolution rules and the subsequent data-driven quality evaluation function that deals with more complicated quality issues. Survivals of game segments become CPs.



Fig. 1. The constructive primitive (CP) generation process for SMB.

### B. Content Space

We observe that it is sufficient to cover rich yet diverse types of levels by using a game segment of 20 in length and 15 in height. Some typical game segment instances are illustrated in Fig. 2.

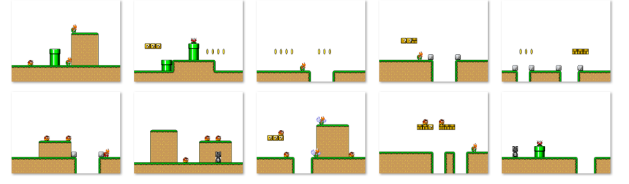


Fig. 2. Game segment instances.

The SMB content is naturally specified by a 2D grid similar to an image. However, this leads to a 300-dimensional content space in our case where there are a lot of redundancies, e.g., the uniform background. In this paper, we employ a list of design elements as our content space representation where a design element refers to an atomic unit used in a procedural level generation, e.g., enemy, boxes, coins, cannon, and gap. By using this representation, we can not only specify the content space concisely but also gain the direct controllability on low-level content features, e.g., coordinates of enemies and coins. As listed in Table I, 85 controllable features are employed in our representation. Such representation is similar to the previous work [5], [6]. In this representation,  $x$ ,  $y$ ,  $width$ ,  $height$ , and  $type$  refer to the  $x$  coordinate,  $y$  coordinate, width, height, and type of each design element, while  $w_{before}$  and  $w_{after}$  refer to width of the platform before and after each tube/cannon. Among these features,  $types$  of gap, tube, boxes and coins are nominal features, while the rests are ordinal features. In our content space, the design elements in each type are sorted in decreasing order along  $x$  dimension.

TABLE I. CONTENT FEATURES

ID	Description
1	$height$ of initial platform
2	number of gaps
3 - 11	$x$ , $width$ and $type$ of the 1st - 3rd gap
12	number of hills
13 - 18	$x$ , $width$ and $height$ of the 1st and 2nd hill
19	number of cannons
20 - 34	$x$ , $y$ , $height$ , $w_{before}$ and $w_{after}$ of the 1st - 3rd cannon
35	number of tubes
36 - 53	$x$ , $y$ , $height$ , $w_{before}$ , $w_{after}$ and $type$ of the 1st - 3rd tube
54	number of boxes
55 - 62	$x$ , $y$ , $width$ and $type$ of the 1st and 2nd boxes
63	number of enemies
64 - 78	$x$ , $y$ and $type$ of the 1st - 5th enemy
79	number of coins
80 - 85	$x$ , $y$ and $width$ of the 1st and 2nd coins

While design element parameters in Table I have a wide range that specifies a huge content space, we confine our concerned content space to a non-trivial region of the content space by setting the maximum number of gaps, hills, tubes, cannons, boxes, coins, and enemies appeared in a game segment are 3, 2, 3, 3, 2, 2, and 5 respectively. These design decisions are made based on our game design knowledge. Consequently, there are roughly  $9.72 \times 10^{37}$  game segments in our content space. This content space should be sufficient for generating content with a variety of geometrical features, level structures and difficulties required by SMB.

The content space defined in Table I is an explicit controllable space, which means that game designers can effectively control the properties of game segments by specifying the desired region of content space. For instance, a pure linear segment can be generated if designers set the number of hills,

$y$  coordinate of tubes and cannons as zero, and set the rest of controllable features on random; a mountainous segment can be generated by setting the number of hills as 1 or 2, and set the rest of controllable features on random.

### C. Conflict Resolution

In our content space, there are quite a number of game segments that contain conflicting design elements. For instance, "...Tube(5,0,2,0,0,normal)...Cannon(5,0,4,0,0)..." represents a game segment of at least one tube and one cannon. The  $x$ ,  $y$ ,  $height$ ,  $w_{before}$ ,  $w_{after}$ , and  $type$  of this tube are 5, 0, 2, 0, 0, and *normal* respectively, while the  $x$ ,  $y$ ,  $height$ ,  $w_{before}$ , and  $w_{after}$  of this cannon are 5, 0, 4, 0, and 0 respectively. We can see from above description that their  $x$  coordinates are same. Thus, the cannon and tube are overlapped together and this conflicting situation makes the segment aesthetically unappealing.

To address this issue, we adapt a class of rules presented in [5], [6] for our requirement. Whenever two design elements in a game segment are overlapped together, this game segment is discarded during CP generation. In our approach, gap, enemy, tube, cannon, boxes, and coins are not allowed to be overlapped with each other, while hills of different heights can be overlapped together. In addition, enemy/tube/cannon can be overlapped with hills.

### D. Learning Constructive Primitives

After filtering out those obviously unappealing game segments, the tailored content space still contains a lot of low quality segments, e.g., segments of unplayable or aesthetically unappealing structures, segments of unreachable or unbalanced resources, and segments that lack a sense of progression. Inspired by the LBPCG work [16], we would learn a quality evaluation function from annotated game segments to remove unplayable/unacceptable segments. To carry out this idea, a binary classifier is trained where its input is the 85D feature vector of a game segment and its output is a binary label that predicts the quality of a game segment. Binary classifier rather than multi-class classifier is chosen since it is easier for annotator to make binary decision about segment quality and avoid using knowledge explicitly. Game segments labeled as positive are CPs and would be used for online level generation described in Sect. III.

To establish a data-driven evaluation function, training examples are required but have to be provided from game developers. As the tailored content space is still huge, it is infeasible to annotate all possible games in this content space. To keep the content space manageable, a proper sampling can be applied to achieve a much smaller data set of the same properties as the content space. Motivated by the success in the LBPCG work [16], we conduct clustering analysis on the data set and further employ active learning based on the clustering results to create a data-driven quality evaluation function (or classifier). Clustering algorithm is used since it can get a better estimation of data distribution, while active learning is chosen to minimize a game developer's efforts in data annotation. In summary, this CP learning process is depicted in Fig. 3.

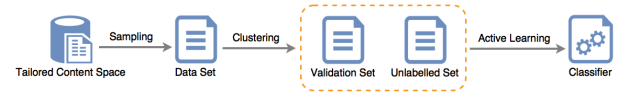


Fig. 3. The constructive primitive learning process.

1) *Sampling*: For sampling, we apply the simple random sampling (SRS) with replacement [17] to the tailored content space for a manageable data set. In contrast to other sampling techniques, SRS is an unbiased sampling technique which can ensure that each game segment in this content space has the equal probability of selection. In addition, it can handle unknown data distribution without any prior knowledge since we do not know the distribution of content space. As a result, we randomly set all the controllable features in the tailored content space to form a data set.

In sampling, an important question is how large a sample should be. Large sample size can guarantee that sample is representative enough, but can also result in increased human work. The size of our data set is determined via the sample size determination (SSD) algorithm suggested in [17] since this algorithm is designed for simple random sampling. According to SSD algorithm, necessary sample size can be calculated as follow:

$$n \approx \frac{z^2 \delta^2}{d^2} = \frac{1.96^2 \times 49.4490}{0.10^2} = 18996.32 \approx 19000 \quad (1)$$

where  $z$  is the  $z$ -score 1.96 for 95% confidence interval, and  $d$  is 0.10 which refers to a small allowable difference. These values are suggested in typical SSD algorithm.  $\delta^2$  is 49.4490 which refers to maximum variance of content space among different features. The value of  $\delta^2$  is determined using an *engineering guessestimate* [18]. With the theoretical justification, the SSD can decide the size of a sampled data set without loss of non-trivial information. By applying the SSD to our tailored content space, it is suggested that a data set of 19,000 games should be sufficient.

2) *Clustering*: We apply the CURE algorithm [19] on the sampled segment set for clustering analysis since this hierarchical clustering algorithm can deal with data set with unknown data distribution and discover the clusters of different sizes. We can achieve good clustering results without any prior knowledge of data distribution within our data set [20]. There are four parameters in CURE algorithm: the number of clusters, sampling rate, shrink factor, and the number of representative points. By using the dendrogram tree achieved, the number of clusters is automatically decided based on the longest  $k$ -cluster lifetime [21]. The rest of parameters are set to defaults suggested in [19]; i.e., 2.5% for sampling rate, 0.5 for shrink factor and 10 representative points, respectively. Due to the existence of two different feature types, i.e. nominal and ordinal, we employ the mixed-variable distance metric [20] in the CURE. After clustering, we found 106 clusters from this sampled data set. Game segments within same cluster tend to have similar structures (e.g., same number of design elements), while segments in different clusters may look different. The clustering results would be used to facilitate active learning.

3) *Active Learning*: For binary classification, there are two error types: *false negative* (type-I error) where a high quality segment is misclassified as low quality and *false positive*

(type-II error) where a low quality segment is misclassified as high quality. Obviously, a type-II error could result in a catastrophic effect while a type-I error simply shrinks the content space slightly. As a result, we formulate our classification as a cost-sensitive learning problem where the type-II error incurs a higher cost. By looking into several state-of-the-art classification techniques, we found that the weighted random forests (WRFs) [22], a cost-sensitive oblique random forests [23] classifier, fully meet our requirements for active learning. Random forests are an ensemble learning method for classification and regression by training different decision trees based on different subsets of the data. Such classifier can handle data set with different feature types, and allows us to know which feature is more important during training. WRFs is a cost-sensitive version of random forests in which the class weights are taken into consideration. Such algorithm is easy to incorporate into typical random forests, and allows us to control the weights of two types of errors easily. In our work, the parameters of WRFs [22] are set via validation as follows: 2:1, 50, 5, 10, and 9 for the cost ratio, the number of trees, the number of combined features, the number of feature groups selected at each node, and depth of trees, respectively.

After clustering, a small number of segments are selected from each cluster to form a validation set in order to evaluate the generalization performance of a classifier during active learning. The number of segments selected from each cluster is proportional to the cluster size. Totally, there are 800 segments in the validation set. One of author annotates each game segment in the validation set by visual inspection. In general, it takes us less than five seconds to annotate a game segment.

During active learning, we randomly choose 100 segments and annotate them via visual inspection to train the initial WRFs. In each iteration, we find 100 segments of the highest uncertainty scores, defined by  $s_i = 1 - P(\hat{y}|x_i)$  where  $\hat{y}$  is the predicted label of segment  $x_i$ , and  $P(\hat{y}|x_i)$  is the probability of this prediction, and annotated them to be examples for re-training WRFs. The active learning stops when the accuracy of WRFs on the validation set no longer increases. Although there are other active learning techniques, our active learning algorithm based on uncertainty sampling is efficient to handle a data set of 19,000 points. Such algorithm is summarized in Algorithm 1.

Once the evaluation function is learned, we use it along with the rules described in Sect. II.C to produce CPs in a generate-and-test way, and a combination of proper CPs via controllable parameters leads to an online procedural level generator as described in next section.

### III. ONLINE LEVEL GENERATION

As described in Sect. II, CPs provide quality building blocks and hence lumping them together can easily lead to a procedural level of aesthetically appealing content with a path between entrance and exit. In SMB, there are a variety of procedural levels that can be categorized based on a number of properties, e.g., density [6], leniency [14], and linearity [14]. As our CPs are represented by design elements, we can generate a procedural level of pre-setting property via the corresponding controllable level generation parameters.

---

#### Algorithm 1 Active Constructive Primitive Learning

---

**Input:** Sampled data set  $U$  and clustering results on  $U$ .

**Output:** WRFs binary classifier.

**Initialization:** Based on the clustering analysis results, create a validation set  $V$  of 800 examples.

**Active Learning:**

Annotate 100 segments randomly selected from  $U$  via visual inspection to form a training set  $L$ . Train WRFs on  $L$  to obtain an initial binary classifier.

**repeat**

**for all**  $x_i \in U$  **do**

    Label  $x_i$  with the current WRFs.

    Calculate the uncertainty score  $s_i$  of  $x_i$ .

**end for**

  Annotate 100 segments of the highest uncertainty score in  $U$  to form a new training set  $L$ .

  Re-train the WRFs with the examples in  $L$ .

**until** The overall accuracy on  $V$  does not increase.

**return** Classifier WRFs.

---

Motivated by the previous works [6], [14], we employ three controllable level generation parameters, i.e., density, leniency, and linearity, to generate a variety of levels online. The density controls the complexity of geometrical structures, e.g., a high density leads to many overlapping hills. The leniency decides the level difficulty in gameplay; intuitively, a high leniency results in an easy-to-play level. The linearity is yet another parameter that ensures there is a linear structure in a generated level; a large value leads to a level of highly linear structures. Each level generation parameter is set to  $\{1, 2, 3\}$ , and carried out by setting the proper values to relevant content features in CPs as defined in Table II. It is worth stating that linearity may conflict with density. Hence, we stipulate that the density and linearity parameters cannot be used together.

TABLE II. PARAMETERS USED IN GAME GENERATION

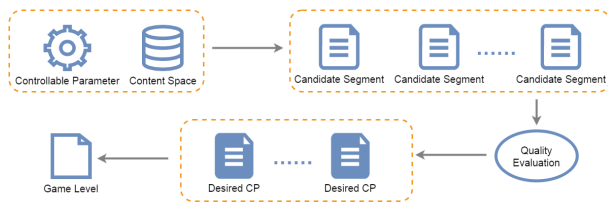
Parameter	Value	Description
Leniency	1	number of enemies $\geq 2$ ; number of gaps $\geq 1$
	2	number of gaps $\leq 2$ ; width of gaps $< 3$ ; 1 $\leq$ number of enemies $\leq 3$ ; enemies without wings
	3	number of enemies $\leq 1$ ; number of gaps $\leq 1$ ; number of cannons = 0; width of gaps $< 3$ ; no turtle enemy
Linearity	1	$0 \leq$ number of hills $\leq 2$ ; height of the first platform = 2
	2	$y$ coordinates of tubes and cannons = 0; number of hills $\leq 1$ ; number of hills $\leq 1$ ; height of the first two platforms = 2
	3	$y$ coordinates of tubes and cannons = 0; number of hills = 0; number of hills = 0; height of the first three platforms = 2
Density	1	$0 \leq$ number of hills $\leq 1$
	2	$0 \leq$ number of hills $\leq 2$
	3	number of hills $\geq 1$ ; number of gaps $\geq 1$

To generate a complete level, we first specify the desired values to controllable parameters that fix the values of relevant content features and set other irrelevant content features in game segments randomly. Thus, game segments of desired properties are generated, and evaluated by rule-based and data-driven quality evaluation functions. Survivals of game segments become CPs, and an iterative process is undertaken by merging the CPs of the specified properties together until reaching a pre-specified length.

As depicted in Fig. 4, our CP-based online generation

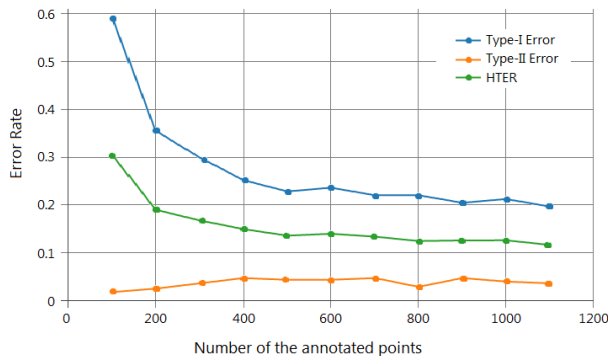


algorithm first uses a generate-and-test method to produce CPs for quality assurance, and a complete procedural level is then constructively generated by sequentially lumping CPs of specified properties together via setting controllable parameters at a local level.

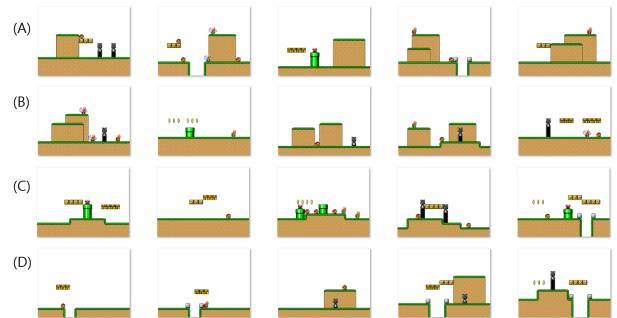


## IV. EXPERIMENTAL RESULTS

### A. Results on Constructive Primitives Learning



While the final HTER is around 11.67%, the corresponding type-I error rate is around 19.66%. It is evident from Fig 5 that our cost-sensitive classifier performs well in minimizing the type-II error; the type-II error rate is approximately 3.69%. Among those segments that yield type-II error, about 0.74% segments contain unreachable resources and the rest segments consist of unexplainable difficulty spikes, imbalanced resources, and aesthetically unappealing structures. While such



By analysing clustering analysis results, we found that segments located in same region of content space tend to be classified as same category. For instance, the 5th example in Fig. 6 (B) and rest examples in its cluster are classified as negative since most instances in this cluster are annotated as negative examples. During training, WRFs finds a single split that optimizes the information gain to classify these instances. Thus, some of them are misclassified. The 2nd example in Fig. 6 (B) is also misclassified due to same reason. In addition, segments of complicated data distribution are likely to be misclassified. For instance, the 5th example in Fig. 6 (C), 4th and 5th examples in Fig. 6 (D) share similarities since they both have boxes above the rock gap, but these instances are located in different regions of content space due to their representations. During training, WRFs has to find three different splits instead of one split to classify these segments. Thus, some of them will be misclassified since these splits cannot optimize information gain. The data distribution of this type of segments is complicated since these data points look similar from perspective of annotator, but spread through

whole content space due to their representations. This is the main source of this type of errors. The 1st, 3rd, and 4th examples in Fig. 6 (B) are also misclassified due to this reason. Moreover, some “rare segments” are likely to be misclassified. The proportion of some types of game segments (see the 1st, 2nd and 3rd examples in Fig. 6 (D)) in whole content space is small. On one hand, those segments are located in several quite small regions of the content space, and unlikely to be selected in both training and validation set. On the other hand, WRFs uses greedy strategy to find an optimal split according to the split criteria. These “rare instances” maybe overlooked and then misclassified since these splits cannot optimize the information gain.

Experimental results demonstrate that our data-driven evaluation function can effectively eliminate low quality segments in terms of our labeling criteria. This argument is supported by the low type-II error achieved from active learning. In addition, this function can implicitly encode multiple quality-related criteria, which could avoid the design and use of multiple evaluation functions.

### B. Quality Analysis

This section further examines the implications and benefits of our data-driven evaluation, and compares our approach to other generators in terms of quality. Unlike [8] that uses human subjects to evaluate their generator, we use different quality measures to compare our system with others since these metrics are capable of evaluating large numbers of levels objectively.

In this experiment, we use strategic resource control ( $f_s$ ), area control ( $f_a$ ), area control balance ( $b_a$ ), and exploration ( $E$ ) defined in [25] as our metrics for evaluation since they are domain-independent metrics which are verified by game developers. Among these metrics,  $f_s$  measures how close treasures (e.g., coins, boxes) are to enemies;  $f_a$  measures the placement of design elements (e.g., enemies, tube flowers, cannons) away from each other;  $b_a$  measures how balanced distribution of design elements;  $E$  evaluates the exploration efforts to reach the exit starting from entrance. Mathematical definitions of these metrics are defined in [25]. Apart from above measures, we also employ Peter Lawford’s A\* agent [24] in small state to evaluate the playability ( $P$ ) of game levels since this agent can survive from most playable game levels no matter how difficult they are.

As we aim to examine the benefits of our data-driven evaluation function, we use our generator to generate two sets of levels. Each level in the first set is composed of randomly generated constructive primitives, while each level in the second set is composed of game segments which survive from conflict resolution rules but are rejected by data-driven evaluation function. For a thorough evaluation, we compare our approach with a number of SMB level generators, including Notch [3], Grammar Evolution (GE) [6], and generators developed for *level generation track of the Mario AI Championship* [4]. Each of level generators generates 100 procedural levels of 200 in width and 15 in height for evaluation in terms of aforementioned metrics. The level generation parameters used in ours are set randomly and others use their default settings. The  $E$  scores are normalized to the range of [0,1].

Table III illustrates the mean and the standard deviation of quality scores achieved from different level generators. We now use one-tailed Wilcoxon rank sum test to make a pairwise comparison between our approach (levels in set 1) and others with respect to different quality metrics. From Table III, it is evident that both ours (levels in set 1), GE generator, and Mawhorter’s generator receive high  $f_s$  scores, which implies that treasures (e.g., coins, boxes) in these levels are close to enemies acting as their “guardians” [25]. In contrast, design elements (e.g., enemies, boxes, coins) in procedural levels of set 2 are arbitrarily lumped together without any reason, which leads to relatively lower  $f_s$  score. Regarding area control ( $f_a$ ) and area control balance ( $b_a$ ), our generator tends to generate levels with significantly higher  $f_a$  score than others and moderate  $b_a$  score, which implies that design elements (e.g., enemies, tube flowers, cannons) in our procedural levels are placed far away from each other, and have balanced distribution. For exploration ( $E$ ), Mawhorter’s generator [11] tends to generate levels with complicated structures and multiple paths, which leads to highest  $E$  score. In contrast, ours generate levels of low  $E$  score in comparison to others due to the initial content space defined in our approach. We found that our density parameter could increase  $E$  score since it controls the complexity of geometrical structures. Such result is not presented here since this experiment only examines the benefits of data-driven evaluation function instead of controllable parameters. For playability ( $P$ ), both ours, Notch generator and Takahashi’s generator achieve a high percentage of playable levels. The playability results of these three approaches are significantly better than the rests. Please notice that  $P$  score value is dependent on A\* agent’s gameplay performance and sampled levels. Procedural levels that A\* agent cannot pass through are not necessarily unplayable.

TABLE III. AVERAGE QUALITY SCORE.

Generator	$f_s$	$f_a$	$b_a$	$E$	$P$
Ours (set 1)	0.34(0.11)	0.46(0.08)	0.48(0.07)	0.20(0.07)	1.00(0.00)
Ours (set 2)	0.29(0.10)	0.36(0.09)	0.45(0.07)	0.18(0.06)	0.92(0.27)
Notch	0.27(0.26)	0.24(0.19)	0.42(0.24)	0.12(0.05)	1.00(0.00)
GE	0.34(0.19)	0.41(0.12)	0.42(0.09)	0.17(0.05)	0.96(0.20)
Weber	0.29(0.05)	0.40(0.07)	0.43(0.06)	0.33(0.14)	0.28(0.45)
Shimizu	0.24(0.05)	0.29(0.04)	0.50(0.06)	0.32(0.07)	0.93(0.26)
Mawhorter	0.32(0.06)	0.42(0.07)	0.43(0.07)	0.44(0.19)	0.81(0.39)
Takahashi	0.23(0.09)	0.19(0.10)	0.54(0.15)	0.18(0.05)	1.00(0.00)
Baumgarten	0.18(0.05)	0.20(0.05)	0.53(0.06)	0.31(0.11)	0.86(0.35)

Extensive simulation results show that our data-driven evaluation could improve the quality of procedural games with respect to different quality measures, and our generator is capable of generating game levels with high  $f_s$ ,  $f_a$ , and  $P$  scores.

### C. Controllability

In this section, we evaluate the controllability of our level generator. In general, controllability can be reflected in the expressive ranges of procedural levels generated with different level generation parameter settings [7]. Expressive range refers to the range and variation of procedural levels according to an evaluation metric [14]. This paper uses *linearity* [14], *density* [6], and *leniency* [14] as our metrics since they can reveal global properties of game levels generated by a level generator. For linearity, we use the method suggested in [13] to find a line that fits the profile of a procedural level, and the coefficient of



determination  $r^2$  is used to estimate the degree of linearity. For density, we count the number of all possible standing positions in a game level [6]. For leniency, we assign a value to each type of game elements as same as used in [6], [7] (i.e., enemy: -1.0, gap, cannon or flower tube: -0.5, and powerup: +1.0). The overall leniency score is the sum of the three values.

As ours have three level generation parameters and each may take one of three values as described in Sect. III, we exhaustively generate nine sets of levels by fixing one parameter with a specific value and randomly setting all other parameters each time. To see the controlling effect clearly, we also generate a set of levels by setting all the parameters randomly. Thus, we achieve 10 level sets where each contains 100 levels for reliability. A game level is confined to a 2D map of 200 in width and 15 in height, as same as the setting in previous work, e.g., [5]. In terms of linearity, density and leniency, the expressive ranges of levels controlled by different parameters are shown in Fig. 7 where it is clearly seen that the levels of a specific property are generated by properly controlling a parameter.

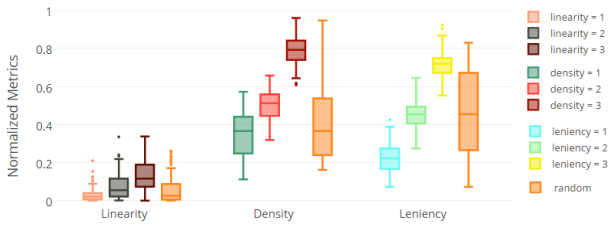


Fig. 7. Expressive ranges of our level generator corresponding to different controllable parameter values.

For exemplification, Fig. 8 illustrates some levels generated by controlling parameters in a specific way. Level in Fig. 8 (C) consists of 14 hills, which leads to highest density value. Levels in Fig. 8 (A) and (B), however, contain 2 and 8 hills respectively, which leads to lower density values. In addition, there are more complicated geometrical structures (e.g., more overlapping hills) in the level shown in Fig. 8 (C) than those shown in Fig. 8 (A) and (B).

Experimental result demonstrates that our generator provides control for designers by controlling relevant content features in CPs locally, and generates a complete procedural level of desired properties.

#### D. Generation Efficiency

Generation efficiency is often evaluated by the actual time taken in a level generation. By testing on a PC (Intel Core i5-3470 processor with 8GB memory), our level generator takes only 0.057 sec on average to generate a procedural level, 200×15 2D map, which should be able to meet the online generation requirements.

### V. DISCUSSION

In this section, we discuss the issues arising from our work and relate ours to pervious works.

Quality assurance is an open challenge in the area of PCG since automatically generated procedural levels are generally worse than those handcrafted levels [12]. Different

types of approaches have been proposed to tackle this issue. For instance, Shaker et al. [6] presented a system which used design grammars and handcrafted fitness functions for generating SMB levels via grammatical evolution algorithm; Smith et al. developed the *Launchpad* generator [7] which used handcrafted critics for quality assurance and designer control. In general, identifying proper constraints and formulating heuristic evaluation functions is difficult since game content is observable but hard to explain and abstract. Thus, Reis et al. [8] merged human-annotated game segments to form complete aesthetical appealing and enjoyable levels. However, they did not train a model to generalize the annotated data. Dahlskog and Togelius [9] used design patterns learned from human-authored SMB levels to generate levels, while Snodgrass and Ontañón [10] learned Markov chains from human-authored SMB levels as constructive rules for level generation. These approaches aim to generate levels with training data (e.g., human-authored levels) instead of domain knowledge. However, inferring/learning reliable constructive rules (or diverse design patterns) from these levels without any domain knowledge might be difficult since the number of human-authored levels are limited (e.g., 32 human-authored levels in SMB). In our approach, we explore and exploit the synergy between rule-based and learning-based methodologies to assure the quality of game content. Easy-to-design rules are employed for removal of apparently unappealing game content, while a learning-based approach addresses the rest of quality assurance issues with a single evaluation function. Our data-driven evaluation function implicitly encodes multiple quality-related criteria based on game developers' judgment on quality of training examples, and improves the quality of procedural game with respect to different quality measures. However, a learning-based approach rarely yields the error-free performance, which could be a potential weakness of such approach.

Another issue in PCG is to effectively control the properties of procedural content. In general, a game designer has to encode the desired properties in handcrafted rules (e.g., theory-driven evaluation functions) in order to control the procedural levels (e.g., [5], [7]). While those evaluation functions may work less efficiently especially for generating procedural levels of a considerable length. In contrast, our online level generator clearly benefits from CPs, it provides effective and efficient control for designers. On the one hand, our generator is proposed based on a direct content representation concerning low-level geometrical features. By controlling relevant content features directly, game content of desired properties are directly selected from the specified region of content space. This process is more efficient than aforementioned approaches since using theory-driven evaluation functions to explore the content space is computationally expensive. On the other hand, our representation is working at a local level for CPs. Thus, our generator generates a procedural level efficiently by integrating CPs of desired properties. It is noticed that the desired level properties have to be specified via setting controllable parameters at a local level. This would be a potential weakness when such properties are unknown or hard to specify. In addition, the expressive ranges of procedural levels are also mainly determined by local controllable parameters. Game developer could generate procedural levels of wide expressive range by tuning controllable parameters locally.

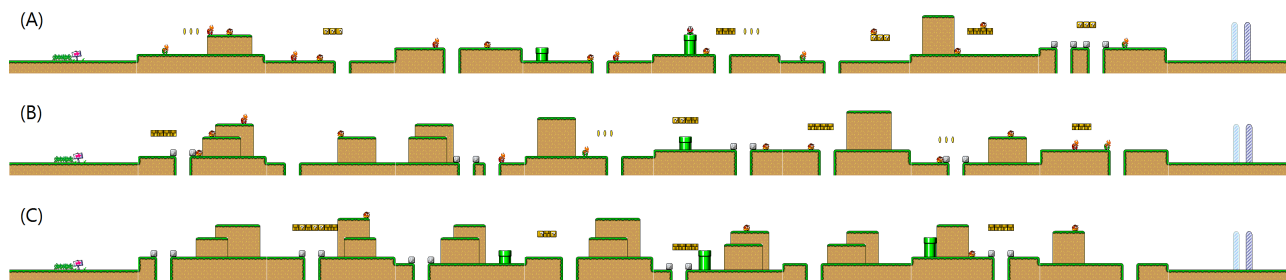


Fig. 8. Exemplar levels generated with different density values. (A)  $density = 1$ . (B)  $density = 2$ . (C)  $density = 3$ .

In general, our online level generator may be viewed as a hybrid PCG approach if we position it in light of the existing taxonomy [1]. On the one hand, we use a generate-and-test method to produce CPs for quality assurance. On the other hand, a procedural level is constructively generated via a number of controllable parameters for effective control. Apparently, ours distinguishes from aforementioned approaches in terms of quality assurance and resultant controllability.

In conclusion, we have presented a novel approach to online level generation in SMB. Our approach can also be used for offline game generation, which allows for using more complex controlling parameters to generate richer content in contrast to our online generation. We explore and exploit the synergy between rule-based and learning-based methodologies to produce controllable yet quality constructive primitives. A complete quality game level can be generated by integrating relevant constructive primitive together via controllable parameters on geometrical features and level properties. We have further carried out a thorough evaluation on our proposed approach and other approaches. The experimental results demonstrate that our approach online generates quality yet controllable levels efficiently. In our ongoing research, we have been working on the application of CPs to generate adaptive games for personalization and extension of this approach to first-person shooter (FPS) games.

#### ACKNOWLEDGMENT

The authors are grateful to N. Shaker for providing SMB procedural level images used in our comparative study and to A. Liapis who responded to our enquiries for useful discussions.

#### REFERENCES

- [1] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Trans. Comput. Intell. AI in Games*, vol. 3, no. 3, pp. 172-186, 2011.
- [2] Nintendo EAD, "Super Mario World," (Game) 1990.
- [3] P. Persson, "Infinite Mario bros," [Online]. Available: <http://www.mojang.com/notch/mario/>.
- [4] N. Shaker, J. Togelius, G. N. Yannakakis, B. Weber, T. Shimizu, T. Hashiyama, N. Sorenson, P. Pasquier, P. Mawhorter, G. Takahashi, G. Smith, and R. Baumgarten, "The 2010 Mario AI championship: Level generation track," *IEEE Trans. Comput. Intell. AI in Games*, vol. 3, no. 4, pp. 332-347, Sep. 2011.
- [5] N. Sorenson, P. Pasquier, and S. DiPaola, "A generic approach to challenge modeling for the procedural creation of video game levels," *IEEE Trans. Comput. Intell. AI in Games*, vol. 3, no. 3, pp. 229-244, Sep. 2011.
- [6] N. Shaker, M. Nicolau, G. N. Yannakakis, J. Togelius, and M. O'Neill, "Evolving levels for Super Mario Bros using grammatical evolution," in *Proc. IEEE Conf. Comput. Intell. Games*, 2012, pp. 304-311.
- [7] G. Smith, J. Whitehead, M. Mateas, M. Treanor, J. March, and M. Cha, "Launchpad: A rhythm-based level generator for 2-d platformers," *IEEE Trans. Comput. Intell. AI in Games*, vol. 3, no. 1, pp. 1-16, Mar. 2011.
- [8] W. M. P. Reis, L. H. S. Lelis, and Y. Gal, "Human computation for procedural content generation in platform games," in *Proc. IEEE Conf. Comput. Intell. Games*, 2015, pp. 99-106.
- [9] S. Dahlskog and J. Togelius, "A multi-level level generator," in *Proc. IEEE Conf. Comput. Intell. Games*, 2014, pp. 1-8.
- [10] S. Snodgrass and S. Ontañón, "Experiments in map generation using markov chains," in *Proc. FDG Workshop on Procedural Content Generation*, 2014.
- [11] P. Mawhorter and M. Mateas, "Procedural level generation using occupancy regulated extension," in *Proc. IEEE Conf. Comput. Intell. Games*, 2010, pp. 351-358.
- [12] J. Togelius, A. J. Champandard, P. L. Lanzi, M. Mateas, A. Paiva, M. Preuss, and K. O. Stanley, "Procedural content generation: goals, challenges and actionable steps," *Artificial and Comput. Intell. in Games*, vol. 6, pp. 61-75, 2013.
- [13] B. Horn, S. Dahlskog, N. Shaker, G. Smith, and J. Togelius, "A comparative evaluation of procedural level generators in Mario AI framework," in *Proc. FDG Workshop on Procedural Content Generation*, 2014.
- [14] G. Smith and J. Whitehead, "Analyzing the expressive range of a level generator," in *Proc. Workshop on Procedural Content Generat. Games*, 2010.
- [15] C. McGuinness and D. Ashlock, "Decomposing the level generation problem with tiles," in *Proc. IEEE Congr. Evol. Comput.*, 2011, pp. 849C856.
- [16] J. Roberts and K. Chen, "Learning-based procedural content generation," *IEEE Trans. Comput. Intell. AI in Games*, vol. 7, no. 1, pp. 88-101, 2015.
- [17] S. K. Thompson, *Sampling (second edition)*. New York: Wiley, 2002.
- [18] Selecting sample sizes [online], <http://www.itl.nist.gov/div898/handbook/k/ppc/section3/ppc333.htm>.
- [19] S. Guha, R. Rastogi, and K. Shim, "CURE: an efficient clustering algorithm for large databases," *ACM SIGMOD Record*, vol. 27, no. 2, 1998.
- [20] J. Han and M. Kamber, *Data mining*. San Francisco, CA, itd: Morgan Kaufmann, 2001.
- [21] A. L. N. Fred and A. K. Jain, "Combining multiple clusterings using evidence accumulation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 6, pp. 835-850, 2005.
- [22] C. Chen, A. Liaw, and L. Breiman, "Using random forest to learn imbalanced data," Univ. California, Berkeley, CA, Tech. Rep., 2004.
- [23] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5-32, 2001.
- [24] J. Togelius, S. Karakowsky, and R. Baumgarten, "The 2009 Mario AI Competition," in *Proc. IEEE Congr. Evol. Comput.*, 2009.
- [25] A. Liapis, G. N. Yannakakis, and J. Togelius, "Towards a generic method of evaluating game levels," in *Proc. AAAI Conf. Artif. Intell. Interact. Digit. Entertain.*, 2013.

# Hyper-heuristic General Video Game Playing

Andre Mendes  
Department of Computer Science  
New York University  
New York, New York 11021  
andre.mendes@nyu.edu

Julian Togelius  
Department of Computer Science  
New York University  
New York, New York 11021  
julian@togelius.com

Andy Nealen  
Department of Computer Science  
New York University  
New York, New York 11021  
andy@nealen.net

**Abstract**—In general video game playing, the challenge is to create agents that play unseen games proficiently. Stochastic tree search algorithms, like Monte Carlo Tree Search, perform relatively well on this task. However, performance is non-transitive: different agents perform best in different games, which means that there is not a single agent that is the best in all the games. Rather, some types of games are dominated by a few agents whereas other different agents dominate other types of games. Thus, it should be possible to construct a hyper-agent that selects from a portfolio, in which constituent sub-agents will play a new game best. Since there is no knowledge about the games, the agent needs to use available features to predict the most suitable algorithm. This work constructs such a hyper-agent using the General Video Game Playing Framework (GVGAI). The proposed method achieves promising results that show the applicability of hyper-heuristics in general video game playing and related tasks.

## I. INTRODUCTION

In order to be generally intelligent, an agent (artificial or natural) needs to be capable of behaving intelligently in a wide range of problems or environments. While one can use artificial intelligence methods to solve a particular problem, the resulting AI application is AI in only a narrow sense. To create general AI, one therefore needs to develop algorithms that are capable of solving a large number of problems. At least, this is the definition of intelligence and artificial intelligence implied by Legg and Hutter [1], variants of which are widely accepted in the Artificial General Intelligence community [2]. In other words, the AI application should not be dependent on the human designer re-tailoring the algorithm for each problem, otherwise we simply have a set of narrow AI solutions, with a human designer choosing which one to apply where.

This is also part of the reasoning behind the General Game Playing (GGP) [3] and General Video Game Playing (GVGAI) [4] competitions. In both, competitors submit game-playing agents, or controllers, that play any game adhering to a given interface. The developers of these controllers do not know at submission time which games their algorithms will be tested on and the winner is the controller which plays these *unseen* games best.

So far, the winners of the GGP and GVGAI competitions are mostly based on generic tree search algorithms. In particular, algorithms based on some version of Monte Carlo Tree Search (MCTS) [5] have been performing well. However, other very general algorithms, such as rolling horizon evolutionary planning [6] have also achieved good results.

It stands to reason that an algorithm that is capable of solving a large number of dissimilar problems will in some way be adaptive, such that it chooses which strategy to adopt depending on the problem. The analogy is that a human approaches different problems in various ways, depending on some familiarity with the type or initial experiences with solving these problems. In an artificial intelligence context, we could imagine an agent that includes several problem-solving algorithms or sub-agents, and chooses which one of them to use every time it encounters a new problem. The general concept of a method that selects sub-level methods to solve a problem seems to have been proposed independently in different lines of research and is couched in different terminologies. These ideas have been expressed in hyper-heuristics [7] [8], algorithm selection [9], meta-learning [10], [11] and ensembles [12].

In this paper, we describe the creation of a “hyper-agent” for general video game playing that utilizes the strengths of multiple individual controllers to play unseen games better than any of them individually. This hyper-agent uses an offline learning approach, i.e. it uses set of trained instances to acquire information about controllers performance and create a selection model that generalizes well for new, unseen games. For clarity, we use the term hyper-agent instead of hyper-controller because it does not directly control the main character but it selects the best controller to do so.

Such an undertaking assumes that playing strength is non-transitive, i.e. different controllers excel at different games, and that there is some kind of regularity to which games are played best by which controllers. It also assumes that we can find game features that let us explore this regularity and use it to predict which controller would play an unknown game best. These assumptions are, as far as we can tell, hitherto untested for our target domain (general video game playing), making this paper the first attempt at using algorithm selection or hyper-heuristic techniques in general video game playing.

For the experiments in this paper, we use the GVGAI framework (the software used for the GVGAI competition), and a number of controllers developed for the competition. Based on a set of games, classifiers are trained to predict which controller excels at a given game. Then a hyper-agent uses the best classifier in games it has never been trained on. The final goal is to choose the best controllers to play each game in real time.

The paper is structured as follows. The next section describes general video game playing and the GVGAI framework as well as some principles of hyper-heuristics and algorithm selection. Section III describes our methodology: which controllers we used as constituent controllers for our hyper-heuristic agent, how we extracted features to use and collected data on controller performance, and train classifiers to predict the best controllers. Section IV describes the results of using these classifiers in a hyper-heuristic agent, as well as an attempt to use the cluster analysis to understand the game space induced by the selected game features.

## II. BACKGROUND

Hyper-heuristics are methods or learning mechanisms for selecting or generating heuristics to solve computational search problems [13]. To classify these methods two components are considered: the nature of the heuristics' search space, and the different sources of feedback information. According to the nature of the search space, the methods can be defined as heuristic selection, when used for selecting existing heuristics, and heuristic generation, when the goal is to use existing heuristics to generate new ones [7], [9]. By this definition, heuristic selection has a large overlap with the algorithm selection problem, where the goal is to select the best algorithm to solve different instances of the problem without modifying the algorithm [14]. In this work, we explore the literature from both fields and although we prefer to use the term hyper-agent, we acknowledge that it can also be considered as an algorithm selection approach.

### A. Hyper-heuristics

With regard to the feedback information, a hyper-heuristic is considered a learning algorithm if it uses feedback from the search process to improve its performance. This feedback can be online and offline. It is online when the learning happens while the algorithm is solving an instance of a problem. It is offline if it uses information in the form of rules or programs, from a set of training instances, that will hopefully generalize well for unseen instances [13][15]. There are also hybrid methodologies that combine online and offline feedback [16], and heuristic selection with heuristic generation [17] [18].

Three requirements can be considered when defining a hyper-heuristic approach. It has to manage a set of low-level heuristics, it searches for a good method rather than for a good solution, and most importantly, it uses only limited problem-specific information in order to generalize well for other applications [19]. The method proposed in this paper fulfills all three requirements and can be classified as a selection hyper-heuristic that uses offline feedback information.

A similar method has been proposed to explore patterns of regularities of two heuristics for constraint satisfaction. The collected performance of heuristics is used to generate a heuristic that selects others in the constructive process to produce a solution [20]. Another study uses feature extraction and hyper-heuristics to improve the problem state representation of irregular packing problems [21].

Most of the applications of hyper-heuristics approaches have been focused on problems in domains such as production scheduling [22] and education timetabling [23]. In the games domain, Li and Kendall use a heuristic selection mechanism integrates a number of existing heuristics for specialized strategic games into an automated game player [24]. Elyasaf et al. evolve heuristics to guide staged deepening search to develop top-notch solvers for the hard game of FreeCell [25]. Using a evolutionary algorithm to pick from a set a low-level heuristics, Salcedo-Sanz et al. proposed a hyper-heuristic method for the puzzle-game Jawbreaker [26] and Benbassat et al. presents two types of approaches, "minimalist" (less human knowledge in the setup) and "maximalist" (using human knowledge in the setup) to evolve game strategies for board games [27].

Although in the same domain, the methods proposed differ from our problem in significant ways such as: the type of games, the type of the heuristics' search space and feedback information, and the information available to the hyper-heuristic about the game and the low-level heuristics.

### B. Algorithm Selection

One influential taxonomy of algorithm selection methods [9] suggests that an algorithm can be identified in steps. First it analyzes the type of *portfolios*, i.e the set of algorithms that will be selected. A portfolio can be static [28], [29] if the algorithms are defined a priori and never change or dynamic if the portfolio changes while solving the problem [30]. From the portfolio, methods can select the single best algorithm or allocate times slots to combine results from different algorithms. Concerning to when they select, the methods can pick before the solving of the actual problem starts or while the problem is being solved [31]. Another important step is how this selection is made. The decision involves, for example, analyzing accuracy, computational cost and time and even number of low-level heuristics to manage [18]. Finally, there is also an essential step that concerns about finding information to help the selection, such as feature selection and extraction [32],[28] and the use of the performance of the selected algorithms in the past.

Based on the problem and data that we have and following the proposed organization, our method can be defined as using a static portfolio, that selects the best single algorithm before the problem is being solved. Our selection method uses machine learning models based on the previous performance of our algorithms (controllers) in each instance (games). To train these models we manually select features available at the start of each game.

Algorithm selection has been applied to domains such as linear algebra [33], linear systems [34] and specially to combinatorial search problems [31], [29]. The use of algorithm selection in games seems to be restricted to game theory [35] and we could not find any applications in video games in the literature.

### C. General video game playing

In the past decade, video games have become increasingly popular as an AI benchmark, as they require a rich repertoire of cognitive capabilities for humans to play well, but can be simulated simply and cheaply on a computer. A number of recent AI competitions have been based on different kinds of video games. The GVGAI competition was created mainly to counteract the problem that these competitions allow participants to tailor their submissions strongly to a particular game. Instead, game-playing controllers submitted to the GVGAI competition are pitted against a number of *unseen* games [4]. This separates it from e.g. the Arcade Learning Environment, where controllers are trained to play a number of well-known games [36]. Another difference is that GVGAI controllers get access to a structured representation of the game state as well as a simulator that allows the use of tree search algorithms to play the games.

So far, the results have shown that methods based on MCTS and MCTS-like algorithms have done very well on this benchmark, but based on the algorithm superiority [37] and in our observation, we see that performance is non-transitive: different controllers play different games best. This can be seen even when adding seemingly minor modifications to the core MCTS algorithm; modifications that improve performance of MCTS on one game may degrade performance on another [38].

## III. METHODS

### A. Games

The games in the GVGAI platform are created using a Video Game Description Language (VGDL) that is designed around objects that interact in two dimensional space [39]. Using level description and game description, VGDL can describe a wide variety of video games, including approximate versions of Space Invaders, Sokoban, Pong, Pac-Man, and Mario Bros.

In this work, we use 41 games available in the GVGAI framework as of December 2015. Each game has five different levels that differ from each other through variations on the locations of sprites, resources available and variations on non-player character (NPC) behavior.

### B. Game-playing controllers

Since the hyper-agent selects the best low-level controllers, its performance is directly related to the performance of the controllers available in its portfolio. In light of that, it seems reasonable to think that the portfolio should be composed of the best controllers. However, based on previous work [40] and on our analysis of the competition, we noticed that the overall champions are able to perform well in most of the games but fail badly in playing some specific ones. Meanwhile other overall low-performance controllers are able to win in these specific games. Therefore, we want a balanced portfolio composed of complimentary controllers with different strengths that, together, win a broader variety of games. Another important characteristic is to select a number

of different controllers that guarantees variability but is not so large that makes hard for the model to choose from [28].

Within these considerations, we use a static portfolio composed of seven controllers. Three of them are high-performing algorithms that were among the best places in 2014 and 2015 competition. The other four are standard algorithms created by the developers of the framework with mediocre overall performance but with good results in specific games where the high-performing algorithms fail.

The controller *adriencx* (ACT), created by Adrien Couëtoux and first place in 2014, uses the algorithm Open Loop Expectimax Tree Search (OLETS) [4], which is inspired by the Hierarchical Open-Loop Optimistic Planning algorithm (HOLOP) [41]. It uses a method called Open Loop Expectimax (OLE) with uses nodes containing a reward function that can store the empirical average reward obtained from simulations that exited in the node and the maximum of reward functions values from the nodes children.

The second place in 2014, created by Jerry Lee, is called *JinJerry* (JJ). JinJerry is based on a MCTS with a selection strategy similar to the one used in [42]. In every game cycle, It creates a one-level tree with the current game state as the root node and the one-action-ahead states of all actions as the leaf nodes. After that, it needs to evaluate the immediate and potential score of all the leaf nodes so that it can use a scoring heuristic to evaluate the states. The action selection strategy is to select the action with a safe state and a high score. If the random actions lead to a state where the game is over, the potential score is not considered. Finally, the potential score will replace the immediate score if it has a higher score.

The champion algorithm for 2015 is YOLOBOT (YB) created by Tobias Joppen, Nils Schroeder and Miriam Moneke. It combines methods like breadth first search and MCTS to analyze the current state and simulations of future states reachable through one-action-ahead. It also creates an observation list of all the sprites reachable in the state and defines the most interesting target based on a function that analyzes empirical values from the observations.

There are also four sample controllers. Sample One Step Look Ahead (SOS) uses a simple heuristic function to evaluate the states reached within one move from the current state and select the action with highest reward. The Genetic Algorithm controller (GA) implements an online (rolling horizon) genetic algorithm, where each individual represents a sequence of actions and its fitness is evaluated using an heuristic function. Small populations are evolved in each game step and the move returned is the first action of the best individual found.

The last two controllers, the sample MCTS (MCTS) and OL-MCTS (OLMCTS), use a similar vanilla Monte Carlo Tree Search implementation [5]. The sample MCTS is extended with common enhancements such as using an Upper Confidence Bound for trees selection and a random expansion. The algorithm evaluates the states by giving a high reward for a won game and a negative reward for a lost game. If the final state is not reached, it uses the score achieved. OL-

MCTS player is a sample MCTS that uses an Open Loop implementation, focusing more on information gained from the current state of the game [4].

### C. Non-Transitive nature

To evaluate the non-transitive nature of the controllers, we played all the games using the procedures explained in Section III-H. Table I shows the sum of victories that each controller achieved for five levels in each game and Table II shows the normalized average scores. The intensity of the cells is proportional to the performance of the controller, where higher and lower values are respectively presented in lighter and darker cells. These results show that our portfolio is composed of complimentary controllers that can excel in a wide variety of games.

### D. Game Features

Features have a direct impact in the performance of our classifier. Although the game in the GVGAI platform is unknown to the controller, the state observation of the game provides information about various elements, such as number and type of NPCs, type of resources, dimensions of the map, action set, and types and number of sprites. We analyzed the classes available in the general state observation of all games and defined 14 features that are presented in each of them and we believe would represent different types of games. Table III shows the features, the group they belong to, and a description of each of them.

The first group is the resources available in the game. We believe that these features provide important information about changes in the rules of the game, such as when Pacman eats a pill and can eat enemies. The second group refers to Non-Player Characters (NPCs) and it provides certain information about the type of game, for example, if the agent has enemies or it is alone (like in a puzzle game). Dimensions is used to give an idea about the space the agent has in the game. Although elements such as block size seem like an unusual choice, Figure 2 shows that it provides useful information in the decision tree model.

The Sprites groups provide information mainly about movable things in the game and how it affects the agent. Portals, similarly to resources, is a good indication of changes in the game. For example, a game with portals can offer more changes because new enemies or resources can come out of them. Finally, the actions group gives very important information about what the user can do in the game, such as move up or down and attack enemies.

### E. Data collection

For feature extraction, we analyzed different games in their start (game tick = 0) and after a period of time (game tick > 0). Except for number of NPCs, all of our features are static, i.e. they remain the same during gameplay. We then created a collector bot that saves the features at the initial state of each level in each game.

In our approach, the labels are the best controllers for each level of the games. In order to compare these controllers,

we ran the experiments described in Section III-H. Using the features extracted and the average performance of the controllers, we created a dataset composed of 205 instances (41 games  $\times$  5 levels) with 14 features and 205 labels (best controller for each instance).

### F. Classification

Using the dataset from the data collection, we utilize multiclass supervised learning to create a model for algorithm selection using methods implemented in the Weka machine learning software [43]. Similar to previous work in the field [44], [45], we choose algorithms that could achieve good accuracy, such as Support Vector Machines (SVM, using libSVM implementation [46]) and Multi-Layer Perceptrons (MLP), and methods that could also report an understandable strategy for the classification, like Decision trees (J48 implementation in Weka) and logistic regression (LR), like in previous works.

Our labels for the classification are the ones obtained during data collection described in section III-E. To prevent overfitting due to the small size of our dataset, we perform two types of cross validation. First we use a normal 10-fold cross-validation (CV-1) where in each round the data is shuffled and partitioned in training and test subsets. Then validation results are averaged over the rounds. In the second method (CV-2), to ensure that the model would be tested with truly *unseen* data, we manually divided the dataset into 10 different subsets, where each test subset is composed of four different games with five levels. We then perform the 10-fold cross validation and in each round the model was trained in 37 games ( $37 \times 5 = 185$  levels/instances) and tested in 4 ( $4 \times 20$  levels/instances) completely unseen games.

### G. Online playing

The hyper-agent works following a sequence of three steps: feature collection, algorithm classification and algorithm selection (Figure 1). In feature collection, the hyper-agent uses the state observation of the game to collect the selected features. Then it normalizes them using the same filter as in the offline training. This creates a dataset similar to the one used for training the model. In algorithm classification, it uses the trained model to predict the best controller to play the current game. Finally in algorithm selection, it uses the output decision and loads the best controller. This controller will play the rest of the game, returning actions using its own defined strategy.

### H. Experiments

To collect data and compare the proposed hyper-agent with those submitted to the GVGAI competition, we use the same software, scoring method and ranking criteria [4]. All the levels for each game are played five times by the seven controllers in our portfolio and by two variations of hyper-agents. This represents a total of 1,025 game plays by each controller/agent and an overall total of almost 10,000 game plays. The experiments were performed locally using the framework available on the competition website.

Three measures were applied to the experiments: victories, score and time. Since it is more important to win the game



TABLE I

NON-TRANSITIVITY FOR VICTORIES - FIVE LEVELS FOR EACH GAME ARE PLAYED FIVE TIMES AND WE SUM THE WIN RATE FOR EACH AGENT. THE INTENSITY OF THE CELLS IS PROPORTIONAL TO THE RESULTS AND THEY INDICATE THAT DIFFERENT AGENTS EXCEL AT DIFFERENT GAMES, SHOWING THE NON-TRANSITIVE BEHAVIOR OF OUR SET OF AGENTS WITH REGARDING TO VICTORIES.

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12	G13	G14	G15	G16	G17	G18	G19	G20	G21
MCTS	25	1	23	0	5	11	4	6	12	2	1	0	0	22	1	3	0	12	23	1	1
OLMCTS	25	1	23	0	5	16	1	7	7	4	2	0	4	20	0	5	0	11	25	3	0
GA	25	1	23	0	6	13	3	6	12	6	0	0	7	19	0	3	0	11	21	0	2
SOS	25	1	23	0	1	14	4	6	13	1	1	0	1	21	0	3	0	14	20	0	2
YB	25	7	25	11	24	22	21	21	13	25	25	1	25	25	6	24	2	1	24	12	20
ACTX	25	15	25	14	6	10	25	4	10	25	5	0	5	25	0	23	14	18	25	1	3
JJ	12	0	25	8	20	25	0	3	10	13	8	0	10	23	2	17	0	9	6	1	1
	G22	G23	G24	G25	G26	G27	G28	G29	G30	G31	G32	G33	G34	G35	G36	G37	G38	G39	G40	G41	Total
MCTS	0	2	3	1	3	0	19	2	0	9	14	1	17	0	0	0	6	25	1	2	258
OLMCTS	0	6	3	0	4	0	14	0	0	6	19	0	10	0	0	0	5	25	0	1	252
GA	0	2	1	0	4	0	19	2	0	5	15	2	16	0	0	0	5	25	5	3	262
SOS	0	3	3	0	4	0	18	4	0	5	14	2	13	0	0	0	5	25	2	3	251
YB	0	9	10	25	20	25	24	25	25	24	10	1	23	0	14	10	10	25	10	5	654
ACTX	0	22	0	2	5	13	4	25	0	6	25	0	22	0	0	18	0	25	2	6	453
JJ	0	0	3	1	6	5	1	17	0	4	5	7	25	0	0	0	25	25	5	5	327

TABLE II

NON-TRANSITIVITY FOR SCORES - IN THE GVGAI FRAMEWORK, DIFFERENT GAMES HAVE DIFFERENT SCORES. IN ORDER TO COMPARE THEM, RESULTS ARE NORMALIZED SO THAT THE ALGORITHM WITH THE HIGHEST SCORE RECEIVES 1, THE LOWEST RECEIVES 0 AND THE OTHERS ARE RANKED BETWEEN THOSE VALUES. SIMILAR TO THE VICTORIES, ALGORITHMS ALSO PRESENT NON-TRANSITIVE BEHAVIOR FOR SCORES.

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12	G13	G14	G15	G16	G17	G18	G19	G20	G21
MCTS	0.23	0.17	0.61	0.41	0.18	0.27	0.16	0.2	0.3	0.14	0.04	0.77	0.06	0.08	0.25	0.22	0.49	0.11	0.14	0.1	0.08
OLMCTS	0.32	0.19	0.48	0.46	0.25	0.35	0.04	0.37	0.02	0.12	0.08	0.66	0.39	0.09	0.23	0.14	0.62	0.1	0.31	0.31	0
GA	0.24	0.18	0.55	0.45	0.2	0.27	0.12	0.25	0.28	0.34	0	0.47	0.34	0.09	0.16	0.15	0.52	0.14	0.14	0.05	0.14
SOS	0.13	0.17	0.43	0.02	0	0.3	0.16	0.28	0.32	0.07	0.04	0.33	0.04	0.11	0.18	0.21	0.39	0.21	0.14	0.05	0.14
YB	1	0.34	0.1	0.86	1	0.77	0.84	1	0.45	0.21	1	1	1	0.57	0.97	0.68	0.08	0.2	0.86	0.81	1
ACTX	0.14	0.92	0.11	0.94	0.45	0.25	1	0.2	0.99	1	0.2	0.53	0.51	1	0.5	0.76	1	0.78	0.67	0.06	0.24
JJ	0.58	0.17	0.14	0.92	0.82	0.87	0	0.2	0.73	0.57	0.32	0.58	0.23	0.49	0.81	0.53	0.7	0.64	0.11	0.2	0.16
	G22	G23	G24	G25	G26	G27	G28	G29	G30	G31	G32	G33	G34	G35	G36	G37	G38	G39	G40	G41	Total
MCTS	0.28	0.23	0.45	0.17	0.03	0.03	0.19	0.11	0.06	0.36	0.45	0.47	0.45	0.07	0.08	0.55	0.87	0	0.1	0.63	0.1
OLMCTS	0.22	0.19	0.31	0.09	0.14	0.06	0.21	0.03	0.12	0.24	0.28	0.55	0.16	0.05	0.04	0.52	0.69	0	0.14	0.28	0.07
GA	0.35	0.24	0.1	0.04	0.14	0.06	0.22	0.11	0.09	0.2	0.45	0.58	0.69	0.01	0.01	0.52	0.86	0	0.31	0.46	0.1
SOS	0.21	0.22	0.3	0.1	0.13	0.03	0.29	0.16	0.13	0.2	0.46	0.45	0.59	0.04	0.03	0	0.86	0	0.23	0.2	0
YB	0	0.34	0.7	1	1	1	1	1	1	1	0.47	0	0.32	0.15	0.8	0.77	0.92	0.79	1	0.98	1
ACTX	0.41	1	0.2	0.22	0.49	0.52	0.16	1	0.95	0.24	0.05	0.07	0.49	0.8	0.33	1	0.12	0.91	0.23	0.7	0.66
JJ	0.5	0.1	0.48	0.21	0.42	0.19	0.03	0.68	0.95	0.08	1	0.98	0.87	0	0.29	0.57	1	0	0.42	0.71	0.53

TABLE III

SELECTED FEATURES USED TO REPRESENT UNKNOWN GAMES.

Group	Feature	Description
Resources	gameHasResources	If there are resources in the game
	avatarHasResources	If the avatar has resources
	nTypeResourcesGame	Number of types of resources
	nTypeResourcesAvatar	Resources that the controller has
NPC	nNPC	Number of NPCs in the game
	nTypeNPC	Number of types of NPCs
Dimensions	Area	The area of the game
	blockSize	The number of pixels of a block
Sprites	nTypeImmovableSprites	Number of types of immovable Sprites
	nTypeMovableSprites	Number of types of movable Sprites
Portals	hasPortals	If the game has portals
	nTypePortals	Number of types of portals
Actions	MoveVertically	If the controller can move vertically
	CanShoot	If the controller can attack enemies

than fail with a high score, the GVGAI competition weighs the number of victories higher than the achieved score. The time measure is used only as a second tie-breaker because high score victories are weighted higher than fast wins.

#### IV. RESULTS

From the results presented in Table IV, we selected SVM and J48 to be selection models for the hyper-agent. SVM achieved the best accuracy in both validations and J48 was the second best in the second validation, which is the most important. Besides, it provides an understandable decision tree (Figure 2) that justifies its selection strategy. Although MLP had good results in the first validation, the second validation shows that model overfits the dataset and its performance is

inferior to ZeroR algorithm that just selects the majority class in the dataset.

##### A. Controller selection at start

With trained models from SVM and J48, we created two variations of the hyper-agents, HA-SVM and HA-J48. They both use the framework for online playing, but differ in the model selected to perform algorithm classification.

To show that the hyper-agents significantly differ from the

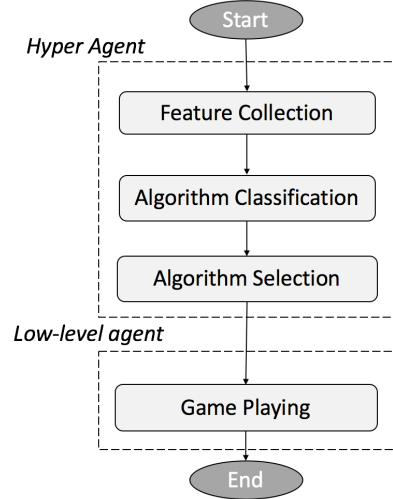


Fig. 1. The framework with steps for online playing.



TABLE IV

CLASSIFICATION ACCURACY FOR THE MACHINE LEARNING ALGORITHMS USING NORMAL CROSS-VALIDATION (CV-1) AND DOMAIN-AWARE CROSS-VALIDATION (CV-2).

Algorithms	Accuracy	
	CV-1	CV-2
SVM	0.7125	<b>0.6373</b>
MLP	0.6976	<b>0.4801</b>
J48	0.6878	<b>0.5936</b>
LR	0.6023	<b>0.5201</b>
ZeroR	0.5042	<b>0.5042</b>

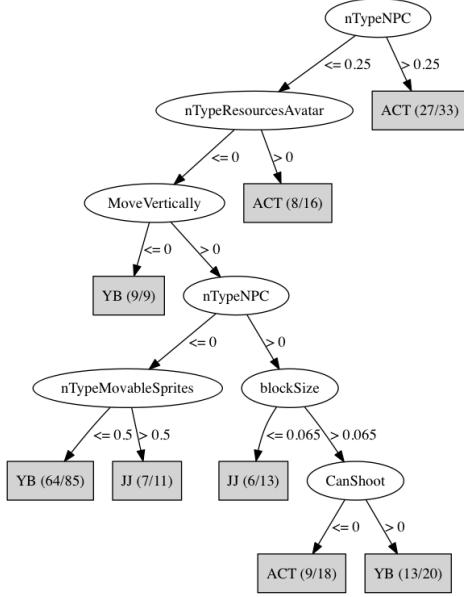


Fig. 2. J48 decision tree using features with high information gain. For each leaf, it shows the controller selected and the number of correctly classified instances out of the total instances reaching the leaf.

other algorithms, we compare their distributions through a statistical test using the a Mann-Whitney-Wilcoxon (MWW) approach [47]. Table V shows the p-values for victories and scores for the hyper-agents and the competition controllers. If the p-value is small ( $p < 0.05$ ), it is possible to reject the null hypothesis, which in this case says that difference is due to random sampling, and conclude instead that the populations are indeed distinct. The victories and scores for HA-J48, and victories for HA-SVM, are proven to be significantly different from the others. However, although HA-SVM achieved higher scores than YB, the null hypothesis cannot be entirely discarded because the p-value between the two distributions is 0.08.

In Table VI we show general results for the controllers and an illustration that visually shows the superior performance of the hyper-agents in victories, scores and time, over standard and competition algorithms. To compare the controllers in each game, we sum the wins for each level as in Table I and average the values for score and time, as in Table II. We consider that a controller dominates a game if it wins more than 3 levels of that game. In Table VII, the total results show that HA-SVM dominates 28 games and HA-J48 25, achieving first and

TABLE V

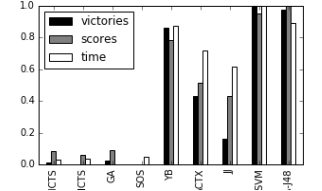
P-VALUES FROM COMPARING THE VICTORIES AND SCORES DISTRIBUTIONS FOR HYPER AND LOW AGENTS USING MWW TEST. IF P-VALUE  $< 0.05$ , THE NULL HYPOTHESIS CAN BE DISCARDED.

Victories	YB	ACT	JJ	Scores	YB	ACT	JJ
HA-SVM	0.014	0.000	0.000	HA-SVM	0.084	0.000	0.000
HA-J48	0.035	0.000	0.000	HA-J48	0.023	0.000	0.000

TABLE VI

LEFT - OVERALL RESULTS WITH TOTAL NUMBER OF VICTORIES AND NORMALIZED VALUES FOR SCORE AND TIME. TIME IS INVERSELY NORMALIZE SO THAT VALUES CLOSER TO 1 MEANS FASTER CONTROLLERS. RIGHT - VISUAL REPRESENTATION WITH ALL METRICS NORMALIZED.

	wins	scores	time
HA-SVM	721	0.95	1.00
HA-J48	709	1.00	0.89
YB	654	0.78	0.87
ACTX	453	0.51	0.72
JJ	327	0.43	0.62
GA	262	0.09	0.00
MCTS	258	0.09	0.03
OLMCTS	252	0.06	0.04
SOS	251	0.00	0.05



second place respectively. When not considering the hyper-agents, YOLOBOT is the best in 21 games, adrienctx in 14 and JinJerry in 6. Standard algorithms performed well in some levels, but none of them dominated any game.

## V. DISCUSSION

The proposed algorithm selection approach uses features to predict the best algorithm for an unseen game. We selected features that were available in the framework and exposed to the controller through the API.

As in many machine learning problems, having good features with high information gain plays an important role in the performance of the classifiers. To better understand our agent, we used a J48 algorithm to see the decisions that the agent makes based on the features available. As expected, features like CanShoot and MoveVertically are present in the selection but with less importance than we anticipated. Since they have good information to differentiate types of games (for humans at least), we thought they would appear in the first or second level of the tree.

With regard to the controllers, the classifier focus on the three "safe" high performance controllers, which suggests high exploitation. The downside is that other controllers in our portfolio are not being used. To balance the trade-off and improve explorations, we increase the tree depth. Although the new tree shows more different controllers being selected, neither the accuracy of the model or the performance of the agent is significantly improved, whereas the complexity of the tree is.

We believe that is happens because if the model selects one of the three high performance controllers, it has a good chance to perform well even if it does not pick the best. On the other hand, if it wrongly selects a "risk" low performance controller, the likelihood of failing in the game is much higher.

TABLE VII

EACH ROW IS A GAME. IN THE LEFT THREE COLUMNS, THE BEST AGENT IS ASSIGNED WITH A CHECK MARK IF IT HAS BEEN SELECTED BY ONE OF THE HYPER-AGENTS, OR WITH A X OTHERWISE. IN THE RIGHT TWO COLUMNS, THE CHECK MARK INDICATES THAT HYPER-AGENT SELECTED THE BEST LOW-AGENT FOR THE GAME.

Games	YB	ACT	JJ	HA-SVM	HA-J48
aliens (G1)	✓			✓	✓
boulderdash (G2)		✓		✓	✓
butterflies (G3)				✓	✓
chase (G4)		X			
frogs (G5)	✓			✓	✓
missilecommand (G6)			X		
portals (G7)		✓		✓	✓
sokoban (G8)	X				
survivezombies (G9)	X				
zelda (G10)		X			
camelrace (G11)	✓			✓	✓
digdug (G12)	✓			✓	✓
firestorms (G13)	✓			✓	✓
infection (G14)		✓		✓	✓
firecaster (G15)	X				
overload (G16)	✓			✓	✓
pacman (G17)		✓		✓	✓
seaquest (G18)		✓		✓	
whackamole (G19)		X			
eggomania (G20)	✓			✓	
bait (G21)	✓			✓	✓
boloadventures (G22)			X		
boulderchase (G23)		X			
brainman (G24)	✓			✓	
catapults (G25)	X				
chipschallenge (G26)	✓			✓	
escape (G27)	✓			✓	✓
jaws (G28)	✓			✓	✓
labyrinth (G29)	✓			✓	✓
lemmings (G30)	✓			✓	✓
modality (G31)	✓			✓	✓
painter (G32)		✓		✓	✓
plants (G33)			✓	✓	✓
plaqueattack (G34)			✓	✓	✓
realportals (G35)		X			
realsokoban (G36)	✓			✓	✓
roguelike (G37)		✓		✓	✓
solarfox (G38)			✓	✓	✓
surround (G39)		X			
thecitadel (G40)	✓			✓	✓
zenpuzzle (G41)		✓			✓
<b>Total</b>	<b>21</b>	<b>14</b>	<b>6</b>	<b>28</b>	<b>25</b>

This trade-off is important to improve the performance our agent because the main advantage of algorithm selection is to use of the strengths of different algorithms in the portfolio. To achieve this, we need to improve the performance of the classifier to select the low performance controllers with higher accuracy.

By extracting and selecting better features, it is likely that the performance of the classifiers could be increased by improving the classification accuracy of the model. In particular, it might be possible to define features based on the first few time frames of game play, including observations on the behavior of moving sprites, spawning sprites etc. It could also be possible to define features that rely on longer observations of the game, and which change during gameplay,

making it possible to dynamically switch between controllers. The construction of features that are not dependent on the API of the GVGAI software will also increase the generality of the method. Another important step is to incorporate a feature selector system to automatically extract different features and train the model more dynamically.

Another important step for this project is to increase the number of games used. While we use cross-validation and statistical significance testing to validate the superior performance of the hyper-agents over its portfolio, having more games would help us to improve the strategy and understand the generality of our approach. That would also make it easier to compare our method with other standard-of-art methods in algorithm selection that uses larger datasets. One possible solution for this, is to *generate* games to train and test on. Initial attempts to generate games in the GVGAI framework are promising, and it is at least possible to automatically generate variations of existing ones [48].

## VI. CONCLUSIONS

This paper has presented an algorithm selection approach to general video game playing in the GVGAI framework. Based on the observation that performance in this domain is non-transitive, a hyper-agent based on several well-playing controllers was constructed. A number of features available to the controllers were extracted and classification models were trained to predict the best controller on each game. These models were used in the hyper-agent to select the best controller to play unknown games. Testing the hyper-agent on unseen games showed that it significantly outperformed the winners of the 2014 and 2015 competitions. These encouraging results suggest that the use of hyper-heuristics and algorithm selection have an important role in general video game playing and related tasks and there are many possibilities for improvement.

For future work, we intent to develop the feature selection step, improve the size of the dataset potentially generating more games automatically and we want to continue exploring the application of hyper-heuristics in general video game playing. We believe that methods with online learning can perform well in this type of problem due to the many changes that happen during game play. We also want to investigate the use of heuristic generation in this the domain to not only select, but also create high performance algorithms.

## ACKNOWLEDGMENT

Andre Mendes acknowledges financial support from CNPq Scholarship and Science Without Borders program.

## REFERENCES

- [1] S. Legg and M. Hutter, "Universal intelligence: A definition of machine intelligence," *Minds and Machines*, vol. 17, no. 4, pp. 391–444, 2007.
- [2] B. Goertzel and C. Pennachin, *Artificial general intelligence*. Springer, 2007, vol. 2.
- [3] M. Genesereth, N. Love, and B. Pell, "General game playing: Overview of the aaai competition," *AI magazine*, vol. 26, no. 2, p. 62, 2005.
- [4] D. Perez, S. Samothrakakis, J. Togelius, T. Schaul, S. Lucas, A. Couetoux, J. Lee, C. Lim, and T. Thompson, "The 2014 general video game playing competition," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

- [5] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton *et al.*, "A survey of monte carlo tree search methods," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 1, pp. 1–43, 2012.
- [6] D. Perez, S. Samothrakis, S. Lucas, and P. Rohlfshagen, "Rolling horizon evolution versus tree search for navigation in single-player real-time games," in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM, 2013, pp. 351–358.
- [7] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches," in *Handbook of metaheuristics*. Springer, 2010, pp. 449–468.
- [8] P. Cowling, G. Kendall, and E. Soubeiga, *A Hyperheuristic Approach to Scheduling a Sales Summit*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 176–190.
- [9] L. Kotthoff, "Algorithm selection for combinatorial search problems: A survey," *AI Magazine*, vol. 35, no. 3, pp. 48–60, 2014.
- [10] R. Vilalta and Y. Drissi, "A perspective view and survey of meta-learning," *Artificial Intelligence Review*, vol. 18, no. 2, pp. 77–95, 2002.
- [11] K. A. Smith-Miles, "Cross-disciplinary perspectives on meta-learning for algorithm selection," *ACM Computing Surveys (CSUR)*, vol. 41, no. 1, p. 6, 2009.
- [12] T. G. Dietterich, "Ensemble methods in machine learning," in *Multiple classifier systems*. Springer, 2000, pp. 1–15.
- [13] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, "Hyper-heuristics: a survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, no. 12, pp. 1695–1724, 2013.
- [14] J. R. Rice, "The algorithm selection problem," 1975.
- [15] P. Ross, S. Schulenburg, J. G. Marín-Blázquez, and E. Hart, "Hyper-heuristics: Learning to combine simple heuristics in bin-packing problems," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '02. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002, pp. 942–948. [Online]. Available: <http://dl.acm.org/citation.cfm?id=646205.682617>
- [16] P. Garrido and M. C. Riff, "Dvrp: A hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic," *Journal of Heuristics*, vol. 16, no. 6, pp. 795–834, Dec. 2010. [Online]. Available: <http://dx.doi.org/10.1007/s10732-010-9126-2>
- [17] J. Maturana, F. Lardeux, and F. Saubion, "Autonomous operator management for evolutionary algorithms," *Journal of Heuristics*, vol. 16, no. 6, pp. 881–909, 2010. [Online]. Available: <http://dx.doi.org/10.1007/s10732-010-9125-3>
- [18] S. Remde, P. Cowling, K. Dahal, N. Colledge, and E. Selensky, "An empirical study of hyperheuristics for managing very large sets of low level heuristics," *Journal of the Operational Research Society*, vol. 63, no. 3, pp. 392–405, 2012. [Online]. Available: <http://dx.doi.org/10.1057/jors.2011.48>
- [19] K. Chakhlevitch and P. Cowling, "Hyperheuristics: recent developments," in *Adaptive and multilevel metaheuristics*. Springer, 2008, pp. 3–29.
- [20] J. C. Ortiz-Bayliss, E. Özcan, A. J. Parkes, and H. Terashima-Marín, "Mapping the performance of heuristics for constraint satisfaction," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE, 2010, pp. 1–8.
- [21] E. López-Camacho, H. Terashima-Marín, P. Ross, and M. Valenzuela-Rendón, "Problem-state representations in a hyper-heuristic approach for the 2d irregular bpp," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. ACM, 2010, pp. 297–298.
- [22] G. Ochoa, J. A. Vázquez-Rodríguez, S. Petrovic, and E. Burke, "Dispatching rules for production scheduling: a hyper-heuristic landscape analysis," in *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*. IEEE, 2009, pp. 1873–1880.
- [23] N. R. Sabar, M. Ayob, R. Qu, and G. Kendall, "A graph coloring constructive hyper-heuristic for examination timetabling problems," *Applied Intelligence*, vol. 37, no. 1, pp. 1–11, 2012.
- [24] J. Li and G. Kendall, "A hyper-heuristic methodology to generate adaptive strategies for games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. PP, no. 99, pp. 1–1, 2015.
- [25] A. Elyasaf, A. Hauptman, and M. Sipper, "Evolutionary design of freecell solvers," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 4, pp. 270–281, Dec 2012.
- [26] S. Salcedo-Sanz, J. Matías-Román, S. Jiménez-Fernández, A. Portilla-Figueras, and L. Cuadra, "An evolutionary-based hyper-heuristic approach for the jawbreaker puzzle," *Applied intelligence*, vol. 40, no. 3, pp. 404–414, 2014.
- [27] A. Benbassat, A. Elyasaf, and M. Sipper, "More or less? two approaches to evolving game-playing strategies," in *Genetic Programming Theory and Practice X*. Springer, 2013, pp. 171–185.
- [28] L. Pulina and A. Tacchella, "A self-adaptive multi-engine solver for quantified boolean formulas," *Constraints*, vol. 14, no. 1, pp. 80–116, 2009.
- [29] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Satzilla: portfolio-based algorithm selection for sat," *Journal of Artificial Intelligence Research*, pp. 565–606, 2008.
- [30] A. S. Fukunaga, "Automated discovery of local search heuristics for satisfiability testing," *Evolutionary Computation*, vol. 16, no. 1, pp. 31–61, 2008.
- [31] E. OMahony, E. Hebrard, A. Holland, C. Nugent, and B. OSullivan, "Using case-based reasoning in an algorithm portfolio for constraint solving," in *Irish Conference on Artificial Intelligence and Cognitive Science*, 2008, pp. 210–216.
- [32] A. Gerevini, A. Saetti, and M. Vallati, "An automatically configurable portfolio-based planner with macro-actions: Pbp," in *ICAPS*. Citeseer, 2009.
- [33] J. Demmel, J. Dongarra, V. Eijkhout, E. Fuentes, A. Petit, R. Vuduc, R. C. Whaley, and K. Yelick, "Self-adapting linear algebra algorithms and software," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 293–312, 2005.
- [34] S. Bhowmick, V. Eijkhout, Y. Freund, E. Fuentes, and D. Keyes, "Application of machine learning to the selection of sparse linear solvers," *Int. J. High Perf. Comput. Appl.*, 2006.
- [35] A. S. Fukunaga, "Genetic algorithm portfolios," in *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, vol. 2. IEEE, 2000, pp. 1304–1311.
- [36] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *J. Artif. Int. Res.*, vol. 47, no. 1, pp. 253–279, May 2013. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2566972.2566979>
- [37] C. E. Brodley, "Addressing the selective superiority problem: Automatic algorithm/model class selection," in *Proceedings of the Tenth International Conference on Machine Learning*, 1993, pp. 17–24.
- [38] F. Frydenberg, K. R. Andersen, S. Risi, and J. Togelius, "Investigating mcts modifications in general video game playing," in *Computational Intelligence and Games (CIG), 2015 IEEE Conference on*. IEEE, 2015, pp. 107–113.
- [39] M. Ebner, J. Levine, S. M. Lucas, T. Schaul, T. Thompson, and J. Togelius, "Towards a video game description language," 2013.
- [40] L. Hong and S. E. Page, "Groups of diverse problem solvers can outperform groups of high-ability problem solvers," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 101, no. 46, pp. 16 385–16 389, 2004.
- [41] A. Weinstein and M. L. Littman, "Bandit-based planning and learning in continuous-action markov decision processes," in *ICAPS*, 2012.
- [42] D. Robles and S. Lucas, "A simple tree search method for playing ms. pac-man," in *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, Sept 2009, pp. 249–255.
- [43] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.
- [44] H. Guo and W. H. Hsu, "A learning-based algorithm selection meta-reasoner for the real-time mpe problem," in *AI 2004: Advances in Artificial Intelligence*. Springer, 2004, pp. 307–318.
- [45] L. Pulina and A. Tacchella, "A multi-engine solver for quantified boolean formulas," in *Principles and Practice of Constraint Programming—CP 2007*. Springer, 2007, pp. 574–589.
- [46] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, 2011.
- [47] M. P. Fay and M. A. Proschan, "Wilcoxon-mann-whitney or t-test? on assumptions for hypothesis tests and multiple interpretations of decision rules," *Statistics surveys*, vol. 4, p. 1, 2010.
- [48] T. S. Nielsen, G. A. Barros, J. Togelius, and M. J. Nelson, "Towards generating arcade game rules with vgdl," in *Computational Intelligence and Games (CIG), 2015 IEEE Conference on*. IEEE, 2015, pp. 185–192.

# Using Association Rule Mining to Predict Opponent Deck Content in Android: Netrunner<sup>†</sup>

Nick Sephton\*, Peter I. Cowling\*, Sam Devlin\*, Victoria J. Hodge\*, and Nicholas H. Slaven<sup>†</sup>

\*York Centre for Complex Systems Analysis, Department of Computer Science, University of York, United Kingdom

Email: njs523@york.ac.uk, peter.cowling@york.ac.uk, sam.devlin@york.ac.uk, victoria.hodge@york.ac.uk

<sup>†</sup>Stainless Games, Isle of Wight, United Kingdom

Email: nicks@stainlessgames.com

**Abstract**—As part of their design, card games often include information that is hidden from opponents and represents a strategic advantage if discovered. A player that can discover this information will be able to alter their strategy based on the nature of that information, and therefore become a more competent opponent. In this paper, we employ association rule-mining techniques for predicting item multisets, and show them to be effective in predicting the content of Netrunner decks. We then apply different modifications based on heuristic knowledge of the Netrunner game, and show the effectiveness of techniques which consider this knowledge during rule generation and prediction.

## I. INTRODUCTION

A variety of games often include incomplete or hidden information as a form of challenge to the players, indeed most such games would be far more trivial if such an element was excluded. Card games in which players bring decks of their own construction to play are now relatively common place, and are represented both in physical card gaming (e.g. Magic: The Gathering<sup>1</sup>), and in digital gaming (e.g. Blizzard Entertainment's Hearthstone<sup>2</sup>). In such games, knowledge of the content of an opponent's deck represents a potentially powerful strategic knowledge which can be exploited to significant advantage. This is true of competition outside of the game domain also, as being able to adequately predict the strategy of a potential competitor will likely give significant advantage.

In this paper we consider a deck of cards to be a multiset consisting of a known number of cards, each of which has a type identifier. We then use a variety of rule-mining techniques applied with heuristic knowledge to attempt to predict the content of the deck after observing a specific number of cards chosen at random. It is important to note also that our game of choice is sufficiently complex, such that constructing a deck in the manner a human might is substantially more difficult than prediction using any method we have attempted here. Human players generally construct decks by identifying a central idea for the deck, then fitting cards into the deck that either support that concept or appeal to the player in some other way. While our techniques here produce similar results, there is no clear identification of

concept, and all cards are connected, not selected for any other appeal.

This research could also be applied outside the realm of games, as this problem represents a highly complex, partially observable system with specific rules which govern the system construction. Optimising association rule mining to these complex requirements is clearly of interest as a general advancement of research in this area. The techniques here could easily be converted for use in other fields which have similar complex requirements on sets or multisets, simply by applying heuristic knowledge to data mining and rule generation processes as performed here.

The remainder of this paper is organised as follows. In section 2, we present a summary of related works on Rule Association Mining and other relevant techniques. Section 3 discusses the Android: Netrunner game which was the main focus for this work. In section 4, we discuss our experimental methods, the methods we used to generate association rules, and also the algorithms which we used to make deck predictions. In section 5, we present our results, and section 6 contains our conclusions and some notes on potential future work.

## II. RELATED WORK

The prediction of an opponent deck is effectively a form of opponent modelling [1], [2], [3], except with the important distinction that we are modelling strategic decisions which took place before the game started. As the opponent can't change their pre-game behaviour due to game experience, we do not need to create a full opponent model, only an estimation of actions which have already been performed. There has been little work in this specific area before, with the exception of a single application of machine learning to the game of Hearthstone [4], which achieved a very high prediction rate on a limited card set.

### A. Association Rule Mining

*Association Rule Mining* is the determination of correlations between a set of items [5]. It is also known as *Market-Basket Analysis*, due to the common usage of determining which products a shopper may purchase based on what is already in their shopping basket. A typical rule-mining algorithm functions by generating rules that describe which items are likely to be included in a partially observed set,

<sup>1</sup><http://magic.wizards.com/>

<sup>2</sup><http://us.battle.net/hearthstone/en/>

given the items in the observable part of the set. Itemsets are drawn from the data such that each itemset describes a correlation between items. Association rule mining is employed in many application areas, including intrusion detection [6], web usage mining [7] and bioinformatics [8].

A commonly used algorithm in association rule mining is *Apriori* [9]. Apriori first generates all 1-itemsets that appear in the data at least a number of times equal to a predetermined support value, then passes this generation onward to create a second generation of 2-itemsets. This process continues until an empty generation is found (that is a generation with no candidates that appear at least *support* times in the data.) Each generation member then creates a single association rule which describe the correlation recognised by that member.

There are many variations on the Apriori technique to generate rules [10]. Most notable of these are a technique which attempts to identify the n-most interesting itemsets for rule generation rather than using a minimum support value [11], [12]. Some techniques also use functional languages rather than support constraints [13], and others use lattice and graphing techniques [14].

### III. ANDROID: NETRUNNER

*Android: Netrunner* is a two-player strategy card game published by Fantasy Flight Games<sup>3</sup>, which includes elements of bluffing and deception. Netrunner is similar to other popular card games such as Magic:The Gathering, and is described as an LCG (Living Card Game [15]).

Due to the nature of the game, the content of an opponent's deck is critical strategy information, and a player who is able to accurately model their opponent's deck is at a substantial advantage. There are currently more than 600 cards released for Netrunner, so accurately modelling a deck is a significant challenge. The combination of the wide number of choices, plus the complex and specific rules for which cards may be included in decks makes Netrunner deck construction a highly intricate process.

During a standard match of Netrunner, opponents do not have access to the content of their opponents deck. Access to such information would provide a substantial advantage to a player, as they would both be able to predict their opponent's likely strategy, and also determine which strategies they are poorly defended against.

Netrunner has a well documented rules structure for deck building<sup>4</sup>. Each deck has a single identity card, which provides a *Side*, *Faction* and a certain amount of *Influence*. Decks may only include cards associated with their side, but may spend influence to include cards from other Factions.

Every Netrunner deck has exactly one *Identity* card which defines some rules for that deck, most notably a *Side*, an amount of *influence* and a *Faction*. There are exactly 2 sides (named *Runner* and *Corp*), and each card in Netrunner is associated with one side and cannot be included in decks associated with the other side. Identities which are from the

corp side must also include a specific number of *agenda points*, which are provided corp cards (the specifics of agenda points are not relevant to this work, other than to recognise that there is a required number of agenda points for some decks to include, which presents an additional restriction upon decks.) All non-identity cards also have a *Faction* and a *Influence Cost*, the latter of which describes the amount of influence which must be paid to include the card in a deck which contains an identity of a different faction.

In this paper, we consider a deck for Netrunner to be a multiset, where no item can appear in a set more than three times. Each set also includes exactly one identity, which is always visible to us (as this is a condition of beginning a game of Netrunner), and also defines a portion of the multiset rules.

## IV. EXPERIMENTATION METHODS

### A. Netrunner Deck Data

Experimentation data consisted of 6000 community made decklists posted on a popular Netrunner community website<sup>5</sup> that allows users to collect and compare decklists. Some filtered based on popularity was performed. Prediction accuracy results are determined by direct comparison of the predicted deck and the original deck and returning a percentage of the cards that match.

---

#### Algorithm 1 GetPredictedDeck(...) for $a_1$

---

```

1: function GETPREDICTEDDECK( $D_{obs}$ ,  $R$ ,  $C$ ,  $n$ )
2:
3:   ##Initialise all cards with rule support
4:   InitCardRuleCounts( $D_{obs}$ ,  $C$ ,  $R$ )
5:
6:   ##Sort cards desc by rule support
7:   sort( $C$ , rulecount, 0)
8:
9:   ##Set predicted deck to include observed deck
10:   $D_{pred} \leftarrow D_{obs}$ 
11:
12:   ##For each card
13:   for all  $c \in C$  do
14:
15:     ##Take the required number of cards
16:      $k = \min\{n - |D_{pred}|, c.MaxCount\}$ 
17:
18:     ##Add them to the predicted deck, if possible
19:      $D_{pred}.AppendMultiple(c, k)$ 

```

---

### B. Apriori Rule Generation

Rules were mined from data using the Apriori method detailed in Agrawal & Srikant [9], with modifications as detailed in sections below. This process generates a large number of *rules*, which describe the relationship between items in the analysed set. These rules are made up of one or more *antecedent* items, and one *consequent* item. The

<sup>3</sup><http://www.fantasyflightgames.com>

<sup>4</sup>[https://images-cdn.fantasyflightgames.com/filer\\_public/2e/66/2e66279a-0b5c-4d12-80b1-754289b5ff0c/adn01\\_rules\\_eng\\_lo-res.pdf](https://images-cdn.fantasyflightgames.com/filer_public/2e/66/2e66279a-0b5c-4d12-80b1-754289b5ff0c/adn01_rules_eng_lo-res.pdf)

<sup>5</sup><http://netrunnerdb.com>

antecedent items is a multiset of items which must be found in any observed set in order for the rule to become active. The consequent item is the item which results from rule activation, and thus the item which will be added to the predicted set. Our rules take the form  $\{A, B, B, C, D\} \rightarrow E$ , where  $A, B, B, C, D$  is the full set of antecedents, and  $E$  is the consequent.

Each rule also has a *support* [16] value, which states how many occurrences of the complete set of antecedents and the consequent appear in the training data, and is useful to describe the magnitude of the effect of the rule. Support is calculated by the formula  $support(X \rightarrow Y) = \sigma(X \cup Y)/N$  [17], where  $(X \rightarrow Y)$  represents a rule, and  $N$  represents the total size of the data set. Each rule also has a confidence value, which measures the reliability of the rule. Confidence is calculated by the formula  $confidence(X \rightarrow Y) = \sigma(X \cup Y)/\sigma(X)$ .

The primary piece of evidence used to model an opponent's deck will be the identity card, as it is always visible, and also provides the constraints for deck construction in the form of faction, side and influence. As other cards are revealed through play, these can be added to the deck with complete confidence. It is usual to have observed a small number of opponent cards during the first turn of play (we estimate 1-4 is usual), and as such we vary the number of observed cards we randomly select to determine the effectiveness of our technique upon different sized sets of cards.

After rules were generated from the data, the set of 6000 decklists were tested using 30 fold cross-validation, with each individual prediction being made based upon a set of randomly selected cards from the decks. As these cards could potentially be duplicates, for each test a minimum of two unique cards are observed.

### C. Apriori Prediction

1) *Standard Apriori Prediction ( $a_1$ )*: The standard Apriori method of prediction is shown in algorithm 1, where  $D_{obs}$  represents the observed known cards,  $n$  represents the size of the observed deck,  $D_{pred}$  represents the predicted deck,  $R$  represents the set of all generated rules, and  $C$  represents the set of all Netrunner cards. In the first step of the algorithm we set the rule counts of each card to 0, then we run through all rules and determine if they are active for the set of cards we have observed ( $D_{obs}$ ). We then set  $D_{pred}$  to contain  $D_{obs}$ , as our prediction will always include the cards we have observed, and this makes further operations easier. We sort all cards by their *rulecount* attribute, and then move through them in decending order of *c.rulecount* until we find sufficient cards to fill the remainder of  $D_{pred}$ .

2) *Modifying for duplicate cards ( $a_2$ )*: A notable error performed by  $a_1$  is number of duplicates which appear in the predicted decklists. As Netrunner decks can include up to three copies of each card<sup>6</sup>, we attempt a technique that

allows us to predict the number of copies of each item in the predicted multiset. Without this modification, the  $a_1$  simply adds the maximum number of each item until it cannot add more, resulting in three copies of each card in the predicted deck.

---

#### Algorithm 2 GetPredictedDeck(...) for $a_2$

---

```

1: function GETPREDICTEDDECK( $D_{obs}, R, C, n$ )
2:
3:   ##Initialise all cards with rule support
4:   InitCardRuleCounts( $D_{obs}, C, R$ )
5:
6:   ##Sort cards desc by rule support
7:   sort( $C, rulecount, 0$ )
8:
9:   ##Set predicted deck to include observed deck
10:   $D_{pred} \leftarrow D_{obs}$ 
11:
12:  ##For each card
13:  for all  $c \in C$  do
14:
15:    ##Take the required number of cards
16:     $k = \min\{n - |D_{pred}|, c.Cardinality\}$ 
17:
18:    ##Add them to the predicted deck, if possible
19:     $D_{pred}.AppendMultiple(c, k)$ 

```

---

In order to modify this behaviour, we make a separate calculation using the rule metadata to determine the number of duplicates included in the original data. We then use this information to include copies in the prediction. This algorithm is very similar to algorithm 1 except that after a card is selected, the rule metadata is averaged to determine the number of duplicates to be included.

Therefore each run of  $GetPrediction\_a_2(D_{obs})$  adds 1-3 cards to  $D_{obs}$ , and bans the included card from further selection. This technique may appear arbitrary, but in the case of duplication in a specific decklist, the nature of the individual card is far more relevant than any patterns between the card and other cards in that deck. For example, some cards are so strong and usable in any deck that they almost always appear in sets of 3, whereas others frequently appear alone due to the narrow field of use or difficulty to fit into a deck.

3) *Prioritising by Influence ( $a_3$ )*: A review of the all data used here shows that 84% of decks in our dataset used all of their influence, 92% used all except 1 point, and 95% used all but 2. Considering that our data likely contains a large number of casual decks, which likely accounts for those not using all of the influence, this is indicative of how important the concern of influence during deck construction.

In order to prioritise influence spends, we change the method of deck prediction so that we first attempt to make predictions which would spend all available influence (both influence and non-influence cards still undergo the duplicate procedure mentioned in section IV-C2 above.) This new method is not shown in algorithm, as the only change is

<sup>6</sup>A few cards have specific rules which break this allow more copies or restrict the number of duplicates, but the vast majority may only appear in sets of 1-3

a sorting  $C$  so that all of the rules with a resultant card that will cost influence appear first, and this is restated later in algorithm 4. Notation is as before, however in the set  $C$  is sorted not only by rulecount, but also by a boolean that represents whether including any given card in  $D_{pred}$  would cost influence. This means that the first predictions made by  $a_3$  will cost influence, and then when all the influence is spent, only cards that do not cost influence will be added.

4) *Using influence during Rule Generation ( $a_4$ ):* Here, we separate item sequences that were generated from influence spend and non-influence spend. This allows us to separate the item sets into two groups, one which represents cards which players have spent influence on, and which represents card sequences that were used “in-faction”. We can then generate specific rules for influence and non-influence spend. In the case that we had insufficient data, the prediction reverted to using all generated rules. This method is shown in algorithm 3. Notation is as before, however in addition  $R_{inf}$  represents rules originally generated from influence sets, and  $R_{noinf}$  represents rules which are generated from non-influence sets only. This algorithm is very similar to algorithm 2 except that *GetPrediction<sub>a4</sub>* uses only rules generated from influence selections when selecting an card that costs influence, and only rules generated from non-influence selections when selecting a card that doesn’t cost influence.

5) *Rule Generation including duplicate cards ( $a_5$ ):* We also attempted to remove the calculation for duplicate cards by allowing the rules to be constructed from duplicate items, and thus we should be able to predict those duplicates with more relevancy to the observed deck, rather than the general attributes of the cards. This algorithm is identical to algorithm  $a_4$ , except that duplicates are calculated based on the number of copies of each card seen in the generated rules rather than our cardinality data. When a rule is determined to be active, instead of checking rule metadata to determine the number of cards to add to the predicted deck, we instead determine the total number of the consequent item that already exist within the predicted deck, and if the required number specified by the rule already exist, we take no action. If the required number is not yet in the deck, we add a single consequent item. For example, if the rule  $\{A, B, C\} \rightarrow B$  becomes active, we check to see if 2 or more  $B$  are included in the predicted deck. If so, we add nothing. If not, we add a single  $B$ .

6) *Prioritising by rulesize ( $a_6$ ):* This modification attempts to give priority to rules which contain more items, as these rules will be less rarely active due to their specificity. However, when these rules are active for an observed card set, they will likely tell us more about the content of the deck than smaller rules. This algorithm is identical to  $a_4$ , except that the rules are sorted by descending rule size, and then  $a_4$  is performed using the set of rules which are the largest size, then descending through the rules until we have completed the deck.

7) *Making confident predictions ( $a_7$ ):* This modification is identical to  $a_6$ , however when we predict a card, we add it to

---

**Algorithm 3** GetPredictedDeck(...) for  $a_4$ 


---

```

1: function GETPREDICTEDDECK( $D_{obs}, R_{inf}, R_{noinf}, C,$ 
    $n$ )
2:
3:   ##Initialise all cards with rule support (inf)
4:   InitCardRuleCounts( $D_{obs}, C, R_{inf}$ )
5:
6:   ##Sort cards desc by rule support
7:   sort( $C, rulecount, 0$ )
8:
9:   ##Set predicted deck to include observed deck
10:   $D_{pred} \leftarrow D_{obs}$ 
11:
12:  ##Spend influence first
13:  for all  $c \in C$  do
14:
15:    ##Take the required number of cards
16:     $k = \min\{ \lfloor (maxinf - inf(D_{pred})) / c.inf \rfloor, c.Cardinality \}$ 
17:
18:    ##Add them to the predicted deck, if possible
19:     $D_{pred}.AppendMultiple(c, k)$ 
20:
21:  ##Initialise all cards with rule support (no inf)
22:  InitCardRuleCounts( $D_{obs}, C, R_{noinf}$ )
23:
24:  ##Then fill the deck with non-influence cards
25:  for all  $c \in C$  do
26:
27:    ##Take the required number of cards
28:     $k = \min\{ n - |D_{pred}|, c.Cardinality \}$ 
29:
30:    ##Add them to the predicted deck, if possible
31:     $D_{pred}.AppendMultiple(c, k)$ 

```

---

the observed card set and check all rules again. So any card we predict to appear in the deck, we assume we are correct for the purposes of further predictions. This final version is shown in algorithm 4.

## V. RESULTS

All results for predictions are shown in figure 3. Use of the mentioned techniques to generate deck predictions is generally successful, completing decks with an accuracy of up to 59% from viewing only 5 cards (roughly 8-10% of the actual deck). However there are some general trends which can be observed. Firstly, as each card (or set of cards) are added to the deck sequentially, we don’t take into account new patterns which may emerge between originally observed cards and cards more recently added. This means that all predictions are based on the original set of observed cards, whereas we would likely have a different effect on prediction if we considered predicted cards to be part of the observed set when making further predictions. We suggest that some of the difference in prediction may be a tendency to form into familiar deck archetypes, as predicted cards would likely support larger patterns already recognised as frequently



---

**Algorithm 4** GetPredictedDeck(...) for  $a_7$ 

---

```
1: function GETPREDICTEDDECK( $D_{obs}, R_{inf}, R_{noinf}, C, n$ )
2:    $D_{pred} \leftarrow D_{obs}$ 
3:   for all  $r \in R_{inf}$  do
4:
5:     ##Initialise all cards with inf rule support
6:     InitCardRuleCounts( $D_{pred}, C, R_{inf}$ )
7:
8:     ##Sort cards desc by rule support
9:     sort( $C, \text{rulecount}, 0$ )
10:
11:    ##Spend influence first
12:    for all  $c \in C$  do
13:
14:      ##Take the required number of cards
15:       $k = \min\{\lfloor (\text{maxinf} - \text{inf}(D_{pred}))/c.\text{inf} \rfloor, c.\text{Cardinality}\}$ 
16:
17:      ##Add them to the predicted deck, if possible
18:       $D_{pred}.\text{AppendMultiple}(c, k)$ 
19:    for all  $r \in R_{noinf}$  do
20:
21:      ##Initialise all cards with non-inf rule support
22:      InitCardRuleCounts( $D_{pred}, C, R_{noinf}$ )
23:
24:      ##Sort cards desc by rule support
25:      sort( $C, \text{rulecount}, 0$ )
26:
27:      ##Fill out deck with non-influence
28:      for all  $c \in C$  do
29:
30:        ##Take the required number of cards
31:         $k = \min\{n - |D_{pred}|, c.\text{Cardinality}\}$ 
32:
33:        ##Add them to the predicted deck, if possible
34:         $D_{pred}.\text{AppendMultiple}(c, k)$ 
35:  return  $D_{pred}$ 
```

---

played decks. This is somewhat consistent with human deck construction however, as players often use existing archetypes to construct decks.

In order to provide a control for experimentation, random selection was tested ( $a_0$ ). Generated decks were still required to observe deck construction rules, but other than that cards were selected randomly from the set of available cards. All predictions using  $a_0$  had an accuracy in the range 0% - 6%, and due to this low accuracy, results are not shown below.

We also attempted to test prediction across a range of different numbers of observed cards. In each of these cases, the identity card was always observed, then an additional number of cards were added. This means in the case of the number of observed cards being zero, only the identity card was observed. In all previous experiments the size of the set has been five, which represents what a player might expect from two complete turns of play. We tested prediction with sets of up to ten viewed cards. We also tested prediction with a set of zero observed cards, which represents the game

before play has begun.

#### A. Default Apriori ( $a_1$ )

Default Apriori allows for predictions of up to 48% accuracy, and while this is somewhat effective, it can be improved upon significantly by the later algorithms which incorporate heuristic knowledge. Different values of minimum support were used to determine the optimum value, which lies close to 15. All of these tests were run on a dataset of size 200 (30-fold cross-validation on a total set of size 6000), so larger values of minimum support will likely cause smaller detail of the dataset to be lost during rule generation. Examination of the decks generated with  $a_1$  also reveals that almost every card is included in triplets, further speaking of the necessity of a modification to address the number of duplicates included.

#### B. Apriori with duplicates ( $a_2$ )

The modification to consider inclusion of duplicates in the predicted deck results in a significant increase in accuracy. The most significant value of minimum support now appears between 10-15, both options resulting in a prediction accuracy of 53%, an increase in accuracy of 5%. This increase in accuracy is certainly related to more accurate predictions on sets of duplicate cards, as due to the nature of the game, certain cards are more often played in sets of 2 or 3, and certain cards are almost always played without duplicates. This modification largely makes the effect that there are no longer automatic inclusions of cards in groups of 3, however it can still be further improved with respect to heuristic data.

#### C. Apriori with Influence Priority ( $a_3$ )

While prioritising the inclusion of cards which cost influence has a positive effect, the effect is marginal, increasing prediction accuracy by less than ~2% at the optimal value of minimum support 10. It is surprising that the effect is so marginal, but upon examining further it is apparent that most (92%) of decks predicted with  $a_1$  and  $a_2$  already include the maximum permitted influence for those decks, so the modification is perhaps not as important to prediction as originally proposed.

Examinations of the individual card selections shows that the influence spends are somewhat inappropriate however, and are somewhat to blame for the inaccuracies of this prediction algorithm.

#### D. Apriori with Influence Filtering ( $a_4$ )

There are several interesting effects in these results. Firstly, the highest accuracy has risen to 57%, an increase of ~4%. Secondly, the optimal value of minimum support has changed to a higher value of 20.

A review of the cards selected by influence spends reveals that they are much more appropriate to the acknowledged deck archetypes, presumably due to the specific use of rules generated entirely from influence spend patterns.

We also start to observe some occasional single-card influence inclusions which are well established in the appropriate archetypes.

### E. Rule Generation including duplicate cards ( $a_5$ )

We can see from the results for  $a_5$  that attempting to determine the number of duplicate cards in a deck from generated rules appears to be less effective than using our data on the normal set count of that card. This is believable, as the number of duplicate cards included is likely to be much more dependent on the nature of the card than on the nature of the deck itself. As our information relates to patterns between cards, we don't necessarily have a good understanding of the nature of the card itself.

It is worth noting however that for some values of minimum support,  $a_5$  is approximately as effective as  $a_3$  and  $a_2$ , meaning that it is still an effective technique, and alternative methods to predict duplicate cards in the deck could be investigated.

### F. Prioritising by rulesize ( $a_6$ )

Giving priority to larger rules has also had a positive effect on prediction accuracy. We can see this effect particularly when minimum support is 20. We attribute this effect to larger rules being more rarely satisfied unless they are highly informative about the configuration of decks. As such, activated large rules should be given priority over activated smaller rules.

### G. Making confident predictions ( $a_7$ )

By adding all predictions to our observed set, we are assuming that all our predictions are correct, and biasing future predictions by this information. This has a positive effect on prediction accuracy at higher values of minimum support, however it has almost no effect at values of 15 and below. This could be explained by some subtlety of rules that are activated with a support of 15 or less, however in this case we would expect the prediction accuracy to be positively affected also, and yet we see that this is not the case.

The extension of our observed set also has another less obvious effect on prediction, which is that it allows activation of rules with larger item sequences, as more items appear in the observed set. This means as  $D_{obs}$  expands, we may observe decks activating larger rules, and effectively falling into archetypes.

### H. Varied Size Observation Set

The results for predictions made with varied observation sets are shown in figure 1. We can see that the overall change in deck prediction accuracy across the total range of tested values is approximately 20%, which while a large change, might be less than we expect from such a change in source data. This illustrates the importance of the identity card which is always viewed, it speaks deeply of the construction of the deck, mostly because the identity card is always active during play, and a substantial portion of the cards included will have some synergy with that identity. This also speaks of the nature of deck construction in Netrunner, which largely consists of modifications to existing archetypes, likely due to smaller synergies between groups of cards. It is also worth noting that at almost all values of observed set size and

minimum support, our algorithms which incorporate heuristic information perform significantly better than default apriori.

We see an understandable increase in prediction accuracy as we increase the size of the observed set, as there are both fewer cards to predict, and also more information is available on the set content. Rules with a higher number of antecedents are also activated, which likely provides more accurate information on the set content.

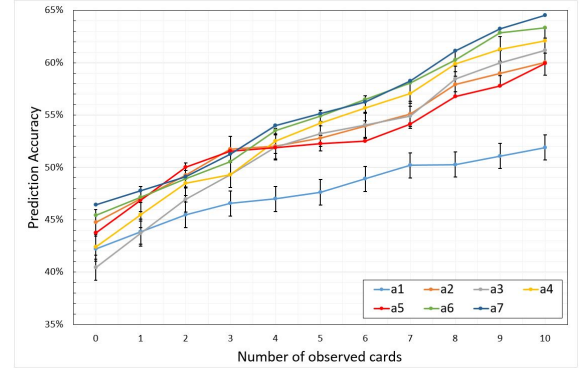


Fig. 1. Varied Size Observation Set

We can also observe that a few of our own techniques ( $a_3$  &  $a_4$ ) perform very poorly when the observed set is very small or empty. As  $a_3$  and  $a_4$  both focus on influence inclusions, this is likely due to a lack of corroborating information from other observed cards to distinguish correct influence selections. As such, the initial influence selections are almost unguided, and as these cards are selected from a much larger set of available cards than regular selections, the picks are more likely to be incorrect without guidance.

There is also an interesting plateau in prediction accuracy around set size 3-6 with algorithm  $a_5$ . This is likely due to the estimation for duplicate cards struggling on smaller set size. As the cards in the observed section of the deck are selected randomly during each test, it is possible that duplicate cards are selected, and as such less information is exposed in certain cases. This might cause a decrease in accuracy when only a small number of unique cards are observed. This calculation is not included in any other algorithm, as it was not effective in increasing accuracy overall, possibly due to this complication.

The results across all experiments grouped by algorithm are shown in figure 2. We can more clearly see a general rise in prediction accuracy here, with the exception of the  $a_5$  algorithm for reasons mentioned above. This is to be expected, as each algorithm following  $a_1$  includes specific heuristic improvements which are targeted to improve efficiency in this specific domain.

Algorithm  $a_5$  shows that our introduction of rule-based cardinality estimations have been unsuccessful in improving prediction efficiency, although this is something we would definitely want to address in future. The current cardinality estimations are unlikely to predict decks with 100% accuracy, for example it will always fail to predict decks that include a unusually small number of a card almost always seen in

sets of 3.

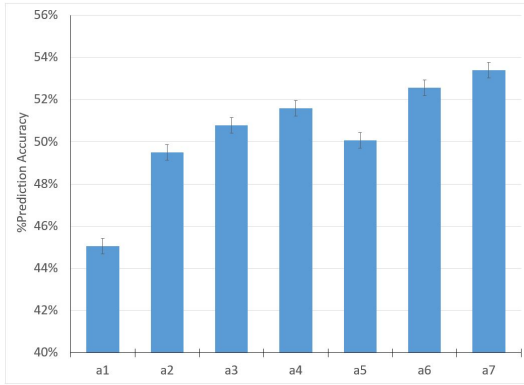


Fig. 2. Cumulative results by Algorithm

## VI. CONCLUSIONS & FURTHER WORK

It can be seen that our modifications to the Apriori technique provide a significant improvement to prediction of decks in Netrunner, showing a maximum improvement of ~13% between the default apriori algorithm ( $a_1$ ) and our optimal modified algorithm ( $a_7$ ).

There are several other opportunities for future work which could be explored. For example, the technique used to separate rules in  $a_4$  could also be applied to identity cards, using only rules generated for each identity to select either the entire deck, or the influence-spend portion of the deck. However this would require a large amount of data, as certain identities are unpopular and may appear only rarely within our current data set, so there would be fewer useful rules generated for these identities. It may also be worth looking at generation by *Faction*, which might yield more interesting results. Also, as our observed cards were randomly selected, they may not adequately represent the real order cards are observed during a game (as it is more common to play certain cards earlier than others.) Biasing generation of the observed set of card may provide a more realistic scenario.

Our attempts to adequately predict the number of duplicate cards within a deck have been some what effective, but there is still work to be done here, as our best prediction is based on our heuristic knowledge of the specific card, rather than knowledge of the card in context. Successfully adding contextual heuristic knowledge into this process will surely lead to more accurate prediction.

We can also look to applying these techniques to other domains, specifically the games mentioned in section 1. Magic the Gathering has a much larger set of active cards, and less stringent deck construction rules, so while this would represent a more challenging target, there is also a much larger amount of data available due to the larger player community and history of the game. Hearthstone likely represents a point of medium complexity, as the card pool is between the other two games mentioned here (approximately 450), and the deck construction rules are more restrictive than Magic, and thus provide more guidance.

A further avenue of research which could be pursue is that of pattern matching within the decks, in order to draw out common patterns which occur within multiple decks, and then using that information to further bias the prediction.

## ACKNOWLEDGEMENTS

The work displayed here was supported by EPSRC (<http://www.epsrc.ac.uk/>), the LSCITS program at the University of York (<http://lscits.cs.bris.ac.uk/>), the NEMOG program at the University of York (<http://www.nemog.org/>), and Stainless Games Ltd (<http://www.stainlessgames.com/>).

## REFERENCES

- [1] F. Southey, M. Bowling, B. Larson, C. Piccione, N. Burch, D. Billings, and C. Rayner, "Bayes' Bluff: Opponent Modelling in Poker," in *Proceedings of the TwentyFirst Conference on Uncertainty in Artificial Intelligence UAI*, 2005, pp. 550–558.
- [2] M. Ponsen, G. Gerritsen, and G. M. J.-B. Chaslot, "Integrating Opponent Models with Monte-Carlo Tree Search in Poker," in *Proc. Conf. Assoc. Adv. Artif. Intell.: Inter. Decis. Theory Game Theory Workshop*, no. February, 2010, pp. 37–42.
- [3] C. Bauckhage, C. Thureau, and G. Sagerer, "Learning human-like opponent behavior for interactive computer games," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 2781, pp. 148–155, 2003. [Online]. Available: <http://www.cs.berkeley.edu/daf/games/webpage/Alpapers/Bauckhage2003-LHL.pdf>
- [4] C. Bursztein and E. Bursztein, "I am a legend: Hacking Hearthstone with machine learning," in *DEFCON 22*, 2014, p. 169. [Online]. Available: <https://cdn.elie.net/talks/I-am-a-legend-defcon-22-slides-final.pdf>
- [5] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," *ACM SIGMOD Record*, vol. 22, no. May, pp. 207–216, 1993.
- [6] W. Lee, S. J. Stolfo, and K. W. Mok, "A data mining framework for building intrusion detection models," in *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, 1999, pp. 120–132.
- [7] J. Srivastava, R. Cooley, M. Deshpande, and P.-n. Tan, "Web usage mining: discovery and applications of usage patterns from Web data," vol. 1, no. 2, pp. 12–23, 2000.
- [8] C. Creighton and S. Hanash, "BIOINFORMATICS Mining gene expression databases for association rules," pp. 79–86, 2003.
- [9] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," *Proceeding VLDB '94 Proceedings of the 20th International Conference on Very Large Data Bases*, vol. 1215, pp. 487–499, 1994.
- [10] J. Hipp, U. Guntzer, and G. Nakhaeizadeh, "Algorithms for association rule mining — a general survey and comparison," *ACM SIGKDD Explorations Newsletter*, vol. 2, no. 1, pp. 58–64, 2000.
- [11] S. C. Ngan, T. Lam, R. C. W. Wong, and A. W. C. Fu, "Mining N-most interesting itemsets without support threshold by the COFI-tree," *International Journal of Business Intelligence and Data Mining*, vol. 1, no. 1, p. 88, 2005. [Online]. Available: <http://www.inderscience.com/link.php?id=7320>
- [12] A. W. C. Fu, R. W.-w. Kwong, and J. Tang, "Mining N-most Interesting Itemsets," in *Proceedings of the 12th International Symposium on Methodologies for Intelligent Systems (ISMIS)*, 2000.
- [13] Z. Hu, W. Chin, and M. Takeichi, "Calculating a new data mining algorithm for market basket analysis," *Practical aspects of declarative languages: second International Workshop, PADL 2000, Boston, MA, USA, January 17-18, 2000: proceedings*, pp. 169–185, 2000.
- [14] J. Hipp, A. Myka, R. Wirth, and U. Guntzer, "A New Algorithm for Faster Mining of Generalized Association Rules," in *Principles of Data Mining and Knowledge Discovery*, 2006, pp. 74–82.
- [15] S. C. Duncan, "Mandatory Upgrades: The Evolving Mechanics and Theme of Android: Netrunner," in *Well Played Summit*, 2014.
- [16] V. Baez-Monroy and S. O'Keefe, "An Associative Memory for Association Rule Mining," *2007 International Joint Conference on Neural Networks*, no. 2, pp. 2227–2232, 2007.
- [17] P.-N. Tan, M. Steinbach, and V. Kumar, "Association Analysis: Basic Concepts and Algorithms," *Introduction to Data mining*, pp. 327–414, 2005. [Online]. Available: <http://www-users.cs.umn.edu/~kumar/dmbook/index.php>

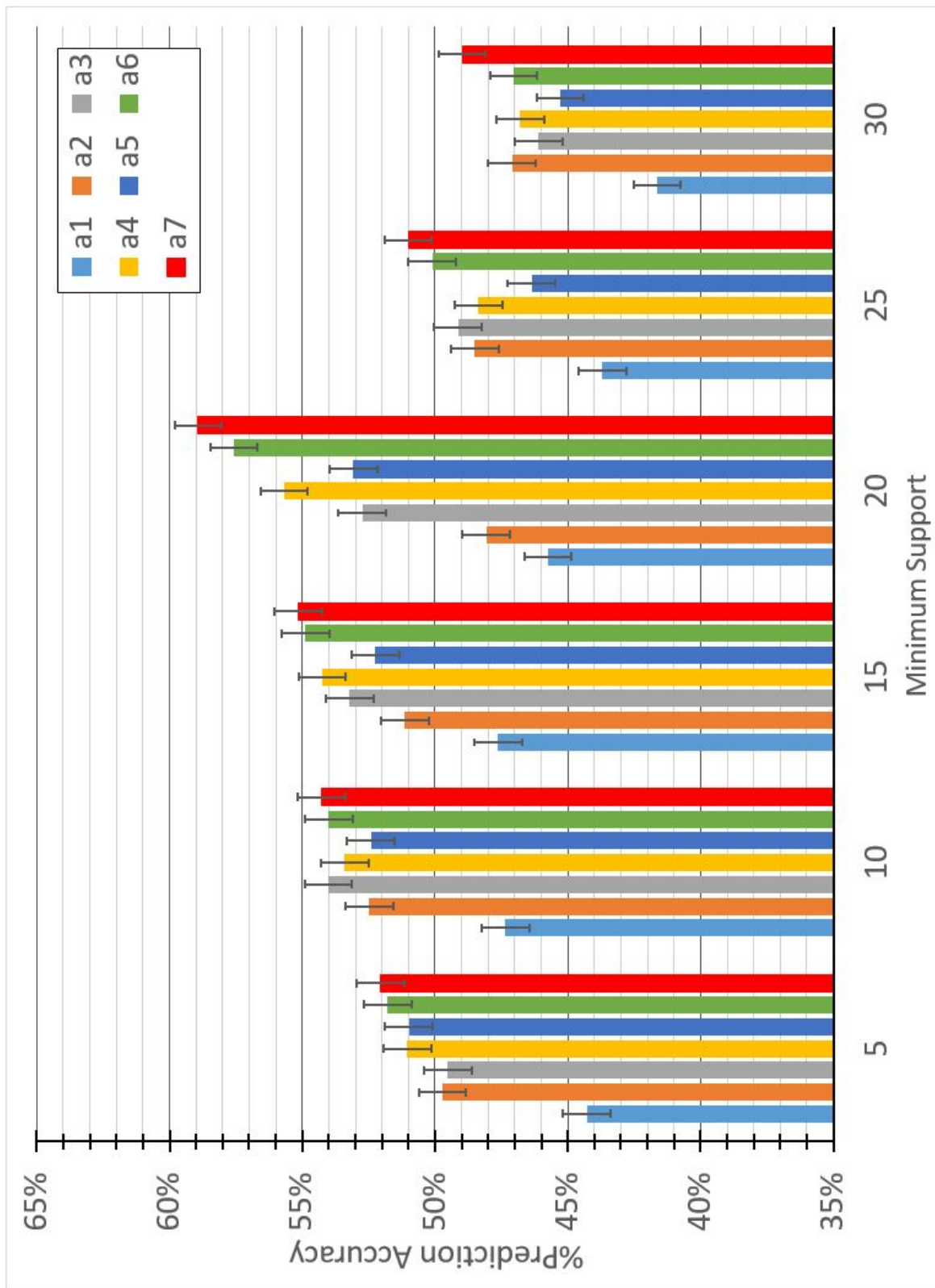


Fig. 3. Results of algorithm runs with varying minimum support values

# Planning Social Actions Through the Others' Eyes for Emergent Storytelling

David B. Carvalho, Esteban G. Clua and Aline Paes

Computer Science Institute

UFF - Federal Fluminense University

Niterói, Rio de Janeiro

Emails: {dbatista, esteban, alinepaes}@ic.uff.br

**Abstract**—Stories have become an important element of games, since they can increase their immersion level by giving the players the context and the motivation to play. However, despite the interactive nature of games, their stories usually do not develop considering every decision and/or action the players are capable of, because depending on the game size, it would take too much effort to author alternative routes for all of them. To make these alternatives viable, an interesting solution would be to procedurally generate them, which could be achieved by using the story generation approaches already developed by many works of the storytelling field. Some of these approaches are based on the simulation of virtual worlds, in which the stories are generated by making the characters that inhabit the worlds act trying to reach their goals. The resulting actions and the world's reactions compose the final story. Since the actions are the building blocks of the stories, the characters' acting capabilities are determinant features of the generation potential of simulations. For instance, it is only possible to generate stories with deception if the characters are capable of deceiving each other. To allow the generation of stories where the characters are capable of manipulation, cooperation and other social behaviors by actively using what the others will do based on what they know and see, we propose a recursive planning approach that deals with the uncertainty of the others' knowledge and with a purposely error-prone perception simulation. To test our proposal we developed a story generation system and designed an adaptation of Little Red Riding Hood world as test scenario. With our approach, the system was capable of generating coherent story variations with deceptive actions.

## I. INTRODUCTION

Presenting different story developments to a game player, as a consequence of each of his/her decisions and actions throughout the game has the potential of increasing the immersion level, because it generates the sensation that his/her actions matter. However, this task is usually impractical given the amount of authoring work it would require. To achieve that while still avoiding considerable human effort, a solution would be to automatically generate these developments. There are already many works that deal with the problem of story generation in the field of automated storytelling [1]–[3]. One of the main concerns of story generation is related to the level of freedom that should be given to the characters. Some works propose generation methods guided by restrictions that should be obeyed by the resulting stories, such as predefined middle and ending situations. Others propose the simulation of a virtual world without restrictions, where the story emerges as the characters that inhabit such a world act trying to reach their goals. These two approaches are called deliberative and simulation-based, respectively [4].

The first approach guarantees stories with reasonable quality, because they will follow the restrictions given by an author. However, this is achieved at the cost of variability, since stories that are not in accordance with the restrictions will not be created, and possibly at the cost of coherence on the characters' behavior, since they may be forced to act against their goals. On the other hand, the simulation approach allows more varied stories, where characters are not forced to execute specific actions. This last characteristic may be an interesting advantage in the scope of games, since the players usually control one of the characters, which means that they also would not be forced to choose their actions. However this lack of restrictions may jeopardize quality, given the little control of the development.

We believe that to increase the quality of stories generated by simulation, the characters still need a series of features of the human behavior that were not entirely explored by previous works. These features could make characters more believable, and act towards situations otherwise unreachable. One behavior that deserves more attention is the social behavior, i.e., to act considering the actions of the others. This behavior is essential to generate stories where a character helps, deceives or simply avoids the others. Since these are common human behaviors, they are present in a number of game stories. Just to name a few, in some horror games, e.g. *Clock Tower*<sup>1</sup> and *Outlast*<sup>2</sup>, the protagonist has to hide continuously to avoid being killed by a stalking antagonist. In the games of the *Hitman* series<sup>3</sup>, the protagonist is allowed to change his clothes to pretend to be part of the staff of restricted areas and have access to them.

To generate stories in which characters make mistakes and consider the others' mistakes, in previous works we developed a perception simulation that updates the characters' knowledge using a process that allows coherent errors regarding attributes and identities [5], and a recursive knowledge structure that represents uncertainty and the incorrect information the others [6] may have. In this work, we propose an enhancement for story generation by simulation, that uses a recursive Partially Observable Markov Decision Process (POMDP) [7] approach to simulate the characters' reasoning mechanism, along with a perception simulation, to allow the characters to plan their actions considering the others' mistakes. We aim at generating stories where each character may actively hide items, pretend to be another character and execute other social behaviors that require knowledge about the others' perceptions.

<sup>1</sup>Clock Tower originally by Human Entertainment

<sup>2</sup>Outlast by Red Barrels: <http://www.redbarrelsgames.com/>

<sup>3</sup>Hitman series by IO Interactive: <https://www.ioi.dk/>

The remainder of this paper is divided as follows: Section II presents related works that focus on making characters plan considering the others. Section III summarizes the world representation and story generation cycle. Section IV presents the proposed planning method. Section V details our test case scenario based on the Little Red Riding Hood, and discusses the generated stories. Finally, Section VI concludes this paper presenting opportunities of future work.

## II. RELATED WORKS

Actions planned considering the others as autonomous and goal-oriented entities were called “social actions” by Castelfranchi [8]. Later, Chang and Soo [9] defined “social planning” as an extension of the conventional planning that includes social actions. They developed a story generation process where the characters’ knowledge is structured to include the knowledge of the others in order to represent what one character knows about what the others know. The characters are capable of social planning using a forward planning method with actions that directly affect the others’ goals, and rules that describe what the others do when they adopt predefined goals.

This knowledge structure that includes the knowledge of the others recursively is derived from a psychology theory called theory of mind [10], which tries to explain how humans are capable of understanding and predicting the others’ behaviors.

Pearce et al. [11] also developed a generation process capable of social planning, but with a backward chaining method. In their approach, actions may be designed with effects that add goals to the others and information about the world to their knowledge. A plan is made adding actions of the planning character based on his goals and knowledge, and adding actions of the others based on the goals and knowledge that the planning character believes they have.

Thespian [12] is another system in which the characters are capable of social planning. The characters’ knowledge is structured to deal with uncertainty, i.e., they are aware that their knowledge is not necessarily correct. To plan considering uncertainty and the actions of the others, the reasoning process is made through a recursive POMDP. During the planning process, to define what another character will do, a recursive call to another planning process is made using the knowledge that the reasoning character believes that the other has. Then, the results of this recursive call are added to the main process as the other’s possible actions.

Alternatively to the planning approach, in which the social behavior emerge by making characters include the others’ actions, the Comme Il Faut system [13] uses actions described directly with social relationships as preconditions and effects called *social exchanges*. With them, a rule based reasoning process decides which social exchange each character must execute to reach the social conditions they want to achieve.

Our approach to simulate social behaviors is similar to Thespian’s proposal, using a recursive POMDP. However, while Thespian’s focus is on simulating emotions and normative social behaviors (social rules such as greet someone back or respond to a given question), ours is on allowing different kinds of coherent mistakes and making the characters actively plan how to cause and/or use the others’ mistakes as necessary.

## III. STORY GENERATION

This section summarizes the architecture used as basis for our simulation, previously proposed in [6], and the story generation process. It is important to make clear that in that work, we presented a previous structure and some changes for it. In this work we are already using the final structure.

### A. World Structure

A world  $W$  is defined as  $\langle M, E, O, C \rangle$ , where

- $M = \{l_1, l_2, \dots, l_l\}$  is a set of locations,
- $E = \{e_1, e_2, \dots, e_e\}$  is a set of events,
- $O = \{o_1, o_2, \dots, o_o\}$  is a set of objects,
- $C = \{c_1, c_2, \dots, c_c\}$  is a set of characters.

A location  $l_K$  is defined as:  $\langle Name, ID, N \rangle$ , where

- $Name$  is a string value,
- $ID$  is a tuple  $\langle ID_{original}, ID_{local} \rangle$  (details further in this section), where  $ID_{original}, ID_{local} \in \mathbb{Z}$ ,
- $N = \{l_1, l_2, \dots, l_{ln}\}$  is a set of neighbor locations.

An object  $o_K$  is defined as:  $\langle Name, ID, A_T \rangle$ , where

- $A_T = \{at_1, at_2, \dots, at_{at}\}$  is a set of attributes.

A character  $c_K$  is defined as:  $\langle Name, ID, A_T, A_c, B, K \rangle$ , where

- $A_c = \{a_1, a_2, \dots, a_a\}$  is the character’s set of actions,
- $B = \{b_1, b_2, \dots, b_b\}$  is a set of goal inference rules, which we call behaviors,
- $K = \{\langle w_0, P_0, G_0 \rangle, \dots, \langle w_n, P_n, G_n \rangle\}$  is the character’s knowledge base, where
  - $w_i$  is a world description,
  - $P_i$  is a probability that represents how much the character believes in  $w_i$ ,
  - $G_i$  is the set of goals that the character adopted in the context of  $w_i$ .

Actions, behaviors and events follow the Planning Domain Definition Language (PDDL) action pattern [14], which is described with sets of parameters, preconditions and effects. However, our behavior description has goal definitions instead of effects, has a set of abandon conditions and also a reward value. Behaviors are used to define how the character manages his goals: whenever the preconditions of a behavior become true, the character adopts the corresponding goal. Whenever the abandon conditions of a behavior that has an adopted goal become true, the adopted goal is abandoned. The actions are used to describe changes that the character knows how to execute. The events are used to describe changes that are not controlled by the characters, like the world’s physics or the addition of a specific information to a character’s knowledge.



This structure was developed with a recursive representation of knowledge, following the theory of mind mentioned in the previous section. Each world has a set of characters, and each character has a set of worlds to represent his knowledge. The characters present in a knowledge world may also have a knowledge, and so on recursively. These recursive steps of knowledge are known as theory of mind levels [15].

Another feature of the knowledge representation adopted here is the support for uncertainty. One of the main problems of planning actions considering the others is the need to guess what they are thinking. Most of this information is usually incorrect or incomplete. For this reason, the knowledge is represented as multiple worlds [16], each one representing a configuration that the character believes as true, with a probability value that represents the confidence that he has in it. Since different conditions may trigger different behaviors, each knowledge world has its own set of adopted goals.

### B. Mistakes

Given that the main world and the worlds represented inside each characters' knowledge are independent, the representation of incorrect information assumed by a character regarding attributes and the existence of objects and characters is straightforward. If inside a character's knowledge world, one element has an attribute value different from the corresponding element's attribute inside the main world, it can be inferred that the character made a mistake regarding that attribute. If a character's knowledge does not have an element that exists in the real world, it means that he does not know that element.

Besides these mistakes, the perception simulation is also able of allowing mistakes about identities, e.g., it allows characters to see an element A and not be sure if they are seeing element A or B. Then, whenever they try to execute an action using B as parameter, they must consider that they may be executing this action with A or B, while the system must actually execute the action with element A. Also, characters must be aware of the possible identification mistakes the others may do. To enable these mistakes, there must be some way of translating elements between different worlds.

The ID tuple described above ( $\langle ID_{original}, ID_{local} \rangle$ ) is used for it. Every element has an  $ID_{local}$  that is unique inside a world, but is shared among the worlds that are in the knowledge base of one character, and an  $ID_{original}$  that identifies elements between worlds that are in different levels. During the story generation, whenever a character perceives one element that he did not see before, he creates a new instance in each of his knowledge worlds to represent the new element. These new instances will receive the same new  $ID_{local}$ , and will receive as  $ID_{original}$  the  $ID_{local}$  of the original world's element. In short, the  $ID_{local}$  is used to identify an element inside one world and to identify which elements, from different knowledge worlds, were created based on the same perception. The  $ID_{original}$  identifies which element originated the perception.

Formally, the function  $T : E_l, W, W \rightarrow E_l$  ( $E_l$  being the set of elements: characters, objects and locations) that translates the element  $p$  from the context of world  $w$  to the context of world  $w'$  is described as follows:

$$T(p, w, w') = \begin{cases} p', El(p', w') \wedge (ID_L(p) = ID_O(p')), & \text{if } owner(w') \in C(w) \\ p', El(p', w') \wedge (ID_O(p) = ID_L(p')), & \text{if } owner(w) \in C(w') \\ p', El(p', w') \wedge (ID_L(p) = ID_L(p')), & \text{if } owner(w) = owner(w') \end{cases}$$

where  $El(p, w)$  is a function that returns *true* if  $p$  is an element of  $w$  and returns *false* otherwise,  $ID_L(p)$  returns the  $ID_{local}$  of  $p$ ,  $ID_O$  returns the  $ID_{original}$  of  $p$ ,  $owner(w)$  is the character who has  $w$  as knowledge world and  $C(w)$  is the set of characters of  $w$ .

Figure 1 presents an example of how such identification works inspired on Little Red Riding Hood story after Little Red sees the Wolf disguised as Grandma. The first diagram shows the original world, its characters, the knowledge of the Wolf, and the knowledge he has about Little Red's knowledge. Notice that in the real world, Grandma, Wolf and Little Red have 0, 1 and 2 as  $ID_{local}$ , respectively. Inside the Wolf's knowledge, each character has the corresponding values as  $ID_{original}$ , which means that the Wolf correctly identified them. Inside what he believes to be Little Red's knowledge, Grandma's  $ID_{original}$  is the same as the  $ID_{local}$  of the Wolf in the superior level, which means that he believes Little Red saw him and identified as Grandma.

The second diagram represents Little Red's knowledge, which is composed of two worlds (1 and 2). In world 1 the Wolf has the same  $ID_{local}$  that Grandma has in world 2. This means that she saw someone and does not know if she saw the Wolf or Grandma. Considering that both have 1 as  $ID_{original}$ , which is the  $ID_{local}$  of the Wolf in the real world, actually, she saw the Wolf.

Since there is no level above the real world, the characters of the real world have an invalid value ( $-1$ ) as  $ID_{original}$ .

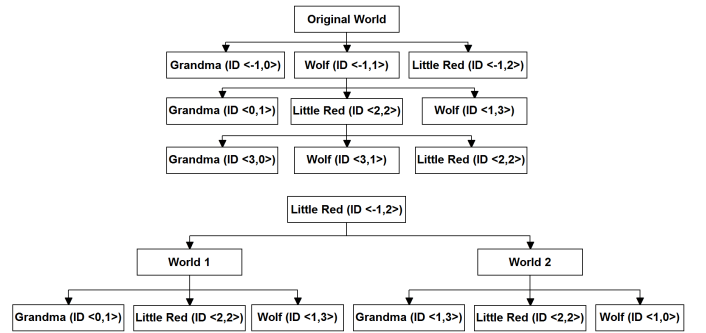


Fig. 1. Example of knowledge representations with identification mistakes.

Technically, IDs translate actions' parameters between worlds. The  $ID_{local}$  translates parameters between knowledge worlds: if Little Red decides to shoot the Wolf, she must consider that she may be shooting at Grandma. The  $ID_{original}$  translates parameters between worlds of different levels: if Little Red shoots at "Grandma" using the context of world 2, in the real world, the Wolf will be shot.



### C. Story Generation Procedure

The story generation is made in a turn based fashion following the cycle presented in Fig. 2. In every turn, the system updates the current character's knowledge with the perception method and then updates his goals using the behavior rules. In the next step, the character selects an action to execute calling the planning process if necessary (it may also use the results of previous planning). The action obtained ( $A'$  in the example) is translated from the character's point of view (POV) to the world's POV (action  $A$ ), and then is executed. Finally, the process uses the events that became executable. This is repeated until no characters are able to create a plan that fulfills their goals or there are no more goals to be fulfilled.

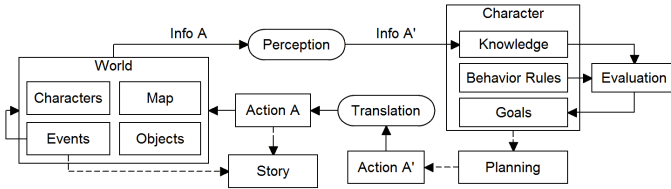


Fig. 2. Story generation cycle.

## IV. RECURSIVE PLANNING WITH PERCEPTION

POMDP is a framework developed to solve sequential decision problems with partial observability and stochastic actions. The framework is composed of a set of states, a set of actions, a transition function, a set of evidences, an observation function and a reward function. Instead of dealing with single states, methods that solve POMDPs use probability distributions over states to represent the uncertainty, which are called belief states. The solution of a POMDP is a policy, which is a function that returns the action that must be executed in each belief state to reach the situations with highest rewards.

In simulation based methods, each character has his own knowledge and executes his own reasoning process to decide which action to execute. Since their knowledge is incomplete, in order to allow them to act considering uncertainty, we decided to use POMDPs as the reasoning process. Below we discuss the adaptations made from what is commonly used and then present the planning method.

### A. POMDP Adaptations

The configurations that a character's knowledge world may assume is the set of states in the POMDP representation. In the same direction, the knowledge worlds he has along with their probabilities compose the initial belief state. The actions the planning character knows how to execute and the actions that he knows the other characters know are the POMDP actions. Despite the framework being capable of handling stochastic actions, for now we decided to use only deterministic actions. With only deterministic actions the transition function, which defines the probability of reaching a state  $s'$  after executing an action  $a$  in state  $s$ , is  $P: S, A, S \rightarrow (0, 1)$ , where  $S$  is the set of states and  $A$  is the set of actions.

Regarding the observation function and the evidences, the perception simulation process could be used as substitute. However, given how our perception simulation works, it would

lead to a combinatorial explosion of information. Using the perception simulation as the observation function, a planning character would consider that any combination of known object and character could appear at any known location. For this reason, we decided not to use the POMDP's observation functionality. It is important to notice that the planning process still uses the perception simulation to allow one character to know what the others will see in the following turns, using what he already knows about the locations where the other characters are. What we are not using is a function of the POMDP framework that would allow the planning character to try to guess what he would perceive in his current location that is different from what he already knows that is there. Despite not using the observation function, we still decided to use POMDP instead of the simpler Markov Decision Process (MDP), because the belief states are necessary to deal with the characters' knowledge uncertainty.

Regarding the reward function, the reward of a state depends only on its own characteristics: if the state fulfills one or more goals, it has the corresponding rewards. However, each state is a knowledge world, which, as explained, has its own goals. Thus the reward of a state is calculated based on its own goals. Considering these aspects, the utility function  $U$  of a belief state  $b$ , when the next acting character is  $c$  is:

$$U(b) = \max_{a \in A_c(b)} \{F(c, a, b) \times \sum_{s \in S} (b(s) \times (R(s) + U(b_a)))\} \quad (1)$$

$$b_a(s') = \alpha \sum_{s \in S} (P(s', a, s) \times b(s)) \quad (2)$$

where  $A_c(b)$  is the set of executable actions of character  $c$  in belief state  $b$ ,  $F(c, a, b)$  is the probability of character  $c$  execute action  $a$  in belief state  $b$  (this probability is detailed in the next subsection),  $S$  is the set of states,  $R(s)$  gives the reward of state  $s$ ,  $b(s)$  is the probability of state  $s$  in belief state  $b$ ,  $b_a$  is the belief state reached after executing action  $a$  in belief state  $b$  and  $\alpha$  is a normalizing constant. These functions are adaptations of the functions presented by Ramirez and Geffner [17]. We derived these functions by removing the observation factor (as mentioned, we decided not to use the observation functionality) and adding the actions' probabilities of each possible actor.

Notice that the behavior rules are analysed to generate new goals before the planning process (as presented in Fig. 2) and not during it. This is done to restrain the characters from executing actions that would generate new goals just to receive the rewards associated to these goals. For instance, suppose the characters have a "stamina" attribute that is decreased when they move and is recovered when they rest, and that they have a behavior rule that gives them the goal to recover their stamina if it reaches zero. If new goals were added during the planning process, it would eventually return a policy that includes the execution of "move" (just to make the stamina reach zero and trigger the behavior rule) followed by the execution of "rest" (to recover the stamina and reach a state with higher reward). However, walking continuously solely to get tired and rest hardly can be considered rational.

Although no behavior rules are evaluated during planning, some of the goals' conditions must be described with parameters that may change according to the world's configuration when a character must plan for the others. In this kind of problem, a character is facing an environment that changes not only as a result of his actions. Therefore, the reasons that justify the adoption of some goals may change based on the actions of the others, and as discussed in the previous paragraph, it is not possible to abandon and re-adopt them during planning. For instance, if a character has a behavior that gives him the goal to be at the same place as another character, and the goal has fixed parameters, he will try to reach the place where the other character was initially, even knowing that he may have moved to a different location. As solution, it must be possible to describe the location that must be reached with a variable that is restricted by the condition of being the location where the other is. Thus, we adapted the goals' description and evaluation to allow conditional parameters.

### B. Planning Algorithm

We decided to use a bounded lookahead method, which iterates only over the list of belief states reachable from the initial one, given that the exploration of the entire space of belief states is prohibitive for the domain size of reasonable story worlds. To consider the actions of the others, each belief state keeps the information about which character must act next, and in each iteration this information is updated to the next acting character (from now on we will refer to the acting character as "actor") that the planning character knows, following the same order of the story generation process. Alg. 1 presents the main procedure.

The process starts using the planning character's knowledge as initial belief state (line 1) and setting the first actor (line 2), which is the planning character. Then, it proceeds entering into the main loop. In the main loop, it computes the actions that may be executed in the current belief state by the current actor (line 9) with their probabilities (as we explain later in this section, these probabilities represent how much the planning character believes that the corresponding actions will be executed). For each action it creates a new belief state (line 11), projecting all the knowledge worlds (or *states* in the context of POMDP) of the current belief state (line 13). With the new state completely created, the process: defines its probability with the probability of the action used for the projection (line 15); calculates its utility (line 16) and defines the next actor (line 17). Finally, the new belief state is added to the list of belief states to be expanded in a later iteration.

The function that defines which actions may be executed in a given belief state by an actor (line 9) is the one that executes the recursive step. Alg. 2 details how this function works. If the current actor is the planning character, the function generates all the executable actions considering the different worlds the character has as knowledge (line 4). Since different worlds have different configurations, some actions may be executable in one world and may not be executable in another. When these actions are used in Alg. 1 to project the knowledge worlds of the current belief state to the new belief state, if an action is not executable in a given world, the world is projected as the planning character attempted to execute the action, but did not succeed. In this situation, the planning character will know that

**Require:** One character and the lookahead bound (*maxLevel*).

**Ensure:** A policy.

```

1: BeliefState initial = getWorlds(character.knowledge);
2: initial.setActor(character.self);
3: ListOfBeliefStates list;
4: list.insert(initial);
5: while list is not empty do
6:   BeliefState current = list.getNextBeliefState();
7:   list.removeNextBeliefState();
8:   if current.level > maxLevel then break;
9:   ActionsAndProbabilities < actions, probabilities > =
       getNextActions(current, maxLevel);
10:  for all action, probability in actions, probabilities do
11:    BeliefState newState;
12:    for all kWorld in current do
13:      newWorld = kWorld.executeAction(contextAction);
14:      newState.insert(newWorld, current.Probability(kWorld));
15:      newState.setProbability(probability);
16:      newState.calculateUtility();
17:      newState.setNextActor();
18:      if newState.actor is the planning character then
19:        increase newState.level;
20:      list.insertWithoutRepeat(newState);
21: return (getSolution(list));

```

Algorithm 1. Planning algorithm.

all the others will learn that some of the action's preconditions are actually false (for more details, please refer to [5]). Finally, the actions are added to the final list of executable actions, all with a probability of 100%. This value can be assured, because these actions only depend on the planning character (if he reaches the current state during the story generation, he can be sure that he will be able to execute one of them).

**Require:** The current state (*current*), with the current actor (*actor*) and the lookahead bound (*maxLevel*).

**Ensure:** The set of actions that can be executed by the actor in the current state with their probabilities.

```

1: ActionsAndProbabilities < actions, probabilities >;
2: if actor is the planning character then
3:   for all kWorld in current do
4:     newActions = kWorld.getActions(actor);
5:     addNewActions(< actions, probabilities >, newActions,
                   kWorld.probability);
6: else if actor is not the planning character then
7:   for all kWorld in current do
8:     translActor = translate actor to kWorld;
9:     kWorld.updateKnowledge(translActor);
10:    newActions = translActor.plan(maxLevel - current.level);
11:    transActions = translate newActions to kWorld;
12:    addNewActions(< actions, probabilities >, transActions,
                  kWorld.probability);
13: return (< actions, probabilities >);

```

Algorithm 2. Returns the executable actions in a given belief state.

If the current actor is not the planning character, the function iterates over the knowledge worlds, translating the actor to their POV (line 8) and updating his knowledge (line 9). This translation procedure is necessary, because as explained in section III, the same character may be interpreted differently in different knowledge worlds. Then the function obtains the actions that the actor will execute by making a recursive call to the planning procedure (line 10). As can be noticed, this recursive call decreases the lookahead bound by the current belief state level. Although each character, when planning his own actions, always use the max lookahead bound as limit (which means that the recursive step would need to do the same to generate the proper expected policy) we decided to use this

simplification to decrease the cost of the recursive phase. In the following turns of the story generation, the planning character continues using the policy he obtained increasing one step of it and, to avoid incoherent expectations, it also increases one step of the policies of the other characters.

Finally, the obtained actions are translated to the context of the knowledge world (line 11) and added to the final list of actions (line 12). The probability of each action is the sum of the probabilities of the worlds that generated them, as detailed in the following equation:

$$F(c, a, b) = \sum_{s \in S} b(s) \times PA(a, c, s) \quad (3)$$

where  $PA(a, c, s)$  returns 1 if  $a$  is one of the actions that the planning character believes that actor  $c$  will execute in state  $s$ , and returns 0 otherwise. In short, the planning character considers that an action is as likely to be executed as the knowledge worlds that suggested it.

The condition that controls the recursive feature is also included in the planning process, but we did not detail it in the presented algorithms for the sake of simplicity. The “setNextActor” function in line 15 of Alg. 1 is responsible for the task. Whenever the planning process reaches a recursion level where the planning character does not have knowledge about the knowledge of the others, the function will only set the next actor as the current planning character. Thus, the function presented in Alg. 2 will not make any recursive calls.

The resulting structure (exemplified in Figure 3) is a tree in which the nodes represent the belief states (where the root node is the initial and the leaf nodes are the final belief states), and the edges represent the actions. This structure resembles the minimax tree, but unlike the minimax algorithm, the actions of the other characters are obtained recursively and all the actions have probabilities to represent the confidence that the planning character has that the action will be executed.

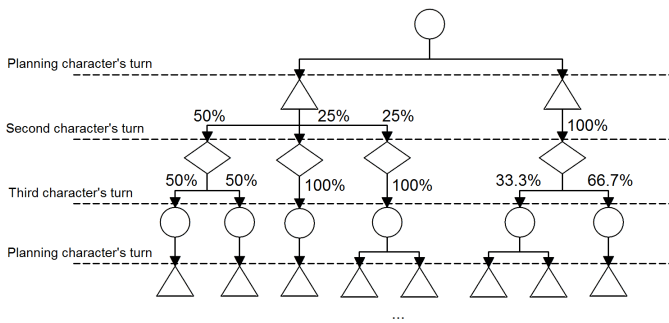


Figure 3. Example of the initial steps of the structure generated by the recursive bounded lookahead with three characters. The nodes are belief states, represented with different forms to define the different actors' turns, and the edges represent the actions, which are annotated with their probabilities. For the sake of simplicity, we removed the probabilities of the planning character's actions. As explained, they all have a probability value of 100%.

## V. SCENARIO EXAMPLE

We decided to use the Little Red Riding Hood story world, based on the Grimm's variation [18], as test scenario, given its famous deception scene based on a perception mistake. This

world was recreated to verify if the system would be able to create coherent variations with as less predefined scripts as possible. In this section we summarize the world description and then discuss the resulting stories.

### A. Little Red Riding Hood World

In our version of the fairy tale we used four characters: Little Red, a wolf, a hunter and Grandma; two objects: the basket of sweets and a dress; and a map composed with Little Red's house, Grandma's house and some locations between them to represent the forest. The method used for authoring was the cycle of generation, implementation and simulation described by Swartjes [2].

The characters were created with 16 attributes, the most relevant ones being *stamina*, *evil* and *defenceless*. *Stamina* is a numeric attribute spent when some actions are executed and is recovered after resting. *Evil* is a boolean attribute that indicates natural disposition and *defenceless* is a boolean attribute that indicates if the character can defend himself. All the characters know the actions for moving, getting near another character, moving away from another character and resting. *Move* is an action that changes the location of the actor, but that cannot be used when he is near another character. The Wolf also knows *wear* and *eat*. The first changes his appearance using an object that he must have. The second puts the action's target inside the acting character's stomach, and also makes the actor appear to be more menacing. As precondition to eat another character, the actor must be evil and must be near the target character. Finally, the hunter knows the action *kill* that kills a target character by cutting his stomach.

In the initial state of the world, the protagonist is at her house, has medium level of stamina and the goal of delivering the basket to Grandma. She starts knowing only her house, the basket of sweets and Grandma. Her behavior rules are: (1) *at the same place as an evil character, or near an unknown person, she receives the goal to be anywhere else* and (2) *in case she knows anyone not evil, she receives the goal to be at the same place as him*. The Wolf begins in a forest next to the protagonist's house, with a high stamina, without any initial goals, knowing the entire forest and the hunter. However he does not know where the hunter is. His behaviors are: (1) *eat anyone that is defenceless*, and (2) *talk to anyone that he did not meet before*, which has a lower reward. Grandma starts at her house, with low stamina, without initial goals and with the same behaviors as the protagonist. She only knows her house and Little Red. The hunter starts in one of the forest locations, with medium stamina, without initial goals and one behavior: *kill anyone that is evil*. He also knows only part of the forest. The basket starts at Little Red's house, and the dress starts at Grandma's house. All the characters were set with two levels of theory of mind, which means that a character's knowledge reaches what the others know, but not what the others know about what he knows. The bounded lookahead limit was set to three steps.

We did not include communication actions that make one character pass any information it has to another character, because of the resulting computational complexity. Then, to recreate the scene where Little Red and the Wolf talk, we included a *talk* action to the knowledge of the Wolf and an

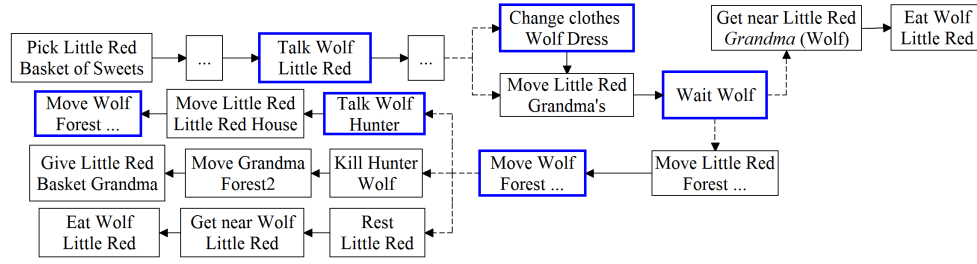


Figure 4. Story variations scheme. Dotted lines represent different developments and blue rectangles represent social actions.

event that inserts into the Wolf's knowledge the character Grandma, after he talks with Little Red. Also we added an event that makes the protagonist know that her grandmother was eaten if she recognizes the Wolf with the disguise.

### B. Generated Stories

We executed the story generation process 25 times and obtained 23 different stories. Most of them only differ from each other regarding the paths that the characters used to traverse the forest, but there were also differences in keypoints. By grouping the stories according to these points, we found four main variations. Figure 4 presents a schematic view of them, where the blue rectangles represent the social actions.

At the beginning of the simulation, Little Red goes to the forest where she necessarily encounters the Wolf, because his starting location is next to her house. At this point, she does not decide to flee, because she does not know that the wolf is evil, while the wolf does not try to eat her, because he does not believe that he will be able to catch her. So he decides to talk to her, which triggers the event that makes him aware of Grandma. Now that he knows her, he adopts the goal of eating her. Knowing the entire forest, the wolf goes straight to Grandma's house, while Little Red has to explore the forest. Even so, sometimes she is able to reach the house at the same time as the wolf, sometimes even before. Reaching the house before Little Red, the wolf eats Grandma and decides to wear the dress to deceive the protagonist. Grandma is not able to flee from the wolf, because she does not have enough stamina.

In the scenarios where Little Red reaches Grandma's house after the wolf disguises himself, sometimes she recognizes him, sometimes she does not. When she does not recognize him, she approaches him to deliver the basket of sweets and ends up being eaten. When she does recognize, the event that makes she know that Grandma was eaten is triggered. Also, she decides that the wolf is evil by perceiving his new menacing appearance, which makes she adopt the goal of being somewhere else (the goal that makes her want to flee). If she encountered the hunter while walking through the forest, she also adopts the goal of being at the same place as him. Either way, she goes out of Grandma's house and the wolf starts trying to reach her, but instead of following her steps, he goes to a place that he believes that she will go. This time he decides to eat her, because he knows that she already spent some of her stamina while traversing the forest.

In some stories Little Red was able to reach the hunter's location, in others she was not. When the hunter's location was not reached, in some scenarios the wolf was able to eat

Little Red, and in others she was able to escape. When she was able to reach the location, the hunter himself not necessarily was able to recognize the wolf. When he recognized the wolf, he killed the antagonist, Grandma went out of his stomach and Little Red delivered the basket of sweets. When he did not recognize the wolf, the wolf decided to talk to him, Little Red ended up going to her house and the wolf went back to the forest. In that scenario with the three characters, Little Red usually had no more stamina, but the antagonist decided to talk to the hunter instead of trying to eat her, because the hunter would see the action and consequently kill him. Afterwards, when the wolf decided to go back to the forest instead of resuming the pursuit, we believed at first that he decided not to follow her because he did not know where she was, but analysing his policy, we verified that he believed that the hunter would eventually follow and kill him, so he decided to flee.

Regarding the situations where Little Red reaches Grandma's house before the wolf, she delivers the basket and afterwards, the wolf eats Grandma in front of her. Then she decides to flee, because she identifies him as evil by seeing him eating Grandma, and the story progresses as already described in the scenarios where she recognizes the wolf using the dress.

### C. Discussion

Despite using deceptive behaviors, all these variations can be generated using predefined scripts. However it would be necessary to specify a different script for each possible development, which also requires previous knowledge about all these developments and some of them were not expected for us. For instance, we expected the wolf to follow Little Red using the same path, as he did in our previous work. Instead, he went to a different location and waited for Little Red to go there. There was even a variation where he accidentally went to the place where the hunter was. In this variation Little Red went to her house, because she did not meet the hunter before, the hunter killed the wolf and Grandma went out of the wolf's stomach. Regarding the disguise action, the wolf is able to plan it, because he knows that when Little Red sees him wearing the dress, she may believe that he is her grandmother and may approach him to deliver the basket of sweets. Then, he would be able to eat her without much effort.

Regarding the authorial effort, the simulation of more reasoning capabilities requires more data during the creation of worlds. For a character to reason about what the others will do, it is necessary for him to know what the others know. This extra knowledge must be predefined or added using some of the system's mechanisms. As presented in Section II, some previous works deal with this problem making the actions

add goals and/or specific information to a target character's knowledge. So if a character executes these actions he will know that the target will adopt the goal and/or know the specific information. We used the same approach in this work, which increases the authorial effort. On the other hand, as already mentioned, with more reasoning capabilities the system requires less predefined scripts to simulate some behaviors. In previous works, we had to adapt the wolf's knowledge to make him avoid eating Little Red at the beginning, while using the recursive planning it was not necessary, because he realized by himself that he would not be able to catch her.

Another disadvantage to authorial effort of automatically dealing with the theory of mind is making the generation process even less predictable, which means that more authoring cycles would be necessary until the author is satisfied with the result. However, we believe that the loss of control is necessary if the main objective is to create unexpected developments.

Unfortunately, our recursive approach increased the run-time. It required 57 minutes on average to generate one story, while in our previous work, without the recursive planning, it required 20 minutes on average. The parameter that most influences the performance is not the number of characters, but the complexity of their knowledge. Approximately 93% of the generation time was used in the wolf's turns, because he is the character that has more information inside his knowledge worlds and also deals with more uncertainties. Since the performance is based on this knowledge complexity, it is difficult to define exactly how the system would perform in the generation of different kinds of stories. Story characters may become more knowledgeable or may become more ignorant. If a character becomes more knowledgeable, he may have less uncertainties, but may have more correct information to think about, while if he becomes more ignorant, he may have less correct information to use, but may have more uncertainties.

Given this limitation, this approach hardly could be used in real time. However, it still could be used by authors as support to help develop a number of different predefined stories that would cover the most important player decisions, and even suggest unexpected, but promising developments.

## VI. CONCLUSION

To increase the potential of simulation-based story generation systems, we developed a recursive planning approach that works with a previously developed perception simulation process. This combination is necessary to generate scenarios where the characters rely on the others' imperfect knowledge to reach their own goals. The emergence of deceitful behavior was demonstrated using a recreation of Little Red Riding Hood world, with which the system was also able to create coherent and unexpected developments.

However, we still need to increase the performance. A promising idea is to avoid using the recursive step in consecutive iterations when what the others know about the world does not change. Also, there are many capabilities that need to be improved. For instance, we had to use an event to make Little Red know that Grandma was eaten by the wolf. If she were capable of abductive reasoning, she could have guessed this information by herself, which would make the generation more independent and decrease the authorial effort.

It is also necessary to verify the system generation potential and quality using a number of different narrative worlds. We intend to create new worlds (not based on famous stories), allow users to create their own characters to see how they would perform and receive their feedback on the generated stories using a questionnaire with focus on the sense of social behavior and quality.

## ACKNOWLEDGMENT

The authors would like to thank the Brazilian National Council of Technological and Scientific Development (CNPq) for supporting this work and the anonymous reviews for helping with the improvement of this paper.

## REFERENCES

- [1] R. M. Young, M. O. Riedl, M. Branly, A. Jhala, R. Martin, and C. Saretto, "An architecture for integrating plan-based behavior generation with interactive game environments," *Journal of Game Development*, vol. 1, no. 1, pp. 51–70, 2004.
- [2] I. Swartjes, "Whose story is it anyway? how improv informs agency and authorship of emergent narrative," Ph.D. dissertation, University of Twente, Enschede, Netherlands, May 2010.
- [3] M. Cavazza, F. Charles, and S. J. Mead, "Character-based interactive storytelling," *Intelligent Systems, IEEE*, vol. 17, no. 4, pp. 17–24, 2002.
- [4] M. O. Riedl and R. M. Young, "An intent-driven planner for multi-agent story generation," in *Proc. of the 3rd Int. Joint Conf. on Autonomous Agents and Multiagent Systems*. IEEE Computer Society, 2004, pp. 186–193.
- [5] D. B. Carvalho, E. G. Clua, C. T. Pozzer, E. B. Passos, and A. Paes, "Simulated perceptions for emergent storytelling," *Computational Intelligence*, 2016.
- [6] D. B. Carvalho, E. Clua, and A. Paes, "Perception simulation in social planning for emergent storytelling," in *2015 IEEE Computational Intelligence and Games (CIG)*, 2015, pp. 75–82.
- [7] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence Journal*, vol. 101, no. 1-2, pp. 99–134, 1998.
- [8] C. Castelfranchi, "Modelling social action for ai agents," *Artificial Intelligence*, vol. 103, no. 1, pp. 157–182, 1998.
- [9] H.-M. Chang and V.-W. Soo, "Planning-based narrative generation in simulated game universes," *IEEE Trans. Comput. Intell. AI in Games*, vol. 1, no. 3, pp. 200–213, 2009.
- [10] D. Premack and G. Woodruff, "Does the chimpanzee have a theory of mind?" *Behavioral and brain sciences*, vol. 1, no. 04, pp. 515–526, 1978.
- [11] C. Pearce, B. Meadows, P. Langley, and M. Barley, "Social planning: Achieving goals by altering others' mental states," in *Proc. of the 28th AAAI Conf. on Artificial Intelligence*. Quebec City: AAAI Press, 2014.
- [12] M. Si, "Thespian: a decision-theoretic framework for interactive narratives," Ph.D. dissertation, University of Southern California, California, United States, May 2010.
- [13] J. McCoy, M. Treanor, B. Samuel, N. Wardrip-Fruin, and M. Mateas, "Comme il faut: A system for authoring playable social models," in *7th Artificial Intelligence and Interactive Digital Ent. Conf.*, 2011.
- [14] A. Gerevini and D. Long, "Bnf description of pddl 3.0," Tech. Rep., 2005. [Online]. Available: <http://www.cs.yale.edu/homes/dvm/papers/pddl-bnf.pdf>
- [15] H. D. S. Reis, "Lie to me: Lying virtual agents," Master's thesis, Universidade Técnica de Lisboa, 2012.
- [16] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi, *Reasoning about knowledge*. MIT press Cambridge, 1995, vol. 4.
- [17] M. Ramírez and H. Geffner, "Goal recognition over pomdps: Inferring the intention of a pomdp agent," in *Proc. of the Twenty-Second Int. Joint Conf. on Artificial Intelligence*. AAAI Press, 2011, pp. 2009–2014.
- [18] J. Grimm and W. Grimm, *The Complete Grimm's Fairy Tales*. Trans. Margaret Hunt and James Stern. New York: Pantheon, 1972.



# Heuristics for Sleep and Heal in Combat

Shuo Xu  
School of Computer Science  
McGill University  
Montréal, Québec, Canada  
shuo.xu@mail.mcgill.ca

Clark Verbrugge  
School of Computer Science  
McGill University  
Montréal, Québec, Canada  
clump@cs.mcgill.ca

**Abstract**—Basic attack and defense actions in games are often extended by more powerful actions, including the ability to temporarily incapacitate an enemy through sleep or stun, the ability to restore health through healing, and others. Use of these abilities can have a dramatic impact on combat outcome, and so is typically strongly limited. This implies a non-trivial decision process, and for an AI to effectively use these actions it must consider the potential benefit, opportunity cost, and the complexity of choosing an appropriate target. In this work we develop a formal model to explore optimized use of sleep and heal in small-scale combat scenarios. We consider different heuristics that can guide the use of such actions; experimental work based on *Pokémon* combats shows that significant improvements are possible over the basic, greedy strategies commonly employed by AI agents. Our work allows for better performance by companion and enemy AIs, and also gives guidance to game designers looking to incorporate advanced combat actions without overly unbalancing combat.

## INTRODUCTION

Multi-agent combat in Role Playing Games (RPGs) is commonly supplemented by powerful abilities, such as sleep or heal, which allow a team to improve their combat chances by (temporarily) disabling an opponent, or by saving an ally from potential death. Unbounded, these abilities can easily trivialize combat and so most games also impose heavy constraints on their use, making the choice of if, when, and on whom to use such an ability a non-trivial part of the game complexity, and one of the skills players must learn and optimize through multiple battles and repeated gameplay. Non-player characters (NPCs) on both the player side and enemy side, however, can also wield these abilities, and while hand-scripted or randomized approaches are commonly used for their speed and simplicity, avoiding the need for expensive search in action selection, the resulting choices do not always meet player expectation of intelligent companions or opponents.

In this work we develop a formal cost-benefit model to represent the impact of sleep and heal in small-scale game combats. We use this analysis to develop simple and efficient heuristics for selecting whether to use sleep or heal instead of attack actions, considering sleep and heal separately as well as in combination. In each case we validate our heuristics through detailed experimental analysis of abstract combat scenarios based on *Pokémon*. Measurements on different performance factors, including win-rate, remaining team health, and total damage dealt show significant improvements over other, com-

mon heuristics, including *Pokémon*'s scripted design approach. Specific contributions of our work include:

- We perform a formal analysis of sleep and heal actions, developing a cost-benefit model that we then use to define an optimizing heuristic for action selection and targeting. As far as we know ours is the first work focused on sleep and heal in combat games.
- Our design is backed by significant experimental work based on *Pokémon* combat and character attributes, showing that our heuristics are manifestly better than common, basic approaches.
- A unified cost-benefit model for both sleep and heal allows us to combine them into a single heuristic, which we evaluate in larger scale situations.

## RELATED WORK

Team combats in RPGs are essentially (small-scale) *attrition games*, where one team must fully eliminate another. These are already known to be computationally complex, even with just basic attack and defense. Furtak and Buro, for instance, present proofs on the complexity of two-player attrition games showing that the problem is computationally hard for most game cases, and deciding the existence of deterministic winning strategies for basic attrition games is PSPACE-hard and in EXPTIME [1]. Ontañón *et al.* provide a survey of existing works on solving AI problems in the commercial game *StarCraft*—a much larger scale example of an attrition game compared to the ones we consider [2]. They discuss topics such as current tactics, strategies, and state-of-art AI agents for *StarCraft*, shedding light on challenges in general attrition games. In smaller contexts, Tremblay *et al.* proposed a greedy heuristic for enemy targeting based on enemy *threat*, a value positively related to enemy attack strength, and negatively to health [3]. The theoretical justification for that heuristic was validated in realistic, FPS-inspired combat scenarios, but still only allows for attack as a combat action.

Combat AIs based on searching through a decision or state space trade runtime performance for improved behaviour that can better adapt to dynamic contexts. Brute-force (DFS or BFS) search does not scale well of course, but can be improved through use of alpha-beta or one of its variants. Work on Fast Alpha-Beta Search, for instance, shows that a search approach can perform better than many scripted strategies in RTS games, and can meet real-time constraints for small scenarios [5].

Even when limited to just attack (or move) scenarios, however, search approaches are necessarily non-exhaustive, truncating search depth to meet timing requirements, and so dependent on good heuristics for state evaluation. Stanescu *et al.* considered the use of Lanchester models, a military approach to estimating combat losses suitable for large army interactions [6]. They use this to improve accuracy of state estimation, significantly improving a search-based StarCraft bot.

More efficient search can also be performed using heuristic search algorithms, such as in Monte Carlo Tree Search (MCTS), a search algorithm that relies on random sampling [7]. Bruce Abramson first experimented with the idea in turn-based two-player games including Tic-tac-toe and Chess [8]. The MCTS algorithm has since been extended to solve AI problems in more genres of modern computer games including real-time games such as *Total War: Rome II* [9], card games such as *Magic: The Gathering* [10], *etc.* MCTS can be complicated to implement in combat games, as state estimates are based on simulating playout, and thus can be highly approximate. Uriarte and Ontañón proposed a combat model for 2-person attrition games, aiming to define, and learn, an accurate *forward model* for state transitions in MCTS search [11], which can then be applied to StarCraft. A paper by Browne *et al.* summarizes recent work on the MCTS algorithm itself [12].

Game research has also explored use of the Rapidly exploring Random Tree (RRT) algorithm for rapid, search-based analysis. RRT was first introduced by LaValle in 1998 to solve path-finding problems [13]. Bruce then adapted it to the sampling-based planning algorithm for discrete space problems in his thesis in 2004 [14], and it has since found use in several game AI systems, including ones aimed at platformers [15], [16], and stealth games [17]. RRT has advantages in flexibility, but practical use in combat analysis has not yet been shown sufficiently effective or efficient [18].

## COMBAT MODEL

Our approach relies on a basic, formal model of combat. We also consider multiple forms of evaluation, as combat success is not strictly boolean in practice. Below we motivate and describe our design for both aspects, followed by a detailed description of the combat parametrization.

### Model

Although sleep and heal are common, near ubiquitous features of RPG combat, there exists enormous variation in how these skills may be incorporated into combat. The ability to use a sleep action, for example, may be unique to one character, or based on character type or acquired abilities, and thus available to multiple agents. The effect itself may apply to only a single, targeted enemy, or a range of enemies based on area or proximity, with the potential to affect agents on the same side too, as friendly fire. It can have different durations and durative properties, especially in terms of whether sleeping characters can or cannot be woken by attacks, and various casting costs as well, invoked as part of a set of actions

from which selection can be done without replacement, or based on character resources, such as mana or an inventory of objects (such as scrolls). Healing has similar complexity in parametrization. In order to make progress in formal modeling, we thus commit to a single, simple formulation, which while perhaps not fully general, allows for concrete results, and can be extended to variant designs.

Abstractly, we assume a turn-based, 2-team attrition game, with agents on one side termed players, and the other side enemies. Each agent has a (static) attack value, a maximum health, and set of possible actions, which minimally includes attack (which defaults to targeting the enemy with highest *threat* [3]), and may include heal and/or sleep, with use of the latter two constrained by a resource cost and initial supply, and for which we will consider multiple targeting heuristics. Each agent also has a state, either healthy, dead, or sleeping. In the latter case, a turn-counter keeps track of the remaining sleep duration, and we disallow sleep actions on a sleeping agent, as the ability to stack sleeps mainly mimics a sleep of longer duration. We do not model occlusion or geometry, and assume deterministic results from actions; this eliminates the probabilistic element, which affects the decision process of course, but does not change the underlying basis for the decision.

### Evaluation

The success of team combat of the sort found in RPG and action games can be measured in different ways. Abstract attrition games usually focus on “last person standing” as a binary measure of success, but in real games, and in comparing effectiveness of different combat choices, this is not always sufficient. Remaining health is an important criterion as well, as health restoration is not always instantaneous or free, and so impacts subsequent combats. Human players may also be strongly invested in (or entirely embodied by) one character in their team, and thus the survival and health of a primary agent may be paramount. Symmetrically, it is not always possible for player teams to fully eliminate the enemy team. For example, in “boss fights” of *World of Warcraft* the enemy leader may not be supposed to be killed. In these situations the goal is to do as much damage as possible before players die or within some time period.

We thus consider three forms of success in evaluating combat scenarios, measured over multiple simulations of the same situation:

- **WIN:** the ratio of combat wins for the player team, expressed as a percentage.
- **HEALTH:** the average sum of player team health, adding up the remaining health of each surviving player team member after each combat terminates. Note that dead characters will have 0 health.
- **DAMAGE:** the average sum of damage done to enemies, adding up the difference between starting and remaining health of each enemy team member after each combat terminates. This is primarily aimed at evaluating simulations where enemies are unbeatable.



### Combat Context and Parametrization

Our model is sufficient to approximate many games. We base our specific combat implementation and scale our attack, health and other attributes based on values from the *Pokémon* game, a well known RPG consisting of turn-based, team-based (1–3 agents/side, although we consider larger combats as well) combat between different classes of creatures—“pokémon” (pocket monsters). The game includes a total of 721 different pokémon (as of 2015) [19], giving us a large set of varied agents for simulation. Note that we do not model all attributes of pokémon, as *defense*, *speed*, and the special attack/defense values complicate damage calculations, without changing the overall process. We do, however, include *Power Points* (PP), a resource which limits the number of each kind of action that can be performed, with sleep and heal much more heavily constrained than attack (which is rarely exhausted).

We also use *Pokémon* to define our baseline AI in terms of enemy action selection. This is a scripted AI, and the exact action selection process is not officially released by Nintendo, but the main decision criteria are reasonably well understood by player communities [20]. An enemy thus

- has an initial 70% probability of targeting a random player with a sleep or paralysis action, or
- uses a healing move (if available) on the ally with lowest health if it is below 25% of the maximum, or
- attacks the player with lowest health.

### SLEEP

The inclusion of sleep as a combat action requires an AI implement two major decision tasks, first to decide which action to take (sleep or attack), and second to find the best target for the action. We assume here that a slept character is incapacitated for a fixed number of rounds  $> 1$ , symbolically represented as *SLEEP\_DURATION*, that only one character in the player team can cast sleep, and the number of sleep casts is limited (we use a limit of 10, the same as in *Pokémon*).

Below we first analyze sleep to determine a solid basis for computing both benefit and lost opportunity cost, and then use this to offer an improved heuristic strategy for making the two main combat decisions mentioned above. We then undertake experiments to examine and compare our design in real combat scenarios inspired by the *Pokémon* games.

### Cost and Benefit

Fundamentally, use of a sleep represents a cost-benefit trade-off. The primary benefit is of course the fact that an enemy is unable to attack, and so reduces the damage suffered by the player team (we do not consider any increase in the vulnerability of the slept enemy, a common trope which also constitutes a benefit). However, as casting sleep replaces an attack, it also represents a lost opportunity cost, reducing the rate at which damage is dealt to enemies, and thus increasing the overall duration of combat, and potentially the amount of damage received. We now formalize the cost and benefit, focusing on cost in terms of reduced damage dealt, and benefit in terms of reduced damage received.

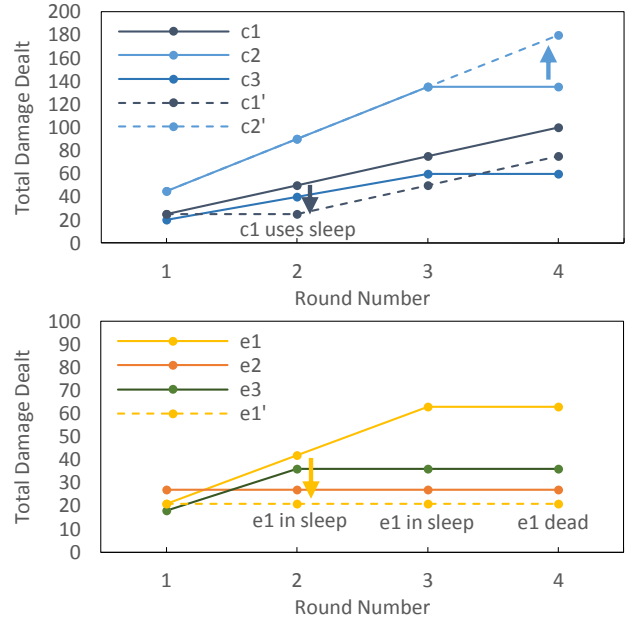


Fig. 1. Time vs. damage inflicted by player team (top), and taken by the player team (bottom).

Figure 1 illustrates the basic trade-offs. In the top graph we show 3 player team members,  $c_1, c_2, c_3$  battling 3 enemies,  $e_1, e_2, e_3$ , plotting total damage dealt by the player team over time (round number). In the absence of sleep, each player does a fixed amount of damage to enemies, the curve flattening out once no viable enemies remain (in this we assume the enemies targeted by  $c_2$  and  $c_3$  die in rounds 2 and 3 respectively, the former event freeing  $c_2$  to join in attacking  $e_3$  on round 3, and leaving only  $c_1$  fighting the remaining enemy). If player  $c_1$  casts sleep on round 2, they do no actual damage on that round, resulting in a bend in their damage curve (marked by an arrow downward), shown as the difference between the solid line (no sleep used) and the dashed line (after sleep is used). This damage reduction is then compensated by player  $c_2$ , who must perform an additional attack in round 4 in order to help kill the last enemy.

The bottom graph in Figure 1 shows the benefit in terms of damage suffered by the player team. Whether sleep is used or not, enemy  $e_2$  is killed in round 2 and enemy  $e_3$  is killed in round 3. Without sleep enemy  $e_1$  is actively attacking and inflicting damage until round 3 (solid line); slept in round 2, however, this part of damage done to the player team is eliminated (dashed line).

We can see from this example that the direct cost of using sleep is the damage loss by the casting player  $c_k$  foregoing an attack move (as well as the cost of using a sleep resource). Under our assumption that sleep casting requires one round, this is simply the caster's attack value,  $c_k.a$  (and if sleep requires multiple rounds to cast this increases linearly, although we do not consider that here). The potential for *overkill*, and targeting choices, however, mean that this is actually an upper

bound, as the full attack power of the caster may not have been necessary in combat. Nevertheless, we use  $c_k.a$  as a cost factor, and instead incorporate imprecision entirely into the benefit factor, as it has a larger impact.

Abstractly, benefit to the player team of using sleep is also simple, merely the product of the slept enemy's attack and SLEEP\_DURATION. Depending on when sleep is cast and targeting choices, however, the slept enemy may die prematurely, during sleep, and thus it is again more correctly an upper bound on benefit, with the exact amount unknown at casting time, also dependent on targeting choices and overkill. As a lower bound, though, we know an enemy cannot be killed faster than if all players join in attacking that one enemy, giving a minimal survival time of a slept enemy of

$$r_{min} = 1 + \left\lceil \frac{\max(0, e_{slept}.h - \sum_{i \neq k} c_i.a)}{\sum_i c_i.a} \right\rceil, \quad (1)$$

where  $c_i$  ranges over players, and  $e_{slept}.h$  is the health of the slept enemy. Note that this assumes the sleep caster is unable to join in attack on the round sleep is cast. A lower bound on benefit is then the product of the minimum of  $r_{min}$  and SLEEP\_DURATION with the attack value of  $e_{slept}$ .

### Decisions

A decision to attack or cast sleep depends on a (likely) positive cost/benefit trade-off. We have several cases to consider, giving us the following decision flow.

- 1) If  $e_{slept}.a * SLEEP\_DURATION < c_k.a$  then the largest possible benefit is lower than the cost, giving us a negative trade-off.
- 2) If  $r_{min} = 1$  then  $e_{slept}$  has low health, and it may be possible for the player team to eliminate  $e_{slept}$  in one round. Sleep may still be effective, depending on targeting choices, but it has reduced value, and other options should be considered.
- 3) When  $r_{min} \geq SLEEP\_DURATION$  sleep is maximally effective, and thus well worth using.
- 4) If  $r_{min} < SLEEP\_DURATION$ , but  $e_{slept}.a > c_k.a$ , then we know that  $r_{min} * e_{slept}.a > c_k.a$ , and thus the trade-off is positive.
- 5) Finally, with  $e_{slept}.a \leq c_k.a$  the benefit lies within the range  $[e_{slept}.a * r_{min}, e_{slept}.a * SLEEP\_DURATION]$ . The trade-off here is ambiguous. However, as we require multi-round attention from the player team in order to eliminate  $e_{slept}$ , use of sleep can still be worthwhile.

If all enemies are in case 1 or 2, then sleep has no or quite limited value. A reasonable basis for choosing to use sleep is thus only when when an enemy can be found to fall in case 3, 4, or 5. We refer to this action choice heuristic as *smart sleep*.

Use of *smart sleep* must also, in general, select from multiple sleep candidates. Benefit is maximized by selecting an enemy with high attack, but the lower bound is improved by choosing enemies with high health, as they are not easily killed otherwise. Note that this is different from attack targeting

heuristics, where low health and high attack has been shown preferable [3]. We will experiment with the actual targeting choices below.

### Experiments

Experimental analysis allows us to evaluate the performance of *smart sleep* in practice, and to observe the impact of different targeting choices. For a more realistic game context, we use values from *Pokémon* (attack, health, and action sets), applying a typical player team of relatively average strength to a wide range of enemy teams, of varying sizes. The latter are controlled by an approximation of the *Pokémon* AI, as described earlier.

The player team consists of 3 pokémon ("Lapras" (#131), "Chandelure" (#609), "Garevoir" (#282)), selected to give us a team that includes agents with and without sleep capabilities (in these experiments healing is disallowed for both teams). Enemy team members are selected randomly from the entire *Pokémon* database, but balanced against the players by ensuring that a team's average attack and health do not vary more than  $\pm 20\%$  from the player team. SLEEP\_DURATION is set to 3 as the default in *Pokémon*. Player team size is fixed at 3, but a range of enemy team sizes between 3 and 6 is considered in order to simulate games of different difficulty, with the upper end of that range representing a game that is extremely challenging.

We considered 12 scenarios, independently varying 3 sleep strategies and 4 targeting heuristics:

Sleep Strategy	Targeting Heuristic
<i>Smart sleep</i>	Highest health
<i>Random sleep</i>	Lowest health
<i>No sleep</i>	Highest attack
	Lowest attack

*No sleep* only uses attack actions, and is a baseline that will show whether sleeping is useful at all, while *random sleep* chooses a sleep action 50% of the time (as long as a healthy target exists), and is intended to see whether the sleep heuristic is important. We simulate each combination and each enemy team size 1000 times. In each run, the choices of enemies are randomized (from Index 001 to Index 721 in the database [19]) to create a specific team size, which is then fixed for all strategy tests within that run to reduce noise in comparing strategies. We evaluate the results in terms of **WIN**, and averaged **HEALTH** scores.

Figure 2 shows the **HEALTH** results for all combinations (note that the y-axis scale changes for clearer visualization of differences among heuristics within each specific chart). In the 3-enemy scenario, we can see that *smart sleep* performs roughly the same as *random sleep*, and both better than *no sleep*. Players always move first in our combats, and so with equal team sizes, as long as some *sleep* is cast individual enemies tend to be killed very quickly, often in the first round. Even if the effect on health is similar, however, *random sleep* is less efficient than *smart sleep* in achieving this result: Figure 3 plots the average number of times sleep is used per

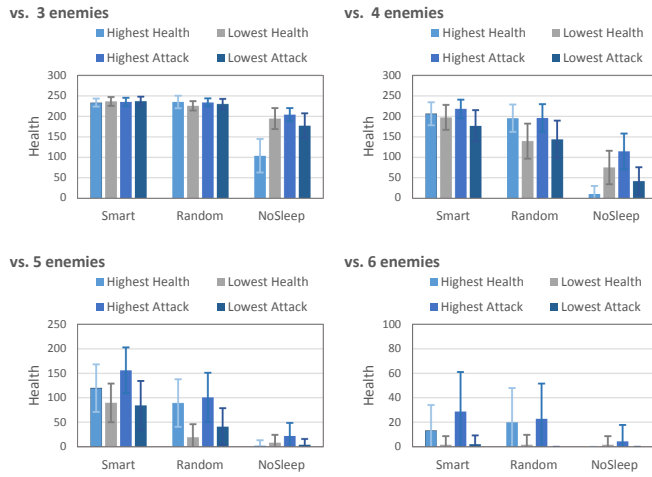


Fig. 2. Total remaining health of players; error bars show  $\pm 1$  standard deviation

combat, and here it can be observed that *random sleep* uses significantly more sleep casts than *smart sleep*.

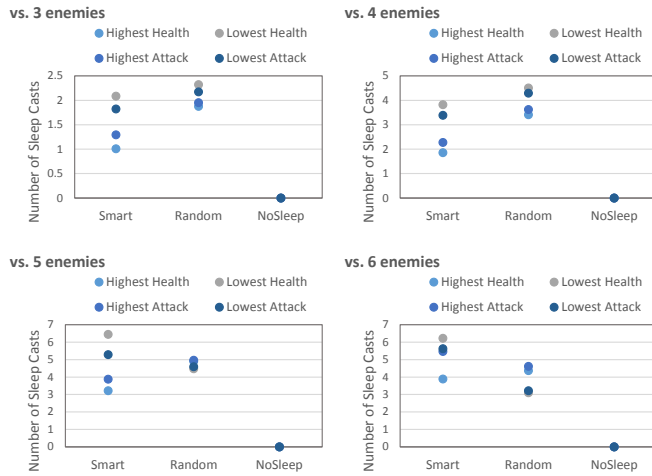


Fig. 3. Averaged total number of sleep casts per combat by players

As the combat becomes more difficult for players *smart sleep* starts to show increasing advantage over *random sleep*. With large enemy teams combat lasts longer, exposing cases 4 and 5 in the *smart sleep* heuristic, and more sleep actions are used accordingly, as seen in Figure 3. Sleep targeting strategies also begin to have an impact on the result in these tougher scenarios. *Highest Health* and *Highest Attack* begin to stand out as superior targeting tactics, confirming our earlier claim that an enemy with either high health or high attack power should be a better target to sleep. Our data shows the latter become preferable at high difficulty levels, although there is also large variance.

Figure 4 shows the **WIN** scores for all combinations. The trend here is similar to the **HEALTH** evaluation data. Surprisingly, however, in the 5 and 6 enemy scenarios, we notice that *random sleep* has very low chance of winning,

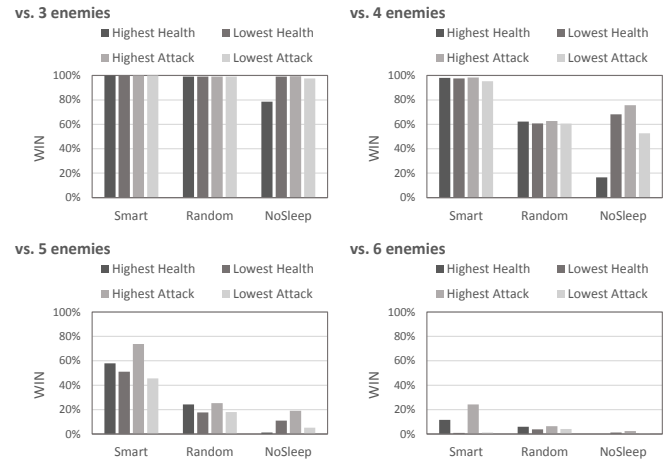


Fig. 4. Percentage win-rate for the player team

barely better than *no sleep*, and this despite the fact that its remaining health score (in winning situations) from Figure 2 does not differ that much from *smart sleep*. This suggests that *smart sleep* is better focused at helping win the combat than at ensuring maximal health, and also indicates the importance of not wasting sleep casts, especially when enemies are stronger.

## HEAL

Healing actions have an effect similar to sleep in that they increase combat survival, although they do so by raising ally health rather than by reducing enemy attacks. We thus follow a similar analysis as for sleep, using a cost/benefit derivation to derive a choice heuristic, and then performing experiments to validate it. As with sleep we use a symbolic value for the most relevant parameter, using **HEAL\_AMOUNT** to represent the amount of health restored by a heal action. In our case we set this to 50% of the target's maximum health, capped to avoid any overhealing. Again following *Pokémon*, heal casts are limited to at most 15 in a single combat.

## Cost and Benefit

In our analysis of sleep, the benefit and the cost are represented in terms of damage decrease, from enemies and from the sleep caster respectively. For heal, the cost remains the same—by foregoing an attack action the player team loses an amount of damage equal to the healer's attack:  $c_k.a.$  Benefit, however, has no direct relation to the damage dealt by a player. In a trivial sense, the benefit is simply the health increase provided to the healing target—the **HEAL\_AMOUNT** itself. Intuitively, however, healing is most useful when applied to a character who would otherwise be killed by the enemy, and this then relates to the amount of damage done to the enemies. The recovered portion of health potentially extends the healed agent's  $c_H$ 's lifetime beyond a given round  $T_B$  by a number of rounds,  $\Delta t$  to  $T_H = T_B + \Delta t$ , and the damage dealt by  $c_H$  during this extra time is the gain for the player team,  $benefit = \Delta t * c_H.a.$

Calculating  $T_B$  and  $T_H$ , and thus  $\Delta t$  is hard because the values depend on the amount of attack directed at a given character, and thus targeting decisions. To ensure that using heal has advantage over using attack, we again find the minimum benefit and see if it is greater than the constant cost. Minimum benefit can be found by considering the smallest  $\Delta t$ , which is produced by the smallest possible  $T_H$  and largest possible  $T_B$ . The latter is maximized when few (or even no) enemies attack, while the former is minimized if all enemies switch attack to  $c_H$  immediately after heal is used. For simplicity, and reflecting the fact that in many real games enemies follow a scripted attacking routine that does not involve reprioritizing or changing targets while in combat, we assume attack strength applied to  $c_H$  is unchanging, and set it to the sum of attacks of all living enemies,  $A = \sum_{e \in E} e.a$ . This gives,

$$\text{benefit} \approx \left( \left\lceil \frac{c_H.h'}{A} \right\rceil - \left\lceil \frac{c_H.h}{A} \right\rceil \right) * c_H.a \quad (2)$$

where  $c_H.h$  and  $c_H.h'$  are the healths of  $c_H$  before and after healing respectively.

The timing of using heal also matters. Assuming any character loses health points healing will naturally improve the **HEALTH** result, but for **WIN** there is no point to using heal when it is unlikely to result in any increased damage dealt. We thus make another hypothesis that heal is used only when an ally is in danger—if one of the player team could potentially be killed within the next round, then saving it has a good chance of achieving a benefit, assuming healing can prevent the untimely death, which is guaranteed if health is raised above  $A$ , the sum of enemy attack values. Healing is thus most useful when  $c_H.h \leq A \wedge c_H.h' > A$ .

### Decisions

Our decision heuristic directly follows the above reasoning. If healing is possible, we determine all players for whom  $c_H.h \leq A \wedge c_H.h' > A$ ; these are allies of  $c_H$  (or  $c_H$  itself) which could benefit from healing. We then compare benefit as computed in formula 2 with the static cost  $c_k.a$ , filtering the candidate list to ones with a positive trade-off. This is our basic *smart heal* heuristic, and we combine this with a default targeting strategy of choosing a candidate with maximal benefit-cost difference.

### Experiments

Experiments were again conducted using a setup similar to our sleep experiments. This time, however, we considered different enemy team sizes, of 2–5 and 8. Sizes 2–5 are meant to model situations in which the utility of heal ranges from ineffective to important to survival. For these we measure performance in terms of **HEALTH**. Enemy team sizes of 3–5 and 8 are measured with **DAMAGE**, as the upper bound of this set is effectively impossible to win, much like some boss fights. (We do not include enemy teams of 6 and 7 for space reasons, and as 8 is a better indication of the upper extreme.) We evaluate all sizes with **WIN**.

Different healing and targeting approaches are also compared. As well as our *smart heal* strategy, we consider *greedy heal*, healing an ally whose health falls below 50% of maximum, as a strategy similar to that used in many games. Again we include random choice, and a baseline of no healing. These strategies are multiplied by 2 different targeting heuristics, either choosing the candidate with maximal benefit as defined above, or simply a random target. This gives us 7 combinations (*no heal* does not have a targeting heuristic):

Heal Strategy	Targeting Heuristic
<i>Smart heal</i>	Smart
<i>Greedy heal</i>	Random
<i>Random heal</i>	
<i>No heal</i>	

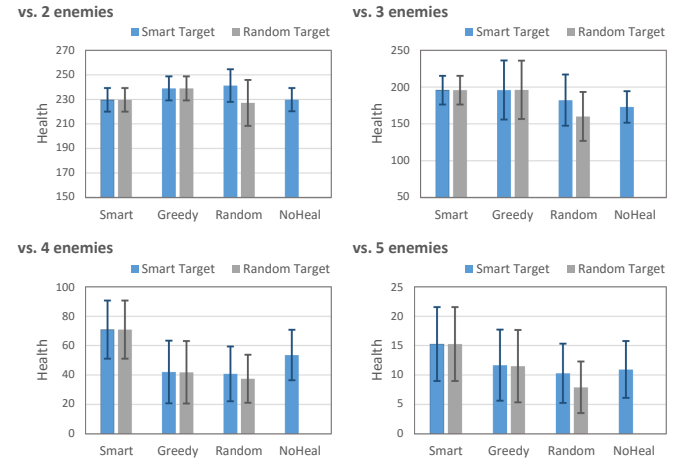


Fig. 5. Total remaining health of players against 2–5 enemies; error bars show  $\pm 1$  standard deviation

Figure 5 shows that *smart heal* does have advantages over other strategies. An exception is for the size 2 enemy team; healing here is not required, and *smart heal* acts like *no heal*. The *greedy heal* and *random heal* strategies, however, achieve better **HEALTH** results, as they heal irrespective of whether it has a survival impact. As the combat becomes more difficult, and especially in the 4 and 5 enemy case, we can see not only that *smart heal* becomes the best strategy, but also that *greedy heal* and *random heal* start to perform even worse than *no heal*. With more enemies, it is possible for many enemies to target the same player and one heal might not be enough to allow  $c_H$  to survive. Evaluating the current combat situation and each agent’s status becomes more crucial in deciding whether heal would be helpful, or would merely result in a lost attack opportunity.

Interestingly, the healing targeting strategy seems to have relatively little influence on the result. This is possibly caused by the enemy strategy of trying to focus on the same target as often implemented in modern games, so that most of the time we would have only one healing candidate with very low health. Smart targeting does improve *random heal*, however, as the improved candidate filtering partly compensates for the

random selection.

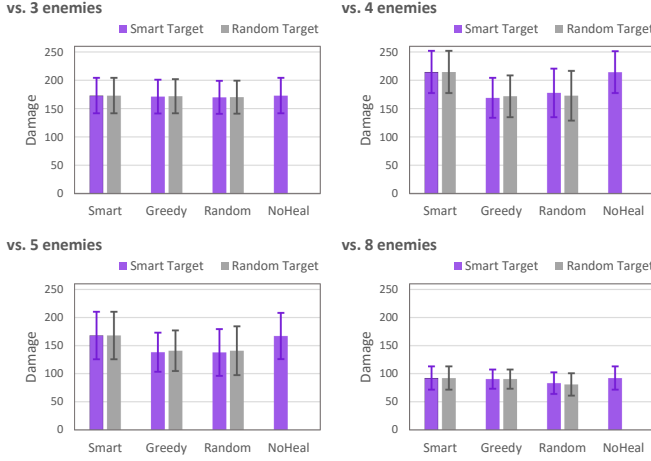


Fig. 6. Total damage dealt by players against 3–5 and 8 enemies; error bars show  $\pm 1$  standard deviation

**DAMAGE** results are shown in Figure 6. Against 3 enemies, damage dealt is the same for all strategies, even though the remaining health is different—all enemies are dying, and so healing strategy does not affect total damage dealt. In more difficult combats against 4 and 5 enemies, *smart heal* shows advantages similar to the **HEALTH** evaluation. Notice that in these cases *no heal* also tends to do well. By not including any heal moves, attacks are maximized, and thus so is the total damage score. This comes, of course, at the cost of having a much lower remaining health, as seen in Figure 5. Once combat becomes greatly unbalanced, such as against 8 enemies, all strategies are again equalized. Healing here is no longer effective, as the large number of enemies can easily eliminate a player at full health in one round, leaving no healing candidates at all.



Fig. 7. Percentage win-rate for players

Finally, **WIN** results are shown for 2–5 enemies (8 is not shown as win rate is uniformly 0%) in Figure 7. In this and

combined with the other data we can see that although *smart heal* is not necessarily better than *greedy heal* or *random heal* in terms of **HEALTH** for easier combats, the advantage of *smart heal* over other strategies gets larger and larger when combat becomes more challenging for companions, as long as winning is feasible, under all types of evaluations.

## SLEEP AND HEAL

Having used damage as a cost/benefit factor in both sleep and heal heuristics, combining our decisions for situations in which both actions are available is straightforward. For this we can test each heuristic independently. If candidates for both sleep and heal exist, then we choose the one with highest net benefit, breaking ties (arbitrarily) in favour of sleep. Targeting follows the strategies determined best for each case—highest attack for sleep, smart targeting for heal, and highest threat for the default attack action.

## Experiments

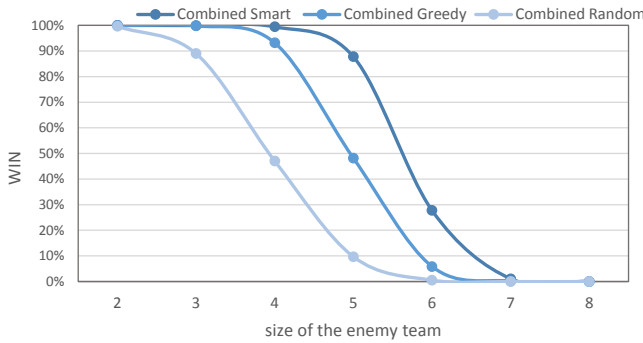
The combination of sleep and heal can be fairly powerful, and so we evaluate our combined heuristic in more complex battle scenarios. We consider two main combat groupings; one with 3 players and enemy team sizes of 2–8, similar to sleep and heal experiments, and a second group of 10 players against 8–18 enemies. In these contexts agents in both teams are randomly selected, each is allowed all 3 actions (sleep, heal, attack), and we also remove the restriction on number of times an action can be used. Although this is now less representative of *Pokémon* itself, this gives us a richer and more balanced combat simulation, with less noise in terms of which agent can do what.

We evaluate our *combined smart* strategy along with *combined greedy* and *combined random* strategies. The greedy form is intended to be similar to the default *Pokémon* AI: if an ally has health below 25% then heal them, otherwise attempt to sleep the enemy with highest attack, otherwise attack. *Combined random* randomly chooses to sleep, heal, or attack with equal probability, and acts as a baseline to see whether any heuristic, even a greedy one, is truly necessary.

Figure 8 shows the **WIN** evaluation for both groupings (**HEALTH** is quite similar, and omitted for space reasons). In the 3-player grouping all strategies decline in effectiveness over a span of 3–4 enemy team sizes, but show a clear separation, with *combined greedy* and *combined smart* able to effectively compete against 1 and 2 larger enemy team sizes than *combined random*, respectively. *Combined smart*'s performance is quite good here, still succeeding at near a 90% rate against a 5-enemy team. In larger combat scenarios, however, this advantage is reduced, and while *combined smart* and *combined greedy* both improve over *combined random*, the separation is much smaller. Our *smart* strategies are based on lower and upper bound computations, and with greater numbers of enemies these ranges become larger with significant overlap, making decisions more arbitrary. We are not able to find best moves as consistently at large scales as we can in the small scale attrition games we have focused on. It is



Win-rate of Companion Team (Size 3)



Win-rate of Companion Team (Size 10)

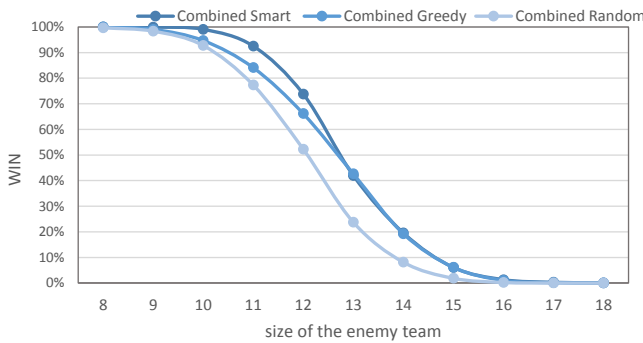


Fig. 8. Percentage win-rate for players with both sleep and heal

possible that less conservative estimation of benefit, perhaps including historical precedent or using probabilistic notions could restore the advantage, and is part of future work.

## CONCLUSIONS & FUTURE WORK

Targeting problems in attribution games are computationally difficult. The addition of common but powerful abilities such as sleep and heal magnify this complexity, and although at their core they represent a conceptually simple trade-off between lost attack opportunities and the sleep/heal effects, the presence of unknown opponent (and companion) action choices and impact of varied parametrization in the skill use and effects make good decision-making non-obvious. Our derivation, heuristic design, and experimental evaluation show that effective and efficient decisions can be made based on relatively simple calculations, and can easily achieve better results than more traditional random, or hard-coded combat choices, for sleep and heal alone or in combination, without necessarily resorting to more expensive tree-search approaches. This makes our results usable in both turn-based and real-time, CPU-constrained contexts.

It may be possible to extend our approach to include other common combat abilities, such as defensive or offensive augmentation, as they have costs and benefits which can also be expressed in terms of damage received or dealt. More complex future work is aimed at incorporating combat geometry, which would let us consider the use of area-effects, where the target

decision is more continuous, potentially affecting friendly as well as hostile units.

## ACKNOWLEDGEMENT

This work supported by the Natural Sciences and Engineering Research Council of Canada, Application ID #249902.

## REFERENCES

- [1] T. Furtak and M. Buro, "On the complexity of two-player attrition games played on graphs," in *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment Conference*. AAAI, 2010.
- [2] S. Ontañón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game AI research and competition in StarCraft," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 5, no. 4, pp. 293–311, 2013.
- [3] J. Tremblay, C. Dragert, and C. Verbrugge, "Target selection for AI companions in FPS games," in *Proceedings of the 9th International Conference on Foundations of Digital Games*, April 2014.
- [4] F. I. Muhammad, "Graph searching implementation in game programming cases using BFS and DFS algorithms," Master's thesis, Sekolah Teknik Elektro dan Informatika, 2012.
- [5] D. Churchill, A. Saffidine, and M. Buro, "Fast heuristic search for RTS game combat scenarios," in *Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AAAI, 2012.
- [6] M. Stanescu, N. Barriga, , and M. Buro, "Using Lanchester attrition laws for combat prediction in StarCraft," in *Proceedings of the Eleventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AAAI, 2015.
- [7] C. Browne, "Monte Carlo tree search," <http://mcts.ai>.
- [8] B. Abramson, *Expected-Outcome Model of Two-Player Games*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1991.
- [9] A. J. Champandard, "Monte-Carlo tree search in TOTAL WAR: ROME II's campaign AI," 2014, <http://aigamedev.com/open/coverage/mcts-rome-ii/>.
- [10] C. D. Ward and P. I. Cowling, "Monte Carlo search applied to card selection in Magic: The Gathering," in *IEEE Symposium on Computational Intelligence and Games*. IEEE, 2009, pp. 9–16.
- [11] A. Uriarte and S. Ontañón, "Automatic learning of combat models for RTS games," in *Proceedings of the Eleventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AAAI, 2015.
- [12] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte Carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.
- [13] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Iowa State University, Tech. Rep., 1998.
- [14] S. Morgan and M. S. Branicky, "Sampling-based planning for discrete spaces," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.
- [15] A. Bauer and Z. Popović, "RRT-based game level analysis, visualization, and visual refinement," in *Proceedings of the Eighth AAAI Conference on Artificial Intelligence for Interactive Digital Entertainment*. AAAI, 2012.
- [16] J. Tremblay, A. Borodovski, and C. Verbrugge, "I can jump! Exploring search algorithms for simulating platformer players," in *Experimental AI in Games Workshop (EXAG 2014)*, October 2014.
- [17] J. Tremblay, P. A. Torres, N. Rivovitch, and C. Verbrugge, "An exploration tool for predicting stealthy behaviour," in *Proceedings of the 2013 AIIDE Workshop on Artificial Intelligence in the Game Design Process*, 2013.
- [18] S. Xu, "Improving companion AI in small-scale attrition games," Master's thesis, McGill University, Montréal, Canada, November 2015.
- [19] "Pokémon essentials wiki," <http://pokemonessentials.wikia.com>.
- [20] "Bulbapedia, the community driven Pokémon encyclopedia," [http://bulbapedia.bulbagarden.net/wiki/Main\\_Page](http://bulbapedia.bulbagarden.net/wiki/Main_Page).

# User Activity Decay in Mobile Games Determined by Simple Differential Equations?

Markus Viljanen\*, Antti Airola, Tapio Pahikkala, Jukka Heikkonen

Department of Information Technology

University of Turku, Finland

\*majuvi@utu.fi

**Abstract**— Decay of population level daily user activity in Tribeflame Ltd.'s mobile games is found to be determined by elementary differential equations. We describe practical methods for investigating laws underlying the decay of daily user activity in a given cohort, known as retention in the gaming industry. Simple decay patterns are found to accurately describe this evolution. In addition to being of academic interest in sharing parallels to population growth and decay dynamics, this finding has immediate applications in the mobile games industry. Utilizing this finding allows using smaller cohorts of users in intermittent paid acquisition tests and enables game performance forecasting over long timespans.

## I. INTRODUCTION

### A. Principal Metrics in Mobile Games

Mobile games are becoming increasingly important in digital industries both in terms of market share and as a source of revenue. Recent development in mobile games has been increasing adoption of the Freemium business model. This development is not driven merely by developers imitating few astonishing success stories; free-to-play (F2P) games offering advertisements and paid extra content are now the dominant form of new releases in mobile games [1]. This transformation has made analytics increasingly central in understanding and improving freemium products. For quantifying a mobile game, retention has been used as a principal metric in the industry [2]. This is explained by freemium product dynamics: the capacity to influence the acquisition of free users is limited, but buying new users is economically feasible if the users monetize over their acquisition price. Since publishers incur virtually no cost for users using the product, assuming revenue is proportional to product use, developers seek to build products that maximize total product use.

Jargon in the mobile games industry and academia is somewhat variable; we use the following convention in order to distinguish various retention measurements. For a group of users starting the same day we define:

- $i$ 'th day **retention**  $R(i)$  is the percentage of users who were playing  $i$  calendar days from the day they started.
- $i$ 'th day **rolling retention**  $A(i)$  is the percentage of users who are active  $i$  calendar days from the day they started, where active users play on that day or any day after it.
- $i$ 'th day **lifetime retention**  $L(i)$  is the percentage of users who played more than  $i$  days in total.

Note that by definition  $R(0)=A(0)=L(0)=100\%$ . This convention makes speaking of  $i$ 'th day metric straightforward and unifies visualization graphs. The statistics offer summaries of activity decay in a group of users starting the same day, known as a cohort. Retention quantifies the decay of active product use over time, whereas Rolling Retention aims to quantify users who have not yet quit since they will eventually use the product again. Lifetime Retention summarizes total product use as counted by separate days used. Note that retention is an unbiased measure, whereas rolling retention and lifetime retention are affected by the maximum observable day.

### B. Related Research

This research can be seen as an extension to our previous research [3] which aimed to understand the behavior of a user as a stochastic process at the individual level. We now work at the population level and attempt to directly model aggregate statistics through simple dynamics. Statistical models of growth and decay dynamics are popular in many fields and have for example been utilized in a similar study of population activity [4] in connection to internet memes.

Most related gaming research has investigated various other aggregate statistics of user activity. Classifying users as purchasers and predicting purchase count was investigated in [1]. A prominent study found that Weibull distribution governs total time played [5-6] in hours. Compelling applicability was found in over 3000 Steam games, suggesting that common psychological mechanics may underlie widely divergent games. Session length and session inter-arrival times have also been found to be Weibull [7-9] or heavy-tailed distribution [10]. The investigations [9-10] seem to be measuring same statistic in daily active user population, but are in reality of altogether different character; their goal is measuring server load at different times as caused by the entire population, whereas we seek to quantify goodness of a game by finding laws that determine the decay of users starting the same day as a function of time. Lifetime considerations are closely related to player churn, which has been predicted by general purpose machine learning methods based on player features and aggregate session statistics [11-14]. Retention as a phrase was used previously in a study which aimed to adaptively improve session level retention by avoiding game states conducive to quitting [16] and in studies formulated as a regression problem predicting next 14-day [14-15] or next month [17] presence based on player features or identifying elements conducive to game design [18]. Measurements of other engagement metrics were studied in [19] and they were found to correlate.



## II. DATA SET

### A. Tribeflame Mobile Games

We gathered retention data from the following five mobile games, which vary from the Benji franchise with tens of millions of players to in-development games Dragon Fortress and Hipster Maze with user acquisition tests. These two games provide an important benchmark for measuring retention within development cycles.

TABLE I. MOBILE GAMES USED IN THE STUDY

Game	Description	Genre	Players
Benji Bananas	"Exciting and fun physics based adventure!"	Endless Runner	62 000 000
Benji Bananas Adventures	"Everyone's favorite swinging monkey returns for more action in the jungle."	Arcade Platformer	11 000 000
Mad-Croc	"Mad Croc is an endless runner where you control the fierce Mad Croc crocodile!"	Endless Runner	140 000
Hipster Maze <sup>a</sup>	"Help this unique sheep in his quest of finding the next big thing!"	Brain Puzzle	3 600
Dragon Fortress <sup>a</sup>	"Build your epic fortress and make it invincible!"	Combat City Builder	11 000

<sup>a</sup>. In Development

We selected the following start dates for retention data, with various technical issues limiting Benji data to a later start date than otherwise would have been available. Mad-Croc data begins at release and in-development games are dated to recent user tests. Collection time restricts data between cohort start date and 19.2 for all games except Dragon Fortress:

TABLE II. NUMBER OF PLAYERS

Game	Cohort Start	Cohort End	Cohort Players
Benji Bananas	2015.09.21	2016.02.19	37558
Benji Bananas Adventures	2015.09.21	2016.02.19	7886
Mad-Croc	2015.12.09	2016.02.19	5352
Hipster Maze <sup>a</sup>	2015.10.23	2016.02.19	921
Dragon Fortress <sup>a</sup>	2015.11.05	2015.12.27	510

### B. Retention Visualization

Calculating and plotting retention is an everyday activity of game analytics. We plot Benji Bananas retention from 2015.09.21, which happens to be Monday, in Figure 1. to illustrate the cohort's decay. The y-axis is log scaled, which is a common practice in visualizing growth rate type plots but has not yet been widely adopted in the industry as far as we are aware of. This tool is extremely useful for visualizing of tiny quantities combined with scientific base-10 notation used in the labels.

Retentions of the cohort are generally decreasing, but the log-scaled y-axis makes deviation clear: retention can also increase. This periodic pattern is caused by an increased weekend activity with contributions by volatility in small populations. The definition of rolling retention and lifetime

retention imply that they are monotonously decreasing from one to zero over integers, making it correct to call them discrete survival functions. The underlying random variable is conceptually 'Days Active' and 'Days Played' with survival measuring the probability of  $P[\text{'Days Active'} > T]$  and  $P[\text{'Days Played'} > N]$ , respectively. Retention looks like a survival function, but by the above discussion does not satisfy the mathematical definition and neither measures any clear variable of utility. This similarity is due to its relation to rolling retention:  $\% \text{Playing} = \% \text{Playing of Active} * \% \text{Active}$  where  $\% \text{Active}$  is given by rolling retention survival and  $\% \text{Playing of Active}$  tends to a constant with oscillations around weekends.

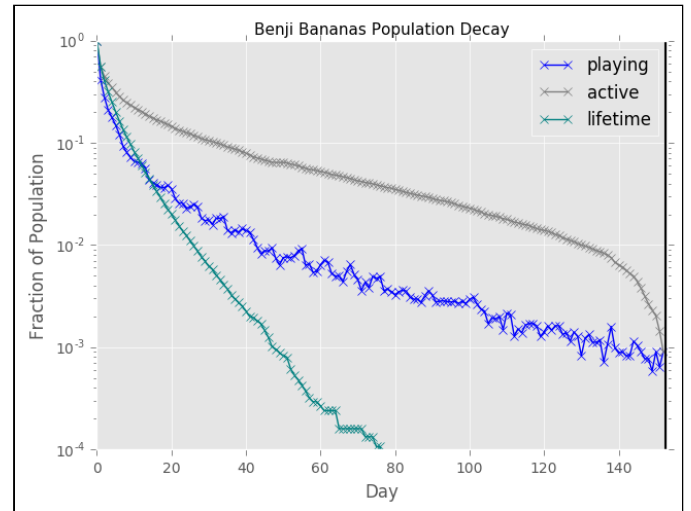


Fig. 1. Benji Bananas retention

There is a complication involved in computing rolling retention and lifetime retention, since we can only observe a certain number of days up to the current day but their definition relies on having knowledge of all future. In practice rolling retention and lifetime retention are computed assuming players do not play after the maximum observable day. This causes an abrupt downward slope in rolling retention we see in Figure 1. and a less discernible bias in lifetime retention. A user level stochastic model we presented previously [3] was able to take this into account by fitting to both inactivity and quitting the game simultaneously. In the aggregate case this is difficult to mitigate. In this study models were not fit with the biased portion in rolling retention tail and this limit is denoted with a vertical dashed line. The goal is to demonstrate that same dynamics apply across retentions and focus on extrapolating retention, which is unbiased by limited observation window.

Finally, we would like to note an important caveat. For mathematical convenience in what follows retention is handled as if it was continuous in the sense that our population function is defined on the whole positive real line. Retention is then defined by its values on positive integers  $0, 1, 2, 3, \dots$ . Only rolling retention having an immediate continuous interpretation makes sense. Retention and lifetime retention are of binned nature; retention at 1.5 days does not imply % of players playing midday the second day played and lifetime retention at 0.25 days does not mean players playing more than 8 hours.

### III. DIFFERENTIAL EQUATION OF DECAY

#### A. Mathematical Formulation

Assume that population at time  $T$  is given by a function denoted  $P(T)$  with  $P(0)=1$ . Then a useful quantity to investigate is the positive instantaneous rate of decay at time  $T$  of the remaining population:

$$H'(T) = -\frac{P'(T)}{P(T)} \quad (1)$$

The decay of the remaining population accumulated up to time  $T$  is given by integrating the decay rate:

$$H(T) = \int_0^T H'(u) du = \int_0^T -\frac{P'(u)}{P(u)} du = -\log(P(T)) \quad (2)$$

We see that the accumulated decay of population remaining is related to population at time  $T$  by a simple relation:

$$P(T) = \exp(-H(T)) \quad (3)$$

Considering retentions as a population function  $P(T)$  notice we implicitly plotted  $H(T)$  previously on an inverted y-axis by log-scaling the y-axis of  $P(T)$ . This plot allowed us to observe underlying regularity of the function, and visually reproduces in Figure 2. on an inverted natural logarithm transformed y-axis without interpolation between data points.

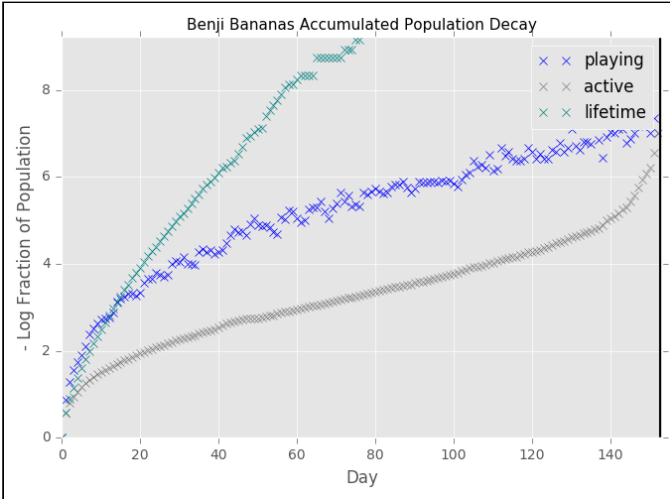


Fig. 2. Benji Bananas accumulated decay  $H(T)$

The usefulness of  $H(T)$ , or taking the negative logarithm of  $P(T)$ , is explained by how it transforms proportional changes in  $P(T)$  to linear changes in  $H(T)$ . For example, suppose that population is decreasing at a constant rate  $q$  from day  $N$  onwards. Then  $P(N+1)=P(N)(1-q)$  corresponds to a linear change in the value of the logarithm  $H(N+1) = -\log[P(N)] + \log[1/(1-q)]$  which visually presents as a slope with a constant angle.

The concept of slope is mathematically formalized in the derivative,  $H'(T)$  of  $H(T)$ , which is the foremost source to look for an explanation for regularity of an increasing function  $H(T)$ . However, it is not possible to directly investigate the derivative from data by calculations or plots because first of all we do not have a continuous function and second the function on discrete support is too noisy to render successive proportional changes  $\Delta P(T)=P(T+1)-P(T)$  informative.

#### B. Decay Rate Investigation

Our goal is to find a representation of the true function through estimating the underlying derivative. This problem is approached in stages:

1. Use methods which provide a smoothed estimate of the function  $H(T)$  and its derivative  $H'(T)$ .
2. Investigate simple functions which are powerful enough to express the derivative  $H'(T)$ .
3. Fit the corresponding population function  $\exp[-H(T)]$  to retention.

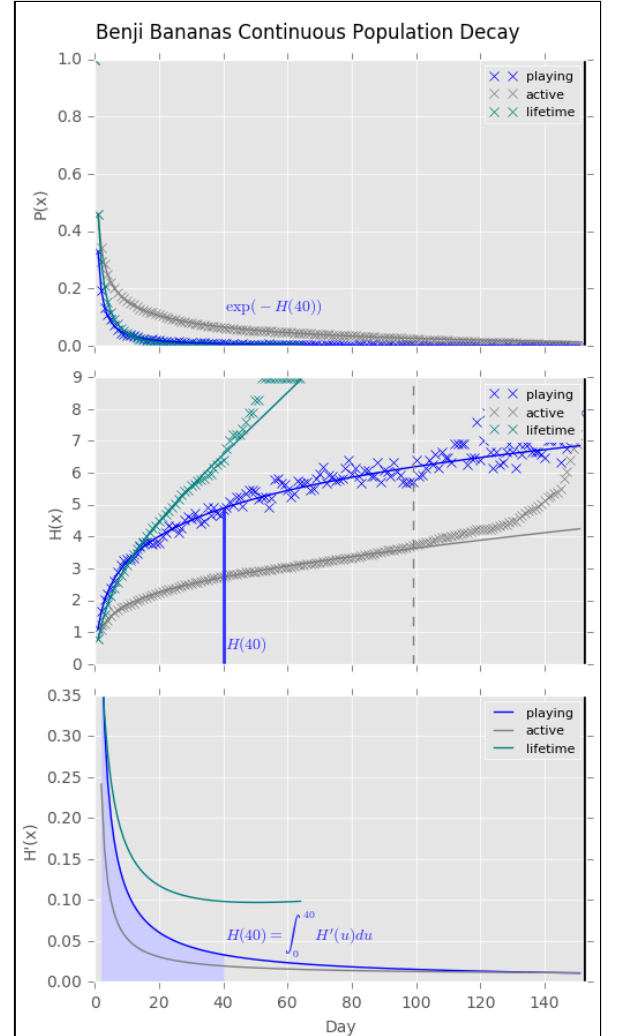


Fig. 3. Benji Bananas continuous modelling through 4<sup>th</sup> degree spline

For the first step, we chose a 4<sup>th</sup> degree smoothing spline which is a piecewise polynomial function  $f$ . It fits data points  $(x_i, y_i)$  by imposing  $w_i$  weighted tradeoff between accuracy  $y_i - f(x_i)$  and smoothness  $s$  according to a fitting budget:  $\sum_i w_i (y_i - f(x_i))^2 \leq s$ . For value  $s=0$  the smoothing spline interpolates through all data points by defining a knot point at each, and as  $s \rightarrow \infty$  it approaches a linear approximation. We defined a value for  $s$  which filtered out the noise caused by weekday activity but caused no bias. The rationale is illustrated in Figure 3.

Also illustrated in Figure 3., and formalized in the fundamental theorem of calculus, is the derivative or ‘slope’  $H'(T)$  conversely defining  $H(T)$  by a definite integral or ‘area under the curve’. We observe highly regular decay of  $H'(T)$  as a function of time with a seemingly asymptotic constant value of decay. While an unsmoothed spline would be sufficient to model a function to any degree of accuracy with respect to a data set, due to knot points it cannot be used for extrapolation.

To model retention, we hypothesize that the decay effect is universal and that there exist simple algebraic functions in the space of  $H'(T)$  which are sufficiently simple to be robust yet expressive enough to model the decay rate. In our previous investigation days played or lifetime retention corresponded to a component of the stochastic model and was verified to follow discrete Weibull distribution implying a continuous Weibull hazard [3]. In the model simple functions implicitly define various population decay functions through integration and take values on integers.

### C. Decay Observed in Tribeflame’s games

We tested the hypothesis of the regularity of decay on retention and rolling retention in Tribeflame’s mobile games and verified its validity in Figure 4 where the maximum observable day is a solid vertical line. We observe the same pattern of decreasing decay rate with an asymptotic constant rate. Smoothing spline interpolation suggests that the decay rate of active users stabilizes quite rapidly, with near constant rate around 30 days and the decay rate of playing users taking two to three times longer to reach the constant neighborhood.

These games exhibit very similar behavior: eventual constant decay rate with differences in initial churning speed. Hipster Maze has least initial churn and Mad-Croc, which utilized aggressive user acquisition, expectedly does worse in the very first days. Only possible exception is Dragon Fortress, which is still in early development with user test cohort consisting of only 510 players. Smoothing spline produced a bathtub like curve, which may reflect size of the data set or the fact that game does not hold population interest. Less volatile estimate of rolling retention verifies consistently higher churning rate, suggesting development should be continued.

Because of its applicability to Tribeflame’s mobile games, we decided to call this pattern Tribeflame retention. We cautiously suggest that it might be the underlying cause for retention in other mobile games as well. Even if this hypothesis is disproven, whatever shape the derivative realizes, this paradigm provides advantageous views into user cohort churn. It is a very interesting scientific question to map the space of decay patterns and look for possible foundational simplicity.

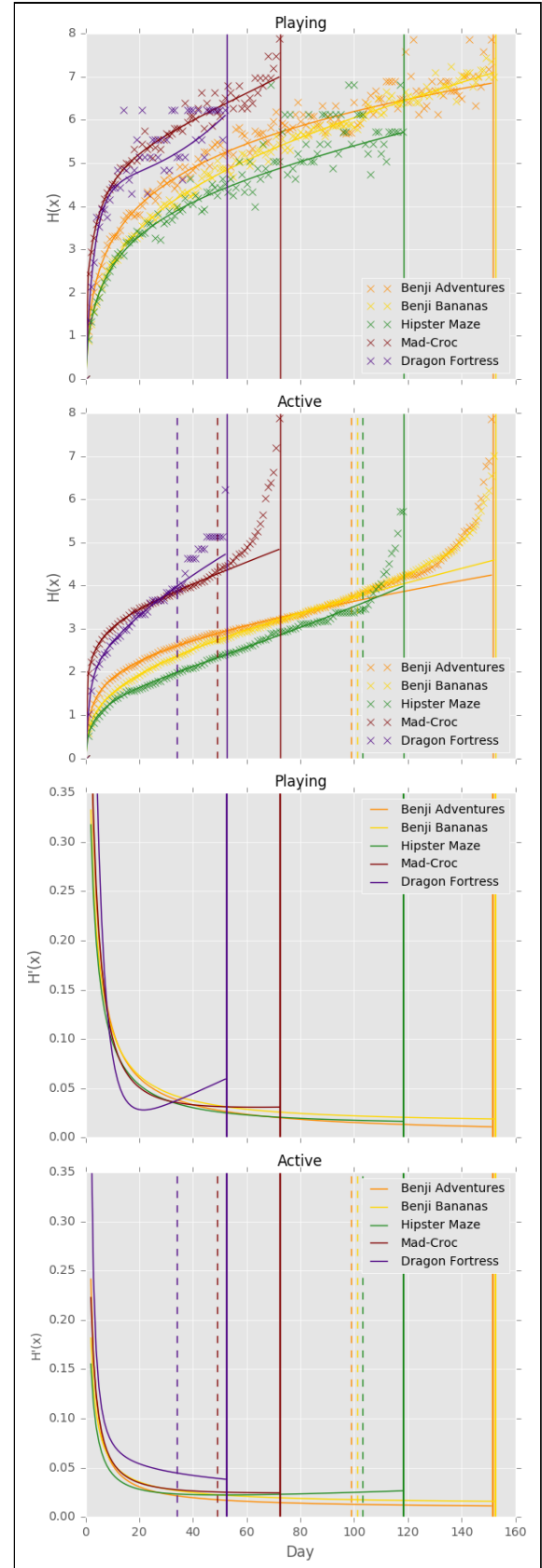


Fig. 4. Decay rates in Tribeflame’s games obtained with splines

#### IV. MODELLING THE DECAY RATE

##### A. Parametrized Models in the Decay Space

We now turn to the second part of our hypothesis, which states the decay pattern exhibits regularity which is possible to parametrize in terms of elementary functions. The decay rate  $H'(x)$  is then defined as a sum of a decaying component  $D(x) \rightarrow 0$  and an asymptotic constant rate  $c \geq 0$ :

$$H'(x) = D(x) + c \quad (4)$$

The following common functions exhibit similarity to the decaying component  $D(x)$ . The parameter intervals in Table III. denote the range where the rate of decay is decreasing, whereas their full range of definition is more extensive. They were parametrized as to make the population function appear simple, but alternatively the constant terms could be parametrized to simplify the rate of decay.

TABLE III. HYPOTHESES OF COMMON FUNCTIONS FOR  $D(x)$

$P(x)$	$D(x)$	Parameters	Related Distribution
$\exp\left[\left(\frac{x}{\beta}\right)^\alpha\right]$	$\frac{\alpha}{\beta} \left(\frac{x}{\beta}\right)^{\alpha-1}$	$0 < \alpha < 1$ $0 < \beta$	Weibull
$\frac{1}{\left(1 + \frac{x}{\beta}\right)^\alpha}$	$\frac{\alpha}{\beta + x}$	$0 < \alpha$ $0 < \beta$	Pareto
$\frac{1}{1 + \left(\frac{x}{\beta}\right)^\alpha}$	$\frac{\alpha}{\beta} \left(\frac{x}{\beta}\right)^{\alpha-1} \frac{1}{\left(1 + \left(\frac{x}{\beta}\right)^\alpha\right)}$	$0 < \alpha \leq 1$ $0 < \beta$	Log-Logistic

Related distribution denotes a distribution for which  $P(T)$  defines a survival function [20]. We refer to the population function by these names but do not adopt the statistical viewpoint because, as explained before, retention is not a survival function and we also extend these decay rates with a constant. Our previous investigation utilized and developed this approach for certain components of individual player behavior and we refer reader to it for more information [3]. The survival functions and related concepts are used extensively in survival analysis with following names [21]:

- $P(T)$ : Survival function.
- $H(T)$ : Cumulative hazard function.
- $H'(T)$ : Hazard function.

##### B. Model Fitting Target and Penalty

In this case, choosing a fitting target and a fitting penalty is a precarious problem of considerable business relevance. One may for example use retention as a central measure in business forecasts [2]. First decision arises in the choice of the fitting target: If one predicts 0.5% retention with a true value of 1.0%, what is an optimistic mistake of equal magnitude? Both answers 1.5% and 2.0% are equally valid from differing viewpoints. Penalties imposed relative to retentions' percentage

points consider 1.5% the correct answer. However, if one penalizes fit relative to the logarithm of retention or  $H(T)$  with these errors, one considers the answer 2.0% because addition in log domain translates to multiplication in ordinary domain. In this case an optimistic bias of saying game is twice as good is equal to having a pessimistic bias of saying game is half as good.

Another source of judgement arises in relative differences and absolute differences. Since retention is a generally decreasing curve which reaches very small values very rapidly, common methods of penalizing deviation from the target such as mean squared error (MSE) may not penalize what is intended. For example, predicting 10% 5-day retention with true value 11% is equally bad to predicting 0% 90-day retention with a true value of 1% which is clearly invalid for forecasting retention. Relative difference based methods like mean absolute percentage error (MAPE) make it possible to quantify deviations from very small quantities. For example, it considers 8% prediction contrasted to a true value of 10% in initial retention as bad as 0.4% prediction versus 0.5% in tail retention, which is more informative prediction from a business perspective. We also utilize mean error (ME) and mean percentage error (MPE) to measure bias. These errors are defined for retention  $P(T)$  and model  $R(T)$  over days  $T=1, \dots, N$ :

$$\mathbb{E}_{\text{MPE}} = \frac{1}{N} \sum_{T=1}^N \frac{P(T) - R(T)}{P(T)} \quad \mathbb{E}_{\text{MAPE}} = \frac{1}{N} \sum_{T=1}^N \left| \frac{P(T) - R(T)}{P(T)} \right| \quad (5)$$

$$\mathbb{E}_{\text{ME}} = \frac{1}{N} \sum_{T=1}^N P(T) - R(T) \quad \mathbb{E}_{\text{MSE}} = \frac{1}{N} \sum_{T=1}^N (P(T) - R(T))^2 \quad (6)$$

Additionally, we may be content to let go of accuracy of fits in the first week or two which is least likely to fit model assumptions and which we have already observed without substantial volatility anyway. In practice one often wishes to analyze the highly volatile tail retention and predict it in advance, with reliable estimates of the accumulated decay being empirically available. The modelling procedure could be adapted by utilizing a suitable penalized weighting procedure with this in mind.

##### C. Power of the Resulting Hypothesis Set

Corresponding to the method the smoothing spline was fit with, we use logarithm domain MSE fit to test the capacity of the parametrized decays, or the hypothesis set, to model the observed decay rate. Since our current goal is to investigate the hypothesis set and not the ability to extrapolate rolling retention, weights were set to zero in the domain where the practical computation of rolling retention caused bias by assuming players have quit. This was denoted in figures by a dashed vertical line.

Fitting the models produced Table IV. and Table V of error measurements: ME  $\pm$  MSE relative to the smoothing spline interpolation. Smallest MSE is denoted bold. We assumed the noise filtered spline to be a sufficient representation of the true



function. As can be deduced from these tables, simple rates are remarkably expressive of the decay phenomena, especially the ones that include a constant term.

TABLE IV.  $H(X)$  ME/MSE RELATIVE TO RETENTION SPLINE

Game	Weibull	Pareto	Log-Logistic	Weibull +c	Pareto +c	Log-Logistic +c
Benji	<b>0.02</b>	-0.04	-0.07	<b>0.02</b>	<b>0.01</b>	<b>0.01</b>
Bananas	$\pm 0.00$	$\pm 0.04$	$\pm 0.07$	$\pm 0.00$	$\pm 0.00$	$\pm 0.00$
Benji	0.06	0.02	0.01	0.05	<b>0.04</b>	<b>0.04</b>
Bananas	$\pm 0.02$	$\pm 0.01$	$\pm 0.01$	$\pm 0.02$	$\pm 0.00$	$\pm 0.00$
Adventures						
Hipster	0.08	<b>0.05</b>	<b>0.04</b>	0.09	<b>0.08</b>	<b>0.08</b>
Maze	$\pm 0.02$	$\pm 0.01$	$\pm 0.01$	$\pm 0.02$	$\pm 0.01$	$\pm 0.01$
Mad-Croc	0.03	0.00	0.00	<b>0.03</b>	<b>0.03</b>	<b>0.03</b>
	$\pm 0.01$	$\pm 0.01$	$\pm 0.01$	$\pm 0.00$	$\pm 0.00$	$\pm 0.00$
Dragon	0.11	<b>0.12</b>	<b>0.12</b>	0.11	<b>0.11</b>	<b>0.11</b>
Fortress	$\pm 0.10$	$\pm 0.05$	$\pm 0.05$	$\pm 0.10$	$\pm 0.05$	$\pm 0.05$

TABLE V.  $H(X)$  ME/MSE RELATIVE TO ROLLING RETENTION SPLINE

Game	Weibull	Pareto	Log-Logistic	Weibull +c	Pareto +c	Log-Logistic +c
Benji	-0.06	-0.18	-0.19	<b>-0.02</b>	<b>0.02</b>	<b>0.00</b>
Bananas	$\pm 0.01$	$\pm 0.10$	$\pm 0.10$	$\pm 0.00$	$\pm 0.00$	$\pm 0.00$
Benji	<b>-0.03</b>	-0.12	-0.11	<b>-0.03</b>	<b>0.00</b>	<b>-0.01</b>
Bananas	$\pm 0.00$	$\pm 0.05$	$\pm 0.04$	$\pm 0.00$	$\pm 0.00$	$\pm 0.00$
Adventures						
Hipster	-0.09	-0.15	-0.16	<b>-0.04</b>	<b>-0.04</b>	<b>-0.04</b>
Maze	$\pm 0.04$	$\pm 0.11$	$\pm 0.12$	$\pm 0.01$	$\pm 0.01$	$\pm 0.01$
Mad-Croc	-0.06	-0.12	-0.11	<b>-0.02</b>	<b>-0.01</b>	<b>-0.01</b>
	$\pm 0.01$	$\pm 0.05$	$\pm 0.04$	$\pm 0.00$	$\pm 0.00$	$\pm 0.00$
Dragon	<b>-0.04</b>	-0.15	-0.14	<b>-0.05</b>	<b>-0.06</b>	-0.07
Fortress	$\pm 0.01$	$\pm 0.07$	$\pm 0.07$	$\pm 0.01$	$\pm 0.01$	$\pm 0.02$

To exclude the possibility of a computation mistake, we visually verified the fidelity as illustrated for Benji Bananas in Figure 5. We added a region of vertical ticks to the spline curve because the Pareto and LogLogistic asymptotic decay rates completely overlap it. In limited models without the constant term it appears Weibull decay approaching zero is capable of providing higher tail decay which mimics a small asymptote resulting in a closer approximation.

#### D. Extrapolation of Retention using the Hypothesis Set

Since the decay rate is defined using a simple formula over the entire real line, it can be extrapolated after initial values have been observed. Therefore it should be possible to extrapolate retention as well, barring effects caused by noise and model deviations. To test actual retention accuracy we used the MPE to measure bias and the MAPE to measure deviation.

The decay function was fit with time limited data, such as 30 days in Table VI., using a simple regularization scheme of weighting the MAPE penalty in the limited data by a square root of the number of observations to trust the less volatile initial estimate more. In this case it is possible for the simpler models to outperform their more generic counterparts if they fit less noise. True error was calculated as an unweighted  $MPE \pm MAPE$  error over the entire data. We highlighted models with smallest average MAPE in each game.

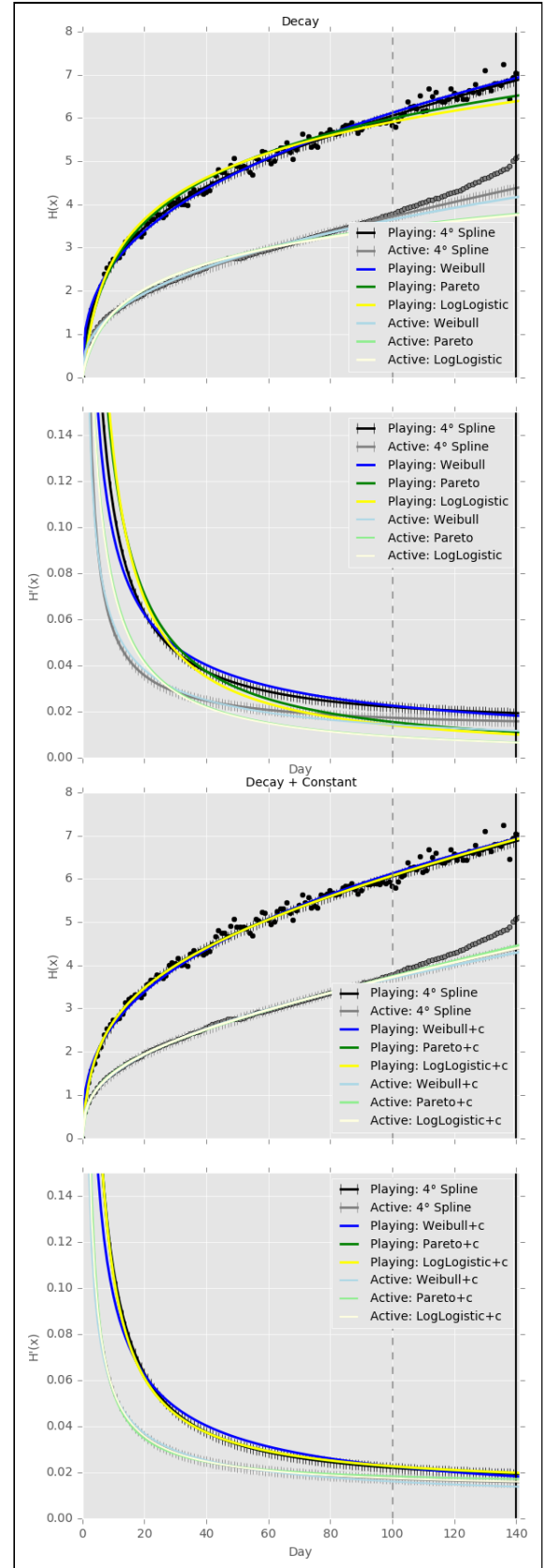


Fig. 5. Benji Bananas model decay rates contrasted to the spline decay rate

TABLE VI. P(X) MPE/MAPE RELATIVE TO EMPIRICAL RETENTION

Game	Weibull	Pareto	Log-Logistic	Weibull +c	Pareto +c	Log-Logistic +c
Benji Bananas	<b>-0.38</b> $\pm 0.39$	0.88 $\pm 0.90$	1.05 $\pm 1.07$	-0.39 $\pm 0.41$	-0.48 $\pm 0.49$	-0.46 $\pm 0.47$
Benji Bananas Adventures	-0.41 $\pm 0.43$	0.50 $\pm 0.53$	0.55 $\pm 0.58$	-0.41 $\pm 0.43$	-0.17 $\pm 0.26$	<b>-0.09</b> $\pm 0.24$
Hipster Maze	-0.32 $\pm 0.39$	0.59 $\pm 0.65$	0.63 $\pm 0.69$	<b>-0.30</b> $\pm 0.38$	-0.42 $\pm 0.46$	-0.37 $\pm 0.43$
Mad-Croc	-0.20 $\pm 0.25$	0.28 $\pm 0.32$	0.28 $\pm 0.32$	-0.20 $\pm 0.25$	0.30 $\pm 0.34$	<b>0.18</b> $\pm 0.24$
Dragon Fortress	-0.11 $\pm 0.41$	-0.09 $\pm 0.39$	-0.08 $\pm 0.38$	-0.27 $\pm 0.51$	-0.10 $\pm 0.39$	<b>-0.08</b> $\pm 0.38$

### E. Visualizations

To visualize extrapolation accuracy as a function of days, we plotted the MPE and MAPE estimates in Benji Bananas Adventures and obtained Figure 6. LogLogistic and Pareto with a constant reach the smallest MAPE error in 3-4 weeks, equivalent to having entire 151 days of data. Estimates appear generally unbiased as well, but variance in estimates suggests that practical methods call for a more regularization:

While we reasoned that 25% MAPE is small for practical purposes and may be caused entirely by noise, it is useful to confirm this intuition by visualizing the asymptotic least error state for the models. Figure 7. verifies this by plotting the log y-axis retention graph with models fit using full data set. Models with asymptotic rates seem to describe data as closely as possible.

These simple experiments demonstrate what we suspected based on the mathematical formula: the dynamic observed in the decay rate is possible to utilize in actual predictive models to achieve future forecasts of considerable reach. More sophisticated fitting methods, including improved noise and outlier tolerance, should be investigated in the future.

Finally, we present simple but limited tests which are nevertheless popular in many fields [22]. These methods are not applicable to models with the asymptotic constant rate  $c > 0$  because algebraic manipulations fail to simplify the corresponding formulas.

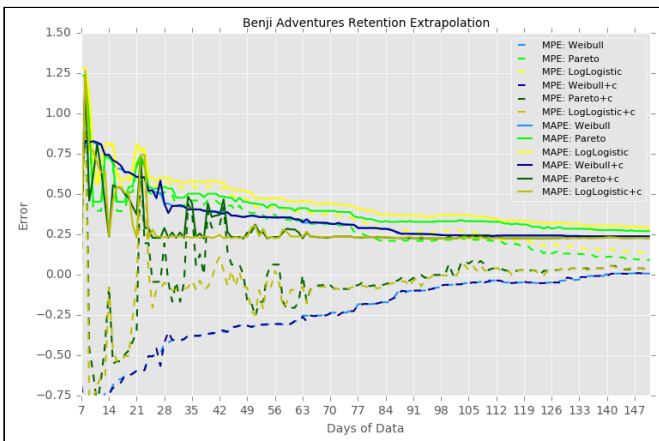


Fig. 6. MPE/MAPE of extrapolated retention for different models and limits.

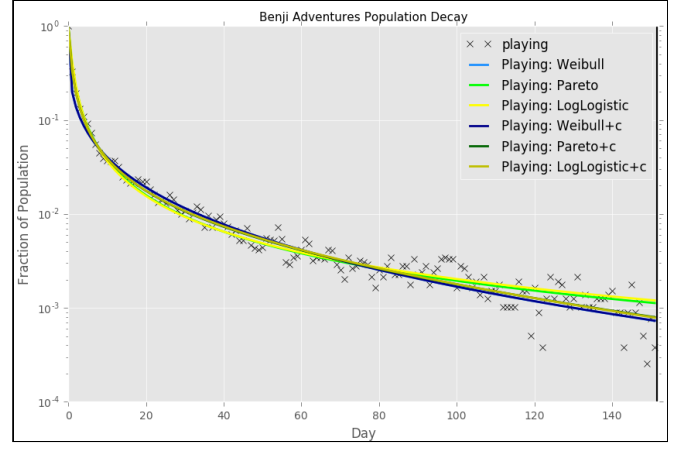


Fig. 7. Different models fit to all available data.

### F. Linearity Tests

For the model corresponding to the Weibull distribution, we can take a logarithm of the accumulated decay  $H(x) = (x/\beta)^\alpha$  and obtain [21]:

$$\log(H(x)) = \alpha \log(x) - \alpha \log(\beta) \quad (7)$$

This implies that  $H(x) = -\log(P(x))$  plotted on a logarithmic x- and y-axis is linear if this law describes the data, and we can even read off the coefficients from the axis. This result is known as a Weibull plot.

For the Pareto decay rate that we used to rigorously define a decay curve, we are not aware of such plots. However, one may adopt a simpler Pareto law with  $P(x) = (x_0/x)^\alpha$  [23] which is defined in the domain  $[x_0, \infty)$  with  $x_0 > 0$ . This is a small concession in practical applications. We then have [24]:

$$\log(P(x)) = \alpha \log(x_0) - \alpha \log(x) \quad (8)$$

This implies retention plotted on a logarithmic x- and y-axis or accumulated decay plotted on a logarithmic x-axis is linear under assumption of this law.

For the LogLogistic model one can use the decay function  $P(x) = 1/(1+(x/\beta)^\alpha)$  to derive the following linearity test that evaluates ‘odds ratio’  $(1-P(x))/P(x)$  on a logarithmic x- and y-axis [21]:

$$\log\left(\frac{1-P(x)}{P(x)}\right) = \beta \log(x) - \beta \log(\alpha) \quad (9)$$

For models with a nonzero asymptotic rate, one may utilize a concept from statistics known as a Q-Q Plot [25] and apply it to retention in Figure 8. First the decay function corresponding to the model is fit and then it is used in the place of a ‘survival function’ in the Q-Q Plot compared with ‘empirical survival’ or retention. If the law applies, the scatter plot should align without bias within the neighborhood of the diagonal  $y=x$ .

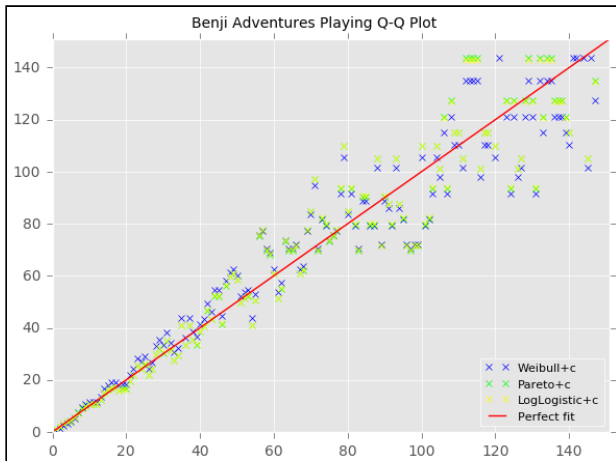


Fig. 8. Different asymptotic models combined in a Q-Q Plot.

### G. Conclusion and Future Work

We demonstrated that in Tribeflame's mobile games the decay rate of daily user activity in a given cohort, or Tribeflame retention, is described by differential equations of striking simplicity. These 2-parameter differential equations which describe the decay rate can be used to derive population decay functions which correspond to widely used distributions. In contrast, 3-parameter modifications which add a constant term result in an asymptotic decay rate and appear to fit the data perfectly. General methods of fitting the decay rate, such as smoothing splines, enable distribution free investigation of the dynamics underlying user activity.

This finding enables the industry construct practical predictive models of considerable accuracy and forecasting ability. We cautiously suggest that these decay rates may be universal and invite other researches to use tools outlined in this paper to verify or disprove the applicability of this law to other mobile games and investigate other possible decay rates.

### ACKNOWLEDGMENT

Tribeflame, Ltd. generously participated in the funding of this research, granted access to wonderful games data and continues to indulge our academic predilection.

### REFERENCES

- [1] R. Sifa, F. Hadiji, A. Drachen and J. Runge, "Predicting Purchase Decisions in Mobile Free to Play Games", *Proc. AIIDE-15*, 2015.
- [2] E. Seufert, "Freemium Economics", Morgan Kaufmann, 2014.
- [3] M. Viljanen, A. Airola, T. Pahikkala, and J. Heikkonen, "Modelling User Retention in Mobile Games", *Proc. IEEE CIG*, 2016..

- [4] C. Bauckhage, K. Kersting, F. Hadiji: Mathematical Models of Fads Explain the Temporal Dynamics of Internet Memes. *Proc. AAAI ICWSM*, 2013.
- [5] R. Sifa, C. Bauckhage and A. Drachen, "The Playtime Principle: Large-scale cross-games interest modeling," *Proc. IEEE CIG*, 2014.
- [6] C. Bauckhage, K. Kersting, R. Sifa, C. Thureau, A. Drachen, and A. Canossa, "How Players Lose Interest in Playing a Game: An Empirical Study Based on Distributions of Total Playing Times," *Proc. IEEE CIG*, 2012.
- [7] D. Pittman and C. Chris GauthierDickey, "Characterizing Virtual Populations in Massively Multiplayer Online Role-Playing Games" *Advances in Multimedia Modeling Volume 5916 of the series Lecture Notes in Computer Science*, 2010.
- [8] C. Chambers, W. Feng, S. Sahu and D. Saha "Measurement-based characterization of a collection of on-line games" *IMC*, 2005.
- [9] W. Feng, D. Brandt and D. Saha, "A Long-term Study of a Popular MMORPG", *Proc. ACM SIGCOMM WNSWG*, 2007.
- [10] T. Henderson, S. Bhatti: Modelling user behaviour in networked games, *Multimedia 2001*, 2001.
- [11] F. Hadiji, R. Sifa, A. Drachen, C. Thureau, K. Kersting and C. Bauckhage, "Predicting player churn in the wild," *Proc. IEEE CIG*, 2014.
- [12] P. Tarng, K. Chen and P. Huang, "On prophesying online gamer departure" in *Proc. 8th Ann. Workshop Netw. Syst. Support Games*, 2009.
- [13] K. Chen, P. Huang and C. Lei "Effect of Network Quality on Player Departure Behavior in Online Games", *IEEE Trans. on Parallel and Distributed Systems*, vol. 20 no. 5, 2009,
- [14] P. Rothenbuehler, J. Runge, F. Garcin and B. Faltings. "Hidden Markov Models for churn prediction", *Proc. of SAI IntelliSys*, 2015.
- [15] H. Xie, S. Devlin, D. Kudenko and P. Cowling, "Predicting player disengagement and first purchase with event-frequency based data representation", *Proc. IEEE CIG*, 2015.
- [16] B. Harrison and D. Roberts, "An Analytic and Psychometric Evaluation of Dynamic Game Adaption for Increasing Session-Level Retention in Casual Games," in *Proc. IEEE CIG*, 2015.
- [17] T. Debeauvais, C. V. Lopes, N. Yee and N. Ducheneaut, "Retention and progression: Seven months in World of Warcraft," in *Proc. 9th Int. Conf. Found. Digit. Games*, 2014.
- [18] B. Weber, M. John, M. Mateas and A. Jhala, "Modeling player retention in Madden NFL 11," in *Proc. 23rd IAAI Conf.*, 2011.
- [19] Z. Lin, C. Lewis, S. Kurniawan and J. Whitehead, "Why players start and stop playing a Chinese social network game," *J. Gam. Virtual Worlds*, vol. 5, no. 3, 2013.
- [20] J. Lawless. "Statistical models and methods for lifetime data", John Wiley & Sons, 2003.
- [21] D. Kleinbaum and M. Klein. *Survival analysis: a self-learning text*, 3rd edn. Springer, 2012.
- [22] P. Cirillo. "Are your data really Pareto distributed?" *Physica A*, 392 (23), 2013.
- [23] R. Clark, S. Cox, and G. Laslett. "Generalizations of power-law distributions applicable to sampled fault-trace lengths: model choice, parameter estimation and caveats" *Geophys. J. Int* 136 357– 372, 1999.
- [24] M. Newman, "Power laws, Pareto distributions and Zipf's law." *Contemporary Physics* 46, 323, 2005.
- [25] A. Ghasemi and S. Zahediasl. Normality tests for statistical analysis: a guide for non-statisticians. *International Journal of Endocrinology and Metabolism*, 10 (2), 2012.



# Platformer Level Design for Player Believability

Elizabeth Camilleri  
Institute of Digital Games  
University of Malta  
Msida, Malta

elizabeth.camilleri.12@um.edu.mt

Georgios N. Yannakakis  
Institute of Digital Games  
University of Malta  
Msida, Malta

georgios.yannakakis@um.edu.mt

Alexiei Dingli  
Dept. of Intelligent Computer Systems  
University of Malta  
Msida, Malta

alexiei.dingli@um.edu.mt

**Abstract**—Player believability is often defined as the ability of a game playing character to convince an observer that it is being controlled by a human. The agent’s behavior is often assumed to be the main contributor to the character’s believability. In this paper we *reframe* this core assumption and instead focus on the impact of the game environment and aspects of game design (such as level design) on the believability of the game character. To investigate the relationship between game content and believability we crowdsource rank-based annotations from subjects that view playthrough videos of various AI and human controlled agents in platformer levels of dissimilar characteristics. For this initial study we use a variant of the well-known Super Mario Bros game. We build support vector machine models of reported believability based on gameplay and level features which are extracted from the videos. The highest performing model predicts perceived player believability of a character with an accuracy of 73.31%, on average, and implies a direct relationship between level features and player believability.

## I. INTRODUCTION

Player believability is a highly subjective notion commonly viewed as the ability of a game playing character to convince observers that it is being controlled by a human player [1]–[4]. The problem of creating human-like<sup>1</sup> agents and measuring believability in agents is among the most challenging and popular research areas in the field of game artificial intelligence (AI) [5], [6]. While research in believable game bots has seen recent advances in games such as Unreal Tournament [7] and Super Mario Bros [8], generic methods for creating such bots are far from being available.

It is commonly assumed that believable game characters make games more immersive and entertaining for players [9] and that believability is solely dependent on the algorithm controlling the character’s behavior. The evident relationship between believability and character control has driven the majority of studies in the area of game AI [5], [6]. However, the extent to which believable behavior in an algorithm-controlled game agent comes about from the controller has recently been questioned [1]. Inspired by speculations in [1] we instead focus on the degree to which believability is a product of the agent’s (player’s) environment. We thus introduce a *game content-centered* view on player believability instead of the traditional *controller-centered* perspective.

Taking a crowdsourcing approach to model perceived believability of player characters in a platformer game variant of

<sup>1</sup>For the purposes of this study *human-likeness* and *believability* are two terms used interchangeably.

the well-known Super Mario Bros, we asked over 350 subjects to annotate believability in playthrough videos of the game. In these videos four different players (two AI-controlled and two humans) play dissimilar level configurations of the game. The extracted gameplay characteristics and level features, on one hand, and the obtained annotations of believability, on the other hand, were used as the input and output, respectively, to construct a believability model. The model was built using rank support vector machines (RankSVMs) [10] as the crowdsourced annotations have an ordinal (rank-based) format [11]. A correlation analysis revealed that the average width of gaps in the level has a linear relationship with reported believability. Further, the best SVM model reaches 73.31% accuracy, on average, and maps level and gameplay features to believability, revealing non-linear relationships between the enemy placement and number of gaps in the level and believability.

This paper is novel in that it introduces an approach for modeling player believability using machine learned representations of crowdsourced annotations of believability. Most importantly, it offers the first empirical assessment of the extent to which level design influences the believability of play and sheds light into the association between game content and player believability.

## II. RELATED WORK

In this section we outline studies which are relevant to this paper including work in believability and its assessment, studies in player modeling as well as earlier work on the impact of content on believability.

### A. Believability, Believable Agents and their Assessment

The notion of believability is highly subjective and cannot be objectively defined trivially. However, in virtual worlds it is generally understood as a form of suspension of the observer’s disbelief or the *ability of a fictional or virtual world or character to give the illusion of life* [1], [12]. With respect to game agents or characters there are two main dimensions of believability in literature: *character believability* (i.e., the character *itself* is perceived to be real through its behaviour, emotive expression or graphical appearance) [1] and *player believability* (i.e., by exhibiting human-like *gameplay* behaviour, the character gives the impression that it is being controlled by

a human *player*) [1]–[4]. This paper focuses on the assessment and modeling of *player believability* in platformer games.

Arguably, gameplay believability increases player engagement since it makes the game interaction more realistic [9]. There is also evidence suggesting that human players tend to prefer playing with or against other human players rather than AI agents due to the unpredictability in human gameplay behavior [1], [9]. Moreover, non-believable behavior such as repetitiveness and predictability (e.g., constantly falling into a gap in a platformer game or getting stuck in areas where human players could easily get through) seems to make games less challenging, thus deterring players [3]. ‘God-like’ behavior (e.g., going through an entire level without taking any damage) may also be considered non-believable [1]. Therefore, believability — as objectively as one can define it — likely lies somewhere in the middle of a gameplay spectrum, where the lower end includes poor, predictable behavior and the upper end includes optimal, god-like behavior; both of which are naturally perceived as non-believable.

Attempts at measuring the believability of playing behaviour include a criteria-based approach [2], [3] where believability is based on how many predefined criteria a playing character is observed to meet. However, a more common approach to measuring believability is through subjective assessment [1], [5], [13]–[17] where, similarly to the traditional Turing Test [18], human subjects are asked to observe a character in a game and indicate whether they believe it is being controlled by a human or by a computer. In this paper we follow the subjective assessment approach and we crowdsource rank annotations of believability given to video recorded player characters of a Super Mario Bros variant.

### B. Player Modeling

Player modeling has been defined as the study of computational models of players in games which includes the detection, modeling, prediction and expression of human player characteristics which are manifested through cognitive, affective and behavioral patterns [19]. Player believability can be viewed as a core component of playing behavior and player experience [1]. One could thus assess and model believability using an approach similar to how other components of player experience have been modeled [20]–[22]. There are two main approaches to modeling players and aspects of the playing experience: the model-based (top-down) and the model-free (bottom-up) approach [19], [23]. In this study we adopt a bottom-up approach for modeling player believability and we derive the computational model from data. The input of the model contains gameplay and game content data [19] of players playing through varied levels of a platformer game whereas the output contains ordinal annotations of believability. To the best of our knowledge, this is the first crowdsourcing-based study for modeling believability in games.

### C. Game Content and Believability

Game content is progressively increasing in demand and volume and, thus, becoming more expensive and time-

consuming to create manually [24]. Procedural Content Generation (PCG), the “algorithmic creation of game content with limited or indirect user input” [25], is a natural and direct response to this challenge. Experience-driven PCG is a framework which views game content as the building block of player experience and involves the procedural generation of content based on a model of player experience [23]. The framework first consists of constructing a model of player experience which takes information about game content and the player and outputs some approximation for the current player experience state. The quality of the generated content is evaluated based on the model and a representation for the content is established. New content is then generated by searching for content that optimizes the player’s experience with respect to the player experience model [23]. This core PCG approach has been applied in several studies to generate content for various player experience states including engagement, frustration, and challenge [20]–[22].

While game content and experience have an apparent direct relationship, no study has ever attempted to quantify the impact of game content on game playing believability; instead, studies in player believability focus on developing the agent’s controller [5]. That said, interactions with game content in BioShock Infinite (Irrational Games, 2013) were used to enhance *character believability* of the player’s NPC companion, Elizabeth [26]. Game mechanics have also been designed to hide non-believable characteristics of a simple algorithm in [27]. However, extending the argument that believable virtual agents must act according to the *context* they are situated within [28], we argue that the *modification* of the environment that characters act (play) within is of utmost importance for the observer’s suspension of disbelief. Inspired by the core suggestions of [1] this study attempts, for the first time, to investigate the relationship between game level architecture and game playing believability and construct models of player believability based on level design features, gameplay characteristics and crowdsourced annotations of believability.

## III. TESTBED GAME

Our testbed game is a variant of Infinite Mario Bros (a platformer game [29] and a popular benchmark in player modeling and content generation studies [20]–[22], [30]–[34]) as modified for the 2011 Mario AI Competition with sprites from an open-source platformer game called SuperTux.

The game therefore consists of a player character called Tux; a number of platforms (separated by gaps) which Tux can run on and jump between; coin collectibles for Tux to collect; and enemies which Tux must avoid or kill (by stomping on them — thus turning them into shells — or by kicking a shell to hit them). The main goal of the player in the game is to get Tux through levels containing these elements. The player is given three ‘lives’, or attempts, to complete the level. Each time Tux gets killed by falling into a gap or touching an enemy (or shell), one life is lost. If no lives are left, the game is over. Collecting coins increases the player’s score. A screenshot of



Fig. 1. Screenshot of the testbed game used in this study.

the game displaying some of the above-mentioned elements is provided in Fig. 1.

#### IV. FEATURE EXTRACTION AND DATA COLLECTION

This section outlines the features extracted from levels and gameplay behaviour and also presents the crowdsourcing process through which believability annotations were collected.

##### A. Level design features

Following the approach of [21], [22], levels of the test-bed game were represented using the following four features: the number of gaps in the level ( $G$ ), the average width of the gaps ( $G_w$ ), the number of enemies ( $E$ ), and the placement of enemies ( $E_p$ ). For a particular game level, each parameter could be in one of two states: high or low. For  $G$ , the low state was set to 2 gaps and the high state was set to 10 gaps, while for  $G_w$ , the low state was set to 3 blocks and the high state was set to 8 blocks. Further, the low value of  $E$  was chosen to be 5 enemies while the high value for this feature was chosen to be 15. The  $E_p$  feature is represented by three values which define the probabilities of an enemy being placed near a gap, near a box, and on a flat surface. For the low state of  $E_p$ , the value of these three probabilities was chosen to be 10%, 10% and 80%, respectively, whereas those for the high state were chosen to be 80%, 10% and 10%. The choice of these features and their corresponding values which were set empirically was inspired by earlier studies in Super Mario level generation [20]–[22], [30], [31], [34]. Given the two states for each of the four level features, the resulting number of all their combinations, and thereby possible levels, is 16. The 16 levels were generated following the approach presented in [21].

##### B. Gameplay features

The following list of fourteen player metrics used for this study is based on earlier feature extraction attempts for level generation in Super Mario Bros [21], [22]: completion time ( $t_C$ ), duration of last life ( $t_{LL}$ ), percentage of time spent running right ( $t_R$ ), percentage of time spent running left ( $t_L$ ), percentage of time spent running ( $t_{Run}$ ), number of times the ‘run’ button was pressed ( $P_{Run}$ ), number of jumps ( $J$ ), number of aimless jumps ( $J_a$ ), number of coins collected ( $C$ ), number of enemy shells kicked ( $S$ ), number of deaths by



Fig. 2. Screenshot of the 4-AFC part of the online questionnaire.

falling into a gap ( $D_g$ ), number of deaths by an enemy ( $D_e$ ), number of enemies killed ( $K$ ), number of enemies killed by kicking a shell ( $K_s$ ).

##### C. Crowdsourced believability annotations

We recorded video clips of four players (two human players and two AI agents) playing through each of the sixteen generated levels. The two humans play the game differently with one clearly performing better than the other. The first AI agent is based on the A\* pathfinding algorithm of Baumgarten [35] which won the Mario AI competition in 2009. The second AI agent is a hard-coded rule-based agent inspired by the REALM agent [36] which won the Turing Track of the Mario AI championship in 2010. Collectively, the four players purposely demonstrate varied behavior across all levels played and are therefore assumed to exhibit different levels of believability. The resulting 64 videos of all 16 levels played by all 4 players were stored for the believability annotation experiment described herein.

Human annotators were asked to view a number of gameplay videos which were randomly selected from the 64 videos without replacement. To establish the ground truth of a highly subjective notion such as believability we followed the rank-based approach for reliable annotation as proposed in [11]. The rank-based approach requires that annotators are provided with instances of the investigated variable in pairs (or more) and are asked to rank those instances according to a particular notion. For eliminating any short-term memory biases we chose to present the 64 videos in pairs, amounting to 2016 unique combinations in total. For each pair viewed, the observer was requested to provide a pairwise preference for believability using the 4-alternative forced choice (4-AFC) protocol [20]–[22]. That is, the subject was asked to specify which of the games in the two videos they believed *was more likely to have been played by a human*. Subjects could pick one of four possible responses; the game in video A, the game in video B, both or neither (see Fig. 2). The two videos in the pair were presented in a random order next to each other so that any primacy or recency effects are eliminated.

Prior to proceeding with video annotation the subjects were required to fill in a brief demographics questionnaire. The questions asked were: age; gender; how often do you play games (possible answers: never; few times a month; few times

a week; few hours a day; many hours a day); how would you rate your skill level in playing video games (possible answers: novice; average; good; excellent); have you ever played platformer games (possible answers: never; a few times, I am a novice player; many times, I am a good player; I am an expert player); have you ever played Super Mario Bros (possible answers: never; a few times, I am a novice player; many times, I am a good player; I am an expert player).

The crowdsourcing experiment was advertised widely and run for a whole month during which a total of 1,605 randomly selected video pairs were ranked by approximately 391 subjects. This amounts to 79.6% of all possible combinations of pairs (i.e., 2016). Subjects reported clearly about their preference (i.e., they selected one of the two videos) in 984 out of the 1,605 available video pairs; this is the *clear* preference dataset used for the analysis in the remaining of this paper.

## V. STATISTICAL ANALYSIS

As a first phase of our analysis, this section presents various descriptive statistics we derived from the obtained data. This includes a brief analysis on the demographical data of the subjects, an analysis of the quality of the believability preferences and a correlation analysis between the game features considered and the reported believability annotations.

### A. Demographical Data

A high-level descriptive statistical analysis on the demographical data of the respondents reveals that the subjects (average age is 36.9 years) are relatively balanced in terms of gender (females: 197; males: 168). However, the distribution of demographical data is slightly biased in terms of gaming skills (e.g., only 15 out of 365 subjects play many hours a day), gaming experience (e.g., only 30 subjects identify themselves as expert gamers), and previous exposure to platformer games (e.g., only 15 subjects identify themselves as expert platformer gamers) and Super Mario Bros (e.g., only 13 subjects identify themselves as expert Super Mario Bros gamers).

### B. Believability Preferences

Assessing the validity of annotations of highly subjective notions such as believability is a challenging task. For instance, a believability preference for one of two videos depicting different AI players (instead of annotating these videos with the *neither* label) is not objectively invalid since AI players of the game may have the capacity of being perceived as believable. The analysis presented here is not intended to test the validity of the reported believability preferences but, rather, to provide an insight into the nature of these annotations following the evaluation approach presented in [1]. Figure 3 offers a first visualization of the crowdsourced believability preferences with respect to the four characters (the two AIs and the two human players) and three types of subjects: *all* subjects, *expert* subjects who consider themselves a good or an expert Super Mario Bros player, and *novice* subjects who either have never played Super Mario Bros before or they consider themselves novice Super Mario Bros players. For

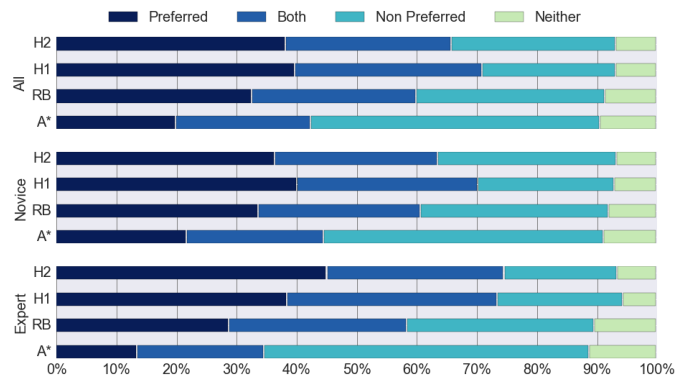


Fig. 3. Percentage of believability annotations across the different player characters and annotator sets. A\* and RB (rule-based) are the two AI players; H1 and H2 are the two human players. “Expert”, “Novice” and “All” indicate the expert, novice and all subjects, respectively.

each of the four players and the three annotator types we depict the percentage of the four possible responses.

What is apparent from Fig. 3 is that expert players, compared to novice players, are capable of better distinguishing the two AIs from the two human players. Further, it is clear that the preferences obtained generally match with the true nature of the players being assessed. In particular, the human players were correctly identified as humans by the majority of subjects. The RB player was able to convince a great number of subjects (especially the novices) that it was likely to be human in most of the video combinations that it was featured. On the other hand, the A\* player, which was much more optimal in its actions, was not perceived as believable (especially by the experts). This observation reinforces the findings of Togelius et al. [1] in that there seems to be some lower and upper boundaries for the relation between the playing skill of the players and their believability. To some extent, this may serve as an indication that the quality of the results obtained in this study are at least on par in terms of validity with those obtained in [1].

Although through crowdsourcing we risked obtaining noisy or incorrect data (e.g., preference pairs provided by subjects who made their choices arbitrarily or who did not understand the questionnaire task), these results show that there is a certain degree of agreement between the responses of the majority of subjects, implying that there might indeed be some common reference point (or ground truth) with regards to human perception of player believability. This finding further reinforces the assumption that aspects of believability can be approximated and there are certain factors that affect believability which are common to many.

### C. Correlation analysis

In this section we examine the relationship between each of the game features and the reported believability preferences through a correlation analysis. The rank correlation coefficient between each of the eighteen gameplay and level features and the crowdsourced preference pairs was calculated using the test

statistic  $c(z) = 1/N \sum_{i=1}^N z_i$ , where  $N$  is the total number of *clear* preference pairs (i.e., where one of the video clips in the pair is preferred over the other) and  $z_i = 1$  if the value of the examined feature in the preferred video of the pair is greater than that of the other video (i.e., there is a match) and  $z_i = -1$  otherwise (i.e., the feature value is lower in the preferred video and, thus, there is a mismatch). Statistical significance was calculated via a one-tailed binomial test.

The correlation coefficients of all 18 features are provided in Table I across the three different subject datasets: *all* subjects (984 clear preferences), *expert* subjects (206 clear preferences), and *novice* subjects (774 clear preferences). It can clearly be observed that — independently of subject class — 7 out of the 18 features are found to be highly and positively correlated ( $p < 0.01$ ) with reported preferences. These include one level feature, and six gameplay features. When all subjects are considered, findings suggest that believability is perceived higher in levels where gaps are wider ( $G_w$ ), and when players spend more time running left ( $t_L$ ), are killed more often by enemies ( $D_e$ ), kill more enemies by kicking shells ( $K_s$ ), kick more shells ( $S$ ), press the ‘run’ button more often ( $P_{RUN}$ ) and take longer to die in their last life ( $t_{LL}$ ). Indeed, running left and making use of shells are considered human playing characteristics and the results are in line with the findings in [1]. An interesting observation is that the above gameplay features have also been found to be associated with reported player *fun* in Super Mario Bros [21].

The difference between the expert and novice subjects is only limited to the degree of effect as some substantial differences are observed in the  $c(z)$  values. Further, it seems that for the expert annotators, as opposed to novice annotators, the number of times the ‘run’ button was pressed and the duration of the player’s last life were not factors that contribute to a character’s believability. On the other hand, the average gap width was not an indicator of believability for the novice annotators.

From these correlations, it is clear that there exist some linear relationships between aspects of level design and perceived player believability. In particular, it seems that the wider the gaps are in a level, the more the behaviour of the players is showcased; as a result, this level feature seems to be a good predictor of player believability across all four players — particularly for the expert annotators. Interestingly enough, the gap width has already been found to be a good predictor of player *challenge*, *predictability*, *anxiety* and *boredom* in an earlier study in the Super Mario Bros game [21]. That said, an additional, preliminary correlation analysis was also conducted to measure the effect of the features on believability for each of the four players separately. While no effects were found for the rule-based agent,  $G_w$  and  $D_g$  are positively and highly correlated ( $p < 0.05$ ) with perceived believability in both the A\* agent and one of the human players (H1). Further,  $K_s$  seems to be a good predictor for the believability of both human players while  $P_{RUN}$ ,  $t_L$  and  $D_e$  are highly correlated with the perceived believability of H1.

The correlation analysis presented above limits our findings

TABLE I  
RANK CORRELATIONS  $c(z)$  BETWEEN ALL EXAMINED FEATURES AND REPORTED BELIEVABILITY PREFERENCES. VALUES IN BOLD INDICATE SIGNIFICANCE ( $p < 0.01$ ).

Feature	All	Expert	Novice
$G_w$	<b>0.1291</b>	<b>0.2821</b>	0.0846
$G$	0.0825	0.0891	0.0840
$E_p$	-0.0503	0.0874	-0.0867
$E$	0.0303	0.1778	-0.0050
$t_L$	<b>0.2483</b>	<b>0.3548</b>	<b>0.2170</b>
$D_e$	<b>0.2134</b>	<b>0.3121</b>	<b>0.1900</b>
$K_s$	<b>0.2911</b>	<b>0.4805</b>	<b>0.2342</b>
$S$	<b>0.3043</b>	<b>0.3898</b>	<b>0.2744</b>
$P_{RUN}$	<b>0.1082</b>	0.0632	<b>0.1179</b>
$t_{LL}$	<b>0.0966</b>	0.0891	<b>0.0964</b>
$J$	0.0700	0.0792	0.0652
$J_a$	0.0636	0.0707	0.0593
$C$	-0.0698	-0.1097	-0.0617
$t_R$	0.0526	0.0051	0.0658
$D_g$	0.0593	0.1389	0.0373
$K$	-0.0309	-0.0726	-0.0232
$t_C$	-0.0260	-0.1179	-0.0041
$t_{RUN}$	-0.0010	0.0647	-0.0213

to *linear* relationships between individual features and annotated believability. However, the effect that features have on believability is clearly dependent on the player (represented by gameplay features). This means that there are relationships in between the features themselves (both level and gameplay features) which may affect how well they can predict believability. Therefore, in the next section, we investigate the creation of *nonlinear* mappings between combinations of features and reported believability via preference learning.

## VI. PREFERENCE LEARNING FOR MODELING BELIEVABILITY

Preference learning is the task of learning global rankings. Assuming that there is an underlying global order that characterizes the provided rank annotations, the data can be machine learned via preference learning and a global ranking of believability can be derived [37]. This section provides a brief description of the preference learning methodology followed to construct models of player believability: in particular we discuss the feature selection method and the preference learning algorithm adopted. For all experiments reported in this paper we used the Preference Learning Toolbox (PLT) [38]. PLT is an open-source software package which integrates various data normalization, feature selection and machine learning algorithms for the task of learning from preferences.

### A. Feature Selection

To select the most meaningful features that maximize the predictive capacity of our computational model of believability we used the Sequential Backward Selection (SBS) algorithm for all experiments presented in this paper. The SBS algorithm starts by feeding the preference learning algorithm with all available features and obtaining a performance value; performance is measured through cross-validation accuracy in this paper. At each iteration, SBS removes from the feature set



the feature which, upon its removal, improves the model's performance (accuracy) the most. This process continues until the feature set contains a single feature or until a certain accuracy threshold is reached.

### B. RankSVM

This study's core aim is to infer a computational mapping between level and gameplay features (input) and the believability preferences (output). While the PLT offers a number of preference learning algorithms to choose from, we choose the RankSVM [10] algorithm for its comparatively low computational effort and well-evidenced performance. RankSVM is the preference learning variant of Support Vector Machines (SVMs) introduced by Joachims [10]. SVMs are models which map training instances to data points in a typically high-dimensional space and attempt to divide the data points into two categories via a hyperplane. The algorithm tries to find the dimension (or hyperplane) which best separates the training instances. Once the model is built, unseen instances are mapped to the space represented by the model and an output is produced based on which half of the space they are mapped to [39]. The mapping of training instances to data points in space is performed using a kernel function. In this study, we use the radial basis function (RBF) kernel as it yields superior performance to linear or polynomial kernels compared via trial experiments.

## VII. BELIEVABILITY MODEL CONSTRUCTION

In this section, we describe the core preference learning experiments carried out in order to investigate both the impact of the annotators' experience with respect to Super Mario Bros and the impact of level content on the accuracy of a believability model. In all experiments reported in this section, features are min-max normalized to  $[0, 1]$  and RankSVM (see Section VI-B) uses the RBF kernel with the default  $\gamma$  parameter value of 1. Finally, three-fold cross validation is used as the performance measure of all derived models.

### A. The Impact of Super Mario Bros Experience

As mentioned earlier in Section IV-C, prior to the video annotation process, subjects were asked if they had ever played Super Mario Bros (i.e., the game that the testbed game is largely based on) before and could respond with one of four different levels of experience. We assume that a level of experience with a particular game may have an impact on a person's perception of believability in that game. That assumption is, in part, validated in the correlation analysis of Section V-C. Thus, in this first round of experiments, we vary the data used for training based on reported experience with Super Mario Bros and investigate their impact on the accuracy of the computational model of perceived believability. For that purpose, as in Section V-C, the full dataset (984 clear preferences) was split into two subsets: the *novice* (774 clear preferences) and the *expert* (206 clear preferences) annotators. We run SBS for each of the three sets and depict progression of the average 3-fold cross-validation accuracy in Fig. 4.

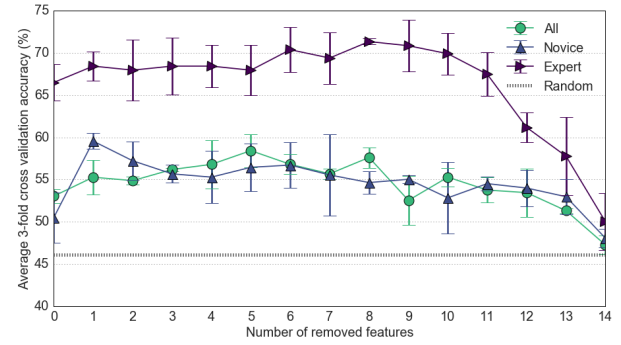


Fig. 4. Impact of Super Mario Bros Experience: 3-fold cross-validation accuracy as features are removed over iterations from the feature set using SBS for all, expert and novice subjects. The x-axis represents the iteration of the SBS process — or the number of removed features. Level features are not considered by SBS in this experiment. The random baseline represents the average accuracy of 10 random multi-layer perceptrons. Error bars depict standard error values.

It is important to note that in this set of experiments, the level features are not considered during the SBS process (i.e., they are purposely forced in the input of the SVM). The purpose of doing so is to allow the model sufficient capacity to capture aspects of reported believability through the content of the level. Furthermore, the performance of such a model can be used as a baseline against any model that does not put an emphasis on content features. In the next section all considered features (gameplay and content) are treated equally, thereby testing the degree to which level features are important for modeling gameplay believability.

The results illustrated in Fig. 4 reveal that the believability annotations provided by subjects who considered themselves to be 'good' or 'expert' players of Super Mario Bros ('Expert') managed to yield significantly higher accuracies compared to the other two datasets (all data and data annotations from novice Super Mario Bros players) and also achieve the highest accuracy improvement over the iterations of SBS. The highest accuracy of a model built solely on expert annotations is that of 71.36% with a corresponding feature set containing all four level features (since they were forced in this set of experiments) and ten gameplay features:  $t_C$  (completion time),  $t_{LL}$  (duration of last life),  $P_{RUN}$  (number of times the run button was pressed),  $S$  (number of kicked shells),  $J$  (number of jumps) and  $J_a$  (aimless jumps). The novice annotators did not seem to yield any significant improvement over the full set of preferences from all annotators. The best accuracy obtained when training RankSVMs on the full dataset is only 58.43%.

### B. The Impact of Level Features

In the second round of experiments we chose to treat all features equally and consider them all during the SBS process so as to examine which would be picked as appropriate for capturing reported believability. Given the successful results of the expert subset of annotators in Section VII-A, in this round of experiments we focus on the expert subset and we compare the accuracy obtained between models that are



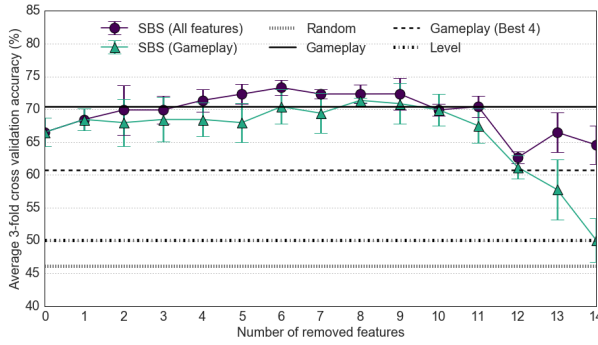


Fig. 5. Impact of level features: 3-fold cross-validation accuracy as features are removed over iterations from the feature set using SBS for the expert annotators. The x-axis represents the iteration of the SBS process — or the number of removed features. Level features are either considered — SBS (All features) — or not — SBS (Gameplay) — by SBS. The random baseline represents the average accuracy of 10 random multi-layer perceptrons. ‘Gameplay’, ‘Gameplay (Best 4)’, and ‘Level’ represent, respectively, the accuracy of SVMs featuring all gameplay, the best 4 gameplay, and all level features. Error bars depict standard error values.

trained when we consider and when we do not consider level features during the feature selection process. The progression of the average model accuracy for the two approaches for the expert annotators is shown in Fig. 5. In addition to the random baseline, the figure contains three more baseline performances: the average 3-fold cross-validation accuracy of an SVM containing solely the gameplay features, another one containing the best four gameplay features, and finally, one containing only the four game level features.

As shown in Fig. 5, the model featuring all (14) gameplay features clearly outperformed the model featuring all (4) level features. The latter is unlikely to be disadvantaged due to the lower number of features since the model featuring the best 4 gameplay features also outperforms it. Nevertheless, the selected features for the best performing model (73.31% accuracy — an improvement over the highest accuracy obtained in Section VII-A) still contains the level features  $G$  (number of gaps),  $G_w$  (average gap width), and  $E_p$  (enemy placement) in addition to 9 gameplay features:  $t_C$ ,  $J$ ,  $P_{Run}$ ,  $t_{Run}$ ,  $t_R$ ,  $K_s$ ,  $S$ ,  $J_a$  and  $D_e$ . Although  $G_w$  was the only level feature found to be correlated with annotated believability, RankSVMs were able to capture non-linear relationships between more level features (i.e.,  $G$  and  $E_p$ ) and gameplay features, and reported believability. Even though the model performance improvement is not large, it is however clear that the best predictors of player believability consist of a combination of gameplay and level features, as also speculated in [1]. Further, it should be noted that the accuracy of 73.31% is considered very satisfactory given the highly subjective notion of believability.

### VIII. DISCUSSION

The core findings of this paper suggest that there is a strong and direct relationship between level design and the perceived believability of a character within a level. This validates, to an extent, the hypothesis that level design influences the

perception of believability (at least in the platformer genre). Nevertheless, a number of limitations of this study are very likely to have hindered even more promising findings from emerging. First, the performance of all four players may have impacted the believability preferences as some subjects might have based their reasoning on the players’ ability to complete levels. More varied and expressive agents would have equipped us with a much wider range of behaviors in the spectrum of believability. Second, the level and gameplay features chosen for this study are a subset of all possible game features that could be encapsulated. We can only envisage that a larger set of features can potentially reveal more information about the relationship we are studying; however, the small level feature set allowed us to both study the relationship and make the crowdsourcing experiment feasible to run (by preventing a combinatorial explosion of the total number of required videos). Third, while RankSVM is a reliable algorithm for learning preferences, other algorithms including backpropagation and neuroevolutionary preference learning must be tested on the data available. Finally, this study is only based on data derived from one particular game; experiments for testing the generality of the methods and the results on other platformer games and other game genres is a future goal of this work.

Despite current limitations, the initial findings of this study suggest that it is possible to optimize player believability merely by modifying the game level architecture, without necessarily adjusting the character behavior per se. Moreover, these findings hint towards the possible use of other forms of game content, beyond levels, to optimize player believability. Finally, other domains such as those of embodied conversational agents and robotics could also benefit from the outcomes of this work by putting an emphasis on modifying the environment within which agents act rather than focusing solely on their controllers to optimize their believable behaviour.

### IX. CONCLUSIONS

In this study, we have introduced a data-driven modeling approach for constructing a mapping between gameplay, level design and believability. To crowdsource that mapping, we recorded a number of videos showing the gameplay of two human and two AI-controlled players — varying in playing style — playing through platformer levels of dissimilar design. Through the use of an online questionnaire, more than 350 annotators provided pairwise preferences of perceived believability for the characters appearing in the videos.

A first statistical analysis of the data revealed that the annotators’ experience with the game impacts their preferences; further, the average gap width of a level and a number of gameplay features were found to be highly correlated to perceived believability. Then, the 1,605 believability preferences, on one hand, and the gameplay and level features of each video, on the other, defined the output and input, respectively, of a preference learning process via RankSVMs that constructed the desired mapping. Through several experiments, we examined the impact of annotators’ experience with Super Mario Bros, and the features considered by the model, on the

model's accuracy. The best accuracy of 73.31% was obtained when both level and gameplay features were considered by the model and the feature selection mechanism. The core findings of this study reveal both a linear and a non-linear relationship between level design and player believability that needs to be further explored in other game genres and domains.

#### ACKNOWLEDGMENTS

We would like to thank all participants of the crowdsourcing experiment. This work has been supported in part by the FP7 Marie Curie CIG project AutoGameDesign (630665).

#### REFERENCES

- [1] J. Togelius, G. N. Yannakakis, S. Karakovskiy, and N. Shaker, *Believable Bots: Can Computers Play Like People?* Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ch. Assessing Believability, pp. 215–230. [Online]. Available: [http://dx.doi.org/10.1007/978-3-642-32323-2\\_9](http://dx.doi.org/10.1007/978-3-642-32323-2_9)
- [2] T. Hinkkanen, J. Kurhila, and T. A. Pasanen, "Framework for evaluating believability of non-player characters in games," in *Workshop on Artificial Intelligence in Games*, 2008, p. 40.
- [3] D. Livingstone, "Turing's test and believable ai in games," *Comput. Entertain.*, vol. 4, no. 1, Jan. 2006. [Online]. Available: <http://doi.acm.org/10.1145/1111293.1111303>
- [4] F. Tencé, C. Buche, P. D. Loor, and O. Marc, "The challenge of believability in video games: Definitions, agents models and imitation learning," *CoRR*, vol. abs/1009.0451, 2010. [Online]. Available: <http://arxiv.org/abs/1009.0451>
- [5] P. Hingston, *Believable Bots*. Springer, 2012.
- [6] G. N. Yannakakis and J. Togelius, "A panorama of artificial and computational intelligence in games," *IEEE Transactions on Computational Intelligence and AI in Games*, 2014.
- [7] J. Schrum, I. V. Karpov, and R. Mäikkyläinen, "Human-like combat behaviour via multiobjective neuroevolution," in *Believable bots*. Springer, 2013, pp. 119–150.
- [8] N. Shaker, J. Togelius, G. N. Yannakakis, L. Poovanna, V. S. Ethiraj, S. J. Johansson, R. G. Reynolds, L. K. Heather, T. Schumann, and M. Gallagher, "The turing test track of the 2012 mario ai championship: entries and evaluation," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013, pp. 1–8.
- [9] J. D. Miles and R. Tashakkori, "Improving the Believability of Non-Player Characters in Simulations," in *Proceedings of the 2nd Conference on Artificial General Intelligence (2009)*. Atlantis Press, 2009.
- [10] T. Joachims, "Optimizing search engines using clickthrough data," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2002, pp. 133–142.
- [11] G. N. Yannakakis and H. P. Martínez, "Ratings are Overrated!" *Frontiers in ICT*, vol. 2, p. 13, 2015.
- [12] J. Bates, *The nature of characters in interactive worlds and the Oz project*, 1992.
- [13] I. Umarov and M. Mozgovoy, "Believable and effective ai agents in virtual worlds: Current state and future perspectives," *International Journal of Gaming and Computer-Mediated Simulations (IJGCMS)*, vol. 4, no. 2, pp. 37–59, 2012.
- [14] S. McGlinchey and D. Livingstone, "What believability testing can tell us," in *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design, and Education*, 2004, pp. 273–277.
- [15] B. Gorman, C. Thureau, C. Bauckhage, and M. Humphrys, "Believability testing and Bayesian imitation in interactive computer games," in *From Animals to Animats 9 : Proceedings. 9th International Conference on Simulation of Adaptive Behavior*, S. Nolfi, Ed. Springer-Verlag, 2006.
- [16] B. Mac Namee, "Proactive persistent agents-using situational intelligence to create support characters in character-centric computer games," Ph.D. dissertation, 2004.
- [17] J. E. Laird and J. C. Duchi, "Creating human-like synthetic characters with multiple skill levels: A case study using the soar quakebot," *Proceedings of the AAAI 2000 Fall Symposium: Simulating Human Agents*, pp. 54–58, November 2000.
- [18] P. Hingston, "A turing test for computer game bots," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 1, no. 3, pp. 169–186, 2009.
- [19] G. N. Yannakakis, P. Spronck, D. Loiacono, and E. André, "Player modeling," *Dagstuhl Follow-Ups*, vol. 6, 2013.
- [20] N. Shaker, S. Asteriadis, G. N. Yannakakis, and K. Karpouzis, "Fusing visual and behavioral cues for modeling user experience in games," *Cybernetics, IEEE Transactions on*, vol. 43, no. 6, pp. 1519–1531, 2013.
- [21] C. Pedersen, J. Togelius, and G. N. Yannakakis, "Modeling player experience for content creation," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 2, no. 1, pp. 54–67, 2010.
- [22] N. Shaker, G. N. Yannakakis, and J. Togelius, "Towards Automatic Personalized Content Generation for Platform Games," 2010.
- [23] G. N. Yannakakis and J. Togelius, "Experience-Driven Procedural Content Generation," *IEEE Transactions on Affective Computing*, vol. 2, no. 3, pp. 147–161, July 2011.
- [24] M. Hendrikx, S. Meijer, J. Van Der Velden, and A. Iosup, "Procedural content generation for games: A survey," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 9, no. 1, pp. 1:1–1:22, Feb. 2013. [Online]. Available: <http://doi.acm.org/10.1145/2422956.2422957>
- [25] J. Togelius, N. Shaker, and M. J. Nelson, "Introduction," in *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*, N. Shaker, J. Togelius, and M. J. Nelson, Eds. Springer, 2015.
- [26] IGN, "BioShock Infinite - The Revolutionary AI Behind Elizabeth," <https://www.youtube.com/watch?v=2viudg2jsE8>, Mar 2013.
- [27] M. Cerny, "Sarah and sally: Creating a likeable and competent ai sidekick for a videogame," in *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.
- [28] F. de Rosi, C. Pelachaud, I. Poggi, V. Carofiglio, and B. De Carolis, "From Greta's Mind to Her Face: Modelling the Dynamics of Affective States in a Conversational Embodied Agent," *Int. J. Hum.-Comput. Stud.*, vol. 59, no. 1-2, pp. 81–118, Jul. 2003. [Online]. Available: [http://dx.doi.org/10.1016/S1071-5819\(03\)00020-X](http://dx.doi.org/10.1016/S1071-5819(03)00020-X)
- [29] G. Smith, M. Cha, and J. Whitehead, "A framework for analysis of 2d platformer levels," in *Proceedings of the 2008 ACM SIGGRAPH Symposium on Video Games*, ser. Sandbox '08. New York, NY, USA: ACM, 2008, pp. 75–80. [Online]. Available: <http://doi.acm.org/10.1145/1401843.1401858>
- [30] N. Shaker, G. N. Yannakakis, and J. Togelius, "Digging deeper into platform game level design: session size and sequential features," in *Applications of Evolutionary Computation*. Springer, 2012, pp. 275–284.
- [31] —, "Crowdsourcing the Aesthetics of Platform Games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 5, no. 3, pp. 276–290, Sept 2013.
- [32] J. Ortega, N. Shaker, J. Togelius, and G. Yannakakis, "Imitating human playing styles in super mario bros," *Entertainment Computing*, vol. 4, no. 2, pp. 93–104, 4 2013.
- [33] N. Shaker, M. Nicolau, G. N. Yannakakis, J. Togelius, and M. O'Neill, "Evolving levels for super mario bros using grammatical evolution," in *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, Sept 2012, pp. 304–311.
- [34] N. Shaker, G. N. Yannakakis, and J. Togelius, "Feature analysis for modeling game content quality," in *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*, Aug 2011, pp. 126–133.
- [35] J. Togelius, S. Karakovskiy, and R. Baumgarten, "The 2009 Mario AI competition," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE, 2010, pp. 1–8.
- [36] S. Bojarski and C. B. Congdon, "REALM: A rule-based evolutionary computation agent that learns to play Mario," in *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*. IEEE, 2010, pp. 83–90.
- [37] L. Rigutini, T. Papini, M. Maggini, and M. Bianchini, *Artificial Neural Networks - ICANN 2008: 18th International Conference, Prague, Czech Republic, September 3-6, 2008, Proceedings, Part II*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, ch. A Neural Network Approach for Learning Object Ranking, pp. 899–908. [Online]. Available: [http://dx.doi.org/10.1007/978-3-540-87559-8\\_93](http://dx.doi.org/10.1007/978-3-540-87559-8_93)
- [38] V. E. Farrugia, H. P. Martínez, and G. N. Yannakakis, "The preference learning toolbox," *arXiv preprint arXiv:1506.01709*, 2015.
- [39] A. Ben-Hur and J. Weston, "A user's guide to support vector machines," *Data mining techniques for the life sciences*, pp. 223–239, 2010.

# Predicting Retention in Sandbox Games with Tensor Factorization-based Representation Learning

Rafet Sifa\*, Sridev Srikanth<sup>†</sup>, Anders Drachen<sup>‡</sup>, Cesar Ojeda\* and Christian Bauckhage\*<sup>†</sup>

\*Fraunhofer IAIS, 53745 St. Augustin, Germany

<sup>†</sup>B-IT, University of Bonn, 53113 Bonn Germany

<sup>‡</sup>Aalborg University and The Pagonis Network, Aalborg, Denmark

**Abstract**—Major commercial (AAA) games increasingly transit to a semi-persistent or persistent format in order to extend the value of the game to the player, and to add new sources of revenue beyond basic retail sales. Given this shift in the design of AAA titles, game analytics needs to address new types of problems, notably the problem of forecasting future player behavior. This is because player retention is a key factor in driving revenue in semi-persistent titles, for example via downloadable content. This paper introduces a model for predicting retention of players in AAA games and provides a tensor-based spatio-temporal model for analyzing player trajectories in 3D games. We show how knowledge as to trajectories can help with predicting player retention. Furthermore, we describe two new algorithms for three way DEDICOM including a fast gradient method and a semi-nonnegative constrained method. These approaches are validated against a detailed behavioral data set from the AAA open-world game *Just Cause 2*.

## I. INTRODUCTION

Aspects of player behavior that are of importance from the point of view game development companies vary. With respect to company sizes, business models, types and formats of games, hardware platforms, and game design, there are different analytical methods. An important aspect is the business model, notably whether Free-to-Play (F2P) or retail-based, which in the former case leads to an interest in predicting the behavior of the players. As of this writing, there is a general lack of predictive models in games being released based on a retail business model. This makes sense given the non-persistent nature of most retail games. However, the situation is changing as the game industry has realized that there are revenue streams available from semi-persistent or persistent games that can be tapped into to increase overall revenue. Therefore, more and more games are released to facilitate longer periods of play, and the lifetime of games is extended.

Changes in how retail based games can be designed to extend the relationship with the player are particularly noticeable among for game productions involving large production and marketing costs, typically referred to as “AAA games”. Here we specifically focus on non-persistent AAA games for which there is now an increasing interest in extending the player-game interaction period by enabling online play via introducing downloadable contents (DLCs) or episodic play, allowing player-generated mods, or similar mechanics. From the point of view of game design, this requires new forms of analytical support including the prediction of player behavior, retention, churn, monetization, or social interactions.

Prediction is however a virtually unexplored topic in AAA games with a few noticeable exceptions [1]. The test case we consider here is the action-adventure game *Just Cause 2* (JC2). JC2 is an Open-World Game (OWG) with large degrees of freedom in player navigation, in world interaction, and narration. While AAA games vary too much in their design to generalize across, it should be noted that spatio-temporal navigation and -tactics are important in many of these games. Spatio-temporal freedom is a characteristic of OWGs but these dimensions are important in any digital game, and have also formed a focus in research in game analytics [2]. Space and time are essentially the dimensions through which user experience occurs, and integrating these in behavioral analysis thus enables the study of gameplay as it is experienced by the player.

## A. Contribution

To the best of our knowledge, the work presented here is the first attempt to build a combined retention prediction model for AAA games using an ensemble approach and tensor models for player-wise representation learning and yielding actionable results (up to 81% accuracy) even for the high degrees of player freedom in the OWG format. Secondly, we show that the spatio-temporal dimensions of player behavior can inform the prediction of retention in OWG games, possibly because these are also the dimensions of the player experience and thus a potential proxy of this experience. This result contrasts work in F2P mobile games where spatio-temporal information has not been utilized but retention/churn prediction has been highly successful [1], [3]. Thirdly, we introduce tensor DEDICOM (T-DEDICOM) into the domain of behavioral analytics and present new algorithms for finding its factors efficiently. That is, we develop a tensor-based spatio-temporal model based on the DEDICOM technique that has previously been successfully applied to mine player based spatio-temporal and migration patterns in games [4], [5]. With T-DEDICOM it now becomes possible to account for the movement of many players in one sitting by allowing the spatio-temporal representation to be learned for each player while retaining global information for interpretation.

## B. Related work

The idea of studying player behavior to inform game design and development dates back to the earliest digital games but

TABLE I: Dataset description

Feature Type	Descriptor(s)
<i>Telemetry</i>	Number of sessions
	Total playtime
	Total absence time
	Number of days
	Number of actions
	Progress
	Parachutes used
	Vehicles used
	Enemies killed
	Weapons used
	Explosives used
	Number of deaths
	Causes of deaths
	Difficulty level
<i>Meta</i>	Heavy-drop item count
	Blackmarket item count
<i>Temporal</i>	Language
	Platform-played
	Time between daily first and last session
	Inter-day time distribution
<i>Composite</i>	Inter-session time distribution
	Correlation coefficients on time
	Intercept and standard error on time
	Mean and deviation on Time

has significantly advanced with the emergence of MMOGs in the 1990s, due to the need to monitor and respond to a persistent customer base. With the rise of social networking platforms (such as Facebook) and mobile devices and the introduction of the freemium business model, behavioral analytics has now become an integral part of game development [2], [6].

Since an exhaustive coverage of work on game analytics is beyond the scope of this paper, we will focus our discussion on two research directions:

1) Behavioral prediction in games has generally targeted either future player (customer) behavior or sought to inform a variety of situations related to Game AI. Regarding the former the focus has been on persistent games, either F2P mobile/online games [1], [7] or MMOGs [8]. Methods range from pattern recognition and historical analysis over forecasting using simple regression methods to standard machine learning techniques such as decision trees [1], [6], support vector machines [7], and Hidden Markov Models (HMMs) [3]. Hadiji et al. [1] and Runge et al. [3] investigated churn prediction in F2P games using a binary classification approach and benchmarked several methods to predict churn. Sifa et al. [6], Xie et al. [7] and others targeted the problem of predicting purchase structures in games and Thawonmas et al. [8] analyzed player revisitation in a MMOG using login frequencies as the principal proxy. Weber et al. [9] used regression modeling to evaluate retention in *Madden NFL 11*.

2) Spatio-temporal analysis of player behavior has revealed temporal features to play an important role in every predictive game analytics [2]. Operating with both spatial and temporal dimensions, however, is comparatively rare in game analytics but has a strong tradition in Game AI where, for instance, agent behavior models require both dimensions to function. Bauckhage et al. [4] adopted DEDICOM to cluster players of *Quake: Arena* and develop waypoint graphs for behavior-based partitioning. This work in particular forms the basis for the spatio-temporal considerations in the work presented here.

### C. Just Cause 2: Gameplay

Just Cause 2 is an open-world (or sandbox) game characterized by a 3D environment covering over 1000 square kilometers of virtual real estate depicting the fictional island nation of Panau and substantial degrees of freedom in terms of spatio-temporal behavior, actions, and narrative progression. The game was published by Square Enix in 2010 and retains an active player base, selling over 6 million units. Similar to other open-world titles, the environment of JC2 is highly varied. Players assume the role of an agent working for an outfit called the Agency. Their goal is to cause disruption leading to the downfall of the resident dictator's reign. This disruption is caused by weakening the power base of the dictator and by taking control of the 9 administrative units of Panau. Players thus have to travel and fight their way across Panau and the game encourages them to traverse the map through missions and exploration, which can be done using a wide variety of vehicles (cars, trucks, boats, planes, helicopters, ...). The game also features a grappling hook which allows players to pull themselves quickly towards stationary or moving objects and can be used in combination with a parachute to rapidly move around.

## II. DATA SET AND PRE-PROCESSING

The data set we used in this study has been extracted from Square Enix metrics suite containing records of randomly sampled 5331 players with 7 months coverage and 10,794,666 recorded timestamped actions with their locations. We used a sample of 3572 of the players that have played at least three days to account for the dependencies between the daily behavior. Our objective is a retention prediction case [1] in which we observe the behavioral activities of each player within a 14 days period and determine if they play the game after the following 7 days. In the following we define important behavioral features based on [1], [6], [10] to provide a baseline feature set for retention prediction and explain the main data structure that allows us to model player movements given spatio-temporal actions of players.

### A. Extracting Behavioral Telemetry Features

To capture aspects of player behavior, we extract game specific and game independent features that also apply to games of similar genre or setting and therefore group them into four categories: *Telemetry*, *Meta*, *Temporal*, and *Composite*.

In the category called *Telemetry* [6], we have a set of aggregated features quantifying players' basic game play aspects (such as number of sessions or total playtime) and game-play related features (such as progress and number of weapons used), whereas the category *Meta* groups information about the language of the player and the platform on which they play the game. Previous studies [1], [6] showed the importance of temporal features for purchase and churn and we group particular time related features quantifying player's appearance under the *Temporal* category. These include information about the inter-session and -day time distribution as well as the distribution of the daily time interval between the earliest

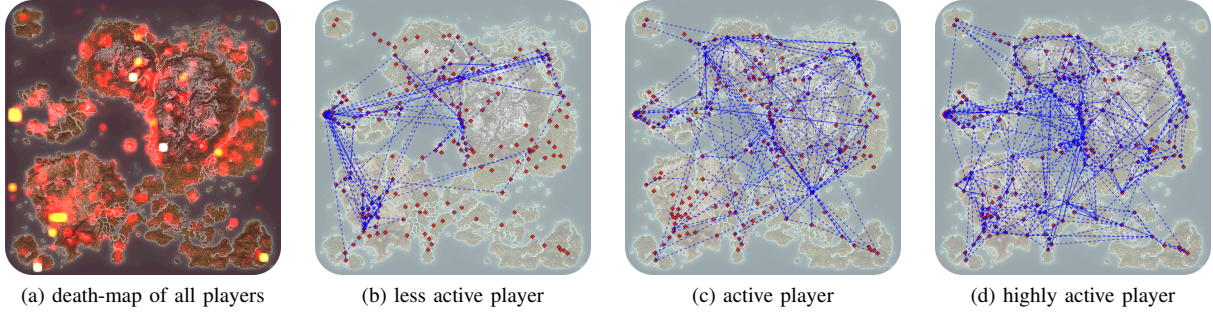


Fig. 1: Spatial activities of players across the islands of Panau. (a) illustrates death-map of all players and (b, c, d) show the movements of three players between the automatically detected waypoints. Best seen in color.

and the latest session. Finally, we consider the daily and session-wise evolution of features in categories *Telemetry* and *Temporal* by defining the category *Composite* that contains features about the mean, deviation, standard error, correlation, correlation coefficient, and the intercept of the values on time. An overview of the extracted behavioral features and their categories are given in Tbl. I.

### B. Waypoint Learning with Neural Networks

Player trajectories in JC2 can be described as locally dense but spatially more scattered than the ones we analyzed in [4] due to the increased size and complexity of the game world and game mechanics, see Fig. 1. It is important to note that the use of land-, sea-, and air-vehicles as well as the game specific use of grappling hook as a means of transportation provide nearly unlimited freedom for navigation.

In order to dynamically learn sectors visited by players, we build a *waypoint transition graph* from all observed players trajectories. Waypoint transition graphs provide an informative way to organize spatio-temporal information. They allow for partitioning game maps into coherent parts and can also account for temporal aspects such as transitions between parts [4], [11]. First, we find *prototypical* waypoints capturing the topology of the data points at hand and assign them to be the vertices of a waypoint transition graph. Formally, given a player's position  $\mathbf{x}_t \in \mathbb{R}^3$  at time  $t$  and player trajectories  $X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_q\}$  that contain  $q \in \mathbb{N}$  observations, we determine  $n \in \mathbb{N}$  waypoints  $W = \{\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_n\}$  where  $\mathbf{w}_i \subset \mathbb{R}^3$  and  $n \ll q$ . Due to the complexity of the world in JC2, we used the Neural Gas Algorithm [11], [12] for learning the map topology. In our experiments, we found the Neural Gas to converge faster and to provide more robust data for predicting player behavior than  $k$ -means clustering. Second of all, once topology-capturing waypoints are obtained, a waypoint transition graph  $G = (V, E)$  with vertices  $V = W$  and edges  $E \subseteq V \times V$  is constructed for *each player* by considering movements between waypoints. Namely, for each time-asymmetric pair of subsequent locations  $(\mathbf{x}_t, \mathbf{x}_{t+1})$ , we find their closest waypoints  $\mathbf{w}_i$  and  $\mathbf{w}_j$  respectively assign a directed edge between the  $i$ -th and  $j$ -th node of the graph.

Figures 1b, 1c and 1d illustrate waypoints learned from the movements of all observed players and movements between waypoints for particular players. In the next section we study a tensor factorization model to partition asymmetric similarity matrices while preserving the directional information.

## III. DEDICOM MODELS

Decomposition into directional components (DEDICOM) is a family of matrix and tensor decomposition methods originally introduced to analyze asymmetric social ties between people [13]. DEDICOM models deal with asymmetric similarity matrices and decompose them into low rank counterparts involving a loading matrix and a family of affinity matrices [4], [5], [13], [14]. DEDICOM has successfully been used in variety of context including social network analysis [15], natural language processing [16], entity resolution [14], population based player churn analysis [5], and spatio-temporal analysis of player trajectories [4]. Following the analogy introduced in [4], we particularly aim to build a player-based spatio-temporal feature learning framework to compress high level in game interactions to low dimensional representations that also preserves directional information of movements. To do so, we study two- and three-way DEDICOM models and propose an easy-to-implement, novel, and hybrid algorithm that applies projected gradient descent to find DEDICOM factors faster than previous methods. For consistency we start with the two-way DEDICOM model that deals with a single similarity matrix and continue with three-way DEDICOM that allows to decompose multiple similarity matrices.

Formally given a matrix  $S \in \mathbb{R}^{n \times n}$  containing asymmetric relations between  $n$  entities and  $k \in \mathbb{N}$  where  $k \ll n$ , DEDICOM aims to find the following factorization

$$S \approx ARA^T \quad (1)$$

where  $A \in \mathbb{R}^{n \times k}$  and  $R \in \mathbb{R}^{k \times k}$ . The resulting loading matrix  $A$  denotes the hidden structures in  $S$  and the affinity matrix  $R$  encodes the asymmetric relations between those structures. Finding a DEDICOM partitioning can be cast as a matrix norm minimization problem with the following loss



function that depends on  $\mathbf{A}$  and  $\mathbf{R}$

$$E(\mathbf{A}, \mathbf{R}) = \left\| \mathbf{S} - \mathbf{A}\mathbf{R}\mathbf{A}^T \right\|^2. \quad (2)$$

Typically, alternating least squares algorithms are used that minimize (2) by fixing one of the factor matrices and solving for the other. Various constrained and unconstrained versions have been studied to increase interpretability and speeding up computations [4], [5], [17]. Examples as to these include semi non-negativity constrained DEDICOM [5] which forces  $\mathbf{R}$  to be non-negative to interpret the relations as proportions and Decomposition Into Simple Models (DESICOM) [17] which constrains matrix  $\mathbf{A}$  to be sparse. For more details about the above models and implementation details, we refer the readers to [4], [5], [15].

The models discussed so far are able handle two dimensional asymmetric similarity data. Given a whole set of asymmetric similarity matrices, we can generalize the above methods to find patterns here as well [13], [14]. Formally, we group a set of  $m \in \mathbb{N}$  asymmetric similarity matrices (a.k.a *slices*)  $\{\mathbf{S}_1, \mathbf{S}_2, \dots, \mathbf{S}_m\}$  in a three dimensional array (or a *third order tensor*)  $\mathcal{S} \in \mathbb{R}^{n \times n \times m}$  where  $s_{ijr}$  defines the directional relation between  $i$ -th and  $j$ -th entity in the  $r$ -th slice. Three dimensional generalizations of DEDICOM have been studied from two points of view to reveal the underlying directional patterns. The first in [13], [15] represents the global relationships between all of the slices by encoding them in an asymmetric affinity matrix and each slice's affinity is doubly scaled by a particular diagonal matrix. The second in [14] represents the affinity matrix of each slice individually allowing for a more relaxed representation with fewer factors to optimize over. In the context of individual player trajectory analysis, the later approach has the advantage of learning a player specific representations. Namely given a tensor  $\mathcal{S}$ , a relaxed Tensor-DEDICOM partitioning is defined as

$$\mathbf{S}_r \approx \mathbf{A}\mathbf{R}_r\mathbf{A}^T \quad \forall r \in [1, 2, \dots, m] \quad (3)$$

where  $\mathbf{A} \in \mathbb{R}^{n \times k}$  is the loading matrix and  $\mathbf{R}_r \in \mathbb{R}^{k \times k}$  is an affinity matrix which is the  $r$ -th slice of  $\mathcal{R} \in \mathbb{R}^{k \times k \times m}$ .

Similar to the procedure for its two-way counterpart, finding a three way DEDICOM partitioning can be cast as a norm minimization problem for the loading matrix  $\mathbf{A}$  and tensor of affinities  $\mathcal{R}$  as

$$E'(\mathbf{A}, \mathbf{R}_1, \mathbf{R}_2, \dots, \mathbf{R}_m) = \sum_{r=1}^m \left\| \mathbf{S}_r - \mathbf{A}\mathbf{R}_r\mathbf{A}^T \right\|^2. \quad (4)$$

Following an alternating least square scheme we can minimize (4) for  $\mathbf{A}$  and each slice of  $\mathcal{R}$  independently by keeping the rest of the factors fixed. It is important to note that the loss function defined in (4) is convex in any arbitrary slice of  $\mathcal{R}$  but not in  $\mathbf{A}$ . This leads us to consider approximate solutions for  $\mathbf{A}$  which can be approached in numerous ways. For this purpose, we generalize the algorithms derived for two way DEDICOM [4], [5], [15] and propose two novel algorithms to find optimal DEDICOM factors and to generalize the notion of semi non-negativity for interpretability of the factors. Both

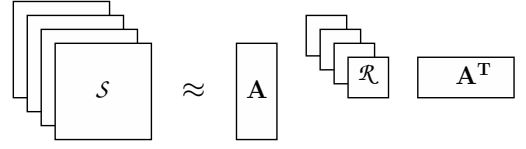


Fig. 2: Illustration of relaxed Three Way DEDICOM. Asymmetrical relationships in  $\mathcal{S} \in \mathbb{R}^{n \times n \times m}$  are decomposed into a combination of a latent factor matrix  $\mathbf{A} \in \mathbb{R}^{n \times k}$  and a collection of asymmetric mode transition matrices  $\mathcal{R} \in \mathbb{R}^{k \times k \times m}$ .

algorithms can be easily implemented by extending the python scripts in [4].

#### A. Approximate Alternating Least Squares Algorithm

Our first algorithm is based on approximating the solution of a matrix equation formed by stacking the data matrices  $\{\mathbf{S}_1, \mathbf{S}_2, \dots, \mathbf{S}_m\}$  and their transposes  $\{\mathbf{S}_1^T, \mathbf{S}_2^T, \dots, \mathbf{S}_m^T\}$  together as  $\mathbf{T} = [\mathbf{S}_1 \mathbf{S}_1^T \dots \mathbf{S}_m \mathbf{S}_m^T]$ . Considering the DEDICOM approximation of the each block of  $\mathbf{T}$  we have

$$\mathbf{T} = \mathbf{A} \begin{bmatrix} \mathbf{R}_1 & \mathbf{R}_1^T & \dots & \mathbf{R}_m & \mathbf{R}_m^T \end{bmatrix} \begin{bmatrix} \mathbf{I}_{2m} \otimes \mathbf{A}^T \end{bmatrix}, \quad (5)$$

where  $\mathbf{I}_{2m}$  is the  $2m \times 2m$  identity matrix and  $\otimes$  denotes the Kronecker product. The approximation comes into play if we solve for  $\mathbf{A}$  by holding  $\mathbf{A}^T$  fixed [4], [14], [15]. Namely, if we define the right hand side of  $\mathbf{A}$  in (5) as

$$\mathbf{B} = \begin{bmatrix} \mathbf{R}_1 & \mathbf{R}_1^T & \dots & \mathbf{R}_m & \mathbf{R}_m^T \end{bmatrix} \begin{bmatrix} \mathbf{I}_{2m} \otimes \mathbf{A}^T \end{bmatrix} \quad (6)$$

and assume  $\mathbf{A}^T$  being fixed, the problem of finding optimal  $\mathbf{A}$  becomes a matrix regression problem with a close form solution given by

$$\mathbf{A} = \mathbf{T}\mathbf{B}^\dagger = \mathbf{T}\mathbf{B}^T(\mathbf{B}\mathbf{B}^T)^{-1}. \quad (7)$$

Therefore, substituting (6) in (7), we obtain the following ALS update for  $\mathbf{A}$ :

$$\mathbf{A} \leftarrow \left( \sum_{r=1}^m \mathbf{S}_r \mathbf{A} \mathbf{R}_r^T + \mathbf{S}_r^T \mathbf{A} \mathbf{R}_r \right) \left( \sum_{r=1}^m \mathbf{C}_r + \mathbf{D}_r \right)^{-1}, \quad (8)$$

where  $\mathbf{C}_r = \mathbf{R}_r \mathbf{A}^T \mathbf{A} \mathbf{R}_r^T$  and  $\mathbf{D}_r = \mathbf{R}_r^T \mathbf{A}^T \mathbf{A} \mathbf{R}_r$ . Next, since minimization (4) for a fixed  $\mathbf{A}$  is a matrix regression problem [5], [15] the update for each slice  $\mathbf{R}_r$  of  $\mathcal{R}$  is defined as

$$\mathbf{R}_r \leftarrow \mathbf{A}^\dagger \mathbf{R}_r \mathbf{A}^{T\dagger} \quad \forall r \in [1, 2, \dots, m]. \quad (9)$$

It is important to note that, when working with large matrices, to avoid numerical instabilities during the implementation and work with smaller matrices, optimization for  $\mathbf{R}_r$  can be carried out by projecting  $\mathcal{X}$  onto a basis of  $\mathbf{A}$  using  $QR$  decomposition of  $\mathbf{A}$  [14], [15].

#### B. Projected Gradient Descent for Constrained DEDICOM

We next derive an algorithm that is based on the generalization of the projected gradient descent algorithm (called Hybrid Orthogonal [HO] DEDICOM) introduced in [5] to find optimal DEDICOM factors. Unlike the approximate ALS algorithm, by introducing an orthogonality constraint, our method uses



**Algorithm 1** Three-way HO-DEDICOM

---

Randomly initialize  $\mathbf{A}$  and  $\mathcal{R}$

**while** Stopping condition is not satisfied **do**

    //Compute the gradient

$$\frac{\partial E'}{\partial \mathbf{A}} \leftarrow -2 \sum_{r=1}^m \left( \mathbf{S}_r^T \mathbf{A} \mathbf{R}_r + \mathbf{S}_r \mathbf{A} \mathbf{R}_r^T \right)$$

    //Update  $\mathbf{A}$  in the gradient's opposite direction

$$\mathbf{A} \leftarrow \mathbf{A} - \eta_A \frac{\partial E'}{\partial \mathbf{A}}$$

    //Project  $\mathbf{A}_{t+1}$  by means of  $QR$ -Decomposition

$$\mathbf{A}, \mathbf{U} \leftarrow QR(\mathbf{A})$$

    //Update slices of  $\mathcal{R}$  in ALS manner

**for**  $r \in [1, 2, \dots, m]$  **do**

$$\mathbf{R}_r \leftarrow \mathbf{A}^T \mathbf{S}_r \mathbf{A}$$

**end for**

**end while**

---

each occurrence of  $\mathbf{A}$  when minimizing (4) and this results in a computationally more efficient method that avoids matrix inversions in the updates of  $\mathbf{A}$  and  $\mathbf{R}$ . Starting with the update for  $\mathbf{A}$ , we can write (4) in terms of traces as

$$E'(\mathbf{A}, \mathbf{R}_1, \mathbf{R}_2, \dots, \mathbf{R}_m) = \sum_{r=1}^m \text{tr}[\mathbf{S}_r^T \mathbf{S}_r] - 2 \text{tr}[\mathbf{S}_r^T \mathbf{A} \mathbf{R}_r \mathbf{A}^T] + \text{tr}[\mathbf{A} \mathbf{R}_r^T \mathbf{A}^T \mathbf{A} \mathbf{R}_r \mathbf{A}^T]. \quad (10)$$

Due to the orthogonality constraint imposed on  $\mathbf{A}$ , i.e.  $\mathbf{A}^T \mathbf{A} = \mathbf{I}_k$ , and the invariance under cyclic permutation of the traces, the gradient matrix of  $E'$  boils down to

$$\frac{\partial E'}{\partial \mathbf{A}} = -2 \sum_{r=1}^m \frac{\partial}{\partial \mathbf{A}} \text{tr}[\mathbf{S}_r^T \mathbf{A} \mathbf{R}_r \mathbf{A}^T] \quad (11)$$

which results in

$$\frac{\partial E'}{\partial \mathbf{A}} = -2 \sum_{r=1}^m \left( \mathbf{S}_r^T \mathbf{A} \mathbf{R}_r + \mathbf{S}_r \mathbf{A} \mathbf{R}_r^T \right). \quad (12)$$

Referring to (9), owing to the convexity with respect to  $\mathcal{R}$  and the orthogonality of  $\mathbf{A}$  each slice of  $\mathcal{R}$  can be updated globally [5] as

$$\mathbf{R}_r \leftarrow \mathbf{A}^T \mathbf{S}_r \mathbf{A} \quad \forall r \in [1, 2, \dots, m]. \quad (13)$$

Algorithm 1 summarizes the steps for DEDICOM with projected gradient descent. In experiments with the same random initial conditions and stopping criteria, we found that HO-DEDICOM converged faster than the Approximate ALS algorithm (3.26% average speedup) but retains almost the same reconstruction error rate (HO-DEDICOM performs 0.25% better on average in terms of the fitting error). Figure 3 shows runtime and reconstruction error comparisons between HO-DEDICOM and Approximate ALS for asymmetric similarity

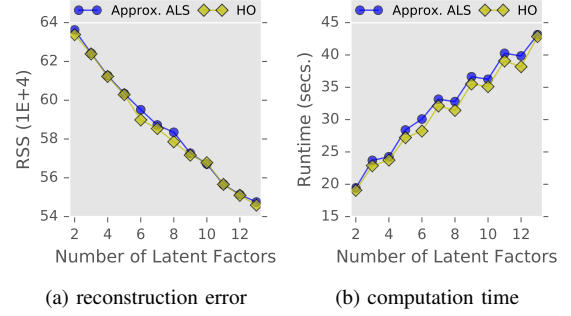


Fig. 3: Evaluation of our proposed algorithm with respect to computation time and reconstruction error from 3126 asymmetric similarity matrices extracted from the waypoint graphs with 200 nodes. While our algorithm yields the same accuracy, it is faster.

matrices extracted from waypoint graphs with 200 nodes covering movement information of 3126 players.

Due to the structure of the DEDICOM approximation, constraining the affinity matrices to be non-negative helps us to interpret both the loadings in  $\mathbf{A}$  and  $\mathcal{R}$  better [5]. This becomes especially useful when we are dealing with data sets with only non-negative values for which the resulting affinity matrices will be equivalent to their compressed versions. When non-negative affinities are required, we transform the update of slices of  $\mathcal{R}$  to a non-negative least square problem as done in [5]. That is, at each ALS step, instead of finding the optimal affinities as done in (13), we find optimal non-negative slices of  $\mathcal{R}$  by solving for

$$E''(\mathbf{R}_r) = \left\| \text{vec}(\mathbf{S}_r) - \left( \mathbf{A} \otimes \mathbf{A} \right) \text{vec}(\mathbf{R}_r) \right\|^2 \quad (14)$$

such that  $r_{ijr} \geq 0 \quad \forall r \in [1, 2, \dots, m] \wedge i, j \in [1, 2, \dots, k]$ . With this latest derivation we conclude the theoretical background about DEDICOM and turn our attention to another tensor factorization model that only deals with symmetric similarity matrices.

#### IV. INDSCAL

To compare our DEDICOM based feature learning framework and emphasize the importance of finding asymmetric affinity matrices for behavior prediction, we briefly study a tensor based multidimensional scaling model called Individual Differences Scaling (INDSCAL) [18]. The model was proposed in [18] as a special (symmetric) case of Canonical Decomposition in which tensors are decomposed into combinations of rank-1 tensors. Formally, given a tensor  $\mathcal{L}$  containing  $m \times n \times n$  similarity matrices in the same domain, INDSCAL partitions each slice as

$$\mathbf{L}_r \approx \mathbf{G} \mathbf{U}_r \mathbf{G}^T \quad \forall r \in [1, 2, \dots, m] \quad (15)$$

where  $\mathbf{G} \in \mathbb{R}^{n \times k}$  is the basis matrix representing importance of the ties of each entity and  $\mathbf{U}_r \in \mathbb{R}^{k \times k}$  is the diagonal salience matrix, which is the  $r$ th slice of  $\mathcal{U} \in \mathbb{R}^{k \times k \times m}$ ,

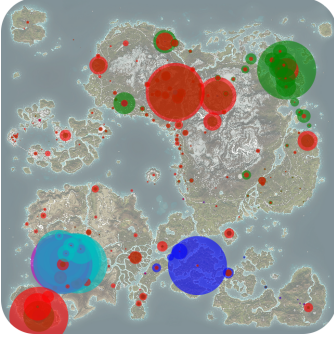


Fig. 4: Color coded bubble diagram of the loadings denoting the sectors that are automatically captured by DEDICOM analysis of the movements of 3126 players. The most important sectors align with the central points in the game play, i.e. the strongholds.

to weight the sum of the rank-1 matrices resulting from the outer product of the columns of the basis matrix. It is important to note that, INDSCAL is a constrained version of the relaxed DEDICOM introduced in (3) to decompose only symmetric similarity matrices. Similar to relaxed DEDICOM, fitting INDSCAL can be cast as a norm minimization problem

$$E'''(\mathbf{G}, \mathbf{U}_1, \mathbf{U}_2, \dots, \mathbf{U}_m) = \sum_{r=1}^m \left\| \mathbf{L}_r - \mathbf{G} \mathbf{U}_r \mathbf{G}^T \right\|^2. \quad (16)$$

A common way to find INDSCAL decompositions is to treat the matrix  $\mathbf{G}$  on the left- and right-hand side in (16) as separate matrices (as we considered for approximating DEDICOM's solution above) and to optimize them independently and at the end these two are expected to be equal. Instead of considering an indirect solution, a direct fitting algorithm for INDSCAL is proposed in [19].

Ten Berge et al. [19] find optimal INDSCAL factors by writing (16) in terms of  $l$ th column of  $\mathbf{G}$  and  $l$ th diagonal elements of slices of  $\mathbf{U}$ . This can be written as the  $l$ th column of  $\mathbf{G}$  as

$$E'''(\mathbf{g}_l, u_{1l}, \dots, u_{ml}) = \sum_{r=1}^m \left\| \mathbf{L}_{rl} - \mathbf{g}_l u_{rl} \mathbf{g}_l^T \right\|^2. \quad (17)$$

where  $\mathbf{L}_{rl} = \mathbf{L}_r - \sum_{i \neq l}^k \mathbf{g}_i u_{ri} \mathbf{g}_i^T$ . Considering an ALS update for unit length constrained  $\mathbf{g}_l$ , if we write (17) in terms of traces and separate the terms that only relate to  $\mathbf{g}_l$ , minimizing (17) keeping the saliences fixed amounts to maximize

$$E'''(\mathbf{g}_l) = \sum_{r=1}^m \mathbf{g}_l (u_{rl} \mathbf{L}_{rl}) \mathbf{g}_l^T = \mathbf{g}_l \left( \sum_{r=1}^m u_{rl} \mathbf{L}_{rl} \right) \mathbf{g}_l^T. \quad (18)$$

Note that the latest derivation in (18) shows that finding a particular optimal basis vectors for INDSCAL boils down to finding solution for quadratic form which is the eigen-vector corresponding to the largest eigen-value of  $\left( \sum_{r=1}^m u_{rl} \mathbf{L}_{rl} \right)$  as

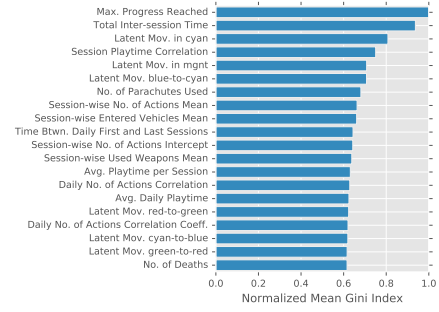


Fig. 5: Normalized gini index values for feature importance extracted from an ensemble of Random Forests that are trained to predict retention. The results indicate that maximum progress reached so far, time dependencies and spati-temporal activities in particular areas of the game map are crucial indicators of future game-play.

the solution. Updating the corresponding saliences of the unit length  $\mathbf{g}_l$  is defined as  $u_{rl} \leftarrow \mathbf{g}_l \mathbf{L}_{rl} \mathbf{g}_l^T \forall r \in [1, 2, \dots, m]$ . Repeating these steps for all of the basis vectors and their corresponding saliences for  $l = [1, 2, \dots, k]$  will yield the optimal INDSCAL factors. For more details about INDSCAL and algorithms for finding its appropriate factors we refer the reader to [18], [19].

## V. RESULTS

Here we describe the empirical results obtained using the above representation learning framework. First, details of how the framework is used to extract features are presented and then we discuss a case study showing the behavior of DEDICOM for comparative player analysis and feature evaluation. Finally, we report cross validation performance comparisons of the different algorithms.

### A. Settings

We consider four cases: a) prediction with purely behavioral features as in [1], [3], [6], b) with purely behavioral features and saliences learned by fitting INDSCAL to the symmetrized similarities between the waypoints and c) + d) constrained and unconstrained affinities obtained by fitting DEDICOM to the asymmetric similarities between the waypoints. As the main aim of the proposed framework is to be used in an agile development cycle, we rely on previous player experiences and build every model by *only* using a training set and evaluate the predictive power of our models using an independent validation set. It is important to note that, when obtaining the saliences  $\mathbf{U}_{test}$  and the affinities  $\mathbf{R}_{test}$  of the test players we use the basis matrices  $\mathbf{G}_{train}$  and  $\mathbf{A}_{train}$  that we obtain by respectively fitting INDSCAL and DEDICOM on the similarity matrices of the players in the training set. We measure the performance of the prediction by observing the accuracy of predicting the returners and the non returners (denoted by **Recall** and **Acc** resp.), the **Precision** of predicting the returner and the geometric and the harmonic

TABLE II: Retention prediction results with cross validation

Algo.	Representation	Acc.	Recall	Precision	G-Mean	F-Score
KNN	Pure	0.68	0.61	0.65	0.65	0.63
	Pure & HO	0.72	0.62	0.70	0.67	0.66
	Pure & SNN-HO	0.76	0.67	0.75	0.71	0.72
	Pure & INDSCAL	0.69	0.63	0.67	0.66	0.65
RF	Pure	0.69	0.64	0.72	0.66	0.68
	Pure & HO	0.70	0.67	0.72	0.68	0.70
	Pure & SNN-HO	<b>0.79</b>	<b>0.69</b>	<b>0.75</b>	<b>0.74</b>	<b>0.73</b>
	Pure & INDSCAL	0.75	0.61	0.77	0.67	0.67
GBC	Pure	0.70	0.62	0.68	0.67	0.66
	Pure & HO	0.73	0.68	0.72	0.71	0.71
	Pure & SNN-HO	0.72	0.68	0.72	0.70	0.70
	Pure & INDSCAL	0.72	0.66	0.70	0.69	0.68
ADA	Pure	0.70	0.65	0.72	0.68	0.69
	Pure & HO	0.72	0.67	0.70	0.69	0.70
	Pure & SNN-HO	<b>0.78</b>	<b>0.69</b>	<b>0.75</b>	<b>0.72</b>	<b>0.73</b>
	Pure & INDSCAL	0.70	0.66	0.71	0.68	0.69
Voting	Pure	0.70	0.67	0.70	0.68	0.68
	Pure & HO	0.72	0.69	0.70	0.71	0.71
	Pure & SNN-HO	<b>0.75</b>	<b>0.71</b>	<b>0.72</b>	<b>0.73</b>	<b>0.72</b>
	Pure & INDSCAL	0.70	0.67	0.70	0.69	0.69
Ensemble	Pure	0.69	0.71	0.69	0.70	0.70
	Pure & HO	<b>0.71</b>	<b>0.75</b>	<b>0.70</b>	<b>0.74</b>	<b>0.73</b>
	Pure & SNN-HO	<b>0.75</b>	<b>0.81</b>	<b>0.75</b>	<b>0.78</b>	<b>0.77</b>
	Pure & INDSCAL	0.72	0.69	0.73	0.71	0.71

mean of Recall and Precision (denoted by **G-Mean** and **F-Score** respectively), see [6] for more details about the use of the composite classification measures for classification tasks for games.

### B. Dissecting Our Framework

As an example of how our framework learns and represents spatio-temporal information, we set aside one eighth of our JC2 player data for testing and use the remainder for training. We extract waypoints and create a waypoint graph containing information about the asymmetric movements between waypoints which we denote as  $S_{train}$ . For ease of interpretation and visualization of the results, we fit SNN DEDICOM on  $S_{train}$  with  $k = 5$  to obtain the global loading matrix  $A_{train}$  and the tensor of player affinities in  $\mathcal{R}_{train}$ . The loadings in  $A_{train}$  represent how much particular waypoints contribute to latent movements of all of the analyzed players. We present the loadings as a color coded bubble diagram in Fig. 4 where each color represents a particular latent role which at the end forms a sector in the map and the size of each point represents how much it contributes to the sector. Analyzing the loadings on the map we notice that highest loadings correspond to some of the faction strongholds in JC2. Stronghold take-over missions are the endpoints in relation to taking control of a district of Panau, and form some of the most complex and time-consuming missions in the game. It is therefore reasonable to expect JC2 players to spend substantial time around these sites.

To see how the affinities in  $\mathcal{R}_{train}$  are structured, we chose the most extreme affinities that are furthest away from each other using  $k$ -maxoid clustering [20]. Fig. 6 displays the resulting *extreme* players in terms of their affinities and shows how diverse the movement behaviors can be. The player in Fig. 6a was active in most of the parts of the map whereas the player in Fig. 6b was active mostly in the green sector. Moreover, activities of the player in Fig. 6c are centered around the blue sector whereas the players in Figures 6d, 6e and 6f focused on the lower left part of the map. As a

final step, we trained an ensemble of *only* Random Forests (RF) [21] (a more extensive analysis with more algorithms is provided below) with the pure behavioral data and the spatio-temporal representations learned with SNN-DEDICOM with  $k = 5$  where we obtained G-mean and F-score values of 0.77 and 0.76 respectively for the players in the test set. Similar to the analysis in [6], we use the embedded feature ranking capability of Random Forests to investigate how much spatio-temporal features contribute to retention prediction. We present the 20 features with the highest importance values in Fig. 5 and observe that the most important indicator of player retention is the maximum progress reached within the observation time window and, similar to the F2P mobile and social games [1], inter-session time plays an important role for determining if the player will remain in the game or not. Other important indicators of retention are the latent movements in the cyan and magenta sector as well as the ones from blue to cyan (note again the relationship with stronghold missions). It is important to note that the magenta and the cyan sectors form an important milestone in the game as they are in a desert area where the player has to move much and they also contain missions that appear much later than missions in the red and the green sectors. Going down in the list, we see that temporal telemetry features appear to be important as well.

### C. Comparative Retention Prediction

Having seen that spatio-temporal features play an important role when it comes to predicting retention, we next analyze what happens if we do not incorporate them into the prediction process and how affinities and the saliences obtained from HO-DEDICOM and INSCAL perform against SNN-HO DEDICOM. We again partition our data into train and test sets to perform cross validation and fit HO- and SNN-DEDICOM on  $S_{train}$  and INDSCAL on the corresponding symmetrized version  $S'_{train} = (S_{train} + S_{train}^t)/2$  where  $S_{train}^t$  denotes the tensor that contains the transpose of the slices of  $S_{train}$ . In this case  $S'_{train}$  denotes the pairwise similarities of each node visited by the player regardless of its direction. Having fitted the models with latent factors  $k \in [2, \dots, 13]$ , we compare how well they perform when predicting retention using various classifiers:  $k$ -Nearest Neighbors (KNN), Random Forest, Gradient Boosting Classifier (GBC), and Ada Boost (ADA). In addition to using single classification models, we used a community based voting classifier that classifies given players based on the results of the 4 algorithms and (similar to the above analysis) we also used an ensemble of these four classifiers together with Linear and Quadratic Discriminant Analysis in which the clustering of the players done by Neural Gas algorithm. Tbl. II summarizes our retention prediction results categorized by the data representation and classifier used. We observe that adding spatio-temporal features always increases the overall prediction performance with respect to the composite metrics indicating the importance of incorporating such features for activity prediction in games. Additionally, we note that the non-negative features extracted from SNN-DEDICOM yield

	mgnt	blue	cyan	green	red		mgnt	blue	cyan	green	red		mgnt	blue	cyan	green	red
mgnt	86.421	0.003	7.309	0.234	7.274	mgnt	0.001	0.000	0.001	0.373	0.021	mgnt	9.278	0.031	2.715	0.030	1.219
blue	0.003	48.025	0.051	0.015	0.172	blue	0.000	0.000	0.000	0.015	0.001	blue	0.031	5129.513	0.000	0.000	0.000
cyan	7.203	0.051	229.170	0.229	2.257	cyan	0.001	0.000	0.002	0.438	0.014	cyan	2.719	0.000	63.756	0.139	0.266
green	0.234	0.025	0.230	408.387	0.000	green	0.373	0.015	0.438	291.414	5.770	green	0.000	0.000	0.000	49.029	0.000
red	7.281	0.174	2.271	0.000	1372.764	red	0.021	0.001	0.014	5.772	1.393	red	2.699	0.000	0.170	0.000	224.154

(a) A “balanced” player

	mgnt	blue	cyan	green	red		mgnt	blue	cyan	green	red		mgnt	blue	cyan	green	red
mgnt	2447.254	0.000	15.212	0.000	0.000	mgnt	760.357	0.024	87.055	0.000	0.000	mgnt	1201.763	0.000	0.000	0.000	0.000
blue	0.000	0.000	0.705	0.000	0.000	blue	0.008	0.000	0.349	0.000	0.000	blue	0.000	0.000	0.060	0.000	0.000
cyan	0.000	0.706	2184.270	0.000	0.000	cyan	35.252	0.350	1083.662	0.000	0.000	cyan	0.000	0.060	185.911	0.000	0.000
green	0.000	0.000	0.000	0.007	0.021	green	0.000	0.000	0.000	0.003	0.007	green	0.000	0.000	0.000	0.001	0.008
red	0.000	0.000	0.000	0.021	0.126	red	0.000	0.000	0.000	0.008	0.040	red	0.000	0.000	0.000	0.008	0.042

(d) Active in lower part of the map

	mgnt	blue	cyan	green	red		mgnt	blue	cyan	green	red		mgnt	blue	cyan	green	red
mgnt	0.001	0.000	0.001	0.373	0.021	mgnt	0.001	0.000	0.001	0.373	0.021	mgnt	9.278	0.031	2.715	0.030	1.219
blue	0.000	0.000	0.000	0.015	0.001	blue	0.000	0.000	0.000	0.015	0.001	blue	0.031	5129.513	0.000	0.000	0.000
cyan	0.001	0.000	0.002	0.438	0.014	cyan	0.001	0.000	0.002	0.438	0.014	cyan	2.719	0.000	63.756	0.139	0.266
green	0.373	0.015	0.438	291.414	5.770	green	0.373	0.015	0.438	291.414	5.770	green	0.000	0.000	0.000	49.029	0.000
red	0.021	0.001	0.014	5.772	1.393	red	0.021	0.001	0.014	5.772	1.393	red	2.699	0.000	0.170	0.000	224.154

(b) Active in green sector

	mgnt	blue	cyan	green	red		mgnt	blue	cyan	green	red		mgnt	blue	cyan	green	red
mgnt	0.001	0.000	0.001	0.373	0.021	mgnt	0.001	0.000	0.001	0.373	0.021	mgnt	9.278	0.031	2.715	0.030	1.219
blue	0.000	0.000	0.000	0.015	0.001	blue	0.000	0.000	0.000	0.015	0.001	blue	0.031	5129.513	0.000	0.000	0.000
cyan	0.001	0.000	0.002	0.438	0.014	cyan	0.001	0.000	0.002	0.438	0.014	cyan	2.719	0.000	63.756	0.139	0.266
green	0.373	0.015	0.438	291.414	5.770	green	0.373	0.015	0.438	291.414	5.770	green	0.000	0.000	0.000	49.029	0.000
red	0.021	0.001	0.014	5.772	1.393	red	0.021	0.001	0.014	5.772	1.393	red	2.699	0.000	0.170	0.000	224.154

(c) Active in the blue sector

Fig. 6: Prototypical affinity matrices as an outcome of  $k$ -maxoid analysis of the vectorized slices of the resulting  $\mathcal{R}_{train}$ . The results show the diversity in types of movements players performed between the sectors found by DEDICOM. See text for more details. Best seen in color.

the best performance which is associated to regularization (when compared to the version with unconstrained affinities) and the fact that the transition count data we are factorizing is indeed non-negative. We observe feature importance ratios similar to the analysis above. For an ensemble of random forests, the hit ratio distribution for the occurrences of features in the top 25 list is 100%, 83% and 16% respectively for SNN-HO DEDICOM, HO DEDICOM and INDSCAL. In summary, using asymmetric non-negative affinities as spatio-temporal features through DEDICOM partitioning not only provides interpretable results but also increases the performance of retention prediction.

## VI. CONCLUSION AND FUTURE WORK

Understanding player interactions and forecasting future behavior are two major tasks for future resource management in AAA game analytics. In this paper, the problem of predicting player retention in these games has been approached through a tensor factorization-based learning framework that integrates spatio-temporal behavior additional to providing descriptive insights. The notion of three way relaxed DEDICOM has been introduced, proposing this as a fast algorithm for retention prediction. Our results indicate that spatio-temporal features are important proxies for user engagement in Open-World Games in a similar capacity to the game-specific and temporal features known from F2P prediction models. Our future work involves using the proposed framework to model player interactions in the context of other AAA and F2P games that allow the player to move freely in the game world.

## ACKNOWLEDGMENT

The authors would like to thank Square Enix for access to the telemetry data from Just Cause 2.

## REFERENCES

- [1] F. Hadji, R. Sifa, A. Drachen, C. Thureau, K. Kersting, and C. Bauckhage, “Predicting Player Churn in the Wild,” in *Proc. of IEEE CIG*, 2014.
- [2] El-Nasr, M. S. and Drachen, A. and Canossa, A., *Game Analytics: Maximizing the Value of Player Data*. Springer, 2013.
- [3] J. Runge, P. Gao, F. Garcin, and B. Faltings, “Churn Prediction for High-value Players in Casual Social Games,” in *Proc. of IEEE CIG*, 2014.
- [4] C. Bauckhage, R. Sifa, A. Drachen, C. Thureau, and F. Hadji, “Beyond Heatmaps: Spatio-Temporal Clustering using Behavior-Based Partitioning of Game Levels,” in *Proc. of IEEE CIG*, 2014.
- [5] R. Sifa, C. Ojeda, and C. Bauckhage, “User Churn Migration Analysis with DEDICOM,” in *Proc. of ACM RecSys*, 2015.
- [6] R. Sifa, F. Hadji, J. Runge, A. Drachen, K. Kersting, and C. Bauckhage, “Predicting Purchase Decisions in Mobile Free-to-Play Games,” in *Proc. of AAAI AIIDE*, 2015.
- [7] H. Xie, S. Devlin, D. Kudenko, and P. Cowling, “Predicting Player Disengagement and First Purchase with Event-frequency Based Data Representation,” in *Proc. of CIG*, 2015.
- [8] R. Thawonmas, K. Yoshida, J.-K. Lou, and K.-T. Chen, “Analysis of revisitations in online games,” *Entertainment Computing*, vol. 2, no. 4, pp. 215–221, 2011.
- [9] B. G. Weber, M. John, M. Mateas, and A. Jhala, “Modeling Player Retention in Madden NFL 11,” in *Proc. of IAAI*, 2011.
- [10] A. Drachen, R. Sifa, C. Bauckhage, and C. Thureau, “Guns, Swords and Data: Clustering of Player Behavior in Computer Games in the Wild,” in *Proc. of IEEE CIG*, 2012.
- [11] R. Sifa and C. Bauckhage, “Archetypal Motion: Supervised Game Behavior Learning with Archetypal Analysis,” in *Proc. of CIG*, 2013.
- [12] T. Martinetz and K. Schulten, “A “neural-gas” network learns topologies,” *Artificial Neural Networks*, pp. 397–402, 1991.
- [13] R. A. Harshman, “Models for Analysis of Asymmetrical Relationships among N Objects or Stimuli,” in *Proc. Joint Meeting of the Psychometric Society and the Society for Mathematical Psychology*, 1978.
- [14] M. Nickel, V. Tresp, and H. Krieger, “A Three-way Model for Collective Learning on Multi-relational Data,” in *Proc. of ICML*, 2011.
- [15] B. Bader, R. Harshman, and T. Kolda, “Temporal Analysis of Semantic Graphs using ASALSAN,” in *Proc. of IEEE ICDM*, 2007.
- [16] P. A. Chew, B. W. Bader, and A. Rozovskaya, “Using DEDICOM for Completely Unsupervised Part-Of-Speech Tagging,” in *Proc. Workshop on Unsupervised and Minimally Supervised Learning of Lexical Semantics*, 2009.
- [17] H. A. Kiers, “DESICOM: Decomposition of Asymmetric Relationships Data into Simple Components,” *Behaviormetrika*, vol. 24, no. 2, pp. 203–217, 1997.
- [18] J. D. Carroll and J.-J. Chang, “Analysis of Individual Differences in Multidimensional Scaling via An N-way Generalization of Eckart-Young Decomposition,” *Psychometrika*, vol. 35, no. 3, pp. 283–319, 1970.
- [19] J. M. ten Berge, H. A. Kiers, and W. P. Krijnen, “Computational Solutions for the Problem of Negative Saliences and Nonsymmetry in INDSCAL,” *Journal of classification*, vol. 10, no. 1, pp. 115–124, 1993.
- [20] C. Bauckhage and R. Sifa, “k-Maxoids Clustering,” in *Proc. of KDML-LWA*, 2015.
- [21] L. Breiman, “Random forests,” *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

# Intrinsically Motivated General Companion NPCs via Coupled Empowerment Maximisation

Christian Guckelsberger  
Computational Creativity Group  
Goldsmiths, University of London  
London, United Kingdom  
Email: c.guckelsberger@gold.ac.uk

Christoph Salge  
Adaptive Systems Research Group  
University of Hertfordshire  
Hatfield, United Kingdom  
Email: c.salge@herts.ac.uk

Simon Colton  
The MetaMakers Institute  
Falmouth University  
Falmouth, United Kingdom  
Email: s.colton@gold.ac.uk

**Abstract**—Non-player characters (NPCs) in games are traditionally hard-coded or dependent on pre-specified goals, and consequently struggle to behave sensibly in ever-changing and possibly unpredictable game worlds. To make them fit for new developments in procedural content generation, we introduce the principle of *Coupled Empowerment Maximisation* as an intrinsic motivation for game NPCs. We focus on the development of a general game companion, designed to support the player in achieving their goals. We evaluate our approach against three intuitive and abstract companion duties. We develop dedicated scenarios for each duty in a dungeon-crawler game testbed, and provide qualitative evidence that the emergent NPC behaviour fulfils these duties. We argue that this generic approach can speed up NPC AI development, improve automatic game evolution and introduce NPCs to full game-generation systems.

## I. INTRODUCTION

*Dogmeat* from *Fallout* or *Ellie* from *The Last of Us* or the pet from *Nethack* – memorable companions are an important part of our gaming experience. But companions can also be a great source of annoyance, especially when their behaviour fails miserably [1]. The vast majority of companions are hard-coded by means of e.g. finite state machines or behaviour trees, and consequently struggle to produce believable or even plausible behaviour in unforeseen contexts ([2], [3]). More advanced companions can adapt their behaviour by means of planning, or by learning a policy via neural networks or traditional reinforcement learning. Nevertheless, they require intense training or pre-specified rewards, which again renders them inflexible especially in sandbox games where players with a large choice of options can change a dynamic world. In the future, the demands on non-player character (NPC) AI in general are likely to increase further [4]. This is particularly emphasised by progress in procedural content generation, which not only focusses on game elements such as levels and game mechanics ([4], [5]), but ultimately aims at generating entire games [6]. How can NPC AI deal with these ever-changing and potentially unpredictable game worlds?

One answer [2] is to drive the NPCs’ behaviour by means of intrinsic motivation, such as artificial curiosity [7] or learning progress [8]. Instead of relying on pre-defined goals or behaviours which might become meaningless when the game changes, intrinsically motivated agents perform “an activity for its inherent satisfactions rather than for some separable consequence” [9]. Models of intrinsic motivation ground behaviour

in the agent’s sensorimotor relationship with the world [8], so changes to the world or the agent’s embodiment are reflected in potentially new behaviour. A curious mouse and a curious bird would consequently behave differently, moderated by their embodiment and environment. In this paper, we will work with the intrinsic motivation of *empowerment* [10], a measure of how much an agent is in control of the world it can perceive. We have previously argued [11] that empowerment reflects an agent’s drive to maintain its own precarious existence, and allows them to adapt to changes in their embodiment and environment. But while empowerment might be very useful to produce an intrinsically motivated *general NPC*, we have to look specifically into how to turn it into a good *companion*.

Players seem to expect a companion to behave differently than a general NPC. For instance, in a qualitative study on companion behaviour, a player said “I dislike that [the companion] prioritises getting to the exit herself over helping [me] first” [1], stressing the delicate balance between support and independence. McGee and Abraham [12] argue that the NPC must account for the player’s goals as part of coordinated decision-making, possibly incorporating uncertainty. To guide our approach, we identify the following three companion duties, which should generalise across a range of game genres:

- 1) *Player Integrity*: Ensure that the player can continue playing the game. Act against any limiting force.
- 2) *Support*: Support and do not hinder the player in achieving their goals. Maintain *operational proximity*, i.e. act towards states where you can support the player.
- 3) *Companion Integrity*: Secure your own existence and ability to act in order to support the player long term.

We did not define any explicit, goal-specific companion duties which could constrain the NPC’s adaptivity. Instead, their goal directedness will arise from the interaction with the player.

We design an intrinsically motivated, general companion NPC based on the *Coupled Empowerment Maximisation* (CEM) principle. CEM establishes a general frame for support by relating an agent’s action selection policy not only to their own, but also to the empowerment of other agents. We provide an intuition and formalisation of the principle. We evaluate our approach qualitatively in a dungeon-crawler testbed, by means of observing whether the NPC fulfils its companion duties. We finish with a discussion, conclusion and future work.



## II. BACKGROUND

A wide body of research exists on companion AI and related notions. While the notion of “companions”, “sidekicks” and “assistants” usually refer to a unidirectional, supportive relationship towards the player (cf. [1]), research on “partners” and “team-mates” puts more emphasis on bidirectional collaboration and shared goals (cf. [12]). We are interested in support; Nevertheless, only few projects from the first category are relevant to us, as the majority specialises in specific game genres. We in contrast propose a notion for general companion-like behaviour in an arbitrary game.

We equip NPCs with the skill to coordinate their actions with the player in a supportive way. Most related work addresses this challenge by modelling the player’s goals explicitly. Fern and Tadepalli [13] represent the player as a noisy utility maximisation agent which is more likely to select actions that have a high utility of completing a given task. The player’s intentions are modelled by means of Markov-Decision-Processes (MDPs) which can capture uncertainty in human action-selection and in the environment. Nguyen et al. [14] extend this approach in their *Collaborative Action Planner with Intention Recognition (CAPIR)*, combining pre-computed MDP action policies and online Bayesian belief updates. They improve the performance of Fern’s and Tadepalli’s framework by decomposing tasks into subtasks. Their approach is evaluated in a maze game where a companion has to help the player kill ghosts. Macindoe, Kaelbling and Lozano-Pérez [15] build on *CAPIR* by steering NPC decision-making by the information it can gain about the player’s intentions. They evaluate their framework in a cooperative pursuit game. Our formalism is also concerned with planning, i.e. the simulation of experience, and is capable of accounting for uncertainty in the player’s behaviour and the environment. We do not model the player’s goals explicitly, but introduce them implicitly into the policy. While the utility function in these projects must be pre-defined, our approach employs intrinsic motivation to overcome this limitation.

If we constrain our scope to intrinsically motivated agents, the body of research becomes much smaller. The work by Merrick and Maher ([2], [16]) is most closely related, looking at how intrinsically motivated reinforcement learning [17] can support NPCs in learning complex tasks in a dynamic game world. They propose two models of motivation as reward signals for Q-learning: an agent’s interest in a new situation, given past experiences, and its competence based on the error in learning policy updates. Their qualitative studies in *Second Life* and a quantitative analysis of behavioural variety and complexity in dedicated RPG testbeds confirm that intrinsic motivation allows agents to adapt their behaviour in an unexpectedly changing environment. In contrast to our study, their NPCs act in solitude, and not in favour of other agents such as the player. In a related study, Forgette and Katchabaw [3] focus on the development of more believable NPC behaviour, by letting them choose actions in order to maintain basic desires such as hunger or social contact. Similar to Merrick

and Maher, their focus is on learning complex sequences of actions to satisfy their motives, while we are interested in establishing supportive, immediate reactions to the player’s actions. Forgette and Katchabaw’s desires can only loosely be considered intrinsic, as they rely on sensor semantics (cf. [8]). Unfortunately, this dependency makes the approach unsuitable in PCG or dynamically changing game worlds.

Empowerment as intrinsic motivation has so far only been employed for general game-playing, but not to steer the behaviour of companion or enemy NPCs. Anthony, Polani and Nehaniv [18] analysed empowerment maximisation to drive player behaviour in Sokoban and Pac-Man, and in the same course proposed several optimisation methods. Mohamed and Rezende [19] focus primarily on optimisation, with a likely application in general game playing.

## III. FORMAL MODEL

We propose the principle of coupled empowerment maximisation (CEM) as an abstract notion for companion-like behaviour in highly flexible and adaptive NPCs. We recently introduced CEM in a co-creativity context [20], and will expand it here to fit the requirements of companion NPCs. We first provide an intuition and definition of the underlying empowerment formalism and its variations. Equipped with this, we show how the different notions come together in CEM to drive the decision-making of NPCs.

### A. Empowerment

Empowerment [10] is defined between an agent’s actuators and sensors. In a deterministic environment, it quantifies the options available to an agent in terms of availability and visibility. In a stochastic setting, this generalises to its potential, perceived influence on the environment. Empowerment is measured in bits; It is zero when the agent has no control over its sensors, i.e. when all actions lead to the same or a random outcome, and it increases when different actions lead to separate perceivable outcomes. For simplicity, we focus on interactions which are discrete in time and space, but continuous implementations exist. An extensive survey of motivations, intuitions and past research can be found in [21].

At the centre of the empowerment definition is the interpretation of an agent’s embodiment as an information-theoretic communication channel between its actuator  $A$  and sensor  $S$ , where in  $\mathcal{A}$  and  $\mathcal{S}$  represent the possible actions and sensor states. The agent’s interaction with the world is usually described as a perception-action loop [22] as in Fig. 1. Modelled by means of a causal Bayesian network, the figure illustrates the turn-wise interaction of the player and a companion NPC, unrolled in time. Each agent is represented by their sensor and actuator, and the black, solid arrows imply causation between these random variables. The companion chooses an action based on its sensor input at time  $t$ , influencing the rest of the environment  $R$  and the player’s sensor in the next time step. The player’s sensor informs the player’s actions, which influence the environment and the companion’s sensor at  $t + 2$ . The latter in turn affects the companion’s actions,



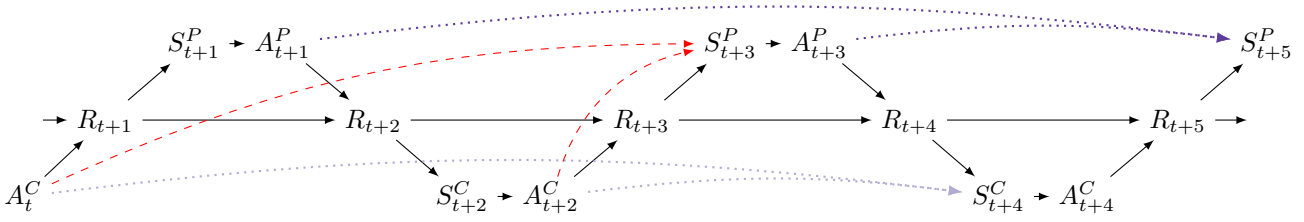


Fig. 1: Perception-action loop for player ( $S^P, A^P$ ) and companion ( $S^C, A^C$ ), illustrating three different types of 2-step empowerment. The agents interact in turnwise order, and are represented by their sensors and actuators. Purple, dotted lines (top) represent player empowerment. Violet, dotted lines (bottom) stand for companion empowerment, and red dashed lines for companion-player transfer empowerment.

and the cycle repeats itself. The environment is affected both by its preceding state and the character's actions. The causal probability distribution  $p(S_{t+2}^C | S_t^C, A_t^C)$  then represents the (potentially noisy) communication channel between the companion's actions *before*-, and its future sensor states *after* the player has performed an action.

Empowerment  $\mathfrak{E}_{s_t}$  is then defined as the maximum potential information flow that could possibly be induced by a suitable choice of actions, in a particular state  $s_t$ . This can be formalised as the channel capacity:

$$\begin{aligned} \mathfrak{E}_{s_t} &= \max_{p(a_t)} I(S_{t+2}; A_t) \\ &= \max_{p(a_t)} \sum_{A, S} p(s_{t+2} | s_t, a_t) p(a_t) \log \frac{p(s_{t+2} | s_t, a_t)}{\sum_A p(s_{t+2} | s_t, a_t) p(a_t)} \end{aligned}$$

Here,  $I(S_{t+2}; A_t)$  represents the mutual information between sensors and actuators. Empowerment as defined here is the maximum amount of information that the active agent can inject into the environment with its actions at  $t$ , and perceive again at  $t+2$ , after the other agent performed. *N-step empowerment* is a generalisation of this principle where not a set of single actions  $A$ , but a set of action sequences  $A_t^s = (A_t; A_{t+2}; \dots; A_{t+2(n-1)})$  and their impact on a future sensor state  $S_{t+2n}$  are evaluated. The parameter  $n$  specifies the agent's lookahead. For a detailed introduction to the general information-theoretic notions, consult [23].

Fig. 1 illustrates the three variants of 2-step empowerment used in this paper: *Companion-* ( $\mathfrak{E}^C$ , ..., bottom) and *player empowerment* ( $\mathfrak{E}^P$ , ..., top) correspond to the NPC's and player's perceived influence on their own future sensor state. *Companion-player transfer empowerment* ( $\mathfrak{E}^T$ , --) maps the companion's actions to the player's future sensor, quantifying the companion's influence on the player's perception.

Empowerment is *local*, i.e. the agent's knowledge of the dynamics  $p(S_{t+2} | S_t, A_t)$  is sufficient to calculate it. The information-theoretic grounding makes it domain-independent and universal, i.e. it can be applied to every agent-world interaction that can be modelled as a perception-action loop. It can thus be computed for arbitrary setups of what a NPC can see and do, and can cope with changes to a game's mechanics, affecting how the NPC can interact with the world and player.

### B. Coupled Empowerment Maximisation

Empowerment does not measure an agent's actual, but rather their potential influence on the environment. The em-

powerment maximisation hypothesis [21] suggests that an agent should, in the absence of any explicit goals, choose actions which likely lead to states with a higher influence on the environment, i.e. potentially more options. *Coupled empowerment maximisation* (CEM) is an extension of this principle to the multi-agent case, and we use it to formalise companion-like behaviour in a very general and flexible way.

Fig. 2 illustrates the turn-wise interaction of player  $P$ , companion  $C$  and enemy  $E$ . Each interaction cycle is initiated by the player performing an action, which the companion will react to, followed by enemies. Each agent can affect the others either explicitly, or implicitly through their impact on the shared game world, which can be quantified by empowerment. Given the previous intuition, we suggest that increasing the empowerment of a goal-directed agent can be considered as supporting them in performing and achieving their tasks. We consequently hypothesise that equipping an NPC with an action selection policy which not only maximises their own- but also the player's empowerment leads to the emergence of companion-like behaviour. We specifically suggest the policy

$$\pi(s_t) = \arg \max_{a_t} \left( \alpha_C \cdot E[\mathfrak{E}_{s_{t+4}}^C]_{a_t} + \alpha_T \cdot E[\mathfrak{E}_{s_{t+4}}^T]_{a_t} + \alpha_P \cdot E[\mathfrak{E}_{s_{t+3}}^P]_{a_t} \right)$$

with parameters  $\alpha_C, \alpha_P, \alpha_T$  representing the influence of each empowerment type in the overall coupling. The CEM principle establishes a dependency between the NPC's decision-making and the player. Given a certain state, the policy returns the action which maximises the NPC's expected coupled empowerment, i.e. the combination of its own expected empowerment  $E[\mathfrak{E}^C]$ , the expected empowerment of the player  $E[\mathfrak{E}^P]$ , and the NPC's influence on the player's sensor state, i.e. expected companion-player transfer empowerment  $E[\mathfrak{E}^T]$ . The latter allows the companion to maintain *operational proximity*, i.e. to put the NPC in a position where it could potentially affect and thereby maximise the player's empowerment. We look at *expected* empowerment, because the companion must consider all possible ways the player could behave.

The calculation of coupled empowerment therefore involves several estimation steps, which are illustrated in Fig. 2. To select an action in  $t+1$ , the NPC has to calculate the expected coupled empowerment for each of its actions  $a_{t+1}$  separately. As a first step, we thus have to estimate which potential player states at  $t+3$  each of the companion's actions and the enemies' reactions might lead to. From there on, we have to anticipate how the player might react, resulting in potential companion states at  $t+4$ . This estimation stage is denoted

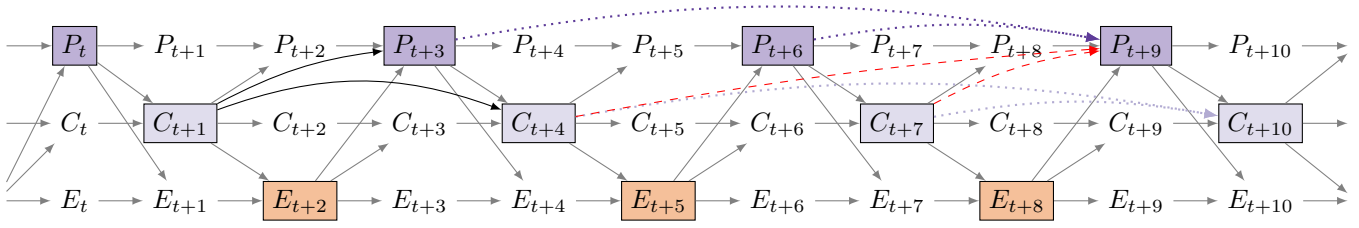


Fig. 2: Causal Bayesian network illustrating the turn-wise interaction of player  $P$ , companion  $C$  and enemies  $E$ . The solid grey lines denote mutual influence. The solid black lines denote the companion's future state estimation, preceding the calculation of 3 types of 2-step empowerment: player- (purple dotted, top), companion- (violet dotted, middle) and companion-player transfer empowerment (red dashed).

by black, solid arrows in Fig. 2. We then calculate player empowerment (...., top) in  $t+3$ , as well as companion- (...., middle) and companion-player transfer empowerment (- -) in  $t+4$ . This involves another  $n$  rounds of estimations, depending on the lookahead  $n$ . Expected coupled empowerment is finally calculated by weighting the individual empowerment values with the probability of the states the NPC's actions trigger.

Note that empowerment maximisation *per se* is not goal-oriented. Nevertheless, coupled empowerment introduces the player's empowerment into the companion's policy, and thus guides the companion by means of the player's goals.

#### IV. EVALUATION: GENERAL COMPANION-LIKE BEHAVIOUR

We claim that maximising coupled empowerment realises companion-like behaviour. We evaluate this claim qualitatively, by designing several dedicated scenarios to probe each of the companion duties outlined earlier. We describe the emergent behaviour, and highlight the contributions of the individual empowerment types coupled in the agent's policy, and how they blend together. Our goal is to create highly flexible and adaptive companions. Since explicit constraints decrease this flexibility, we start with the plain formalism, highlight drawbacks in the various scenarios, and propose modifications which maintain this flexibility as needed. Importantly, we do not tailor the formalism to each scenario; instead, we extend it successively to eventually generalise across all of them.

##### A. Testbed

The following experiments were set in a minimal *dungeon-crawler* game in which the player, supported by a companion NPC, has to navigate through rooms connected by corridors and defeat enemies in order to reach a goal state. Agents interact in turn-wise order, starting with the player, and followed by the companion and enemies. All agents have health points, and can either move one step in each direction, shoot, or idle. They can only hit other agents within a certain range in their view direction, which changes with movement. While the player's actions cannot be exactly predicted by the companion, the enemies act, for the sake of simplicity, deterministically: They always shoot at, or chase, the closest non-enemy.

We chose this game type for various reasons: It traditionally relies on procedural content generation and elements of chance, and therefore poses interesting challenges to a general NPC policy. Classic examples such as *Nethack*, but also more

recent variants such as *Hashtag Dungeon* illustrate how our minimal testbed can be extended to introduce new challenges to the formalism. Dungeon crawlers are traditionally discrete in time and space and thus simplify the computation and analysis of agent behaviour. The core mechanics are grounded in the behaviour of living beings, and thus connect with the biological origins of empowerment [11].

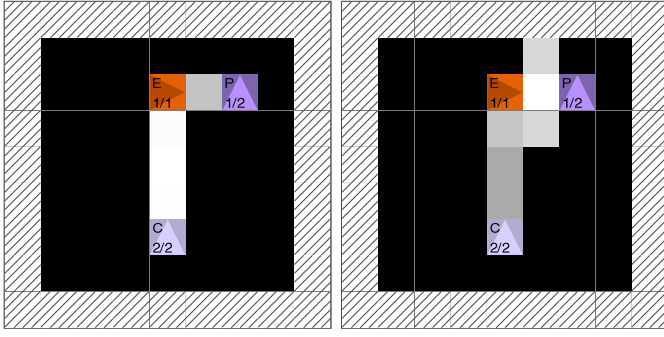
The player and companion sensors are local, non-overlapping and asymmetric. We model locality by only accounting for entities such as other characters in a maximum distance of two units around the agent. They are represented by an id and their relative position. Sensors do not overlap, i.e. they only comprise the agent's own absolute position, rotation, and health. They are asymmetric, in that the player sensor also comprises the game status (running, lost, won).

The non-overlapping sensor would make our NPC strictly egocentric, if it only maximised its own empowerment. In our simulations though, we weight the player's empowerment the most by  $\alpha_P = 0.5$ , the companion's own empowerment by  $\alpha_C = 0.2$ , and the companion-to-player transfer empowerment by  $\alpha_T = 0.3$ , in order to reflect the companion duty hierarchy. These values were determined by experimentation, and work across all scenarios. They can be varied to some extent, and we will provide details on their limits in each scenario. By default, we assume a lookahead of  $n = 2$ .

##### B. Duty 1: Ensure Player Integrity

A companion must protect the player, and prevent its death. Fig. 3 show a scenario in which the player is directly threatened: An enemy faces the player, ready to shoot. The companion in turn faces the enemy, and could therefore rescue the player. The figures illustrate the empowerment values relevant to the policy, by means of mapping them as greyscale values to different positions in the scene. Brighter hues indicate higher empowerment. The player, companion and enemies are represented by purple, violet and orange squares with letters "P", "C" and "E", respectively. The numbers on the bottom specify their current and maximum health.

In Fig. 3a, the player position was fixed and the companion moved around. The value at a particular location corresponds to the player's empowerment, if the companion was in that position and chose to shoot. According to the core formalism (cf. sec. III-A), empowerment would drop to zero if the player was killed. Consequently, it is highest if the companion either faces the enemy in close range, expressing the potential to



(a) Player empowerment, given that the companion chose to shoot in a certain position,  $n = 2$ . (b) Companion-player transfer empowerment for non-local sensor,  $n = 2$ .

Fig. 3: 1st scenario. The player is threatened to be killed by an enemy. The companion faces the enemy, and could rescue the player.

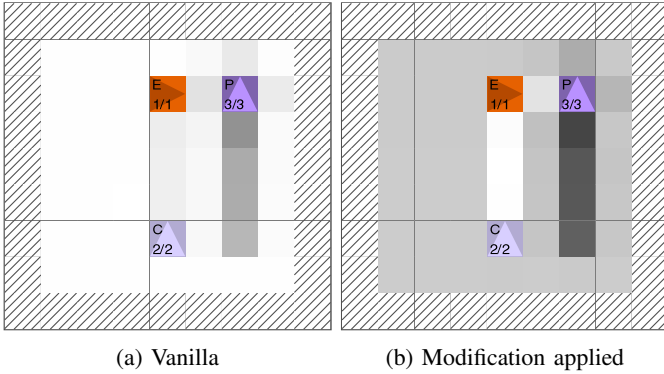


Fig. 4: 1st scenario. Player empowerment, given the companion shoots in a certain position, and player health  $> n$ ,  $n=2$ . Health-performance consistency (right) indicates clearly where to shoot.

shoot, or steps between the two to take the bullet. It is less preferable for the player to be faced by the companion, because the latter could turn its weapon against the ally. Fig. 3b illustrates the companion's role as bodyguard by means of companion-to-player transfer empowerment, i.e. its influence on the player's future sensor state. It shows that the companion could still save the player by stepping in from the side.

CEM makes the companion kill the enemy for any value of  $\alpha_C$  and  $\alpha_T$ , as long as  $\alpha_P > 0$ . If  $\alpha_P = 0$ , the companion does not have access to the player's sensor state. As their sensors do not overlap, the companion would not "care" about the player. Importantly, the companion would defend the player even if the enemies did not pose a threat to itself.

Unfortunately, the companion would only protect the player as long as its lookahead  $n$  is larger than the player's health. The reason can be found in an actual inconsistency present in many (video) games: In nature, a living being's health not only indicates its closeness to death, but also corresponds to a decline in the ability to interact successfully with the world [11]. In games, health or similar labels for fitness often only represent a mere warning, and affect the agent's performance irregularly or only when dropping to zero. A companion can thus only foresee the tragic consequences of the enemy's actions if it evaluates the environment dynamics far enough

ahead. This can be expensive to compute, but could also result in overcautious overall behaviour. We therefore suggest to make the relationship between a character's health and actual performance more consistent, by introducing noise into the agent's state transition probabilities:

$$p(S_{t+3}|s_t, a_t)^* = \begin{bmatrix} p(s_{t+3}^1|s_t, a_t) \\ p(s_{t+3}^2|s_t, a_t) \\ \vdots \\ p(s_{t+3}^D|s_t, a_t) \end{bmatrix} \odot \begin{bmatrix} \gamma \\ \gamma \\ \vdots \\ 1 - \gamma \end{bmatrix}, \gamma = \frac{h_t}{h_{max}}$$

Here,  $h_t$  and  $h_{max}$  stands for the agent's current and maximum health, as representative for some arbitrary fitness label. The state  $s_{t+3}^D$  resembles the agent's default follow-up state, e.g. the state resulting from idling. The more an agent's health decreases, the more likely it becomes that its actions will lead to the default state. Applied to all available actions, the modification will lead to a consistent, gradual decrease of an agent's empowerment with their health.

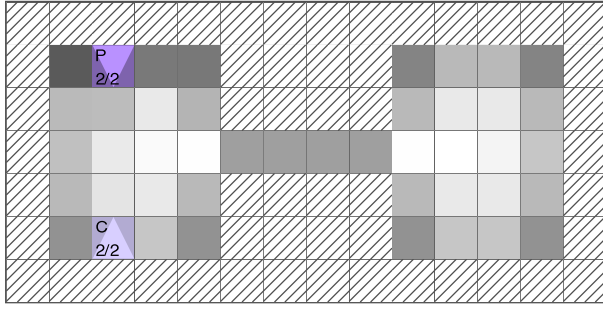
Fig. 4 illustrates the effect of this modification in the previous scenario. Here, *health-performance consistency* allows the companion to clearly differentiate between actions that contribute to the player's empowerment, despite a short lookahead  $n < h_t$ . The companion not only acts when the player faces death, but also protects the latter from being harmed. We will assume this modification by default in the following scenarios. The behaviour can be watched online<sup>1</sup>.

### C. Duty 2: Support the Player

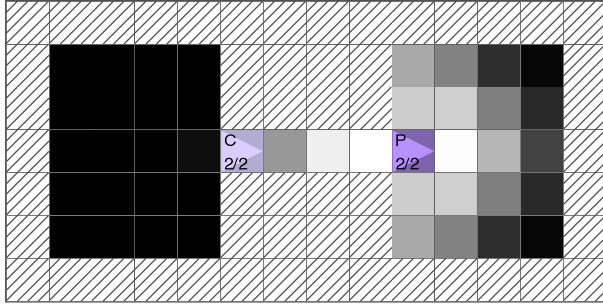
In order to support the player in achieving their goals, the companion should strive for states in which it can affect the player and its perceivable environment best. Depending on the NPC's action set, this *operational proximity* can be different from *spatial proximity*: We can imagine an NPC which can operate terminals, but has no capacity for melee attacks. Such a companion might support the player most by staying remote, where it could e.g. unlock doors or trap the player's enemies. In our second scenario in Fig. 5, spatial proximity is key and we expect the companion to stay close to the player and follow them from one room to the other.

However, this is not self-evident: The companion's empowerment (Fig. 5a) is particularly low at the room edges and corners, but also in the corridor. Here, the NPC's sequences of navigational actions collapse into very few follow-up states, as the agent can neither move north nor south. If the NPC's policy was only about maximising its own empowerment, it would move to the centre of the current room, and avoid the corridors. Nevertheless, adding companion-player transfer empowerment to the equation renders all states but the ones in which the player can be directly influenced less attractive (Fig. 5b). When coupled with the other empowerment types, it compensates for the barrier induced by the companion's empowerment (Fig. 5c). For the default setup and  $\alpha_T \geq 0.3$ , the companion consequently follows the player through the corridor and maintains spatial proximity. Note that this scenario

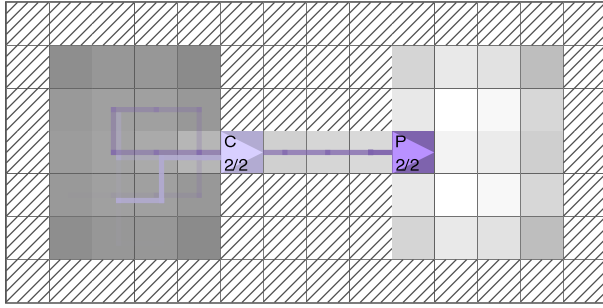
<sup>1</sup>Video on duty 1 (Ensure Player Integrity): [http://y2u.be/uh3J\\_ENh11M](http://y2u.be/uh3J_ENh11M)



(a) Companion empowerment,  $n=2$ .



(b) Companion-player transfer empowerment,  $n=2$ .



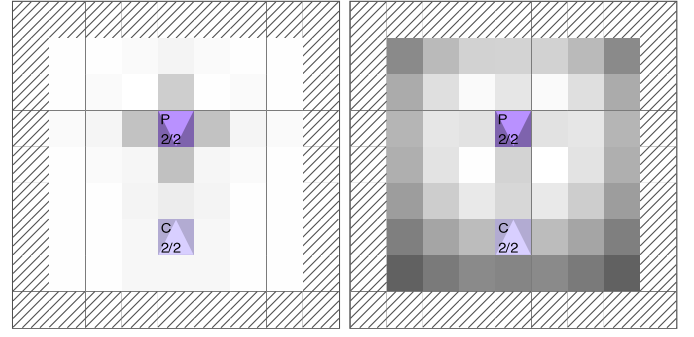
(c) Coupled empowerment with movement trace,  $n=2$ .

Fig. 5: 2nd scenario. The transfer empowerment in the coupling allows the companion to maintain operational proximity, and thus to follow the player through bottlenecks such as a narrow corridor.

only represents one example of an empowerment bottleneck, and that our formalism should generalise to other situations.

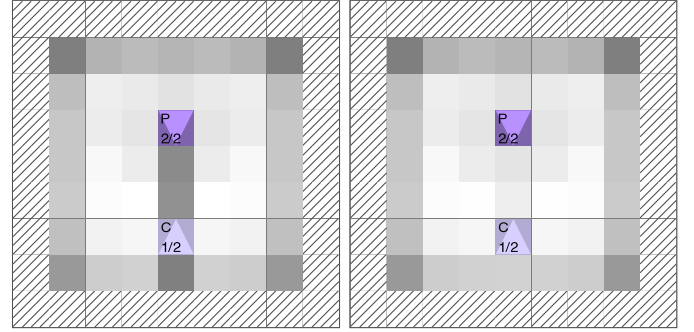
The notion of support implies not hindering the player from reaching their goals. In our third scenario, we check that the companion does not block the player's movement at any time, while maintaining spatial proximity. Fig. 6a shows the player's empowerment for different companion positions. The values are low around the player, because the companion would constrain its movement. The same applies to the companion's periphery in respect to the player. Left alone, they would add up and lead to generally repellent behaviour; Nevertheless, the transfer empowerment in the coupling (Fig. 6b) makes the companion maintain spatial proximity, while not blocking the player whenever possible. It will consequently prefer the corners around the player and either go ahead or follow.

Fig. 7a, showing companion empowerment for different companion positions, highlights a shortcoming in the initial formalism: Here, low values between companion and player illustrate that the player can occasionally be perceived as



(a) Player empowerment, given (b) Companion-player transfer empowerment,  $n=2$ .

Fig. 6: 3rd scenario. The companion is reflected as an impediment to movement in the player's empowerment. Once coupled, it makes the companion seek proximity off the player's axes.



(a) Health-Performance Consistency (b) Health-Performance Consistency and Minimal Synchronicity

Fig. 7: 3rd scenario. Without applying the trust function, the player will be reflected as thread in the companion's empowerment.

a threat. This is the case because the companion considers scenarios where the player's actions would harm it as feasible as any other, and weights them equally in the calculation of expected empowerment. As a consequence, the companion might e.g. "flee" from the player, preferring not to stay within their shooting range. We suggest that this behaviour is unnatural for the interaction of a supporting agent and one which benefits from this support and its continuity. In general terms, it appears necessary for companion-like behaviour to emerge that player and companion realise *trust*, in terms of not considering actions from their ally which threaten their existence significantly. We consequently applied a correction function to the action transition probabilities *after* the calculation of empowerment, in order to remove actions counteracting such trust from the expected empowerment calculation. Actions are included in the trusted set  $A_P^* \subseteq A_P$  as follows:

$$a \in A_P^* \Leftrightarrow \neg (p(s_{t+3}|s_{t+2}, a) > \tau \wedge E^C(s_{t+3}) = 0 \wedge E^C(s_{t+2}) > 0)$$

The function removes player actions which reduce the companion's empowerment to zero with probability  $\tau > 0$ . The companion therefore still considers player actions which are unlikely to be fatal, but might benefit the player significantly. Not only the player, but also the enemies could decrease the companion's empowerment to zero. In order to not confuse



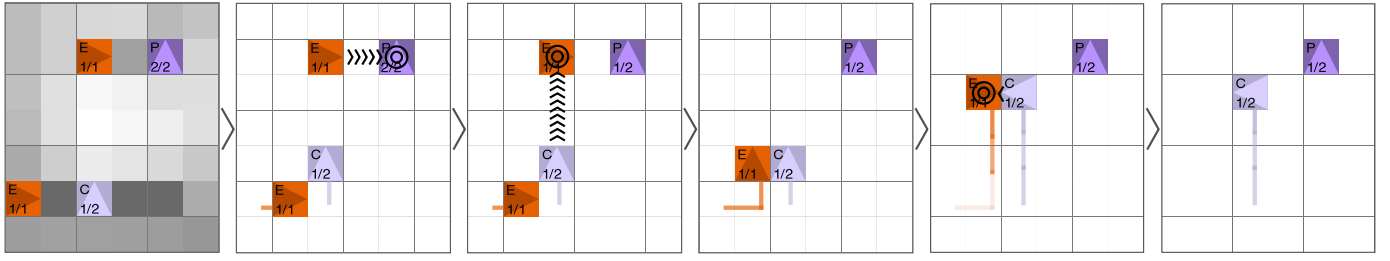


Fig. 8: 4th scenario: Companion and player are threatened simultaneously. Successive moves from left to right: the companion escapes its own death, rescues the player, and finally defends itself. Left: coupled empowerment, for  $n=2$ . Arrows indicate shooting.

both effects, we calculate and compare companion empowerment before and after the player performed. Applied to the previous scenario (Fig. 7b) the expected empowerment of remaining in the player's shooting range would increase, making this action more likely to be performed. It also only applies when agents are close to death, i.e. empowerment can actually be decreased to zero in a single step. We will assume the modification to be present in the following scenarios. Our observations were documented in a video available online<sup>2</sup>.

#### D. Duty 3: Ensure Own Integrity

We demonstrated earlier that the companion will protect the player from threats. Various earlier studies ([18], [24]) showed that empowerment maximisation makes agents death-averse, allowing the NPC to defend itself against threats. In our fifth scenario, we look at a more complex dilemma addressing both the duties to protect the player and itself: What happens if companion and player are threatened at the same time? More specifically, what happens if the player might only be harmed, compared to the situation where it might be killed if the companion did not react immediately?

Fig. 8 illustrates the outcome of the first case by means of a series of movements. The first image in the series shows the coupled empowerment in the initial situation. Here, the dark area between the companion and the enemy on the left renders the latter as a threat, while the white area towards the other enemy highlights the companion's potential to save the player from harm. The following images illustrate that the companion will first escape from the enemy on the left, while accepting that the player will be harmed. It will then kill the player's enemy before the latter kills the player. The remaining enemy and companion follow each other, until the companion eventually kills the enemy to save its own existence.

In the second case, the initial situation is the same, but the player's health is set to one. For the default parameter configuration and  $\alpha_P > 0.5$ , the companion will sacrifice itself to rescue the player from death, thus fulfilling the hierarchy of duties outlined earlier. A video with both scenarios is online<sup>3</sup>.

### V. DISCUSSION

Our experiments provide evidence that CEM enables the emergence of companion-like behaviour. We proposed to extend the formalism with a consistent relationship between

an agent's fitness indicator, e.g. health, and its performance. This modification represents a heuristic when the NPC does not have a sufficiently large lookahead  $n$ . It thus reduces computational complexity, but is not necessary for supportive behaviour to emerge. Our *trust* function represents a minimal case of coordination in anticipation, which complements the coordination in the agent's policy established by the CEM principle. In contrast to health-performance consistency, we suggest that coordination in anticipation is strictly necessary for supportive behaviour, and that generalising *trust* to both negative and positive effects on player and companion would result in more fine-tuned, supportive behaviour. We therefore propose to weight the likelihood of actions gradually according to  $p(a) \propto \Delta \mathcal{E}_s$  in future implementations. Coordination must be extended into the calculation of empowerment itself, in order to create more reliable biases for the policy. Learning the actual distribution of actions as part of player modelling would also contribute significantly to coordination, and speed up calculation by pruning the search tree.

We fixed the parameters of  $\alpha_P, \alpha_C, \alpha_T$  to allow for sensible behaviour across all scenarios. This exact configuration might not work in an arbitrary game; We therefore suggest two alternative solutions: Looking closely at the companion duties, we claim that ensuring its own and the player's integrity emerges from general player support. We could translate this hierarchy into the policy, and consequently choose actions primarily to maximise player empowerment, followed by transfer and companion empowerment. Alternatively or in addition, we propose to evolve the NPC's parameters by means of automated play-throughs with a general game-playing agent.

Our goal was to investigate the richness of behaviour induced by the CEM principle, and we consequently abstained from any unnecessary constraints. Employing this principle in an actual game might nevertheless require explicit constraints to meet two industry requirements: predictability and performance. Empowerment as an intrinsic motivation allows for maximum adaptivity and flexibility in NPCs, but as a consequence might trigger surprising behaviour. If predictability is a strict requirement, we can fix the behaviour emerging from CEM before deployment, or use the formalism as a mere intuition pump to assist designers. Alternatively, we can define illegal behaviours as constraints on top of the policy, if the designer favours surprisingness over adaptivity. Such explicit constraints can also help in decreasing computational complexity, by pruning the search tree. Empowerment can be

<sup>2</sup>Video on duty 2 (Support the Player): <http://y2u.be/6g9qoa5BdwU>

<sup>3</sup>Video on duty 3 (Ensure Own Integrity): <http://y2u.be/z3gZ0iGE0wg>

approximated and serve as tie-breaker to increase behavioural variety. Optimisation methods allow for larger lookaheads, and thus more behavioural complexity. Existing optimisations use Monte-Carlo sampling [21], the *information-bottleneck* method [18] and deep neural networks [19].

## VI. CONCLUSION & FUTURE WORK

We formalised and evaluated the principle of coupled empowerment maximisation (CEM) to design an intrinsically motivated NPC capable of companion-like behaviour. We started with the raw formalism, and successively added modifications which generalise across the scenarios. Our experiments show that CEM establishes a sufficiently general frame for companion-like behaviour by inducing the player's goal into the companion's policy, making it unnecessary to specify the NPC behaviour explicitly at design time. Given our experimental evidence and the universality of the formalism, we hypothesise that the principle generalises to a wide range of game scenarios and genres.

If this proves correct, the flexibility and adaptivity of CEM could make NPCs fit for the most recent challenges in the games industry and academic research. It could allow industry to save efforts and reduce costs of manually authoring NPC behaviour, especially in games with a strong focus on procedurally generated content. Even if a game relies strongly on scripting, our formalism can help in establishing a default mode of interaction with the player and other agents. In automatic game evolution and rapid prototyping, NPCs driven by CEM will allow us to stretch the parameter space and search regions where pre-defined NPCs would break. It could enhance research in computational game creativity, by increasing novelty and surprisingness in NPC behaviour, and serve as cornerstone in the automatic generation of complete games, which presently does not incorporate NPCs. Given these developments, we hope to see NPCs soon in general game playing competitions.

We plan to expand this research in three directions. First, we want to investigate how well the principle generalises by changing the mechanics and increasing the complexity of our testbed. We particularly plan to look at shared resources such as health packs or ammunition, additional actions which might complement with the player's skills, and more sophisticated mechanics such as traps or door openers. A second potential branch of future research concerns human player experience. We are interested in evaluating qualitatively how enjoyable it is to play with our companions, especially in respect to the player's perception of agency. We suggest that empowerment relates closely to agency [11], and that varying the policy parameters might affect the player's locus of control, and the perceived "character" of the NPCs. Finally, we want to investigate how well this principle can be reversed to generate enemy NPCs, demonstrating non-obvious ways to be antagonistic. All branches will require more work in reducing computational complexity and improving coordination.

## ACKNOWLEDGMENTS

The authors thank Jeremy Gow for helpful comments on an earlier draft. CG is funded by EPSRC grant [EP/L015846/1] (IGGI). CS received funding from the H2020-641321 socSMCs FET Proactive project. SC is funded by ERA Chair Project GRO (621403).

## REFERENCES

- [1] M. Cerny, "Sarah and Sally : Creating a Likeable and Competent AI Sidekick for a Videogame," in *Proc. AIIDE Workshop EXAG*, 2015.
- [2] K. Merrick, "Modeling Motivation for Adaptive Nonplayer Characters in Dynamic Computer Game Worlds," *CiE*, vol. 5, no. 4, p. 1, 2008.
- [3] J. Forgette and M. Katchabaw, "Enabling Motivated Believable Agents With Reinforcement Learning," *Games Media Entert.*, pp. 1–8, 2014.
- [4] G. Smith, "The Future of Procedural Content Generation in Games," in *Proc. AIIDE Workshop EXAG*, 2014.
- [5] A. Liapis, G. N. Yannakakis, and J. Togelius, "Computational Game Creativity," in *Proc. 5th Int. Conf. Computational Creativity*, 2014.
- [6] M. Cook, S. Colton, and J. Gow, "The ANGELINA Videogame Design System, Part I," *IEEE TCIAIG*, no. c, pp. 1–12, 2016.
- [7] J. Schmidhuber, "Developmental Robotics, Optimal Artificial Curiosity, Creativity, Music, and the Fine Arts," *Connection Science*, vol. 18, no. 2, pp. 173–187, 2006.
- [8] P.-Y. Oudeyer and F. Kaplan, "How Can We Define Intrinsic Motivation?" in *Proc. 8th Conf. Epigenetic Robotics*, 2008, pp. 93–101.
- [9] R. Ryan and E. Deci, "Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions," *Contemporary Educational Psychology*, vol. 25, no. 1, pp. 54–67, 2000.
- [10] A. S. Klyubin, D. Polani, and C. L. Nehaniv, "Keep Your Options Open: An Information-Based Driving Principle for Sensorimotor Systems," *PloS one*, vol. 3, no. 12, pp. 1–14, 2008.
- [11] C. Guckelsberger and C. Salge, "Does Empowerment Maximisation Allow for Enactive Artificial Agents?" in *Proc. 15th Conf. ALIFE (to appear)*, 2016.
- [12] K. McGee and A. T. Abraham, "Real-Time Team-Mate AI in Games: a Definition, Survey, & Critique," in *Proc. 5th Conf. FDG*, 2010, pp. 124–131.
- [13] A. Fern and P. Tadepalli, "A Computational Decision Theory for Interactive Assistants," in *Proc. 23rd Conf. NIPS*, 2010, pp. 577–585.
- [14] T.-H. D. Nguyen, D. Hsu, W.-S. Lee, T.-Y. Leong, L. P. Kaelbling, T. Lozano-Pérez, and A. H. Grant, "CAPIR: Collaborative Action Planning with Intention Recognition," in *Proc. 7th Conf. AIIDE*, 2011.
- [15] O. Macindoe, L. P. Kaelbling, and T. Lozano-Pérez, "POMCoP: Belief Space Planning for Sidekicks in Cooperative Games," in *Proc. 8th Conf. AIIDE*, 2012.
- [16] K. E. Merrick and M. L. Maher, *Motivated Reinforcement Learning. Curious Characters for Multiuser Games*. Springer, 2009.
- [17] S. Singh, A. Barto, and N. Chentanez, "Intrinsically Motivated Reinforcement Learning," *Proc. 18th Conf. NIPS*, vol. 17, pp. 1281–1288, 2004.
- [18] T. Anthony, D. Polani, and C. L. Nehaniv, "General Self-Motivation and Strategy Identification: Case Studies based on Sokoban and Pac-Man," *IEEE TCIAIG*, vol. 6, no. 1, pp. 1–17, 2014.
- [19] S. Mohamed and D. Rezende, "Stochastic Variational Information Maximisation," *Proc. 29th Conf. NIPS*, pp. 1–9, 2015.
- [20] C. Guckelsberger, C. Salge, R. Saunders, and S. Colton, "Supportive and Antagonistic Behaviour in Distributed Computational Creativity via Coupled Empowerment Maximisation," in *Proc. 7th Int. Conf. Computational Creativity*, 2016.
- [21] C. Salge, C. Glackin, and D. Polani, "Empowerment – an Introduction," *Guided Self-Organization: Inception*, pp. 67–114, 2014.
- [22] H. Touchette and S. Lloyd, "Information-Theoretic Approach to the Study of Control Systems," *Physica A*, vol. 331, 2004.
- [23] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed. Wiley-Interscience, 2006.
- [24] C. Guckelsberger and D. Polani, "Effects of Anticipation in Individually Motivated Behaviour on Survival and Control in a Multi-Agent Scenario with Resource Constraints," *Entropy*, vol. 16, no. 6, pp. 3357–3378, 2014.
- [25] D. Jallov, S. Risi, and J. Togelius, "EvoCommander: A Novel Game Based on Evolving and Switching Between Artificial Brains," *IEEE TCIAIG*, 2016.



# Generating Heuristics for Novice Players

Fernando de Mesentier Silva, Aaron Isaksen, Julian Togelius, Andy Nealen  
Tandon School of Engineering, New York University

**Abstract**—We consider the problem of generating compact sub-optimal game-playing heuristics that can be understood and easily executed by novices. In particular, we seek to find heuristics that can lead to good play while at the same time be expressed as fast and frugal trees or short decision lists. This has applications in automatically generating tutorials and instructions for playing games, but also in analyzing game design and measuring game depth. We use the classic game Blackjack as a testbed, and compare condition induction with the RIPPER algorithm, exhaustive-greedy search in statement space, genetic programming and axis-aligned search. We find that all of these methods can find compact well-playing heuristics under the given constraints, with axis-aligned search performing particularly well.

## I. INTRODUCTION

Much artificial intelligence and game theory research is focused on quickly finding optimal moves in games; however discovering optimal moves is often not practical for human players. Novices look for ways to understand basic concepts, players may have a limited amount of information in their working memory, experts may try to trick their opponents using gambits or other risky moves, and most games are just too complicated for humans (or machines) to evaluate in real-time to make optimal moves. In this paper, we focus on algorithmically generating simple introductory heuristics for novice players.

Nonetheless, novices are not the only types of players who use simplified models to make decisions. Instead of humans acting as purely rational agents making optimal decisions, the behavioral economic theory of *bounded rationality* claims that people make decisions based on a limited amount of available information and decision-making time [1], [2]. The process of making best guesses instead of optimally rational decisions is called *satisficing*. By using simple heuristics to make decisions, accuracy can actually improve over more complicated algorithms because simple guidelines are easier to execute without errors [3].

Teaching beginners a good simple strategy for a new game can be a challenge but essential for enjoyment of the game. Winning a match gives a sense of pleasure, but learning how to play and improving can lead to a sense of accomplishment [4]. A game in which it is difficult to learn basic strategies may be overwhelming for new players; a game in which high-performing strategies are easy to come up with may not be entertaining in the long run. Strategies that lead to moves that are effective, simple to execute, easy to remember, and allow the player to further improve are ideal. Examples of simple heuristics for well-known games include playing on the middle

and corners before the sides in Tic-Tac-Toe, while in Chess a popular heuristic is learning the relative value of the pieces or simple opening moves to begin the game.

Many well-respected games allow the player to “climb a heuristic ladder” in which they learn deeper and more complicated strategies [5]. The length of the heuristic ladder can give a sense of a game’s depth. Strategies used by beginner players at Chess may be very different from those used by professionals. Players look for others of the same skill level to play against, and as they gather more experience their gameplay improves leading to more sophisticated and effective moves. Strategies that once looked overwhelming and confusing can become accessible. Games such as Chess and Go have such a high-dimensional game state space that they permit strategies only accessible to players of significantly higher skill levels, measurable in Go by the Kyu and Dan rankings and in Chess by Elo and Titles. On the other hand, easier games like Tic-Tac-Toe are more accessible for new players for having relatively simple and effective moves as a result of a low-dimensional space of strategies. Players of Tic-Tac-Toe don’t need years of training to improve their strategies to reach an optimal strategy; therefore they usually stop playing it after learning or discovering the optimal strategy as there is no longer any reason to learn and improve.

A Fast and Frugal Tree (FFT) is a particularly good form of human-usable heuristic and is commonly used in bounded rationality theory. FFTs are easy to process and in practice have very good performance over more complicated algorithms [6]. A FFT is a type of binary decision tree where at each decision node, one path leads to a terminal action and the other path either leads to a fast and frugal sub-tree or a default action, as shown Figure 1. These trees can also be implemented as series of if/elseif/else statements or as a decision list [7], [8]. In this paper, we generate FFTs that can be used by beginners for effectively playing the game of Blackjack.

In this work we describe some techniques to generate easy to understand and effective fast and frugal heuristics. We evaluate and compare the results produced by the various methods. Our test case game is Blackjack, as a powerful heuristic called Basic Strategy is already known for the game. This allows us to verify that our results agree with existing theory on Blackjack [9]. Additionally, Blackjack is a two player game where the second player (dealer) must always play by a pre-determined algorithm only after the first player has completed their actions – this makes it easier to analyze than games where both players take turns and are more free in their action selection.

Expert agents to efficiently play games have been explored for many games, including Chess [10], Othello [11], Checkers [12] and Go [13]. These agents are meant to win as often as possible given an efficient amount of computation, and in many

This work was done with the support of CAPES, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil.

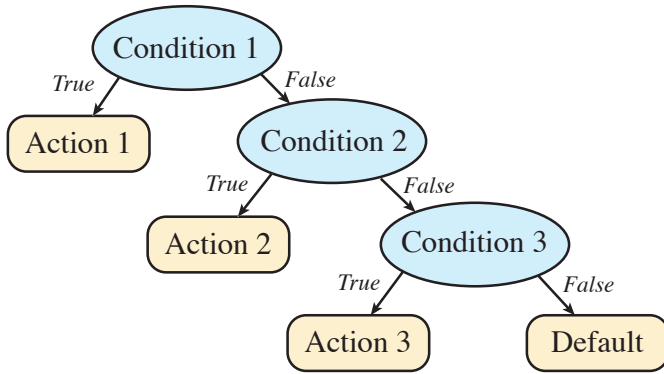


Fig. 1. Fast and Frugal Trees are a type of binary decision tree used in bounded rationality theory. They are effective and easy to learn heuristic for humans. In this paper, we generate fast and frugal trees – also known as decision lists – for beginning strategy for Blackjack.

cases they are able to compete on the level of world champion human players. The game playing heuristics we introduce in this paper are of much lower skill level as we expect a trade-off between performance and simplicity. The closer these simple heuristics get to optimal play in terms of expected value, the less need a player has to improve, and thus could possibly be used as a metric to estimate the potential depth of a game.

Evolutionary algorithms are especially effective at developing strategies for playing games at a high-performing level. Genetic Programming was used to evolve players for traditional adversarial board and card games such as Chess endgame [14], Lose Checkers [15], Backgammon [16] and Poker [17]. Adversarial video games such as Core War [18] and Robocode [19] incorporate evolved agents. Other work on evolving strategies include: Evolution of neurons for a neuro-network using reinforcement learning to generate controllers capable of playing levels on a Super Mario Bros clone [20], using genetic programming to create players for a solo variant of the game Pong [21], evolving agents to play the game Pac-Man [22] and evolving controllers with general driving skills [23]. Blackjack strategies have been evolved using strategy tables [24], genetic programming trees [25], and neural networks [26], [27] but these were focused on improving expert play, not focused on simple guidelines for novices. Blackjack AIs have been used to model and predict human gambling behaviors [28], in particular the ways that humans process and recall information [29].

Similarly to Hyper-heuristics [30] we search in heuristic space, however we only search for novice-friendly heuristics for a single problem, instead of switching between heuristics.

In the following sections, we first describe the rules of Blackjack and the Fast and Frugal Tree format we use for generating novice heuristics. The next four sections each describe a particular method for generating compact heuristics, as well as the results from applying these algorithms and heuristics to Blackjack. Finally we discuss the takeaways of this comparative study, as well as the prospects for generalizing the method to other games.

## II. BLACKJACK

For evaluating different methods for generating fast and frugal heuristics for novice players, we selected Blackjack as a study case. Blackjack is a well-known gambling game played all over the world. It uses one or more standard 52 card decks, but 8 decks is typical in modern casinos. In this paper we assume an infinite number of decks to avoid issues with counting cards or distributional effects caused by drawing without replacement.

Blackjack is played against a dealer. The player tries to build a hand of cards that beats the dealer's hand, without going over 21 points. The game starts with a player betting and then two cards are dealt to the player while the dealer gets one card face down and one face up. Play proceeds with the player making all their moves followed by the dealer.

The player has 4 actions to choose from: *hit*, the player receives the next card from the deck; *stand*, the player stops and plays proceeds to the dealer; *double down*, the player doubles their bet, receives exactly one more card and then plays proceeds to the dealer; and *split*, the player splits their two cards into two new hands and the player matches the initial bet for the new second hand. A player can only split when they have 2 cards and they are of the same value. The player loses immediately if they goes over 21 points, called a *bust*. After hitting or splitting, the player can continue to make more actions until they bust or stand.

Cards in Blackjack are worth their face value, independent of their suit, with the exception of the Jack, Queen and King which are all worth 10 points and the Ace which is worth either 1 or 11. The Ace takes on the value of whichever makes the hand worth the most amount of points without busting. If the hand contains an Ace that can still change value from 11 to 1 it is called *Soft*.

After the player made all their moves for a hand, play proceeds to the dealer. The dealer plays his moves following an algorithm: they *hit* while the total amount of points in their hand is below 17. When they reaches 17 or more they *stand*. If the dealer's score exceeds 21 the dealer busts and the player wins the hand (if they have not already busted).

There is one special case: when a hand is composed by an Ace and one other card that is worth 10 points it is considered a Blackjack. A Blackjack beats any other hand of 21 or less, except another Blackjack, which results in a tie.

In Blackjack the dealer, who plays for the casino, has the advantage over the player. That mainly comes from the fact he plays last and so the player can lose the game by *busting* without the dealer ever having to make a move. Although the game has an element of randomness in the card drawing, Blackjack has some skill [31]. Strategies for playing the game to reduce the dealer's advantage go from what is called Basic Strategy [9] or Optimum Strategy [32], which closely relates to the work we do in this paper, to more complex such as card counting, which we do not address as it is not for novice play.

The Basic Strategy shown on Figure 2 determines the best move for the player over the hand of the player and the card the dealer is showing. Special cases occur when the player holds an Ace or a hand that can be split.

		Player Hand																														Dealer Card															
		5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21											A,2	A,3	A,4	A,5	A,6	A,7	A,8	A,9	A,10	2,2	3,3	4,4	5,5	6,6	7,7	8,8	9,9	10,10	A,A
Dealer Card	2	h	h	h	h	h	d	d	h	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	h	h	h	h	h	s	s	s	s	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	
	3	h	h	h	h	h	d	d	h	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	h	h	h	h	h	s	s	s	s	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp		
	4	h	h	h	h	h	d	d	h	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	h	h	h	h	h	s	s	s	s	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp		
	5	h	h	h	h	h	d	d	h	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	h	h	h	h	h	s	s	s	s	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp		
	6	h	h	h	h	h	d	d	h	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	h	h	h	h	h	s	s	s	s	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp		
	7	h	h	h	h	h	d	d	h	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	h	h	h	h	h	s	s	s	s	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp		
	8	h	h	h	h	h	d	d	h	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	h	h	h	h	h	s	s	s	s	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp		
	9	h	h	h	h	h	d	d	h	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	h	h	h	h	h	s	s	s	s	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp		
	10	h	h	h	h	h	h	d	h	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	h	h	h	h	h	s	s	s	s	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp		
	A	h	h	h	h	h	h	h	h	h	h	h	h	h	h	s	s	s	s	s	s	s	s	s	s	s	h	h	h	h	h	s	s	s	s	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp		

		Player Hand																				Dealer Card																						
		A,2	A,3	A,4	A,5	A,6	A,7	A,8	A,9	A,10											2,2	3,3	4,4	5,5	6,6	7,7	8,8	9,9	10,10	A,A														
Dealer Card	2	h	h	h	h	h	h	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	h	h	h	h	h	s	s	s	s	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp
	3	h	h	h	h	h	d	d	d	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	h	h	h	h	h	s	s	s	s	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	
	4	h	h	d	d	d	d	d	d	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	h	h	h	h	h	s	s	s	s	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	
	5	d	d	d	d	d	d	d	d	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	h	h	h	h	h	s	s	s	s	sp	sp	sp	sp	sp	sp	sp	sp	sp		
	6	d	d	d	d	d	d	d	d	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	h	h	h	h	h	s	s	s	s	sp	sp	sp	sp	sp	sp	sp	sp	sp		
	7	h	h	h	h	h	h	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	h	h	h	h	h	s	s	s	s	sp	sp	sp	sp	sp	sp	sp	sp	sp		
	8	h	h	h	h	h	h	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	h	h	h	h	h	s	s	s	s	sp	sp	sp	sp	sp	sp	sp	sp	sp		
	9	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	s	s	s	s	sp	sp	sp	sp	sp	sp	sp	sp	sp		
	10	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	s	s	s	s	sp	sp	sp	sp	sp	sp	sp	sp	sp		
	A	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	s	s	s	s	sp	sp	sp	sp	sp	sp	sp	sp	sp		

		Player Hand																				Dealer Card																				
		2,2	3,3	4,4	5,5	6,6	7,7	8,8	9,9	10,10	A,A											2,2	3,3	4,4	5,5	6,6	7,7	8,8	9,9	10,10	A,A											
Dealer Card	2	sp	sp	h	d	sp	sp	sp	sp	s	sp	sp	sp	sp	sp	sp	sp	sp	h	d	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp
	3	sp	sp	h	d	sp	sp	sp	sp	s	sp	sp	sp	sp	sp	sp	sp	sp	h	d	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp		
	4	sp	sp	h	d	sp	sp	sp	sp	s	sp	sp	sp	sp	sp	sp	sp	sp	h	d	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp		
	5	sp	sp	sp	d	sp	sp	sp	sp	s	sp	sp	sp	sp	sp	sp	sp	sp	h	d	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp		
	6	sp	sp	sp	d	sp	sp	sp	sp	s	sp	sp	sp	sp	sp	sp	sp	sp	h	d	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp			
	7	sp	sp	h	d	h	sp	sp	s	sp	sp	sp	sp	sp	sp	sp	sp	sp	h	d	h	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp			
	8	h	h	h	d	h	h	sp	sp	s	sp	sp	sp	sp	sp	sp	sp	sp	h	d	h	h	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp	sp				
	9	h	h	h	d	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h				
	10	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h				
	A	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h				

Fig. 2. The Basic Strategy table. *s*=stand, *h*=hit, *d*=double down and *sp*=split. By our definition, the table has a fitness of 0 (expected loss of -0.0051875).

Despite being simple to read, the Basic Strategy has several levels of granularity and so it can be hard to remember all possible scenarios. Additionally, referring to the card while playing is allowed but error prone, especially for beginner level players. In the next sections we present several methods for generating simple novice level heuristics for Blackjack.

### III. HEURISTIC REPRESENTATION

The simple human-playable heuristics we generate can be represented as fast and frugal trees, decision lists, or as a series of if/elseif/else statements. For compactness, we use the if/elseif/else format for this paper but they are all equivalent. The heuristics are composed of any number of condition/action pair and a default action. Each condition/action pair is represented as if/elseif-statements. Conditions are formed of one or more clauses that must all be true for a condition to be true. The default action is taken if all the conditions are not satisfied, and is represented by an else-statement.

Each clause is permitted to analyze  $P_{pts}$ , the total number of points currently held by the player, and  $D_{pts}$ , the number of points on the dealer's visible card.  $P_{pts}$  can be compared to  $\pi_{lower}$  and  $\pi_{upper}$ , parameterized lower and upper bounds for  $P_{pts}$ .  $\delta_{lower}$  and  $\delta_{upper}$  are parameters for lower and upper bounds for the clauses matching  $D_{pts}$ .

Two conditional boolean statements can also be checked, *canSplit* and *isSoft*. If a player's hand has two cards and they are both of the same value, *canSplit* is *True*, *False* otherwise. If there is still an Ace in the player's hand that can change value, *isSoft* is *True*, on any other case it is *False*.

With such, our heuristics are formed as a set of statements with the following structure:

```

if CONDITION 1 then ACTION 1
else if CONDITION 2 then ACTION 2
else if ... then ...
else DEFAULT ACTION

```

where each condition is formed by conjoining together one or more clauses selected from  $\pi_{lower} \leq P_{pts}$ ,  $P_{pts} \leq \pi_{upper}$ ,  $\delta_{lower} \leq D_{pts}$ ,  $D_{pts} \leq \delta_{upper}$ , *canSplit*, *not canSplit*, *isSoft* and *not isSoft*.

We define the complexity of a heuristic by summing up the total number of clauses plus the number of actions plus 1 for the default action. This coarse definition does not take into account that smaller numbers might be easier to memorize

than longer ones, or some orderings of conditions might be easier to recall. However, our definition allows us a simple way of comparing more complex heuristics with simpler ones.

We define fitness to be the expected value of the heuristic minus the expected value of Basic Strategy.

### IV. METHODS FOR FAST SIMULATION

In order to evaluate the quality of a heuristic, we need to test it against a large number of possible hands. We are thus incentivized to make our simulation run as fast as possible. We do this by (1) precomputing as many results as possible and (2) parallelizing the calculation.

Dealer will act only after having complete information of what the Player's actions have been. Therefore, given (1) the final sum of the Player's cards, (2) assuming an infinite deck (i.e. choosing with replacement) where the distribution of cards is the same for every draw, and (3) a fixed algorithm that the Dealer must follow on their turn, we can precompute all possible plays for the Dealer once the Player has finished their turn. This precomputation allows us to store the expected value of a Player's final score given the Dealer's visible card.

We calculate the table of expected values as follows. Given every possible hand of cards for the Dealer and final score for the Player, there are a total of 79,489 situations that might occur. For each possible hand, we know if the game is a win (+1), tie (0), or loss (-1) given a Player's final score. For each final score for the Player and face-up Dealer card, we calculate the expected value by summing the number of wins minus losses, dividing by the total number of possible hands given that first card of the Dealer. We store these expected values in a table. Instead of playing the Dealer hands, we just look up the expected value from the table and use the precomputed result of stopping with that player score against the dealer's visible card. Double downs need to multiply the expected value by 2, and Blackjacks for the Player count as 1.5 wins instead of 1. Splits require special handling, because the outcome of the Player's two hands are not independent. Given the Player's two final scores from the split, we calculate the expected value of the hands together and store this in a separate table.

Unfortunately, we can't precompute the Player's hands because the Player is able to stop at any time which gives rise to too many possible hands to iterate through (and some of them are extremely unlikely to occur, such as a hand with

ten 2's and one ace). However, we can be sure to deal out all possible starting 2 cards for the Player and 1 starting card for the Dealer to ensure the simulations cover a wide range of the most likely outcomes. For each of these  $13 \times 13 \times 13 = 2,197$  starting conditions, we simulate between 100 and 500 games, depending on the speed and accuracy required.

Because every game is independent, we can parallelize the simulation across multiple cores. We use the python multiprocessing library to split up the calculation across cores on a single computer. In addition, we use Cython to compile the simulation steps into faster C++.

Finally to avoid sampling errors where running equivalent strategies would give different results depending on which cards were drawn, we use the same seed for every simulation. This ensures that equivalent strategies will get exactly the same fitness every time we run the simulation.

## V. INDUCTING HEURISTICS FROM THE BASIC STRATEGY

Following our expressions format we can generate decision lists [7] as our heuristics, as long as we have a database to extract our expressions from. For every possible initial configuration of the game, between the player's and dealer's hands, we simulated 200,000 games of Blackjack for each of the possible moves for the player's hand. Starting from the games with the higher starting hand value and working in descending order to the games with the lowest value, we tried to generate a table analogous to the Basic Strategy. From the average score of the playouts, the move with highest score was selected. By building the table in descending order of points, we could reference previous results to decide the next action to take after a move draws a new card. The database has 550 entries. We compared the results to the moves that would have been picked by the Basic Strategy. Both pick the same move for 94% of the entries in the data. The case in which they differ were mostly when picking *split* as a move with the Basic Strategy. With further analysis, we could tell that increasing the number of simulations would make the diverging data points converge to the Basic Strategy selections. For this reason, we decided to generate a database in the same format, but using the move selection according to the Basic Strategy.

To extract the heuristics from the database we use the Repeated Incremental Pruning to Produce Error Reduction (RIPPER) [33] algorithm. The algorithm uses a grow and prune approach, followed by a revision stage. The database is split into two,  $\frac{2}{3}$  is used in the grow step and the other  $\frac{1}{3}$  in the prune step. For each class in the data, from the least prevalent to the most, conditions to classify that class are grown from the database by adding clauses until maximum information gain is reached. Pruning is then done on the condition to maximize a target function. The revision stage then analyzes each condition, in the order they were learned, and generate two new candidates for it. One, the replacement, is obtained by growing and pruning a new condition, where pruning looks to minimize the error on the set where the condition was replaced by this new candidate. The other, the revision, is generated by greedily adding more clauses to the condition. Finally a decision is made based on a heuristic to decide which to keep,

the original condition, the replacement or the revision. To the conditions a resulting action is added. For Blackjack, the possible moves represent the classes. The most prevalent class, in this case *hit*, is used as the default move.

```

if canSplit and  $11 \leq P_{pts} \leq 18$  and  $D_{pts} \leq 6$  then
    SPLIT
else if canSplit and isSoft then SPLIT
else if canSplit and  $P_{pts} \leq 6$  and  $D_{pts} \leq 7$  then
    SPLIT
else if canSplit and  $P_{pts} \leq 8$  and  $5 \leq D_{pts} \leq 6$  then
    SPLIT
else if canSplit and  $13 \leq P_{pts} \leq 18$  then SPLIT
else if  $10 \leq P_{pts} \leq 11$  and  $D_{pts} \leq 9$  then
    DOUBLEDOWN
else if isSoft and  $P_{pts} \leq 18$  and  $5 \leq D_{pts} \leq 6$  then
    DOUBLEDOWN
else if  $9 \leq P_{pts} \leq 11$  and  $3 \leq D_{pts} \leq 6$  then
    DOUBLEDOWN
else if isSoft and  $17 \leq P_{pts} \leq 18$  and  $3 \leq D_{pts} \leq 4$ 
then
    DOUBLEDOWN
else if  $17 \leq P_{pts}$  then STAND
else if  $10 \leq P_{pts}$  and  $D_{pts} \leq 6$  then STAND
else HIT

```

Fig. 3. The heuristic generated using the RIPPER algorithm. The set correctly classified 92% of the entries in the database used. It has a fitness of -0.0134.

The results are shown in figure 3. The algorithm came up with a set of 11 statements, plus the default move. To simplify readability of the statements, we do a post-processing of the results: reduce the set of clauses in the conjunctions of a condition, if there are clauses whose coverage is already part of another clause in the same condition; reorder the clauses for readability, ordered from lower to upper bound coverage. Since one of our goals was to reduce the granularity of heuristic complexity in respect to the basic strategy, we believe that the algorithm successfully achieves that up to a point. The set generated misclassified 8% of the entries, meaning that it chooses a different move that contained in the database. By looking at the proximity in fitness, the misclassified scenarios have a very low impact on the outcome.

In terms of generating novice level heuristics, the one inducted by RIPPER is unwieldy. The conditions containing *canSplit* and *isSoft* together with the number of statements appears to be convoluted for a beginner. To address such, we decided to remove *isSoft* from the possible clauses and run the algorithm again, but we were still left with 8 rules and too much complexity. So we ran the algorithm once more removing both *isSoft* and *canSplit* from the possible clauses. Figure 4 shows the new heuristic, which is a lot closer to our goal, having cleaner conditions and almost half the total statements.

As the algorithm goes, it looks to generate conditions to classify members of each class, from the least prevalent to the most. Since the statements that make up the heuristic the algorithm generates start from least used moves, the later statements have much more impact on improving the score for the player. Since the heuristic generated by RIPPER has

```

if  $P_{pts} \leq 4$  and  $D_{pts} \leq 7$  then SPLIT
else if  $10 \leq P_{pts} \leq 11$  and  $D_{pts} \leq 10$  then DOUBLEDOWN
else if  $9 \leq P_{pts} \leq 10$  and  $4 \leq D_{pts} \leq 6$  then
  DOUBLEDOWN
else if  $13 \leq P_{pts}$  and  $D_{pts} \leq 6$  then STAND
else if  $17 \leq P_{pts}$  then STAND
else if  $10 \leq P_{pts}$  and  $4 \leq D_{pts} \leq 6$  then STAND
else HIT

```

Fig. 4. Using the RIPPER algorithm the heuristic generated from our database without *canSplit* and *isSplit*. The set correctly classify 83% of the entries in the database used. This heuristic has a fitness of -0.0234.

its coverage dependent on the order of the conditions found, rearranging their positions would change the outcome. We believe that having a heuristic in which the statements are presented in descending order of their positive impact on the gameplay would lead to a more flexible heuristic, i.e. if the number of statements in the heuristic is too overwhelming for a novice player, we could discard the bottom ones and have a tighter set size with effective fitness value. We also wanted to test the hypothesis of whether a heuristic generated in such fashion could outperform the decision list generated by RIPPER using a smaller number of statements, since we know that playing the Basic Strategy was more profitable.

## VI. EXHAUSTIVE-GREEDY SEARCH

To produce a heuristic with the most impactful statement in each step we decided to use a exhaustive-greedy search algorithm. After playing the game for each of the possible statements, it picks the one that achieves the best average score. The algorithm starts with just the default action and with each new iteration the candidate statements are tested at the bottom of the current set, before the default move. Each selected statement is appended right before the default action. We wanted to create a set of statements that could be flexible, so that we can reduce the complexity by trimming the statements on the heuristic while trying to have low impact on its fitness. We then append the rules to enabling removing the later statements without impacting the previous.

The algorithm tries different default actions when searching for a heuristic. Having *double down* or *split* as default moves did not generate statements with good fitness as they were a much smaller part of the strategy when compared to the two other moves. This lead to a lengthier set, as statements are added to cover both *hit* and *stand*. The 1 statement heuristic found proved *hit* to be a better default action for the algorithm we were using. Figure 5 shows the same 1 statement heuristic found with *hit* and *stand* as the default action.

```

if  $16 \leq P_{pts}$  then STAND      if  $P_{pts} \leq 15$  then HIT
else HIT                        else STAND

```

Fig. 5. The first statement found using exhaustive-greedy search with *hit* and *stand* as default moves. Fitness is -0.0546.

Since the exhaustive-greedy search algorithm added statements to the end of the set, the first statement found had a

great impact. The 1 step heuristics found picked the same moves in every setup, but the space left for the next ones to be found is very different. The *stand* default move set could only grow new statements for when the player had more than 15 points, which is a much more limited space than the one where player has less than 16 points. And when looking at the Basic Strategy we can see that the higher hand value strategies are composed mostly of *stand* since the chances of *busting* are very high. With such conditions, we decided to keep *hit* as the default action and reduce the search space that the algorithm had to explore. Since RIPPER also has the same default move, it is also easier to compare the heuristics generated.

To run the algorithm we needed to generate all possible statements to evaluate. When generating them, we can apply domain knowledge to reduce the search space: *canSplit* only needs to be called if the move being targeted is *split* and we don't need to constraint the player points by odd numbers in that scenario. Generating all statements to cover all possible combinations of conditions and moves, we have 9,405 statements for *stand*, 9,405 for *double down* and 3,025 for *split*, for a total of 21,835 different statements. Because the search space is already this large, we decided not to generate heuristics that use *isSoft*, as it would result in doubling the number of possible statements, for what it seemed to be a small gain in fitness. Figure 6 shows the heuristic found.

```

if  $16 \leq P_{pts}$  then STAND
else if  $9 \leq P_{pts} \leq 11$  and  $D_{pts} \leq 8$  then DOUBLEDOWN
else if  $13 \leq P_{pts} \leq 15$  and  $D_{pts} \leq 6$  then STAND
else HIT

```

Fig. 6. Greedy search selecting the highest average scoring statement. The algorithm reached a local maximum on this 3 step heuristic. Fitness is -0.0302.

When comparing both, the RIPPER simpler set, shown on Figure 4, outperforms the exhaustive-greedy search by 0.007 average fitness score. That result comes from increasing complexity: the RIPPER heuristic has 3 more statements.

Greedy search for the best heuristic revealed an interesting aspect of the game: if we had a single step to follow it would be for the player to *stand* when having 16 points or more. That contradicts a first impression from looking at the Basic Strategy that the player should *stand* on 17 or more and on 16 if the dealer has 6 or less points, *hit* otherwise. Following the nature of our decision structure, since we are only looking at a 1-statement heuristic, what the greedy selection shows is that, if we look at only the player points, when having to make a choice of which move to make at 16 points, in average, is better to *stand* than to *hit*. This is specially interesting since 16 is considered the worst hand in Blackjack [34].

Another interesting result was how fast the local optimum was reached. The contender for fourth statement had no impact since its condition was already covered in the set. As a result of such and the observation that a heuristic that would more closely resemble the Basic Strategy could have a better average, we believe that greedily searching for a heuristic is very prone to getting stuck on local optimums. In order to get better results from searching we decided to expand our greedy approach.

## VII. EXPANDING THE EXHAUSTIVE-GREEDY SEARCH

Keeping the exhaustive search approach, we try to generate a new heuristic by making the greedy aspect more expansive. For that, we decided to no longer pick only the best statement in each step, but instead register the top 5 ranking statements for each step. On every iteration of the algorithm works in the same fashion as exhaustive-greedy search, but now the 5 top results are store and iterated upon separately. After the last iteration we take the set with the best average score in the last step. Figure 7 shows the heuristic found.

```

if  $17 \leq P_{pts}$  then STAND
else if  $13 \leq P_{pts}$  and  $D_{pts} \leq 6$  then STAND
else if  $10 \leq P_{pts} \leq 11$  and  $3 \leq D_{pts} \leq 9$  then DOUBLE
else HIT

```

Fig. 7. Using a greedy exhaustive search and considering the top 5 average scoring statements. The top resulting heuristic the algorithm found is shown. This set is a local maximum. This heuristic has a fitness of -0.0247.

Searching over the 5 best result shows that the heuristic found by the greedy algorithm was a local optimum, despite also being subject to such. The heuristic found outperforms and is as simple as the one found by exhaustive-greedy, and is also closer to the Basic Strategy. The first two statements found are exactly on par with the Basic Strategy, but looking at the bottom, we can see that, to match the Basic Strategy, it needed to also include *double down* when the dealer is showing a 2. That result comes from the fact that the case of the dealer showing a 2 against a player's hand of 10 or 11 is present in only a very small number of games, if at all.

Another downside is that the new heuristic found was another local optimum. Running another loop of the algorithm finds no new statements, at the top 5, that cover a new scenario. We can see from looking at the Basic Strategy layout that there are other cases the set does not cover.

A better search algorithm would be needed to look for the best n-statement heuristic algorithm. By limiting the size of our search space by removing *isSoft* there is still a very large search space. An exhaustive search of the full space would guarantee the optimum heuristic, but would also be very computationally expensive, even by pruning the statements that do not further cover new elements. To look further for a better strategy for generating simple heuristics we then turn to algorithms that employ different search strategies.

## VIII. AXIS-ALIGNED SEARCH

Another way to search the space of possible heuristics is to use an axis-aligned search algorithm, modifying each parameter in the conditions individually along each single dimension. For example, for a 1-condition heuristic, we would individually test all lower bounds  $\beta_{lower}$  for  $P_{pts}$  keeping all other values fixed, then all upper bounds  $\beta_{upper}$  keeping all other values fixed, etc. also exploring the lower and upper bounds for  $D_{pts}$  and all actions for the condition and default action. Because we do not have a large number of possible values for each parameter (between 3 and 21 depending on the parameter), we can search all values along each single dimension – on the

order of around 100 different heuristics to test for a 1-condition Blackjack heuristic. However, we don't need to evaluate all of these heuristics because many of them are equivalent, so we first pass through and remove any heuristics which are equivalent to any others. This has the effect of only needing to search approximately 75 statements per iteration for a 1-condition heuristic, approximately 210 for a 3-condition heuristic, and approximately 375 statements per iteration for a 5-condition heuristic. This axis-aligned search has the benefit that it can get out of local maximum as long as another maximum exists anywhere on one of the dimensions holding all other dimensions fixed. For games where parameters may be too many values to search completely, one could use importance sampling or local neighborhood search to find a smaller number of parameters to choose along each dimension.

The algorithm proceeds as follows. We begin from a random statement, and find its fitness. We then find all unique statements along each individual dimension and find their fitnesses. We take the one with the best fitness and repeat the algorithm from there. When we find a statement that has the best fitness of any of its candidates, we terminate and return the best candidate found, which is a local best but not necessarily a global best. Because this is not guaranteed to find the global best (as is also the case for simulated annealing and other optimization algorithms), we repeat the process several times and return the best from all runs. We find that axis-aligned search performs very well, finding some of the highest fitness heuristics for a given complexity, but it has a high variance and often finds badly performing ones. It is therefore essential to run several times. It can also be used as a final touch-up process for other algorithms to search for any final improvements.

For 1-condition heuristics, axis-aligned search would very often find the optimal heuristics presented in Figure 5. The best performing 3-condition heuristic is presented in Figure 8. Split/Hit refers to the player using the split action when legal, otherwise they would treat this as a hit. No other algorithm found this highly performing heuristic, with a high fitness of -0.0221 especially given its relatively low complexity.

```

if  $17 \leq P_{pts}$  then STAND
else if  $13 \leq P_{pts}$  and  $D_{pts} \leq 6$  then STAND
else if  $10 \leq P_{pts} \leq 11$  and  $D_{pts} \leq 9$  then DOUBLEDOWN
else SPLIT/HIT

```

Fig. 8. Starting from random conditions, the best 3-condition heuristic observed for multiple runs of the Axis-Aligned Search algorithm. The heuristic has a fitness of -0.0221.

## IX. GENETIC PROGRAMMING

Finally, we examined using genetic programming [35] to find the best decision list strategies at a given complexity for use by novice players. Our approach is similar to that used in [25]. We used the DEAP [36] framework for genetic algorithms (GA), which includes implementations for various types of stochastic optimization. We experimented with a basic genetic algorithm as well as  $(\mu + \lambda) - ES$  [37].

We test the population of potential strategies by simulating  $13*13*13*500 = 1,098,500$  games for each individual. The



genotype contains the default action and a list of conditions bounds and actions. Each simple condition is made of four possible clauses  $\pi_{lower} \leq P_{pts}$ ,  $P_{pts} \leq \pi_{upper}$ ,  $\delta_{lower} \leq D_{pts}$ , and  $D_{pts} \leq \delta_{upper}$  as described in Section III. Because a clause is meaningless if  $\pi_{upper} < \pi_{lower}$  or  $\delta_{upper} < \delta_{lower}$ , each condition instead stores 4 positive integers  $\pi_{lower}, \pi_{upper} - \pi_{lower}, \delta_{lower}, \delta_{upper} - \delta_{lower}$  to ensure that upper bounds are never smaller than lower bounds. During mutation, we ensure that the ranges are not violated by clamping to sensible values that can only occur in Blackjack. We also include a boolean for each clause, that allows the clause to be easily turned on and off during mutation. For complex strategies that are allowed to include *canSplit* and *isSoft* this adds 3 more variants where each item can be  $=$ ,  $\neq$ , or *don't care*.

The hyperparameters used for controlling the genetic algorithm needed to be tuned depending on the complexity of the statements that were to be generated. For heuristics with 1 to 3 conditions, we found good results using 30% chance of modifying each value in a genotype, 20% chance to flip on/off a clause, 20% chance to shuffle the order of condition/actions, 40% chance to create an offspring with mutation, and 20% chance to create an offspring with crossover, a population size of 100, and 30-50 generations. For heuristics with 5 or more conditions, we found better results by using a 10% chance of modifying values, leaving all clauses on, and not shuffling the clauses, and switching to a  $(\mu + \lambda) - ES$  framework with  $\mu = 20$  and  $\lambda = 200$  for 100 generations.

Our GP did not find as highly optimized heuristics as the axis-aligned method, but on most runs would find something adequate. In general, GP seemed to have a smaller variance in final fitness but did not have the overall best results for a given complexity. However, we believe that with more generations, larger populations, and more highly-tuned hyperparameters the GA would be likely to perform better.

## X. DISCUSSION

We used several methods to generate heuristics for Blackjack. The results show expressions of different lengths, heuristic complexity and fitness. The different methods also have different run-time computational complexity. The RIPPER algorithm is very fast since the database only had 550 entries, but relies on an existing database of optimal moves. Generating the database to cross check the Basic Strategy was costly, but needed to be done only once. Doing exhaustive-greedy search was costly due to the amount of possible statements, and our top 5 greedy variation was even worse, being exponentially more expensive for generating more complex heuristics. Genetic Programming running time is related to the size of the population and number of generations needed to converge to a good solution, and requires tuning of hyperparameters. Meanwhile Axis-Aligned Search searches a much smaller number of statements, compared to the exhaustive-greedy searches, but often finds bad performing heuristics given a bad random starting point. Axis-aligned search will likely pose problems for games with significantly more parameter space to search without some tuning.

The results show an interesting relationship between the heuristics complexity and fitness. Figure 9 shows the com-

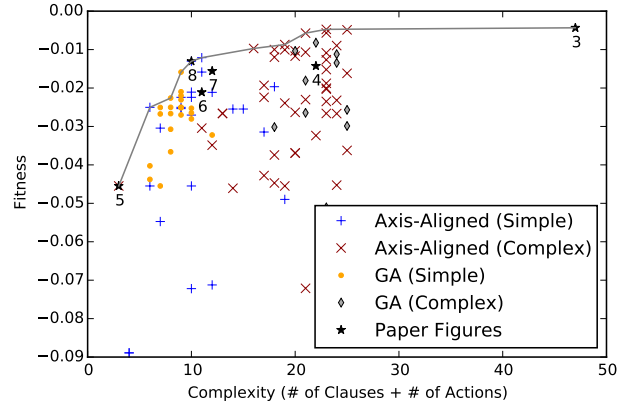


Fig. 9. Comparison of heuristic complexity vs fitness for various heuristics. The numbers in the graph refer to the corresponding Figure index for the different heuristics shown throughout this paper. 3 is RIPPER (complex), 4 is RIPPER (simple), 5 is 1-rule exhaustive-greedy, 6 is 3-rule exhaustive-greedy, 7 is top-5 greedy and 8 is Axis-Aligned.

parison of different runs of each of our algorithms, plotting the complexity of the heuristic against its fitness. *Simple* heuristics do not use the *isSoft* or *canSplit* clauses, and *complex* heuristics use all possible clauses. This shows that low-complexity rules can have a large improvement in fitness, but then smaller gains in fitness require larger increases in complexity. RIPPER expressions show up as outliers in terms of complexity, having twice as many causes and actions as most of the expressions found, but also have the heuristic with the highest ranked fitness. The highest fitted Axis-Aligned complex heuristics are half as complex as the top fitted RIPPER, but have very close fitness. By analyzing the hull atop the fitted results over the increasing complexity, we can see the trend in fitness gain is much greater on the lower complexity results.

We can observe the impact of increasing the space of possible clauses by introducing of *canSplit* and *isSoft* leads to a small fitness gain, but a much higher increase in complexity. The best heuristics generated with their addition range from 2 to close to 5 times more complexity.

We believe that our framework can be extended to other games for generating beginning heuristics, but certainly with some significant challenges. One of the major challenges is finding a primitive set of operations to include in the conditions and actions. For blackjack, these primitives are relatively obvious and easy to implement (perhaps because we have the Basic Strategy to refer to) but in other games, even simple ones like Tic-Tac-Toe, the conditions and actions can become far more complicated to encode.

For games that have a distinction between tactics and strategy, we predict that different trees, conditions, and actions would be required to represent differences between short-term tactical moves and long term high-level planning. Perhaps a player would need to first use a FFT heuristic to figure out a high-level goal to reach, and then another FFT to implement that goal.

Some games have an existing known good strategy, either obtained through collective wisdom gathered after collectively many hours, years, or centuries of study and play, or by game

tree exploration using minimax or Monte Carlo Tree Search. In this case, using a simplification algorithm such as RIPPER can make sense. For others, when the game is newly designed and doesn't yet have known-good strategies [38] or the output of a computational creativity game generation system [39], strategies need to be developed from the ground up, as we do with the axis-aligned search and genetic algorithm methods.

We plan in the future to confirm that our heuristics are indeed easy to learn and execute by novice players, by comparing accuracy and speed at which novices can perform the correct Blackjack plays given a fixed set of cards. Such a study can also give us a more accurate complexity measure for clauses.

## XI. CONCLUSIONS AND FUTURE WORK

This paper poses the problem of algorithmically generating compact heuristics that can be easily learned by novice human players. As an example we used Blackjack, a simple game for which the optimal strategy is already known but requires a significant amount of time to learn. We explored four different approaches to discovering simple fast and frugal heuristics for novice-level Blackjack. It was found that inducing conditions from a known strategy table with the RIPPER algorithm resulted in heuristics that were too large and not much better than much smaller heuristics. Exhaustive search for expressions that were linked together with greedy search was found to be comparatively slow and suffer from the tendency to find globally suboptimal choices inherent in the greedy search. Genetic programming and axis-aligned search were both able to find the desired compact yet effective heuristics. While we have shown that finding these heuristics is doable for Blackjack, the challenge of scaling up to games with larger state and action spaces where optimal play is not already known remains. We hypothesize that solving this will involve automatic extraction of relevant behavioral and positional primitives.

## REFERENCES

- [1] H. A. Simon, "Theories of bounded rationality," *Decision and organization*, vol. 1, no. 1, pp. 161–176, 1972.
- [2] D. Kahneman, "Maps of bounded rationality: Psychology for behavioral economics," *The American economic review*, vol. 93, no. 5, pp. 1449–1475, 2003.
- [3] G. Gigerenzer and D. G. Goldstein, "Reasoning the fast and frugal way: models of bounded rationality," *Psychological review*, vol. 103, no. 4, p. 650, 1996.
- [4] J. P. Gee, "Learning by design: Good video games as learning machines," *E-Learning and Digital Media*, vol. 2, no. 1, pp. 5–16, 2005.
- [5] G. S. Elias, R. Garfield, K. R. Gutschera, and P. Whitley, *Characteristics of games*. MIT Press, 2012.
- [6] G. Gigerenzer, "Fast and frugal heuristics: The tools of bounded rationality," *Blackwell handbook of judgment and decision making*, pp. 62–88, 2004.
- [7] R. L. Rivest, "Learning decision lists," *Machine learning*, vol. 2, no. 3, pp. 229–246, 1987.
- [8] D. Ashlock, M. Joenks, J. R. Koza, and W. Banzhaf, "Isac lists, a different representation for program," in *Genetic Programming 1998*. Morgan Kaufmann, 1998, pp. 3–10.
- [9] E. O. Thorp, *Beat the dealer: A winning strategy for the game of twenty-one*. Vintage, 1966.
- [10] M. Campbell, A. J. Hoane, and F.-h. Hsu, "Deep blue," *Artificial intelligence*, vol. 134, no. 1, pp. 57–83, 2002.
- [11] M. Buro, "From simple features to sophisticated evaluation functions," in *Computers and Games*. Springer, 1998, pp. 126–145.
- [12] J. Schaeffer, J. Culberson, N. Treloar, B. Knight, P. Lu, and D. Szafron, "A world championship caliber checkers program," *Artificial Intelligence*, vol. 53, no. 2, pp. 273–289, 1992.
- [13] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
- [14] A. Hauptman and M. Sipper, *GP-endchess: Using genetic programming to evolve chess endgame players*. Springer, 2005.
- [15] A. Benbassat and M. Sipper, "Evolving lose-checkers players using genetic programming," in *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*. IEEE, 2010, pp. 30–37.
- [16] Y. Azaria and M. Sipper, "Gp-gammon: Genetically programming backgammon players," *Genetic Programming and Evolvable Machines*, vol. 6, no. 3, pp. 283–300, 2005.
- [17] L. Barone and L. While, "An adaptive learning model for simplified poker using evolutionary algorithms," in *Congress on Evolutionary Computation, CEC 99*, vol. 1. IEEE, 1999.
- [18] B. Vowk, A. S. Wait, and C. Schmidt, "An evolutionary approach generates human competitive corewar programs," in *Workshop and Tutorial Proceedings Ninth International Conference on the Simulation and Synthesis of Living Systems (Alife XI)*. Citeseer, 2004, pp. 33–36.
- [19] Y. Shichel, E. Ziserman, and M. Sipper, "Gp-robocode: Using genetic programming to evolve robocode players," in *8th European Conference on Genetic Programming*, vol. 3447, 2005, pp. 143–154.
- [20] J. Togelius, S. Karakovskiy, J. Koutník, and J. Schmidhuber, "Super mario evolution," in *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*. IEEE, 2009, pp. 156–161.
- [21] W. B. Langdon and R. Poll, "Evolutionary solo pong players," in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol. 3. IEEE, 2005, pp. 2621–2628.
- [22] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. MIT press, 1992, vol. 1.
- [23] A. Agapitos, J. Togelius, and S. M. Lucas, "Evolving controllers for simulated car racing using object oriented genetic programming," in *Genetic and evolutionary computation*. ACM, 2007, pp. 1543–1550.
- [24] D. B. Fogel, "Evolving strategies in blackjack," in *Congress on Evolutionary Computation, CEC2004*, vol. 2, 2004, pp. 1427–1434.
- [25] M. Jelev and S. Koch, "Genetic blackjack," 2009.
- [26] A. Pérez-Urbe and E. Sanchez, "Blackjack as a test bed for learning strategies in neural networks," in *Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on*, vol. 3. IEEE, 1998, pp. 2022–2027.
- [27] G. Kendall and C. Smith, "The evolution of blackjack strategies," in *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, vol. 4. IEEE, 2003, pp. 2474–2481.
- [28] M. R. Schiller and F. R. Gobet, "A comparison between cognitive and ai models of blackjack strategy learning," in *KI 2012: Advances in Artificial Intelligence*. Springer, 2012, pp. 143–155.
- [29] F. Gobet, J. Retschitzki, and A. de Voogt, *Moves in mind: The psychology of board games*. Psychology Press, 2004.
- [30] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, "Hyper-heuristics: A survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, no. 12, pp. 1695–1724, 2013.
- [31] L. Humble, *The World Greatest Blackjack Book*. Main St. Books, 2010.
- [32] R. R. Baldwin, W. E. Cantey, H. Maisel, and J. P. McDermott, "The optimum strategy in blackjack," *Journal of the American Statistical Association*, vol. 51, no. 275, pp. 429–439, 1956.
- [33] W. W. Cohen, "Fast effective rule induction," in *Intl Conference on Machine Learning*, 1995, pp. 115–123.
- [34] L. Revere, *Playing Blackjack as a Business: A Professional Player's Approach to the Game of 21*. Lyle Stuart, 2000.
- [35] R. Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza, *A field guide to genetic programming*. Lulu.com, 2008.
- [36] D. Rainville, F.-A. Fortin, M.-A. Gardner, M. Parizeau, C. Gagné *et al.*, "Deap: A python framework for evolutionary algorithms," in *Genetic and evolutionary computation*. ACM, 2012, pp. 85–92.
- [37] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies—a comprehensive introduction," *Natural computing*, vol. 1, no. 1, pp. 3–52, 2002.
- [38] A. Isaksen, M. Ismail, S. J. Brams, and A. Nealen, "Catch-up: A game in which the lead alternates," *Game & Puzzle Design*, vol. 1, no. 2, 2015.
- [39] C. Browne and F. Maire, "Evolutionary game design," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 2, no. 1, pp. 1–16, 2010.

# Constrained Surprise Search for Content Generation

Daniele Gravina  
Institute of Digital Games  
University of Malta  
Msida 2080, Malta  
daniele.gravina@um.edu.mt

Antonios Liapis  
Institute of Digital Games  
University of Malta  
Msida 2080, Malta  
antonios.liapis@um.edu.mt

Georgios N. Yannakakis  
Institute of Digital Games  
University of Malta  
Msida 2080, Malta  
georgios.yannakakis@um.edu.mt

**Abstract**—In procedural content generation, it is often desirable to create artifacts which not only fulfill certain playability constraints but are also able to surprise the player with unexpected potential uses. This paper applies a divergent evolutionary search method based on surprise to the constrained problem of generating balanced and efficient sets of weapons for the *Unreal Tournament III* shooter game. The proposed constrained surprise search algorithm ensures that pairs of weapons are sufficiently balanced and effective while also rewarding unexpected uses of these weapons during game simulations with artificial agents. Results in the paper demonstrate that searching for surprise can create functionally diverse weapons which require new gameplay patterns of weapon use in the game.

## I. INTRODUCTION

Procedural content generation has been used since the 1980s in the game industry to quickly and computationally efficiently create elaborate structures such as the dungeons of *Rogue* (Toy and Wichman 1980) or the universe of *ELITE* (Acornsoft 1984). Commercially, procedural content generation (PCG) is used primarily for two reasons: (a) to cut down on development effort and time, and (b) to create unexpected, unique experiences every time the game is played, thus increasing its lifetime and replayability value. Due to the former, small teams of game developers have been able to (procedurally) create grandiose gameworlds such as those in *Minecraft* (Mojang 2011) and *No Man's Sky* (Hello Games 2016). Due to the latter, games such as *Civilization V* (Firaxis 2010) have been immensely successful in retaining a userbase engaged despite the lack of e.g. an overarching campaign.

Academic interest in PCG has often used search-based processes [1] such as evolutionary computation to create game content which optimize one or more game qualities deemed important by the designers. To a degree, such gameplay qualities are required from content in order to ensure the game being playable and fair between players (e.g. in a competitive game). On the other hand, a core motivation of commercial PCG is the element of surprise it can elicit from players. While the majority of PCG research focuses on creating one final artifact which exemplifies the desired properties of its type, there are several attempts at creating a diverse set of content using e.g. multi-objective optimization [2], multi-modal optimization [3] and novelty search [4].

This paper is inspired by earlier work on creating sets of diverse artifacts [5], and applies a recently introduced divergent search algorithm, namely *surprise search* [6], [7],

on the task of procedural content generation. In particular, the goal is generating pairs of weapons for a competitive first-person shooter game: the two weapons must be usable and balanced between them, but also exhibit surprising behavioral properties (i.e. different weapon pairs would allow different types of gameplay or strategies to emerge). Towards that end, constraints on usability and balance are satisfied via a feasible-infeasible two-population approach (FI-2pop GA) which guides infeasible content towards feasibility [8]. In the feasible population, however, the weapon pairs evolve towards surprising behaviors, i.e. behaviors that were not predicted based on the previous generations. This *constrained surprise search* algorithm is shown to create more diverse content than objective-driven search. Moreover, its behavior and performance is shown to be different than randomly assigned fitness scores applied on the feasible population.

## II. BACKGROUND

This section outlines the algorithm of *surprise search* which, in this paper, is framed within *constrained optimization* for the *procedural content generation* domain.

### A. Procedural Content Generation

Compared to the historical use of procedural content generation in games, academic interest in PCG from the perspective of artificial intelligence (AI) is relatively recent. PCG research focuses on expanding the generative algorithms, going beyond *constructive* approaches [1] which are carefully crafted scripts used in the game industry to produce a limited range of content which is however guaranteed to be playable. PCG research on the other hand has used many different sets of algorithms, often revolving around evolutionary computation and constraint satisfaction, among others. Broadly, evolutionary computation under the umbrella term *search-based PCG* [1] evolves a large population of artifacts towards a certain objective, usually pertaining to in-game quality. Constraint satisfaction, on the other hand, uses a carefully selected set of constraints to ensure that all of the generated content is playable [9].

### B. Constrained Optimization and PCG

While it would seem that constraint satisfaction and search-based PCG are incompatible in terms of design approach, there have been several attempts to integrate playability constraints to search-based PCG [10], [4]. Often, the simplest solution is

to assign a minimal fitness score and kill off the infeasible individual [3]. In highly constrained spaces, however, this is not a desirable strategy as most genotypical information is lost [11]. Indeed, if a population consists only of infeasible results then assigning a minimal fitness results in random search. Instead, constrained optimization often utilizes *penalty functions* [12] which reduce the fitness score of an infeasible individual. Designing a penalty function can become as challenging as the optimization problem itself, however, as very high penalties can kill off all infeasible results while very low penalties can lead to extraneous exploration of the infeasible search space. A more recent solution to constrained optimization is the feasible-infeasible two-population (FI-2pop) genetic algorithm [8], which evolves two separate populations towards optimizing a problem-dependent objective (in the feasible population) and minimizing the distance to feasibility (in the infeasible population). The feasible population contains only individuals which satisfy all constraints, while the infeasible population contains individuals which fail one or more constraints; feasible offspring of infeasible individuals migrate to the feasible population and vice versa. The benefit of the two-population approach is that (a) there is no competition between feasible and infeasible individuals, and (b) any search strategy can be applied to either the feasible or the infeasible population. Earlier research on game level generation has explored the use of novelty search in the feasible population or in both populations [4], in order to ensure that playable (due to the constraints) yet diverse (due to the divergent search) game levels were being produced. The current paper explores the use of surprise search [7], a recent but promising divergent search method, on the feasible population for the purposes of creating balanced but surprising weapons.

### C. Surprise search

Surprise search [6], [7] is a new algorithm for evolutionary divergent search which rewards unexpected — rather than unseen — behaviors. Surprise search uses a prediction model to construct the expected outcomes at the current stage of evolution; when evaluating the actual outcomes in the population, it rewards those which deviate from the expected [13]. This mimics a self-surprise process [14], where individuals who do not conform to the evolutionary trend are selected and ensuingly create their own trend which new individuals must again diverge from. The algorithm has been shown to outperform objective search in deceptive problems and to be more robust than novelty search in a maze navigation task [7]. Surprise search is composed by two main modules: a *predictive* model based on past behaviors and a *distance* formula to assess deviation from the expected outcomes. Surprise search uses the prediction model ( $m$ ) to create a speculative ‘current’ population, based on  $h$  previous generations; the model considers a degree of local (or global) behavioral information (expressed by  $k$ ). The predictive model is described in eq. (1); more details about  $m$ ,  $h$  and  $k$  are found in [7].

$$p = m(h, k) \quad (1)$$

The surprise score, used for selecting individual  $i$  in the current population, is based on the distance of the closest  $n$  prediction points obtained with the prediction model  $m$ :

$$s(i) = \frac{1}{n} \sum_{j=0}^n d_s(i, p_{i,j}) \quad (2)$$

where  $d_s$  is the domain-dependent measure of behavioral difference between an individual  $i$  and its expected behavior,  $p_{i,j}$  is the  $j$ -closest prediction point (expected behavior) to individual  $i$  and  $n$  is the number of prediction points considered;  $n$  is a problem-dependent parameter determined empirically.

## III. METHODOLOGY

The goal of the generative algorithms is the creation of pairs of usable and balanced weapons which exhibit surprising behavioral characteristics. The weapons are used in the commercially successful *Unreal Tournament III* (Epic Games 2007) game (UT3). Besides its commercial appeal, UT3 has well-designed game levels and AI modules which allow for simulations of game matches in order to derive behavioral properties of the weapons. Weapons in UT3 are already quite diverse, which allows the genetic algorithm to explore different sets of parameters such as bouncing bullets, grenades affected by gravity, or exploding projectiles.

### A. Representation & Genetic Operators

In the genotype, each weapon is represented by 11 parameters with different value ranges and in-game properties as shown in Table I. Since the generator evolves pairs of weapons (one per player in a deathmatch FPS game), the genotype therefore consists of 22 chromosomes, 11 for each weapon. Evolution is carried out by applying simulated binary crossover with a 60% probability, and simulated binary mutation with a 5% probability. These parameters have been chosen empirically via pre-experimentation conducted in [5]. Simulated binary crossover [15] applies a polynomial probability distribution (controlled by the maximum and minimum values of each parameter in Table I) to chromosomes; another parameter ( $\eta = 20$  as suggested in [15]) controls how much the offspring will resemble their parents. This crossover strategy ensures that the weapon of each player will be a combination of parameters of weapons used by the same player (i.e. weapons cannot be assigned to a different player from generation to generation). Simulated binary mutation performs a similar modification with a chance of 5% for each parameter in the gene. Using the same  $\eta$  value, modifications via mutation depend on the value range of each weapon parameter (e.g. low  $\eta$  values result in large mutations).

### B. Simulations

The two weapons evolved in this scenario are tested by two AI-controlled agents competing for the highest number of kills in a UT3 level. Experiments in this paper use the *Biohazard* UT3 level, which is small and thus ideal for one-versus-one matches; moreover, it consists of two separate floors which makes logging player positions easier. Each player is given

TABLE I  
PARAMETERS OF EACH WEAPON WITH THEIR CORRESPONDING VALUE RANGE AND DESCRIPTION.

Name	Value Range	Description
Rate of Fire (ROF)	[0, 4]	Number of bullets shot per second
Spread (Spr)	[0, 3]	This parameter affects the random deviation of the bullets trajectory: the higher the spread the less accurate the shooting.
ShotCost (SC)	[1, 9]	Number of bullets shot at once by the weapon.
Lifetime (L)	[0, 100]	Amount of time the bullets remain in game when shot.
Speed (Sp)	[0, 10000]	Speed of bullet when shot.
Damage (Dmg)	[0, 100]	The amount of damage that each shot deals when it hits an opponent. In case of $SC > 1$ , each bullet has $Dmg/SC$ damage per bullet.
Collision Radius (CR)	[0, 100]	Radius of the collision sphere of bullets (for hitting enemies).
Gravity (Gr)	[-250, 250]	Gravity force applied to bullets: the larger the value, the stronger the $g$ acceleration applied to the bullet. For positive values, gravity is reversed (the bullet goes upwards).
Explosive (Exp)	[0, 300]	When a bullet hits a target (opponent, object or wall), it generates an explosion with radius equal to this parameter. All players within the radius of an explosion receive splash damage (a fraction of the weapon's damage depending on distance).
Ammo (A)	[1, 999]	Maximum amount of ammunition; all ammo packs increase ammo up to this value.
Bounce (B)	[0, 1]	Boolean value that says if the projectile will bounce when it hits a wall.

one weapon and ammunition as defined in the genotype; if they pick up any weapon or ammo in the level then the ammo for their generated weapon increases by the ammo value in the genotype (i.e. players cannot pick up other weapons, including the other player's weapon). This ensures that each player tests only one weapon: these simulations allow for a comparison in terms of balance of the weapons, as well as for evaluating their effectiveness (if it kills the opponent often) and safety (if it does not result in the wielders shooting themselves). Moreover, simulations are used to create a map of the death locations of each player; these act as behavioral characteristics of the weapons and are used to assess unexpected behaviors in the surprise search algorithm. Simulations last up to a *time limit* of 1200 seconds or until a *score limit* of 20 is reached in terms of the total number of kills of the two players.

Since simulations require that AI agents use weapons of variable quality, the system provides suggestions to the AI behavior based on each weapon's parameters. For instance, if the weapon is a fast repeater (i.e. a rate of fire above 2) the AI is instructed to use it for long sequences of shots. If its bullets have a long lifetime, high speed and low spread, the AI is instructed to use it for long distance shots. Finally if the bullet has an explosive value above 50, the AI is instructed to treat it as a *splash weapon*, which relies less on accuracy.

### C. Constraints

There are certain playability requirements for the generated weapons: balance, effectiveness and safety. Each of these properties can be evaluated as a scalar value, via heuristics discussed below, based on simulations between AI controlled agents. A pair of weapons is considered playable (i.e. feasible) if each property is above a specific threshold. Moreover, for infeasible individuals the heuristics can be used to derive the distance from feasibility with regards to each constraint.

*Balance* is computed as the Shannon Entropy [16] of the kills obtained by the two agents:

$$f_b = \frac{1}{n} \sum_{j=0}^n \left( \frac{k_i}{K} \log \left( \frac{k_i}{K} \right) \right) \quad (3)$$

where  $K$  is the total number of kills obtained by the two agents in a simulation,  $k_i$  are the kills obtained by  $i$ -th bot and  $n$  is the number of players per simulation.

*Effectiveness* is calculated by dividing the total number of kills obtained in the simulation by the maximum *score limit*:

$$f_e = \frac{K}{S_{max}} \quad (4)$$

where  $S_{max}$  is the score limit ( $S_{max} = 20$  in this study) which must be attained for the level to be considered completed before the time limit expires.

*Safety* is introduced due to initial random weapons being dangerous to the wielder (due to high explosive values), and its goal is to make evaluations more robust against noise. Safety is computed in eq. (5) where the exponent is the number of suicides (i.e. deaths not scored as another player's kill):

$$f_s = 0.9^{D-K} \quad (5)$$

where  $D$  is the total number of agents' deaths in a simulation.

The feasibility constraint is satisfied if  $f_b \geq 0.9$ ;  $f_e \geq 1$  (i.e. if exactly 20 kills are scored);  $f_s \geq 0.9$  (i.e. if there's at most one suicide). The rationale for the strict thresholds for effectiveness and safety are to avoid creating sparse heatmaps of death locations (due to low effectiveness) or death locations originating from suicides (due to low safety).

### D. Constrained Surprise Search

Constrained surprise search fuses the properties of FI-2pop constrained optimization [8] with surprise search [7] evolving the feasible population. The proposed algorithm uses two populations which evolve towards different goals. The feasible population contains individuals which satisfy all constraints listed above, while the infeasible population contains individuals which have at least one of safety, balance and efficiency below the minimal threshold. The infeasible population assigns its members a fitness equal to  $f_b + f_e + f_s$ , regardless of whether some of the values of these properties are above the feasibility threshold. This favors individuals which satisfy more constraints to others which satisfy no constraints,

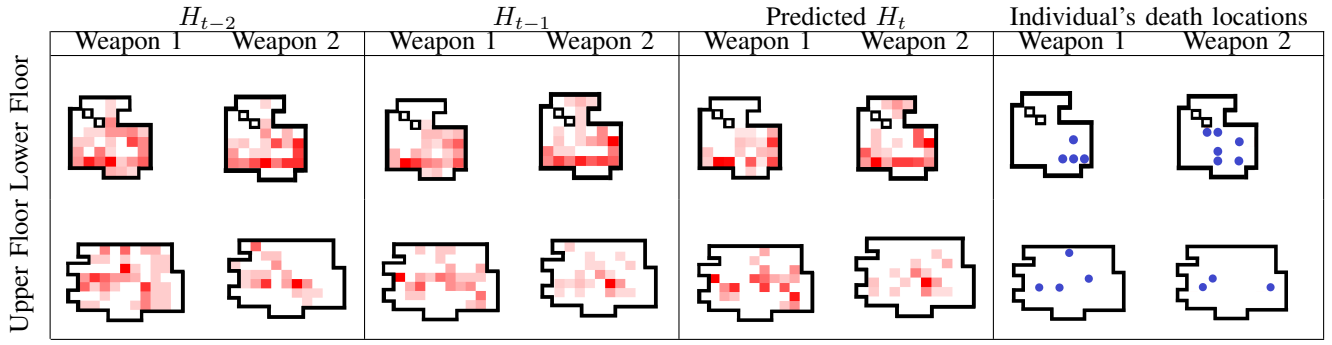


Fig. 1. An example of the prediction model of surprise search in this paper, at generation  $t$ . The first two sets of heatmaps are computed in the last two generations,  $H_{t-2}$  and  $H_{t-1}$ ; the death location density is always normalized per floor. Using linear interpolation, the difference  $H_{t-1} - H_{t-2}$  is computed and applied to  $H_{t-1}$  to derive the predicted current population's  $H_t$  truncated to  $[0, 1]$ . An individual's death locations are mapped to  $H_t$  to calculate the surprise score per eq. (6).

although some averaging artifacts may occur. Unlike traditional FI-2pop approaches, the infeasible population attempts to maximize this value, as the three properties act as objectives (with minimal value constraints).

Due to the highly constrained search space that evolution has to tackle, several steps are taken to make search in the feasible population more effective. Both populations select parents based on tournament selection (tournament size of 3), but the best individual of each population is copied as-is to the next (elitism of 1). This elitism ensures that at least one feasible individual will remain in the feasible population. Moreover, if the feasible population is smaller than the infeasible population, an *offspring boost* [4] is applied: the offspring boost forces the (larger) infeasible population to only produce offspring equal to 50% of the total population while the feasible population produces more offspring than it has parents, equal also to 50% of the total population.

In the feasible population, surprise search attempts to deviate from predicted behavioral trends of the current population. *Behavior* of a weapon is considered to be the playtraces of the player who wields it, and in particular the locations where their opponent died in this one-versus-one deathmatch game. Since the genotype contains two weapons and the *Biohazard* level consists of two floors, this creates a total of 4 heatmaps of death locations of each player. These *heatmaps* assign each death on a tile of a low-resolution grid (10 by 13 tiles per floor), incrementing the value (or heat) of that tile; example heatmaps are shown in Fig. 1. Note that heatmaps are normalized to a range of  $[0, 1]$  based on the maximum heat value of each map (i.e. per floor and per player).

Surprise search attempts to deviate, therefore, from the expected heatmaps of this generation: i.e. have death locations which are unexpected based on the current evolutionary trends. Surprise search focuses on diverging from predictions  $p$  (see eq. (1)) of the current population, calculated by observing the previous generations' behavioral changes. This paper uses only the populations of the last two generations ( $h = 2$ ; eq. (1)) to predict the current population, applying a linear interpolation ( $m$  is a linear regression model in eq. (1)). The model,  $m$ , considers the population as a whole ( $k = 1$ ; eq. (1)). In

short, when predicting the heatmaps  $H_t$  for a population at generation  $t$ , the heatmaps of the population at  $t - 2$  ( $H_{t-2}$ ) is subtracted from those of the population at  $t - 1$  ( $H_{t-1}$ ) to calculate  $\Delta H$ . The prediction of  $H_t$  is obtained by adding  $\Delta H$  to  $H_{t-1}$ , ensuring that its values fall within  $[0, 1]$  in all 4 heatmaps. Figure 1 illustrates this procedure.

In order to derive a surprise score for an individual  $i$  (which the surprise search algorithm attempts to maximize), the locations of its agents' deaths are mapped to the appropriate cell of predicted heatmaps  $H_t$  (depending on floor and player). The surprise score is calculated in eq. (6), as the complementary of the average cell values of  $H_t$  where deaths occurred in individual  $i$ . This rewards individuals which diverge from the predicted consensus of the general population.

$$s(i) = 1 - \frac{1}{D} \sum_{d \in D} H_t(d) \quad (6)$$

where  $D$  is the number of deaths of all agents in individual  $i$  and  $H_t(d)$  the cell value of  $H_t$  at the location of death  $d$ . This rewards individuals which diverge from the predicted consensus of the general population.

This evaluation of surprise is different from that of novelty used in novelty search [17], as the latter deviates from the actual population rather than its prediction. By using a prediction of the population, surprise search creates data (heatmaps in this case) which may never be attainable: diverging from such may push search in unexpected areas of the space.

#### IV. EXPERIMENTS

This paper aims to generate surprising weapons which have a modicum of balance, safety and efficiency. Towards evaluating constrained surprise search in terms of these different priorities, several tests are performed on the results of 25 independent optimization runs of constrained surprise search. The algorithm is compared with two baseline algorithms in terms of constraint satisfaction and diversity preservation; the most diverse solutions of constrained surprise search are then assessed in terms of their use by AI agents in game simulations; finally, a sample set of weapons evolved by constrained surprise search is presented in detail, showcasing how the different weapons are surprising yet balanced.



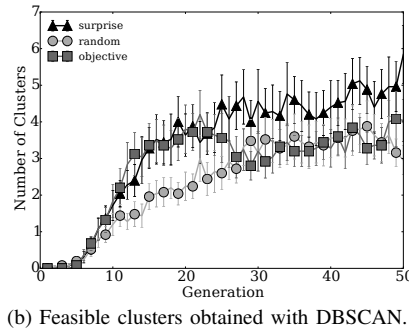
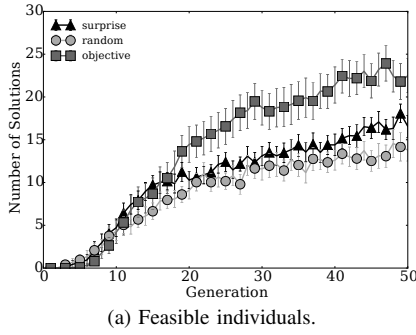


Fig. 2. Progress of two different performance metrics over the course of evolution averaged from 25 independent runs. Error bars depict standard error.

#### A. Comparison with other methods

In order to assess the performance of constrained surprise search, its outcomes and overall optimization progress will be compared to two benchmarks: (a) objective search, which attempts to optimize the sum of balance, safety and efficiency ( $f_b + f_s + f_e$ ), (b) constrained random search, which uses the FI-2pop paradigm but performs random search on the feasible population. The objective search does not need two populations, as it essentially amounts to the search in the infeasible population without minimal feasibility requirements. The constrained random search evolves the infeasible population to maximize  $f_b + f_s + f_e$  while the feasible population assigns a random fitness within  $[0, 1)$  to each of its members.

For the purposes of comparing the performance of the three algorithms, it is not straightforward which performance metrics are most appropriate for evaluating surprise or diversity. On one hand, it is relevant to evaluate how well-suited each algorithm is for constrained optimization: for that reason we use the *number of feasible individuals* as a measure of how the different search methods on the feasible space (in the case of surprise and random search) may create infeasible offspring from feasible parents. In order to evaluate diversity in the feasible population, we use the *pairwise genotypic distance of feasible individuals* to measure how different (at least in terms of weapon parameters) the genotypes are. The genotypes' values are normalized between  $[0, 1]$  via min-max normalization based on each parameter's value range in Table I. However, it should be noted that the number of feasible individuals in the population may not be sufficiently diverse, and thus a smaller set of the most diverse results would be more appropriate both for in-game use and as a performance metric. In previous work [3], this set of "solutions" was discovered via  $k$ -medoids where  $k$  was a property specified by the designer. In this paper, such solutions are obtained by DBSCAN [18] which can return a variable number of clusters (and their medoids) depending on the distribution of data. Therefore, it is possible to gauge the diversity of the population based on the *number of clusters found by DBSCAN*. DBSCAN is a density-based clustering technique [18] which groups individuals based on their nearest neighbor distance.

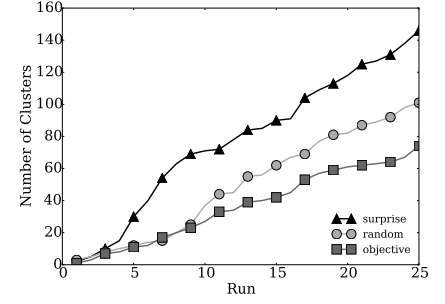


Fig. 3. Feasible clusters obtained with DBSCAN in the combined final populations of multiple evolutionary runs.

DBSCAN depends on two parameters: a distance  $\epsilon$ , which is the distance from a randomly chosen point for considering its neighbors, and the minimum number of points within  $\epsilon$  from a random point in order to be considered a cluster; in this paper  $\epsilon$  is 0.2 and the minimum number of points is set to 1.

Reported results are collected from 25 independent optimization runs per approach; evolution lasts for 50 generations and is performed on a total population size of 50 individuals. Reported significance is obtained from two-tailed Mann-Whitney U-tests at a 5% significance level.

Figure 2a shows the number of feasible individuals for each method, as evolution progresses. It is immediately obvious that at the start of evolution there are no feasible individuals, which indicates a highly constrained search space. Since all methods evolve infeasible individuals towards the same objective, it is not surprising that all methods discover the first feasible individual in approximately the same generation, i.e. generation 10 or so. More interestingly, the number of feasible individuals (in a total population of 50) keeps increasing throughout evolution as more and more infeasible individuals approach the border of feasibility. With the offspring boost, ideally the feasible individuals would be equal to the infeasible ones; however, feasible parents are more likely to create infeasible offspring than the reverse and thus feasible individuals are fewer. Objective-driven search tends to create more feasible results, while interestingly random search on the feasible population is not more destructive (in terms of feasible population size) than surprise search. Objective-driven search is expected to create more feasible individuals, primarily due to the fact that it uses a single population: therefore, feasible results are more likely to get selected (multiple times) and thus create more feasible offspring. Despite the feasible offspring boost in the FI-2pop approaches, this single population approach which continually tries to improve upon the constraints (even after all constraints are satisfied) is more efficient at creating feasible results.

DBSCAN is able to identify distinct clusters, so the number of clusters should be an indication of the population's diversity: essentially, DBSCAN plays the role of a designer choosing the most representative weapons (the clusters' medoids). The number of clusters found among feasible individuals (for a threshold of 0.2) is shown in Fig. 2b. While the

TABLE II

PERFORMANCE METRICS AT THE END OF 50 GENERATIONS. RESULTS ARE AVERAGED FROM 25 INDEPENDENT RUNS, WITH THE STANDARD ERROR SHOWN IN PARENTHESES. DIVERSITY REFERS TO THE AVERAGE PAIRWISE GENETIC DISTANCE.

	Surprise	Objective	Random
Feasible Individuals	16.6 (1.28)	22.5 (1.98)	13.8 (1.29)
Feasible Clusters	5.84 (0.78)	4.04 (0.58)	2.96 (0.40)
Individuals' Diversity	0.29 (0.03)	0.26 (0.02)	0.24 (0.02)
Medoids' Diversity	0.33 (0.04)	0.28 (0.03)	0.26 (0.04)

objective-based approach creates more feasible individuals, it is obvious that these individuals are genotypically similar leading to fewer clusters than the smaller feasible population of constrained surprise search. As the first feasible individuals appear in the population, both objective-driven search and surprise search start with a diverse population (and thus many feasible clusters). However, as objective-based selection prioritizes feasible individuals almost exclusively (i.e. when the number of feasible individuals increases after generation 20), objective-driven search converges to a few promising areas of the search space resulting in a drop in the number of clusters. By comparison, the behavioral surprise prioritized by surprise search manages to better preserve the genetic diversity of the initial feasible individuals. It should be noted that in the 25 runs performed for the reported results, there is a large deviation, on average, between the number of clusters for every approach, as can be gleaned from Fig. 2b.

Table II shows the final scores of the different performance metrics at the end of 50 generations. While constrained random search is able to maintain a sufficiently large feasible population, the number of distinct clusters found by DBSCAN is significantly lower ( $p < 0.05$ ) than those of constrained surprise search. Objective search creates significantly more feasible individuals on average than constrained surprise search ( $p < 0.05$ ), but they are not as diverse (based on the number of clusters); due to large deviations in the number of clusters, significance can not be established. As additional metrics, the average pairwise genotypic distance of all feasible individuals and of the cluster medoids is compared. Constrained surprise search tends to create more genotypically different medoids than both objective search and constrained random search.

As another measure of diversity, DBSCAN is applied on the final populations of multiple evolutionary runs and the number of discovered clusters is plotted in Fig. 3. Surprise search seems robust at finding clusters in the combined populations, and thus does not converge to the same local optima in every run. With few runs, the differences between random and objective search are minimal while surprise search finds far more clusters; the order of evolutionary runs being combined could have a minor effect in this. DBSCAN finds 146 feasible clusters in 415 feasible individuals collected from 25 runs of surprise search, versus 101 clusters in 563 feasible individuals of objective search. Indicatively, Fig. 3 shows that collecting 40 distinct weapon pairs (i.e. medoids) requires 6 runs of surprise search versus 11 and 14 runs of objective and random

TABLE III

GAMEPLAY METRICS OF ALL CLUSTER MEDOIDS OF ALL 25 INDEPENDENT RUNS OF CONSTRAINED SURPRISE SEARCH. RESULTS ARE AVERAGED FROM 10 SIMULATIONS, WITH DEVIATION BETWEEN MEDOIDS IN PARENTHESES.

	Both floors	1st Floor	2nd Floor
Total Kills	15.14 (0.27)	10.81 (0.18)	4.29 (0.15)
Kills 1st	7.83 (0.21)	4.32 (0.22)	3.51 (0.08)
Kills 2nd	7.3 (0.21)	5.11 (0.18)	2.19 (0.08)
Balance	0.87 (0.009)	0.84 (0.011)	0.71 (0.016)
Entropy 1st	0.71 (0.002)	0.63 (0.001)	0.74 (0.003)
Entropy 2nd	0.72 (0.003)	0.64 (0.002)	0.76 (0.004)

search respectively. Combined with a slightly higher medoid diversity, surprise search seems preferable for discovering more diverse weapons at a lower computational cost.

### B. Gameplay Qualities of Weapons

In order to assess a modicum of the gameplay uses of the generated weapons, all cluster medoids discovered in the 25 surprise search runs were tested in 10 simulations each. Table III shows certain gameplay metrics of these weapons, averaged from 10 simulations: '1st' and '2nd' refers to the first and second player, while 'entropy' refers to Shannon's Entropy of the death locations' heatmaps on both floors.

From Table III, it is clear that the number of total kills are on average below the minimal playability thresholds set in Section III-C; this points to a very noisy simulation-based evaluation. During optimization, it seems that a single simulation can decide that a weapon pair is playable in one generation and reject the same pair in the next. Based on the number of kills for each player, the calculated balance for the weapon pair is 0.87 on average, which is also slightly below the minimal threshold for  $f_b$ . Suicides on the other hand remain low at an average of 0.44 (standard error of 0.04), and therefore the safety constraint is always satisfied.

An interesting insight from Table III is the gameplay difference between the two floors: significantly more kills occur in the 1st floor for both weapons ( $p < 0.05$ ), which should not be surprising since the second floor is not as accessible and offers a better vantage point to fire at enemies below. On the other hand, deaths in the second floor are significantly more dispersed spatially (based on Shannon's entropy) than those in the first floor ( $p < 0.05$ ): this is also obvious from  $H_{t-1}$  and  $H_{t-2}$  of Fig. 1. This can be traced to the fact that the second floor is in theory larger; moreover the second floor includes narrow bridges which partition the space and therefore players tend to die on opposite ends of that floor. The differences in both number of deaths and entropy of death locations could affect the performance of surprise search, however, which currently normalizes each heatmap individually; if the first floor has far more deaths than the second floor, treating them equally in terms of surprise places unnecessary impact on the second floor. A potential solution in future experiments could be to normalize heatmaps based on the total number of deaths rather than the number of deaths per floor and per player.

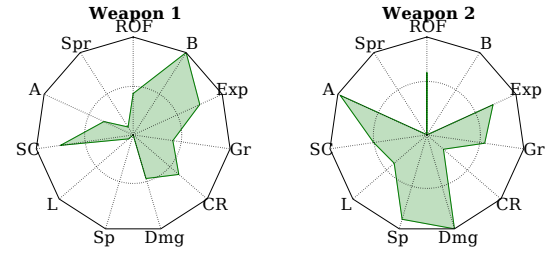
### C. Sample Weapons

In order to discuss a sample of the generated weapons in more detail, a weapon pair was chosen agnostically among the DBSCAN cluster medoids for surprise search; there is no assumption that this weapon pair exemplifies all generated content. For the sake of screenshots, weapons use the shock rifle weapon model and bullet effects in UT3.

Figure 4a shows a genotype collected after 50 generations of constrained surprise search. The first weapon fires very concentrated projectiles (maximum shot cost and very low spread) with no gravity and very low speed; the second weapon shoots very fast projectiles, with no spread and high damage. Trying these two weapons as a player, one quickly realizes that the first weapon creates ‘mines’ around the map (see Figure 4b): its bullets are extremely slow, with a large blast area (explosive, high collision radius) and they can also bounce on walls or the level’s floor. Moreover, these ‘mines’ are fired in clusters (high shot cost) as seen in Fig. 4b, thus costing a lot of ammo (of which the weapon has little). This weapon seems over-powered, allowing its wielder to control the map; however as the bullets do not have a long lifetime, they are effective only if the opponent is nearby and running towards the wielder. Meanwhile, the second weapon is very similar to a rifle: high-damage fast bullets which shoot straight (trivial gravity effects) with a very low collision radius, thus requiring precise aiming. Unlike traditional rifles, however, the weapon’s bullets have some explosive qualities. Obviously, a match-up between these two weapons requires a very different strategy from each player: the first weapon requires its wielder to move around the level, laying ‘mines’ in chokepoints when the other player is nearby. Meanwhile, the player with the second weapon does not need to move as much (also in order to avoid any mines) as she can fire her high-speed, precise and lethal bullets from a remote location.

### V. DISCUSSION

The primary goal of this paper was to discover balanced and efficient, yet surprising weapons via constrained surprise search. Results indicate that constrained surprise search tends to evolve genotypically more diverse pairs of weapons, which have unexpected in-game uses. The FI-2pop paradigm also allows this method to discover feasible individuals quickly and consistently; preliminary experiments with surprise search or random search on a single population without constraints failed to find feasible results at any point of their evolution during 50 generations. Comparing constrained surprise search with constrained random search shows a (statistically) significant improvement of the former in terms of number of clusters: this indicates that surprise search is substantially different in terms of both process and results than random search. On the other hand, performing objective search on a single population finds significantly more feasible individuals; due to the selection pressure towards feasible individuals, however, much of the genetic diversity is eventually lost. While objective search has more feasible individuals, those are not necessarily as diverse or interesting as in surprise search.



(a) Sample weapon pair evolved via constrained surprise search.



(b) Weapon 1, evolved via constrained surprise search.

Fig. 4. Example of weapons evolved via constrained surprise search

There is an obvious limitation to the generalizability of the results reported in this paper, for several reasons: (a) the high deviation in the performance metrics from one run to the next makes assessing significance problematic, (b) there are many parameters which could be fine-tuned to improve both the algorithms and the assessment method. Regarding the deviation found in reported values, the need for simulations of full games (lasting up to 20 minutes) to assess each individual prevents extensive experiments for parameter tuning or for more runs to assess significance. Future work should explore how surprise search performs with different behavioral characteristics, as well as compare its performance against more methods including two-population objective search or two-population novelty search [4]. Finally, the choice of  $\epsilon$  in DBSCAN also affects the results and conclusions; with a much lower  $\epsilon$  the more numerous individuals of objective search create more clusters (e.g. with one individual per cluster), and with a much higher  $\epsilon$  all methods create a single cluster.

Surprise search in this particular problem predicts the behavior of the population as a whole, based on the behavior of the previous two populations. In the general predictive model of eq. (1), the current approach uses the simplest predictive model  $m$  (a linear model), the shortest history ( $h = 2$ ) and there is no locality as the model aggregates the entire population ( $k = 1$ ); for this reason the distance calculation in eq. (2) considers one neighbor ( $n = 1$ ). Obviously there is a broad range of parameters to explore in order to improve the performance of surprise search, such as using a non-linear regression model or including a form of archive as in novelty search [17]. Another possible improvement could be choosing another behavioral characteristic to deviate from:

currently the system considers a “heatmap” of death locations; this heatmap is relatively sparse, also considering that the level has two floors. In many cases the differences between two such heatmaps is circumstantial, also due to the high stochasticity of combat; this was mitigated by using the lowest locality for surprise search and aggregating a heatmap for the entire population. Other behavioral characteristics could be considered, such as players’ movements in the level or scalar gameplay values such as the entropy and ratio of kills in each floor or the distance between players at the time of death.

Finally, it should be noted that simulations with AI opponents are a necessity due to the numerous matches which need to be played per generation during evolution. The enemy behavior in UT3 is quite well-designed and competitive (at least for novice players), unlike many open-source games such as *Cube 2* used in other experiments [19]. However, the AI in UT3 is designed for the weapons included in the game which have specific parameters and uses; with generated weapons, unexpected combinations of weapon parameters such as high spread on a sniper rifle may result in less than ideal agent performance. Currently, this is somewhat mitigated when initializing simulations by instructing the AI that certain weapons with certain properties should be treated as UT3 weapons (e.g. high lifetime weapons use the sniper rifle AI module). However, completely unique weapons which do not have any AI module may be impossible to use: for instance, a mine weapon requires a very different navigational strategy, going through chokepoints and leaving mines. While the AI could be improved, it is perhaps more interesting to test the final weapons generated by each approach in a user survey with competing expert players. Observing emergent gameplay strategies, and how the meta-game is affected by unexpected uses of these weapons can offer a better insight of the usability and balance of the generated weapon pairs.

## VI. CONCLUSION

This paper introduced a constrained form of surprise search and applied it on a procedural content generation problem. The first goal of the generator is to create pairs of weapons for *Unreal Tournament III*, which have a balanced and efficient performance when played in simulated matches with AI agents. Towards that end, a feasible-infeasible two-population genetic algorithm was employed to maximize balance, efficiency and safety on an infeasible population until those values were above a required threshold. The second goal of the generator is to create pairs of weapons which have surprising properties and can result in interesting, unconventional gameplay. For this purpose, surprise search is applied on the feasible population, attempting to deviate from the expected behaviors of the AI agents (i.e. their death locations) which were predicted from past generations. Results in the paper show that the feasible-infeasible approach is able to find feasible individuals quickly and reliably, and that surprise search tends to create more diverse (if not always more) content. Since the reported experiment is small-scale, there is a broad range of future directions for improving surprise search and its parameters, testing it

against more algorithms, and exploring other behavioral or gameplay characteristics other than death locations.

## ACKNOWLEDGMENT

This work has been supported, in part, by the FP7 Marie Curie CIG project AutoGameDesign (project no: 630665) and the Horizon 2020 project CrossCult (project no: 693150).

## REFERENCES

- [1] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, “Search-Based Procedural Content Generation: A Taxonomy and Survey,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 172–186, Sept 2011.
- [2] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelbck, G. N. Yannakakis, and C. Grappiolo, “Controllable procedural map generation via multiobjective evolution,” *Genetic Programming and Evolvable Machines*, 2013.
- [3] M. Preuss, A. Liapis, and J. Togelius, “Searching for good and diverse game levels,” in *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, 2014.
- [4] A. Liapis, G. N. Yannakakis, and J. Togelius, “Constrained novelty search: A study on game content generation,” *Evolutionary Computation*, vol. 23, no. 1, pp. 101–129, 2015.
- [5] D. Gravina and D. Loiacono, “Procedural weapons generation for Unreal Tournament III,” in *Proceedings of the IEEE Conference on Games Entertainment Media*, 2015.
- [6] G. N. Yannakakis and A. Liapis, “Searching for surprise,” in *Proceedings of the International Conference on Computational Creativity*, 2016.
- [7] D. Gravina, A. Liapis, and G. N. Yannakakis, “Surprise search: Beyond objectives and novelty,” in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2016.
- [8] S. O. Kimbrough, G. J. Koehler, M. Lu, and D. H. Wood, “On a feasible-infeasible two-population (fi-2pop) genetic algorithm for constrained optimization: Distance tracing and no free lunch,” *European Journal of Operational Research*, vol. 190, no. 2, pp. 310–327, 2008.
- [9] A. M. Smith, E. Butler, and Z. Popović, “Quantifying over play: Constraining undesirable solutions in puzzle design,” in *Proceedings of ACM Conference on Foundations of Digital Games*, 2013.
- [10] N. Sorenson, P. Pasquier, and S. DiPaola, “A generic approach to challenge modeling for the procedural creation of video game levels,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, 2011.
- [11] Z. Michalewicz, “Do not kill unfeasible individuals,” in *Proceedings of the Fourth Intelligent Information Systems Workshop*, 1995, pp. 110–123.
- [12] C. A. Coello Coello, “Constraint-handling techniques used with evolutionary algorithms,” in *Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation*. ACM, 2010, pp. 2603–2624.
- [13] K. Grace, M. L. Maher, D. Fisher, and K. Brady, “Modeling expectation for evaluating surprise in design creativity,” in *Design Computing and Cognition*, 2014.
- [14] K. Grace and M. L. Maher, “What to expect when you’re expecting: The role of unexpectedness in computationally evaluating creativity,” in *Proceedings of the International Conference on Computational Creativity*, 2014.
- [15] K. Deb and R. B. Agrawal, “Simulated binary crossover for continuous search space,” vol. 9, pp. 115–148, 1995.
- [16] C. E. Shannon, “A mathematical theory of communication,” *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 5, no. 1, pp. 3–55, 2001.
- [17] J. Lehman and K. O. Stanley, “Abandoning objectives: Evolution through the search for novelty alone,” *Evolutionary computation*, vol. 19, no. 2, 2011.
- [18] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise,” in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 1996.
- [19] P. L. Lanzi, D. Loiacono, and R. Stucchi, “Evolving maps for match balancing in first person shooters,” in *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2014.

# Using Opponent Models to Train Inexperienced Synthetic Agents in Social Environments

Chairi Kiourt

School of Science and Technology  
Hellenic Open University  
Patras, Greece  
chairik@eap.gr

Dimitris Kalles

School of Science and Technology  
Hellenic Open University  
Patras, Greece  
kalles@eap.g

**Abstract**— This paper investigates the learning progress of inexperienced agents in competitive game playing social environments. We aim to determine the effect of a knowledgeable opponent on a novice learner. For that purpose, we used synthetic agents whose playing behaviors were developed through diverse reinforcement learning set-ups, such as exploitation-vs-exploration trade-off, learning backup and speed of learning, as opponents, and a self-trained agent. The paper concludes by highlighting the effect of diverse knowledgeable synthetic agents in the learning trajectory of an inexperienced agent in competitive multi-agent environments.

**Keywords**—component; Multi-agent systems; Opponent based learning; synthetic agents; Social learning

## I. INTRODUCTION

Organizational Learning (OL) has become an increasingly exiting area for studying simulated learning agents [1] [2] [3] [4] [5]. Computer social simulation is a way of modeling and understanding social and economic processes [4], while social learning is concerned with how humans learn from observations of others' actions [2] [3] [6] [7] [8]. Typically, when many agents interact with each one in an environment, they constitute Multi-Agent Systems (MASs) and Social Organizations (SOs), which attempt to simulate traits of human behavior [4] [9]. Social simulation involves artificial agents with different characteristics (synthetic agents), which interact with other agents, possibly employing a mix of co-operative and competitive attitudes, towards the investigation of social learning phenomena [7] [8] [10].

Social Learning (SL) is important in diverse scientific areas, such as sociology and economics, as well as artificial intelligence (AI) mechanisms for game playing to study complex systems and learning mechanisms [2] [4] [7] [10]. For a game agent, the social environment is composed of all relevant entities, such as rules, pay offs and penalties, and all other (competing or co-operating) agents [2] [7] [8] [10]. Learning in a game is said to occur when an agent changes a strategy or a tactic in response to new information, thus mimicking human playing behavior [2] [9] [7] [8] [11]. Caballero *et al.* [12] simulated many experiments on Multi-Agent-Based Social Simulation platforms to present an autonomous social world from which agents can cre-

ate social strategies and behaviors, while Marivate [7] demonstrated that the efficiency of an agent, which has played in a social environment, is better than that of a self-playing agent. Kiourt and Kalles [8] developed social environments by employing a zero-sum game in tournaments, comparing the progress of socially trained agents against self-trained ones, and highlighting the advantages of socially trained agents. They have also studied the interactions between agents in the same group as a means to improve the group's representative in subsequent inter-group social tournaments [10].

For that purpose, we used synthetic agents whose playing behaviors were developed through diverse reinforcement learning set-ups, such as exploitation-vs-exploration trade-off, learning backup and speed of learning, as opponents, and a self-trained agent.

The main motivation of this work focuses on the investigation of how agents learn by exploiting their opponents' models in competitive social environment. We study the learning evolution of inexperienced agents against opponents with diverse playing characteristics. The main contribution of this study is to highlight that equally inexperienced agents diverge pretty fast with respect to their playing performance, depending on their set-up on how to learn. We present tournaments with different sequences of opponents to determine the best learning sequence for inexperienced agents and we provide experimental evidence that self-trained agents have limited potential as trainers with respect to their opponents. We frame these findings in the context of a generic research question on how to select the best opponent to learn from in a social environment.

The rest of this paper is structured in three sections. The next section provides a brief background knowledge of game-playing social learning environment aspects and the synthetic agent characters using a home-grown board game as a proof of concept. The third section describes the experimentation set up based on the synthetic agents. The forth section discusses the experimental results. Finally, the last concludes the paper with some important key points and some future directions.

## II. A BRIEF BACKGROUND DESCRIPTION

The RLGame [13] is played by two players on an  $n \times n$  square board. Two  $a \times a$  square bases are on opposite board corners;

these are initially populated by  $\beta$  pawns for each player, with the white player starting off the lower left base and the black player starting off the upper right one. The goal is to move a pawn into the opponent's base or to force all opponent pawns out of the board.

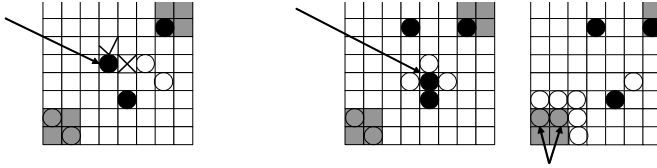


Fig. 1. Example of game rules application

The implementation of the RLGame rules is depicted in Fig. 1. In the leftmost board in Fig. 1, the pawn indicated by the arrow demonstrates a legal (“tick”) and an illegal (“cross”) move, the illegal move being due to the rule that does not allow decreasing the distance from the home (black) base. The rightmost boards demonstrate the loss of pawns, with arrows showing pawn casualties. A “trapped” pawn automatically draws away from the game, either in the middle of the board or when there is no free square next to its base of the white pawn.

The learning mechanism used (Fig. 2) is reinforcement learning inspired and is based on approximating a value function with a neural network [3] [5], with similar techniques already documented [14]. Each autonomous (back propagation) neural network [15] [16] is trained after each move. The board positions for the next possible move are used as input-layer nodes. The hidden layer consists of half as many hidden nodes. A single node in the output layer denotes the extent of the expectation to win when one starts from a specific game-board configuration and then makes a specific move. After each move the values of the neural network are updated through the temporal difference learning method, which is a combination of Monte Carlo and dynamic programming [15] [16]. As a result, collective training is accomplished by putting an agent against other agents so that knowledge (experience) is accumulated.

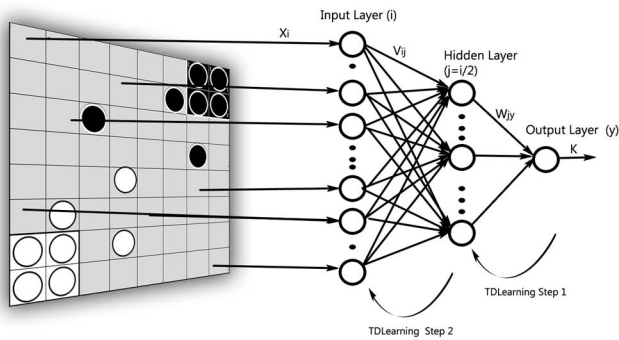


Fig. 2. Learning mechanism of RLGame.

The agent's goal is to learn an optimal strategy that will maximize the expected sum of rewards within a specific amount of time, determining which action should be taken next, given the current state of the environment. The strategy to select between moves is  $\epsilon$ -Greedy ( $\epsilon$ ), with  $\epsilon$  denoting the probability to

select the best move (exploitation), according to present knowledge, and  $1-\epsilon$  denoting a random move (exploration) [1]. The learning mechanism is associated with two additional learning parameters, *Gamma* ( $\gamma$ ) and *Lambda* ( $\lambda$ ). A risky or a conservative agent behavior is determined by the  $\gamma$  parameter, which specifies the learning strategy of the agent and determines the values of future payoffs, with values in  $[0,1]$ ; effectively, large values are associated with long-term strategies. The speed and quality of agent learning is associated with  $\lambda$ , which is the learning rate of the neural network, also in  $[0,1]$ . Small values of  $\lambda$  can result in slow, smooth learning; large values could lead to accelerated, unstable learning. These properties are what we, henceforth, term as “characteristic values” for the playing agents.

RLGame was transformed into a tool for studying MAS via its tournament version, RLGTournament [8] [17] [18] [10] [19], implementing a Round Robin scheme to pair participants against each other. RLGTournament fits the description both of an autonomous organization and of a social environment, as defined by Ferber [3]. Depending on the number of the agents, social categories can be split into sub-categories of: micro-social environment, environment composed of agent groups and global societies, which are the next level of the cooperation and competition extremes of the social organizations [3] [5].

### III. THE EXPERIMENTAL SETTING

We now report on a series of experiments to simulate and analyze how an inexperienced agent with a *good* set up of learning mechanism, learns based on different opponents' models [10]; to maximize its potential for performance improvement we endow it with the learning characteristics of the top performer of a previous tournament [10] (which is why we refer to it as “*good*”). Using the learning set-up of this top performer, we generated four clones, with the same parameters but without experience, one for each sequence of the TABLE I; in every sequence we name this agent as *Plr3\** and we subject each clone to a test against each of the four different opponents' models of TABLE I. Each experiment is composed with different sequential opponents with various characteristics, which will determine the best learning evolution and progress of an inexperienced agent with good learning mechanism set up (which is starting to compete based on zero experience). The opponent's and the testing agent's configurations and the experiment setups are shown in TABLE I.

We chose three agents, one from each class (Good Playing - GP, Moderate Playing - MB, Bad playing - BP) [10], as opponents, and we also added one self-trained agent. A self-trained agent's experience is based on playing against, and learning with, an opponent with an identical configuration. The self-trained agent has played the same number of games as the socially-trained ones [10], about 12,500 games, using default values based on Sutton and Barto [16].



TABLE I. DIFFERENT SEQUENTIAL OPPONENTS BASED EXPERIMENTS

Matches Sequences	1st				2nd				3rd				4th			
	Plr3*				Plr3*				Plr3*				Plr3*			
1st	Plr8				Plr76				Plr20				PlrS			
2nd	Plr76				Plr20				PlrS				Plr8			
3rd	Plr20				PlrS				Plr8				Plr76			
4th	PlrS				Plr8				Plr76				Plr20			

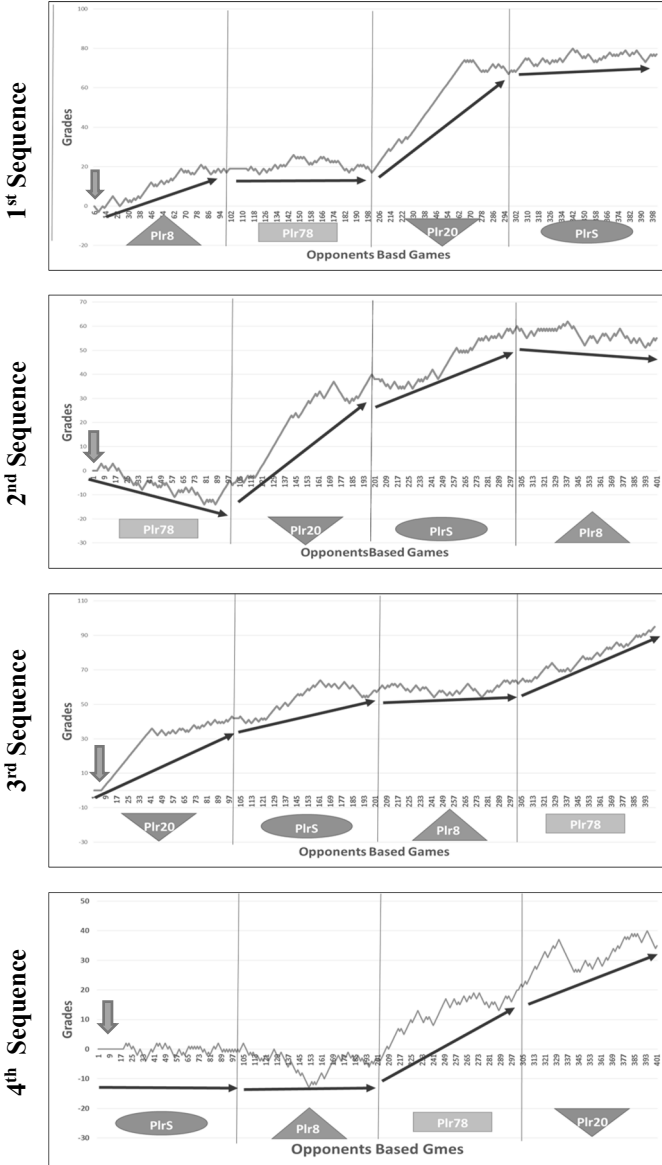


Fig. 3. Plr3\* progress and evolution analysis based on the sequential experiments

The opponents based model experiments are shown in TABLE I, where the configurations of the four different experiments are also presented. In each experiment, *Plr3\** faces the

same agents, but with a different order, for 100 games per match. Basically, the sequences of TABLE I roughly correspond to experimental scenarios; for example, in the first sequence, *Plr3\** is initially tested against a good playing agent (*Plr8*), then against a moderate playing agent (*Plr76*), then against a bad playing agent (*Plr20*) and, finally, against a self-trained agent.

The graphs in Fig. 3 demonstrate the progress and the evolution of *Plr3\** based on the experimental sequences of TABLE I. We show four graphs for each sequence. The continuous line of the graphs (y axis) shows the cumulative Grades of *Plr3\**, while the x axis shown the number of the games as separated into match sequences. Grades, is the sum of games won by each agent during the tournament (simply, a crude agent rating tool). Each win contributes +1 while each loss contributes -1 to the grade. The arrows below the straight lines shows how *Plr3\** performs relatively to each opponent. At the bottom of each sequence graph we show the opponents and their classes (a triangle with the top looking up, denotes the good playing opponent, a rectangle denotes the moderate playing opponent, a reverse triangle denotes the bad playing opponent and an oval denotes the self-trained opponent).

A summary view of how *Plr3\** wins and loses per match is presented and analyzed in TABLE II. *Plr3\** won most of the games in nearly all of the matches; by inspecting the average number of wins we confirm it is the best agent. *Plr3\** faced more difficulties against agents with similar configurations to its coming from good playing classes but won more than 70% of the games against bad playing agents. Quite as importantly, when *Plr3\** faced self-trained agents, it performed consistently better, suggesting that self-training experience is not powerful even against non-experienced agents.

TABLE II. PLR3\* GAME WINS PER MATCH

Matches Sequences	Wins per Game				AVG Wins
	1st	2nd	3rd	4th	
1st	60	55	76	54	61%
2nd	49	72	62	48	58%
3rd	74	60	54	67	64%
4th	59	49	63	58	57%

#### IV. A BRIEF DISCUSSION OF KEY FINDINGS

We now highlight and discuss some key points:

- In all four different experiments, *Plr3\** concludes some of its games with draw, at the beginning of the session, until it develops some experience. Draws are due to a game being interrupted and finished at an earlier stage, with the number of moves limited to 128, to avoid wasting time and knowledge [10]. Such draw-initiated sessions are shown by the blue arrows in the graphs of Fig. 3.
- *Plr3\** demonstrates the most stable progress-evolution when it starts against low level opponents and, thereafter, faces higher-level opponents, as shown in the

third sequence of Fig. 3, with an overall win rate of about 64%.

- *Plr3\** performs very well against low level opponents.
- In all matches against the self-trained opponent, *Plr3\** reports a win rate of about 60%, which is notable improvement compared to the less than 50% scores reported for the games in the 2<sup>nd</sup>, 3<sup>rd</sup> and the 4<sup>th</sup> experiments against the good playing agent, *Plr8*. As this behavior is also confirmed in last three sequences/experiments, it suggests that the good performance of the *Plr3\** against the self-trained agent did not enhance its experience, since in the next matches against the good playing agent, *Plr8*, *Plr3\** did not perform well.

The overall view of these experiments shows that the best progress and stable evolution of a good playing agent arises with opponents of increasing capacity, which can be interpreted as “it takes a step at a time to build useful experience”.

## V. CONCLUSIONS AND DIRECTIONS FOR FUTURE WORK

We have confirmed that a socially trained agent, with an initially strong learning mechanism [10], performs better against self-trained agents, even if the self-trained agent has quite some experience (in our case, *PlrS* has played about 12,000 iterated games before participating in the experiment described in this paper). We have also demonstrated that an opponent is an important key point on an agent’s learning progress in a social environment. We have observed that an agent with a good set up of its learning mechanism progresses better when facing less favorably set-up agents in its first games (as the third sequence of Fig. 3 suggested) and that such first games between an inexperienced agent and low-level opponents can be quite tedious and long.

The above experimental results suggest that if an agent could be equipped with a mechanism to choose an opponent from a group of diverse synthetic agents, to improve its chances of better and faster learning, it should attempt to locate opponents with initially similar learning characteristics and then attempt to gradually move towards opponents with stronger learning set-ups, as it gathers experience.

In addition to the above, we plan to study the opponent selection problem under constraints of limited information about tournament participants. We also plan to investigate how agents may decide to select their opponents in a limited number of games so as to not waste their effort quotas and to improve their (individual and collective) learning (and playing) behavior. This, we also plan to couple with an investigation of the dynamic manipulation of the characteristic values, which could produce more powerful agents, with the ability to dynamically match their effort to their opponents.

## REFERENCES

- [1] J. G. March, "Exploration and Exploitation in Organizational Learning," *Organization Science*, vol. 2, no. 1, pp. 71-87, 1991.
- [2] J. Ferber, *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*.: Addison-Wesley, 1999.
- [3] J. Ferber, O. Gutknecht, and F. Michel,.: Springer Berlin Heidelberg, 2003, pp. 214 - 230.
- [4] N. Gilbert and K. G. Troitzsch, *Simulation for the Social Scientist*.: Open University Press, 2nd ed, 2005.
- [5] Y. Shoham and K. B. Leyton, *Multiagent Systems: Algorithmic, Game-Theoretic and Logical Foundations*. New York: Cambridge University Press, 2009.
- [6] Y. Marom, G. Maistros, and G. Hayes, "Experiments with a Social Learning Model," *Adaptive Behavior*, pp. 209-240, 2001.
- [7] V. N. Marivate, "Social Learning Methods in Board Game Agents," in *IEEE Symposium Computational Intelligence and Games*, Perth, Australia, 2008, pp. 323-328.
- [8] C. Kiourt and D. Kalles, "Social Reinforcement Learning in Game Playing," in *IEEE International Conference on Tools with Artificial Intelligence*, Athens, Greece, 2012, pp. 322-326.
- [9] M. Lopes, F. S. Melo, B. Kenward, and J. Santos-Victor, "A Computational Model of Social-Learning Mechanisms," *Adaptive Behavior*, pp. 467-483, 2009.
- [10] C. Kiourt and D. Kalles, "Learning in Multi Agent Social Environments with Opponent Models," *Multi-Agent Systems and Agreement Technologies (13th European Conference, EUMAS 2015, and Third International Conference, AT 2015)*, pp. 137-144, 2016.
- [11] B. Al-Khateeb and G. Kendall, "Introducing a Round Robin Tournament into Evolutionary In-dividual and Social Learning Checkers,," in *Developments in E-systems Engineering (DeSE)*, Dubai, United Arab Emirates, 2011, pp. 294 - 299.
- [12] A. Caballero, J. Botia, and A. Gomez-Skarmeta, "Using cognitive agents in social simulations," *Engineering Applications of Artificial Intelligence*, vol. 24, no. 7, pp. 1098-1109, 2011.
- [13] D. Kalles and P. Kanellopoulos, "Verifying Game Design and Playing Strategies using Reinforcement Learning," , Las Vegas, 2001.
- [14] G. Tesauro, "Temporal Difference Learning and TD-Gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 56-68, 1995.
- [15] R. Sutton, "Learning to Predict by the Methods of Temporal Differences," *Machine Learning*, vol. 3, no. 1, pp. 9-44, 1988.
- [16] R. Sutton and A. Barto, *Reinforcement Learning-An Introduction*.: MIT Press, Cambridge, MA, 1998.
- [17] C. Kiourt and D. Kalles, "Building A Social Multi-Agent System Simulation Management Toolbox," in *6th Balkan Conference in Informatics*, Thessaloniki, Greece, 2013, pp. 66-70.
- [18] C. Kiourt and D. Kalles, "A Platform for Large-Scale Game-Playing Multi-Agent Systems on a High Performance Computing Infrastructure," *International Journal of Multiagent and Grid Systems*, vol. 12, no. 1, pp. 35-54, 2016.
- [19] C. Kiourt, G. Pavlidis, and D. Kalles, "Human Rating Methods on Multi-Agent Systems," *Multi-Agent Systems and Agreement Technologies (13th European Conference, EUMAS 2015, and Third International Conference, AT 2015)*, pp. 129-136, 2016.
- [20] D. Kalles and P. Kanellopoulos, "On Verifying Game Design and Playing Strategies using Reinforcement Learning," in *Proceedings of ACM Symposium on Applied Computing, special track on Artificial Intelligence and Computation Logic*, Las Vegas, 2001.

# Procedural Generation of Complex Stable Structures for Angry Birds Levels

Matthew Stephenson

Research School of Computer Science  
Australian National University  
Canberra, Australia  
matthew.stephenson@anu.edu.au

Jochen Renz

Research School of Computer Science  
Australian National University  
Canberra, Australia  
jochen.renz@anu.edu.au

**Abstract**—This paper presents a procedural content generation algorithm for the physics-based puzzle game Angry Birds. The proposed algorithm creates complex stable structures using a variety of 2D objects. These are generated without the aid of pre-defined substructures or composite elements. The structures created are evaluated based on a fitness function which considers several important structural aspects. The results of this analysis in turn affects the likelihood of particular objects being chosen in future generations. Experiments were conducted on the generated structures in order to evaluate the algorithm's expressivity. The results show that the proposed method can generate a wide variety of 2D structures with different attributes and sizes.

## I. INTRODUCTION

Procedural content generation (PCG) is a major area of investigation within the video game industry [1]. It is typically defined as the automatic creation of aspects of a game which affect gameplay other than non-player characters (NPCs) and the game engine [2]. PCG is commonly used to create new unique experiences for players without the need to design every possibility manually. This can dramatically cut a game's development time, as well as increasing available content and reducing memory consumption [3]. PCG can also be used to learn about the player's abilities and adapt the game's content accordingly [4].

Previous research has investigated the use of PCG for many different types of game content, including vehicles [5], weapons [6] and rulesets [7]. Level generation, or the generation of certain level aspects, is one of the most popular uses of PCG and has been implemented in many different game types. These include real-time strategy games [8], role-playing games [9], platform games [10], racing games [11] and arcade games [12].

Physics-based puzzle games such as Angry Birds, Bad Piggies, Crayon Physics and World of Goo have increased in popularity in recent years and provide many interesting challenges for PCG. However, as far as we can tell, very little research has been done on this particular area of PCG. A small collection of studies have explored PCG for the physics-based game Cut the Rope [13], [14], as well as the popular mobile game Angry Birds [15], [16], [17].

Physics-based games make PCG more difficult for a variety of reasons. Firstly, there are typically many constraints that dictate the types of content that can be created. Any PCG

algorithm must be aware of the physical limitations of its environment and create content that functions as expected, e.g. a procedurally generated car must be able to drive and steer. Secondly, the state and action spaces are typically very large. This makes the task of determining if a procedurally generated level can be completed extremely difficult, especially for increasingly complex levels and content. Lastly, the variety of content that the algorithm can create must not be significantly reduced by any constraints imposed. The main appeal of PCG is that a large and diverse range of content can be created. Designing algorithms with restrictions that are unnecessarily strict will severely limit its PCG capabilities.

Previous research into PCG for Angry Birds has been rather basic in terms of the complexity of the structures they generate. These prior methods create Angry Birds levels by generating columns of either single objects or small predefined structures [16]. These columns are then recombined using simple genetic algorithms in an attempt to maximize structural stability [15], [17]. Whilst this method is suitable for creating primitive structures in Angry Birds levels, it cannot generate anything more complex than an array of single columns.

This paper presents a search-based procedural content generator for the Angry Birds game which can create complex stable structures using a variety of different objects. The structures are evaluated using an improved fitness function which measures various important aspects. These include the structure's block count, pig count, aspect-ratio and pig dispersion. The probability of selecting certain block types during the construction process is evolved over successive generations, using this function as the optimisation criterion.

Several experiments were conducted to analyze the expressivity and of the structure generator. Metrics such as frequency, linearity, density and leniency were calculated to describe the characteristics of the content generated.

## II. ANGRY BIRDS

Angry Birds is a physics-based puzzle game where the player uses a slingshot to shoot birds at structures composed of blocks, with pigs placed within or around them. The player's objective is to kill all the pigs using the birds provided. A typical Angry Birds level, as shown in Figure 1, contains a slingshot, birds, pigs and a collection of blocks arranged in

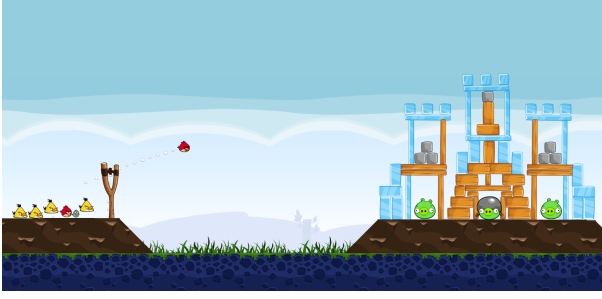


Fig. 1: Screenshot of a level from the Angry Birds game.

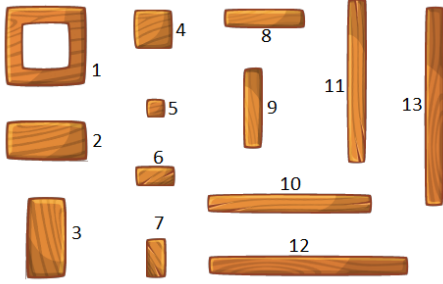


Fig. 2: The thirteen different block types available.

one or more structures. The ground is usually flat but can vary in height for certain difficult levels. Each block in the game can have multiple different shapes as well as being made of several possible materials.

Angry Birds is a commercial game developed by Rovio Entertainment who do not provide an open-source version of their code. Instead we use a Unity-based clone of the Angry Birds game developed by Lucas Ferreira [15], which is open-source and available to download from GitHub. This clone provides many of the necessary elements to simulate our procedurally generated structures in a realistic physics environment. There are currently eight different rectangular blocks available, of which five can be rotated ninety degrees to create a new block type. This gives a total of thirteen different block variants with which to build our structure, see Figure 2. Each block is also assigned one of three materials (wood, ice or stone), bringing the number of possible options to thirty nine.

### III. PROCEDURAL STRUCTURE GENERATION

The proposed structure generator operates by recursively adding rows of blocks to the bottom of the already generated structure. This process continues until a desired number of rows are reached. Unlike previous methods, our structure is created using only the original block types and does not require any composite elements to be created prior to structure generation. This vastly increases the number of possible structures that can be constructed, whilst also allowing greater algorithm flexibility to satisfy conditions and restrictions which may be imposed. The complexity of a generated structure can be defined in a manner similar to that of Kolmogorov complexity [18]. The extensive amount of variation that can occur within each structure, including the number, size, orientation and

### Algorithm 1 Structure Generation

---

```

1:  $currentRow \leftarrow 1$ 
2:  $blockType \leftarrow SelectBlockType(probabilityTable)$ 
3:  $currentStructure \leftarrow InitializeFirstRow(blockType)$ 
4: while  $currentRow < desiredRow$  do
5:    $subsets \leftarrow SubsetCombinations(currentStructure)$ 
6:    $blockType \leftarrow SelectBlockType(probabilityTable)$ 
7:    $currentStructure \leftarrow AddRow(blockType, subsets)$ 
8:    $currentRow \leftarrow currentRow + 1$ 
9: end while
10:  $PopulateStructure(currentStructure)$ 
11:  $EvaluateStructure(currentStructure)$ 

```

---

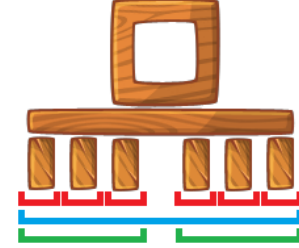


Fig. 3: The bottom row of this structure has three possible subset combinations: each block is in a separate set (red), all blocks are in a single set (blue), and the three left/right blocks are partitioned into two sets (green).

positioning of blocks used, allows our generator to create a diverse range of complex structures. Algorithm 1 provides an overview of the proposed generator, with a more detailed explanation given below.

#### A. Structure Generation

First, a starting block type is selected at random from all possible variants. This block type will become the peak(s) of the structure, beneath which all other blocks will be placed. For our implementation up to three blocks can be placed at the top of the structure at varying distances apart, with the number of peaks being chosen at random. Initially we are only concerned about the local positions of blocks relative to each other with the world positions being calculated after the structure has been fully generated.

After the first row has been initialized we recursively add more rows of blocks to the bottom of the currently generated structure. The blocks at the base of the structure are split into subsets based on the distances between them. All possible subset combinations are then recorded, see Figure 3. A new block type is then selected at random. For each possible subset combination there are now three possible supporting block placement options:

- Blocks are placed underneath the middle of each subset.
- Blocks are placed underneath the edges of each subset.
- Blocks are placed underneath both the middle and edges of each subset.

All three of these possibilities are shown in Figure 4. Each of these options is created for all subsets using the selected block type, after which they are tested for validity. Any case where blocks overlap each other is deemed invalid and is removed as a possible option. In addition, each object in the structure's bottom row is tested for local support by the new

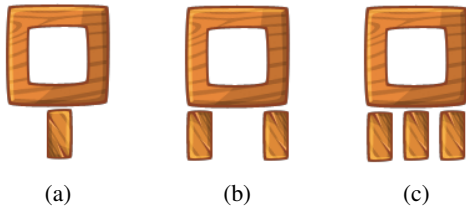


Fig. 4: The three possible supporting block placement options for a single block subset: middle (a), edges (b), both middle and edges (c).

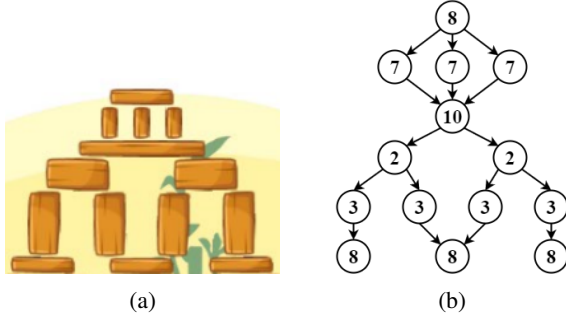


Fig. 5: An example of a generated structure (a) and its corresponding directed acyclic graph representation (b).

row. Each block in the bottom row of the current structure must be supported from below, either at its middle position or both of its edge positions. Any case that does not fulfil this requirement is also deemed invalid. After validity checks have been performed for all possible supporting block locations and subset combinations, one of the valid options is selected at random. If no valid options are available then a new block type is selected and the process repeated. The selected option is then used as the structure's new bottom row. This process is repeated until the desired number of rows is reached. Once the structure is complete each block is assigned a random material.

Any structure generated using this method can be depicted as one or more directed acyclic graphs, with each node representing a specific block. Each block is a descendant of the blocks that it supports (supportees) and subsequently an ancestor of the blocks that support it (supporters), see Figure 5. This can be extremely useful for other stability analysis techniques, such as identifying structural weak points [19].

### B. Pig Placement

Once the structure has been fully created it is populated with pigs. First, the space directly above the middle of each block is analyzed to see if there is space for a pig to fit such that it doesn't overlap any other blocks. If this is not possible for a particular block then the positions directly above the edges of the block are checked as well. Any positions that are found to be big enough to place a pig are recorded. Next, we test all the possible ground positions that are within the structure (to a set precision). Again we check for any overlap with nearby blocks and valid positions are recorded. We then randomly choose a position from all the valid possibilities and place a pig at the given location. Any remaining pig locations that would overlap the newly placed pig are removed and another location is chosen at random. This continues until there are no more valid locations or a desired number of pigs is reached.



Fig. 6: An example structure that has local stability but is globally unstable.

This process ensures that the structure will always contain at least one pig, as a pig can always be placed on top of the structure's peak block(s).

### C. Global Stability Analysis

Whilst our structure generation method ensures that each block has local stability, the global stability of the structure must be determined after its construction, see Figure 6. As all the relevant physics parameters (mass, density, friction and location) of blocks and pigs are known beforehand we can calculate the global stability of our structure exactly [20]. It is also possible to use qualitative stability analysis techniques to estimate the stability of the structure more quickly, whilst sacrificing some accuracy [21] [22]. Unfortunately, the Unity Engine upon which the Angry Birds clone is based suffers from simulation inaccuracies. These minor discrepancies cause structures which are theoretically stable to collapse within the simulation if given enough time. Currently, the only way to be certain that the structure will not collapse in this environment is to place the structure within a level and record if any blocks move a significant distance from their origin point [15]. If the structure is deemed unstable using the chosen approach then it is abandoned and a new structure is generated.

### D. Structure Placement

Once the structure has been fully generated it can be placed within the Angry Birds level. For the clone implementation, levels are specified as xml files with the block and pig locations given as coordinates in world space. First, we take the bottom row of our structure and place it on top of the level's ground (the location of the ground is fixed within the level). We then continue adding additional rows on top of the structure's base until all rows have been placed. Pig locations are then converted to their corresponding world coordinates and placed within the level as well. It is also possible to place multiple structures within the same level at different locations.

## IV. FITNESS FUNCTION

In order to evaluate individual structures against each other we define a fitness function to measure certain desirable properties. This fitness function calculates a fitness value for a given structure, with a lower fitness value indicating a more desirable structure. A fitness function has been proposed in previous Angry Birds papers [15], [16] for a similar reason but we believe it has several limitations in its current form. The original fitness function takes into account the structure's

simulated velocity over time (used to measure the stability of the structure) as well as the number of blocks and pigs used. Our method analyzes stability outside of the fitness function, automatically rejecting a structure if it is deemed unstable. This provides the user with more freedom over which approach to use and will allow any new stability estimation techniques to integrate seamlessly with our algorithm. Our fitness function also improves upon the previous implementation by updating the analysis of certain parameters, as well as proposing some new ones of our own. These can be separated into four distinct factors, number of pigs, number of blocks, structure aspect ratio and pig dispersion; each of which can affect the fitness value of a structure. We believe that this new function provides a broader and more sophisticated analysis of the structures generated by our algorithm.

#### A. Number of Pigs

This is the only component of the original fitness function that has not been altered. Simply put, the more pigs that are present within a structure the more desirable the structure.  $|p|$  is defined as the total number of pigs in the structure. This section of the fitness function is described by equation (1):

$$\frac{1}{1 + |p|} \quad (1)$$

#### B. Number of Blocks

The original fitness function defined this component as the difference between the desired and actual number of blocks, divided by the difference between the maximum and actual number of blocks. While this was appropriate for simple columns of blocks it becomes very impractical when used for more complex structures. This is because the maximum number of blocks that a structure could theoretically contain grows exponentially as the number of rows increases. For example, a ten row structure generated using our method typically contains between twenty and sixty blocks, but the maximum number it could theoretically contain is 88,572 (structure with three peak blocks and each block having three supporting blocks). This means that the value for this component of the fitness function will become insignificant for any structures with a medium to high number of rows. Instead, we suggest a more suitable calculation, where the difference between the desired number of blocks  $B$  and the actual number of blocks  $|b|$  is multiplied by a set factor  $X$ . This factor is used to adjust how much of an impact the difference between the desired and actual number of blocks has on the structure's overall fitness value. This section of the fitness function is described by equation (2):

$$X(\sqrt{(B - |b|)^2}) \quad (2)$$

#### C. Structure Aspect Ratio

One of the new components that we have added to our fitness function is the structure's width to height ratio (aspect ratio). Similar to the previous component, the maximum aspect ratio for any structure can be extremely large depending on the number of rows. This means that any attempt to normalize the

ratio by dividing by the maximum would severely reduce the effectiveness of this component. Instead, we simply multiply the difference between the desired ratio  $R$  and the actual ratio  $|r|$  by a set factor  $Y$ . This factor is used to adjust how much of an impact the difference between the desired and actual structure aspect ratio has on the structure's overall fitness value. This section of the fitness function is described by equation (3):

$$Y(\sqrt{(R - |r|)^2}) \quad (3)$$

#### D. Pig Dispersion

The other component that we have added to our fitness function is the dispersion, or spread, of pigs throughout the structure. The theory here is that structures with pigs located throughout them will be more desirable than structures with the pigs all grouped together. There are several methods that are currently available for measuring the spread of points (or in our case pigs) throughout a 2D space.

1) *Variance from center point*: This method estimates the dispersion of pigs by calculating the variance for the Euclidean distance between each pig's position and the mean position of all pigs. This value is then normalized by dividing it by the length of the diagonal of the structure's bounding box.

2) *Mean nearest neighbor distance*: This method estimates the dispersion of pigs by calculating the mean of the nearest neighbor distances for each pig [23]. This value is then normalized by dividing it by the length of the diagonal of the structure's bounding box.

3) *Morisita's index of dispersion*: This method first divides the structure's bounding box into a set number  $Q$  of equally sized quadrats. The number of pigs in each quadrat  $n_i$  is then counted and used together with the total number of pigs  $N$  to calculate Morisita's index of dispersion [24], described by equation (4):

$$MI = Q \left( \frac{\sum_{i=1}^Q n_i(n_i - 1)}{N(N - 1)} \right) \quad (4)$$

4) *Pig surrounding area overlap*: This method was created specifically to address limitations which were identified in the previous methods and so provides a robust estimation of pig dispersion. First, the total width and height of the structure is divided by the square root of the number of pigs. A rectangle with this new width and height is then placed at the location of each pig within the structure. If none of these rectangles overlap then their total area would equal the area of the structure's bounding box. However, it is likely that some of these rectangles will overlap those that are nearby, resulting in a lesser value. The total area that all the rectangles cover is then calculated and normalized by dividing it by the area of the structure's bounding box (maximum possible area).

5) *Comparison of methods*: Whilst all the methods described above give suitable estimations of pig dispersion for the majority of generated structures, there are several cases where they can give unreliable results. To compare all the methods, each was tested on four different structures, see Figure 7, and the results are given in Table I.



TABLE I  
COMPARISON OF PIG DISPERSION ESTIMATION METHODS

	Mean Variance	Mean Nearest Neighbor	Morisita's Index of Dispersion	Surrounding Area Overlap
Structure a	0.7314	0.0763	0.3333	0.5782
Structure b	0.3613	0.2568	0.6667	0.8908
Structure c	0.1592	0.0763	0.2778	0.3263
Structure d	0.5092	0.0763	0.5556	0.5958

In Figure 7, we can see that although the pigs are more dispersed in (b) than in (a) the mean variance from center point was higher for (a) than (b). This is because this method essentially rewards structures with pigs placed away from the center point, rather than structures with pigs dispersed throughout. A single grouping (c) would correctly give a very low dispersion value but two separate groupings results in an incorrect estimation.

In Figure 7, we can also see that although the pigs are more dispersed in (d) than in (c) the mean nearest neighbor distance is the same for both. This is because this method only uses the distance between each pig and its nearest neighbor to estimate pig dispersion. Having groupings of two pigs at multiple locations gives the same value as having all pigs at one location.

The problem with Morisita's index of dispersion is that although it gave good estimations for the structures tested, it relies on the number of quadrats to be chosen effectively. For this comparison, we created nine quadrats (3x3) but a different number of quadrats would have yielded quite a different result. This means that this method is only accurate when there are a large number of pigs available, so that each quadrat contains a sufficient number of pigs to be representationally accurate.

Our own method for estimating pig dispersion, based on measuring the overlap of each pig's surrounding area, performed well in all cases and can be normalized effectively. This method was therefore chosen to be used in our fitness function, where  $d$  defines the dispersion value. The set factor  $Z$  is used to adjust how much of an impact the dispersion of pigs has on the structure's overall fitness value. This section of the fitness function is described by equation (5).

$$Z(1 - d) \quad (5)$$

#### E. Complete Fitness Function

The sum of all these separate components for number of pigs, number of blocks, structure aspect ratio and pig dispersion makes up the complete fitness function, described by equation (6):

$$F = \frac{1}{1+|p|} + X(\sqrt{(B - |b|)^2}) + Y(\sqrt{(R - |r|)^2}) + Z(1 - d) \quad (6)$$

#### V. PROBABILITY TABLE

Instead of randomly selecting a block type during structure generation in an unbiased manner, a probability table can be used to alter the chance of a particular block type being selected. Each of the block types available is allocated a probability of being selected, with all probabilities summing

to one. Whilst this probability table allows for more designer control, it can also be optimized automatically using a training algorithm and our fitness function. The training algorithm attempts to find structures which minimise the fitness function for the given parameters. Each training algorithm iteration creates nine different structures (a single generation) and uses the fitness function to rank them from most desirable ( $R = 9$ ) to least desirable ( $R = 1$ ). The frequency of block types in each structure is then used to update the corresponding sections of the probability table using equation (7):

$$P_i = P_i + \frac{\sum_{R=1}^9 (S_{Ri})(R - 5)}{n \sum_{R=1}^9 (S_R)} \quad (7)$$

$P_i$  represents the probability table value for block  $i$ ,  $S_{Ri}$  represents the number of  $i$  blocks that the structure with rank  $R$  contains,  $S_R$  represents the total number of blocks that the structure with rank  $R$  contains, and  $n$  is an update factor which influences the speed at which the probability table values converge. If the probability table value for any block type is more than one then it is reduced to one. Likewise, any probability table value less than zero is increased to zero. After the probability table has been fully updated the values are renormalized so that they again sum to one. The probability table can be updated recursively over many generations using this technique.

The ability to update the probability table with the fitness function can be used to provide greater direction over what types of structures are created. Each component of the fitness function can be weighted to indicate how much emphasis should be placed on each factor. This allows the user to alter the parameters of the fitness function and hence tailor the output of the structure generator to suit their needs. For example, if the user prefers structures that are tall and thin, rather than wide and short, then the desired structure aspect ratio is set very low and the corresponding section of the fitness function weighted to give more of an impact on the structure's overall fitness value. The probability table is then repeatedly updated using this fitness function, after which the mean aspect ratio of structures generated using this new probability table will be less than before. Whilst this method does not guarantee that certain requirements will be met (e.g. the structure's height must be greater than its width) it can be used to improve the probability of such a structure being created without severely restricting the generator's expressivity.

#### VI. EXPERIMENTS AND RESULTS

Several experiments were carried out to test different components of the structure generator and fitness function.

##### A. Probability Table Optimisation

As previously discussed, a probability table for block type selection can be optimized over many generations using our specified fitness function. We therefore updated our probability table over 200 separate generations, with nine structures in each generation, for a total of 1800 structures. Each structure had ten rows and for our fitness function we defined:  $B = 40$ ,

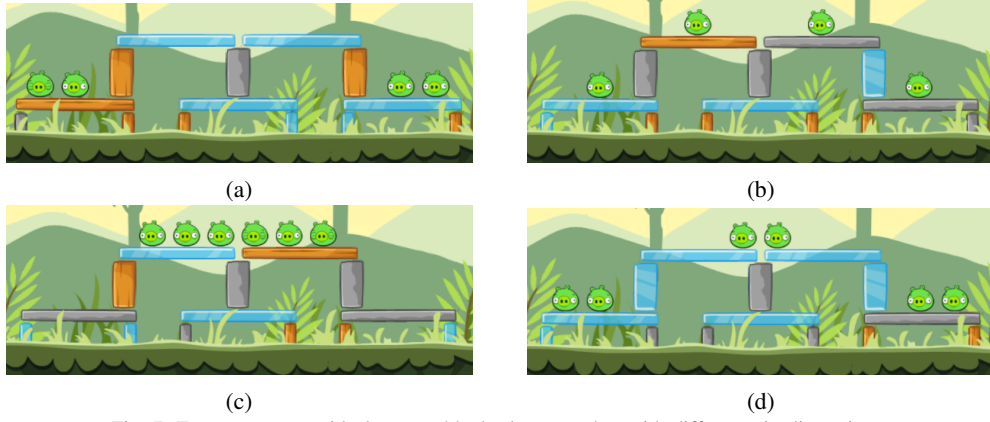


Fig. 7: Four structures with the same block placement but with different pig dispersions.

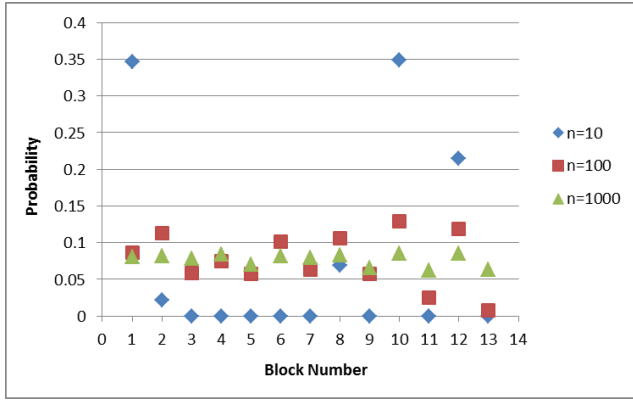


Fig. 8: Probability table values for each block type after 200 generations.

$R = 2.0$ ,  $X = 0.01$ ,  $Y = 0.2$  and  $Z = 1.0$ . We then compared three different update factors of  $n = 10$ ,  $n = 100$  and  $n = 1000$ , with the probability for each block type initially set to  $1/13$ . The result of this experiment is illustrated in Figure 8.

For  $n = 10$ , only five block types had a probability greater than zero. These were block types 1, 2, 8, 10 and 12, with block types 1 and 10 taking almost 70% of the probability between them. This is a clear indication that the update factor is set too low, as once the probability for a block type is near zero it is very difficult for it to increase again. This places an overemphasis on the fitness function, increasing the likelihood of creating a desirable structure, but greatly reducing the range of structures that can be generated.

For  $n = 1000$ , the probability values changed very little even after 200 generations. This suggests that the update factor is set too high and that the probability table values are not being updated by a significant amount for each generation.

For  $n = 100$ , the probability values have been updated a reasonable amount but the change is not so large as to significantly reduce the structure generator's expressivity. The probability values for block types 1, 2, 6, 8, 10 and 12 increased, whilst the values for block types 3, 4, 5, 7, 9, 11 and 13 decreased.

As a result of this experiment, an optimized probability table was created using 200 generations and  $n = 100$  for

each of three different row values, five, ten and fifteen. These probability tables were then used when analyzing the generator's expressivity.

### B. Expressivity analysis

Expressivity analysis has been described and implemented in many previous content generation papers as a means of comparing and contrasting different techniques. This is typically expressed as a metric which indicates the generator's strengths and weaknesses in various capacities. In this paper we define four measures based on metrics used in previous research [14], [15], [25]: frequency, linearity, density and leniency. Frequency evaluates the number of times that a block occurs within a structure. Linearity measures the width and height of each structure. Density provides a measure for the amount of 'free space' within a structure. Leniency estimates the difficulty of a structure, taking into account pig and block numbers. These metrics will allow our structure generator to be compared against any future methods. Presently however, there are no suitable prior algorithms with which to compare ours against.

For our experiments we generated 200 stable structures for each of three different row values, five, ten and fifteen. Each of these 200 structure groups was then sampled to find the average and standard deviation for the frequency, linearity, density and leniency. Example structures created using our generation algorithm are displayed in Figure 9.

Figure 10 shows the results of frequency sampling for structures with five rows. The average number of blocks is 12.72 with a standard deviation of 7.08. The average number of pigs is 3.07 with a standard deviation of 1.92. Figure 11 shows the frequency results for structures with ten rows. The average number of blocks is 27.39 with a standard deviation of 14.07. The average number of pigs is 4.93 with a standard deviation of 3.28. Figure 12 shows the frequency results for structures with fifteen rows. The average number of blocks is 47.07 with a standard deviation of 24.59. The average number of pigs is 7.54 with a standard deviation of 5.44.

The increase in pig numbers for structures with more rows is likely due to the increased number of blocks and hence the increased availability of viable pig locations. However, the pig

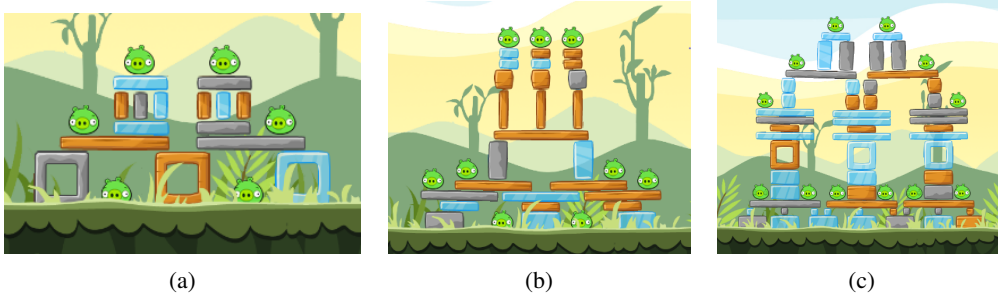


Fig. 9: Three example generated structures with five rows (a), ten rows (b) and fifteen rows (c).

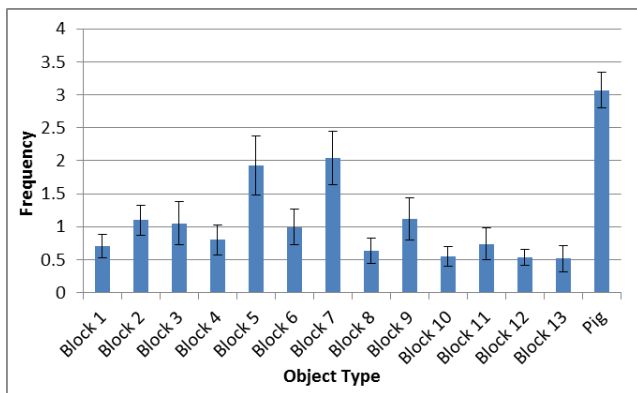


Fig. 10: Average and 95% confidence interval for block type frequency in structures with five rows.

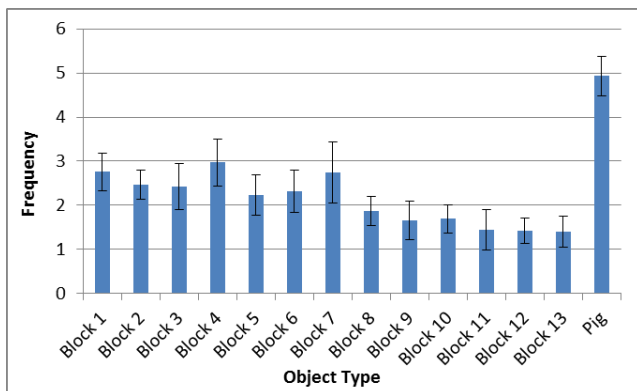


Fig. 11: Average and 95% confidence interval for block type frequency in structures with ten rows.

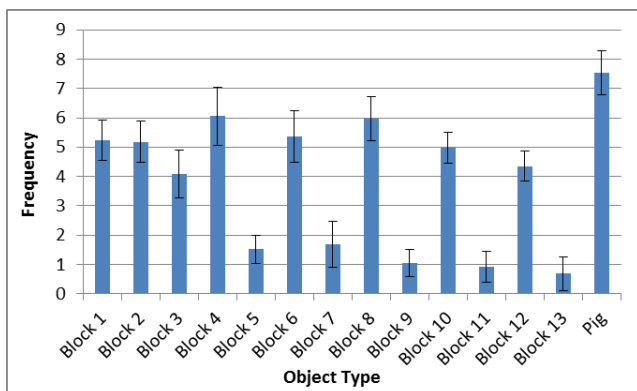


Fig. 12: Average and 95% confidence interval for block type frequency in structures with fifteen rows.

frequency relative to the block frequency was much greater for smaller structures than the larger ones. This is probably caused by the fact that the total number of pigs within a structure has a much greater impact on the fitness function for structures with a low number of blocks.

The relative frequencies of each block type also varied for structures of different sizes. Structures with fewer rows tended to favour smaller block types such as 5 and 7. This was likely due to the fact that their small width allowed more of them to fit within each row, which increased the total block count, and their small height meant that they did not decrease the structure's aspect ratio as much as taller blocks. Structures with more rows tended to favour the wider block types, as these both decreased the total block count and increased the structure's aspect ratio.

Linearity was measured using both the average width ( $\mu_W$ ) and height ( $\mu_H$ ) of all generated structures for each row amount, see Table II. The large standard deviation ( $\sigma$ ) shows that the structures created can differ greatly in terms of their width and height, indicating a large variation in the block arrangement of the generated structures.

The density of a structure was measured by summing the areas of all blocks within the structure and dividing this by the total area of the structure itself, including all sections of empty space that it contains. The average density ( $\mu_D$ ) for each row amount, as well as the standard deviation ( $\sigma$ ), is provided in Table II. The density of a structure appears to decrease as the number of rows increases, meaning that larger structures are likely to have more empty space within them and are therefore less robust than their smaller counterparts.

For many prior and current content generation methods, leniency is measured by analyzing the presence of certain objects within the subject [25], [26]. For this experiment we defined leniency using the number of pigs  $|p|$  and blocks  $|b|$  that are present within the structure, described by equation (8):

$$\text{Leniency} = -2|p| - |b| \quad (8)$$

Although primitive, this formula gives a rough estimate of how difficult it will be to kill all the pigs located within the given structure. The average leniency ( $\mu_L$ ) for each row amount, as well as the standard deviation ( $\sigma$ ), is provided in Table II. The leniency of a structure can be seen to increase with the number of rows, due to the expanded number of blocks and pigs that are present within the structure. This

TABLE II

LINEARITY, DENSITY AND LENIENCY FOR STRUCTURES WITH 5, 10 AND 15 ROWS

Rows	Width ( $\mu_W \sigma$ )	Height ( $\mu_H \sigma$ )	Density ( $\mu_D \sigma$ )	Leniency ( $\mu_L \sigma$ )
5	2.651 1.727	2.841 0.995	0.701 0.186	-18.86 10.22
10	3.631 1.765	5.749 1.563	0.653 0.169	-37.25 17.14
15	6.349 2.450	6.353 1.274	0.612 0.126	-62.15 25.92

information can be used to influence other important aspects within the Angry Birds game, such as the number of birds provided or the ordering of certain levels.

## VII. CONCLUSIONS AND FUTURE WORK

This paper has presented a search-based procedural content generation algorithm for creating complex stable structures within the video game Angry Birds. The algorithm builds structures using a top-down approach, with block types selected using a specified probability table. Each generated structure is symmetrical and can be represented as a directed acyclic graph. The structures created are populated with pig targets and analyzed for global stability. Other factors such as a varying number of peaks, multiple locations for support block placement and several possible materials, ensure that the range of possible structures is extensive and diverse.

Each generated structure is evaluated using a fitness function which considers the pig number, block number, aspect ratio and pig dispersion. This function can also be used to evolve the probability table by updating each block's chance of selection over many different generations. Each section of the fitness function can also be given a different weighting, allowing the user to define which aspects of the structure are most important.

Our structure generator was evaluated in terms of its expressivity and optimization potential. Four metrics were defined to investigate important aspects of the generated structures: frequency, linearity, density and leniency. The results of this analysis demonstrated that our structure generator can create a wide range of structures with many different attributes.

Future work could be to develop algorithms which create structures that can contain multiple block types and angles within each row. Additional research could also be conducted into estimating the number of birds required to kill all pigs within a given structure. This information could then be combined with our structure generation algorithm to create a full procedural level generator for Angry Birds.

## REFERENCES

- [1] M. Hendrikx, S. Meijer, J. V. D. Velden, and A. Iosup, "Procedural content generation for games: A survey," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 9, no. 1, pp. 1–22, 2013.
- [2] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 172–186, 2011.
- [3] S. Dahlskog and J. Togelius, "Patterns and procedural content generation: Revisiting mario in world 1 level 1," in *Proceedings of the First Workshop on Design Patterns in Games*. ACM, 2012, pp. 1:1–1:8.
- [4] G. N. Yannakakis and J. Togelius, "Experience-driven procedural content generation," *IEEE Transactions on Affective Computing*, vol. 2, no. 3, pp. 147–161, 2011.
- [5] A. Liapis, G. N. Yannakakis, and J. Togelius, "Optimizing visual properties of game content through neuroevolution," in *Artificial Intelligence for Interactive Digital Entertainment Conference*, 2011.
- [6] E. J. Hastings, R. K. Guha, and K. O. Stanley, "Evolving content in the galactic arms race video game," in *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, 2009, pp. 241–248.
- [7] C. Browne, "Automatic generation and evaluation of recombination games," Thesis, Queensland University of Technology, 2008.
- [8] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelback, G. N. Yannakakis, and C. Grappiolo, "Controllable procedural map generation via multiobjective evolution," *Genetic Programming and Evolvable Machines*, vol. 14, no. 2, pp. 245–277, 2013.
- [9] V. Valtchanov and J. A. Brown, "Evolving dungeon crawler levels with relative placement," in *Proceedings of the Fifth International C\* Conference on Computer Science and Software Engineering*. ACM, 2012, pp. 27–35.
- [10] L. Ferreira, L. Pereira, and C. Toledo, "A multi-population genetic algorithm for procedural generation of levels for platform games," in *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*. ACM, 2014, pp. 45–46.
- [11] L. Cardamone, D. Loiacono, and P. L. Lanzi, "Interactive evolution for the procedural generation of tracks in a high-end racing game," in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*. ACM, 2011, pp. 395–402.
- [12] M. Cook and S. Colton, "Multi-faceted evolution of simple arcade games," in *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, 2011, Conference Proceedings, pp. 289–296.
- [13] N. Shaker, M. Shaker, and J. Togelius, "Evolving playable content for cut the rope through a simulation-based approach," in *Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2013.
- [14] M. Shaker, M. H. Sarhan, O. A. Naameh, N. Shaker, and J. Togelius, "Automatic generation and analysis of physics-based puzzle games," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, 2013, pp. 1–8.
- [15] L. Ferreira and C. Toledo, "A search-based approach for generating angry birds levels," in *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*, 2014, pp. 1–8.
- [16] —, "Generating levels for physics-based puzzle games with estimation of distribution algorithms," in *Proceedings of the 11th Conference on Advances in Computer Entertainment Technology*. ACM, 2014, pp. 25:1–25:6.
- [17] M. Kaidan, C. Y. Chu, T. Harada, and R. Thawonmas, "Procedural generation of angry birds levels that adapt to the player's skills using genetic algorithm," in *2015 IEEE 4th Global Conference on Consumer Electronics (GCCE)*, 2015, pp. 535–536.
- [18] A. Kolmogorov, "Three approaches to the quantitative definition of information," *Problems Inform. Transmission*, vol. 1, no. 1, pp. 1–7, 1965.
- [19] P. Zhang and J. Renz, "Qualitative spatial representation and reasoning in angry birds: The extended rectangle algebra," *Fourteenth International Conference on the Principles of Knowledge Representation and Reasoning*, 2014.
- [20] A. G. M. Blum and B. Neumann, "A stability test for configurations of blocks," Massachusetts Institute of Technology, Tech. Rep., 1970.
- [21] Z. Jia, A. Gallagher, A. Saxena, and T. Chen, "3d-based reasoning with blocks, support, and stability," in *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, 2013, pp. 1–8.
- [22] X. Ge, J. Renz, and P. Zhang, "Visual detection of unknown objects in video games using qualitative stability analysis," *IEEE Transactions on Computational Intelligence and AI in Games*, 2015.
- [23] M. Dry, K. Preiss, and J. Wagemans, "Clustering, randomness, and regularity: Spatial distributions and human performance on the traveling salesperson problem and minimum spanning tree problem," *The Journal of Problem Solving*, vol. 4, no. 1, 2012.
- [24] M. Morisita, "Measuring the dispersion of individuals and analysis of the distribution pattern," Thesis, Kyushu University, 1959.
- [25] G. Smith and J. Whitehead, "Analyzing the expressive range of a level generator," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM, 2010, pp. 4:1–4:7.
- [26] D. Wheat, M. Masek, C. P. Lam, and P. Hingston, "Modeling perceived difficulty in game levels," in *Proceedings of the Australasian Computer Science Week Multiconference*. ACM, 2016, pp. 74:1–74:8.

# Monte-Carlo Simulation Balancing Revisited

Tobias Graf  
University of Paderborn  
International Graduate School "Dynamic Intelligent Systems"  
Email: tobiasg@mail.upb.de

Marco Platzner  
University of Paderborn  
Email: platzner@upb.de

**Abstract**—Simulation Balancing is an optimization algorithm to automatically tune the parameters of a playout policy used inside a Monte Carlo Tree Search. The algorithm fits a policy so that the expected result of a policy matches given target values of the training set. Up to now it has been successfully applied to Computer Go on small  $9 \times 9$  boards but failed for larger board sizes like  $19 \times 19$ . On these large boards apprenticeship learning, which fits a policy so that it closely follows an expert, continues to be the algorithm of choice. In this paper we introduce several improvements to the original simulation balancing algorithm and test their effectiveness in Computer Go. The proposed additions remove the necessity to generate target values by deep searches, optimize faster and make the algorithm less prone to overfitting. The experiments show that simulation balancing improves the playing strength of a Go program using apprenticeship learning by more than 200 ELO on the large board size  $19 \times 19$ .

## I. INTRODUCTION

Monte-Carlo Tree-Search (MCTS) [1] is one of the most important tree search algorithms in game playing especially in Computer Go. It builds up a search tree and evaluates the resulting positions using playouts. In a playout moves are randomly played out until the game ends where the game result can be determined by the rules of the game. The expected result of games is then used in the tree search as evaluation function.

Moves in a playout can be selected uniformly randomly or with the help of expert knowledge. This type of knowledge can be encoded by rules and patterns to assign each move a probability to be played. To find a good playout policy several techniques were developed. They range from finding good patterns by hand [2] to automatically determine weights by supervised learning [3], reinforcement learning [4] or simulation balancing [5]. Simulation balancing stands out from all algorithms as it is specifically developed for evaluating positions in MCTS. While supervised learning [3] and reinforcement learning [4] learn a policy which is a strong player in itself simulation balancing improves the expected result of a policy. As this is the primary usage in MCTS it was argued that this leads to stronger game playing programs [5]. Simulation balancing was compared to supervised learning and reinforcement learning on  $5 \times 5$  and  $6 \times 6$  Go and showed to be superior [5].

Later, simulation balancing was tested on more realistic  $9 \times 9$  and  $19 \times 19$  boards [6]. While simulation balancing showed to be superior to apprenticeship learning on  $9 \times 9$  it failed to do so on  $19 \times 19$ . Simulation balancing was also tested [7] in the open source program pachi [8] and while it marginally improved the playing strength apprenticeship learning proved to be the better algorithm. As a result up to now it is not clear

whether apprenticeship learning or simulation balancing is the better algorithm for policies in MCTS.

In this paper we reexamine simulation balancing and compare it to apprenticeship learning on  $19 \times 19$  Go. We introduce a few improvements to simulation balancing and show that also on large boards simulation balancing is the far superior algorithm improving our Go program by more than 200 ELO. The results confirm the original idea of simulation balancing that a balanced policy is more important than a strong policy [5].

The contributions of our paper are:

- Before running simulation balancing we pre-train the policy by optimizing the log likelihood on expert games. This not only decreases training time but eventually results in a superior policy as the optimization finds a better local minimum.
- We use the optimization algorithm ADAM [9] to improve upon plain stochastic gradient descent. This change is especially important for infrequent features in the policy that might have a major impact on playing strength.
- We demonstrate that target values can be extracted from expert games instead of using results from deep searches. This simplifies simulation balancing, allows for larger training sets and removes any inherent bias which is involved when using deep searches.
- We show that overfitting is a major problem with simulation balancing. In addition to using a large training set we control overfitting by a L2 regularization which keeps the learned policy close to the pre-trained policy.
- We run experiments to measure the playing strength in Computer Go. Contrary to previous papers [10] results show that simulation balancing improves policy weights even on boards with size  $19 \times 19$ .

The remainder of this paper is structured as follows:

In section II we provide background information on simulation balancing. In section III we introduce several extensions to the standard simulation balancing algorithm. In section IV we apply this algorithm to Computer Go. We evaluate the effect of the extensions to the algorithm, examine how strong simulation balancing is affected by overfitting and show the results of experiments on playing strength. Finally, in section V we draw our conclusion and point to future directions.

## II. BACKGROUND

In this section we introduce background information on the simulation balancing algorithm as presented in [5]. If we had a strong playout policy (always playing the best move), sampling this policy would result in a perfect evaluation of positions. Unfortunately, we will not be able to learn a playout policy which always plays the best move and as a consequence it will introduce errors in the evaluation. Small mistakes on every move can accumulate over the long playouts into a large overall error. So even a very strong policy can result in a bad overall evaluation if the small mistakes on every move accumulate unfavourably. The main idea of simulation balancing is to not optimize a policy to be strong (without any control of how the errors accumulate) but to produce a balanced policy so that the small errors on every move cancel each other out during the whole playout.

Throughout the paper we assume to use a *softmax policy* to generate playouts:

$$\pi_\theta(s, a) = \frac{e^{\phi(s, a)^T \theta}}{\sum_b e^{\phi(s, b)^T \theta}} \quad (1)$$

The policy  $\pi_\theta(s, a)$  specifies the probability of playing the move  $a$  in position  $s$ . It is calculated from the feature vector  $\phi(s, a) \in \mathbb{R}^n$  and a weight vector  $\theta \in \mathbb{R}^n$ . Some example features used in practice can be seen in Section IV. The gradient of the log of this policy is:

$$\nabla_\theta \log \pi_\theta(s, a) = \phi(s, a) - \sum_b \pi_\theta(s, b) \phi(s, b) \quad (2)$$

To simplify notation in the following we abbreviate this gradient with  $\psi_\theta(s, a)$ .

To learn the weights for a policy automatically we can fit the policy to follow moves from expert-games as closely as possible. To achieve this we can use apprenticeship learning and maximize the log likelihood of the policy on a set of expert games [5]. The resulting algorithm optimizes the log likelihood using the gradient:

$$\nabla \log \prod_{i=1}^n \pi_\theta(s_i, a_i) = \sum_{i=1}^n \nabla \log \pi_\theta(s_i, a_i) = \sum_{i=1}^n \psi_\theta(s_i, a_i) \quad (3)$$

For optimization we use a form of stochastic gradient descent which automatically determines learning-rates for all parameters by approximating the diagonal hessian matrix [11]. Another popular algorithm not used in this paper is based on the minorization-maximization principle [3].

In Monte Carlo Tree Search given a position  $s$  we are only interested in the result<sup>1</sup>  $z$  of the policy  $\pi_\theta$  starting in position  $s$ . The idea of simulation balancing is to optimize a policy so that this expected result  $\mathbb{E}_{\pi_\theta}[z|s]$  is similar to a given target value. The training set now consists of positions  $s_i$  and target values  $V^*(s_i)$  and we minimize the squared error:

$$\frac{1}{2} \sum_{i=1}^n (V^*(s_i) - \mathbb{E}_{\pi_\theta}[z|s_i])^2 \quad (4)$$

<sup>1</sup>This assumption is only true for plain MCTS and does not hold for extensions like RAVE [4].

To optimize the policy with stochastic gradient descent we need the gradient of the squared error  $(V^*(s_i) - \mathbb{E}_{\pi_\theta}[z|s_i])^2$  which is:

$$\nabla_\theta \frac{1}{2} (V^*(s_i) - \mathbb{E}_{\pi_\theta}[z|s_i])^2 = (V^*(s_i) - \mathbb{E}_{\pi_\theta}[z|s_i]) \cdot -\nabla_\theta \mathbb{E}_{\pi_\theta}[z|s_i]$$

The first part of the gradient  $(V^*(s_i) - \mathbb{E}_{\pi_\theta}[z|s_i])$  can be estimated by sampling random games following policy  $\pi_\theta$  in position  $s_i$  and averaging the results  $z$ . The second part  $\nabla_\theta \mathbb{E}_{\pi_\theta}[z|s_i]$  can also be sampled using insights from policy gradient reinforcement learning. If a game is defined as a sequence of states and actions  $g = (s_1, a_1, \dots, s_n, a_n)$  and a result of the game  $z(g)$  is 0 for a loss and 1 for a win then the gradient of the policy  $\pi_\theta$  can be determined using the REINFORCE algorithm [12]:

$$\begin{aligned} \nabla_\theta \mathbb{E}_{\pi_\theta}[z|s] &= \sum_g \nabla_\theta p_{\pi_\theta}(g) z(g) \\ &= \sum_g \nabla_\theta \left( \prod_i \pi_\theta(s_i, a_i) \right) z(g) \\ &= \sum_g \left( \prod_i \pi_\theta(s_i, a_i) \sum_j \frac{\nabla \pi_\theta(s_j, a_j)}{\pi_\theta(s_j, a_j)} \right) z(g) \\ &= \sum_g \left[ p_{\pi_\theta}(g) z(g) \sum_j \frac{\nabla \pi_\theta(s_j, a_j)}{\pi_\theta(s_j, a_j)} \right] \\ &= \sum_g \left[ p_{\pi_\theta}(g) z(g) \sum_j \nabla \log \pi_\theta(s_j, a_j) \right] \\ &= \mathbb{E}_{\pi_\theta} \left[ z(g) \sum_j \psi_\theta(s_j, a_j) \right] \end{aligned} \quad (5)$$

The first sum is over all possible games starting in position  $s$  weighted by  $p_{\pi_\theta}(g)$ , the probability of their occurrence under policy  $\pi_\theta$ . With this result to estimate  $\nabla_\theta \mathbb{E}_{\pi_\theta}[z|s]$  a number of random playouts is sampled using  $\pi_\theta$  starting in  $s$  while averaging  $z(g) \sum_j \psi_\theta(s_j, a_j)$ .

As shown both parts of the gradient can be approximated by running several playouts. To minimize the squared error of the training set we use two independent samples and apply stochastic gradient descent. The final algorithm is shown in Algorithm 1.

In practice we only have expert games with positions and move decisions of strong players but no target value for simulation balancing. In [5] and [6] target values were calculated by running a deep Monte Carlo Tree Search using apprenticeship learning as policy. If the search is deep enough this serves as a good approximation to the true value  $V^*(s)$ .

## III. EXTENDED SIMULATION BALANCING

To make simulation balancing more effective and easier to apply we used several extensions to the standard algorithm.

Instead of starting with a weight vector of zeroes we use apprenticeship learning [5] to pre-train the weights on expert



---

**Algorithm 1** Standard Simulation Balancing

---

```

 $\theta_0 \leftarrow 0$ 
for  $t = 0$  to  $T$  do
   $(s_1, V^*(s_1)) \leftarrow$  Random choice from training set
   $V \leftarrow 0$ 
  for  $j = 1$  to  $M$  do
    simulate  $(s_1, a_1, \dots, s_N, a_N, z)$  following  $\pi_{\theta_t}$ 
     $V \leftarrow V + z$ 
  end for
   $V \leftarrow \frac{V}{M}$ 
  for  $i = 1$  to  $M$  do
    simulate  $(s_1, a_1, \dots, s_N, a_N, z)$  following  $\pi_{\theta_t}$ 
     $g \leftarrow g + z \sum_{n=1}^N \psi_{\theta_t}(s_n, a_n)$ 
  end for
   $g \leftarrow \frac{g}{M}$ 
   $\theta_{t+1} \leftarrow \theta_t + \alpha(V^*(s_1) - V)g$ 
end for

```

---

games by optimizing the log-likelihood

$$\sum_i \log \pi_{\theta}(s_i, a_i)$$

which fits the policy to follow the moves chosen by an expert as closely as possible. In the simulation balancing algorithm instead of using zero weights the weight vector is initialized by the weights from apprenticeship learning. This has the advantage to start from a sensible policy which does not have to learn everything from scratch. It also the advantage that the initial weights can be learned by more powerful optimization algorithms like stochastic gradient descent with learning rates based on the diagonal hessian [11] or minorization-maximization [3].

In [6] it was shown that using 1/-1 for win/loss game results leads to faster learning than 1/0. This can be reframed to using 1/0 results but using a baseline  $b$  of 0.5, i.e. transforming the results  $z \leftarrow z - b$ . This is common practice in policy gradient reinforcement learning and we can choose any constant baseline without changing the expectation:

$$\begin{aligned}
& \mathbb{E}_{\pi_{\theta}} \left[ (z(g) - b) \sum_j \nabla \log \pi_{\theta}(s_j, a_j) \right] \\
&= \sum_g \left[ p_{\pi_{\theta}}(g) z(g) \sum_j \nabla \log \pi_{\theta}(s_j, a_j) \right] - \\
& \quad b \sum_g \left[ p_{\pi_{\theta}}(g) \sum_j \nabla \log \pi_{\theta}(s_j, a_j) \right] \\
&= \mathbb{E}_{\pi_{\theta}} \left[ z(g) \sum_j \nabla \log \pi_{\theta}(s_j, a_j) \right] - b \sum_g \nabla p_{\pi_{\theta}}(g)
\end{aligned}$$

$p_{\pi_{\theta}}(g)$  is the probability of a game under policy  $\pi_{\theta}$ . The last term then drops, as from  $\sum_g p_{\pi_{\theta}}(g) = 1$  it follows that  $\sum_g \nabla p_{\pi_{\theta}}(g) = 0$ . While 0.5 is a sensible choice for the baseline based on the reasoning that most positions are about equal we replace it by the average reward baseline. This removes one hyper-parameter to tune and allows for better learning if positions are not equal. As we already

independently sample  $M$  playouts to obtain the average reward  $V$  this baseline comes with no additional costs.

To avoid overfitting we regularize the policy by the L2-Norm. Instead of penalizing large values  $\|\theta_t\|_2^2$  we use the difference to the initial policy  $\|\theta_0 - \theta_t\|_2^2$ . In this way we can control how much the policy learned by simulation balancing is allowed to differ from the apprenticeship learning policy. If we interpret the initial policy learned by apprenticeship learning as a strong policy and the policy learned by simulation balancing as a balanced policy as argued in [5] then this form of regularization has an additional benefit: It explicitly controls the compromise between having a strong policy and a balanced policy. Experiments in [6] have shown that a balanced policy can play bad moves to achieve the desired result. Using this type of regularization allows us to distract the simulation balancing algorithm of this kind of trade-off.

Finally, we replaced plain stochastic gradient descent by the optimizer ADAM [9]. ADAM uses exponential moving averages of the mean  $m$  and the variance  $v$  of the gradients to update the weight vector. It is invariant to diagonal rescaling and each weight has its own learning rate. This improves learning in Computer Go as policy features appear with very different frequencies. Moreover, the moving average of the mean provides a kind of momentum and smooths the noisy gradients of simulation balancing. While it uses four parameters (learning rate  $\alpha$ , exponential decay rates  $\beta_1, \beta_2$  and minimum variance  $\epsilon$ ) these parameters were simple to set. We used recommended default values of  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$ ,  $\epsilon = 10^{-8}$  and only fine-tuned the learning rate  $\alpha$ .

---

**Algorithm 2** Improved Simulation Balancing

---

```

 $\theta_0 \leftarrow$  Weights from Apprenticeship Learning
 $m_0 \leftarrow 0$ 
 $v_0 \leftarrow 0$ 
for  $t = 1$  to  $T$  do
   $(s_1, V^*(s_1)) \leftarrow$  Random choice from training set
   $V \leftarrow 0$ 
  for  $j = 1$  to  $M$  do
    simulate  $(s_1, a_1, \dots, s_N, a_N, z)$  following  $\pi_{\theta_{t-1}}$ 
     $V \leftarrow V + z$ 
  end for
   $V \leftarrow \frac{V}{M}$ 
  for  $i = 1$  to  $M$  do
    simulate  $(s_1, a_1, \dots, s_N, a_N, z)$  following  $\pi_{\theta_{t-1}}$ 
     $g \leftarrow g + (z - V) \sum_{n=1}^N \psi_{\theta_{t-1}}(s_n, a_n)$ 
  end for
   $g \leftarrow \frac{g}{M}$ 
   $l \leftarrow (V^*(s_1) - V) \cdot g + \lambda(\theta_0 - \theta_t)$ 
   $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1)l$ 
   $v_t \leftarrow \beta_2 \cdot v_{t-1} + l \odot l$   $\triangleright$  element wise square
   $\hat{m}_t \leftarrow \frac{m_t}{1 - \beta_1^t}$ 
   $\hat{v}_t \leftarrow \frac{v_t}{1 - \beta_2^t}$ 
   $\theta_t \leftarrow \theta_{t-1} + \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$   $\triangleright$  element wise division
end for

```

---

Previously, the target values in the training set were generated by performing a deep MCTS [5] [6]. As this is a very time consuming process only small training sets of up to 10,000 positions have been used so far [6]. But if the training set is too

small overfitting is a major problem as we will show in Section IV. Moreover, if there is a bias in the policy which is used in the MCTS it is difficult to reduce this bias by performing a deep search. In Computer Go for example policies tend to like the centre of the board more than territory on the side. If such a policy is used inside MCTS the overall search will certainly have the same properties especially if the board is large and long term effects are not visible inside the search tree. To counter these problems we propose to use positions from expert games and the result of these games as target value. The target values (0/1) are less fine-grained than those of a MCTS but are less biased and there are plenty of expert games available. To avoid any potential problems with correlations between successive positions inside a game we only use one position/result-pair per game for the training set.

To decrease the training time we run the algorithm in parallel on 16 machines<sup>2</sup> by using mini-batches of positions of size 128. Positions in a mini-batch are evenly distributed across all machines and the result for the average results  $V$  and the gradient  $g$  are averaged before updating the parameters. With  $M$  large the number of simulations is so high that communication-costs are negligible. Moreover, the playouts in each of the two loops in simulation balancing are independent so that they can be distributed across all cores on a single machine. Using MPI and OpenMP the resulting algorithm with  $M = 1024$  takes about 45 min for one epoch trough the training set of 200,000 positions.

#### IV. EXPERIMENTAL RESULTS

##### A. Dataset

We used games<sup>3</sup> with the board size  $19 \times 19$  which were played on KGS with players having at least 4 dan strength using only no-handicap games with at least 150 moves. The resulting 270,000 expert games are used to create the dataset for positions and target values. Of each game we randomly chose one position and the result of the game as target value. We used 200,000 positions for the training set and 70,000 for the validation set.

##### B. Policy Features

The playout policy used in the experiments uses  $3 \times 3$  patterns and local features dependent of the last move (7,487 features in total). These are typical features of playout policies in Go and are similar to those used in a previous paper on Simulation Balancing [10]. Figure 1 illustrates some of the features.  $3 \times 3$  patterns contain atari-information of the four

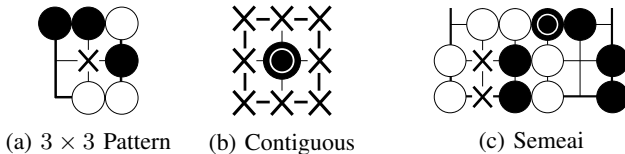


Fig. 1: Features of playout policy

direct neighbours and are grouped so that they are invariant to rotation/mirroring/colour-reversal.

The local features are:

- 1) Contiguous to the last move (distinguished by distance  $|dx| + |dy| + \max(|dx|, |dy|)$  for distances 2 and 3)
- 2) Save new atari-string by capturing
- 3) Save new atari-string by capturing but resulting in self-atari
- 4) Save new atari-string by extending (distinguished by group-size)
- 5) Save new atari-string by extending but resulting in self-atari (distinguished if resulting group has nakade shape)
- 6) Solve ko by capturing
- 7) 2-point semeai: if the last move threatens to kill a 2-liberty group any move which kills a neighbouring string within 2 moves has this feature
- 8) 3-point semeai: if the last move threatens to kill a 3-liberty group any move which kills a neighbouring string within 3 moves has this feature
- 9) 4-5-point semeai heuristic: if the last move reduces a string to 4 or 5 liberties and this string cannot increase its liberties (by a move on its own liberties) then any move on a liberty of a neighbouring string of the opponent which also cannot increase its liberties has this feature
- 10) Make-Eye: if the last move threatened to destroy an eye the move that creates the eye has this feature
- 11) Destroy-Eye: if the last move threatened to make an eye the move that destroys the eye has this feature
- 12) Nakade: if the last move created a nakade-shape the move on the vital point has this feature

##### C. Algorithmic Improvements

We conducted several experiments to measure the impact of our improvements to simulation balancing using a leave-one-out scheme. We remove the natural gradient, the average reward baseline or the initialization of the weights from the full algorithm (see Algorithm 2) and compare these variants using five epochs of training. We use the error of the training set to measure the differences between all variants and not of the validation set to exclude overfitting effects and only concentrate on the quality of the optimizing algorithm. For the same reason we also turned off the L2-regularization for all variants. The learning-rate for plain stochastic gradient descent was set to  $\alpha = 10$ . When using ADAM the learning-rate was decreased to  $\alpha = 0.01$ . We used a sample size of  $M = 1024$  and a mini-batch size of 128 using the parallel version as outlined above.

The results are shown in Figure 2. Leaving out the average reward baseline is almost equal to the full algorithm. This shows that a 0.5 baseline was a good default choice. The average reward baseline removes this parameter from the simulation balancing algorithm but does not improve the speed of learning. Using pre-training for the weights accelerates learning a lot. Moreover, simulation balancing can only find a local minimum. It seems to be important to start from good weights to achieve a low error. The ADAM solver improves

<sup>2</sup>Each machine has two Intel Xeon E5-2670 (16 cores total), 2.6GHz, 64 GByte main memory, connected with QDR InfiniBand PCIe3, 40 Gbit/s Mellanox

<sup>3</sup><http://u-go.net/gamerecords-4d/>

the speed of learning in contrast to stochastic gradient descent and achieves a lower error.

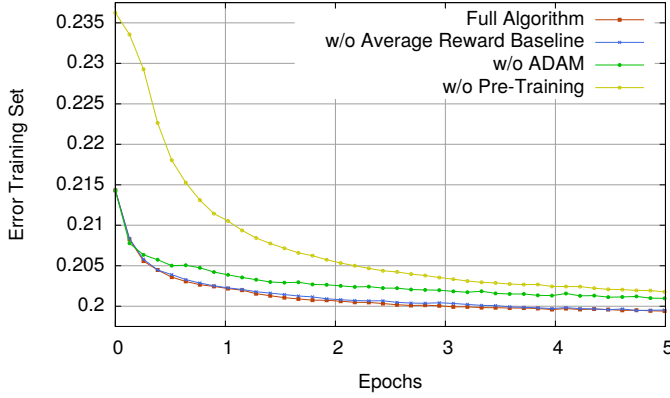


Fig. 2: Leave-One-Out Experiments

#### D. Evaluation Error

The mean squared error on the validation set is 0.2143 for apprenticeship learning while simulation balancing decreased the error to 0.2002 with  $\lambda = 1e - 7$ . Figure 3 shows the mean squared error depending on the game-phase each consisting of 15 moves. The MSE is calculated on 100,000 random positions and sampling 100 playouts for each position. In the opening phase apprenticeship learning and simulation balancing have a similar error and only in the middle game and endgame the policy is considerably improved. The failure to improve the policy in the opening might be attributed to the rather small  $3 \times 3$  patterns used in the policy which cannot capture opening patterns in Go.

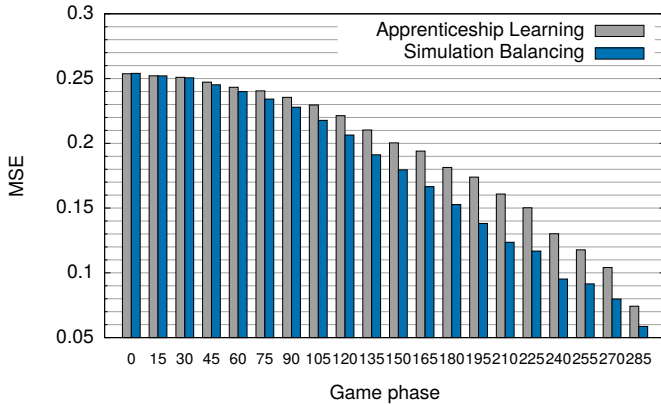


Fig. 3: Mean Squared Error of Apprenticeship Learning and Simulation Balancing for different game phases, 100,000 random positions, 100 playouts sampled at each position,  $\lambda = 1e - 7$

#### E. Overfitting and Regularization

To see how simulation balancing is affected by overfitting we decreased the amount of positions in the training set from 200,000 to 10,000 and kept the validation set as it is with 70,000 positions. Regularization was turned off.

In Figure 4 we show training and validation errors for this small training set. The gap between the error of the training set and the validation set increases during training and the error of the validation set even starts to get larger in the middle of training which is a clear sign of overfitting. In Figure 5 we show training and validation errors for the original training set with 200,000 positions. While there is still some gap between training set and validation set error the amount of overfitting is reduced considerably. This shows that simulation balancing is strongly affected by overfitting if the training set is too small. This might also explain earlier failures of simulation balancing on larger boards which used training sets of at most 10,000 positions [6].

To counter overfitting we used a large training set and added L2-regularization. In Figure 6 we show the effects of L2-regularization for different values for  $\lambda$ . We see that we can effectively control how much the policy is allowed to adapt to the training set.

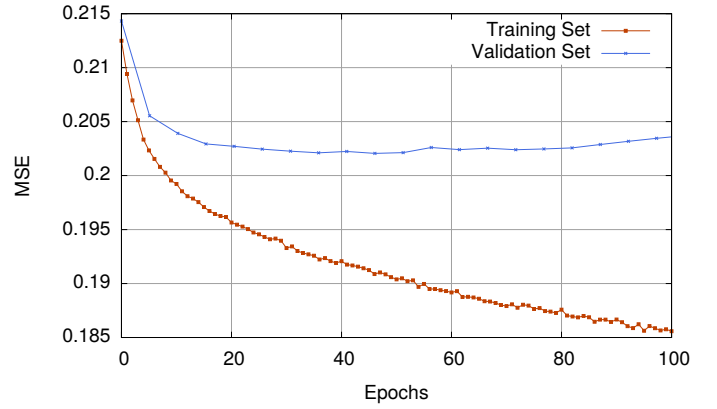


Fig. 4: Overfitting with a small training set with 10,000 positions

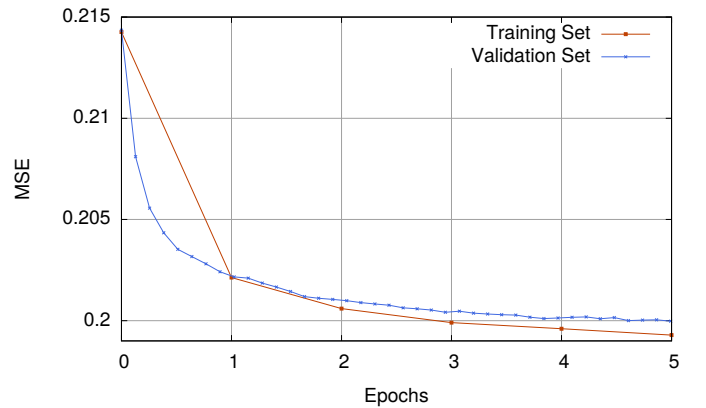


Fig. 5: Overfitting with a large training set with 200,000 positions

#### F. Playing Strength

We also measured the effect on the playing strength of our Go program Abakus. It uses RAVE [4], progressive widening

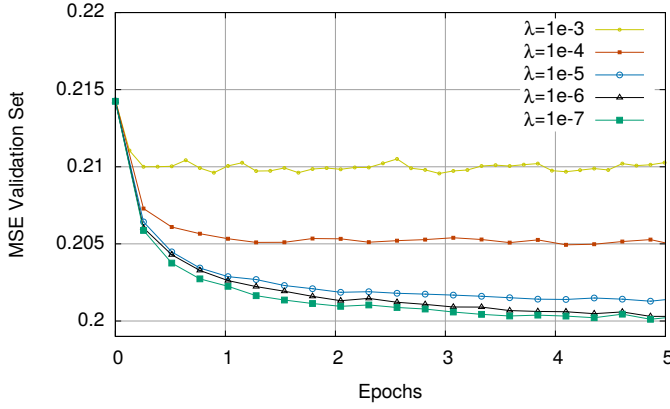


Fig. 6: L2-Regularization

[13], progressive bias [14] and a large amount of knowledge (shape and common fate graph patterns [15] as well as Deep Convolutional Neural Networks [16] [17]) in the tree search part. Playout weights of apprenticeship learning and simulation balancing are improved during the tree search with adaptive playouts [18]. Abakus holds a 6-Dan rank on the internet server KGS<sup>4</sup>.

To measure the playing strength we run tournaments against the open-source program Pachi [8]. As Abakus is considerably stronger than Pachi we used handicap games with 7 stones to level the chances. In addition to handicap games we also run a tournament with even games against Abakus itself. While self-play games can easily overrate an improvement they have the advantage to fully concentrate on the difference of the playout policies. With all factors involving the tree search or the knowledge applied inside the tree the same they give additional insight. We use 5s thinking time per move which translates into about 50,000 playouts/move for Abakus and 135,000 playouts/move for Pachi. Results are shown in Table I. MCTS with a playout policy learned by simulation balancing is more than 200 ELO stronger than MCTS with a policy learned by apprenticeship learning. Simulation Balancing uses  $\lambda = 1e-4$  which achieved the strongest playing strength.

Algorithm	Winrate	ELO
Apprenticeship Learning (vs. Pachi)	39.9% $\pm$ 2.8	-71 [-92,-51]
Simulation Balancing (vs. Pachi)	71.6% $\pm$ 2.6	161 [140,183]
Simulation Balancing (vs. Apprenticeship)	84.3% $\pm$ 2.1	292 [267,321]

TABLE I: Playing strength of abakus against pachi with 7 stones handicap or no handicap against itself, 1200 games played for each entry, 5s per move, 95% confidence intervals

Table II shows the playing strength with varying L2-regularization. An interesting observation is that a policy with a trade-off between strong and balanced achieved the best results and not the policy with the lowest validation error. This shows that regularizing the weights in relation to the apprenticeship policy is not only important to decrease the effect of overfitting. It is also crucial to achieve a strong playing strength by adjusting the relation of strength and balance in the playout policy. A reason for this might be

that our MCTS implementation uses RAVE [4] and adaptive playouts [18] which work more effective with a strong playout policy.

Regularization	Winrate	ELO	MSE Validation Set
1e-3	60.5% $\pm$ 2.8	74 [54, 94]	0.2102
1e-4	71.6% $\pm$ 2.6	161 [140,183]	0.2051
1e-5	66.5% $\pm$ 2.7	119 [99,140]	0.2014
1e-6	63.9% $\pm$ 2.7	99 [79,120]	0.2003
1e-7	62.4% $\pm$ 2.7	88 [68, 109]	0.2002
Apprenticeship learning	39.9% $\pm$ 2.8	-71 [-92,-51]	0.2143

TABLE II: Playing strength of abakus against pachi, 7 stones handicap, 1200 games played for each entry, 5s per move, 95% confidence intervals

### G. Weights

In Table III feature weights for local features are shown for apprenticeship learning and simulation balancing. We show the weights which achieved the best playing strength ( $\lambda=1e-4$ ) and the ones achieving the best mean squared error ( $\lambda=1e-7$ ). In general weights for capturing or semeai features are raised. This leads to a policy which plays tactical moves much more deterministically than a policy learnt by apprenticeship learning. This is in line with similar findings in previous simulation balancing experiments [6]. If only little regularization is used this effect is especially strong.

In addition, the contiguity weight is increased letting the policy play more around the last move. It is interesting that the simulation balancing algorithm can learn this automatically as it is well known that playing around the last move in playout policies in Computer Go is very valuable [2].

Feature	Apprenticeship Learning	Simulation Balancing $\lambda=1e-4$	Simulation Balancing $\lambda=1e-7$
Contiguous (distance 2)	9.5	66.7	139.7
Contiguous (distance 3)	6.7	31.2	115.5
Atari-Capture	470.6	2158.1	22,026.5
Atari-Capture-Self-Atari	0.3	0.3	0.8
Atari-Extend-Small	15.1	43.2	46.0
Atari-Extend-Big	106.5	163.0	1,143.1
Atari-Extend-Huge	183.2	426.0	11,680.1
Atari-Extend-Self-Atari	0.2	0.1	0.8
Atari-Extend-Self-Atari-Nakade	1.8	1.8	1.5
Solve-Ko	7.5	9.1	4.8
Semeai-2	165.6	306.4	14,013.0
Semeai-3	74.4	105.5	17,508.5
Semeai-n	6.4	8.8	2,550.0
Make-Eye	2.8	4.4	6.1
Destroy-Eye	6.0	10.3	53.1
Nakade	2.1	3.6	24.3

TABLE III: Comparison of feature weights for apprenticeship learning and simulation balancing for  $\lambda=1e-4$  and  $\lambda=1e-7$ , Gamma values of weights are shown,  $\gamma_i = e^{\theta_i}$

## V. CONCLUSIONS AND FURTHER WORK

In this paper we investigated simulation balancing which is an algorithm to optimize the parameters of a playout policy inside a Monte-Carlo Tree-Search. The algorithm was specifically designed to optimize policies for MCTS but so far has failed in practice on large boards in Computer Go [6] [7].

<sup>4</sup>www.gokgs.com

We added several improvements to the standard simulation balancing algorithm. Instead of starting from a random policy the weights are initialized by apprenticeship learning which optimizes the log-likelihood of the policy to match move decisions from experts. This proved to speed up training and resulted in a better local minima of the policy. Moreover, we added an average reward baseline to reduce the variance of the gradient. While this did not improve learning it removed a hyper-parameter from the algorithm. We replaced plain stochastic gradient descent by ADAM [9] which is an extension to SGD so that each weights has its own learning rate. As the policy has a lot of infrequent but important features this improved training-time and quality. Finally, we added L2-regularisation to avoid overfitting and control the trade-off between a strong and a balanced policy. Experiments on playing strength showed that simulation balancing produces much better policies than apprenticeship learning resulting in a strength improvement of more than 200 ELO. We also showed that neither a strong nor a balanced policy got the best playing strength but a mixture controlled by the L2-regularization. To simplify simulation balancing and to remove any inherent bias we did not generate a training set based on deep searches but used an expert database. From each game we took one random position and the game result as a target value. This allowed us to use much larger training sets than before which was shown to be important to decrease the amount of overfitting.

Future work includes testing simulation balancing in combination with value networks as used by AlphaGo [19]. AlphaGo uses deep convolutional neural networks to provide much better evaluations than playout policies. The evaluation of the value networks  $v_\theta(s)$  and the average results of the playout policy are then mixed in MCTS to provide a value estimate. While AlphaGo used apprenticeship learning to learn the policy weights simulation balancing might improve evaluations further. In this setting it might be useful to replace the cost function to minimize in Equation 4 by

$$\frac{1}{2} \sum_{i=1}^n (V^*(s_i) - [(1 - \gamma)v_\theta(s_i) + \gamma \mathbb{E}_{\pi_\theta}[z|s_i]])^2 \quad (6)$$

. In this way the policy weights are not optimized independently from the value network but are learned to complement the value network in reducing the MSE in the best possible way.

## REFERENCES

- [1] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, March 2012.
- [2] Y. Wang and S. Gelly, "Modifications of uct and sequence-like simulations for monte-carlo go," in *IEEE Symposium on Computational Intelligence and Games, 2007. CIG 2007.*, April 2007, pp. 175–182.
- [3] R. Coulom, "Computing Elo Ratings of Move Patterns in the Game of Go," *ICGA Journal*, vol. 30, no. 4, pp. 198–208, 2007.
- [4] S. Gelly and D. Silver, "Combining Online and Offline Knowledge in UCT," in *Proceedings of the 24th International Conference on Machine Learning*, ser. ICML '07, New York, NY, USA, 2007, pp. 273–280.
- [5] D. Silver and G. Tesauro, "Monte-Carlo simulation balancing," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML '09, New York, NY, USA: ACM, 2009, pp. 945–952.
- [6] S.-C. Huang, R. Coulom, and S.-S. Lin, "Monte-Carlo Simulation Balancing in Practice," in *Proceedings of the 7th International Conference on Computers and Games*, ser. CG'10, Berlin, Heidelberg: Springer-Verlag, 2011, pp. 81–92.
- [7] D. LaPlante and C. Nota, "Improvements to MCTS Simulation Policies in Go," Project Report, 2014.
- [8] P. Baudiš and J.-I. Gailly, "PACHI: State of the Art Open Source Go Program," in *Advances in Computer Games*, ser. Lecture Notes in Computer Science, H. van den Herik and A. Plaat, Eds. Springer Berlin Heidelberg, 2012, vol. 7168, pp. 24–38.
- [9] D. P. Kingma and J. L. Ba, "ADAM: A Method For Stochastic Optimization," in *The International Conference on Learning Representations (ICLR)*, 2015.
- [10] S.-C. Huang, R. Coulom, and S.-S. Lin, "Monte-Carlo Simulation Balancing in Practice," in *Proceedings of the 7th International Conference on Computers and Games*, ser. CG'10, Berlin, Heidelberg: Springer-Verlag, 2011, pp. 81–92.
- [11] T. Schaul, S. Zhang, and Y. LeCun, "No More Pesky Learning Rates," in *International Conference on Machine Learning (ICML)*, 2013.
- [12] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 1992.
- [13] K. Ikeda and S. Vennot, "Efficiency of Static Knowledge Bias in Monte-Carlo Tree Search," in *Computers and Games 2013*, 2013.
- [14] G. Chaslot, M. Winands, J. Uiterwijk, H. van den Herik, and B. Bouzy, "Progressive strategies for Monte-Carlo Tree Search," *New Mathematics and Natural Computation*, vol. 4, no. 3, pp. 343–357, 2008.
- [15] T. Graf and M. Platzner, "Common fate graph patterns in Monte Carlo Tree Search for computer go," in *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*, Aug 2014, pp. 1–8.
- [16] C. Clark and A. Storkey, "Training Deep Convolutional Neural Networks to Play Go," in *Proceedings of The 32nd International Conference on Machine Learning*, 2015.
- [17] C. Maddison, A. Huang, I. Sutskever, and D. Silver, "Move Evaluation in Go Using Deep Convolutional Neural Networks," in *International Conference on Learning Representations*, 2015.
- [18] T. Graf and M. Platzner, "Adaptive Playouts in Monte-Carlo Tree Search with Policy-Gradient Reinforcement Learning," in *Advances in Computer Games*, ser. Lecture Notes in Computer Science, A. Plaat, J. van den Herik, and W. Kesters, Eds. Springer International Publishing, 2015, vol. 9525, pp. 1–11.
- [19] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan 2016, article.

# Modeling Believable Game Characters

Hanneke Kersjes and Pieter Spronck  
Tilburg center for Cognition and Communication  
Tilburg University  
Tilburg, The Netherlands  
Email: hannekersjes@gmail.com, p.spronck@uvt.nl

**Abstract**—The behavior of virtual characters in computer games is usually determined solely by decision trees or finite state machines, which is detrimental to the characters' believability. It has been argued that enhancing the virtual characters with emotions, personalities, and moods, may make their behavior more diverse and thus more believable. Most research in this direction is based on existing (socio-)psychological literature, but not tested in a suitable experimental setting where humans interact with the virtual characters.

In our research, we use a simplified version of the personality model of Ochs et al. [1], which we test in a game which has human participants interact with three agents with different personalities: an extraverted agent, a neurotic agent, and a neutral agent. The model only influences the agents' emotions, which are only exhibited by their facial expressions. The participants were asked to assess the agents' personality based on six possible traits.

We found that the participants considered the neurotic agent as the most neurotic, while there are also indications that the extraverted agent was considered the most extraverted. We conclude that players will indeed distinguish personality differences between agents based on their facial expression of emotions. Therefore, using a personality model may make it easy for game developers to quickly create a high variety of virtual characters, who exhibit individual behaviors, making them more believable.

## I. INTRODUCTION

Over the last decades, computer games have evolved from plain and simple environments, to extensive virtual worlds, with refined visuals and options for players to make personal adjustments [2], [3]. The degree of realism that is achieved by the graphical representation of the game is, however, being shortchanged by the simplistic implementation of the non-player characters (virtual agents) in games, of which the behavior is often controlled by finite state machines [3] rather than responding to the human player. Such agents can hardly be called "believable". Creating more believable virtual agents is not necessarily about making them as realistic as possible [4]; on the contrary: Bates [5] describes the believability of an agent not as creating a reliable or honest virtual agent, but "one that provides the illusion of life", so that the player can have a "suspension of disbelief". Therefore creating a game that is credible is more important than creating one that is realistic [1].

A common solution used by game developers to make virtual agents respond to the human player is to code all possible options in a decision tree [1]. However, creating such a tree, even if only a limited number of options is taken into account, is time-intensive and costly, both during development

and testing. Games therefore limit the creation of complex interactions with a few choice non-player characters, turning the remainder into a series of clones which all respond to the player in exactly the same way.

An extensive attempt to tackle the problem of creating believable virtual agents for games was done by Ochs et al. [1], who developed a model which takes the virtual agent's personality and social relations into account, to determine its emotional state. They claim that their model is consistent with results from (social) psychology. However, they never tested their model in experiments with real life participants, to determine if human players actually consider the agents believable. McRorie et al. [6] evaluated the believability of virtual agents with human participants, but these evaluations were limited to the participants watching pre-recorded footage of interactions with virtual agents, rather than interacting with the agents themselves.

In our research we extend upon the work by Ochs et al. [1] and McRorie et al. [6] by actually testing a model for believable virtual agents in interaction with humans. We use a simplified version of the model of Ochs et al. [1]. We test this model on real life participants, who have to interact directly with the virtual agents, to find out to what extent the agents are indeed believable according to the participants.

## II. BELIEVABILITY OF VIRTUAL AGENTS

When discussing the believability of virtual agents, various researchers list different factors of importance. McRorie et al. [6] consider personality as a determinant factor for the believability. Ochs et al. [1] list emotions and social relations. Gebhard [8] add moods to the mix. Regardless the factors, Ortony [7] argues for internal consistency. In practice, the implementation of behavioral characteristics in virtual agents is not based on existing psychological knowledge, but on intuition of the game designers [6]. McRorie et al. [6] reason that an approach to designing virtual agents based on social psychology will result in more believable characters.

In the following subsections, we provide some theoretical background for the most commonly named factors of believability of virtual agents, as well as examples of recent research on modeling these factors.

### A. Emotions

The ability to express emotions is key for the believability of virtual agents, as it shows that the virtual agent "cares"



about what is happening in its surrounding environment [5]. Virtual agents who do not show any emotions do not provide an “illusion of life” and therefore do not generate compassion in players.

However, it is not easy to describe what, exactly, emotions are. Despite being extensively investigated, researchers have not yet agreed on a uniform definition of emotions. There is no agreed-upon clear distinction between the terms “emotion”, “mood”, “attitude” and “feeling” [9]. Literature often focuses on Ekman’s notion of six basic emotions: happiness, sadness, fear, anger, surprise and disgust [10]. Du, Tao and Martinez [11] describe a group of compound emotions, which can be constructed by combining basic emotions, such as “happily surprised” and “sadly disgusted”.

In designing virtual agents with believable emotions three main challenges occur: (1) identifying the specific circumstances in which a certain emotion should appear, (2) determining how to display these emotions, and (3) defining which behavior on the part of the agent will be the result [1].

As for identifying circumstances, the Ortony, Clore and Collings (OCC) model provides a popular approach to appraise events in research concerning the simulation of emotions [12]. The OCC model determines which of 22 types of emotion occurs using three different factors: the consequences of events, the actions of the agents, and the aspects of objects [13].

Displaying emotions in virtual agents is usually attempted via facial expressions, gestures, and tone of voice, with facial expressions being most important [14]. A well-known system that can be used for animating facial expressions in virtual agents is the Facial Action Coding System (FACS) [15]. The system measures facial expressions in terms of Action Units (AUs), which are the smallest possible units of moving muscles in the face. For each facial expression, certain combinations of AUs are activated with different intensities.

Amini et al. [16] developed the HapFACS software, which can simulate AUs described by FACS on virtual agents. The system works together with Haptik software, which creates virtual avatars. The advantage of using the software is that the AUs are FACS validated [16]. The main disadvantage is that it only runs on outdated operating systems.

### B. Personality

Personality can be described as “a pattern of behavioral, temperamental, emotional, and mental traits for an individual”, including its “longterm tendencies” [13]. It can be interpreted by “encoding and decoding [...] mainly non-verbal cues” [14]. According to Albeck and Badler [13], personality is key to creating believable autonomous agents. Ochs et al. [1] state that virtual agents with a personality will contribute to the consistency of a game, and thus to believability of the agents.

There are two well-known psychological models that describe personalities. The first is the model by Eysenck [17], which uses only two traits: extraversion versus introversion and neuroticism versus emotional stability. The second (and most commonly used) model is the five-factor model by Costa

and McRae [18], which adds to Eysenck’s model the factors openness, agreeableness, and conscientiousness.

McRorie et al. [6] explain how people tend to make automatic judgments about the personalities of their interaction partners, so they should also do this with virtual agents. Modern game characters already feature facial expressions and head and eye movements, on which players will base personality judgments. To test their theory, McRorie et al. [6] used still images of their designed virtual agents with different personalities as well as video clips, where the interaction between the virtual agents with a (male) player was shown. They measured three values: the virtual agents’ believability, consistency and familiarity. However, none of the participants in the study interacted with the virtual agents themselves.

### C. Moods

Moods are often considered similar to, but not the same as, emotions. Ekman [10] points out several differences between moods and emotions. The three main differences are: (1) moods last longer than emotions (though it is unclear what lengths of time are meant), (2) moods do not have a unique facial expression, contrary to emotions, and (3) moods may have an influence on emotions. As an example for the third difference, consider irritation (a mood) and anger (an emotion). When one is irritated, one is quicker to get angry, and being irritated causes the anger to last longer.

Gebhard [8] used mood as a factor in his “layered model of affect”, called ALMA, which was intended to simulate believable behavior in virtual agents. According to Gebhard, the three types of affect are emotions (short-term affect), moods (medium-term affect), and personality (long-term affect). These three types interact with each other in the ALMA model, which was implemented in virtual agents. However, whether or not this model improves believability of agents has not been investigated yet.

### D. Social relations

Most models of social relations contain a finite set of variables, each of them representing one of the dimensions of the social relation between two (virtual) agents [1]. Ochs et al. [1] state that each model uses different variables; there are no pre-set requirements that researchers have reached consensus on. An example of the implementation of social relations in a game is found in the work of McCoy et al. [19].

### E. Ochs’ model

Ochs et al. [1] did an extensive attempt to design virtual agents based on knowledge of the field of (social) psychology. Instead of having a focus on a single component that is important for the believability of a virtual agent (emotions, personality, moods or social relations), they developed a model, aimed at game developers, which calculates the virtual agents’ emotional state and its resulting behavior, by taking the virtual agents’ personality and social relations into account. They argue that this results in more consistent behavior, increasing the believability of the virtual agents.

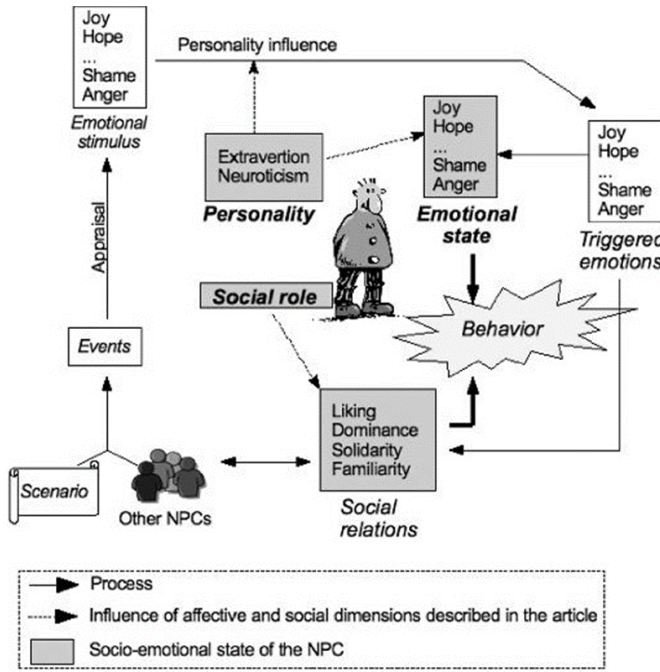


Fig. 1. Ochs' model (from Ochs et al. [1]).

Ochs' model (see Figure 1) is based on several theories in (social) psychology. It uses an adapted version of the OCC model to model emotions. Two differences with the OCC model is that Ochs' model uses 10 instead of 22 emotions (namely joy, hope, disappointment, distress, fear, relief, pride, admiration, shame, and anger), and that it does not require the definition of explicit agent goals (because, as Ochs reasons, they might often be difficult to apply for a game). Instead of goals, Ochs' model uses "actions, objects, and other characters of the environment" [1].

For personalities, Ochs' model uses the original Eysenck two-factor model. The influence of personality on emotions is based on Gebhard's ALMA model [8] (despite the fact that ALMA is used with moods rather than emotions). In Ochs' model, the intensity of an emotion is determined by the personality dimensions, as follows: a higher score on the extraversion dimension increases the intensity of "positive" emotions (such as joy, hope, pride, and relief), while a higher score on neuroticism increases the intensity of "negative" emotions (such as sadness, frustration, irritation, and anger). This idea of linking the effect of personality to emotions in this way follows from the work by Watson and Clark [20].

Social relationships are covered by Ochs' model by calculating the effects of "liking", "dominance", "solidarity", and "familiarity".

While Ochs' model is quite extensive, we note three shortcomings in its design and the reasoning behind the design:

- 1) No motivation is given for reducing the number of emotions from 22 to 10, and for adapting the ALMA model from moods to emotions;
- 2) While it is claimed that the model will result in believ-

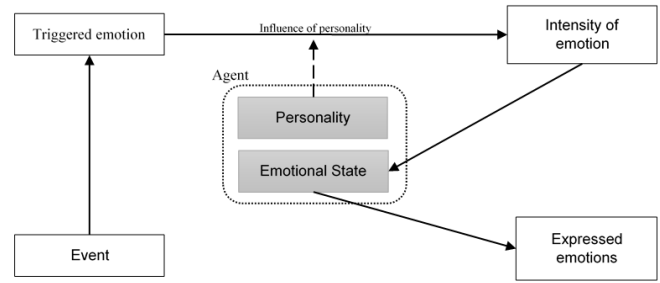


Fig. 2. Simplified version of Ochs' model.

able, consistent behavior, the model omits a description of how behavior follows from the model; and

- 3) The model was never tested with real-life participants; the only tests done with the model were on correctness of implementation, not on the effect on the believability of agents controlled by the model.

In our research, we aim to (partly) resolve the third of these shortcomings, by testing the effect of agents controlled by a version of Ochs' model on humans interacting with these agents.

### III. EXPERIMENTAL SETUP

For our experiments, we derived a simplified version of Ochs' model [1] to control agents with a predefined personality in an interaction with human participants.

#### A. Model design

The primary goal of our research is to investigate if personalities in a virtual agent, simulated by our model, are recognized by real-life participants. Since we focus on the recognition of the virtual agents personality, we have to exclude other interfering aspects of the model of Ochs et al. [1]. The simplified version of Ochs' model that we used is shown in Figure 2. There are two important differences between our model and Ochs' model:

- 1) In our model we do not account for attitudes. In Ochs' model, attitudes influence the intensity of emotions. Since we want to measure the effect of personality, which also influences the intensity of emotions, we have to leave attitudes out of the model.
- 2) Neither do we account for social relations in our model. In Ochs' model, social relations influence behavior next to emotional states. Since in our model we want to measure the effect of personality, which interacts with emotions which influence behavior, we need to exclude any other factors that influence behavior.

As can be seen in Figure 2, in our model the virtual agent is characterized by its personality, which influences the intensity of emotions triggered by events, and thus the emotional state of the virtual agent. The agent's emotional state determines its expression of these emotions.

Emotions can be expressed in different ways, such as visual appearance and behavior; however, in our experiments

we focus exclusively on facial expressions. Moreover, rather than including all ten emotions covered by Ochs' model, we chose to focus on only two combinations of emotions: joy-distress and admiration-anger. The reason to select these two combinations is that they are relatively easy to trigger and recognize, as they are direct reactions to events occurring at a present moment, instead of being related to a larger context of the past or future (according to the definition of these emotions by Ochs et al. [1]).

### B. Implementation

We implemented our model in a small game, in which the player has an interaction with three different virtual agents. The model allows the creation of a wide variety of agents, but for our experiments we decided to focus on three which are fundamentally different: (1) an extraverted agent which scores high on the extraversion dimension, but neutral on the neuroticism dimension; (2) a neurotic agent which scores high on the neuroticism dimension, but neutral on the extraversion dimension; and (3) a neutral agent, which scores neutral on both dimensions.

Each agent has an emotional state, consisting of values for joy, distress, admiration, and anger. In our game, the values for the emotions in the emotional state range from 0 to 2, with an initial value of 0.

Events in the game are defined as 3-uplets consisting of (1) an independent event that occurs or an agent that performs an action, (2) a positive or negative effect that follows, and (3) the virtual agent that experiences the effect. This information is used to determine which emotion occurs following the event. Emotions that may follow an independent event are joy (positive) or distress (negative). Emotions that may follow the action of an agent are admiration (positive) or anger (negative).

After the type of emotion is determined, the intensity is calculated. In our game the initial intensity of each emotion is 1, which is further influenced by the personality of the agent. For the extraverted agent, for positive emotions the intensity value is doubled, while for the neurotic agent, for negative emotions the intensity is doubled. For agents of other personality types (e.g., slightly neurotic but also a bit extraverted) obvious adaptations to these values can be made.

The experienced emotion's intensity is then added to the corresponding value in the agent's emotional state, in our implementation capped at the boundary of 2.

After every event in the game, all values in the emotional state of the agent are decreased by a "decrease rate". In Ochs' model, a logarithmic decrease rate is used, which we simplified to a linear rate (of which the effect happened to be close to what happens in Ochs' model), which decreases each of the values by 0.2 with a lower bound of 0.

Finally, the agent's emotional state determines its facial expression in the game. For this, we took the dominant emotion (i.e., the one with the highest value) from the emotional state, and selected a facial expression corresponding to the intensity of that emotion. Examples of facial expressions are shown in Figure 3: rows from top to bottom representing joy, distress,

admiration, and anger, columns from left to right representing intensity values 0.4, 0.8, 1.2, 1.6, and 2.0. The emotional expressions were generated with the FaceGen software.

FaceGen has pre-programmed expressions for "anger". For "distress", we used the pre-programmed "sadness" expressions. For "joy", according to Du et al. [11] AUs 6, 12, and 25 are used, which we managed to approach using detailed FaceGen sliders. For "admiration" we consulted Parke and Waters [21], which describe an admiration expression as generated by "the muscles of astonishment associated with those of joy", which we also implemented using detailed FaceGen sliders. We pre-tested the expressions with ten participants, asking them to describe the emotion associated with a facial expression. Most participants labeled our initial creations correctly, except for "admiration". We tweaked the expression for "admiration" and offered it to a second group of ten test participants, of which five specifically recognized the face as expressing "admiration", and none of the others labeled it as a negative emotion. This was the expression that we used in the experiments.

Note that we are not claiming that personality and emotions *only* influence facial expressions in practice: in fact, they influence all aspects of interaction, including speech patterns [22], gestures [23], and behavioral responses [18]. However, to be able to correctly assign the players' assessments to particular elements in the game, we wanted to limit the effects of the model to a single feature, for which we chose facial expressions.

### C. Game

We implemented our game in Ren'Py, a tool for creating visual novels. Players control the game by making menu choices in their interaction with virtual agents. The simple storyline of our game had the participant fulfill the role of a detective who is investigating a robbery. The player had to interrogate three virtual agents: the extraverted, neurotic, and neutral agents that we defined. Two short excerpts from a dialogue are displayed in Table I, including the emotional effect of the player's statements on the agent, and whether this effect was caused by an event, or an action of the player. These excerpts provide only one possible choice for each of the player's dialogue options. Note that in the actual game the dialogue is much longer, and includes a situation in which the player suspects the agent of being involved in the robbery.

The choices that the player made had little effect on the progression of the story; while it might seem to the participant that their choices drove the interrogation, they actually always led to the same follow-up responses of the virtual agents. The interactions were slightly different between the three agents that the player encountered, but the order in which the interactions were presented to the participants were always the same. However, the three personalities imprinted on the agents differed between the participants: the six different orders in which the three personalities could be presented to the participants all occurred the same number of times. For example, while the first agent encountered always had



Fig. 3. Facial expressions.

a particular interaction with the participant, one-third of the participants had this interaction with the extraverted agent, one-third with the neurotic agent, and one-third with the neutral agent. The emotional impact of the events was only visible in the facial expressions of the agents.

In each interaction, five different events occurred that triggered an emotion. Three of those were negative, one of them caused by the participant. Two were positive, one of them caused by the participant.

After each of the three interactions with the virtual agents, the participants had to indicate to what extent they thought the virtual agents had certain personality traits. This was questioned by means of five-point Likert scales, where one resembled “totally disagree” and five resembled “totally agree”. The participants did this for the following six statements: (1) the agent is introverted, (2) the agent is emotionally stable, (3) the agent is orderly, (4) the agent is extraverted, (5) the agent is neurotic, and (6) the agent is intelligent. The third

and sixth statement served as control statements: in theory, the scores on these two statements should not differ between the personalities.

At the end of the game, the participants had to answer a few questions on their gender, age, education and experience with games.

#### D. Participants

36 people participated in the experiment, who were between 21 and 57 years old ( $M = 27.31$ ,  $SD = 10.11$ ). Of these participants, 16 were male and 20 female. Four people attended higher professional education, 32 went to university. 16 of the participants stated they did not have any experience with computer games at all. The participants were equally distributed over the six conditions.

TABLE I  
TWO EXCERPTS FROM A DIALOGUE.

Speaker	Text	Effect
Player:	"You have a personal storage locker at the bank. Is that correct?"	
Agent:	"Yes."	
Player:	"Can you tell me what was in it?"	
Agent:	"Why? What is this about?"	
Player:	"The bank was robbed this weekend. I am investigating the robbery."	Negative (event)
Agent:	"I understand. There were a few family valuables in the locker. Jewelry and a watch."	
Player:	"Some of the personal lockers were broken into. I am afraid that yours was also emptied."	Negative (event)
Agent:	"Will I ever see my possessions again?"	
Player:	"I can tell you that some of the stolen goods were recovered this morning."	Positive (event)
Agent:	"Did you also find my jewelry?"	
Player:	"If you give me a moment, I will check." (Player makes a phone call)	
Player:	"My colleague tells me that it looks like some of your possessions have indeed been found. I will make sure that they are returned to you as soon as possible."	Positive (action)

#### IV. RESULTS

Figure 4 shows the average assessment of the participants over the personality traits of the three virtual agents (exact numbers follow in Table III, and more numerical details have been reported by Kersjes [24]). The six personality traits that are assessed are shown in groups of three bars, of which the left represents the score for the extraverted agent, the middle for the neurotic agent, and the right for the neutral agent. We stress once more that by balancing the order in which the virtual agent personalities were presented to the participants, the differences between the assessments are the sole result of the participants' observations of the facial expressions of the virtual agents, which were generated by our personality model.

A visual inspection shows some notable differences between the assessments:

- All three agents score higher on "extraverted" than on "introverted". This indicates that all of them are considered to be leaning towards being extraverted, though for the extraverted agent the difference between the scores is a bit higher than for the other two agents.
- Both the extraverted and neutral agent score higher on "emotionally stable" than on "neurotic". For the neurotic agent, however, these scores are almost equal. This indicates that the participants assess the neurotic agent as considerably more neurotic than the other two agents.
- Finally, the extraverted agent scores higher on both "orderly" and "intelligent" than the other two agents. The neurotic agent scores lowest on these traits, though the difference with the neutral agent is small.

##### A. Reliability and factor analysis

In order to further analyze these results, a reliability and factor analysis was conducted. All scores on each of the six traits were combined into six general (dependent) variables. The personality of the agents served as an independent variable in this way of grouping the data. The polarity of "introverted" and "neurotic" was reversed. The internal consistency of the

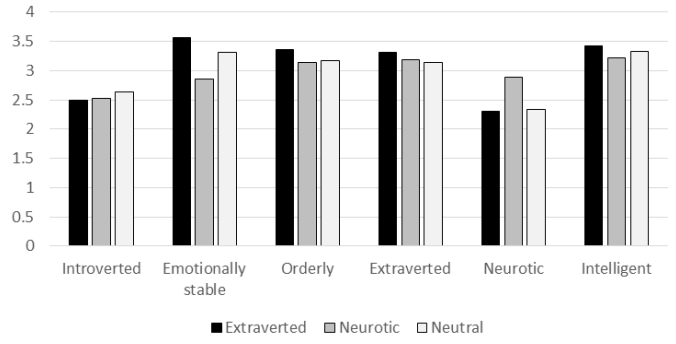


Fig. 4. Assessment of personality types.

TABLE II  
FACTOR ANALYSIS RESULTS.

Scale	Introverted-extraverted	Emotionally stable-neurotic	Orderly-intelligent
Introverted (reversed)	.83		
Emotionally Stable		.73	.24
Orderly		.11	.44
Extraverted	.77		
Neurotic (reversed)		.84	.10
Intelligent		.12	.67

six measured traits was low (Cronbach's  $\alpha = .45$ ), thus all items appeared worthy of retention.

A principal factor analysis was conducted, resulting in three factors. Factor loadings are displayed in Table II. The clustered items indicated that the first factor represents the introverted-extraverted trait, the second factor represents the emotionally stable-neurotic trait, and the third factor represents the two control variables: orderly and intelligent. We added these factors as three new variables, consisting of (1) the mean score of extraverted and reversed introverted, (2) the mean score of emotionally stable and reversed neurotic, and (3) the mean score of orderly and intelligent.

For the nine variables (the six original scores and the three new ones) we did an analysis of variance with repeated measures. Table III shows the means and (between parentheses) standard deviations for each of the scores.

The analysis of variance shows the following:

- Between the agents, there is no significant difference for the scores on "introverted", "orderly", "extraverted", "intelligent", "introverted-extraverted", or "orderly-intelligent".
- Between the agents, there is a significant difference for "neurotic" ( $F(2, 70) = 7.02, p < .005$ ). The score for the neurotic agent is significantly higher than the scores for the other two agents. There is no significant difference between the scores for the extraverted and neutral agents.
- Between the agents, there is a significant difference for "emotionally stable" ( $F(2, 70) = 8.72, p < .001$ ). The score for the neurotic agent is significantly lower than the scores for the other two agents. There is no significant difference between the scores for the extraverted and neutral agents.

TABLE III  
AVERAGE SCORES ON THE PERSONALITY TRAITS AND FACTORS.

	Extraverted	Neurotic	Neutral
Introverted	2.50 (.97)	2.53 (.88)	2.64 (.90)
Emotionally stable	3.56 (.70)	2.86 (.93)	3.31 (.75)
Orderly	3.36 (.87)	3.14 (.80)	3.17 (.74)
Extraverted	3.31 (.89)	3.19 (.82)	3.14 (.93)
Neurotic	2.31 (.67)	2.89 (1.01)	2.33 (.72)
Intelligent	3.42 (.65)	3.22 (.54)	3.33 (.76)
Introverted-extraverted	3.40 (.85)	3.33 (.74)	3.25 (.85)
Emotional stable-neurotic	3.63 (.58)	2.99 (.90)	3.49 (.65)
Orderly-intelligent	3.39 (.62)	3.18 (.56)	3.25 (.58)

- Between the agents, there is a significant difference for “emotionally stable-neurotic” ( $F(2, 70) = 10.15, p < .001$ ). The score for the neurotic agent is significantly lower than the scores for the other two agents. There is no significant difference between the scores for the extraverted and neutral agents.

### B. Conversation order

While the personalities were presented in varying orders to the participants, the textual aspects of the story always occurred in the same order. To see whether there was a difference between the three conversations that the participants had, we did an analysis of variance for these conversations. We found a significant difference for the “introverted” scores ( $F(2, 70) = 4.69, p < .05$ ), indicating that the participants assessed the second agent as significantly more introverted ( $M = 2.92, SD = .97$ ) than the other two agents ( $M = 2.33, SD = .79$  and  $M = 2.42, SD = .87$ ). We found the same for the “extraverted” score ( $F(2, 70) = 9.45, p < .001$ ), where the second agent was significantly less extraverted ( $M = 2.78, SD = .80$ ) than the other two agents ( $M = 3.58, SD = .69$  and  $M = 3.28, SD = .94$ ). These results indicate that the second conversation gave textually the impression of the agent being more introverted than the agents in the other two conversations.

## V. DISCUSSION

We found significant differences between the assessment of the participants of the agents’ emotional stability and neuroticism. The agent for whom, based on our model, the facial expressions were created representing a neurotic personality, was deemed, by the participants, as considerably more neurotic than the other two agents. Note that the main effect of having the neurotic personality was that the agent showed a more intense response to negative events than the other two agents, while showing the same response to positive events as the neutral agent. We may therefore conclude that the model manages to display a recognizable neurotic personality.

The participants did not clearly recognize the extraverted personality, though the individual scores that the extraverted agent received seem to indicate that this personality scored higher on “extraverted” and lower on “introverted” than the other two agents. We offer two possible reasons why the results for being extraverted were less pronounced than the results for being neurotic.

Firstly, as explained above, we found that the participants considered the second agent they encountered as significantly more introverted than the other two agents. This indicates that the textual cues of the second conversation were representative of an introverted personality. While we had intended the results to only depend on facial expressions, we now know that the results for introversion and extraversion were influenced by the conversation texts. In future research, this can be resolved by not only varying the order of the agent personalities, but also the order of the conversations.

Secondly, each conversation contained three events that triggered negative emotions (which make a difference for the neurotic agent), while only two events triggered positive emotions (which make a difference for the extraverted agent), which also occurred late in the conversations. In retrospect, this might have caused the neurotic traits to be recognized more easily. In future research a longer and more diverse interaction with the agents may make it easier to recognize all personality traits.

One interesting observation on our results is that, while we included two control traits that should not differ between the personalities, from Figure 4 we can see that the participants considered the extraverted agent to be more orderly and more intelligent than the other two agents, and that the neurotic agent was considered less orderly and less intelligent than the other two. Evidently, subconsciously people automatically associate extraversion with being orderly and intelligent.

As for the use of a personality model by game developers: the idea is that the model determines emotional state based on settings for personality (and perhaps also moods and social relations). This emotional state can then be used to determine not only facial expressions, but also actual behaviors, body language, tone of voice, and choice of words. In our research we only used facial expressions to allow us to draw solid conclusions on participants recognizing personality features; however, game developers naturally would want to include more extensive influences of emotional states. This would help the diversity and believability of the portrayed characters.

## VI. CONCLUSION

One approach to create more believable characters in games is to provide characters with a personality, which automatically influences the characters’ behavior and (emotional) expressions without extra coding on the part of the game developers. For instance, two guards could have exactly the same programmed conversations with the player, but by using a personality model to generate their visual characteristics (such as facial expressions), these two guards can give the impression to the player of being two fundamentally distinct characters. In a more extensive implementation where also behaviors are concerned, one guard could have a personality that responds more intensively to negative emotions than the other, and thus be quicker to get angry and try to arrest the player.

Few personality models for game characters are known, and as far as we know none of them have been tested



with human participants interacting with virtual characters. Therefore, there is little evidence that humans indeed think the virtual agents driven by such a personality model behave in a believable or realistic way. In our research, we presented a simplified version of the model of Ochs et al. [1], which we use to let three different agents interact with human participants in a game; the three agents have an extraverted, neurotic, and neutral personality. In our experimental setup, personality only influences the facial expressions of the agents.

Our results showed that the neurotic agent was indeed deemed to be more neurotic than the other two agents. While the results for the extraverted agent were not significant, a visual inspection (see Figure 4) shows that it was deemed to be more extraverted than the other two agents. It should be noted that the results for the extraverted agent might have been influenced by the fact that the second agent that the participants interacted with was considered to be more introverted than the other two, regardless of its actual personality. This issue may be resolved in future research.

We conclude that using a personality model to control the behavior and emotional expressions of virtual characters may indeed have human players regard these characters as fundamentally different with recognizable personalities, even if the interaction or storyline remains unchanged. Therefore, using a personality model may make virtual characters more diverse, more human-like, and thus more believable.

## REFERENCES

- [1] M. Ochs, N. Sabouret, and V. V. Corruble, "Simulation of dynamics of non-player characters' emotions and social relations in games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 1, no. 4, pp. 281–297, 2009.
- [2] S. Priesterjahn, O. Kramer, A. Weimer, and A. Göbels, "Evolution of human-competitive agents in modern computer games," in *IEEE Congress on Evolutionary Computation*. IEEE Press, 2006, pp. 777–784.
- [3] C. Thureau, C. Bauckhage, and G. Sagerer, "Learning human-like movement behavior for computer games," in *In From Animals to Animals 8*. MIT Press, 2004, pp. 315–323.
- [4] E. Hudlicka and J. Broekens, "Foundations for modelling emotions in game characters: Modelling emotion effects on cognition," in *3rd International Conference on Affective Computing and Intelligent Interaction and Workshops*. IEEE, 2009.
- [5] J. Bates, "The role of emotions in believable agents," *Communications of the ACM*, vol. 37, no. 7, pp. 122–125, 1994.
- [6] M. McRorie, I. Sneddon, G. McKeown, E. Bevacqua, E. D. Sevin, and C. Pelachaud, "Evaluation of four designed virtual agent personalities," *IEEE Transactions on Affective Computing*, vol. 3, pp. 311–322, 2012.
- [7] A. Ortony, "On making believable emotional agents believable," in *Emotions in humans and artifacts*, R. Trappl, P. Petta, and S. Payr, Eds. MIT Press, 2003.
- [8] P. Gebhard, "Alma – a layered model of affect," in *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05)*, 2005, pp. 29–36.
- [9] I. Lopatovska and I. Arapakis, "Theories, methods and current research on emotions in library and information science, information retrieval and human-computer interaction," *Information Processing & Management*, vol. 47, pp. 575–592, 2011.
- [10] P. Ekman, "Moods, emotions and traits," in *The nature of emotion*. New York: Oxford University Press, 1994, pp. 56–58.
- [11] S. Du, Y. Tao, and A. Martinez, "Compound facial expressions of emotion," in *Proceedings of the National Academy of Sciences of the United States of America*, vol. 111, 2014.
- [12] A. Egges, S. Kshirsagar, and N. Magnenat-Thalmann, "Generic personality and emotion simulation for conversational agents," *Computer animation and virtual worlds*, vol. 15, pp. 1–13, 2004.
- [13] J. Albeck and N. Badler, "Embodied autonomous agents," in *Handbook of virtual environments: design, implementation, and applications*. Mahwah, NJ: Lawrence Erlbaum Associates, 2002, pp. 313–332.
- [14] M. Argyle, *Bodily communication*. London, UK: Methuen, 1975.
- [15] P. Ekman and W. Friesen, *Facial Action Coding System: A Technique for the Measurement of Facial Movement*. Palo Alto: Consulting Psychologists Press, 1978.
- [16] R. Amini and C. Lisetti, "Hapfacs: An open source api/software to generate face-based expressions for ecas animation and for corpus generation," in *Proceedings of the 2013 Humaine Association Conference on Affective Computing and Intelligent Interaction (ACII)*. IEEE Press, 2013, pp. 270–275.
- [17] H. J. Eysenck, *The Biological Basis of Personality*. Springfield, IL: Thomas, 1967.
- [18] R. McCrae and P. Costa, "Personality trait structure as a human universal," *The American Psychologist*, vol. 52, no. 5, pp. 509–516, 1997.
- [19] J. McCoy, M. Mateas, and N. Wardrip-Fruin, "Comme il faut: A system for simulating social games between autonomous characters," in *Proceedings of the 8th Digital Art and Culture Conference (DAC)*, Irvine, CA, 2009.
- [20] D. Watson and L. Clark, "On traits and temperament: General and specific factors of emotional experience and their relation to the five factor model," *Journal of personality*, vol. 60, pp. 441–476, 1992.
- [21] F. Parke and K. Waters, *Computer facial animation*, 2008.
- [22] F. Mairesse and M. Walker, "Towards personality-based user adaptation: Psychologically informed stylistic language generation," *User Modeling and User-Adapted Interaction*, vol. 20, no. 3, pp. 227–278, 2010.
- [23] C. Hu, M. Walker, M. Neff, and J. F. Tree, "Storytelling agents with personality and adaptivity," in *Proceedings of Intelligent Virtual Agents*, W. Brinkman, J. Broekens, and D. Heylen, Eds., 2015.
- [24] H. Kersjes, *Modelling believable personalities in virtual agents*, ser. M.Sc. thesis. Tilburg, The Netherlands: Tilburg University, 2015.

# Heterogeneous Team Deep Q-Learning in Low-Dimensional Multi-Agent Environments

Mateusz Kurek, Wojciech Jaśkowski

Institute of Computing Science, Poznan University of Technology, Poznań, Poland

wjaskowski@cs.put.poznan.pl

**Abstract**—Deep Q-Learning is an effective reinforcement learning method, which has recently obtained human-level performance for a set of Atari 2600 games. Remarkably, the system was trained on the high-dimensional raw visual data. Is Deep Q-Learning equally valid for problems involving a low-dimensional state space? To answer this question, we evaluate the components of Deep Q-Learning (deep architecture, experience replay, target network freezing, and meta-state) on a Keepaway soccer problem, where the state is described only by 13 variables. The results indicate that although experience replay indeed improves the agent performance, target network freezing and meta-state slow down the learning process. Moreover, the deep architecture does not help for this task since a rather shallow network with just two hidden layers worked the best. By selecting the best settings, and employing heterogeneous team learning, we were able to outperform all previous methods applied to Keepaway soccer using a fraction of the runner-up's computational expense. These results extend our understanding of the Deep Q-Learning effectiveness for low-dimensional reinforcement learning tasks.

**Keywords:**

## I. INTRODUCTION

Deep learning [25] is currently a hot topic in machine learning. Deep neural networks, which are neural networks with many hidden layers [1], have been successfully applied to supervised learning problems involving high-dimensional data in the fields of image processing [14], speech recognition [4], or natural language processing [28].

Up until recently, however, deep neural networks were considered to exhibit unstable dynamics in reinforcement learning settings. Things have changed with the seminal DeepMind's paper [18], which demonstrated how to make deep reinforcement learning obtain human-level performance on a large set of Atari 2600 games. Their method, called Deep Q-Learning involved Q-learning, deep convolutional neural networks, and a set of additional techniques: experience replay with minibatches, target network freezing, a modified version of the RMSProp backpropagation algorithm, and merging subsequent states into a meta-state. The input to the learning system involved high-dimensional visual pixel data.

In this work, we ask the question whether the Deep Q-Learning techniques used for high-dimensional inputs are equally effective for reinforcement learning problems with low-dimensional states. To this aim, we evaluate the components of DeepMind's Deep Q-Learning on a challenging Keepaway soccer task.

Keepaway soccer [33] is a task within the RoboCup Soccer simulator, which goal is to control a team of players to

maintain the possession of the ball away from the opposing team. It presents many challenges to machine learning methods which include the continuous state space, uncertainty, multiple agents, and delayed effects of players' actions. The mechanics of the game are physically simulated, which makes the problem complex enough so it cannot be solved trivially. On the other hand, the simulation time is limited thus complete machine learning approaches are computationally feasible. This is why, it has been a popular benchmark for reinforcement learning methods, neuroevolution and, genetic programming.

The contributions of this work include an experimental analysis of the components of the DeepMind's Deep Q-Learning techniques on the Keepaway soccer task. We demonstrate that although experience replay with minibatches indeed improves the agent performance, target network freezing and meta-state slow down the learning process. Our results indicate that deep architectures do not help the agents and shallow networks with only two hidden layers are enough. We hypothesize that this is due to the low-dimensionality of the input data in Keepaway soccer.

In addition, as the Keepaway soccer is a multi-agent problem, we treat it as an opportunity to compare homogeneous and heterogeneous learning. The latter resulted in outperforming all previous methods applied to this task using only a fraction of the computational expense of the runner-up, Symbiotic Bid-based (SBB) GP [11]. The results indicate a considerable potential of (at least some of) the components of Deep Q-learning also when applied to low-dimensional problems.

## II. RELATED WORK

Reinforcement learning [36] has been applied in the past to fields such as robotics [13], operations research [21], or economics [19]. One of the first application of reinforcement learning is Tesauro's TD-Gammon [40], a computer program trained to play Backgammon at super-human level using temporal different learning and self-play. Other successful applications of reinforcement learning to games include Go [26], Othello [38], Chess [27], and 2048 [37].

The Keepaway soccer problem was first proposed by Stone et al. in 2001 [32]. Since then, it has been approached by a number learning methods [6], [33], [34], [30], [43], [31], [10], [7], [44], [23], [5], [2], [11], [12]. Below, we mention only the selected ones.

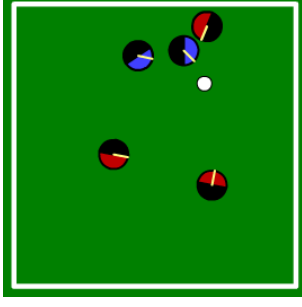


Figure 1. 3 vs. 2 Keepaway soccer

Stone et al. [31] used SARSA( $\lambda$ ) with different function approximators: linear tile-coding (CMAC), radial basis functions (RBF) and neural networks. Several work [29], [39] treated the problem with NeuroEvolution of Augmenting Topologies (NEAT) or HyperNEAT [42].

The best results for Keepaway soccer to date have been obtained by symbiotic bid-based (SBB) genetic programming [15] by Kelly et al [11]. A control policy is defined by a team of programs that coevolve, each specializing on a subcomponent of the task.

### III. KEEPAWAY SOCCER

Keepaway soccer is a problem within the RoboCup Soccer simulator [3], in which the task for one team (the keepers) is to maintain possession of the ball in a rectangular field, preventing another team (the takers) from taking it over [33]. Keepaway soccer is a challenging problem involving a continuous, partially observable state space, multiple agents, and noisy actions.

The most popular configuration in Keepaway soccer, which we focus on in this paper, involve  $n = 3$  keepers and  $m = n - 1 = 2$  takers playing in  $20\text{ m} \times 20\text{ m}$  region (see Fig. 1) but other settings have also been considered in the past.

At the beginning of the game, all the players are placed in predefined locations of the playing field. In each turn, they move or kick the ball. The game ends when the ball leaves the playing field or any of the takers take it over.

Each agent acts based on the information about relative distances and angles to the other objects in the world. The state consist of i) distances from the players to the center of the playing region; ii) distances from the keeper possessing the ball to its teammates; iii) distances of each keeper to its closest opponent; and iv) angles between the keeper possessing the ball, some other keeper and an opponent. For the 3 vs. 2 version, the state is described by 13 variables.

It is worth to note that the simulator introduces noise to the state variables.

In the simplified, standardized keepaway player framework, we consider here, only the keeper that currently possesses the ball can be controlled. The rest of the keepers follow a predefined behavior — they either wait for the ball or go towards the ball if none of its teammates is in the possession of the ball. The keeper possessing the ball can make one of  $n$

possible actions: it may hold the ball or pass it to one of its teammates. The movement of the keepers is controlled by an arbitrary algorithm.

The framework comes with two fixed policies for the players: *Random* (choose the action randomly) and *Hand-coded* (a.k.a. *All-to-ball*, always go towards the ball). In the standardized Keepaway soccer, the takers behave according to the *Hand-coded* policy.

The Keepaway soccer task is a reinforcement learning problem, in which rewards are provided for each simulator second of ball possession.

## IV. METHODS

### A. Reinforcement Learning

Reinforcement learning [36] is a class of Markov Decision Process (MDP). An MDP is a discrete-time, stochastic control process, defined as a quintuple  $(S, A, P, R, \gamma)$ , in which:

- $S$  is a finite set of states ( $s_t \in S$ , where  $s_t$  is state observed at (discrete) time  $t$ ),
- $A(s_t)$  is a finite set of possible actions in state  $s_t$  ( $a_t \in A(s_t)$ ,  $A(s) \subseteq A$ , where  $a_t$  is action executed at time  $t$ ),
- $P(s, a, s') = P(s_{t+1} = s' | s_t = s, a_t = a)$  is the probability, that executing action  $a$  in state  $s$  at time  $t$  will lead to state  $s'$  at time  $t + 1$ ,
- $R(s, a)$  is function describing the immediate (or expected immediate) reward for taking action  $a$  in state  $s$ ,
- $\gamma \in (0, 1]$  is the discount factor, which denotes the difference in priorities between the immediate reward and future rewards.

The solution to the MDP is a policy  $\pi : S \rightarrow A$ . The policy should maximize the expected discounted cumulative reward:

$$\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_{t+1}],$$

where  $r_t$  is a reward obtained in step  $t$ .

The reinforcement learning problem is an MDP in which the model of the environment ( $R$  or  $P$ ) is unknown.

### B. Q-Learning

Q-learning is a model-free, off-policy, temporal difference [35] algorithm to solve reinforcement learning problems. It learns the action-value function  $Q$  defined as:

$$Q^\pi(s, a) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0 = s, a_0 = a, a_t = \pi(s_t)].$$

The optimal  $Q = Q^{\pi^*}$  value fulfills the Bellman equation:

$$Q(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s, a, s') \max_{a' \in A(s')} Q(s', a') \quad (1)$$

for all  $s \in S$ . The optimal policy is greedy w.r.t to  $Q$ :

$$\pi^*(s) = \operatorname{argmax}_{a \in A(s)} Q(s, a)$$

Due to the search space size, the  $Q$  function has often to be approximated, which is done by choosing a family of functions  $Q_\theta$ , where  $\theta$  is a vector of parameters to learn.

To learn  $\theta$ , the agent gathers training examples by interacting with the environment according to its behavioral policy. One training example is a quadruple  $(s, a, r, s')$ . The agent iteratively updates  $\theta$  to minimize the error on the one-step prediction:

$$L_{\theta}(s, a) = \frac{1}{2} (Q_{\theta}(s, a) - Q_{target})^2,$$

where

$$Q_{target}(s, a) = R(s, a) + \gamma \max_{a' \in A(s')} Q_{\theta}(s', a').$$

Q-learning minimizes the loss function by following the gradient:

$$\theta \leftarrow \theta - \eta (Q_{\theta}(s, a) - Q_{target}(s, a)) \frac{\partial Q_{\theta}(s, a)}{\partial \theta}.$$

### C. $\epsilon$ -greedy Policy

During collecting the training examples, the agent follows a behavioral policy. As making actions greedily according to the current policy may lead to a local optima, a common choice is the  $\epsilon$ -greedy policy, which consists in making a greedy action w.r.t. to the current state-action function with probability  $1-\epsilon$  and a random one with probability  $\epsilon$ , where  $\epsilon \in (0, 1)$  is the exploration rate.

$\epsilon$  can be fixed during the learning but annealing it often leads to better results. Here, we decrease it linearly from 0.1 towards  $e_f \in (0, 0.1)$ , which is reached after the first  $K$  learning episodes.

### D. Deep Q-Networks

The action-value function can be approximated by a linear weighted function of hand-designed features. This has been a common choice since the Q-learning with a linear function approximator has converge guarantees [36]. Neural networks, which are nonlinear function approximators, are known to be notoriously unstable (or even to diverge) when coupled with Q-learning [41]. Their unstable behavior is especially evident when many hidden layers were used (deep architectures).

Recently, however, in a breakthrough paper, Mnih et al. [18] proposed a set of techniques, called Deep Q-Learning in the following, to stabilize the Q-learning with (deep) neural networks. The techniques include experience replay with minibatches, target network freezing, and Root Mean Squared Gradient (RMSProp) algorithm for backpropagation. We describe the techniques in the following sections.

1) *Experience Replay and Minibatches*: Experience replay is originally due to [16]. In this idea, the data used for learning is randomly sampled at each step from a memory of agent's previous transitions. In result, it removes the correlations in the sequence of training examples and smooths the training distribution over many past behaviors reducing oscillations of the learning process. Another advantage of this mechanism is the reuse of a single experience in many weights updates, which allows for greater data efficiency [18].

To perform the experience replay, agent's transitions at each time step  $t$  are stored in replay memory  $D$  as a tuple  $e_t =$

```

1: function DEEPQLearning
2:    $D \leftarrow \emptyset$  ▷ Replay memory
3:    $Q \leftarrow \text{RANDOMWEIGHTS}()$  ▷ initialize  $Q$ 
4:   while not stop condition satisfied do
5:      $s_0 \leftarrow \text{INITIALSTATE}()$ 
6:     for time step  $t$  in current episode do
7:       if  $\text{RANDOMVALUE}() < \epsilon$  then
8:          $a_t \leftarrow \text{RANDOMACTION}()$ 
9:       else
10:         $a_t = \arg \max_{a'} Q(s_t, a')$ 
11:       $r_t \leftarrow \text{EXECUTEACTION}(a_t)$ 
12:       $s_{t+1} \leftarrow \text{OBSERVESTATE}()$ 
13:      Store  $(s_t, r_t, a_t, s_{t+1})$  in  $D$ 
14:      for  $(s_i, r_i, a_i, s_{i+1})$  in  $\text{SAMPLE}(M, D)$  do
15:         $y_j = \begin{cases} r_i & \text{if } \text{terminal}(s_{i+1}) \\ r_i + \gamma \max_{a'} Q_{\theta}(s_{i+1}, a') & \text{otherwise} \end{cases}$ 
16:       $L \leftarrow \sum_{j=1}^M (y_j - Q_{\theta}(s_j, a_j))^2$  ▷ Backprop

```

Figure 2. Deep Q-Learning with experience replay and minibatches.

$(s_t, a_t, r_t, s_{t+1})$ , where  $s_t$  is state observed at time  $t$ ,  $a_t$  is action performed in state  $s_t$ ,  $r_t$  is the obtained reward, and  $s_{t+1}$  is the resulting state after taking action  $a_t$ . The replay memory  $D = \{e_1, e_2, \dots, e_n\}$  stores the last  $N$  such tuples.

Another technique, used together with experience replay, is *minibatch learning*, which consists in learning more than one training example at each step. It makes the learning process less prone to outliers and noises as the gradient computed at each step uses more training examples. At each step of the training, dataset  $D$  is sampled uniformly to get a minibatch of experiences of size  $M$  ( $(s, a, r, s') \sim U(D)$ ). The Deep Q-Learning algorithm with experience replay and minibatches is presented in Fig. 2.

2) *Target Q-Network Freezing*: Target Q-Network freezing [18] consists in maintaining two separate networks: i) a target network, called  $Q_{\theta^-}$  in the following, with a fixed set of 'old' parameters  $\theta^-$  for generating target Q-values, used in the Q-learning process, and ii) a network  $Q_{\theta}$  for interacting with the environment, with the current set of parameters  $\theta$ .

Target Q-values are calculated using the old set of parameters  $\theta^-$ :

$$Q_{target}(s, a) = R(s, a) + \gamma \max_{a'} Q_{\theta^-}(s', a')$$

At every update iteration, the current parameters  $\theta$  are updated to minimize the mean-squared Bellman error w.r.t the old parameters  $\theta^-$  by optimizing the following loss function:

$$L_{\theta} = \mathbb{E}_{s,a,r,s',i \sim D} [(R(s, a) + \gamma \max_{a'} Q_{\theta^-}(s', a') - Q_{\theta}(s, a))^2]$$

Every  $C$  steps, the parameters from the Q-network are copied to the target Q-network. Generating the learning targets using an old set of weights adds a delay between the time an update to  $Q$  is made and the time the update affects the targets, counteracting oscillations and divergence.

3) *Root Mean Squared Gradient (RMSProp)*: The most common learning method is stochastic gradient descent (SGD). SGD assumes that the learning rate is the same for each parameter being learned, which works poorly when the gradient values vary since for some components, it leads to large changes and tiny changes for the others. RProp [22] solves this problem by i) using only the sign of the gradient and ii) adapting the step size separately for each weight. Unfortunately, RProp does not work well with minibatches [8].

RMSProp [8] is an extension to RProp and SGD, which combines the robustness of RProp, efficiency of minibatches, and effective averaging of gradients over minibatches (unlike RProp). RMSProp keeps track of the previous gradients and divide the updates by the average magnitude of the gradient over the last several updates, which leads, in practice, to maintaining separate learning rate  $\eta_i$  for each weight. This allows modifying each parameter  $\theta_i$  according to its previous magnitudes, preventing from taking it into account with too large or too small weight. RMSProp is parametrized by gradient moving average decay factor  $\rho$  (0.9 by default), overall learning rate  $\eta$  and  $\varepsilon$  ( $10^{-6}$  by default) to maintain the numerical stability. The learning rate  $\eta_i$  at time step  $t$  is calculated as follows:

$$g_i^{(t)} = \rho g_i^{(t-1)} + (1 - \rho) \left( \frac{\partial L}{\partial \theta_i} \right)^2$$

$$\eta_i^{(t)} = \frac{\eta}{\sqrt{g_i^{(t)} + \varepsilon}}$$

Mnih et al. [18] introduced a slightly different version of the RMSProp algorithm, which, besides of using the squared gradients from the past, keeps track of the regular gradients as well:

$$h_i^{(t)} = \rho h_i^{(t-1)} + (1 - \rho) \frac{\partial L}{\partial \theta_i}$$

$$\eta_i^{(t)} = \frac{\eta}{\sqrt{g_i^{(t)} - (h_i^{(t)})^2 + \varepsilon}}$$

## V. EXPERIMENTS IN DEEP Q-NETWORK

In this section, we evaluate the components of the DQN framework in the context of the Keepaway Soccer task.

### A. Experimental Setup

In all experiments, we use the classical Keepaway setup with 3 vs. 2 players in a 20 m  $\times$  20 m region, with the *hand-coded* policy for takers. Learning time has been limited to 20 000 episodes. The controller *score* denotes the time of a single episode in simulator seconds. By controller *performance* we mean the expected score, which we estimate by averaging the duration of a number of episodes. To monitor the learning progress, every 500 learning episodes, we play 100 testing episodes. The performance of the final controller is evaluated on 1000 testing episodes.

The *learning time* is the time of the learning process measured in real-time units (minutes). The experiments were

Table I  
THE PARAMETER VALUES TESTED IN THE PRELIMINARY EXPERIMENTS.  
BEST FOUND VALUES WERE MARKED BOLD.

Parameter	Tested values
Discount factor $\gamma$	0.99, <b>1.0</b>
Learning rate $\eta$	0.001, 0.0005, <b>0.0001</b> , 0.00005, 0.00001
Exploration time $K$	5000, 10000, <b>15000</b> , 20000
Final exploration $e_f$	<b>0.01</b> , 0.1, 0.05

Table II  
INFLUENCE OF EXPERIENCE REPLAY AND MINIBATCH SIZE ON AGENT'S  
FINAL PERFORMANCE

Memory size	Minibatch size	Mean score [s]	Learning time [min]
1	1	$8.3 \pm 1.5$	$192 \pm 17$
10000	1	$6.2 \pm 0.7$	$154 \pm 12$
10000	4	$14.2 \pm 3.3$	$293 \pm 6$
10000	8	$14.3 \pm 3.0$	$344 \pm 14$
10000	16	$15.4 \pm 1.5$	$405 \pm 10$
10000	32	$17.8 \pm 1.8$	$500 \pm 15$
10000	64	$18.1 \pm 1.3$	$750 \pm 31$
10000	128	$17.7 \pm 1.0$	$620 \pm 21$

conducted with Intel® Core™ i7-950 3.7GHz. The Keepaway simulator is quite sophisticated thus it is the computational bottleneck of the experiment (the time required for evaluating the network or updating its weights is negligible).

Each experimental run was repeated 6 times unless stated otherwise.

### B. Preliminary Parameter Search

In preliminary trial-and-error experiments, we evaluated several numerical parameters of the learning framework using a 13-30-100-50-3 (three full-connected hidden layers) network. The network's weights were uniformly initialized by the He's method [9]. The tested and best-found parameter values are shown in Table I. These values are used in subsequent experiments.

In addition, we also have verified that increasing the number of learning episodes over 20 000 does not improve the score. Also, although decreasing the learning rate  $\eta$  over time is a common choice in Q-learning, we found out that this does not improve the results for this task.

### C. Experience Replay and Minibatches

In the first experiment, we investigated whether learning with experience replay and minibatches is more effective than on-line learning (with no experience replay and minibatch size equal to 1), how the size of the minibatch affects the overall performance, and whether the claims made in Section IV-D1 that using experience replay with minibatches increase the stability of the learning process are correct.

The results of the experiment are shown in Table II and Fig. 3. The baseline algorithm did not use experience memory

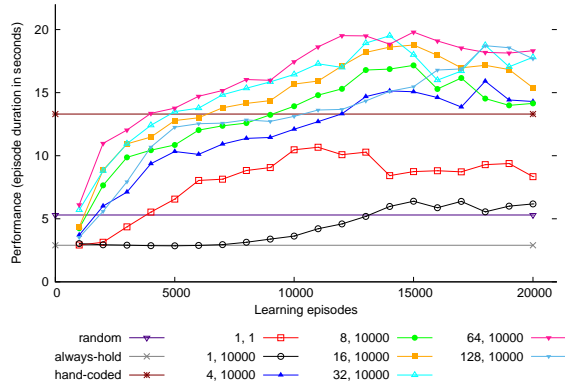


Figure 3. Influence of the experience replay and minibatches on the average episode duration.

(and thus experience replay). We can see that experience replay alone makes the learning process significantly slower. However, coupled with learning from minibatches, the results improve considerably. Already for minibatches of size 4, we observe a 70% performance improvement over the baseline. The larger the minibatch, the higher is the controller score, however, increasing the minibatch size from 64 to 128 does not help further. The difference in the learning time are mostly due to the episode durations of the experiments.

Importantly, except the minibatch of size 1, increasing the size of the minibatch decreases the standard deviation of the performance. It indicates that the minibatches stabilize the learning process.

Since the difference between the results with 32 and 64 sizes of the minibatch is minor while, at the same time, the experiments with minibatch size of 64 take approximately 1.5 times longer to run, in the subsequent experiments minibatch of size 32 with memory of size 10000 will be used (the minibatch of size 64 will be used only to learn champion policy).

#### D. Backpropagation Algorithms

The Deep Q-Learning used by the DeepMind team used its version of the Hinton's RMSProp. Does this backpropagation algorithm also pay off in our settings? Fig. 4 shows the results of the experiment conducted to answer this question. The experiment was performed using experience replay with minibatch of size 32 and the replay memory of size 10000. The results are not conclusive. The DeepMind's version seems slightly better and it is also characterized by a slightly smaller variance, but we found no statistically significant difference between the two versions of the RMSProp.

We will use the DeepMind's version of RMSProp in further experiments.

#### E. Target Network Freezing

In this experiment, different target network update frequencies were compared: every 100, 1000, and 10000 steps. Earlier research has shown (see Section IV-D2) that freezing weights of the target  $Q$ -network leads to higher stability of the

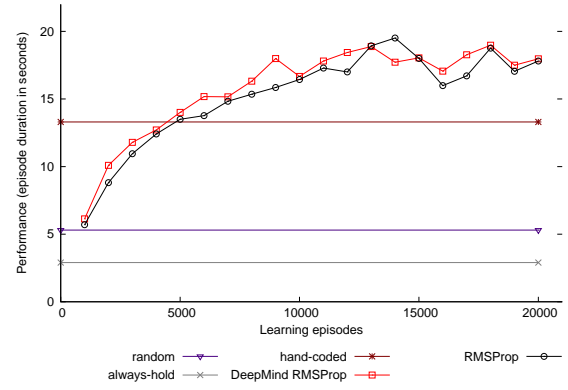


Figure 4. Comparison of backpropagation algorithms.

Table III  
TARGET NETWORK FREEZING.

Target network update frequency	Mean score [s]
no freezing (baseline)	$17.9 \pm 2.1$
100	$14.5 \pm 2.6$
1000	$16.9 \pm 3.7$
10000	$16.1 \pm 1.4$

learning process and lower divergence between the consecutive results returned by the neural network. The experiments were performed with minibatch of size 32, the replay memory of size 10000 and used DeepMind's RMSProp.

The results of the experiment are shown in Table III and Fig. 5. Contrary to the expectations, the results indicate that target network freezing does not provide any improvement over the baseline version, for which only one neural network is maintained. The differences between different update schemes are statistically insignificant. Moreover, as shown in Fig. 5, the neural network with the update frequency of 10 000 steps is learning significantly slower than the other ones. That is why this experiment was extended over 20 000 episodes.

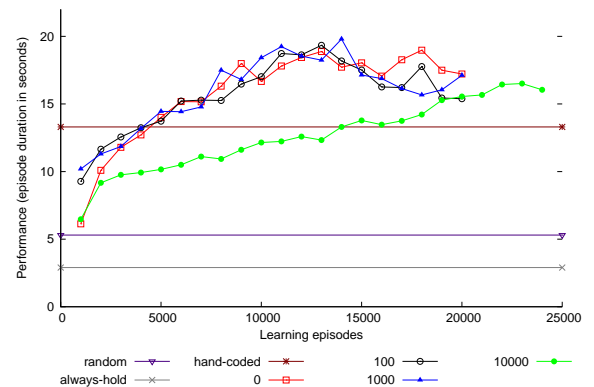


Figure 5. How target network update frequency influence on average episode duration.



Table IV  
NEURAL NETWORK ARCHITECTURES FOR KEEPAWAY SOCCER.

Neural network architecture	Mean score [s]
13-400-200-100-50-25-3	$16.2 \pm 5.0$
13-200-100-50-3	$15.8 \pm 0.5$
13-200-100-50-3	$17.1 \pm 1.0$
13-100-200-50-3	$16.9 \pm 2.0$
13-30-100-30-3	$17.9 \pm 2.1$
13-200-30-3	$17.8 \pm 1.6$
13-200-50-3	$18.1 \pm 2.2$

#### F. Neural Network Architectures

The goal of this experiment was to choose the best neural networks architecture for the Keepaway Soccer problem. During this experiment, six neural networks with two to five hidden layers with different numbers of nodes in the hidden layers were tested. All of these networks have the rectifier activation function in hidden layers and no activation function in the output layer. The input layer of the neural network always has 13 units (the size of Keepaway’s state) while the output layer consists of 3 neurons, each corresponds to the number of possible actions available to the keeper that currently possesses the ball.

All experiments were performed with experience replay with minibatch of size 32, the replay memory of size 10000, DeepMind’s RMSProp and no target network freezing.

The results of this experiment are shown in Table IV. The differences between the architectures of neural networks are relatively small. We also have found that in the initial phase of learning, deep neural networks have slightly better performance than the shallow ones, but the overall trend shows no advantage of using deep architectures. The best result was obtained by the smallest neural network 13-200-50-3 consisting of only two hidden layers.

No advantage of deep architectures caused by the low dimensionality of the input state. The results suggest that there is no need to model complex non-linear relationships in a problem with such simple input as Keepaway Soccer or the learning algorithms used are not able to effectively make use of it.

#### G. Meta-State Learning

Since the Keepaway Soccer is partially observable, in the next experiment, we check whether combining multiple consecutive states into a single *meta-state* leads to better performance. A similar technique was effective for deep reinforcement learning in the video game playing domain [18].

To this aim, consecutive states are merged into a single vector of size  $13n$ , where  $n$  is the number of recent states (meta-state size). This should, theoretically, provide the agent more information about the actual state of the environment. For example, the agent could reason whether the takers are moving towards him.

Meta-states of sizes 2 and 4 were tested. All trials were performed with minibatch of size 32, the memory of size

Table V  
META-STATE LEARNING

Meta-state size	Mean score [s]
1 (baseline)	$18.1 \pm 2.2$
2	$12.0 \pm 2.2$
4	$9.4 \pm 2.1$

Table VI  
HOMOGENEOUS AND HETEROGENEOUS TEAM LEARNING (THE FINAL EVALUATION OF 25 RUNS).

Team learning	Mean score [s]	Median score [s]
homogeneous	$18.48 \pm 1.30$	18.23
heterogeneous	$19.6 \pm 1.1$	19.49

10000, DeepMind’s RMSProp, no target network freezing and with the 13-200-50-3 neural network architecture.

The results of the experiment are shown in Table V. Unexpectedly, combining the consecutive states into a single meta-state, significantly decreases the agent’s performance compared to the baseline. This might be caused by the agent’s inability to either extract valuable information from the meta-state, or relate consecutive values of variables.

#### VI. TEAM LEARNING

In the previous experiments, we learned a single policy for the three keepers (homogeneous team learning). They were using the same experience replay memory and were modifying weights of the single neural network. In the final experiment, we ask the question whether it pays off to learn a separate policy for each agent, which is often called *heterogeneous team learning* [20]. In such settings, each agent has its own experience replay memory and its own neural network. Theoretically, the results should be the same as when using homogeneous team, since the optimal policy is the same for each agent. Nevertheless, when function approximation is used, the theoretical optimal policy is, in general, not achievable, thus learning three neural networks might, in practice, improve the performance of the team. Note that the agents have no possibility of (direct) communication except observing the behavior of the other keepers.

For the experiment, we use the best settings found in the previous sections, that is, the minibatch of size 64, the replay memory of size 10000, DeepMind’s RMSProp backpropagation, no target network freezing, no meta-state, and the 13-200-30-3 network architecture. To more precisely evaluate the learning method, this time, 25 individual learning runs were performed.

The results of the experiment are shown in Fig. 6, and Table VI. As expected, the final results of both approaches are close to each other, but statistically, the heterogeneous team learning is performing significantly better than homogeneous team learning (t-test,  $\alpha = 0.05$ ). What is worth noticing is the learning speed (see Fig. 6). The heterogeneous learning

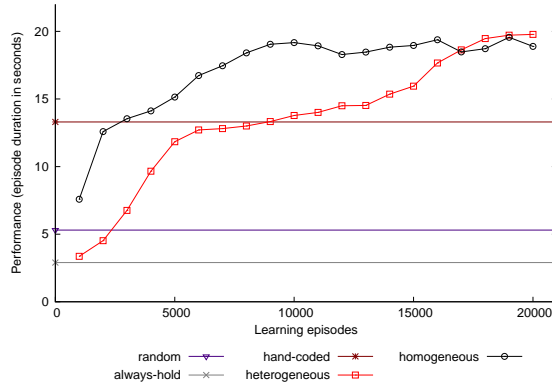


Figure 6. Dynamics of the team learning.

Table VII

COMPARISON OF THE APPROACHES TO KEEPAWAY SOCCER. THE “±” SIGN PRECEDES THE STANDARD DEVIATION.

Method	Mean score [s]	Learning hours
Always Hold [31]	$2.9 \pm 1.0$	-
Random [31]	$5.3 \pm 1.8$	-
SARSA with Neural Network [31]	$10.1 \pm 0.3$	30
Hand-coded [31]	$13.3 \pm 8.3$	-
NEAT [39]	$14.1 \pm 1.8$	800
SARSA with RBF [31]	$14.2 \pm 3.1$	30
EANT [17]	$14.9 \pm 1.3$	200
HyperNEAT [42]	$15.4 \pm 1.3$	50-200
SARSA with CMAC [31]	$15.7 \pm 2.8$	30
SBB with diversity [11]	$18.5 \pm 1.9$	1739
(heterogeneous) Deep Q-Learning	$19.6 \pm 1.1$	56.5

is significantly slower than in the homogeneous one. This is because heterogenous teams need to learn 3 times more parameters.

## VII. COMPARISON WITH OTHER APPROACHES

The comparison of all the previous results in the Keepaway soccer domain together with our Deep Q-Learning approach is shown in Table VII. It indicates that our method outperforms all of the previously published results in terms of the average keepers’ possession time (the mean score). Notice that our method is also characterized by relatively a low variance.

Comparing Deep Q-Learning to the runner up, the best published to date method (SBB with diversity [11]), we observe that, although the presented approach is only slightly better in terms of the mean score, importantly, it needed 30 times less learning time (the simulator’s hours) to achieve this score. What is more, SBB is conceptually more sophisticated than the deep reinforcement learning and, unlike the deep Q-Learning, it requires providing some domain knowledge, i.e., designing the genetic programming operators.

## VIII. SUMMARY AND CONCLUSIONS

In this paper, we evaluated the components of Deep Q-Learning on a challenging multi-agent task of the Keepaway soccer, which involves, in contrast to the original general

video game playing application of Deep Q-Learning, low-dimensional states.

The results of the experiments showed that some of the deep learning techniques, indeed, increase the performance of the agent, while the others have negative effect. In particular, experience replay and minibatch learning significantly improve the results. RMSProp makes Q-learning for this problem possible since SGD did not converge at all. However, the DeepMind’s RMSProp was found not better than the classical one. Target network freezing did not improve the results. On the contrary, when the target network update is rare, the learning process slows down considerably. Also, composing several subsequent states into a single meta-state (to alleviate the effect of partial observability) does not pay of for Keepaway. Finally, we found out that shallow, two-hidden-layer neural networks are enough for this task. Deep architectures do not improve the results. We speculate that it is due to the low-dimensionality of the Keepaway’s state space.

In addition, for the Keepaway soccer it is profitable to use heterogeneous team learning. Despite making the learning initially slower than the homogeneous one, it lead to better results, eventually.

The results of the experiments demonstrate that Deep Q-Learning applied for Keepaway soccer performs better than any other previously published method. Not only it is better in terms of the average score but it also uses significantly less computation effort to learn its policy. Compared to the best to date result, achieved by genetic programming (SBB) agent [11], the Deep Q-Learning agent outperforms it using only 1/30 computation time required by SBB.

The experiments confirmed that deep reinforcement learning is a suitable and effective method not only for highly-dimensional problems (that need to utilize convolutional neural networks) but also for low-dimensional problems such as Keepaway soccer.

There are many aspects of the proposed approach that are worth further investigation. First, experience replay is limited in some respects — the memory does not differentiate the important transitions from the unimportant ones. A more sophisticated sampling strategy such as prioritized experience replay might be considered [24]. Second, the presented agent is trained to play on the  $20\text{ m} \times 20\text{ m}$  region and it is evaluated in the same environment. Further research can investigate how well does the learned agent generalize to other sizes of the playing field [11]. Finally, some of the plots shown in Section V contain traces of unstable or cycling dynamics around 12000-16000 learning episodes. This effect requires further research.

## ACKNOWLEDGMENT

W. Jaśkowski acknowledges the support from the Polish National Science Centre grant no. DEC-2013/09/D/ST6/03932.

## REFERENCES

- [1] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *Lecture Notes in Computer Science (including*

subseries *Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics*), 7700 LECTU:437–478, 2012.

- [2] Luiz A. Celiberto, Jackson P. Matsuura, Ramón López De Mántaras, and Reinaldo A C Bianchi. Reinforcement learning with case-based heuristics for RoboCup soccer keepaway. *Proceedings - 2012 Brazilian Robotics Symposium and Latin American Robotics Symposium, SBR-LARS 2012*, pages 7–13, 2012.
- [3] Mao Chen, Ehsan Foroughi, Fredrik Heintz, ZhanXiang Huang, Spiros Kapetanakis, Kostas Kostiadis, Johan Kummeneje, Itsuki Noda, Oliver Obst, Pat Riley, Timo Steffens, and Xiang Yin Yi Wang. The robocup soccer simulator. <http://sourceforge.net/projects/sserver/>, 1997–2015. [Online; accessed 12-September-2015].
- [4] Li Deng, Jinyu Li, Jui-Ting Huang, Kaisheng Yao, Dong Yu, Frank Seide, Mike Seltzer, Geoffrey Zweig, Xiaodong He, Julia Williams, et al. Recent advances in deep learning for speech research at microsoft. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8604–8608. IEEE, 2013.
- [5] Sam Devlin, Marek Grzes, and Daniel Kudenko. Multi-agent, reward shaping for robocup keepaway. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 3, AAMAS '11*, pages 1227–1228, Richland, SC, 2011. International Foundation for Autonomous Agents and Multiagent Systems.
- [6] Anthony Di Pietro, Lyndon While, and Luigi Barone. Learning in RoboCup Keepaway using Evolutionary Algorithms. 2002.
- [7] Yang Gao and Francesca Toni. Argumentation-based reinforcement learning for robocup soccer takeaway. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems, AAMAS '14*, pages 1411–1412, Richland, SC, 2014. International Foundation for Autonomous Agents and Multiagent Systems.
- [8] Kevin Swersky Geoffrey Hinton, Nitish Srivastava. rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning. [http://www.cs.toronto.edu/tijmen/csc321/slides/lecture\\_slides\\_lec6.pdf](http://www.cs.toronto.edu/tijmen/csc321/slides/lecture_slides_lec6.pdf), 2012.
- [9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.
- [10] Tobias Jung and Daniel Polani. Learning RoboCup-Keepaway with kernels. *Journal of Machine Learning Research - Proceedings Track*, 1:33–57, 2007.
- [11] Stephen Kelly and Malcolm I. Heywood. Genotypic versus behavioural diversity for teams of programs under the 4-v-3 keepaway soccer task. pages 3110–3111, 2014.
- [12] Stephen Kelly and Malcolm I Heywood. On Diversity , Teaming , and Hierarchical Policies : Observations from the Keepaway Soccer Task. *EuroGP*, pages 75–86, 2014.
- [13] Jens Kober and Jan Peters. Reinforcement learning in robotics: A survey. In *Reinforcement Learning*, pages 579–610. Springer, 2012.
- [14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, page 2012.
- [15] Peter Lichodziejewski and Malcolm I. Heywood. Managing team-based problem solving with symbiotic bid-based genetic programming. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, pages 363–370, New York, NY, USA, 2008. ACM.
- [16] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3-4):293–321, 1992.
- [17] Jan H Metzen, Mark Edgington, Yohannes Kassahun, and Frank Kirchner. Analysis of an evolutionary reinforcement learning method in a multiagent domain. {AAMAS} '08: *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, (Aamas):291–298, 2008.
- [18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei a Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.
- [19] John Moody and Matthew Saffell. Learning to trade via direct reinforcement. *Neural Networks, IEEE Transactions on*, 12(4):875–889, 2001.
- [20] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.
- [21] Scott Proper and Prasad Tadepalli. Scaling model-based average-reward reinforcement learning for product delivery. In *Machine Learning: ECML 2006*, pages 735–742. Springer, 2006.
- [22] Martin Riedmiller. Rprop-description and implementation details technical report, january 1994.
- [23] Toru Sawa and Toshihiko Watanabe. Learning of keepaway task for RoboCup soccer agent based on Fuzzy Q-Learning. *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, pages 250–256, 2011.
- [24] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2015.
- [25] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.
- [26] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.
- [27] Ioannis E. Skoulakis and Michail G. Lagoudakis. Efficient Reinforcement Learning in Adversarial Games. In *2012 IEEE 24th International Conference on Tools with Artificial Intelligence*, pages 704–711. IEEE, nov 2012.
- [28] Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank, 2013.
- [29] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evol. Comput.*, 10(2):99–127, June 2002.
- [30] Peter Stone. Reinforcement Learning for RoboCup Soccer Keepaway. *Adaptive Behavior*, 13:165–188, 2005.
- [31] Peter Stone, Gregory Kuhlmann, Matthew E Taylor, and Yaxin Liu. Keepaway Soccer: From Machine Learning Testbed to Benchmark. *Lncs*, 4020:93–105, 2006.
- [32] Peter Stone and Richard S. Sutton. Scaling reinforcement learning toward RoboCup soccer. *Icml*, (June):537–544, 2001.
- [33] Peter Stone and Richard S Sutton. Keepaway Soccer: A Machine Learning Testbed. *Lecture Notes in Computer Science*, 2377:207–237, 2002.
- [34] Peter Stone, Richard S Sutton, and Gregory Kuhlmann. Reinforcement Learning for RoboCup-Soccer Keepaway. *Adaptive Behavior*, 13:165–188, 2005.
- [35] Richard S Sutton and Andrew G Barto. *Reinforcement Learning : An Introduction*. 2012.
- [36] R.S. Sutton and A.G. Barto. *Reinforcement learning*, volume 9. MIT Press, 1998.
- [37] Marcin Szubert and Wojciech Jaskowski. Temporal difference learning of n-tuple networks for the game 2048. In *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*, pages 1–8. IEEE, 2014.
- [38] Marcin Szubert, Wojciech Jaskowski, and Krzysztof Krawiec. On scalability, generalization, and hybridization of coevolutionary learning: a case study for othello. *IEEE Transactions on Computational Intelligence and AI in Games*, 5(3):214–226, 2013.
- [39] Matthew E Taylor, Shimon Whiteson, and Peter Stone. Comparing evolutionary and temporal difference methods in a reinforcement learning domain. *Proceedings of the 8th annual conference on Genetic and evolutionary computation GECCO 06*, (July):1321, 2006.
- [40] Gerald Tesauero. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68, March 1995.
- [41] John N. Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. Technical report, IEEE Transactions on Automatic Control, 1997.
- [42] Phillip Verbancsics and Kenneth O Stanley. Evolving Static Representations for Task Transfer. *Journal of Machine Learning Research*, 11:1737–1769, 2010.
- [43] Shimon Whiteson, Nate Kohl, Risto Miikkulainen, and Peter Stone. Evolving keepaway soccer players through task decomposition. *Genetic and Evolutionary Computation Conference 2003, Proceedings of the*, 59(1):356–368, 2005.
- [44] Shimon Whiteson, Matthew E. Taylor, and Peter Stone. Critical factors in the empirical performance of temporal difference and evolutionary methods for reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 21:1–35, 2010.

# Transfer Learning for Cross-Game Prediction of Player Experience

Noor Shaker, *Member, IEEE*  
Aalborg University  
Copenhagen, Denmark  
Email: noor.shaker@gmail.com

Mohamed Abou-Zleikha  
Independent Researcher  
Denmark  
Email: mhdaz81@gmail.com

**Abstract**—Several studies on cross-domain users’ behaviour revealed generic personality trails and behavioural patterns. This paper, proposes quantitative approaches to use the knowledge of player behaviour in one game to seed the process of building player experience models in another. We investigate two settings: in the *supervised feature mapping* method, we use labeled datasets about players’ behaviour in two games. The goal is to establish a mapping between the features so that the models build on one dataset could be used on the other by simple feature replacement. For the *unsupervised transfer learning* scenario, our goal is to find a shared space of correlated features based on unlabelled data. The features in the shared space are then used to construct models for one game that directly work on the transferred features of the other game. We implemented and analysed the two approaches and we show that transferring the knowledge of player experience between domains is indeed possible and ultimately useful when studying players’ behaviour and when designing user studies.

## I. INTRODUCTION

The game market has witnessed a drastic change over the past few years especially with the raise of mobiles as a new platform. This naturally led to more data about players which in turn attracted the attention of data analytics experts. Their goal is to benefit from the wealth of the data available to build tools and knowledge that could potentially help game companies make better games and decisions.

The wealth of data has also gathered interest from researchers working on analysing and modelling players’ behaviours. While some advances have been made through existing work on modelling and understanding players, there is one important limitation of current player experience modelling approaches: it is heavily context dependent, i.e. to understand how players react and play in a new game, most models need to be rebuilt from scratch, using freshly collected data for the new game. Labelled data is what is usually important for modelling player behaviour. Obtaining labels is usually labour and time consuming and in many cases it is expensive, if not impossible, to gather the needed data and train the models with every new game. Consequently, machine learning methods can be utilised to reduce the amount of new data needed, and/or to facilitate ways in which knowledge learned from previous data in one game can be transformed to another. This would ultimately lead to reusable models and faster learning. We argue that this should be the case for all domains where knowledge is transferred and used for predicting player experience.

indeed exists generic behaviours that transform across games and genres [1], [2], [3].

The notion of generic users’ behaviour has seeded many studies on *Generic User Modeling* systems [1], [4]. So far, this field has attracted limited attention from researchers within the computer games domain. In this paper, we use the term *Generic Models of Player Experience* (GMPE) to describe models that can be efficiently used to model player experience in more than one game. The goal is to utilise knowledge about players’ behaviour in one domain to seed the process of understanding players’ experience in another. We refer to this process as *transfer learning*.

In our preliminary work [5], generic models of player experience are build by manually selecting behavioural features that works well when predicting experience in two games. Infinite Mario Bros. (IMB) and a First-Person Shooter (FPS) game were used as testbeds. A subset of the behavioural features collected was carefully chosen by game design experts. The goal was to create a new set of abstract features that contains only those applicable in both games (the feature indicating the number of enemies killed is a classical example of an abstract feature). Models of player experience were then built from the resultant set. The approach was promising in terms of the accuracies obtained but one of its main limitations is that it is labour-intensive requiring human experts to go through all the collected features and to identify rational mapping from one game space to another. Our goal in this work is to overcome this limitation by automatically building models that perform well in modelling player experience in more than one game.

More specifically, we are trying to answer the two following questions:

- Are there features that are generally important to model player experience across game genres? And if so,
- can machine learning methods be used to identify such features? and
- how can we implement methods that use the knowledge about one domain to learn about a similar other?

To the best of our knowledge, our work is novel in two ways: 1) It is the first time transfer learning methods are employed to predict player experience in two games; and (2) it is the first time data-driven mapping of behaviour features is presented and used for predicting player experience.

To achieve our goals, we experimented with two settings. In the *supervised feature mapping* approach, we train the models to automatically learn a mapping from the features of the one game dataset (called the source) to another (the target). The goal is to identify relationships between the features so that the same models can be used to predict PE in both datasets simply by substituting one feature set with the other. The approach is supervised as it requires labeled datasets for training. In the second method, *unsupervised transfer learning*, we assume we have labels for the source dataset and only behaviour information for the target (no labels). We first project the features of the source and the target into a shared space by identifying correlations between the features. The resultant shared features are then used to optimise PE models on both datasets.

The first setting is useful when we have labels available for both datasets, and we are interested in preserving the meaning of the features and understanding the similarity between the games. In the unsupervised scenario, we assume we have labeled instances for one dataset and we rely on the correlation between the features to establish a transformation. This setting is interesting when we want to minimise the need of collecting labels for a new game.

We would like to note that in this paper, we are mainly concerned about establishing an accurate mapping among the features and less about whether the mapped features are semantically related. For instance, the method might discover that the number of deaths in Super Mario Bros. and the shooting accuracy in a FPS game are similarly correlated with frustration although they are semantically different.

## II. RELATED WORK

### A. Player Experience Modelling

There is an abundance of studies presented in the literature on constructing computational models of emotion [6], [7], [8], [9]. There are also theories about user emotion applied specifically in the games domain [3], [2]. Estimating affective and cognitive states in conditions of rich human-computer interaction, such as in games, is a field of growing academic and commercial interest. Several studies with varying success can be found on constructing models of Player Experience (PE) in different game genres independently [10], [11]. While the PE Models (PEMs) constructed in some of these studies achieved reasonable rates of accuracy (70-90%), they are still confined to predicting PE of a specific game and they have not yet put to use or investigated in cross-games studies.

Different studies on how people play games revealed a number of behavioural playing patterns and generic personality traits [2], [3]. These studies support our argument that knowledge about player behaviour in one game can be transferred to another. This in turn suggests that there exists features that can be used to model PE across games. If we are to identify such features, the process of collecting new data for each new game and designing new user studies will be drastically improved. The process could be optimised so that the

inform the process of new data collection. There will also be no need to reinitiate the process for every new game. Alternatively, knowledge gained from previous experiments could be transferred to the new domain and ultimately boosts new experiments.

### B. Transfer Learning

Individuals from academia and the games industry have invested in collecting and understanding gameplay data. The majority of research on player behaviour analysis however has been confined to individual games. This is in part due to the lack of publicly available datasets that cover more than one game, with only a few exceptions [12], [13]. Moreover, although access to behavioural data can sometimes be granted, obtaining clean and labelled instances is usually a tedious process. Therefore, most cross-games studies rely on clustering, pattern extractions and other unsupervised learning approaches that do not require users' annotation [12], [14], [15].

Approaches to understanding players' behaviour across games have so far been limited to analysing play traces, i.e. sequences of player actions corresponding to one play of the game [16]. This is done through the use of an edit distance metric and its application is limited to cases where the full trace of player action is recorded. In other studies [17], [18], the authors used features that are generic by definition, such as play time, session frequency and session length, to predict player departure. Although this direction is interesting, the input space used is rather limited (as the method only applies to scalable features) and consequently, the method does not allow thorough insights about more in-depth relationships between behavioural features. One natural extension to the aforementioned approach is to identify other features that might not be intuitively generic, but can be empirically extracted through data-driven approaches. This could be done through defining a fitness measure that captures the significance of the relationship between the features. The fitness measure we use in this paper is how well the features work together when predicting players' experience.

## III. DOMAIN AND DATA DESCRIPTION

In the following we give a brief description of both games used in our study and the data collected.

### A. Infinite Mario Bros.

SMB is a very popular 2D platform game published by Nintendo. An open source clone of the game, called Infinite Mario Bros (IMB) is used in this study as well as in many other previous ones [11], [19], [20]. The clone is modified to permit control over level generation and thereafter provide different variations of content for players to experience and compare. In this study, we created variations through differing the values of six chosen content features including the number of gaps, enemies, coins etc. The process of parametrising content generation lead to 40 different levels that vary in at least one aspect of the chosen features. More information about game parametrisation and content can be found in [11].

### B. First-Person Shooter: Sauerbraten

A second dataset from a FPS game is used. The game is from a different game genre and is built on the 3D game engine *Cube*. The game in our experiments is played in the single player mode and the goal is to collect the highest score possible by traversing the arena for two minutes, killing as many of the enemies as possible and avoid being hit. The player is equipped with a weapon and she can collect other types of weapons and resources. Every time the player is killed, she loses one point and she is re-spawned again as long as she still has time left to play. The same procedure for generating content variations as in IMB is followed: important content features expected to impact the game experience are identified by game experts. Each identified feature is given two possible values that allow clear distinction between game variants. The combinations of all possible values assigned for all chosen features are then used to create the game levels used for data collection. We chose four features for content generation, which results in 16 different game variations.

### C. Study Procedure

Using the content generated for both games, two independent datasets were collected containing different number of participants. Game surveys were conducted to collect information about players' demographics as well as their interaction with the games along with their reported affective states. The same protocol is followed for data collection in both datasets. According to this protocol, players are presented with a pair of two levels that differ along one or more aspects of game content. While playing, detailed information about player behaviour and actions were recorded. After playing each pair, players were asked to report their emotional/behavioural states following the four-alternative forced choice protocol: Pairwise preferences were adopted where the questionnaires presented are of the form: "Which game was more *E*?" where *E* can be on of the three states: *engagement*, *frustration* and *challenge*. The possible answers are: (1) game A [B] was more/less *E* than game B [A] (2) both equally or (3) neither. Full details about the procedure followed can be found in [11], [21]

In what follows, we describe the procedure followed to collect the data and the characteristics of each dataset.

1) *Dataset 1: Super Mario Bros*: An executable version of the software was uploaded online and participants were invited to play the game and answer the questionnaire. The final dataset consists of a total of 273 unique players who played 780 game pairs (1560 game levels). The data was preprocessed to remove the pairs in which players reported unclear preferences (their answers were neither or both equally), and after this step, the number of pairs remained were 597, 531 and 629 for engagement, frustration and challenge, respectively. Several representative features of player behaviour were extracted and Table I presents a subset of these features. The full set contains 30 features which can be found in [11].

2) *Dataset 2: First-Person Shooter*: A data collection event was organised and advertised over social networks where people are invited to participate. The event was held at the university over two days and a total number of 62 students participated. Each player was asked to play at least one pair (two game sessions, each last for two minutes) and she can play as many pairs as she wants. The final dataset consists of 124 pairs and after removing the pairs with unclear preferences 115, 111 and 112 pairs remained for engagement, frustration and challenge, respectively. Players' behavioural features were extracted as indicators of players' style. Table I presents a subset of these features (the total set contains 23 features). The context features presented are the ones used to design the variations of the game content presented to the players (full details of the procedure can be found in [21]).

Notice that although the games are from two different genre, they share some characteristics that might help the search for generalised features. In other settings, where the similarities between the games are less obvious (such as when comparing a game like Civilization to SMB.), the methods we propose might help discover correlations between the features that are unexpected. This however requires further future investigations.

## IV. PREFERENCE LEARNING WITH MULTIVARIATE ADAPTIVE REGRESSION SPLINE MODEL

Multivariate adaptive regression spline (MARS) is a popular nonparametric model proposed by Friedman to solve regression problems [22]. The key idea is to segment the space of inputs into regions of varying sizes that can be fit with linear or cubic splines. Each of the regions has its own regression submodel and the size of the regions is learned by the model given the layout of the input. During the learning process, MARS attempts to fit adaptive non-linear regressions to define relationships between a response variable and some set of predictors through a forward/backward iterative approach. The adaptive non-linear regression uses piecewise basis functions (also known as terms) that are defined in pairs, using a knot or value of a variable that defines an inflection point along the range of a predictor. These knots (parameters) are also empirically learned.

Practically speaking, suppose we have the input data  $X = [x_1, \dots, x_n]$  where  $n$  is the size of the input space, MARS is defined as:

$$y = f(X) = \beta_0 + \sum_{i=1}^M \beta_i f_i(X) \quad (1)$$

The model is a weighted (by  $\beta_i$ ) sum of basis functions  $f_i(X)$  where  $M$  is the number of the basis functions. The function  $f(X)$  is defined as:

$$f_i(X) = \prod_{k=1}^K h_{X_v, k}(X_v) \quad (2)$$

where  $X_v$  is the variable with index  $v$  and  $K$  is the number of terms in the function ( $k \in [1..K_{max}]$ ). Setting  $K_{max} = 1$



TABLE I  
FEATURES EXTRACTED FROM SUPER MARIO BROS AND FIRST PERSON SHOOTER DATA RECORDED.

Category	Feature	Description
Infinite Mario Bros		
Time	$t_{comp}$	Completion time
	$t_{play}$	Playing duration of last life over total time spent on the level
	$t_{jump}$	Time spent jumping (%)
	$t_{left}$	Time spent moving left (%)
	$t_{right}$	Time spent moving right (%)
	$t_{run}$	Time spent running (%)
	$t_{small}$	Time spent in Small Mario mode (%)
	$t_{big}$	Time spent in Big Mario mode (%)
Interaction with items	$n_{coin}$	Free coins collected (%)
	$n_{coinBlock}$	Coin blocks pressed or coin rocks destroyed (%)
Interaction with enemies	$k_{goomba}$	Times the player kills a goomba or a koopa (%)
	$k_{stomp}$	Opponents died from stomping (%)
Death	$d_{total}$	Total number of deaths
	$d_{cause}$	Cause of the last death
Miscellaneous	$n_{mode}$	Number of times the player shifted the mode (Small, Big, Fire)
	$n_{jump}$	Number of times the jump button was pressed
First Person Shooter		
Context features	$E$	Number of enemies
	$G$	Number of gaps
	$G_w$	Average width of gaps
	$E_p$	Placement of enemies
	$t_{life}$	Duration of play
	$t_{weapon}$	Time spent using weapons (%)
	$t_{shoot}$	Time spent shooting (%)
Time	$t_{still}$	Time spent not moving (%)
	$t_{jump}$	Time spent jumping (%)
	$t_{exp}$	Time spent using explosive weapons(%)
	$n_{health}$	Health items collected (%)
	$n_{armour}$	Armours collected (%)
Interaction with items	$e_{kill}$	Number of times the player kills an enemy (%)
	$p_{hit}$	Number of times the player receives a hit from an enemy (%)
	$e_{hit}$	Number of times the player hits an enemy (%)
Interaction with enemies	$n_{death}$	Number of times the player died
	$s_{acc}$	Shooting accuracy
Miscellaneous	$E$	Number of enemies
	$E_{skill}$	Skill level of enemies
	$W_{type}$	Type of weapons including explosive and non-explosive weapons
	$H$	Number of health items
	$R$	Number of resources such as bullets and armors

makes the model is additive only, and for  $K_{max} = 2$  the model allows a maximum of two variables multiplication.  $h_{X_v,k}(X_v)$  represents the basis function used to build the model. There are a number of basis functions that are usually defined. The most used are the hinge function and product of a set of hinge functions which are defined as:

$$h_{x,k}(x) = \max(0, x - t_{x,k}) \quad \text{or} \quad \max(0, t_{x,k} - x) \quad (3)$$

where  $t_{x,k}$  is a constant called *knot*. The *knot* value defines the "pieces" of the piecewise linear regression which is also determined from the data of variable  $x$ .

The method works in two main steps: in the forward stage, an increasingly larger number of basis functions are added to the model so that least squares goodness-of-fit criterion is maximised. As this might yield over-fitting, a backward procedure is then applied; the model is pruned by removing those basis functions that are associated with the smallest increase in the goodness-of-fit. This is done using the

measure of the goodness of fit that takes into account both the residual error and the model complexity:

$$GCV = \frac{\sum_{i=1}^n (y_i - f(x_i))^2}{(1 - \frac{\hat{c}}{n})^2} \quad (4)$$

where  $n$  is the number of samples in the training data and  $\hat{c}$  is the effective number of parameters and is calculated as:

$$\hat{c} = c + \frac{p * (c - 1)}{2} \quad (5)$$

where  $c$  is the number of independent basis function, and  $p$  is the penalty of adding a basis function.

## V. EVOLVING MARS MODELS

Grammatical Evolution (GE) is an evolutionary algorithm based on Grammatical Programming (GP) [23]. GE relies on a linear genome representation. The population of the evolutionary algorithm is initialised randomly consisting of variable-length integer vectors; the syntax of possible solutions is specified through a context-free grammar. GE uses

```

<Tree> := <Constant> + <Exp>
<Exp> := <Constant> * <BasisFunction> + <Exp>
        | <Constant> * <BasisFunction>
<BasisFunction> := <HingeFunction> * <BasisFunction>
        | <HingeFunction>
        | <Empty>
<HingeFunction> := max(0, <Feature> - <Knot>)
        | max(0, <Knot> - <Feature>)
<Feature> := feature1 | feature2 | ...
<Constant> := [min, max]
<Knot> := [0, 1]
<Empty> := []

```

Fig. 1. The grammar employed to specify the structure of MARS model.

the grammar (usually written in Backus Naur Form (BNF)) to guide the construction of the phenotype output.

Each chromosome in GE is a vector of codons. Each codon is an integer number used to select a production rule from the BNF grammar in the genotype-to-phenotype mapping. A complete program is generated by selecting production rules from the grammar until all non-terminal rules are mapped. The resultant string is evaluated with a fitness function to give a score to the genome.

In this paper, a Design Grammar (DG) is defined to specify the structure of possible solutions (MARS models in our case) as can be seen in Fig. 1. The DG is defined in a way that allows the construction of model's trees where each node represents a possible basis function. A MARS model is constructed by creating a basis function and adding it to the model. According to the grammar, the basis function can be a hinge function or a multiplication of two or more functions. An option of adding an empty node is also added to the grammar to facilitate model simplification by deleting some of the already added functions through the mutation operator.

#### A. The Fitness Function

The goodness of the models evolved is evaluated with a fitness function. The GCV measure (described in Section IV) is usually used for this purpose. However, as we are dealing with pairwise preference data, and since our target values are defined on pairs of instances rather than individuals, we need to revise the definition of GCV. The measure we define, named *Pairwise Generalised Cross Validation* (PGCV), uses the notion of agreement between the model's outputs of a pair of instances and the actual pairwise preference expressed by the users, i.e. the output given to the preferred instance in a pair should be higher than that given to the unpreferred instance. For this purpose, instead of using the residual sum of squares to calculate the error, we used Kendall tau distance. Practically speaking, the PGCV is calculated as:

$$PGCV = \frac{\sum_{i=1}^n u(y_{i.A}, y_{i.B})}{(1 - \frac{\hat{\epsilon}}{n})^2} \quad (6)$$

$$u(a, b) = \begin{cases} 1 & \text{if } f(a) \leq f(b) \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

where  $n$  is the number of pairs in the dataset and  $f(\cdot)$  is the model output.

```

y = C0 + C1 * <HingeFunction1>
    + C2 * <HingeFunction2>
    + C3 * <HingeFunction3> + ...
<HingeFunction1> := max(0, <Feature> - K0)
<HingeFunction2> := max(0, K1 - <Feature>)
<HingeFunction3> := max(0, K2 - <Feature>)
...
<Feature> := f1 | f2 | ... | fn

```

Fig. 2. Example model evolved with the design grammar where  $K_i$  is the knot in function  $i$ ,  $C_i$  is the weight of the function  $i$  and  $Feature$  is the set of input features.

## VI. TRANSFER LEARNING

We experimented with two settings to build generic models of PE. In what follows, we describe each in details.

#### A. Supervised Feature Mapping

Since we are interested in building models that are accurate on both datasets, we adopt ideas from previous work on transfer learning [24], [25]. In the Supervised Feature Mapping (SFM) setting, models are configured to learn two tasks by training them on one and testing on another. In our implementation, we start from models optimised on one dataset and search for the best mapping of features from the second dataset that yields minimum lose in prediction accuracy. The process allows evolution to handle the transfer learning problem in two independent phases: (1) optimise models of PE on one task, and (2) use the constructed models as templates to find the mapping of features that works well on both tasks. This allows the approach to focus on optimising for an individual task while the pre-trained parameters seed the search and optimisation for the second task. Naturally, the method relies on the significance of the similarity between the behaviour of the features (according to reported PE).

More specifically, we use one of the datasets as a source,  $D_s$ , and we train MARS models,  $M_s$ , to predict PE from  $S$ . The results are models where the features,  $F_s = f_{s1}, f_{s1}, \dots, f_{sn}$ , are chosen from the source dataset. In the next stage, we fine tune the models by fixing the models' knots, weights, number and shape of the basis functions, and optimise for the feature mapping: we select a set of features  $F_t = f_{t1}, f_{t1}, \dots, f_{tn}$  from the target dataset,  $D_t$ , that substitutes  $F_s$  with minimal loss in accuracy.

Practically, this is achieved by GE through evolving the grammar. Fig. 2 shows a generalisation of and example grammar evolved from a source task. We then apply a second evolution process where all parameters in the grammar are set as constants (the ones shown in capital letters) except for the features (denoted in bold in the figure). Evolution in this stage fixates on finding the optimal mapping of features given the learned parameters by validating the model performance on the target dataset.

#### B. Unsupervised Transfer Learning

While using models built on one task and fine-tuned on another, this is a valid and fast approach for the problem of

transfer learning [26], the previous approach assumes that the problem is fully supervised, i.e. labels are available for both the source and the target datasets. Unlike unlabelled data which is usually available and relatively easy to collect, labels are usually sparse and hard to gather. We thereafter investigated another unsupervised approach for building transfer learning.

In the UnSupervised Transfer Learning (USTL) setting, we project the features of our domains on a common space that we use later as input to our modelling approach. This is achieved by applying an *unsupervised domain adaptation* method which attempts to establish a relationship between two domains. The traditional approach is to learn a new domain-invariant feature representation by looking for a new projection space. Naturally, Principle Component Analysis- (PCA) based methods have been investigated for this purpose [27], [28], [29].

In our implementation, we follow the same procedure suggested in [30] for unsupervised model adaptation. The idea is to generate intermediate representations of the source and target datasets in the form of subspaces (composed of  $d$  eigenvectors inferred by a PCA). The next step would be to find a transformation function that moves the source subspace closer to the target subspace through a subspace alignment.

Formally speaking, for  $D_s$  and  $D_t$ , and to perform the subspace alignment model adaptation, we first apply PCA, with  $d$  number of components, on  $D_s$  and  $D_t$  (note that  $d$  is constant in both dataset). This results in two projection matrices,  $X_s$  and  $X_t$  for the source and the target, respectively. The next step is to learn a linear transformation function that align the source subspace coordinate system to the one of the target. To achieve this, we need to learn an alignment metric  $X_a = X_s X_s' X_t$ . Multiplying  $D_s$  and  $D_t$  with their corresponding projection matrices,  $X_a$  and  $X_t$ , results in a transformation from the original feature space to the shared one. Once  $D_s$  and  $D_t$  are both in the same feature space ( $D'_s$  and  $D'_t$ ), we can run machine learning methods to build models that can be trained on  $D'_s$  and tested on  $D'_t$ .

This approach is unsupervised in the sense that it does not require target label information. We assume we only have the labels for  $D_s$  and the models can be constructed on the source data while relying on the correlations among the features of the source and the target. Once constructed, models can be employed to provide prediction on the target.

## VII. EXPERIMENTS

We conducted experiments to evaluate the two proposed strategies. in the following sections we present the setup used and the results obtained.

### A. Experimental Setup

As previously mentioned, IMB and a FPS games were used to evaluate the approach. The open source GEVA software [31] is used as a core to implement the needed functionalities. The experimental parameters used are the following: 10 runs each ran for 100 generations with a population size of 100. The tournament selection method of size 5, inter-flip mutation with probability 0.1, one point crossover with probability 0.9, and 0 maximum wraps were allowed. All parameters were assigned experimentally.

For model generation, the number of basis functions is bounded between 6 and 30. The penalty value of the *PGCV* is set to one. To simplify the process of finding and analysing the mapping of features, we set  $K = 1$  of evolved MARS models, meaning that we do not permit multiplications of basis functions.

To evaluate the supervised approach, we evolve MARS models on a source dataset and calculate the accuracy on the other using 10-fold cross validation. The best model obtained from this step is then used to optimise the feature mapping. Fine-tuning the model for feature mapping is done through another step of evolution on the target dataset while measuring the accuracy through 10-fold cross validation.

For unsupervised transfer learning, and since we have a shared space of features, the models are trained on the transformed features of the source dataset (which we assume is labeled) using 10-fold cross validation. We then calculate the accuracy of the models on the transformed version of the target dataset. We used 10 components for PCA.

All experiments are repeated twice where we switch the source and the target to study order effect. Significance effect is observed through the t-test with  $p\text{-value} \leq 0.05$ . All the results are significantly different unless otherwise mentioned.

*B. Analysis*

### B. Analysis

In order to evaluate both approaches to construct models that are accurate across two datasets, we conducted a number of experiments where we use one of the dataset as a source for constructing the models while we use the other as a target. To accommodate for order effect, we repeat the experiments while switching the order of the datasets so that the source becomes the target and vice versa. The results obtained from the possible combinations are presented in Table II.

1) *General Observations:* All models are built to predict pairwise preferences of engagement, frustration and challenge as reported by players. Most models achieved reliable performance with accuracies above 60% (models with random guess in our case would achieve 50% accuracy). The model performance depends on the predicted affected state and the dataset used to train and evaluate the model. When models are trained and evaluated on the same dataset (the cells with shade in the table), the best performing models were those for predicting frustration and challenge from the FPS dataset. In general, it appears that predicting affective states in the IMB dataset is harder than predicting them from the FPS dataset. When predicting challenge, for instance, models trained and evaluate on the IMB dataset achieved around 61% accuracy compared to 85% for the ones trained and tested on the FPS dataset.

While the model performance and generalisation capabilities vary, it appears that both methods work quite well. The accuracies obtained from the generalised models are 58% for the supervised feature mapping and 21% for the unsupervised feature mapping.

above 55% for the unsupervised transfer learning scenario. In the best cases, models of accuracies as high as 83% are built for predicting challenge in the FPS dataset by models trained on IMB data.

2) *Supervised Feature Mapping*: In the supervised feature mapping scenario, one would expect the models to perform better on the source dataset. This is mainly because the models' parameters and shape are optimised for that particular dataset. While this assumption holds for most cases, it is not always true. The models trained on IMB for predicting challenge, for instance, and then used to find the optimal feature mapping from the FPS dataset achieved higher accuracies than those trained and tested solely on IMB (83% for FPS vs. 71% for IMB). It appears that in these cases, testing the models on the target dataset allows them to generalise well and steers them away from overfitting.

The best models to generalise when searching for the best mapping are those for predicting frustration. These models are the best after comparing their accuracies against those for predicting other affective states (their performance is usually comparable) and by observing the difference in the their reported accuracies when trained and evaluated on the same dataset vs. when trained on a source and generalised for a target (the combined difference is minimal).

3) *Unsupervised Transfer Learning*: For the unsupervised transfer learning task, the results are more consistent and the behaviour is more predictable: all models perform better on the source dataset than on the target. In this case, however, it seems that models that perform well on the source dataset does not necessarily generalise as well. Most models trained on FPS as a source are of high accuracies, yet they perform relatively poor when the transformed features taken from the IMB dataset are set as input. We anticipate that overfitting has a greater effect in this case and we expect further investigation of methods that overcome this issue to bring more insights on this matter.

What is interesting to note is that models trained on a transformed version of the feature achieved higher accuracies than those trained on the original feature set (for single-domain modelling). When reducing the dimensionality of the feature space with PCA and training the models on the transformed features, the accuracies obtained on the FPS data are 86%, 87% and 87% which are significantly better than those obtained from the original features set when predicting engagement, frustration and challenge (83%, 85% and 73%). These findings, however, applies only on the FPS as we did not observed the same consistency in behaviour when predicting the affective states from the IMB dataset. We anticipate that the better performance observed by the unsupervised method is due to the advantage gained from reducing the dimensionality of the feature space by PCA. We therefore expect future comparison where PCA is also applied in the supervised case to highlight more insights on the effect of the input space on the performance.

We also looked at the methods' sensitivity to the order of using

TABLE II  
AVERAGE AND STANDARD DEVIATION ERRORS OF THE MODELS WHEN TRAINED AND TESTED ON DIFFERENT CONFIGURATIONS OF THE INFINITE MARIO BROS AND FIRST PERSON SHOOTER DATASETS FOLLOWING THE SUPERVISED FEATURE MAPPING (SFM) AND THE UNSUPERVISED TRANSFER LEARNING (USTL) METHODS.

Source \ Target	SFM		USTL	
	IMB	FPS	IMB	FPS
<i>Engagement</i>				
IMB	0.38±0.04	0.31±0.11	0.35±0.06	0.35±0.07
FPS	0.42±0.04	0.17±0.11	0.46±0.04	0.14±0.17
<i>Challenge</i>				
IMB	0.29±0.06	0.17±0.08	0.33±0.07	0.45±0.05
FPS	0.32±0.04	0.15±0.09	0.47±0.04	0.13±0.10
<i>Frustration</i>				
IMB	0.28±0.07	0.31±0.08	0.28±0.06	0.35±0.07
FPS	0.28±0.04	0.27±0.09	0.42±0.05	0.13±0.10

scenarios, we observed that IMB dataset serves a better starting point for model construction: most of the models first trained on this dataset and then generalised achieved better accuracies than when the reverse order is used. The only exception is when predicting frustration in the supervised case: the models constructed on FPS and then generalised are 3% better than those first trained on IMB and optimised for FPS. We hypothesise that this effect might be related to the nature of features: features collected for IMB convey rich information about players' behaviour that makes them better descriptors of the experience. The findings revealed that feature transformation is a good approach for modelling PE in a specific domain but it does not perform as well when the goal is to create generic models. Supervised feature mapping, on the other hand, yield models that are more consistent across domains although less accurate.

### C. Comparison with Manual Feature Abstraction

We mentioned in the introduction that we previously conducted a preliminary experiment where generic models of PE in the same two games are constructed [5]. The input feature set in that study included generic features manually selected by game experts. The same evolutionary-based approach is used to construct model on the combined feature set. As the goal of this paper is to automatically build accurate models on more than one game, we believe comparing the results could give some insights on the success of the proposed methods. The results obtained in the previous study showed that generic models of 70.88%, 79.81% and 76.89% could be built for engagement, frustration and challenge. Compared to the results in this paper, it seems that automatic discovery of generic transformable knowledge is an efficient and viable alternative. The results obtained are in most cases comparable (the results obtained by the best configurations are 69%, 83% and 72% for the same affective states) indicating a promise for the current approach.

## VIII. CONCLUSIONS

The paper presents an approach to construct models of PE that could work well on more than one game. We

collected rich information of players' behaviour in two quite dissimilar games and we proposed two methods for building cross-domain experience models. In the *supervised feature mapping* setting, our goal is to establish a mapping between the features of a source and a target dataset that permits using the same models in both games simply by switching the input space. In the *unsupervised transfer learning* scenario, we assume that only one of our dataset is labeled and our goal is to transfer the knowledge we learned from labelled data by establishing a new shared space of correlated features in both datasets.

The results obtained showed promising results for both approaches. The performance appears to be related to the affective state being modelled, to the order in which the data is used for model construction and to the type of the method used. In general, it appears that starting with models built on rich information is more promising for knowledge transformation. Naturally, there seems to be a tradeoff between the performance on a specific dataset and the cross-domain performance: the more generalised the model, the less accurate it is on a specific dataset. Further investigations should be performed to find the best tradeoff between overfitting and generalisation.

This paper provides initial results towards building generic models of PE. There is a wide window for future investigations: the features used in this study are purely statistical indicators of frequencies of actions and content, other type of features could be included that provide richer information such as patterns of behaviour or sequences of actions. Our study and results are limited with the specific games used.

The domains in which we applied the methods are rather representative and by no means inclusive of other game types or genres. Further investigations should be undertaken to explore the applicability of the method to the wider space of possible games.

#### ACKNOWLEDGMENTS

The research was supported in part by the Danish Research Agency, Ministry of Science, Technology and Innovation; project "PlayGALe" (1337-00172).

#### REFERENCES

- [1] A. Kobsa, "Generic user modeling systems," *User modeling and user-adapted interaction*, vol. 11, no. 1-2, pp. 49–63, 2001.
- [2] R. Koster, *A theory of fun for game design*. Paraglyph press, 2004.
- [3] T. Malone, "What makes computer games fun?" New York, NY, USA: ACM, 1981.
- [4] D. Heckmann, *Ubiquitous user modeling*. IOS Press, 2006, vol. 297.
- [5] N. Shaker, M. Shaker, and M. Abou-Zleikha, "Towards generic models of player experience," in *Proceedings, the Eleventh Aaai Conference on Artificial Intelligence and Interactive Digital Entertainment (aiide-15)*. AAAI Press, 2015.
- [6] M. Wöllmer, M. Kaiser, F. Eyben, B. Schuller, and G. Rigoll, "Lstm-modeling of continuous emotions in an audiovisual affect recognition framework," *Image and Vision Computing*, vol. 31, no. 2, pp. 153–163, 2013.
- [7] R. Calvo, S. D'Mello *et al.*, "Affect detection: An interdisciplinary review of models, methods, and their applications," *Affective Computing, IEEE Transactions on*, vol. 1, no. 1, pp. 18–37, 2010.
- [8] A. Ortony, G. Clore, and A. Collins, *The cognitive structure of* **2016 IEEE Conference on Computational Intelligence and Games (CIG'16)**
- [9] C. Conati, "Probabilistic assessment of user's emotions in educational games," *Applied Artificial Intelligence*, vol. 16, no. 7-8, pp. 555–575, 2002.
- [10] G. N. Yannakakis and J. Hallam, "Real-time Game Adaptation for Optimizing Player Satisfaction," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 1, no. 2, pp. 121–133, June 2009.
- [11] N. Shaker, G. N. Yannakakis, and J. Togelius, "Crowdsourcing the aesthetics of platform games," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 5, no. 3, pp. 276–290, 2013.
- [12] C. Bauckhage, K. Kersting, R. Sifa, C. Thureau, A. Drachen, and A. Canossa, "How players lose interest in playing a game: An empirical study based on distributions of total playing times," in *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*. IEEE, 2012, pp. 139–146.
- [13] C. Chambers, W.-c. Feng, S. Sahu, and D. Saha, "Measurement-based characterization of a collection of on-line games," in *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*. USENIX Association, 2005, pp. 1–1.
- [14] R. Sifa, A. Drachen, and C. Bauckhage, "Large-scale cross-game player behavior analysis on steam," in *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.
- [15] A. Drachen, C. Thureau, R. Sifa, and C. Bauckhage, "A comparison of methods for player clustering via behavioral telemetry," *arXiv preprint arXiv:1407.3950*, 2014.
- [16] J. C. Osborn and M. Mateas, "A game-independent play trace dissimilarity metric," *Foundations of Digital Games*, 2014.
- [17] J. Runge, P. Gao, F. Garcin, and B. Faltings, "Churn prediction for high-value players in casual social games," in *Computational Intelligence and Games (CIG), IEEE Conference on*, 2014, pp. 1–8.
- [18] F. Hadji, R. Sifa, A. Drachen, C. Thureau, K. Kersting, and C. Bauckhage, "Predicting player churn in the wild," in *Computational Intelligence and Games (CIG), IEEE Conference on*, 2014, pp. 1–8.
- [19] J. Ortega, N. Shaker, J. Togelius, and G. Yannakakis, "Imitating human playing styles in super mario bros," *Entertainment Computing*, 2013.
- [20] S. Bakkes, S. Whiteson *et al.*, "Towards challenge balancing for personalised game spaces," 2014.
- [21] N. Shaker, M. Shaker, I. Abu-Abdallah, M. Al-Zengi, and M. H. Sarhan, "A quantitative approach for modelling and personalizing player experience in first-person shooter games," in *UMAP Workshops*, 2013.
- [22] J. H. Friedman, "Multivariate adaptive regression splines," *The annals of statistics*, pp. 1–67, 1991.
- [23] M. O'Neill and C. Ryan, "Grammatical evolution," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 349–358, 2001.
- [24] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.
- [25] P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. LeCun, "Pedestrian detection with unsupervised multi-stage feature learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 3626–3633.
- [26] Y. Gong, Y. Jia, T. Leung, A. Toshev, and S. Ioffe, "Deep convolutional ranking for multilabel image annotation," *arXiv preprint arXiv:1312.4894*, 2013.
- [27] B. Chen, W. Lam, I. Tsang, and T.-L. Wong, "Extracting discriminative concepts for domain adaptation in text mining," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 179–188.
- [28] S. J. Pan, J. T. Kwok, and Q. Yang, "Transfer learning via dimensionality reduction," in *AAAI*, vol. 8, 2008, pp. 677–682.
- [29] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang, "Domain adaptation via transfer component analysis," *Neural Networks, IEEE Transactions on*, vol. 22, no. 2, pp. 199–210, 2011.
- [30] B. Fernando, A. Habrard, M. Sebban, and T. Tuytelaars, "Unsupervised visual domain adaptation using subspace alignment," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 2960–2967.
- [31] M. O'Neill, E. Hemberg, C. Gilligan, E. Bartley, J. McDermott, and A. Brabazon, "Geva: grammatical evolution in java," *ACM SIGEVOlution*, vol. 3, no. 2, pp. 17–22, 2008.

# I am a legend: hacking Hearthstone using statistical learning methods

Elie Bursztein  
Google  
elieb@google.com

**Abstract**—In this paper, we demonstrate the feasibility of a competitive player using statistical learning methods to gain an edge while playing a collectible card game (CCG) online. We showcase how our attacks work in practice against the most popular online CCG, *Hearthstone: Heroes of World of Warcraft*, which had over 50 million players as of April 2016. Like online poker, the large and regular cash prizes of *Hearthstone*'s online tournaments make it a prime target for cheaters in search of a quick score. As of 2016, over \$3,000,000 in prize money has been distributed in tournaments, and the best players earned over \$10,000 from purely online tournaments.

In this paper, we present the first algorithm that is able to learn and exploit the structure of card decks to predict with very high accuracy which cards an opponent will play in future turns. We evaluate it on real *Hearthstone* games and show that at its peak, between turns three and five of a game, this algorithm is able to predict the most probable future card with an accuracy above 95%. This attack was called “game breaking” by Blizzard, the creator of *Hearthstone*.

## I. INTRODUCTION

Over the last 20 years, since the inception of *Magic the Gathering* [34], collectible card games (CCGs) have emerged as one of the most pervasive forms of contemporary game play. In North America alone, sales were estimated to be above \$800 million in 2008 [9]. Given the popularity of CCGs, it is not surprising that in the last few years, with the rise of mobile devices and tablets, computer-based CCGs have emerged as one of the largest parts of the e-sport scene. In particular, the most popular online CCG, *Hearthstone: Heroes of World of Warcraft* [11], which had over 50 million players as of April 2016 [26], is one of the most profitable games for e-sport players. As of 2016, over \$3,000,000 in prize money has been distributed in *Hearthstone* tournaments since its creation in 2014 [12]. The ranking shows that the best players earned over \$10,000 from purely online tournaments and over \$200,000 in total [12]. Given *Hearthstone*'s popularity, the amount of money at stake in an online tournament and that most offline tournaments with prizes have an online qualification phase, it is important to understand how players can attack *Hearthstone* to gain an unfair edge.

**Contribution.** To address this need, this paper, to the best of our knowledge, describes the first security analysis of an online CCG and it is the first to develop statistical learning attacks [17] against CCG games. This paper presents the first algorithm that is able to learn and exploit the structure of

card decks to predict with very high accuracy which cards an opponent will play in future turns. Using a dataset of 50,000 game replays collected in May 2014, we demonstrate that our statistical learning attack works in practice by successfully using it against *Hearthstone*. At its peak, between turns three and five of a game, our algorithm is able to predict the most probable future card with an accuracy above 95%. As recounted in the ethics section later in the introduction, the effectiveness of this attack was validated by the creator of *Hearthstone*, Blizzard Entertainment, which deemed it “game breaking.”

**Ethics.** We reached out to Blizzard Entertainment to disclose our findings prior to publication but we did not get any response. However, after we presented some of the findings reported in this paper at a famous hacking conference, Blizzard reached out to us. During the conversation, one of the main game designers acknowledged that the deck prediction attack demonstrated was indeed game breaking and asked us to not publish our prediction tool or dataset. Following Blizzard's request, we agreed not to release the code for our prediction tool or the full dataset.

**Outline.** The remainder of the paper is organized as follows. In Section II, we provide the background needed to understand the state of game security research, what a CCG is and how a *Hearthstone* game is played out in particular. In this section, we also discuss our threat model and how our dataset was created. In Section III, we focus on predicting which cards the opposing player will play in future turns. We describe how our prediction attack works and report the results of our evaluation against our dataset. In Section IV, we briefly evaluate the game's most predictive metrics using a decision tree to shed light on the most important factors that players need to consider while playing *Hearthstone*. Finally, we conclude in Section V.



## II. BACKGROUND

In this section, we provide the necessary background. We start by discussing the state of game security research. Then we briefly explain what collectible card games (CCGs) are. Next, we describe how a *Hearthstone* game is played, its win conditions and where the game complexity stems from. Next, we discuss the threat model and attack types considered in this paper. Finally, we explain how the dataset used in this paper was collected.

**Security research, machine learning and games.** Researching how to exploit games and detecting cheaters using machine learning has the long-standing tradition of being a very prolific security research topic. Detecting MMORPG bots, including *World of Warcraft* ones, was studied in [16], [22]. A technique to build a map hack and defend against them was presented in [5]. Using machine learning to detect aim bots for *Unreal 3* was presented in [15]. The research presented in [35], [1] extended the use of machine learning for aim-bot detection to other FPSs (First Person Shooters). Applying fuzzing to attack online games and in particular *League of Legends* was presented in [6].

**Collectible Cards Games.** Also known as trading card games (TCGs),<sup>1</sup> CCGs are broadly defined as turn-based strategy games that use a pool of cards (usually hundreds) that are collected by players. These games rose in popularity in 1993 with the release of *Magic The Gathering* [34] even though the origin of the genre can be traced back to a baseball card game published in 1904 [9]. What makes CCG different from traditional card games is that players cannot buy all the cards at once; they need to collect them by buying *boosters* to expand their collection. Cards have different rarity and a booster randomly includes a few common cards (the less rare ones) and at least one card that is rare or “better”. *Hearthstone* has five levels of rarity: *sets* cards (which are given for free to players), *common* cards, *rare* cards, *epic* cards and *legendary* cards. Each *Hearthstone* booster comprises five cards selected at random with at least one card that is rare or better. *Hearthstone* is considered a “live” CCG, which means that its creator, Blizzard, keeps adding new cards to it in *expansion sets*, which are released regularly. During a game, players play with decks that they constructed from their personal collection of cards. The construction of a deck is strictly regulated. The number of cards a deck contains is usually restricted (30 exactly for *Hearthstone*) as is the number of copies of a given card that can be played (two of the same kind for *Hearthstone*, except for legendary cards, which are restricted to a single copy).

Beyond their recreational and competitive purposes, CCGs are also used for education, including teaching kids about diseases [28] or professionals about computer security [10].

<sup>1</sup>The acronyms CCG and TCG are used interchangeably and they co-exist due to licensing issues. In this paper, we will exclusively use the CCG acronym as *Hearthstone* is marketed as an online CCG by Blizzard.

From a theoretical standpoint, CCG games are viewed as imperfect information games with randomness. Previous research on CCGs focused on either optimizing how to collect cards [3] or taking a theoretical approach to determining the best way to play the game [8].

To the best of our knowledge, no prior work exists on predicting what cards the opposing player will play in future turns, on finding over-powered cards or on using machine learning to predict a game’s outcome. However while there is no formal research on it, we note the existence of bots [2] that are designed to play the game instead of humans for leveling purposes. Those bots work by analyzing the current board state and attempting to make the best play possible. The only documented attempt to exploit a game design flaw using statistical learning to gain a competitive edge was the use of a genetic algorithm to find an optimal build order in *Starcraft 2*. This led to the discovery of a very efficient and yet counterintuitive build order called the “7 roach timing-attack” [4].

**Hearthstone.** *Hearthstone: Heroes of Warcraft* is a free-to-play digital CCG that was initially released in early 2014 by Blizzard entertainment [11]. As of May 2016, *Hearthstone* has more than 50 million player accounts registered worldwide [26], making it the most popular online CCG. As of 2016, over \$3,000,000 in prize money has been distributed in *Hearthstone* tournaments since its creation [12]. The ranking shows that the best player earned over \$10,000 from purely online tournaments and over \$200,000 in total [12]. This makes *Hearthstone* one of the top ten most profitable games for pro-players and a target of choice for cheaters, given the large online cash prizes and that most offline tournaments will financially reward each player who qualifies to the offline phase through the online qualification phase.



Fig. 1. Screenshot of the *Hearthstone* game board with the most important features illustrated

A *Hearthstone* game, as other CCGs, is a turn-based match between two opponents represented by their chosen “hero,” an important character from Warcraft lore that has 30 health points, as depicted in Figure 1.

At the start of the game, each player draws three cards from their deck, which comprises 30 cards selected by the player from their card collection before the game started. At the start of their turn, the player draws a new card from their deck. While most cards are available to heroes of any class, a substantial portion is limited to a specific class, giving each hero its own strengths and unique possibilities. Each card or hero power requires the player to spend a specific amount of *mana* to play it, strategically limiting the player's options.

At the beginning of turns one through to ten, the player's mana pool is replenished and increased by one, allowing them to play more powerful cards as turns pass. After turn ten, the player's mana pool is replenished and capped to ten mana points. We note that for game balancing reasons, the player who starts second gets an extra card from their deck to equalize the draw count and a "bonus" card, called the coin, which gives them a free mana point when played. A Hearthstone match ends when one or both players have reached zero health, or choose to concede.

Building the best deck possible is an essential skill and many archetypes of decks exist. Deck archetypes are characterized by the distribution of the mana cost of the cards they contain, which is referred to as the *mana curve*. A low mana curve (having many cheap cards) is called an *aggro deck* and is usually very good early in a game. A balanced curve is referred to as a *mid-range deck* and a deck that contains mostly powerful cards with high mana cost is known as a *control deck*. Last but not least, the *combo deck* refers to a deck that leverages very powerful synergy between specific cards to kill the opposing player, usually in a single turn. Blizzard aggressively balances the game by *nerfing* decks (which means making them less efficient) that exhibit a too high win rate or "make the game not fun," which usually means combo decks that can kill an opponent in a single turn [31].

Hearthstone offers four types of card: *minions*, which are creatures that exist on the board as visible in Figure 1. A minion has a set of attack and health points and sometimes special abilities, which are described in the text of the card. A minion's current attack points are displayed in a yellow circle located on the bottom left of the card, whereas a minion's current health is represented as a red drop located on the bottom right of the card. The second type of card comprises *spells*, which are abilities directly played from the player's hand. The third type of card comprises *weapons*, which can be equipped one at a time by the player's hero to give it the ability to attack for a defined amount of attack points and time. Finally, *secrets* are cards that are played on the board but are hidden behind a question mark. They are triggered by specific conditions including if the opponent plays a minion or an opponent's creature tries to attack the player's hero.

**Threat model.** The threat model considered in this paper is the passive attacker model, which assumes that the attacker has access to the state of the game and the ability to observe all its network traffic. We also assume that the attacker has access to a vast set of previous game replays, which can be used to apply various statistical learning methods. This attacker model was chosen over the active attacker model because it matches what a cheater can realistically do during competitive Hearthstone games that are observed by referees, casters and a crowd that will immediately spot any kind of active tampering. In an online tournament, the referees see only the game screen via the in-game interface and the players' faces through their webcam, which leaves players full latitude to run the tools of their choice before and during games. Such passive attacks are far from being theoretic: for example, some players, due to screen reflections on the webcam feed, have been caught watching a casting video stream during a tournament to know their opponent hands [24].

**Attack types.** In this paper, we focus on how an attacker can use a machine-learning algorithm during the game to predict the most likely cards that the opponent will play in future turns. As in any strategy game, anticipating the other player's moves is essential for winning, as it informs the player's decision-making process. For example, a player routinely has to decide between playing an extra creature to further a tempo advantage or given the likelihood that the opponent will play a board-clear, to hold off to reduce the value that the opponent will gain from their board-clear. Such decisions made under time (a turn is 75 seconds) and tournament pressure are very difficult. This is even more true when the opponent plays secrets, which are cards that are in play but not revealed, as the type of secret played and its trigger condition need to be evaluated carefully. As recounted in the introduction, the effectiveness of our attacks was validated by *Hearthstone* game designers themselves to the point where they called the predicting attack "game breaking" and asked us to not publish our prediction tool.

**Attack generality.** In this paper, we analyze *Hearthstone* as it was in April and May 2014. Since then, many cards have been added and some cards have been changed (nerfed). While those changes affect the ranking of the most powerful cards currently available and the most popular decklists, our attacks and methods remain equally applicable, as they target the core mechanics of the game not specific cards or bugs. For the same reasons, our attacks work against any CCG.

**Dataset.** For the paper, we use a dataset of 50,000 anonymous replays that came from hundreds of players who used a game-recording tool. Players use this tool to keep track of and analyze their performance. As Blizzard does not provide a replay features, game replays were recorded from the game logs. As a result, our replays contain only partial information regarding the opposing player's deck: we only get a log when a card is drawn, played, activated or killed. If a card is drawn but never played, we know only that the opponent had an unknown card in their hand.

However, this is not an issue because we are interested only in predicting the opponent's next move based on what the player knows. Replays were collected between April and May 2014.

### III. PREDICTING AN OPPONENT'S FUTURE PLAYS



Fig. 2. Our prediction tool in action. Predicted cards are displayed on the bottom left.

In this section, we demonstrate how we built a tool, as illustrated in Figure 2, that can predict what the opposing player will play in future turns based on the previous cards they played. These predictions give the player an edge because anticipating the other player's moves is essential for winning in *Hearthstone*, since this guesswork informs the player's decision-making process. For example, players routinely wonder if they should play an extra creature to further a tempo advantage or given the likelihood that the opponent will play a board-clear, whether they should hold off to reduce the value that the opponent will gain from their board-clear. Being able to figure out under time (a turn is 75 seconds) and tournament pressure the optimal play in these complex situations is what separates the best players from the rest.

The average win rate on the online ladder for the best decks for good players is around 53% while pro-players commonly reach 70% with the same decks [20]. One may think that a drawback of this attack is that it requires observing quite a few played cards before making good predictions. However, our evaluation shows that that is not the case: by turn two, our algorithm already makes accurate predictions and is able, for instance, to predict two cards that the opponent will play in the future with a success rate over 50%: 66.3% for the best prediction and 44.1% for the second best prediction.

This section is structured as follows: first we discuss the underlying reasons that make *Hearthstone* predictable, then we explain how we are able to read game events programmatically. Next, we present our machine-learned ranking algorithm. Finally, we evaluate our tool on a dataset of 50,000 replays and show that it has up to a 96.2% success rate at predicting a card that will be played by the opponent in the future when its accuracy peaks between turns three and five.

#### A. Why is *Hearthstone* predictable?

*Hearthstone* in its original release, the one considered in this paper, had 465 cards that players could choose from to build their deck of 30 cards. It has a limit of two copies of a given card (see Section II for more background). This hypergeometric distribution [33] of choosing 30 cards out of 930 leads theoretically to a space of potential decks that is extremely large: roughly  $4.7^{2358}$  possible decks. However, in practice the decks played have a lot of predictable structure due to the following reasons: First, some cards are restricted to a specific hero class. Second, by design many cards are meant to work best when played with other very specific cards. Third, a significant number of cards are just bad or under-powered, which means that they are almost never played in competitive games. Last but not least, a huge fraction of players rely on *netdecking* [25], which is the act of using a deck made by someone else (usually a pro-player) and found online.

As of 2015, the netdecking phenomenon has become so mainstream that weekly blog posts provide and rank the top decks currently played [30]. As a result, the distribution of cards that are used by players is heavily skewed. For example, the site *HearthPwn*, which has a very large set of decklists supplied by players, reports that the card *Keeper of the Groove* is used in 76.89% of druid decks [18]. Similarly, the *Northshire Cleric* is used in 84.16% of priest decks [19]. While self-reported decks may not accurately reflect the real distribution of what cards are played, the aforementioned statistics still convey that there is broad agreement among players on what are the must played cards. Our tool exploits this underlying structure by learning from game replays which cards are played significantly more often than others and which cards are often played together.

#### B. How to read the game state

Before delving into how our prediction algorithm works, it is worth explaining how our tool gets access to a player's actions, including which cards were played and how many cards were drawn. As Blizzard does not provide an API for this, over time, four main ways have been devised by the *Hearthstone* community. The first relies on DLL injection [13] and hooks the main functions of the game. This approach allows one not only to get a player's actions but also generates them (e.g., play a card). This makes it the method of choice for *Hearthstone* bots [2] that automate game playing for leveling purposes. Unsurprisingly, this method is also the one that Blizzard actively looks for and bans using its Warden [32], as it is used for these forbidden automation and map hacks [5]. Two alternative approaches were developed in the early days of *Hearthstone* to track games: the first relies on network traffic sniffing to capture players' actions at the network level. The second one uses image recognition techniques and continually takes game screenshots. Both approaches were deprecated in May 2014 when reverse engineering of the game client revealed it was possible to turn on debug logs that would provide enough information to track game actions [14].



Since then, every game tracker, including our prediction tool, moved to this method as it does not violate Blizzard's terms of service, like network sniffing, and is more reliable than image recognition. After setting *Hearthstone* in debug mode, our tool continually reads the logs and parses the entries to recreate an internal representation of the game state.

### C. Machine-learned ranking algorithm

As outlined earlier, the goal of our tool is to provide a ranked list of cards that are in the opponent's deck and are yet to be played, based on cards previously played. To achieve this goal, we built a machine-learned ranking system [7] that given a set of previous cards returns a ranked list of cards by their likelihood of being played. An example of our tool output is visible on the bottom left of Figure 2.



Fig. 3. *Hearthstone* card sequence as a bag of bigrams

Our machine-learned ranking system uses a modified version of Markov chains [23], where card co-occurrences are modeled as a bag of bigrams [27]. As illustrated in Figure 3, extracting a bag of bigrams from a card sequence involves extracting all possible combinations of card pairs regardless of the order they were played. While experimenting with various data models, we found that using a model requiring more memory (e.g., trigrams, 4-grams, ...) or even adding more structure to the model (e.g., using regular bigrams to learn the exact card sequence) yielded worse results. We provide an analysis of the reasons behind this counterintuitive result and a comparison between the performance of bigrams and trigrams in the evaluation part of this section. Until then, we focus on describing the approach that performs the best: modeling card sequences as a bag of bigrams.

**Learning phase.** To learn its model, our tool extracts for each game replay the sequence of cards played by the opponent and constructs a bag of bigrams. Those bags of bigrams are then used to construct the *occurrence table*, which for a given card returns the list of co-occurred cards with the number of times this co-occurrence happened. At prediction time, this allows it to find the popular pair/combo efficiently, as they are the ones that have the largest number of co-occurrences.



Fig. 4. Prediction phase example

**Prediction phase.** During a game, as depicted in Figure 2, each time the opponent plays a card, the algorithm outputs a list of potential next cards ordered by their probability of being played. Under the hood, as exemplified in Figure 4, the tool leverages the occurrence table constructed in the learning phase to construct a ranked list of cards as follows: First, the algorithm uses the occurrence table to retrieve all cards that were co-played with the cards already played and the frequency of those co-occurrences. In our example, depicted in Figure 4, the algorithm looks up the cards that were co-played with *Deadly Poison* and *Shiv*. These lookups led it to retrieve the *Blade Flurry* card, which was co-played 350 times with *Deadly Poison* and 400 times with *Shiv*. It also retrieves the *Fan of Knives* card, which was co-played 500 times with *Deadly Poison*, and *Amani Berserker*, which was played 400 times with *Shiv*. Next, the algorithm sums up card appearances, removes cards already played and reverse sorts the remaining potential cards based on the frequency of their co-occurrences. In our example, *Blade Flurry* ends up being the most seen/probable card with a count of 750, *Fan of Knives* is second with a count of 500, and *Amani Berserker* is last with a count of 400. Cards with a frequency below a certain threshold, *Amani Berserker* in our example, are cut off to remove improbable or noisy cards. While not depicted in the example for clarity, before returning the list, the algorithm normalizes the card count by the total count, such that the returned list has a percentage associated with each card and not the raw count.

### D. Evaluation

For the evaluation, we used 45,000 games from our dataset for training and 5000 games for testing. See Section II for the dataset collection methodology. For each test game, the algorithm was asked at the end of each turn to predict what would be the top ten cards that would be played in future turns. We then compared each prediction with the cards that were effectively played in the upcoming turns and marked a prediction as correct if the card was effectively played. The result of our evaluation is summarized in Figure 5.

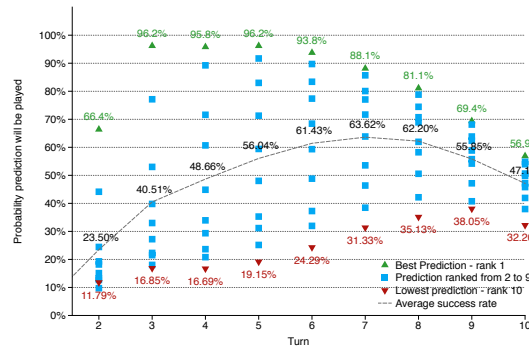


Fig. 5. Success rate for top ten prediction bucketed by turn

To evaluate if our ranking algorithm is successful at ranking the cards with the highest chance of being played higher, we evaluated the success rate of each rank independently. Additionally, in our evaluation, to account for the fact that as turns pass, the amount of information (number of cards played) available to the algorithm increases and that the number of cards yet to be played decreases, we bucketed the result by turn. This allows us to understand how the tradeoff between more information and having a smaller set to predict from plays out.

The first observation we can make is that the ranking function performs as intended. In every case, the highest ranked prediction is the one with the highest success rate, as visible in Figure 5. Similarly, the prediction ranked number two always has a better success rate than the one ranked below it and so forth. The only misranking happens for the lowest prediction (rank tenth) at turn 2, where the prediction ranked tenth has a higher success rate than the one ranked ninth. This is likely due to the fact that the algorithm has seen very few cards at that stage (likely one) and, therefore, cannot separate the predictions.

Secondly, we observe that the algorithm is very successful at predicting what the opponent will play, with the best prediction having a 96.2% success rate for turns three and five. Throughout the first ten turns of the game, the best prediction has a success rate above 56.9%, which shows that the algorithm successfully learned the hidden correlations between cards and is able to use it to make accurate to very accurate predictions reliably throughout the game.

Thirdly, we see that indeed the tradeoff between having more information and a smaller set of cards yet to play to predict from does affect the algorithm's performance. In the first seven turns, the average prediction success rate steadily climbs from 23.5% to 63.2%. Then as the game progresses and the number of turns left and, therefore, the number of cards the opponent will play are both getting smaller and smaller, the accuracy drops back and reaches 47.5% by turn ten.

This shows that past a certain point, more information cannot counterbalance the fact that the pool of cards yet to be played is getting too small.

Finally, we note that the delta between the best prediction success rate and the worst one decreases steadily as the turns pass. This is again expected because the amount of mana the player has has increased, which leads to a greater number of valid plays as turns pass.

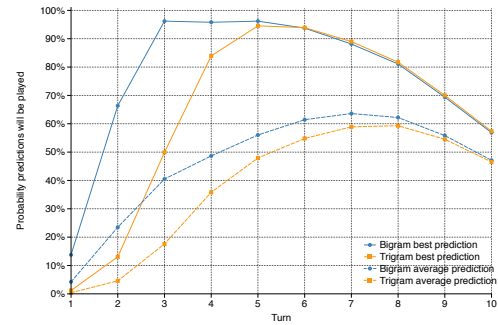


Fig. 6. Bigram versus trigram success rate

During our experimentation, we attempted to use a model requiring more memory by using longer  $n$ -grams. However, none of these alternative models yielded better results than bigrams. Similarly, every attempt to use a more constrained model, such as strict bigrams, in the hope of exploiting further the relations between cards, yielded worse results than our bag of bigrams. This negative result is best illustrated by comparing the success rate of the algorithm when using bigrams and trigrams. As visible in Figure 6, the best trigram-based prediction is significantly worse for turns one to five and marginally better for turns six to ten. However, the average prediction success rate is worse for every turn and by quite a huge margin for the early turns.

This is a counterintuitive result because in other cases, including predicting words typed, spell-checking and predicting almost any type of sequence [21], using a model requiring more memory leads to increased accuracy. This result is partially explained because players draw cards at random, which adds baseline noise to each game. Also the order in which a player plays cards during a given turn may not influence the outcome, which makes all sequence permutations likely to see some play. Another potential reason that favors having a “laxer” model is that players run decks with slight variations, such as including one or two copies of a given card. That being said, it may be possible that a model requiring even more memory, such as LSTM/RNN [29], and using significantly more data, could outperform the current approach. Given that our tool is already good enough at helping an attacker figure what their opponent will play in future turns, this investigation is left as an open question.

#### IV. PREDICTIVE FEATURES

In this section, we briefly explore the use of machine learning to predict the outcome of a *Hearthstone* game. Understanding the most predictive metrics sheds light on what are the optimal strategies for playing *Hearthstone* and can help players to make better decisions. Over the years, the *Hearthstone* community has come up with a few metrics that are routinely used to predict who will win a game. However, until this work, there was no formal analysis of how good those metrics are at predicting game outcomes. Drawing on our experience, forum discussions and game analysis by pro-players, we compiled and formalized the following list of metrics:

- **Mana efficiency:** The mana efficiency metric is the difference between how much mana the player spent versus how much their opponent spent. So, if the player has spent 4 mana and their opponent has spent 2, then the mana efficiency will be  $4 - 2 = 2$ . Conversely, if the opponent spent 6 mana and the player spent only 2 mana, the mana advantage is  $2 - 6 = -4$ . Given that according to our card model (Section III), the power of a card is (almost) perfectly reflected by its mana price, we were expecting that this metric would be the most predictive. As reported earlier, this is the case, which supports even further the validity of our model and its assumptions.
- **Board mana advantage:** The board mana advantage is computed as the difference between the sum of the mana for the cards that the player has on the board versus the sum of the mana for the cards the opponent has on the board. According to our model, this metric should be a better predictor than the sheer number of creatures on the board, as our model implies that a 3-mana creature is more powerful than two creatures that cost 1 mana.
- **Board advantage:** The board advantage measures the difference between how many minions the player has versus how many minions the opponent has in play.
- **Draw advantage:** The draw advantage measures who drew the most cards by calculating the difference between how many cards the player has drawn so far and how many cards their opponent has drawn. For instance, if the player has drawn four cards and their opponent has drawn two, then the draw advantage is  $4 - 2 = 2$ . Our model predicts that the value of having a card is a fraction of the card's mana cost, so this metric should be a weak indicator.
- **Hand size advantage:** This metric refers to how many cards the player has in hand versus how many cards the opponent has. It is a somewhat strange metric, but many casters and players refer to it, so it was included. If the player has two cards in their hand and the opponent has three cards, then the hand size advantage is  $2 - 3 = -1$ .

To evaluate which metric has the most predictive power, we used the standard random forest feature selection algorithm, as it outputs a ranked list of features with their predictive power. To do this, we modeled games as a binary classification problem and used our metrics as features.

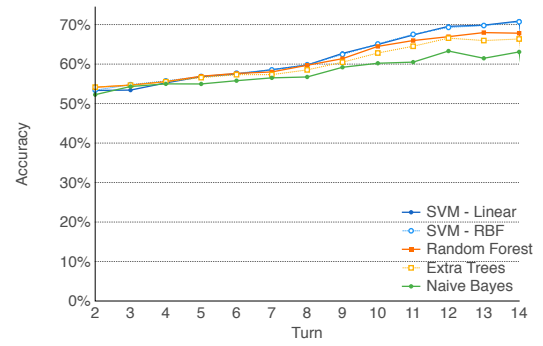


Fig. 7. Various algorithms accuracy at predicting the winner

The feature vectors are constructed by outputting the value of each metric at the end of each turn for the first 14 turns, the cumulative value of the features at the end of each turn for the first 14 turns of the game, and the value and cumulative value of the metrics at the end of the game. We then trained a random forest on our dataset using 45,000 games and tested it on 5000 games. To be thorough, on top of training a random forest algorithm, we also trained several classifiers, namely a naive Bayes, an extra tree, a support vector machine (SVM) with a linear kernel, and a SVM sigmoid kernel grid search. The accuracy of the various algorithms on our dataset is reported in Figure 7. All classifiers got better as turns passed, and the accuracy of each classifier is above the baseline of 50%. The accuracy of the random forest is very close to the best classifiers, which are based on SVM. This gave us confidence that using it to rank features will give meaningful results.

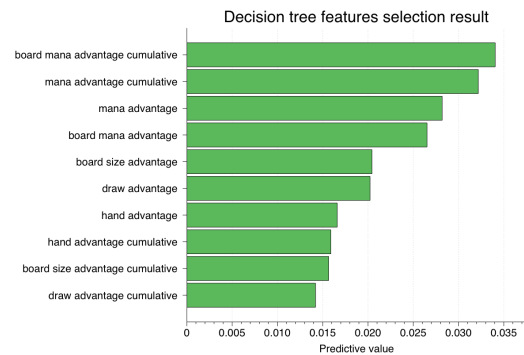


Fig. 8. Feature prediction power

The results for the feature selection algorithm are reported in Figure 8. As visible in this chart, the cumulative metrics that track mana usage (mana advantage and board mana) are the most predictive. We expected that the mana advantage metric would be the best, instead of the board advantage, but with hindsight this makes sense. A player's minions that are still in play after a turn gives them a lasting advantage.



While the board size advantage, draw advantage and hand advantage are significantly less predictive, the results confirm the community's intuition that those metrics are important. We note that the metric selected by the algorithm as most relevant, was the metric at the end of the game not the metric at a particular turn, which suggests that there is no turn that is more important than another, unlike what some of the Hearthstone community believes, which is that turns two and three are critical.

## V. CONCLUSION

In this paper, we demonstrated the feasibility for a competitive player to use statistical learning methods to gain an edge while playing collectible card games online. We showcased how our attacks work in practice against the most popular online CCG *Hearthstone: Heroes of World of Warcraft*, which had over 30 million players as of May 2015. We devised a statistical learning algorithm to attack *Hearthstone*. It learns and exploits the structure of card decks to predict with very high accuracy what an opponent will play in future turns. At its peak, between turns three and five of the game, this algorithm is able to predict the most probable next card with an accuracy above 95%.

## REFERENCES

- [1] Hashem Alayed, Fotos Frangoudes, and Clifford Neuman. Behavioral-based cheating detection in online first person shooters using machine learning techniques. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pages 1–8. IEEE, 2013.
- [2] Arstechnica. Hearthstone bot maker closes shop after blizzard crackdown. <http://arstechnica.com/gaming/2014/11/hearthstone-bot-maker-closes-shop-after-blizzard-crackdown/>, Nov 2014.
- [3] Robert A Bosch. Optimal card-collecting strategies for magic: The gathering. *The College Mathematics Journal*, 31(1):15, 2000.
- [4] Louis Brandy. Using genetic algorithms to find starcraft 2 build orders. <http://lbrandy.com/blog/2010/11/using-genetic-algorithms-to-find-starcraft-2-build-orders/>, Nov 2010.
- [5] Elie Bursztein, Mike Hamburg, Jocelyn Lagarenne, and Dan Boneh. Openconflict: Preventing real time map hacks in online games. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 506–520. IEEE, 2011.
- [6] Elie Bursztein and Patrick Samy. Fuzzing online games. In *Defcon 20*, 2011.
- [7] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136. ACM, 2007.
- [8] Peter Cowling, Colin D Ward, Edward J Powley, et al. Ensemble determinization in monte carlo tree search for the imperfect information card game magic: The gathering. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(4):241–257, 2012.
- [9] B David-Marshall, J v Dreunen, and M Wang. Trading card game industry-from the t to the c to the g. Retrieved December, 12:2013, 2010.
- [10] Tamara Denning, Adam Lerner, Adam Shostack, and Tadayoshi Kohno. Control-alt-hack: the design and evaluation of a card game for computer security awareness and education. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 915–928. ACM, 2013.
- [11] Blizzard Entertainment. *Hearthstone: Heroes of warcraft*. <http://us.battle.net/hearthstone/en/>, March 2014.
- [12] esportearning. *Hearthstone earning*. [http://www.esportearnings.com/games/328-hearthstone-heroes-of-warcraft/top\\_players\\_online](http://www.esportearnings.com/games/328-hearthstone-heroes-of-warcraft/top_players_online).
- [13] Stephen Fewer. Reflective dll injection. *Harmony Security, Version*, 1, 2008.
- [14] Flipperbw. Simple hearthstone logging - see your complete play history without tcp, screen capture, or violating the tos. [https://www.reddit.com/r/hearthstone/comments/268fkk/simple\\_hearthstone\\_logging\\_see\\_your\\_complete\\_play](https://www.reddit.com/r/hearthstone/comments/268fkk/simple_hearthstone_logging_see_your_complete_play), May 2014.
- [15] Luca Galli, Daniele Loiacono, Luigi Cardamone, and Pier Luca Lanzi. A cheating detection framework for unreal tournament iii: A machine learning approach. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 266–272. IEEE, 2011.
- [16] Steven Gianvecchio, Zhenyu Wu, Mengjun Xie, and Haining Wang. Battle of botcraft: fighting bots in online games with human observational proofs. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 256–268. ACM, 2009.
- [17] Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, 27(2):83–85, 2005.
- [18] HearthPwn. Keeper of the groove statistics. <http://www.hearthpwn.com/cards/459-keeper-of-the-grove>, 2015.
- [19] HearthPwn. Northshire cleric. <http://www.hearthpwn.com/cards/600-northshire-cleric>, 2015.
- [20] Hearthstats. Ladder statistics. <http://hearthstats.net/dashboards>, Feb 2016.
- [21] James H Martin and Daniel Jurafsky. Speech and language processing. *International Edition*, 2000.
- [22] Stefan Mitterhofer, Christopher Kruegel, Engin Kirda, and Christian Platzer. Server-side bot detection in massively multiplayer online games. *IEEE Security & Privacy*, 2009.
- [23] James R Norris. *Markov chains*. Cambridge university press, 1998.
- [24] Pcgamer. *Hearthstone's team archon releases hosty hours after cheating allegations*. <http://www.pcgamer.com/hearthstones-team-archon-releases-hosty-hours-after-cheating-allegations/>, Jan 2015.
- [25] *Hearthstone players*. The art of netdecking. <http://hearthstoneplayers.com/art-netdecking/>, Jun 2014.
- [26] Polygon. *Hearthstone now has 50 million players*. <http://www.polygon.com/2016/4/26/11511890/hearthstone-50-million-players-2016>, April 2016.
- [27] Claude E Shannon. Prediction and entropy of printed english. *Bell system technical journal*, 30(1):50–64, 1951.
- [28] Richard A Steinman and Mary T Blastos. A trading-card game teaching about host defence. *Medical education*, 36(12):1201–1208, 2002.
- [29] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *INTERSPEECH*, 2012.
- [30] Tempostorm. The meta snapshot. <https://tempostorm.com/hearthstone/meta-snapshot/meta-snapshot-36-blizzcon-hype>.
- [31] Steve Watts. Opinion: Why hearthstone's patron warrior nerf goes too far. <http://www.shacknews.com/article/91731/opinion-why-hearthstones-patron-warrior-nerf-goes-too-far>, Oct. 2015.
- [32] Wikipedia. *Blizzard warden*. [https://en.wikipedia.org/wiki/Warden\\_\(software\)](https://en.wikipedia.org/wiki/Warden_(software)).
- [33] Wikipedia. *Hypergeometric distribution*. [https://en.wikipedia.org/wiki/Hypergeometric\\_distribution](https://en.wikipedia.org/wiki/Hypergeometric_distribution).
- [34] Wikipedia. *Magic: The gathering*. [https://en.wikipedia.org/wiki/Magic:\\_The\\_Gathering](https://en.wikipedia.org/wiki/Magic:_The_Gathering), 1993.
- [35] Su-Yang Yu, Nils Hammerla, Jeff Yan, and Peter Andras. A statistical aimbot detection method for online fps games. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–8. IEEE, 2012.

# Pruning and Preprocessing Methods for Inventory-Aware Pathfinding

Davide Aversa

Department of Computer, Control,  
and Management Engineering  
Sapienza University of Rome  
Rome, Italy  
Email: [aversa@dis.uniroma1.it](mailto:aversa@dis.uniroma1.it)

Sebastian Sardina

School of Computer Science  
and Information Technology  
RMIT University  
Melbourne, Australia  
Email: [sebastian.sardina@cs.rmit.edu.au](mailto:sebastian.sardina@cs.rmit.edu.au)

Stavros Vassos

Department of Computer, Control,  
and Management Engineering  
Sapienza University of Rome  
Rome, Italy  
Email: [stavros@dis.uniroma1.it](mailto:stavros@dis.uniroma1.it)

**Abstract**—Inventory-Aware Pathfinding is concerned with finding paths while taking into account that picking up items, e.g., keys, allow the character to unlock blocked pathways, e.g., locked doors. In this work we present a pruning method and a preprocessing method that can improve significantly the scalability of such approaches. We apply our methods to the recent approach of Inventory-Driven Jump-Point Search (InvJPS). First, we introduce InvJPS+ that allows to prune large parts of the search space by favoring short detours to pick up items, offering a trade-off between efficiency and optimality. Second, we propose a preprocessing step that allows to decide on runtime which items, e.g., keys, are worth using thus pruning potentially unnecessary items before the search starts. We show results for combinations of the pruning and preprocessing methods illustrating the best choices over various scenarios.

## I. INTRODUCTION

Pathfinding is a common search-based technique that allows a character to move from one location to another within a game world in a reasonable way, e.g., avoiding obstacles and using the fastest way to go. Pathfinding is central in many game genres and significant work is carried out in the commercial videogames industry toward improving the believability of the navigation of characters such as making the resulting paths smoother and more realistic, e.g., [1], [2], [3].

Nonetheless, there are certain aspects related to character navigation that have been studied in academic artificial intelligence (AI) which have not been explored much in the game industry. In particular, some recent work investigates how existing pathfinding approaches can be extended to account for additional capabilities of the character beyond moving [4] so as to be able to answer questions of the form: “What is the fastest way for a character to go from A to B when they can also pick up items that open blocked pathways?”. From a theoretical perspective this is a special case of AI planning that merges navigational pathfinding with task-based deliberation. Certainly, such questions could be handled as a special case of STRIPS-like planning [5], a well-studied AI approach,<sup>1</sup> but the challenge for us is to identify practical restricted AI solutions that fit well in the videogame requirements.

<sup>1</sup>E.g., the International Conference on Automated Planning and Scheduling (ICAPS) is an established forum for researchers and practitioners in planning and scheduling: [www.icaps-conference.org](http://www.icaps-conference.org).

Our starting point then is the *Inventory JPS (InvJPS)* pathfinding algorithm [4], which extended the award-winning *Jump-Point Search (JPS)* technique [6], [7] to search over the space of paths that are feasible when the character is also able to collect (and use) objects that can open up blocked pathways. For simplicity, the analysis in [4] focused on the use of *keys* that may open *locked doors*. The experimental results over maps from the “Moving AI” benchmark [8] showed that InvJPS (i) adds little overhead in the cases where keys are not needed for finding a path, and (ii) has a runtime that is still practical, provided there are a small number of keys and a feasible path (with some of such keys).

Nonetheless, the results in [4] also demonstrated that the worst case scenario is in fact one that is not uncommon in practice, namely, when *there is no path* from the start location to the destination. The reason is that, because InvJPS seeks an *optimal path*, it ought to take into account all possible ways of moving and collecting/using keys. Intuitively, this can be seen as searching over many different “versions” of the map, one per possible combination of keys available. So, when there is no path whatsoever, the algorithm goes through *all* those maps before deeming the destination unreachable.

In this work we propose, then, two optimization techniques to address this issue. First, we develop a *pruning mechanism* to discard some of the map “variations” with the price of giving up optimality. The idea is to *favor picking up a key that are close-by*, even if they may not be useful after all, and *give up searching for a faster path without such key*. The resulting algorithm, which we shall refer as InvJPS+, provides the necessary parameters for exploring the trade-off between optimal paths and this type of “detours.” Our results show that, in the cases that no path exists, InvJPS+ yields an answer much faster than InvJPS, while in the cases that a path does exist, the overhead in the path length is reasonable.

The second technique involves a *preprocessing method* to speed up runtime performance even more. Concretely, the method assesses the door locations offline and generates a *connectivity graph*, which supports the identification, at runtime, of subsets of keys that are sufficient to find a path. In this way, by looking only into combinations of such key sets, InvJPS+ can be constrained to search over a *fixed small upperbound*

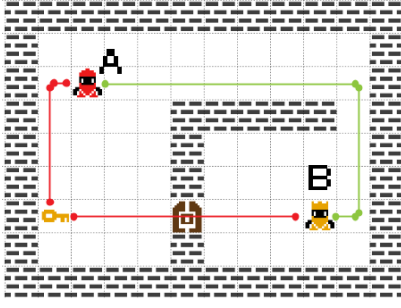


Fig. 1. An inventory-aware pathfinding example.

of keys. Different strategies on selecting such sets offer then an additional trade-off, between runtime performance and completeness. We investigate how these strategies work with InvJPS+, and under which conditions they offer an inventory-aware pathfinding solution that is practical over all cases.

## II. INVENTORY-AWARE PATHFINDING

In this paper we deal with *Inventory-Aware Pathfinding* according to which the character may acquire capabilities that allow them to open blocked pathways and use them to find paths in order to move from one location to the map to another. We follow the assumptions of [4] and also focus on locked doors and keys as a convenient way to talk about blocked pathways and capabilities that can be acquired.

In particular, the inventory-aware pathfinding problem is an *one-shot search problem* as follows. Given:

- a grid-based map  $M$  as a set of locations  $\{m_{11}, m_{12}, \dots\}$ ;
- a set  $O \subseteq M$  of locations that are blocked (and cannot be traversed by the agent);
- a function  $adj : M \mapsto 2^M$  denoting the adjacency relation among locations ( $adj(x)$  denotes the set of locations that adjacent to location  $x$ );
- a set of items  $\mathcal{I}$  (e.g., objects or capabilities) that may be scattered in the map (and that the agent is able to acquire when co-located);
- a function  $obj : M \mapsto 2^{\mathcal{I}}$  stating the items present in each location ( $obj(x) = \emptyset$  denotes no items at node  $x$ );
- a function  $req : M \mapsto 2^{\mathcal{I}}$  stating which items (in combination) are required to traverse a location; and
- a start location  $S$  and destination location  $G$ ,

Find an *optimal* (i.e., *shortest*) path  $\sigma = x_1, x_2, \dots, x_n$ , with  $n \geq 1$ , such that:

- $x_1 = S$  and  $x_n = G$ ;
- $x_{i+1} \in adj(x_i)$  and  $x_i \notin O$ , for every  $i \in \{1, \dots, n-1\}$ ;
- $\bigcup_{j < i} obj(x_j) \subseteq req(x_i)$ , for every  $i \in \{1, \dots, n\}$ .

That is, we are interested in finding the shortest path from start to destination under the constraint that some locations along the path may require the agent to have previously visited other special locations (last point above).

Note that the map may be dynamic but the algorithm considers a static snapshot of the map with the location of keys and doors known. Also, all keys are assumed to be acquired immediately upon reaching a location that has keys, and we do not take into account any cost for picking up or carrying

them (essentially an inventory of unbounded size is assumed). Finally, one key can open multiple doors and one door can be opened by a single key (or a single combination of keys).

For example in Fig. 1 the character wants to get from A to B, there is one key, and one door. The character can go around to reach the destination or pick up the key and go through the door; the optimal (shortest) path depends on the position of the character, the key, the door, and the destination. In the example, the cost for the two paths is the same but a change in the position of the key can make one of the two shorter.

### A. Jump-Point Search (JPS)

JPS [6], [7] is a search type pathfinding approach designed for uniform cost grid-based domains. Its key insight is that when searching a grid there is a great deal of symmetry: if paths have the same start and end location, then what occurs in between is basically the same set of moves (vectors) but in a different order. Thus, instead of expanding all reachable nodes, JPS “jumps” from a potential turning point to another turning point, expanding only those nodes that might require a change of direction, essentially searching over an abstracted map consisting of these so-called *jump-points*. There are formal rules that specify which nodes qualify as jump-points and a proof that guarantees that this abstraction maintains optimality.

As explained in [9], JPS operates, technically, like A\* by working through the grid systematically, maintaining an open list, and selecting and opening nodes from that list using the same best-first evaluation function as A\*. When a node  $n$  in the open list is expanded, A\* retrieves all its unblocked adjacent nodes and adds them to the open list. Instead, JPS generates on-the-fly a *vector of travel* from  $n$  through the adjacent nodes that follow the direction of the search and adds in the open list the resulting nodes in which the traveling direction in the optimal path may change.

### B. Inventory-aware Jump-Point Search (InvJPS)

As seen in [4], JPS can be further elaborated in a principled way to accommodate inventory-driven path planning. The new algorithm, called *inventory-driven JPS* (InvJPS) is obtained by modifying JPS in three simple ways.

First, InvJPS extends the node representation so that it includes also the inventory that the character is carrying along with the current location. So, for a map  $M$  a node for InvJPS is a pair  $\langle x, I \rangle$  where  $x$  is a location in  $M$  and  $I$  is a subset of elements from set  $\mathcal{I}$  of all possible items. During search when an agent is at a location that has items, these are placed all in the inventory instantly. The items that a character carries allow them to traverse also nodes that are marked as blocked but are labeled by  $req$  with a set of items included in the inventory  $I$ .

Second, InvJPS treats any location containing a key as an “intermediate goal”. During the search over jump-points when a location is found that contains a key, the corresponding node is considered an *inventory jump-point* and is added to the open list along with the other jump-points.

Third, when an inventory jump-point is expanded, InvJPS considers vectors to *all possible directions*, including also

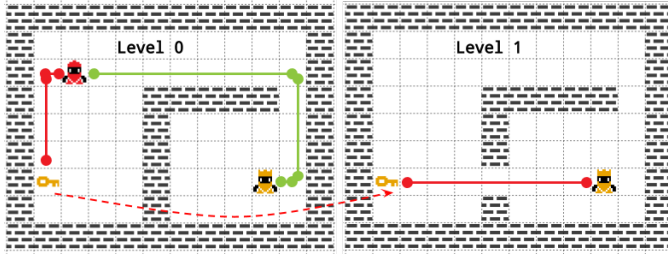


Fig. 2. A key “teleports” the character to a different version of the map.

going back undoing the path traversed so far. The fact is that, with the new inventory acquired, nodes that looked “blocked” before may have now become traversable.

### III. INVJPS+: A PRUNING-BASED VARIANT FOR INVENTORY-AWARE PATHFINDING

We introduce InvJPS+, a variant of InvJPS that can be fine-tuned with two parameters,  $\theta$  and  $r$ , to be practical in many of the cases that original InvJPS suffers. The main idea is to give up optimality in the sense of forcing the search to do *short detours* in order to pick up and carry keys, and disregard the search alternatives that do not get to acquire these keys.

A central idea behind original InvJPS is the concept of *key-levels* (or *item-levels* in the general case that we consider items that unlock capabilities). The idea is that searching in the extended space of inventory-aware nodes can be seen as searching over multiple instances of the original map, one per key combination that can be acquired: for each set of keys  $K$ , there is a copy of the original map in which doors opened by keys in  $K$  is replaced by a free tile and the rest of the doors are replaced by a blocked tile. Locations in the map that contain keys act then as if they “teleport” the character from one instance of the map to the other. For example, Fig. 2 shows the same map in Fig. 1 as viewed by the InvJPS algorithm in terms two key-levels defined by the only key in the map. Technically, a key-level  $L_K$  is the set of nodes  $\langle x, I \rangle$  in the search space which have an inventory  $I = K$ .

InvJPS+ exploits the fact that the number of reachable locations is monotonically increasing as more keys are acquired, e.g., obtaining a key can allow the character to visit locations that lie behind a closed door. As a consequence, if we have two nodes  $n$  and  $m$  with  $N$  and  $M$  keys respectively, both in the same location  $x$  but on different key-levels  $L_N$  and  $L_M$ , and if  $M$  is a superset of the keys in  $L_N$ , then node  $n$  *subsumes*  $m$ , in the sense that from  $n$  one can reach at least all those nodes that can be reached from  $m$ .

Of course, the cost of getting to  $n$  may be different than  $m$ , and this is where a trade-off arises. We choose to quantify with  $\theta$  the amount of extra cost we allow for such nodes like  $n$  to suffer while being able to subsume a node like  $m$  and filter it out of the search procedure. This  $\theta$  essentially quantifies how *long* can a detour be for going to location  $x$  but with getting some keys that may be nearby. Moreover it may be reasonable for a node  $n$  to subsume a node  $m$  that is *not in the same location*  $x$ , but nearby at a short distance. A second radius

#### Algorithm 1 $n$ Is Better Than $m$

---

```

function ISBETTERTHAN(nodeA, nodeB,  $\theta$ )
  moreKeys  $\leftarrow$  nodeB.keys  $\subset$  nodeA.keys
  costLess  $\leftarrow$  nodeA.g  $\leq$  nodeB.g +  $\theta$ 
  return moreKeys and costLess

```

---

parameter  $r$  is therefore considered in InvJPS+ that allows to relax the location conditions of filtering.

We first discuss the details of these parameters and then, considering also the implementation effort required, we focus on three variants: (i) “*optimal InvJPS+*” that preserves optimality, (ii) “*local InvJPS+* with  $\theta$ ” that performs “local” filtering with respect to the  $r$  parameter, and (iii) “*global InvJPS+* with  $\theta$ ” that performs “global” filtering. Then we look into indicative values for  $\theta$  and report on the effort and precision of the approaches over benchmark maps. Finally, we summarize some concluding points at the end of the section.

#### A. Parameter $\theta$ : Filtering nodes to favor short detours

The basis for our extensions of InvJPS is the idea that whenever there are nodes in the open list over the same location  $x$ , we want to look into the cost and the inventory of each one and decide whether one should subsume the other. InvJPS+ with threshold  $\theta$  looks for cases where  $inv(n_1)$  is a superset of  $inv(n_2)$  and  $g(n_1) - g(n_2) < \theta$ , in which case  $n_2$  is *filtered* and removed from the open list. InvJPS+ performs this check, as shown in Algorithms 1,2, every time a new node  $n$  is inserted in the open list. If the check is positive, the new node is directly discarded (and put in the closed list).

InvJPS+ with  $\theta$  is a simple extension that can be easily implemented over InvJPS (or any inventory-aware pathfinding approach), and when  $\theta = 0$  it also maintains optimality. As the search procedure needs to access nodes in the open and closed list based on their position, a hash table or similar structure is required to allow this lookup to be performed efficiently.

*Proposition 1:* InvJPS+ with  $\theta = 0$  returns optimal paths in terms of path length.

So far we looked into InvJPS+ with  $\theta$  that may filter a node in favor of another at the same location  $x$ . Nonetheless, there are cases in which we would like the search to also consider comparing nodes that are not exactly on the same location  $x$ . We describe first a case that is specifically related to the way InvJPS works over jump-points, and then a case that applies to all inventory-aware search algorithms.

#### B. Parameter $r$ : Relaxing filtered jump-point location

Looking at the different key-levels of the map it is often the case that jump-points are not aligned with each other among different levels, e.g, when going to a room and coming back the jump-points of entering and leaving the room may not be on the same location  $x$ . A simpler case can be seen in Fig. 3, where going around an obstacle in fact is through jump-points at different locations, depending whether a nearby key has been picked up first. This can be easily verified following the rules for generating jump-points of JPS (hence InvJPS), and has to do with the fact that keys introduce new starting points

---

**Algorithm 2** Filtering on Push

---

```
▷ ThereIsABetterNodeThan( $n$ ) looks in open and closed list for a
node  $m$  such that IsBetterThan( $m, n$ ) is True.
if ThereIsABetterNodeThan( $n$ ) then
    closedlist.append( $n$ )
else
    subsumed  $\leftarrow$  FindSubsumedBy( $n, \text{openlist}$ )
    SetAllVisited(subsumed)
    if DescendantPruningIsActive then
        subsumed  $\leftarrow$  FindSubsumedBy( $n, \text{closedlist}$ )
        for  $cNode \in \text{subsumed}$  do
            DescendantPruning( $cNode$ )
    openlist.push( $n$ )
```

---

for vectors that are considered only in the key-level when the corresponding node is introduced in search.

Therefore, in order to find good filtering candidates we may need to look “around” the node in question over a radius  $r$ . This allows to fine-tune the behavior of InvJPS+. For practical purposes related to implementation we distinguish between two extreme cases: when  $r$  is a small number of nodes, e.g., 0-3, we refer to the resulting approach as *local InvJPS+* (with  $\theta$ ), otherwise, when  $r$  is infinite allowing to look for filter candidates in all of the open and closed list we refer to the resulting approach as *global InvJPS+* (with  $\theta$ ). It will become more clear in the next part why global filtering is useful; for now note that the conditions of the  $\theta$  parameter ensure that we only filter out nodes that are subsumed by other nodes representing appropriate (short wrt  $\theta$ ) detours.

As for the implementation, when adding a node  $n$  in the open list essentially we need to call Algorithm 1 and compare with every node in the open and closed list that is within a radius distance  $r$  from the position of  $n$ . In local InvJPS+ this requires a number of calls for all positions around  $n$ , while in the global InvJPS+ this requires traversing the lists and comparing with all nodes there.

### C. Retrospective pruning

We now note an additional pruning mechanism for removing nodes that are not captured by the previous approach. InvJPS+ with threshold  $\theta$  and radius  $r$  removes all nodes in the *open list* that are “worse” than a current node in question. But what if there is a node in the *closed list* that is worse in this sense? In this case we cannot filter the subsumed node (as it is already in the closed list), however, we should in fact remove all nodes that are *descendants* of this node and are currently in the open list. Another way to look at it, is that part of the fringe of the search that needs to be pruned may not be still exactly at the location  $x$  of the new node that is pushed in the open list.

About implementation, a simple way to handle it is to extend the search nodes with two other fields: a list of all the children of  $n$ , and a “pruned” flag. When a node  $n$  in the closed list is subsumed by a new node, we simply set the flag as true for  $n$  and recursively for each children node of  $n$  until we reach nodes in the open list or we found other pruned nodes. Every node flagged as pruned is then filtered immediately when taken out from the open list.

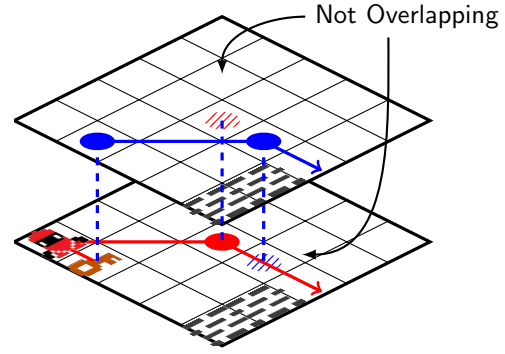


Fig. 3. Jump-points not overlapping over different key-levels.

### D. Experiments

Now we proceed to show some experimental results for InvJPS+ with threshold  $\theta$  for local and global filtering. To that end, we developed a benchmark on top of python-based P4<sup>2</sup> path planing simulator [9]. In local InvJPS+ we chose  $r = 2$ , as preliminary experiments showed this is able to handle well the case of non-overlapping jump-points we identified in part B, and in global InvJPS+ we also employed the retrospective pruning mechanism in order to account for the case of non-overlapping jump-points we identified in part C. As the scenario that (regular) InvJPS suffers is when no path can be found, we focus on this so as to see the worst case for the effort needed. Based on our analysis, it is illustrative to see thresholds of 0, 50, 100 and 200 nodes. We tested this scenario using 4 indicative maps taken from the *Moving AI Benchmark* [8], namely “AR0011SR”, “AR0012SR”, “AR0602SR”, and “AR0013SR”. In each map we ran several pathfinding queries from a random starting position to an unreachable destination. We focus on expanded nodes to highlight the benefit of filtering in pruning the search space.<sup>3</sup>

In Fig. 4 we see that (regular) InvJPS is exponential in the number of keys, because, when there is no path the algorithm is forced to search the full search space. The optimal version of InvJPS+ is still exponential with a small improvement. Allowing for suboptimal solutions, though, provides a wide range of speedup factors. The two groups of suboptimal algorithms, namely local InvJPS+ and global InvJPS+, perform much better with a much slower exponential behavior, that is almost linear in the range of 0-10 keys for global InvJPS+.

It is important, however, to measure the degree of suboptimality of the algorithms. In order to do this we used regular maps in which we added unnecessary keys, in which case the optimal path can be found without picking up and using keys. Regular InvJPS would be able to find the shortest paths that do not use keys. On the other hand, InvJPS+ with threshold  $\theta$  greater than zero favors detours for picking up keys, and the paths found are expected to be suboptimal (i.e., longer) due to these detours. As in this case the whole search space would not be explored, for higher variety on the results we used 20

<sup>2</sup>P4 path planing simulator: <https://bitbucket.org/ssardina/soft-p4-sim-core>.

<sup>3</sup>A runtime comparison between InvJPS and InvJPS+ should take into account that the filtering function takes 7% of the total search time on average.



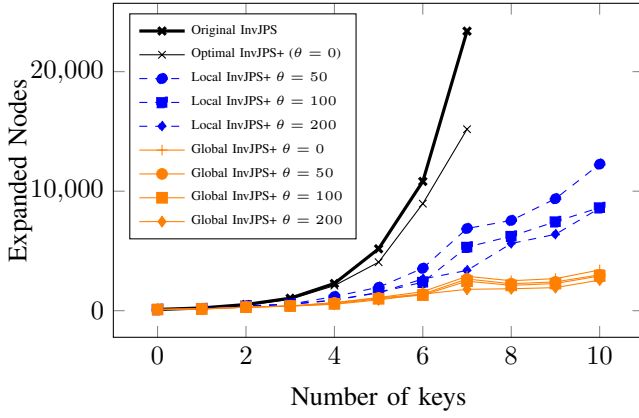


Fig. 4. InvJPS+ over scenarios that no path exists.

maps from the MovingAI benchmark [8], along with a subset of the queries that come along with the maps.

The results show an evident increase in the path length for global InvJPS+ with a maximum of a 30% longer path when there are 50 unnecessary keys on the map, compared to optimal InvJPS+. On the other hand, the local variant shows a negligible 3-5% increase. This is reflected in the expanded nodes, shown in Fig. 5. Note also that the increase of expanded nodes for the suboptimal variants compared to the optimal InvJPS+ is very small in actual numbers over the cases when paths exist. This is because the challenging cases with high complexity are those that no path exists. In particular, note that in Fig. 5 the expanded nodes are in the order of hundreds while in Fig. 4 are in the order of tens of thousands.

Another way to assess InvJPS+ is to compare the expanded nodes with those expanded by regular JPS in the same cases, as a way to understand the runtime effort. Note that InvJPS+ for the case of 0 keys expands exactly the same nodes as regular JPS. Observing then the two graphs we see that for a small number of keys InvJPS+ is comparable to JPS, hence having a practical runtime for use in a real videogame setting. Nonetheless, in general there may be many keys on the map that are not useful for a particular pathfinding query but bring the complexity up anyway. With the preprocessing step we introduce next, we can address this by identifying a small subset of keys that are useful for each pathfinding query, and offer InvJPS+ approaches that are practical over all scenarios.

#### IV. A PREPROCESSING METHOD FOR INVENTORY-AWARE PATHFINDING

In this section we introduce a preprocessing method that aims to help the inventory-aware search procedure focus on a promising subset of keys in the map and disregard the rest. As the search complexity is intrinsically exponential to the number of keys on the map, disregarding even a small amount of keys can significantly improve overall performance. However, except for some trivial case such as when keys are not linked to any door, it is not easy to identify which key is useful and which is not during search. We have developed a preprocessing method for this task that is divided in two steps: (i) an *offline* preprocessing step that generates a logical

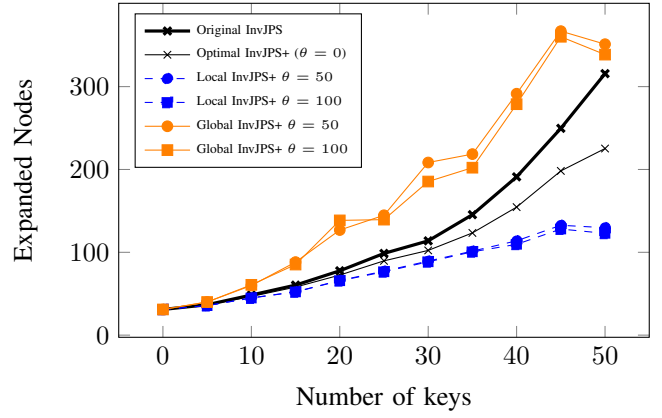


Fig. 5. InvJPS+ over scenarios that keys are not necessary for the path.

description of the connections between regions and doors on the map and (ii) an *online* pre-search step that, given the starting and destination position and the position of every key, can generate a subset of *necessary* (but, in general, not *sufficient*) keys for the specific path in question.

As we will see there are different strategies for generating such key-sets offering a trade-off between *completeness* (to be able to find a path if one exists) and efficiency. Also, the preprocessing method is independent from the actual search method and can be used with any inventory-aware pathfinding approach. Here we will show results with the InvJPS+ variants.

##### A. Offline preprocessing step

The goal of the offline preprocessing step is to analyze the game map and extract information on the different regions that are connected with doors. The algorithm takes as input the map representation annotated with the location of every door and generates two outputs. The first one is the *connected area labeling map* ( $\mathcal{L}_M$ ), a copy of the map in which every location is labeled with an integer number and with the property that  $label(x)$  is equal to  $label(x')$  if and only if it is possible to find a path between locations  $x$  and  $x'$  without using any door. The second output is a *doors connectivity graph*  $\mathcal{G}$  in which each vertex is a label in  $\mathcal{L}_M$  and each edge is a door connecting  $l_i$  with  $l_j$  annotated with the connecting door  $d$ .

The labeling map is constructed assuming that all doors are closed and running a *binary connected area labeling* algorithm, a fast parallelizable algorithm originally designed for image segmentation [10]. The connectivity graph, instead, is generated by searching for doors along the perimeter of each area and adding a connection edge for each one. This algorithm depends only on the topology of the map and on the position of the doors (i.e., the position of keys is not required), so it is possible to generate the pair  $\langle \mathcal{L}_M, \mathcal{G} \rangle$  for every map at design phase and serialize the output on disk as appropriate.

##### B. Online pre-search step: 4 strategies for promising keys

This step is intended to be performed at the beginning of every pathfinding query. It takes as input the starting and goal location, the map, and the preprocessing pair  $\langle \mathcal{L}_M, \mathcal{G} \rangle$  for the given map. The output is a set  $\hat{K}$  of keys that act as hints so



---

**Algorithm 3** Compute the *Necessary-Doors-Sets*

---

**Precondition:**  $s$  starting location,  $g$  goal,  $labels$  the preprocessed label map,  $cGraph$  the connectivity graph.

```
startLabel  $\leftarrow labels[s]$ 
endLabel  $\leftarrow labels[g]$ 
paths  $\leftarrow []$ 
if startLabel = endLabel then
  add(paths,[])
currentLabel  $\leftarrow endLabel$ 
openPaths = [(null, currentLabel)]
while openPaths is not empty do
  currentPath  $\leftarrow pop(openPaths)$ 
  currentLabel  $\leftarrow currentPath.label$ 
  if currentLabel == startLabel then
    append(paths,currentPath)
    continue
  adjacentLabels  $\leftarrow adjacent(cGraph, currentLabel)$ 
  for label, door  $\in adjacentLabels$  do
    if label  $\notin currentPath$  then
      newPath  $\leftarrow insert(currentPath, (label, door))$ 
      append(openPaths, newPath)
return PathToNDS(paths)
```

---

---

**Algorithm 4** The Recursive Iteration

---

**Precondition:**  $s$  starting location,  $g$  goal,  $map$  the map,  $labels$  the preprocessed label map,  $cGraph$  the connectivity graph.

```
K  $\leftarrow NecessaryKeys(s, g, labels, cGraph, map)$ 
for  $i \leftarrow \{0, n-1\}$  do
  newK  $\leftarrow K$ 
  for  $k \in K$  do
    tmp  $\leftarrow NecessaryKeys(s, k, labels, cGraph, map)$ 
    newK  $\leftarrow \text{union}(newK, tmp)$ 
  if K = newK then
    break
  else
    K  $\leftarrow newK$ 
return K
```

---

that the inventory-aware search method can restrict the search to consider only keys in  $\hat{K}$  and ignore the rest. Depending on the strategy adopted,  $\hat{K}$  may be a “best bet” small set that has high probability to be sufficient, or a larger “complete” set of keys in the sense that if a path exists then it is guaranteed that the destination can be reached with this subset of keys.

We now specify some strategies for generating  $\hat{K}$ . First we use the connectivity graph in order to compute the so-called *necessary doors sets* (NDS): a set of door-sets each of which corresponds to an acyclic path in  $\mathcal{G}$  from the area of the starting position to the area of destination. Algorithm 3 shows how NDS can be computed. NDS contains high-level plans in terms of which doors the character need to pass in order to reach the destination. This set of doors-sets is necessary in the sense that if a path exists then one of the doors-set is necessary to be in the path. However, as the NDS does not take into account the positions of the keys, it is easy to see that such a doors-set may be not sufficient. For example, reaching door  $d$  in the doors-set  $\{d\}$  option may in reality require that the character passes through another door to pick up the key for  $d$ . This can be computed at runtime using the keys location by expanding

the high-level plan that a doors-set represents.

From NDS we can easily compute the corresponding *necessary keys sets* (NKS) by substituting each door in every set in NDS with a key that is required to open it. From the NKS we can generate the set  $\hat{K}$  to be used in the inventory-aware pathfinding search. This set is a *candidate set* that we can specify with different strategies. Similar to the previous discussion for doors, under conditions we can also show that such set is “complete” in the sense that, using  $\hat{K}$ , InvJPS+ can always find a solution if a path exists. We study the behavior of some variants based on the following two simple strategies.

a) *FirstNKS-n*: A simple approach is to take the *smallest set in NKS* ignoring all the others. This solution is greedy in the sense that it picks the option that looks most promising in terms of efficiency as it uses the minimal amount of keys. It is expected to have fast runtime as well as to produce suboptimal paths because in general the location of the keys matters for finding optimal paths (e.g., a path with more keys may be optimal because they are near the character).

b) *AllNKS-n*: A more cautious approach is to define  $\hat{K}$  as the union of all the sets in NKS. It is expected to have slower runtime as it will use more keys but a better performance with respect to optimality as all options (and the corresponding keys) are considered in the search.

Both approaches are incomplete in general: the generated  $\hat{K}$  may not be sufficient for reaching the destination because a key may be behind a closed door, as we discussed earlier. Nonetheless, at runtime the locations of keys is available, therefore we can identify which keys are behind doors and which are the required extra keys, and add them to the set  $\hat{K}$ . This can be done with a recursive procedure that starts with the set  $K_0$  generated directly by the NDS as before and at each step generates a set  $K_{i+1}$  that enlarges  $K_i$  with keys that allow to reach keys in  $K_i$ . Algorithm 4 describes this recursive step. Then FirstNKS- $n$  and AllNKS- $n$  operate in this manner with the  $n$  parameter denoting the number of recursion steps.

We identify two border cases for each approach: one when  $n = 1$  which we call “*optimistic*” and one when the recursion step is repeated until a fixed point is reach which we call “*recursive*”. This leads to four variants, namely FirstNKS-Optimistic, FirstNKS-Recursive, AllNKS-Optimistic, AllNKS-Recursive. The last one is complete in the following sense.

*Proposition 2:* If there exists a path for an inventory-aware pathfinding instance, then it can be reached using a subset of the keys generated by AllNKS-Recursive.

### C. Experiments

We now proceed to evaluate the preprocessing method, in particular the four variants FirstNKS-Optimistic, FirstNKS-Recursive, AllNKS-Optimistic, AllNKS-Recursive, in combination with two filtering methods from the previous section, namely optimal InvJPS+, and local InvJPS+ with  $\theta = 100$  that are representative of the space of behaviors.

We used a map from *Baldur’s Gate II* (“AR0602SR” of the Moving AI benchmark [8]) and implemented the following setting. We used 14 doors which divide the map in 13 different

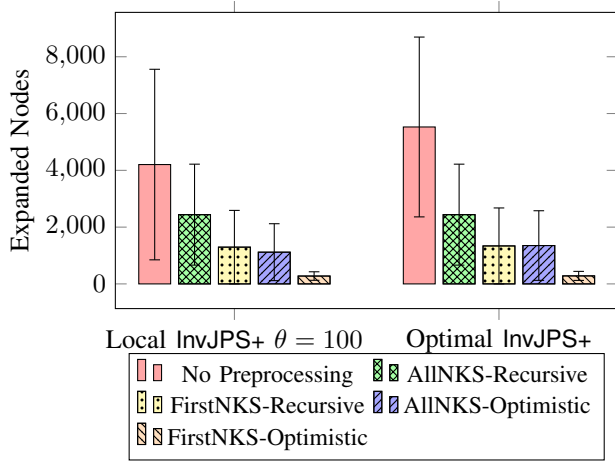


Fig. 6. Average expanded nodes over all generated paths.

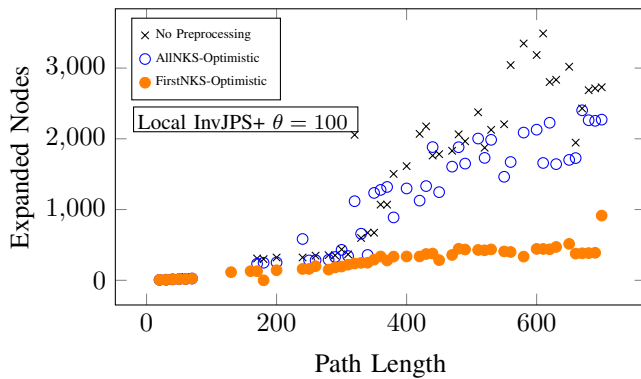


Fig. 7. Expanded nodes with respect to path length for the subset of paths that optimistic approaches are complete (and coincide with the recursive).

areas. We manually distributed 14 keys on the map (one for each door) in such a way that every key is reachable starting from the central area. This is ensured by avoiding cyclic requirements, e.g., a two-step cycle is when reaching key  $k_1$  requires to get  $k_2$  and reaching  $k_2$  requires to get  $k_1$ . Then we specified 300 random paths that start from the central area, which guarantees that there exists a solution for every path. In Fig. 6 we see the average expanded nodes over the set of paths for each combination of approaches.

As expected, optimistic approaches expand the least amount of nodes due to being incomplete: FirstNKS-Optimistic and AllNKS-Optimistic show an average of 61% completeness (with small differences in the combined search approach), that is, they return a solution for 61% of the solvable pathfinding instances. There are a few additional interesting outcomes. First, AllNKS-Optimistic performs similarly to FirstNKS-Recursive in terms of expanded nodes, and the latter is complete in this scenario even though not complete in general. Second, the average performance of the preprocessing approaches is almost unaffected by the suboptimal nature of the combined search approach. As the suboptimal behavior in InvJPS+ is mostly caused by detours for picking up unnecessary keys, with the preprocessing step such detours are avoided since most of the unnecessary keys are removed in advance.

In Fig.6 we average over different scenarios, e.g., cases where paths exist or not, and over different behaviors, e.g. with respect to completeness. In Fig.7 we look closer on the subset of paths where the approaches are complete, in order to compare the same search behavior, and report on the average expanded nodes over the path length. In this way it is also more clear to see how the performance scales in terms of detours that typically happen in longer paths.

Finally, we verify that the following common border cases are handled by our preprocessing method with a similar (or better) performance as regular pathfinding: (i) when there are only unnecessary keys the preprocessing method allows to disregard all keys (e.g., InvJPS+ would work exactly as JPS), (ii) in the case in which there is no path from start to destination by a fixed obstacle (e.g., a wall or door that cannot be opened), the preprocessing method will identify this cases and return with an answer in the pre-search step.

## V. RELATED AND FUTURE WORK

There are several works for improving A\* in path planning, e.g., RWA\* [11], RTA\* [12], and DAS [13], and preliminary efforts to extend JPS to non-uniform have been reported in [9]. All these approaches can be extended to deal with inventory-aware pathfinding, for instance, by extending the search state with inventory information. However, because the exponential nature of inventory-aware search lies in the exponential increase of the search space with every additional key, they are all expected to suffer the same degradation as the original InvJPS or Inventory A\*. At the same time, they are also expected to enjoy the benefits from our filtering and preprocessing methods.

Other works try to address the pathfinding problem by using hierarchical abstraction, e.g., Near-Optimal Hierarchical Pathfinding (HPA\*), by state-space pruning, e.g., [14], Swamps [15], or a combination of the former by exploiting path symmetry, e.g., Rectangular Symmetry Reduction (RSR) [16]. In their current form, none of these existing approaches address inventory-driven path planning, but can also be extended in a similar way. Hierarchical approaches can benefit from the partial decoupling between the planning and pathfinding problem by considering the inventory only on the higher level of abstraction. Moreover, we think hierarchical approaches can obtain potential greater benefits from the preprocessing approaches presented in the previous sections.

Another promising direction is to use the NKS information to obtain more informed heuristics. As InvJPS+ uses regular heuristics, the search is attracted towards the goal even in those key-levels in which it is impossible to find a solution. In these levels, the heuristic should instead move the search horizon *away from the goal* and *toward promising keys as sub-goals*. As each path must have one of the keys-set in NKS as a subset of the used keys, this can identify those key-levels where we should avoid using the goal heuristic.

Inventory-aware pathfinding is essentially a special case of the so-called classical planning problem. There is work that analyzes the parametrized complexity of planning, such as [17]

TABLE I  
BEST APPROACHES FOR VARIOUS PRACTICAL SCENARIOS

Scenario	Suggested Algorithm for Runtime	Suggested Algorithm for Optimality
Small static map with low keys complexity	No Preprocessing Optimal InvJPS+	No preprocessing Optimal InvJPS+
Large static map with low keys complexity	No Preprocessing Local InvJPS+ $\theta = 100$	No Preprocessing Local InvJPS+ $\theta = 100$
Map rapidly changing with high keys complexity	No Preprocessing Global InvJPS+ $\theta = 100$	No Preprocessing Local InvJPS+ $\theta = 100$
Static map with high keys complexity	FirstNKS-Recursive Preprocessing Local InvJPS+ $\theta = 100$	AllNKS-Recursive Preprocessing Local InvJPS+ $\theta = 100$
Static map with high keys complexity (without completeness constraints)	FirstNKS-Optimistic Preprocessing Local InvJPS+ $\theta = 100$	AllNKS-Optimistic Preprocessing Local InvJPS+ $\theta = 100$
Overall	AllNKS-Optimistic Preprocessing Local InvJPS+ $\theta = 100$	AllNKS-Recursive Preprocessing Local InvJPS+ $\theta = 100$

which summarizes the existing results and the connections between them. Nonetheless, the existing results for parametrized complexity of classical planning e.g., in terms of the number of propositional actions, maximum number of occurrences of a variable, etc, are not appropriate for characterizing inventory-pathfinding classes. This is because when bounded the studied parameters do not allow for expressing interesting inventory-aware pathfinding problems. On a different direction, perhaps a practical way to characterize the difficulty of inventory-aware pathfinding instances can be investigated by means of the number of recursion steps needed to achieve completeness for the FirstNKS- $n$  and AllNKS- $n$  preprocessing approaches.

## VI. CONCLUSION

In this work we look into the problem of inventory-aware pathfinding and present a pruning and a preprocessing method for improving the performance of search approaches in this domain. Inventory-aware pathfinding solutions can be useful to better support characters (players or non-players), so that when the game narrative engine (or player) directs a character to navigate to a certain location, the underlying path planner can figure out how to achieve this also by collecting and using objects if necessary. Observe that, when using a regular (not inventory-aware) navigational approach this task may fail despite the fact that an actual solution may exist.

We evaluated these methods over the only related approach in the literature (to the best of our knowledge), namely the so-called Inventory-Driven Jump-Point Search (InvJPS). Our results show that with these methods we remedy the problem of InvJPS that shows exponential, hence not useful in practice, behavior in the common scenario that no path exist for a given pathfinding instance. Moreover, our methods provide a space for fine-tuning the intended behavior in terms of trade-offs between performance on the one side and optimality and completeness on the other side. The choice of which method to implement depends on the requirements of the videogame application. In Table I we summarize the best approach for some common scenarios in terms of runtime and optimality. Finally, we note that the proposed methods are independent of

the search algorithm used and can be applied to any inventory-aware pathfinding solution.

## REFERENCES

- [1] M. Pinter, "Toward more realistic pathfinding," 2001. [Online]. Available: [http://www.gamasutra.com/view/feature/131505/toward\\_more\\_realistic\\_pathfinding.php](http://www.gamasutra.com/view/feature/131505/toward_more_realistic_pathfinding.php)
- [2] X. Xu and K. Zou, "Smooth path algorithm based on A\* in games," in *Advances in Computer Science, Environment, Ecoinformatics, and Education*. Springer, 2011, pp. 15–21.
- [3] R. H. Abiyev, N. Akkaya, E. Aytac, I. Günsel, and A. Çağman, "Improved path-finding algorithm for robot soccers," *Journal of Automation and Control Engineering Vol.*, vol. 3, no. 5, 2015.
- [4] D. Aversa, S. Sardina, and S. Vassos, "Path planning with inventory-driven jump-point-search," in *11th Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015, extended version at <http://arxiv.org/abs/1607.00715arXiv:1607.00715> [cs.AI].
- [5] R. E. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, pp. 189–208, 1971.
- [6] D. D. Harabor and A. Grastien, "The JPS pathfinding system," in *Proc. of the Annual Symposium on Combinatorial Search (SoCS)*, 2012.
- [7] —, "Improving jump point search," in *Proc. of the Int. Conference on Automated Planning and Scheduling (ICAPS)*, 2014, pp. 128–135.
- [8] N. Sturtevant, "Benchmarks for grid-based pathfinding," *Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 144 – 148, 2012.
- [9] P. Masters, "Extending the use of Jump Point Search: Failure and successes," 2014, Honours Thesis. RMIT University.
- [10] Y. Han and R. A. Wagner, "An efficient and fast parallel-connected component algorithm," *Journal of the ACM (JACM)*, vol. 37, no. 3, pp. 626–642, 1990.
- [11] S. Richter, J. T. Thayer, and W. Ruml, "The joy of forgetting: Faster anytime search via restarting," in *Proc. of the Int. Conference on Automated Planning and Scheduling (ICAPS)*, 2010, pp. 137–144.
- [12] R. E. Korf, "Real-time heuristic search," *Artificial Intelligence*, vol. 42, no. 2, pp. 189–211, 1990.
- [13] A. J. Dionne, J. T. Thayer, and W. Ruml, "Deadline-aware search using on-line measures of behavior," in *Proc. of the Annual Symposium on Combinatorial Search (SoCS)*, 2011, pp. 39–46.
- [14] A. Botea, M. Müller, and J. Schaeffer, "Near optimal hierarchical pathfinding," *Journal of Game Development*, vol. 1, pp. 7–28, 2004.
- [15] N. Pochter, A. Zohar, and J. S. Rosenschein, "Using swamps to improve optimal pathfinding," in *Proc. of the Int. Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2009, pp. 1163–1164.
- [16] D. D. Harabor, A. Botea, and P. Kilby, "Path symmetries in undirected uniform-cost grids," in *Proceedings of the Ninth Symposium on Abstraction, Reformulation, and Approximation, SARA 2011*, 2011.
- [17] M. Kronegger, A. Pfandler, and R. Pichler, "Parameterized complexity of optimal planning: A detailed map," in *IJCAI*, 2013.

# Biometrics and classifier fusion to predict the fun-factor in video gaming

Andrea Clerico<sup>1</sup>, Cindy Chamberland<sup>2,3,4</sup>, Mark Parent<sup>2</sup>, Pierre-Emmanuel Michon<sup>3,4</sup>, Sébastien Tremblay<sup>2</sup>, Tiago H. Falk<sup>1</sup>, Jean-Christophe Gagnon<sup>5</sup> and Philip Jackson<sup>2,3,4</sup>

<sup>1</sup>Institut National de la Recherche Scientifique

<sup>2</sup>Université Laval

<sup>3</sup>Centre Interdisciplinaire de Recherche en Réadaptation et Intégration Sociale

<sup>4</sup>Centre de Recherche de l'Institut Universitaire en Santé Mentale de Québec

<sup>5</sup>Ubisoft Québec

**Abstract**—The key to the development of adaptive gameplay is the capability to monitor and predict in real time the players experience (or, herein, fun factor). To achieve this goal, we rely on biometrics and machine learning algorithms to capture a physiological signature that reflects the player's affective state during the game. In this paper, we report research and development effort into the real time monitoring of the player's level of fun during a commercially available video game session using physiological signals. The use of a triple-classifier system allows the transformation of players' physiological responses and their fluctuation into a single yet multifaceted measure of fun, using a non-linear gameplay. Our results suggest that cardiac and respiratory activities provide the best predictive power. Moreover, the level of performance reached when classifying the level of fun (70% accuracy) shows that the use of machine learning approaches with physiological measures can contribute to predicting players experience in an objective manner.

## I. INTRODUCTION

The video game industry has seen in the past decades an exponential market growth. Only in the last 10 years in the United States the revenues coming from computer and video games increased imposingly, from 7.3 in 2004 to 15.4 bn \$ in 2014. If the money spent on accessories and hardware is also considered, the figure grows to 22.41 bn \$ [1]. Players value that their money are more cleverly spent with videogames compared to movies or music [2]. One reason for the preference of videogames over other means of entertainment is that having an immediate feedback keeps the engagement high. One factor of retention of players is whether their gameplay experience is positive [1].

Quantifying the extent to which a players experience is positive throughout his or her gameplay remains a challenge. Efforts have been made by numerous playtest laboratories in the gaming industry but there are still shortcomings to the objective evaluation of the player's fun. Most experiments are conducted using empirical and subjective methods (i.e. interviews, focus groups questionnaires). In the academia, systematic methods have been explored, in particular the use of affective computing technologies [3], [4] and machine learning algorithms, but with limited combinations of physiological measures and its weak association with the fun-factor. In this context, the field of automated affective states computation has grown with the aim of creating affective video games.

Most of the work has focused on the comprehension of simple parameters of emotions and cognitive states studied in the affective Brain Computer Interface field [3], such as valence and arousal. Some authors also suggest that fun is not a unitary concept, which might add to the challenge of quantifying it. Lazzaros [5] model proposes 4 types of fun (e.g.: players seeking hard fun enjoy challenges, but players seeking people fun play for the social interaction). Poels, Kort and IJsselstein [6] suggest up to 9 dimensions to describe the video game experience. Evidence suggests that the different dimensions of fun are associated with distinct neurophysiological patterns [7]. These various reactions might increase the difficulty of using physiological measures to assess fun. Additionally, some authors suggest that fun is not directly measurable. Sweetser and Wyeth [8] suggest a model of enjoyment based on the flow theory [9] while others (e.g.: Calleja [10]) center their models on incorporation (i.e.: assimilation in the game while giving a sense of embodiment to the player). In a first effort to capture the relationship between several physiological and behavioral markers with the players experience, we chose to conceptualize fun as unidimensional since it is the easiest way for players to report their experience in relation to a videogame.

In the gaming literature, fun has been related to positive player reactions during a gameplay session. It has been linked to emotional experience but it is not considered as an emotion itself [11]. It is generally linked to different affective states, but as described in Pagulayan et al. [12], since games are intended to be fun, assessing fun implies assessing the overall quality of the game. The same authors [12] also point out that there might be a need to consider fun as being different in every user, thereby attributing a high contribution of individual influences to its assessment. Moreover, the work of Nacke et al. [13] mentions that when studying video games with physiological signals, there is a need to connect also other affective measures (e.g. behavioral responses) to establish relationships between the players experience and physiological responses (here we used a continuous measure of fun that will be described in Sec. II-B3). Thus, the fun-factor can potentially be analyzed in a study that combines objective measures such as physiological responses with subjective components qualifying the player's

experience [14], [15].

Computational models of fun have been designed in the past with the purpose of generating personalized game levels. For instance, in the work of Pedersen et al. [16], the authors were able to predict player emotions (e.g. fun, challenge and frustration) using preference learning and neuroevolution and the well known console game Super Mario Bros. A weighted non-linear computational model (e.g. artificial neural network) for reported emotions was constructed and the authors concluded that fun is the hardest dimension to model with a nonlinear perceptron and the least correlated with the features they extracted. Moreover, Shaker et al. [17] modeled player's fun value in platform games using a Multi Layer Perceptron Model. The authors obtained 69.66% of accuracy when modeling fun (using 58 features), but they also highlighted the limitation of post-experience analysis. For this reason they propose the use of physiological measures for further investigation, together with an increased number of features.

Several studies used physiological signals to quantify affective states during video game play ([18], [19], [14]). For instance, in the work of [15], the authors quantified emotional experience under the two dimensions of valence and arousal, to determine real-time emotional states. The latter were estimated using physiological signals such as Electrodermal Activity (EDA), Electrocardiography (ECG) and Electromyography (EMG). From the data of the two affective dimensions (i.e. valence and arousal), the authors were able to determine five distinct states during gameplay: boredom, challenge, excitement, frustration and fun. The problem that the authors revealed was that there are no guidelines for transforming assessments of arousal and valence into levels of fun in a continuous scale. Moreover, several affective states (i.e. boredom, frustration, challenge, anxiety, excitement) have been shown to correlate with ECG, EDA and EMG [20], but not a lot of effort has been made on the evaluation of the fun-factor itself.

Furthermore, using only signal processing techniques reduces a meaningful and natural interaction with the game. The introduction of a machine with automated emotional intelligence based on physiological responses would be able to learn negative and positive inputs and take care of player's need. Many studies have focused on detecting and learning emotional states, combining biometric signals and machine learning algorithms. In the work presented by Liu et al. [21], the authors studied different machine learning techniques for affective computing tasks. The authors used anxiety, engagement, boredom, frustration and anger as affective states and a questionnaire for self-reporting. The best performance was obtained using the Support Vector Machine (SVM) classifier in comparison with K-Nearest Neighbor, Regression Tree, Bayesian Networks. But developing video game affective systems, however, is a challenging task and many open problems and questions persist. For example, which physiological signal modalities should be combined to measure an affective dimension directly related to the level of player's level of

fun? Which features convey such task effectively? Which characteristics of the classifier are better adapted to the task at hand? Is it really relevant to address the detection of fun as an individual factor?

In recent years, effort has been made for real time adaptation of video games using biometrics. For instance in the work of Rani et al. [22], the authors classified three level of intensity (low, medium and high) for different emotions (engagement, anxiety, boredom, frustration and anger) using a Pong game and anagram puzzle. Parameters of the game were manipulated to elicit the required affective response. Cardiac activity, EDA, EMG and skin temperature were used along with four different classification methods. The best accuracy result was reached with the SVM classifier and 86% of accuracy. The results are promising but the work focused on the study of affective dimensions that only give an idea of a general fun level. Another example of work that used a more complex and dynamic system, designed to adjust several parameters in the game over time based on the player's physiological signals, was conducted by Chanel et al. [23]. In this case the authors reached 53% of accuracy using an SVM classifier when discriminating three emotional states (boredom, engagement and anxiety) using peripheral signals as EDA, blood pressure, heart rate, Respiration (RESP), temperature and self reported labels. The number of features extracted might be determinant for the final result. In this last example there is a few features (only 14 features were computed from the signals). The extraction of a large number of features allows the machine learning algorithm to have access to a larger amount of information, otherwise not detectable in the case of a limited number of features.

The purpose of this paper is the design of a predictive model that is able to discriminate the fun experience of players, based on their physiological responses, as measured by indicators of ECG, EMG, EDA and RESP, together with a self-reported continuous measure of fun and the best classification system. Our main goal is to identify a physiological signature of the fun-factor associated with positive gaming experiences, together with the best classifier traits in order to create an adaptive video game according to prediction of players' affective and cognitive states. To achieve this, we used an innovative method that allows for continuous rating of fun during gameplay. This method provides an advantage over subjective measures by providing a high-temporal resolution of the fun rating instead of a single value for a given time period. Furthermore, fun ratings are converted to trends (i.e.: ordinal scales), which is considered to reduce biases associated with human self-rating of emotions [24]. Finally, the present study uses an off-the-shelf and modern video game, thus increasing its ecological validity and application to future work.

## II. METHODS AND MATERIALS

### A. Participants

Sixty-two participants (5 women and 57 men) aged between 18 and 35 years ( $M = 25.9$ ,  $SD = 4.9$ ) were recruited from Université Laval and from Ubisoft Québec's volunteer

database to participate in a single two-hour experiment session. They all had prior game experience with the Assassin's Creed series, but had never played the specific title used in the current study. They all had normal or corrected-to normal vision and audition, and reported having no cognitive or neurological impairment. Participants received 20\$ for their participation at the end of the experiment.

## B. Apparatus and procedure

1) *Computer game*: Participants were asked to play the computer version of Assassin's Creed Unity -an action/adventure game developed by Ubisoft in 2014 (see a screenshot of the game at the top of Fig. 1)- with an Xbox 360 Controller. Two missions were specifically selected for this experiment: 'The prophet' and 'The escape'. The objective of this action-adventure game taking place in Paris during the French Revolution, played from a third-person view, is to complete pre-determined objectives to progress through the story. It is a non-linear gameplay, meaning that outside of the prefixed quests, the player can freely roam in the open world; thus giving the player more degrees of freedom compared to a linear gameplay.

2) *Procedure*: Participants read through a tutorial displayed on the computer screen describing the gameplay mechanics and explaining the procedure required to perform the different possible actions with the Xbox Controller. Participants were then familiarized with the game environment during a period of 5 minutes in which they had to complete seven objectives, all associated with the gameplay mechanics described in the tutorial (e.g., use a smoke bomb, climb up a building, assassinate an enemy while using a firearm). After successful completion of the objectives, a physiological resting baseline was recorded during a 3-minute period. Participants were instructed to remain calm and to refrain from moving while they were looking at a black fixation cross on a white screen and heard white noise via their headphones. Participants were then asked to play the first mission (presented in a counterbalanced order). The mission ended either when it was completed or after 15 minutes if participants failed completing it. The same procedure was repeated for the second mission.



Fig. 1: Graphic interface developed in order to give players a visual feedback of their fun ratings.



Fig. 2: USB controller (PowerMate, Griffin Technology) used to rate the level of fun.

3) *Continuous measure of fun*: After each mission, participants were required to watch a playback of their game session and to rate continuously the fun they felt during the game using a USB controller (PowerMate, Griffin Technology) shown in Fig. 2. This USB controller was an infinite control knob with no feedback (clicks) on the knob position. This controller was linked to a custom-made visual interface that allowed online graphic representation of the participant's evaluation of fun in real-time. As shown at the bottom of Fig. 1, the green areas correspond to positive levels of fun, while negative levels were depicted in the red areas. The level of fun was sampled at 30 Hz. Fun ratings were then transposed to a -100 to 100 scale for analysis.

4) *Psychophysiological measurement*: The player's physiological signals were collected during the two missions and during the replays (data from the replay were not used in this study). Electrodermal, cardiac, electromyographic and respiratory activities were recorded using a MP150 Biopac system (Biopac System Inc., Santa Barbara, CA). Electrodermal activity was measured using two pre-gelled electrodes placed on the palm of the left hand (the site was chosen in order not to have interferences with the controller). Cardiac activity was measured with three thoracic electrocardiogram electrodes placed in a Lead II configuration. The electromyographic signal was detected from the long abductor muscle of the right thumb with three pre-gelled electrodes placed on the right forearm. A respiration belt transducer placed around the player's chest measured respiratory activity (see Fig. 3 for the system's design). Cardiac activity was sampled at 1000 Hz whereas respiratory and electrodermal activities were sampled at 125 Hz using the Acqknowledge 4.3 data acquisition software. All psychophysiological signals were up-sampled to 1000 Hz (for synchronization purpose) and bandpass filtered (EDA 0-1 Hz, ECG 1-20 Hz, EMG 10-500 Hz and RESP 0-0.7 Hz). No further preprocessing has been conducted on the physiological signals since this study was designed as a prequel to a real time application. The signals and the self-assessment measure of fun were divided into epochs. The epochs were designed to last five seconds and an overlapping window of 2.5 seconds.



### C. Analysis of fun

Three different situations of the self-reported measure of fun were identified. The first two corresponded to an increasing and a decreasing trend in the fun rating, respectively. For classification purposes, they were identified as the first two classes and labeled fun ‘increasing’ or ‘decreasing’ (+1 and -1 respectively). The third possible situation arose when studying a stable segment of fun rating, which provided two additional classes. We considered as high-stable fun the ratings that were stable but at a level above the average fun value of the whole mission, and low-stable fun at levels below the average level. These two other classes were labeled as fun ‘above-average’ or ‘below-average’ (+0 and -0 respectively). To summarize, out of each epoch, the classification analysis will output one of the four classes described above (see Sec. II-F below). Depending of the sign of the class (positive or negative) the software will then apply the modification in the game. In particular, with a positive output (+1 and +0), fun was either increasing or stable but over the average, thus in a adaptive scenario the game would not need any adjustment because the player is categorized as satisfied. However, in the case of negative classes (-1 and -0), a real time adjustment of the game would become essential to increase the fun.

### D. Feature Extraction

A total number of 488 features were extracted from the four physiological signals as follows:

**ECG** The largest number of features is obtained from each electrocardiographic signal, such as 90 extracted features categorized into four different groups: spectral power with the fast Fourier transform (FFT) in multiple subbands, statistical components (average, min, max etc.), statistical features extracted from the analysis of separated parts of the QRS complex [25], and, finally, from the analysis of the Heart Rate Variability (HRV). Moreover, normalized versions of

these 90 features were also computed. Normalization was performed based on the three minute baseline resting period, using:

$$ratio\xi_{norm,i} = 10 \log \left( \xi_i^{epoch} / \xi_i^{baseline} \right), \quad (1)$$

with  $\xi_{norm,i}$  being the ‘i’-th feature. As such, a total of 180 ECG features were extracted.

- EMG** The electromyographic signal provided 53 features, that can be divided into three groups containing spectral, statistical evaluation and the sensitivity to change (first and second derivative) features. As with the ECG features, normalization was performed using baseline data, thus totaling 106 EMG features.
- RESP** From the Respiration signal, 74 features were obtained. These can be grouped into 3 different classes: rate of change, statistical and spectral analysis. Seventy-four additional features derived from the baseline normalization technique yielded a total of 148 features.
- EDA** Lastly, EDA provided 27 features (statistical, spectral and rate of change groups) from the band-passed signal. A total of 54 features were extracted including the baseline-normalized versions.

### E. Feature Selection

Due to the large number of features extracted (i.e., 488), and particularly in cases of feature fusion techniques, such large number of features may result in classifier overfitting. As such, the so-called mRMR [26] feature selection algorithm is used. mRMR is a mutual information based algorithm that finds near-optimal features using forward selection with the chosen features maximizing the combined max-min criteria. Two criteria are applied as one: the maximum-relevance criterion (maximization of the average of mutual information between features and labels) and the minimum-redundancy criterion (minimization of the average mutual information between two chosen features). In the present work, 20% of the available data was set aside for feature ranking. The remaining 80% was used for classifier training and testing in a cross-validation scheme. Such partitioning corresponds to having 10 samples per class for testing and 40 samples per class for training. More details can be found in Section II-F.

The features were grouped into three separate sets. The first one included the non-normalized features coming from the four physiological signals, the second set consisted in the normalized ones, and the last group was constituted from a fusion of the first two. Feature ranking was conducted for the non-normalized feature set alone, for the normalized feature set alone and the combined feature set on a per-subject basis. Then, the first ten selected features were further ranked based on the number of times they were selected across all participants.

### F. Classification

Support Vector Machine (SVM) classifiers have been used in the present work. Given its widespread use, a description



Fig. 3: Experiment set-up for dataset collection.

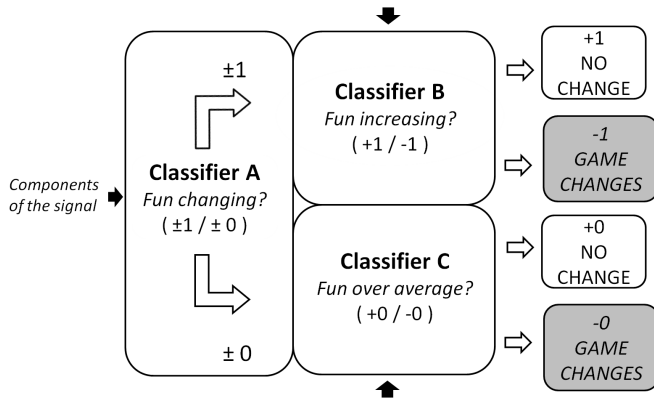


Fig. 4: Classification scheme designed to detect the level of fun and decide if the game needs real-time adjustment.

of the support vector machine approach is not included here and the interested reader is referred to [27] for more details. SVM classifiers are trained on three different (cascaded) binary classification problems, as depicted by Fig. 4, namely i.e. detecting fun changing/non-changing (classifier A), increasing/decreasing (classifier B) and above/below average (classifier C). The first, termed classifier A, discriminates between fun ‘changing’ or ‘non-changing’. Based on this output, the second classifier to be used is decided. If the output of classifier A is fun ‘changing’, classifier B will successively discriminate between fun ‘increasing’ or ‘decreasing’. If the output of classifier A is fun ‘non-changing’, then classifier C will discern between fun ‘above-average’ or ‘below-average’.

In order to discover the best classification modality, two feature- and one decision-level fusion strategies are tested with the remaining 80% of the data. Regarding feature fusion strategies, the first one aims at testing the capacity to predict the output using the entire dataset within a 10-fold cross-validation scheme, whereas in the second case a per-subject classification with a Leave-one-sample-out (LOSO) cross validation scheme has been tested for comparison. In both cases, default SVM parameters have been used throughout our analyses (i.e.,  $\lambda = 1$  and  $\gamma_{RBF} = 0.01$ ); moreover, a Radial Basis Function kernel was used and implemented with the Scikit-learn library in Python [28]. Lastly an optimally weighted decision fusion scheme has been tested [29]. First, the training data of the normalized feature sets for each physiological signal modality have been treated separately for both feature ranking and per-subject classification. Next, based on the performance achieved, a weight has been determined for the four signal modalities. According to the decision fusion technique used, the parameter  $t_i$  is the achieved performance for a particular modality, on the training dataset, such that the sum across all modalities equals unity [29]. The  $t_i$  parameter is calculated as follows:

$$t_i = \frac{A_i}{\sum_{i=1}^N \alpha_i A_i} \quad (2)$$

where,  $A_i$  is the accuracy obtained training the dataset be-

TABLE I: Percentage of participation of each physiological signal (ECG, EMG, EDA and RESP) for the classification schemes (classifiers A, B and C), as well as for the three methods together.

	Classifier A	Classifier B	Classifier C	Total
ECG	39 %	33 %	23 %	32 %
EMG	14 %	21 %	25 %	20 %
RESP	28 %	31 %	36 %	31 %
EDA	19 %	15 %	16 %	17 %

longing to a particular modality,  $N$  is the number of modalities and  $\alpha_i$  are the weights corresponding to each modality ( $\sum_{i=1}^N \alpha_i = 1$ ). Optimally weighted decision fusion relies on optimal weights for each of the four modalities which are obtained calculating the  $\alpha_i$  values that result in the best performance on the training set.

### III. RESULTS AND DISCUSSION

#### A. Feature Ranking

Here, only the feature ranking analysis conducted on the normalized feature set is reported as it resulted in the highest accuracy. Table I shows the percentage of features used to reach the best accuracy from each of the four signals (EDA, ECG, EMG and RESP) for each of the three classifiers separately and for the total. As can be seen, ECG and RESP play a relevant role, representing almost two thirds of the total amount of top-ranked features.

An in-depth analysis on the features ranked by the mRMR algorithm has been conducted in order to understand the most relevant signals and the top-ranked features. Across all physiological signals, two thirds of the contribution comes from ECG and RESP, with a peak of 67% in the case of classifier A. Moderate importance can be attributed to EMG

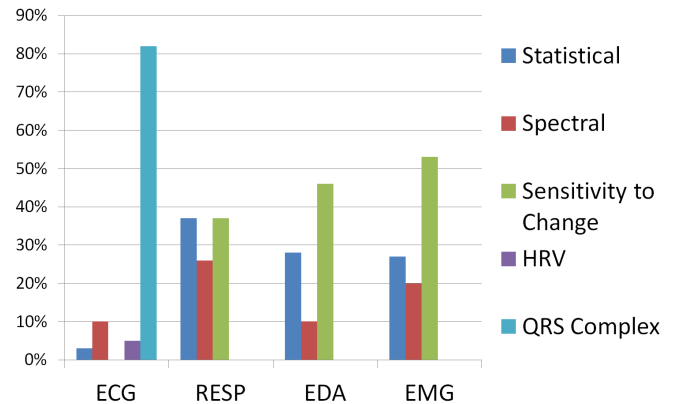


Fig. 5: Different contribution of the groups of features extracted (see Sec. II-D) for each signal: ECG (spectral, statistical, HRV and QRS complex), EMG, EDA and RESP (spectral, statistical and sensitivity to change).

and EDA, contributing on average one third of the top features. One potential reason for EMG playing a secondary role could be due to arm movement artifacts, as well as fatigue, which has been shown to affect EMG characteristics [30]. The low temporal resolution of EDA could also be a factor limiting the contribution of this signal. On the contrary, ECG and RESP have been previously connected to emotional expressivity, and their contribution is relevant to understanding inter-individual differences regarding players' physiological responses [31].

Fig. 5 shows the type of features more frequently selected by the feature ranking algorithm, for each of the four signal modalities. As previously explained, three groups of features can be differentiated for RESP, EDA and EMG (statistical, spectral and sensitivity to change), whereas for ECG four groups are identified (statistical, spectral, QRS analysis and HRV test). As can be seen, the analysis conducted on the first and second derivative of the signal is predominant for EMG and EDA. Whereas for the ECG, the study of QRS complex prevails over the other three groups. RESP, in turn, had the same number of features coming from the rate of change and statistical measures. This situation could be due to its low temporal resolution and its relationship with the ECG signal [32].

In order to give a better sense of the most used features, for each signal and for each classifier, the top-10 components have been analyzed. Within the ECG class, one third of the investigated top-features represent information about S and T waves, whereas another 20% is attributed to information measured from the segment between the P and Q waves. Moreover, two out of three classifiers had as their top-10 features the root-mean-square of the band-passed signal. Regarding EMG, 40% of the first 10 selected features were chosen from the analysis of the first derivative of the signal. Furthermore, by analyzing the repetition of the components, the presence of the power spectrum between 50 and 100 Hz has been ranked as top-feature for all the three classifiers. For RESP and EDA the most significant features (respectively 40% and 37%) were spectral related features. In particular, the power spectrum in the range of 0-0.7 Hz for RESP and 0-0.4 Hz for EDA.

### B. Classification

Table II reports the highest accuracies and F1 scores [33] achieved with the individual and fused feature sets, for both all-dataset (10-fold validation) and per-subject (LOSO) classification schemes, as well as the number of features required to achieve such results. Values followed by an asterisk indicate significantly higher than chance according to an independent one-sample t-test ( $p < 0.01$ ). As can be seen, the normalized feature sets achieves the best accuracy and F1 score results for the three classifiers. The best performance is obtained with a per-subject LOSO classification and in particular with the classifier C, reaching 70% discrimination accuracy.

Three main concepts can be inferred from feature-fusion classification results. First, fun is a dimension that can be detected by physiological signals, and in particular it is easier to measure the level of fun than its trend. When we compare

the performance between the three classifiers, the best performance in accuracy is obtained when classifying the level of fun (fun 'above-average' and 'below-average' for classifier C), whereas performance is lower when using discriminating tendencies (fun 'increasing/decreasing' in classifiers A and B). Furthermore, a per-subject classification surpassed the one conducted using the full dataset. This effect can be related to the fun conceived as an individual factor [12]. A support to this idea also comes from the evaluation of the performances of the three feature sets. In fact, the normalized set outperforms or balances the level reached by the fusion set, thus showing the importance of per-subject normalization for automated fun assessment. Additionally, when comparing the number of features in a per-subject classification, two out of three classifiers (A and C) reached the best results with the normalized feature sets while using the minimum number of features. Classifier B with the fused feature set, on the other hand, required only one third of the components needed by the normalized set. For what concerns the all-dataset 10-fold cross validation classification, the best accuracy result was always reached with fewer features compared to the other classification schemes, but at the same time resulting in lower classification performance.

In turn, Table III shows the performances achieved with the decision level fusion scheme for the three classifiers. While decision level fusion of classifiers A and C did not lead to gains over simple feature fusion, decision level fusion of classifier B did improve the accuracy with a gain over the feature level fusion of 6 % for the normalized feature set and 4 % for the non-normalized set.

While decision level fusion with classifiers trained on these four separate modalities (ECG/EMG/EDA/RESP) resulted in an improvement only for classifier B, decision level fusion did result in further improvements, particularly when discriminating the increasing/decreasing dimension, thus suggesting the complementarity of the four physiological responses. Higher contribution rate is attributed to ECG (classifiers B and C). RESP holds more decision fusion weight in classifier A, whereas EDA contributed with the third highest weight. Decision level fusion was previously shown to be a useful tool for affective state recognition [33].

## IV. CONCLUSION AND FUTURE WORK

In this work, a triple-classifier system was tested for automated fun-level state recognition during video game sessions. Experimental results showed relevant performances in terms of accuracy. Feature level fusion has been proved to work better when detecting the level of fun, whereas decision level fusion when discriminating trends (70% and 57% respectively). Moreover, the importance of attributing an individual component to the players' fun-factor was demonstrated and essential physiological features are detected. Such findings suggest the importance of a robust adaptive video game based on personal characteristics of player's physiological signals, and capable of maintaining a high level of fun.

TABLE II: Performance comparison of SVM classifiers for different feature sets and feature-level fusion along with the required number of features needed to achieve such results. Asterisks indicate whether the accuracy or the F1-score distribution over subjects is significantly higher than chance according to an independent one-sample t-test ( $p < 0.01$ ). ‘Per-subj’ corresponds to per-subject LOSO results, whereas ‘All-dataset’ to 10-fold cross-validation on the entire dataset.

Classifier A									
	Non-Normalized Features			Normalized Features			Feature Fusion		
	Accuracy	F1 Score	No. Features	Accuracy	F1 Score	No. Features	Accuracy	F1 Score	No. Features
Per-Subj	0.59*	0.59*	87	0.60*	0.60*	69	0.59*	0.60*	70
All-dataset	0.55	0.55	35	0.54	0.54	35	0.55	0.55	32

Classifier B									
	Non-Normalized Features			Normalized Features			Feature Fusion		
	Accuracy	F1 Score	No. Features	Accuracy	F1 Score	No. Features	Accuracy	F1 Score	No. Features
Per-Subj	0.54*	0.53*	53	0.54*	0.53*	91	0.54*	0.53*	36
All-dataset	0.51	0.50	18	0.50	0.53	22	0.50	0.51	4

Classifier C									
	Non-Normalized Features			Normalized Features			Feature Fusion		
	Accuracy	F1 Score	No. Features	Accuracy	F1 Score	No. Features	Accuracy	F1 Score	No. Features
Per-Subj	0.69*	0.68*	54	0.70*	0.70*	69	0.70*	0.69*	101
All-dataset	0.55	0.52	30	0.55	0.52	40	0.54	0.52	20

TABLE III: Performance comparison of SVM classifiers for different decision-level fusion schemes. Asterisks indicate whether the accuracy or the F1-score distribution over subjects is significantly higher than chance according to an independent one-sample t-test ( $p < 0.01$ ).

	Non-Norm. Feats		Norm. Feats		Feature Fusion	
	Acc.	F1 Score	Acc.	F1 Score	Acc.	F1 Score
Class. A	0.57*	0.57 *	0.56 *	0.56*	0.57 *	0.57*
Class. B	0.56*	0.55 *	0.57*	0.57*	0.56*	0.55*
Class. C	0.68*	0.69*	0.70*	0.70*	0.69*	0.69*

Game design can be significantly improved when conducting analysis of cognitive and affective neuro-ergonomics. To improve the performance of the proposed classifier system and automated game affective tasks, supplementary steps could be undertaken. First, the binary classification tasks performed here could be replaced by a regression task where the actual continuous value of the fun could be predicted. A second improvement would be to introduce a personalized calibration before starting the game, in order to train the classifiers based on personal traits of the subject. Third, classification performance could be improved by selecting optimal classification models by tuning hyperparameters (based on the individual signature) and explore different fusion strategies. Despite being innovative, the continuous rating of fun performed after the videogame session does have its drawbacks: it might not represent the actual fun that was perceived during the play and it relies on participants memory of their enjoyment

which could have been forgotten and/or biased. Furthermore, participants did not have the possibility to rate fun on more than one dimension. Still, the outcomes of the present work can be applied to the development of a real-time adaptable intelligent game with application in console as well as online gaming. As a matter of fact, the present work is part of the FUNii (interactive intelligent) project [31] that aims at the development of an intelligent and interactive system capable of predicting the player’s level of fun and adjusting the game to maximize that value.

#### ACKNOWLEDGMENT

This research was supported by a collaborative research and development grant to Philip L. Jackson and Sébastien Tremblay from the National Sciences and Engineering Research Council of Canada (NSERC) in collaboration with Ubisoft Québec. The authors would like to thank Jérémy Bergeron-Boucher, Marc-André Bouchard, Roxanne Poulin, Eric Arsenault, Sophie Regueiro for assistance in data collection and organization, as well as Ludovic Lefebvre for the support provided at the Ubisoft Québec studio.

#### REFERENCES

- [1] D. E. S. A. Washington, Ed., *2015 sales, demographics and usage data: Essential facts about the computer and video game industry*, Entertainment Software Association, 2015.
- [2] Newzoo, Ed., *Global games market*, Games Market Research, 2015.
- [3] R. W. Picard, *Affective computing*. MIT press, 2000.
- [4] F. Portnoy, R. Aseron, M. Harrington, K. Kremer, T. Nichols, and V. Zammito, “Facing the human factors challenges in game design a discussion panel,” in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 55, no. 1. SAGE Publications, 2011, pp. 520–524.
- [5] N. Lazzaro, “Why we play games: Four keys to more emotion without story,” 2004.

- [6] K. Poels, Y. De Kort, and W. Ijsselstein, "It is always a lot of fun!: exploring dimensions of digital game experience using focus group methodology," in *Proceedings of the 2007 conference on Future Play*. ACM, 2007, pp. 83–89.
- [7] C. Bateman and L. E. Nacke, "The neurobiology of play," in *Proceedings of the International Academic Conference on the Future of Game Design and Technology*. ACM, 2010, pp. 1–8.
- [8] P. Sweetser and P. Wyeth, "Gameflow: a model for evaluating player enjoyment in games," *Computers in Entertainment (CIE)*, vol. 3, no. 3, pp. 3–3, 2005.
- [9] M. Csikszentmihalyi, "Flow: The psychology of optimal experience," 1990.
- [10] G. Calleja, "Revising immersion: A conceptual model for the analysis of digital game involvement," *Situated Play*, pp. 24–28, 2007.
- [11] P. M. Desmet. (2003) Measuring emotions: development and application of an instrument to measure emotional responses to products.
- [12] R. J. Pagulayan, K. Keeker, D. Wixon, R. L. Romero, and T. Fuller, "User-centered design in games," 2002.
- [13] L. E. Nacke, "Games user research and physiological game evaluation," in *Game User Experience Evaluation*. Springer, 2015, pp. 63–86.
- [14] R. L. Mandryk, K. M. Inkpen, and T. W. Calvert, "Using psychophysiological techniques to measure user experience with entertainment technologies," *Behaviour & information technology*, vol. 25, no. 2, pp. 141–158, 2006.
- [15] R. L. Mandryk and M. S. Atkins, "A fuzzy physiological approach for continuously modeling emotion during interaction with play technologies," *International journal of human-computer studies*, vol. 65, no. 4, pp. 329–347, 2007.
- [16] C. Pedersen, J. Togelius, and G. N. Yannakakis, "Modeling player experience in super mario bros," in *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*. IEEE, 2009, pp. 132–139.
- [17] N. Shaker, G. N. Yannakakis, and J. Togelius, "Towards automatic personalized content generation for platform games," in *AIIDE*, 2010.
- [18] S. Tognetti, M. Garbarino, A. Bonarini, and M. Matteucci, "Modeling enjoyment preference from physiological responses in a car racing game," in *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*. IEEE, 2010, pp. 321–328.
- [19] G. N. Yannakakis and J. Hallam, "Entertainment modeling through physiology in physical play," *International Journal of Human-Computer Studies*, vol. 66, no. 10, pp. 741–755, 2008.
- [20] J. M. Kivikangas, G. Chanele, B. Cowley, I. Ekman, M. Salminen, S. Järvelä, and N. Ravaja, "A review of the use of psychophysiological methods in game research," *Journal of Gaming & Virtual Worlds*, vol. 3, no. 3, pp. 181–199, 2011.
- [21] C. Liu, P. Rani, and N. Sarkar, "An empirical study of machine learning techniques for affect recognition in human-robot interaction," in *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*. IEEE, 2005, pp. 2662–2667.
- [22] P. Rani, N. Sarkar, and C. Liu, "Maintaining optimal challenge in computer games through real-time physiological feedback," in *Proceedings of the 11th international conference on human computer interaction*, vol. 58, 2005.
- [23] G. Chanele, C. Rebetez, M. Bétrancourt, and T. Pun, "Boredom, engagement and anxiety as indicators for adaptation to difficulty in games," in *Proceedings of the 12th international conference on Entertainment and media in the ubiquitous era*. ACM, 2008, pp. 13–17.
- [24] H. P. Martinez, G. N. Yannakakis, and J. Hallam, "Dont classify ratings of affect; rank them!" *IEEE Transactions on Affective Computing*, vol. 5, no. 3, pp. 314–326, 2014.
- [25] Y. Xu, G. Liu, M. Hao, W. Wen, and X. Huang, "Analysis of affective ECG signals toward emotion recognition," *Journal of Electronics (China)*, vol. 27, no. 1, pp. 8–14, 2010.
- [26] H. Peng, F. Long, and C. Ding, "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 8, pp. 1226–1238, 2005.
- [27] B. Schölkopf and A. J. Smola, *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT press, 2002.
- [28] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [29] S. Koelstra and I. Patras, "Fusion of facial expressions and EEG for implicit affective tagging," *Image and Vision Computing*, vol. 31, no. 2, pp. 164–174, 2013.
- [30] N. Vuillerme, V. Nougier, and N. Teasdale, "Effects of lower limbs muscular fatigue on anticipatory postural adjustments during arm motions in humans," *Journal of sports medicine and physical fitness*, vol. 42, no. 3, p. 289, 2002.
- [31] C. Chamberland, M. Grégoire, P.-E. Michon, J.-C. Gagnon, P. L. Jackson, and S. Tremblay, "A cognitive and affective neuroergonomics approach to game design," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 59, no. 1. SAGE Publications, 2015, pp. 1075–1079.
- [32] J. Boyle, N. Bidargaddi, A. Sarela, and M. Karunanithi, "Automatic detection of respiration rate from ambulatory single-lead ECG," *Information Technology in Biomedicine, IEEE Transactions on*, vol. 13, no. 6, pp. 890–896, 2009.
- [33] S. Koelstra *et al.*, "DEAP: A database for emotion analysis; using physiological signals," *IEEE Transactions on Affective Computing*, vol. 3, no. 1, pp. 18–31, 2012.

# Recovering Visibility and Dodging Obstacles in Pursuit-Evasion Games

Ahmed Abdelkader  
Department of Computer Science  
University of Maryland  
College Park, Maryland 20742  
Email: akader@cs.umd.edu

**Abstract**—Pursuit-evasion games encompass a wide range of planning problems with a variety of constraints on the motion of agents. We study the visibility-based variant where a pursuer is required to keep an evader in sight, while the evader is assumed to attempt to hide as soon as possible. This is particularly relevant in the context of video games where non-player characters of varying skill levels frequently chase after and attack the player.

In this paper, we show that a simple dual formulation of the problem can be integrated into the traditional model to derive optimal strategies that tolerate interruptions in visibility resulting from motion among obstacles. Furthermore, using the enhanced model we propose a competitive procedure to maintain the optimal strategies in a dynamic environment where obstacles can change both shape and location. We prove the correctness of our algorithms and present results for different maps.

## I. INTRODUCTION

Pursuit-evasion games have received considerable attention in both the AI planning and robotics communities which resulted in a wealth of results. In the visibility-based variant, the problem of deciding whether the evader possesses an escape strategy is known to be NP-hard [10]. Analytical solutions to the problem for limited obstacle geometries have been derived by appealing to differential game theory [2]. Other variants of the problem has been studied under complete information [7], imperfect information [13] and partially-observable spaces [12]. More realistic models of agents with a limited range of vision have also been considered [4].

The solution method we are interested in is based on backward induction. Starting from terminal states, where the outcome of the game is known, the outcome for earlier states can be determined recursively by considering the actions available at each state. It may be viewed as a discrete analogue to integrating a system of differential equations from a set of initial conditions. This intuitive recursive formulation easily lends itself to dynamic programming which provides an efficient solution to a variety of problems.

In the same spirit, backward induction can also be used for optimization problems like cost-to-go Bellman formulations to path planning problems, e.g., [17]. To bound the number of states that need to be explored, sampling approaches are typically used as in Rapidly-exploring Random Trees (RRTs) [9] or adaptively refined meshes [16]. More traditionally, uniform grids continue to be a standard tool to model domains for planning problems. Recent works on any-angle path planning,

e.g., [3], have made it possible to overcome the unrealistic trajectories generated by such grid techniques.

Alternatively, precomputation has been considered to stay close to optimality at the cost of higher storage. Scalable precomputed search trees (SPST) is one recent example where RRT type trees are computed to provide uniform coverage of a domain [8]. Similar approaches utilizing roadmaps have been reported for the problem of pursuit-evasion [14]. Roadmap based techniques have been applied to the visibility-based variant as well, e.g., [6].

In this paper, we develop an enhanced model for visibility-based pursuit-evasion that allows us to compute optimal strategies for two interesting scenarios particularly relevant to video games. In the first scenario, a *tolerance parameter* is specified to allow interruptions in visibility of bounded duration. In the second scenario, the map is allowed to change, i.e., obstacles can change both shape and location. We reuse the algorithmic framework we presented earlier for computing a strategy matrix by backward induction [1]. *To the best of our knowledge, these are the first algorithms to compute optimal pursuit-evasion strategies in these scenarios and only heuristic-based or suboptimal limited-depth approaches were known.* This enables the design of more intelligent computer players and also helps with level design to assess the difficulty of different layouts and choose entry points for respawning.

The rest of the paper is organized as follows. In Section 2, we define the visibility-based pursuit-evasion game and recall the solution method we will be using in this study. Section III introduces the dual formulation, which is key to the remainder of the paper, and studies the first scenario where we relax the hard visibility constraints with a tolerance parameter. Then in Sections IV and V, we continue to demonstrate how similar techniques can be used to extend this solution method to dynamic environments where obstacles can change both shape and location. Finally, we conclude in Section VI.

## II. THE VISIBILITY-BASED PURSUIT-EVASION GAME

The pursuit-evasion game studied here can be defined as follows. We are given two agents: a pursuer ( $p$ ) and an evader ( $e$ ) at known initial positions in an environment with obstacles that block both motion and visibility. The pursuer is required to keep the evader in sight, while the evader is assumed to attempt to break the pursuer's line of sight in the shortest amount of



time possible. Both players have complete information about the other's location and move at bounded speeds. Hence, the first natural question is to decide for a given environment, initial positions, and maximum speeds of both players, whether the evader has an escape strategy.

The solution method we presented earlier [1] uses a grid map discretization of the environment and assumes both players take turns to move between cells of this grid, which bears similarity to cop-robber games on graphs [5]. Per the description of the game, the game state can be completely determined by the locations of both players, denoted by the ordered pair  $(p, e)$ , and which of the two players moves next.

This method is summarized in Algorithm 1. Given a grid map of dimensions  $w \times h$ , computation is performed on a  $(w \times h) \times (w \times h)$  boolean matrix that stores for each pair of locations whether or not the evader has an escape strategy. Starting at terminal states, which are pairs  $(p, e)$  where  $e$  is not visible to  $p$ , the game can be decided for these states (Line 3). This is implemented by a simple procedure,  $M.vis(p, e)$ , that checks whether the line connecting  $e$  to  $p$  passes through any of the obstacles. To decide the game for earlier states, standard backward induction is performed (Line 4) as described in more details in procedure InductionLoop. Note that we set  $S[p, e] = 1$  iff the pursuer starting at position  $p$  cannot keep the evader starting at position  $e$  in sight, with the evader moving first.

---

**Algorithm 1:** Decides the game for a given map.

---

**Input :** A  $(w \times h)$  grid map of the environment  $M$ .  
**Output:** The strategy matrix  $S$ .  
1 **begin**  
2   InitVisibility( $M, S$ );  
3   InductionLoop( $S$ );  
4   **return**  $S$

---

The function InductionLoop repeatedly evaluates the escape condition for each game state, which may only be available after adjacent states have been determined. Once an escape strategy is found, there is no need to process the state again. The escape conditions can be expressed as the recurrence relation of the form:

$$S[p, e] = \bigvee_{e' \in \mathcal{N}(e)} \bigwedge_{p' \in \mathcal{N}(p)} S[p', e']. \quad (1)$$

We do not attempt to optimize the implementations here to keep the presentation as simple as possible. More elaborate optimizations along with their theoretical analysis were discussed in [1]. We use  $\mathcal{N}$  to denote the neighborhood of locations the player can reach in a single turn, which implicitly depends on its speed. Letting  $N$  be the size of the map, i.e.,  $w \times h$ , and  $\kappa$  be the largest size of a neighborhood  $\mathcal{N}$ , we recall the following result established in [1]:

**Theorem 1.** (Visibility Induction [1]) *Algorithm 1 decides the discretized game for a general environment in  $\mathcal{O}(\kappa^2 N^3)$ .*

---

**Function** InitVisibility( $M, S$ )

---

**Input :** A grid map  $M$  and a strategy matrix  $S$ .  
**Output:** The initialized strategy matrix  $S$ .

```

1 begin
2    $S \leftarrow 0$ ;
3   foreach  $p \in w \times h$  do
4     foreach  $e \in w \times h$  do
5       if  $\neg M.vis(p, e)$  then
6          $S[p, e] \leftarrow 1$ ;
7   return  $S$ 

```

---



---

**Procedure** InductionLoop( $S$ )

---

**Input :** A strategy matrix  $S$ .

**Data:** A secondary  $(w \times h) \times (w \times h)$  binary matrix  $S'$ .

```

1 begin
2    $S' \leftarrow 0$ ;
3    $iter \leftarrow 0$ ;
4   while  $S' \neq S$  do
5      $S' \leftarrow S$ ;
6     foreach  $p \in w \times h$  do
7       foreach  $e \in w \times h$  do
8         foreach  $e' \in \mathcal{N}(e)$  do
9            $isExit \leftarrow True$ ;
10          foreach  $p' \in \mathcal{N}(p)$  do
11            if  $S'[p', e'] = 0$  then
12               $isExit \leftarrow False$ ;
13          if  $isExit = True$  then
14             $S[p, e] \leftarrow 1$ ;
15            break;
16      $iter \leftarrow iter + 1$ ;
17   return  $S$ ;

```

---

In the next section, we formulate the dual game and describe the dual induction loop which is key to the algorithms in Sections III and IV.

### III. RECOVERING LOST VISIBILITY

In order to tolerate visibility interruptions, we do not terminate the game and declare that the pursuer has lost as soon as line of sight visibility is broken. Instead, we introduce a parameter  $d$  that controls how long we allow the evader to remain out of the evader's sight in one streak. The pursuer would then seek strategies that can recover visibility to the evader if that is possible to achieve within  $d - 1$  steps, and the evader only wins if it is able to hide for at least  $d$  consecutive steps. The optimal strategy for the evader is still to find the fastest way to *win*. As such, the evader does not favor intermediate visibility interruptions if they do not lead to a sooner victory.

We introduce the *dual game* to model the situation after visibility is lost. In this phase, the evader attempts to remain out of the pursuer's sight as long as possible, while the pursuer attempts to recover visibility to the evader as soon as possible. Note that in the original game the evader's objective was

to minimize the visibility time, while in this phase it is to maximize the occlusion time. Similarly, in the original game, the pursuer's objective was to maximize the visibility time, while in this phase it is to minimize the occlusion time. In a sense, the agents exchange their roles but the dynamics stay the same.

For this reason, we refer to this situation as the dual game. We obtain a corresponding recurrence relation for recovering visibility as the logical negation of the escape conditions in Equation 1:

$$S[p, e] = \bigwedge_{e' \in \mathcal{N}(e)} \bigvee_{p' \in \mathcal{N}(p)} \neg S[p', e']. \quad (2)$$

Observe that the dual game is only defined for pairs of player positions that are not mutually visible. These are exactly the pairs that defined the terminal states for the original game. It is clear that in order to allow the game to proceed as long as visibility can be recovered within  $d$  steps, we need to exclude those pairs from the terminal states. With that, all that is needed is to run a *dual induction* on the non-visible pairs to get the *relaxed* terminal states. Then, running the original induction backwards from the restricted set of terminal states yields the desired strategies.

These steps are summarized in Algorithm 2. After initializing the matrix by marking all pairs that are not mutually visible, the procedure DualInductionLoop is invoked. Each iteration in this procedure bears strong similarity to the original induction loop. However, the result is that for certain pairs initialized as terminal, with the evader winning and the pursuer losing, this decision is simply undone (Line 16).

---

**Algorithm 2:** Tolerating interruptions in visibility.

---

**Input :** A strategy matrix  $S$ , grid map  $M$ , tolerance  $d$ .  
**1 begin**  
 2   InitVisibility( $M$ ,  $S$ );  
 3   DualInductionLoop( $S$ ,  $M$ ,  $d$ );  
 4   InductionLoop( $S$ );  
 5   **return**  $S$ ;

---

Using the updated terminal states, Algorithm 2 computes the pursuit-evasion strategies for all pairs of initial positions to maintain visibility as long as possible, while tolerating interruptions in visibility within  $d$  steps. As we are essentially reusing the induction loop studied in [1], we get the same bound on the running time. In addition, the same optimizations can be applied to speed up the computation.

The correctness of the DualInductionLoop procedure is established in the next lemma.

**Lemma 2.** *The DualInductionLoop correctly computes strategies to recover visibility in less than  $d$  steps, if any.*

*Proof.* For the base case, when  $iter = 0$ ,  $S[p, e] = 0$  iff  $(p, e)$  are mutually visible, as initialized by InitVisibility. Then, when  $iter = i$ ,  $S[p, e]$  is assigned 0 (Line 16) iff  $e$  does not have a neighbor  $e'$  such that no neighbor  $p'$  of  $p$  satisfies

---

**Procedure** DualInductionLoop( $S$ ,  $M$ ,  $d$ )

---

**Input :** A strategy matrix  $S$ , grid map  $M$ , tolerance  $d$ .  
**Data:** A secondary  $(w \times h) \times (w \times h)$  binary matrix  $S'$ .  
**1 begin**  
 2    $iter \leftarrow 0$ ;  
 3   **while**  $iter < d$  **and**  $S' \neq S$  **do**  
 4      $S' \leftarrow S$ ;  
 5     **foreach**  $p \in w \times h$  **do**  
 6       **foreach**  $e \in w \times h$  **do**  
 7           $hasExit \leftarrow \text{False}$ ;  
 8          **foreach**  $e' \in \mathcal{N}(e)$  **do**  
 9             $isExit \leftarrow \text{True}$ ;  
 10            **foreach**  $p' \in \mathcal{N}(p)$  **do**  
 11              **if**  $M.vis(p', e')$  **or**  $S'[p', e'] = 0$   
 12                **then**  
 13                   $isExit \leftarrow \text{False}$ ;  
 14                **if**  $isExit = \text{True}$  **then**  
 15                   $hasExit \leftarrow \text{True}$ ;  
 16                **if**  $hasExit = \text{False}$  **then**  
 17                   $S[p, e] \leftarrow 0$ ;  
 18             $iter \leftarrow iter + 1$ ;  
 19     **return**  $S$ ;

---

$M.vis(p', e')$  or  $S'[p', e'] = 0$ . If  $M.vis(p', e')$  is true, then  $p'$  has direct visibility to  $e'$ . Otherwise, if  $S'[p', e'] = 0$ , then by the induction hypothesis,  $p'$  has a strategy that guarantees visibility to  $e'$  is recovered within  $i - 1$  steps.  $\square$

Following with an invocation of the original induction loop in procedure InductionLoop, the next theorem proves the correctness of the whole scheme, which relaxes the result in Theorem 1 to scenarios with less strict visibility requirements.

**Theorem 3.** *Algorithm 2 decides the discretized game for a general environment in  $\mathcal{O}(\kappa^2 N^3)$ , tolerating arbitrary interruptions in visibility of  $d = \mathcal{O}(N)$  steps.*

*Proof.* By invoking the DualInductionLoop (Line 2),  $S[p, e] = 1$  iff  $e$  has a strategy to hide out of sight for at least  $d$  steps as established in Lemma 2. Then, the InductionLoop is invoked (Line 3) starting at  $iter = 0$  with  $S$  as returned from DualInductionLoop. For  $iter = i$  in the InductionLoop,  $S[p, e]$  is assigned 1 (Line 14) iff  $e$  has a neighbor  $e'$  such that for all neighbors  $p'$  of  $p$  we have that  $S'[p', e'] = 1$ . By the induction hypothesis, it follows that for any such  $p'$ , it must be the case that by  $iter = i - 1$ ,  $e'$  has found an escape strategy to stay out of  $p'$ 's sight for at least  $d$  steps.

The bound on the running time follows by Theorem 1. Observe that for large values of  $d$ , the overhead of running the DualInductionLoop cannot be greater than the worst case for running InductionLoop itself. Hence, the total running time has the same bound.  $\square$

Figure 1 shows a pursuer with all evader locations it cannot keep in sight colored in gray. We compare the traditional scenario of zero tolerance against allowing broken visibility for 5 turns. Beyond deciding which initial conditions enable each player to win, the computed strategy matrix can be used for trajectory planning as discussed in [1]. Figure 2 shows a basic example of successful tracking although visibility was initially broken.

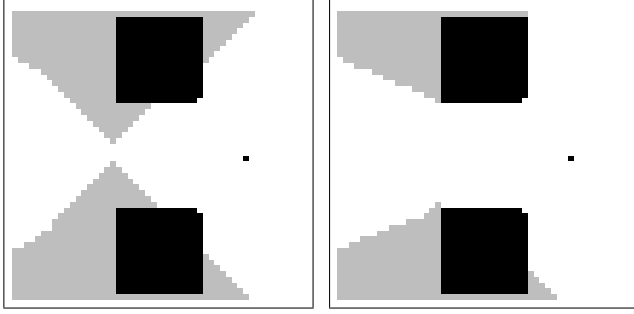


Fig. 1. Pursuer view in the case with  $d = 0$  (left) vs.  $d = 5$  (right).

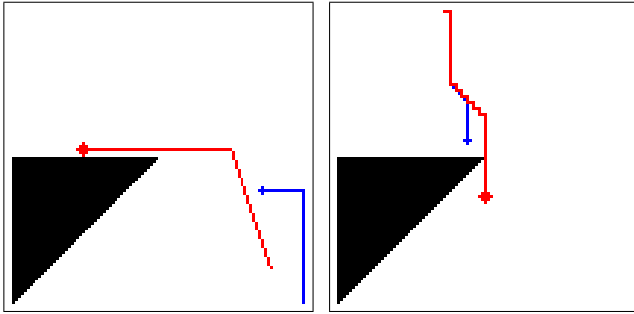


Fig. 2. A pursuer recovering visibility around an infinite corner.

#### IV. MOVING OBSTACLES BY ADD/REMOVE

The interplay between the classical game and its dual as seen in Algorithm 2 yields new insights into computed strategies as encoded in the matrix  $S$ . In the original game, mutual visibility is established and the evader attempts to hide altering an entry in the matrix from 0 to 1. In the dual game, visibility is broken and the pursuer attempts to recover it altering an entry in the matrix from 1 to 0.

Using this enhanced understanding, we study the visibility-based pursuit-evasion game in a dynamic environment where obstacles can change both shape and location. Naturally, in a scenario like that, initially established visibility can get broken by the changes in the environment, rather than the actions of the agents. It follows that the agents need to update their strategies to match the current environment. To keep the presentation simple, we do not tolerate interruptions in visibility in this section. However, the same method from Section III can be applied to relax visibility tests.

In this section, we present a procedure to maintain the optimality of precomputed strategies that can offer considerable savings compared to recomputing a strategy matrix

from scratch. We use a slightly modified version of the dual induction as listed in the ConservativeDualInductionLoop procedure. This ensures that dual updates do not enable pursuers to chase after evaders they do not see directly.

---

#### Procedure ConservativeDualInductionLoop( $S, M$ )

---

**Input** : A strategy matrix  $S$ , grid map  $M$ .

**Data**: A secondary  $(w \times h) \times (w \times h)$  binary matrix  $S'$ .

---

```

1 begin
2   iter ← 0;
3   while  $S' \neq S$  do
4      $S' \leftarrow S$ ;
5     foreach  $p \in w \times h$  do
6       foreach  $e \in w \times h$  do
7         if  $\neg M.vis(p, e)$  then
8           continue;
9         hasExit ← False;
10        foreach  $e' \in \mathcal{N}(e)$  do
11          isExit ← True;
12          foreach  $p' \in \mathcal{N}(p)$  do
13            if  $S'[p', e'] = 0$  then
14              isExit ← False;
15          if isExit = True then
16            hasExit ← True;
17        if hasExit = False then
18           $S[p, e] \leftarrow 0$ ;
19      iter ← iter + 1;
20  return  $S$ ;

```

---

We use a simple `diff` model to capture the motion of obstacles. We keep track of all grid cells that witness a change in occupancy. It is clear that any change in the environment resulting from a change in the shape or location of obstacles can be expressed as introducing new obstacles at a subset of grid cells and removing existing obstacles from another subset.

To remove obstacles, we first need to establish line-of-sight visibility only between those pairs of positions that were blocked by the removed obstacles. Eventually, some of these pairs may terminate with the evader finding an escape strategy. This means we need to run the original induction loop to find such strategies, if any. The updated strategies propagate to other pairs that may use the newly found routes to improve their outcomes. Adding obstacles is slightly trickier as the added obstacles block both visibility and mobility. For example, an added obstacle may not necessarily help an evader if it does not provide a shorter escape trajectory and instead requires that the evader move around it to reach a more secure exit while a faster pursuer is getting closer which makes it harder for the evader to win.

Both adding and removing obstacles, can be performed in one shot as shown in Algorithm 3. The algorithm simply updates line-of-sight visibility to the limited set of player positions dictated by the updates. Once these updates are established, new strategies are computed and propagated by consecutive invocation of the induction procedures.

**Algorithm 3:** Updates strategies by a diff of the grid map.

---

**Input :** A strategy matrix  $S$ , grid map  $M$ , map diff  $(M^+, M^-)$ .

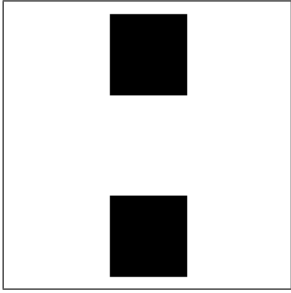
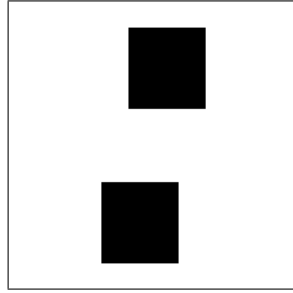
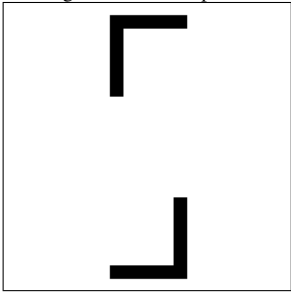
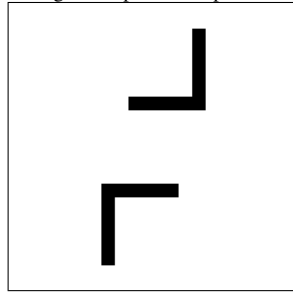
```

1 begin
2    $M \leftarrow M + M^+ - M^-$ ;
3   foreach  $p \in w \times h$  do
4     foreach  $e \in w \times h$  do
5       if  $M.vis(p, e)$  and  $\neg M^-.vis(p, e)$  then
6          $S[p, e] \leftarrow 0$ ;
7       else if  $\neg M^+.vis(p, e)$  then
8          $S[p, e] \leftarrow 1$ ;
9   InductionLoop( $S$ );
10  ConservativeDualInductionLoop( $S, M$ );
11  return  $S$ ;
```

---

Note that some of the decisions applied by the first invocation will need to be corrected by the second one. In fact, the order of invocation does not matter in the correctness of the result. Depending on the required updates, it can be more efficient to start with one type of induction or the other. The order shown here proved to be faster in our experiments moving obstacles by small offsets. For larger shifts, it is more efficient to compute a new matrix from scratch.

Figure 3 shows an initial map with two square obstacles. In Figure 4, the two obstacles have moved diagonally in two opposite directions. Figures 5 and 6 show the difference in occupancy between the initial and final maps. Locations that are no longer occupied by obstacles are denoted by  $M^-$  and those that receive new obstacles are denoted by  $M^+$ .

Fig. 3. Initial map  $M$ .Fig. 4. Updated map  $M'$ .Fig. 5. Removed diff  $M^-$ .Fig. 6. Added diff  $M^+$ .

The correctness of Algorithm 3 is established in the next theorem.

**Theorem 4.** Algorithm 3 correctly updates the strategy matrix in a discretized game given a diff map of the environment.

*Proof.* We argue that the returned strategy matrix is *correct*, i.e., for any pair  $(p, e)$  in the returned matrix,  $S[p, e] = 1$  iff the evader has an escape strategy.

Keeping in mind that the input strategy matrix  $S$  was correct and that the visibility constraints of the updated map were enforced (Lines 2-8), it follows that after invoking InductionLoop (Line 9) any pair  $(p, e)$  where the evader has an escape strategy will have  $S[p, e] = 1$ . By the assumption that  $e$  has an escape strategy, there will eventually be neighbors  $e' \in \mathcal{N}(e)$  with  $S[p', e'] = 1 \forall p' \in \mathcal{N}(p)$  that satisfy the escape conditions for  $e$ , which InductionLoop detects correctly.

It remains to show that for all pairs where the evader does not have an escape strategy,  $S[p, e] = 0$ . This is achieved by the invocation of ConservativeDualInductionLoop (Line 10). Similar to the preceding argument, by the assumption that  $p$  can keep  $e$  in sight indefinitely, there will eventually be a neighbor  $p' \in \mathcal{N}(p)$  for each  $e' \in \mathcal{N}(e)$  with  $S[p', e'] = 0$  that satisfies the recovery conditions for  $p$ , which ConservativeDualInductionLoop detects correctly. Otherwise, if  $e$  does have an escape strategy, the recovery conditions for  $p$  must fail eventually.  $\square$

## V. CONTINUOUSLY MOVING OBSTACLES

Unlike the case in the previous section where obstacles move unexpectedly, it might be the case that their motion trajectories can be estimated in advance. In that case, line-of-sight visibility between pairs of locations becomes a function of time, and the players need to plan their motions taking this into account.

In our discrete setting, assuming a time horizon of  $T$  steps, we only need access to  $vis(p, e)$  at each step  $t$ . This can be encoded as a sequences of matrices  $\{M_t\}$  with  $t = 1 \dots T$ . This can be computed efficiently for obstacles with nice shapes as in [15]. Working backwards from the last step  $T$ , we can easily identify terminal states either directly by a visibility test. Given these terminal states, we run backward induction on  $t$ . We say that an evader wins at time  $t$  if line-of-sight visibility is broken at  $t$  or if the evader is guaranteed an exit at a later time step. Introducing a step index to capture the dependence on time, the recurrence relation for this case can be written as:

$$S[p, e, t] = \neg v(p, e, t) \vee \bigvee_{e' \in \mathcal{N}(e)} \bigwedge_{p' \in \mathcal{N}(p)} S[p', e', t+1]. \quad (3)$$

Algorithm 4 implements the induction for this case. Next, we establish its correctness.

**Theorem 5.** Algorithm 4 decides the discretized game for a sequence of maps  $\{M_t\}$ , with  $t = 1 \dots T$ , in  $\mathcal{O}(\kappa^2 N^2 T)$ .

*Proof.* For the base case, at  $t = T$ , we have that  $S' = 0$ . It follows that  $S[p, e, T] = 1$  only if  $\exists e' \in \mathcal{N}(e)$  such that  $\forall p' \in \mathcal{N}(p)$  we have  $\neg M_T.vis(p', e')$  and the condition in (Line 11) is never satisfied for  $e'$ .

Then, at iteration  $t = i$ , if  $S[p, e, t]$  is set to 1, it must be the case that all pursuer actions  $p'$  failed the test in (Line 11) for at least one evader action  $e'$ , i.e., either visibility is already broken and  $M_i.vis(p, e)$  is false or  $S[p', e', j + 1] = 1$ , which by the induction hypothesis means that an escape strategy for  $e$  at a later step is available through  $e'$ .

Observing that the algorithm performs exactly  $T$  iterations, the bound on the running time follows.  $\square$

---

**Algorithm 4:** Decides the game for a dynamic map.

---

**Input :** A sequence of maps  $\{M_t\}$ ,  $t = 1 \dots T$ .  
**Data:** A secondary  $(w \times h) \times (w \times h)$  binary matrix  $S'$ .

```

1 begin
2    $S \leftarrow 0$ ;
3    $t \leftarrow T$ ;
4   while  $t > 0$  do
5      $S' \leftarrow S$ ;
6     foreach  $p \in w \times h$  do
7       foreach  $e \in w \times h$  do
8         foreach  $e' \in \mathcal{N}(e)$  do
9            $isExit \leftarrow \text{True}$ ;
10          foreach  $p' \in \mathcal{N}(p)$  do
11            if  $M_t.vis(p, e)$  and  $S'[p', e'] = 0$ 
12              then
13                 $isExit \leftarrow \text{False}$ ;
14            if  $isExit = \text{True}$  then
15               $S[p, e] \leftarrow 1$ ;
16              break;
17           $t \leftarrow t - 1$ ;
18   return  $S$ ;

```

---

Assuming the environment does not change after the time horizon  $T$ , we may wish to let the game proceed on this fixed situation. This can easily be accommodated by replacing (Line 2) in Algorithm 4 with an invocation of Algorithm 1 on  $M_T$ . Looking at the proof for Theorem 5, this would only change the base case in an obvious way.

It is also possible to tolerate limited interruptions in visibility in this case as well. However, this requires the use of counters rather than boolean values in the strategy matrices. By incrementing the counter for each step the evader stays out of the pursuer's sight, we can detect when it completes  $d$  steps or when the counter should be reset. A similar technique was applied in [1] to compute the fastest escape trajectory and the corresponding optimal pursuit trajectory, where the original recurrence relation is written as a min-max over such counters, rather than an or-and of booleans.

## VI. CONCLUSION AND FUTURE WORK

We presented a novel dual formulation to the standard visibility-based pursuit-evasion game that allows an easy way to relax the visibility constraints. To the best of our knowledge, this is the first algorithm to compute optimal pursuit-evasion strategies that accommodate recovering visibility once it is lost. Combined with the original formulation, we derived a

competitive update procedure to maintain the optimality of the computed strategies in dynamic environments where obstacles change both shape and location. We proved the correctness of our algorithm and presented basic experimental results for simple maps to demonstrate the contribution.

To make the discretized model more practical, it would be interesting to consider state space reduction such that only few game states are represented explicitly. For a fixed initial position, the approach in [11] seems promising.

## REFERENCES

- [1] Ahmed Abdelkader and Hazem El-Alfy. Visibility induction for discretized pursuit-evasion games. In *AAAI Conference on Artificial Intelligence*. AAAI Press, 2012.
- [2] Sourabh Bhattacharya and Seth Hutchinson. On the existence of nash equilibrium for a two player pursuit-evasion game with visibility constraints. In *Algorithmic Foundation of Robotics VIII*, pages 251–265. Springer, 2010.
- [3] Kenny Daniel, Alex Nash, Sven Koenig, and Ariel Felner. Theta\*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research*, pages 533–579, 2010.
- [4] Brian P Gerkey, Sebastian Thrun, and Geoff Gordon. Visibility-based pursuit-evasion with limited field of view. *The International Journal of Robotics Research*, 25(4):299–315, 2006.
- [5] Geňa Hahn and Gary MacGillivray. A note on k-cop, 1-robber games on graphs. *Discrete mathematics*, 306(19):2492–2497, 2006.
- [6] Volkan Isler, Dengfeng Sun, and Shankar Sastry. Roadmap based pursuit-evasion and collision avoidance. In *Robotics: Science and Systems*, volume 1, pages 257–264, 2005.
- [7] Kyle Klein and Subhash Suri. Complete information pursuit evasion in polygonal environments. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.
- [8] Manfred Lau and James J. Kuffner. Precomputed search trees: Planning for interactive goal-driven animation. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '06, pages 299–308, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.
- [9] Steven M Lavalle and James J Kuffner Jr. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*. Citeseer, 2000.
- [10] Rafael Murrieta-Cid, Raul Monroy, Seth Hutchinson, and Jean-Paul Laumond. A complexity result for the pursuit-evasion game of maintaining visibility of a moving evader. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2657–2664. IEEE, 2008.
- [11] Valentin Polishchuk, Esther M Arkin, Alon Efrat, Christian Knauer, Joseph SB Mitchell, Guenter Rote, Lena Schlipf, and Topi Talvitie. Shortest path to a segment and quickest visibility queries. *Journal of Computational Geometry*, 7(2):77–100, 2016.
- [12] Eric Raboin, Ugrur Kuter, and Dana Nau. Generating strategies for multi-agent pursuit-evasion games in partially observable euclidean space. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1201–1202. International Foundation for Autonomous Agents and Multiagent Systems, 2012.
- [13] Eric Raboin, Dana Nau, Ugrur Kuter, Satyandra K Gupta, and Petr Svec. Strategy generation in multi-agent imperfect-information pursuit games. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 947–954. International Foundation for Autonomous Agents and Multiagent Systems, 2010.
- [14] Samuel Rodriguez, Jory Denny, Takis Zourntos, and Nancy M Amato. Toward simulating realistic pursuit-evasion using a roadmap-based approach. In *Motion in Games*, pages 82–93. Springer, 2010.
- [15] Y-HR Tsai, L-T Cheng, Stanley Osher, Paul Burchard, and Guillermo Sapiro. Visibility and its dynamics in a pde based implicit framework. *Journal of Computational Physics*, 199(1):260–290, 2004.
- [16] Dmitry S Yershov and Emilio Frazzoli. Asymptotically optimal feedback planning: Fmm meets adaptive mesh refinement. In *Algorithmic Foundations of Robotics XI*, pages 695–710. Springer, 2015.
- [17] Dmitry S Yershov and Steven M LaValle. Simplicial Dijkstra and A\* algorithms: From graphs to continuous spaces. *Advanced Robotics*, 26(17):2065–2085, 2012.

# Believable Self-Learning AI for World of Tennis

Maxim Mozgovoy, Marina Purgina  
The University of Aizu  
Aizu-Wakamatsu, Japan  
{mozgovoy, d8172102}@u-aizu.ac.jp

Iskander Umarov  
Helium9 Games, s.r.o.  
Prague, Czech Republic  
isk.umarov@gmail.com

**Abstract**—We describe a method used to build a practical AI system for a mobile game of tennis. The chosen approach had to support two goals: (1) provide a large number of believable and diverse AI characters, and (2) let the users train AI “ghost” characters able to substitute them. We achieve these goals by learning AI agents from collected behavior data of human-controlled characters. The acquired knowledge is used by a case-based reasoning algorithm to perform human-like decision making. Our experiments show that the resulting agents indeed exhibit a variety of recognizable play styles, resembling the play styles of their human trainers. The resulting AI system demonstrated stable decision making, adequate for use in a real commercial game project.

**Keywords**—game AI; believability; case-based reasoning.

## I. INTRODUCTION

The task of designing a game AI begins with a fundamental question: what is the purpose of AI in the given particular game? What kind of AI will contribute to the overall success of the game? Various aspects of “good game AI” are currently being discussed in the community, and include, in particular, believability, fun, and high skill level [1, 2]. The diversity of these aspects can be explained with the diversity of computer games and game genres. As Kevin Dill summarizes, “*The one thing that is universally true is that games are about creating a particular experience for the player—whatever that experience may be. The purpose of Game AI... is to support that experience*” [3].

In the present work, we will describe the AI system for the upcoming mobile free-to-play game *World of Tennis* [4]. Since free-to-play games are typically designed “*for a (very) long duration of play*” [5] to increase in-app spending (and *World of Tennis* is no exception), one of the principal goals of AI in this case is to provide a diverse and lasting experience, keeping player attention for prolonged time periods. Consequently, we decided to focus on the following game elements:

- *Believable, diverse AI characters.* To ensure long-term player retention, AI characters have to be diverse, fun to play with, and exhibit distinct play styles.
- *Different AI characters for different player profiles.* Like most free-to-play games, *World of Tennis* implements an extensive system of character upgrades, which encourage players to experiment with their play styles. For example, characters with low *running speed* skill values should generally stay close to the central axis of the court to maximize their chances of receiving

opponents’ shots. However, faster characters encourage more experimental and risky play styles. The AI system should provide interesting and challenging opponents for all variations of game characters.

- *The ability to train your “ghost” character.* Human-trainable “ghost” characters provide additional elements of gameplay. We want the AI to be able to serve as a substitute for the user, and complete the game session automatically if the user has no time or wish to do it.

These game elements are consistent with the goals of our previous research projects, dedicated to the AI of boxing and soccer [6, 7]. Thus, we decided to rely on the same method: learning AI play styles from real people, and using case-based decision making during the game. The ultimate goal of such an AI system is to replicate the actions of its human trainer. By accomplishing it, the AI will be able to support the aforementioned game elements, since the diversity of actual *World of Tennis* players will ensure the diversity of AI agents, and the players will be able to train their own “ghosts”.

## II. RELATED WORK

While *train your “ghost” character* capability (similar to “creating your own ghost” function in *Tekken 6* [8]) as a user-end feature immediately suggests a method based on human behavior observation, in general the task of building diverse and believable agents can be accomplished in different ways. We should note that the problem of creating a large family of AI-controlled characters is rarely addressed. Among them we can mention the work [9] that proposes to use multiobjective evolutionary algorithms to create the whole populations of NPCs. Due to stochastic nature of evolutionary algorithms, they can produce any required number of distinct characters.

When believability is explicitly stated as a goal, one of the following three general approaches is typically used:

1. *Rely on expert knowledge.* Decision-making logic can be directly hand-coded according to an expert view of the given domain. In certain cases this solution can be adequate, and result in a solid AI system [10].
2. *Mimic human decision making process.* A system can be designed on the basis of contemporary psychological theories of human behavior [11].
3. *Rely on actual logs of human behavior.* In the most straightforward form, this approach can be reduced to replaying human decision-making logs verbatim [12], but usually it is implemented via machine learning:



agents are trained on human data to provide the same actions as human players in similar situations [13, 14].

Our approach falls into the third category. We use logs of actual human-controlled characters to train the AI system. When the AI agent performs decision making, it relies on the acquired knowledge for case-based reasoning. The most relevant predecessor of the present project is our earlier work [7], dedicated to the game of boxing. Tennis and boxing actually share several notable aspects: both games feature one vs. one gameplay in a closed rectangular-shaped space; neither game requires long-term strategical thinking, but allow enough room for exhibiting a variety of play styles.

### III. WORLD OF TENNIS: ENGINE AND GAMEPLAY

Our previous experience shows that even minor changes in gameplay may require notable modifications of the AI learning and decision making procedures, so let us briefly discuss the game mechanics of World of Tennis, as it affects AI design.

World of Tennis is a mobile version of a conventional one vs. one tennis sports game. The player sees the whole court shown with a fixed camera: there is no need of scrolling or camera adjustments. The player always controls the bottom character, while the top character is controlled by the AI system (see Fig. 1).

There are only two possible action types available to the player. By tapping the screen area of the own side of the court, the player directs the character to the desired target point (*SetMovePoint* action). By tapping the screen area of the opponent side of the court, the player sets the target point for the next shot (*SetHitPoint* action). The game engine calculates the optimal shot parameters according to the current player skill values. For example, a player with a higher *shot power* value is able to hit the ball with higher speed. By tapping the opponent side of the court twice, the player forces the game engine to perform a high lob shot, typically used to loft the ball over the opponent.

In Fig. 1, the bottom player's active move point is shown with a cube (located near the center of the player side of the court), while the active hit point is represented with a sphere (located near the opponent).

When the player performs a *SetHitPoint* action, the game engine displays a *shot circle* (shown as a white circle near the center of the opponent side of the court in the Fig. 1). This circle represents the accuracy of the next player shot, and starts shrinking over time as soon as the opponent returns the shot. The actual target of the shot will be randomly selected within the circle limits. This way, the game engine encourages the players to choose their hit points as early as possible to ensure higher accuracy. However, the shot circle shrinks much quicker for the players with high *accuracy* skill values, so they have more flexibility in tactics.

The actual shot is performed as soon as the ball comes close to the player, and the next hit point is set. There are no actions required for making the shot: setting a hit point is sufficient. Furthermore, once the ball is shot towards the player, the system automatically starts steering the player to the

optimal ball receive location (it also depends on character skill values). Hence, the game process is strictly divided into distinct phases. When the ball is moving towards the player, the player can only set the next hit point, since its movements are controlled automatically. When the ball is moving towards the opponent, the player can only move to the specified court location. While it is also possible to set the next hit point in this phase, the game engine will not process it until the opponent returns the shot.

The general motivation of these design decisions is to facilitate tactical rather than pure arcade gameplay. The player is encouraged to select winning hit point and move point locations, while the game engine takes care of the rest.

If the player has no time or desire to finish an ongoing game, it is possible to pass the control to the "ghost" character at any moment. Ideally, the "ghost" should be able to replicate the player behavior style as closely as possible.

### IV. AI AGENTS: IMPLEMENTATION DETAILS

#### A. Learning Human Behavior

The core element of our AI system is its knowledge representation mechanism. We rely on the somewhat extended notation of a finite-state machine (FSM), later referred as *acting graph* [15]. Acting graph consists of  $N$  independent automatically generated FSMs representing an individual agent's knowledge with different levels of granularity (see Table I). Each state of the  $i$ -th machine incorporates  $A_i$  attributes representing a certain game situation. The states are connected with action-labeled edges. Unlike states, actions do not have levels of granularity, and are always represented with a complete set of attributes:

- Action type (*SetHitPoint* or *SetMovePoint*).
- Shot type (*None* for *SetMovePoint* actions, *Auto* or *Lob* for *SetHitPoint* actions).
- Target point position (x and y coordinates on the court).



Fig. 1. Actual screenshot of *World of Tennis*.

In addition, states and actions have auxiliary attributes, containing their usage frequency, timestamps, and other optional data fields.

The number of independent FSMs (i.e., levels of granularity) as well as the choice of their attributes is currently done manually in accordance with expert knowledge. The present design of attribute sets is a trade-off between AI quality, pure business goals, and mobile platform restrictions. While the AI, in principle, should be tuned for the best accuracy, we also had to set reasonable limits for the expected knowledgebase size, CPU load and required learning time. One of the business decisions was to ensure that any given “ghost” is able to reproduce basic player style elements after 7-10 minutes of realtime training.

TABLE I. ACTING GRAPH CONFIGURATION

FSM	Next level status <sup>d</sup>	Game Situation Attributes
FSM <sub>0</sub>		Player phase (serve / hit / receive / etc.)
		Player coordinates <sup>a</sup>
	M	Player <i>move point</i> coordinates <sup>ac</sup>
	M	Player actual <i>hit point</i> coordinates <sup>ac</sup>
	M	Opponent coordinates <sup>a</sup>
	R	Opponent <i>move point</i> coordinates <sup>ac</sup>
	R	Opponent actual <i>hit point</i> coordinates <sup>ac</sup>
	R	Player shot type (serve / top spin / lob / etc.) <sup>c</sup>
FSM <sub>1</sub>		Player phase (serve / hit / receive / etc.)
	M	Player coordinates <sup>a</sup>
		Player <i>move point</i> coordinates <sup>bc</sup>
		Player actual <i>hit point</i> coordinates <sup>bc</sup>
		Opponent coordinates <sup>b</sup>
FSM <sub>2</sub>		Player phase (serve / hit / receive / etc.)
		Player <i>move point</i> coordinates <sup>bc</sup>
		Player coordinates <sup>b</sup>
		Player actual <i>hit point</i> coordinates <sup>bc</sup>
		Opponent coordinates <sup>b</sup>

<sup>a</sup>. An integer value inside a 10×10 grid.

<sup>b</sup>. An integer value inside a 5×5 grid.

<sup>c</sup>. If exists.

<sup>d</sup>. M: attribute will be modified on the next level;

R: attribute will be removed on the next level.

During the game, the AI system observes the actions of the human-controlled character and inserts into the FSMs the (*state*, *action*, *state*) triplets, actually occurred in the tennis match [15]. As a result, we obtain a system of static FSMs that encode agent knowledge (this approach is similar to learning behavior trees from observation, discussed in [16]). The resulting structure can be also treated as a set of three Markov decision processes, representing human behavior.

## B. Decision Making Mechanism

In the simplest case, an AI agent can choose the next action by finding a match for the current game situation in the acting graph. If the agent learned a triple (*state*, *action*, *state*), it can decide that the best move in the game situation *state* is to perform *action*, and the anticipated outcome will be *state*. In practice, however, it is improbable that the agent will be always able to retrieve actions from the FSM<sub>0</sub> graph, most accurately representing user intentions. The procedure of case-based reasoning requires approximate matching, based in World of Tennis on the following three instruments.

1. If searching FSM<sub>0</sub> yields no results, we can repeat the same operation with reduced attribute sets for FSM<sub>1</sub> and FSM<sub>2</sub>. The resulting matches will be less precise, as any state retrieved from FSM<sub>1</sub> or FSM<sub>2</sub> represents the actual game situation with fewer details. Therefore, the retrieved actions might not accurately reflect user intentions anymore.
2. We can optionally turn off the requirement of action adjacency in decision making mode. For example, if the agent performed the action *A*<sub>1</sub> in the state 1, it should then continue acting from the state 2 (see Fig. 2). However, if the perfect match for the current game situation is a state other than 2, we can ignore the current sequence of actions forming player strategy, and continue from the best matching state. This relaxation can greatly increase the number of relevant actions on higher levels of abstraction. Say, if the state (2, 4) of FSM<sub>1</sub> corresponds to the states 2 and 4 of FSM<sub>0</sub>, and the previous agent action was *A*<sub>1</sub>, the only match in FSM<sub>1</sub> that preserves action adjacency would be *A*<sub>3</sub>. By relaxing the adjacency requirement, we can retrieve both *A*<sub>3</sub> and *A*<sub>4</sub>.
3. Numerical attributes (such as player coordinates) can optionally be matched approximately with a specified range of error. In the current implementation we only allow errors within a range [-1, 1], i.e., for a given value *V* the values *V* - 1, *V*, and *V* + 1 will be treated as acceptable matches.

The complete decision making procedure performs a sequence of queries until at least one matching action is found (see Table II). Each subsequent query implements further relaxations and thus can potentially retrieve more actions, albeit at the cost of reduced accuracy.

## C. Decision Making Points

One of the surprisingly difficult tasks for the AI subsystem design is to decide *when* to act. The frequency of AI decisions should be carefully balanced. Short time intervals between decision making points help the AI to react to ongoing changes on the court. However, it may cause jolted movements of the AI agents, since subsequent calls to the AI system override previous move and hit points. Furthermore, they increase CPU load. The present solution is based on the following observations:

1. The player moves to occupy a certain advantageous court location. It is unlikely that a reasonable game

strategy requires complex movements between the player shot and the subsequent opponent shot. The next move point can be set right after the player shot.

2. The player can set a hit point right after opponent shot. It is unlikely that a reasonable game strategy will require more than one revision of a hit point (since it will reduce the accuracy of the shot).

Hence, the interval between two consecutive player shots contains three decision making points: 1) right after the player shot (*SetMovePoint* expected), 2) right after the opponent shot (*SetHitPoint* expected), and 3) right after the ball covers a half of the distance between the players (*SetHitPoint* expected). One may notice that this algorithm actually shortens AI reaction time comparing to its human trainer.

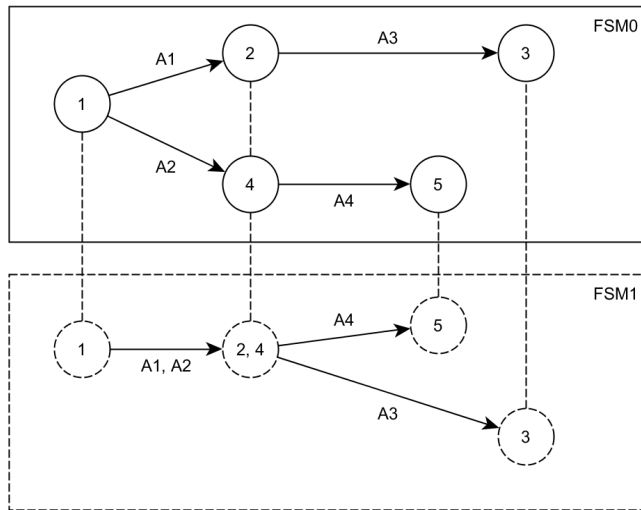


Fig. 2. Optional preservation of player strategy.

#### D. Action Ranking

When the decision making procedure has to rely on relaxed search conditions, it often returns a list containing a relatively large number (dozens) of actions, only approximately applicable in the current game situation. To improve agent behavior, we implemented the action ranking algorithm, based on the following simple rules.

1. Initially, each action is assigned a numerical weight equals to  $freq \times age$ , where  $freq$  is the number of occurrences of the given action in learning sessions, and  $age$  is the cardinal number of the current learning session. This way, both frequent and more recent actions are prioritized. The primary goal of  $age$  is to make the AI adjust to new patterns of player behavior that emerge due to character upgrades.
2. If the resulting list of actions contains an action similar to the one currently being executed by the agent, the system aborts decision making, and lets the agent to continue. The rationale is to keep the ongoing player activity if the AI system can prove that the current action is still relevant (i.e., present in the action list). Two actions are considered similar if the Euclidean

distance between their target points is less than *MinDist* value, currently set to one meter. This rule effectively suppresses unnecessary *SetHitPoint* actions set at the third decision making point.

After ranking, a weighted random choice is used to select the action to be returned to the game engine.

TABLE II. CASE-BASED REASONING WITH QUERIES

Query	FSM level	Require action adjacency	Range search on attributes
1	0	Yes	—
2	0	No	—
3	1	Yes	—
4	1	No	—
5	1	Yes	Player coordinates
6	1	No	Player coordinates
7	2	Yes	—
8	2	No	—
9	2	Yes	Player coordinates Player actual <i>hit point</i> coordinates Opponent coordinates
10	2	No	Player coordinates Player actual <i>hit point</i> coordinates Opponent coordinates

#### V. PLAY STYLE COMPARISON METHOD

Since an AI-controlled character in the game is typically intended to serve as a “ghost” of a certain human player, it has to exhibit a similar play style and demonstrate a comparable skill level as its human trainer. Evaluating skill level of an AI agent is a rather straightforward task: we can play a series of matches between two agents and check the final scores. In contrast, comparing play styles of two agents is a far more complex problem.

The task of play styles comparison is often discussed in connection with a more general task of evaluating character’s believability, i.e. its ability to provide the illusion of being controlled by a real human player. Since believable characters are not necessarily obtained by learning from human behavior, “gold standard” human behavior patterns might not be available. Consequently, believability assessment is often implemented in a form of a Turing test, where game observers have to evaluate believability of characters they see [17].

In our case, it is possible to evaluate human-likeness of an AI-controlled character via direct comparison of AI-generated and human-generated samples of game play. In its turn, there is no universal scheme of game style comparison, as relevant game style features highly depend on a particular game. For example, promising results for a first-person shooter game were obtained with player trajectories comparison [18], while in Super Mario Bros. a scalar *performance score* function incorporating various player achievements was shown to serve as a reasonable criterion of behavior similarity [13].

The discussion of features that constitute a play style for tennis deserves a separate study. For the current purposes, we decided to develop the simplest possible scheme that can separate human players reliably. Presumably, humans exhibit distinct play styles, while the same person keeps a consistent

play style across games. Therefore, the desired similarity function in most cases should produce higher similarity values for the same player in different games, and lower values for different human players.

One of the approaches that turned out to be unreliable was based on plain heat map comparison. We represented the character side of the court with a two-dimensional table ( $10 \times 10$  cells), then used actual game logs to calculate the probability of character presence in each court cell. Such probability tables (heat maps) were transformed into one-dimensional arrays (vectors), and compared using a dot product, yielding a similarity value within a range  $[0, 1]$ . However, the experiments showed that even pairs of completely unrelated characters often generate similar heat maps.

Our current comparison algorithm uses a modified heat map approach, based on a presumption that a play style is defined with explicit player actions rather than character presence in certain court locations. The algorithm first builds an independent heat map for each character's *SetMovePoint* and *SetHitPoint* actions found in a game log:

```

For each SetMovePoint action a:
    MoveF[a.x, a.y]++
    MoveCount++

For each SetHitPoint action a:
    HitF[a.x, a.y]++
    HitCount++

For x = 0...9 and y = 0...9:
    MoveP[x, y] = MoveF[x, y] / MoveCount
    HitP[x, y] = HitF[x, y] / HitCount

```

Action target coordinates *a.x* and *a.y* are here represented with scaled integer values lying in the range  $[0, 9]$ . Two resulting heat maps of each character are then converted into vectors and concatenated. The obtained vector serves as a character's "behavior fingerprint", and can be compared with other vectors with a dot product.

## VI. EXPERIMENTAL RESULTS

To evaluate the quality of the resulting AI system we performed a series of experiments, answering the following research questions:

*Q1.* Do human players exhibit distinguishable play styles?

*Q2.* Do AI "ghost" characters exhibit distinguishable play styles, similar to the styles of their human trainers?

*Q3.* Do AI "ghost" characters exhibit tennis skills, comparable to the skills of their human trainers?

### A. Preparation of Experimental Dataset

In our experiments we rely on behavior data of eight human players  $H_1, \dots, H_8$ , and one "AI coach". Human players highly vary in their skill level, but all of them have some experience of World of Tennis gameplay. The AI coach is a "ghost", deliberately trained to be a reasonable-skilled opponent for entry-level players. All the characters in the game have comparable skill values (running speed, shot power, etc.), kept with no significant adjustments during the experiments.

A single *game session* in the experiments consists of a short match that lasts until a) one of the players scores 7 points; and b) the difference between scored points of the opponents is at least two. On average, the actual duration of an individual game session is around 2 minutes. Any number of game sessions can be used to train a character's AI "ghost" or to construct a behavior fingerprint.

The actual sessions were designed to replicate the real everyday World of Tennis activities. All the human players were put into the same "league" and assigned opponents according to a round robin scheme. Thus, each player has to play seven game sessions as a host against another player's AI "ghost", and seven game sessions as a guest, serving as an AI "ghost" opponent for another player. Each league day begins with a training, when a player has a chance to play 1-2 game sessions against the AI coach or, starting from day two, against the AI "ghost" of the player currently leading in the tournament. After the training, the player has to play the next scheduled game.

In total, it gives us (per player) 7 league games played as a host, 7 league games played as a guest, and 7.88 training games on average. Our present experiments do not take into account that the "ghost" are trained on the fly, which means that their skill level and ability to capture human play styles should presumably be higher for the latter game sessions.

### B. Primary Experiments

*Experiment 1.* We used the obtained game data to build a behavior fingerprint of each human player  $H_i$ , and compared the fingerprints of every human character pair  $(H_i, H_j)$ . To check whether a character's play style is consistent across different games, we also compared two fingerprints of the same human characters, obtained on three randomly selected sessions. The results of fingerprint comparison are shown in Table III.

TABLE III. HUMAN PLAY STYLE SIMILARITY

<b>H<sub>1</sub></b>	0.96								
<b>H<sub>2</sub></b>	0.58	0.94							
<b>H<sub>3</sub></b>	0.54	0.76	0.93						
<b>H<sub>4</sub></b>	0.75	0.58	0.55	0.93					
<b>H<sub>5</sub></b>	0.88	0.49	0.56	0.57	0.97				
<b>H<sub>6</sub></b>	0.77	0.59	0.67	0.83	0.64	0.96			
<b>H<sub>7</sub></b>	0.81	0.69	0.60	0.74	0.69	0.68	0.96		
<b>H<sub>8</sub></b>	0.74	0.78	0.62	0.81	0.57	0.72	0.89	0.93	
	<b>H<sub>1</sub></b>	<b>H<sub>2</sub></b>	<b>H<sub>3</sub></b>	<b>H<sub>4</sub></b>	<b>H<sub>5</sub></b>	<b>H<sub>6</sub></b>	<b>H<sub>7</sub></b>	<b>H<sub>8</sub></b>	

$$\begin{aligned} \text{mean}(\{(H_i, H_j), i = j\}) &= 0.95, \sigma = 0.02 \\ \text{mean}(\{(H_i, H_j), i \neq j\}) &= 0.68, \sigma = 0.11 \end{aligned}$$

The similarity values show that different fingerprints of the same human player are consistently closer than the fingerprints of two distinct players. Since fingerprints are essentially heat maps, it is also easy to visualize them to show the differences in human play styles (see Fig. 3).

One may notice that the present fingerprinting scheme was *explicitly designed* to separate human players, and therefore cannot reliably prove that humans indeed exhibit play styles that are perceived as distinct by external observers. However, since the fingerprinting algorithm is a very straightforward

simplification of human behavior data, we believe that major contradictions between the automated scheme and human evaluations are unlikely. Obviously, the current algorithm is unable to detect subtle differences in individual play styles, thus the average similarity between different human players is high (0.74).

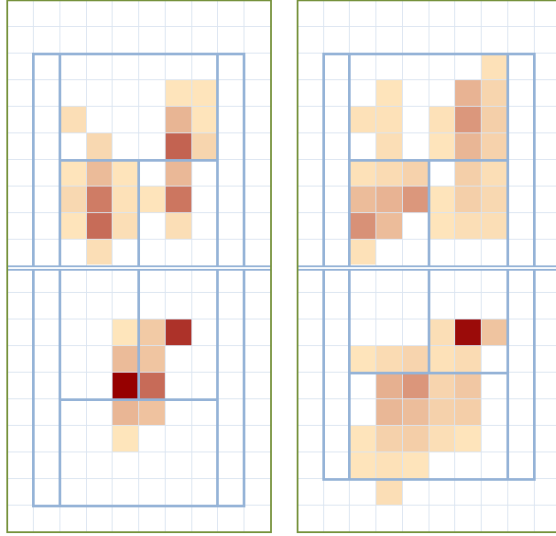


Fig. 3. Heat maps of  $H_1$  and  $H_2$  (players are always in the bottom half of the court; the top half is used to show *SetHitPoint* targets).

*Experiment 2.* We repeated the steps of *Experiment 1* for the games played by AI “ghost” characters  $G_1, \dots, G_8$ , and compared behavior fingerprints of human and “ghost” characters to obtain human/ghost and ghost/ghost similarity values (see Table IV and Table V).

Table IV shows that the “ghost” players indeed exhibit diversity of behavior, comparable to their human trainers. “Ghosts” tend to be consistent in their play style, and each “ghost” is clearly identifiable. Despite several exceptions (such as relatively low  $G_7$ - $G_7$  similarity), on average behavior data of “ghosts” is comparable to the results shown by human players.

Table V demonstrates that our current algorithm is indeed able to produce AI “ghosts” exhibiting play style closer resembling their human trainers rather than other human players or other “ghosts”.

TABLE IV. AI “GHOST” PLAY STYLE SIMILARITY

$G_1$	0.91								
$G_2$	0.67	0.95							
$G_3$	0.74	0.77	0.92						
$G_4$	0.82	0.47	0.69	0.94					
$G_5$	0.75	0.44	0.64	0.72	0.96				
$G_6$	0.88	0.59	0.79	0.90	0.74	0.85			
$G_7$	0.84	0.49	0.69	0.86	0.81	0.87	0.80		
$G_8$	0.84	0.53	0.69	0.78	0.89	0.82	0.92	0.91	
	$G_1$	$G_2$	$G_3$	$G_4$	$G_5$	$G_6$	$G_7$	$G_8$	

$$\begin{aligned} \text{mean}(\{(G_i, G_j), i = j\}) &= 0.91, \sigma = 0.05 \\ \text{mean}(\{(G_i, G_j), i \neq j\}) &= 0.74, \sigma = 0.13 \end{aligned}$$

*Experiment 3.* We calculated the total number of points scored by the human players and the AI “ghosts” in league games, and visualized them (see Fig. 4). This experiment shows that the “ghosts” can play on par with human opponents. Their slightly better performance can be explained primarily with higher “discipline”: AI-controlled characters never miss their actions, while human players tend to play casually, and sometimes may skip the next hit or move point.

During the experiments, we collected additional data that may be useful to evaluate the resulting AI system. These numbers are summarized in Table VI.

TABLE V. HUMAN / AI “GHOST” PLAY STYLE SIMILARITY

	$H_1$	$H_2$	$H_3$	$H_4$	$H_5$	$H_6$	$H_7$	$H_8$
$G_1$	0.89	0.82	0.66	0.77	0.73	0.75	0.82	0.84
$G_2$	0.38	0.89	0.69	0.47	0.35	0.50	0.47	0.62
$G_3$	0.57	0.80	0.95	0.63	0.50	0.75	0.63	0.68
$G_4$	0.85	0.61	0.61	0.94	0.70	0.92	0.80	0.81
$G_5$	0.85	0.57	0.69	0.60	0.97	0.69	0.70	0.60
$G_6$	0.83	0.74	0.73	0.86	0.71	0.88	0.82	0.84
$G_7$	0.86	0.69	0.70	0.77	0.79	0.79	0.88	0.87
$G_8$	0.86	0.73	0.73	0.70	0.88	0.73	0.84	0.83

$$\begin{aligned} \text{mean}(\{(H_i, G_j), i = j\}) &= 0.90, \sigma = 0.05 \\ \text{mean}(\{(H_i, G_j), i \neq j\}) &= 0.71, \sigma = 0.13 \end{aligned}$$

TABLE VI. ADDITIONAL EXPERIMENTAL DATA

Actions matched in $\text{FSM}_0$	18%
Actions matched in $\text{FSM}_1$	56%
Actions matched in $\text{FSM}_2$	18%
No matching actions found	8%
Adjacent (strategy-keeping) actions	14%
Average AI “ghost” graph size ( $\text{FSM}_0$ )	525 nodes

## VII. DISCUSSION

Before discussing obtained results, let us once again express the view that the purpose of game AI is to support a particular game experience. This way, our goal was to create an AI system for a specific type of a mobile tennis game, and certain design decisions were influenced by game designers’ views of typical game/player interaction scenarios. Since the game is still in the development stage, we believe that both business requirements and the game engine will be updated, so we will have to amend certain AI elements accordingly.

Our first goal was to create a variety of believable and diverse characters. As a preliminary step, we had to make sure that the game engine itself allows the players to exhibit distinguishable play styles, since one may argue that the difference between two human players can be merely explained with their different reaction time and differently upgraded characters. However, Experiment 1 shows that each person tends to adhere to the same distinctly recognizable characteristic behavior pattern (at least over a short span of 10-20 game sessions), thus answer positively to the research question  $Q_1$ . Therefore, believable AI characters should also be able to exhibit a variety of recognizable game play styles.

Experiment 2 confirms that our method is indeed able to produce believable and diverse AI “ghosts” that resemble their



human trainers (research question  $Q_2$ ). The diversity of “ghosts” is comparable to the diversity of human players. A similarity score of two distinct “ghosts” in the experiment is 0.74 on average, while each “ghost” exhibits consistent play style across the game sessions (yielding a self-similarity score of 0.91 on average). These results are comparable to the similarity values obtained from human players in the Experiment 1. The similarity scores between the “ghosts” and their respective human trainers vary. In most cases, the closest match to a “ghost” is indeed its human trainer (with the average similarity score 0.90).

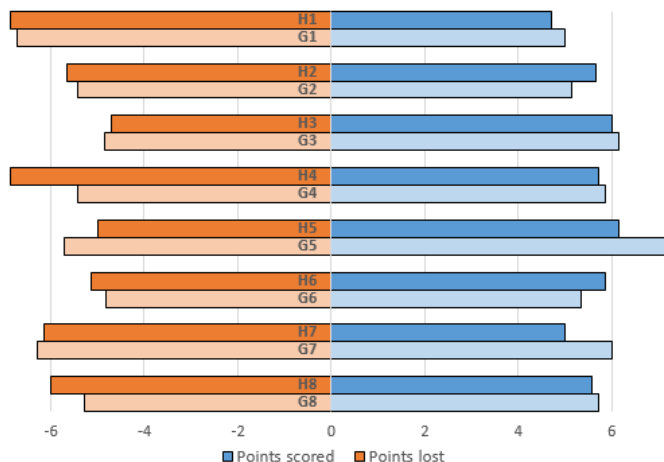


Fig. 1. Number of points lost and scored (on average per game) by the human players and their respective AI “ghosts”.

Experiment 3 demonstrated that in the AI “ghosts” are able to compete with people successfully, and in the present version of the AI system perform even slightly better than their human trainers (probably, due to their perfect discipline). From the business perspective, we are satisfied with this moderate handicap the “ghosts” have. Human players always play against the AI, so the players generally have no expectations about the skills of their opponents. At the same time, each player expects own “ghost” to perform equally well, so in this case stronger AI should be more appealing to the users.

### VIII. CONCLUSION

We demonstrated a practical learning by observation-based method used to create an AI system for a mobile tennis game. The chosen approach provides reliable decision making, and is able to produce a variety of diverse human-like opponents, preserving play style of their human trainers. Believability and diversity of AI-controlled characters were demonstrated with an objective method based on behavior fingerprint comparison. Furthermore, the AI agents are able to achieve the same skill level as human players.

The proposed approach is based on the earlier AI project for the game of boxing, so we believe that this method can be adapted to a variety of game genres. At the same time, every game is different, and even small changes in a game engine or gameplay may require considerable redesign of AI system. In case of tennis, the core elements of decision making system were kept intact, however, the design of granularity levels, the

choice of attributes and queries had to be reconsidered. The resulting AI system is resource-efficient, and can work on an average (iPhone 4-class) mobile device.

Currently several core decision making subsystems (such as configuration of FSMs and the list of queries) are designed manually, on the basis of expert knowledge. While automating these tasks is not a priority for us now, it can be a fruitful topic for future research.

### REFERENCES

- [1] N. Taatgen, M. van Oploo, J. Braaksma, and J. Niemantsverdriet, “How to Construct a Believable Opponent using Cognitive Modeling in the Game of Set,” in *5th International Conference on Cognitive Modeling*, 2003, pp. 201–206.
- [2] S. Johnson, “Playing to Lose: AI and “CIVILIZATION” (Lecture),” Game Developers Conference, 2008.
- [3] K. Dill, “What is Game AI?,” in *Game AI pro: Collected wisdom of game AI professionals*, S. Rabin, Ed, pp. 3–10.
- [4] *World of Tennis*. Available: [www.worldoftennis.com](http://www.worldoftennis.com).
- [5] P. Luban, *The Design of Free-To-Play Games*. Available: [http://www.gamasutra.com/view/feature/6552/the\\_design\\_of\\_frektoplay\\_games\\_.php](http://www.gamasutra.com/view/feature/6552/the_design_of_frektoplay_games_.php) (2016, Mar. 31).
- [6] M. Mozgovoy and I. Umarov, “Believable team behavior: Towards behavior capture AI for the game of soccer,” in *8th International Conference on Complex Systems*, 2011, pp. 1554–1564.
- [7] M. Mozgovoy and I. Umarov, “Building a believable agent for a 3D boxing simulation game,” in *Second International Conference on Computer Research and Development*, 2010, pp. 46–50.
- [8] Bandai Namco Games, *Tekken 6 Instruction Manual (PSP)*, 2009.
- [9] A. Agapitos, J. Togelius, S. M. Lucas, J. Schmidhuber, and A. Konstantinidis, “Generating diverse opponents with multiobjective evolution,” in *IEEE Symposium on Computational Intelligence and Games*, 2008, pp. 135–142.
- [10] A. M. Mora, F. Aisa, P. García-Sánchez, P. Á. Castillo, and J. J. Merelo, “Modelling a Human-Like Bot in a First Person Shooter Game,” *International Journal of Creative Interfaces and Computer Graphics (IJICIG)*, vol. 6, no. 1, pp. 21–37, 2015.
- [11] Asensio, Joan Marc Llargues *et al*, “Artificial Intelligence approaches for the generation and assessment of believable human-like behaviour in virtual characters,” *Expert Systems with Applications*, vol. 41, no. 16, pp. 7281–7290, 2014.
- [12] I. V. Karpov, J. Schrum, and R. Miikkulainen, “Believable Bot Navigation via Playback of Human Traces,” in *Believable Bots: Can Computers Play Like People?*, P. Hingston, Ed, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 151–170.
- [13] J. Ortega, N. Shaker, J. Togelius, and G. N. Yannakakis, “Imitating Human Playing Styles in Super Mario Bros,” *Entertainment Computing*, vol. 4, no. 2, pp. 93–104, 2013.
- [14] G. Lee, M. Luo, F. Zambetta, and X. Li, “Learning a Super Mario controller from examples of human play,” in *2014 IEEE Congress on Evolutionary Computation (CEC)*, 2014, pp. 1–8.
- [15] M. Mozgovoy and I. Umarov, “Behavior Capture with Acting Graph: A Knowledgebase for a Game AI System,” in *Databases in Networked Information Systems (DNIS): 7th International Workshop*, S. Kikuchi, A. Madaan, S. Sachdeva, and S. Bhalla, Eds, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 68–77.
- [16] G. Robertson and I. Watson, “Building behavior trees from observations in real-time strategy games,” in *International Symposium on Innovations in Intelligent Systems and Applications (INISTA)*, 2015, pp. 1–7.
- [17] J. Togelius, G. N. Yannakakis, S. Karakovskiy, and N. Shaker, “Assessing believability,” in *Believable bots*: Springer, 2013, pp. 215–230.
- [18] H.-K. Pao, K.-T. Chen, and H.-C. Chang, “Game Bot Detection via Avatar Trajectory Analysis,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 3, pp. 162–175, 2010.



# Design Influence on Player Retention : A Method Based on Time Varying Survival Analysis

Thibault Allart<sup>\*†§</sup>, Guillaume Levieux<sup>†</sup>, Michel Pierfitte<sup>\*</sup>, Agathe Guilloux<sup>§</sup> and Stephane Natkin<sup>†</sup>

<sup>\*</sup>Ubisoft Production Internationale

126 rue de Lagny, 93100 Montreuil-Sous-Bois

<sup>†</sup>Conservatoire National des Arts et Metiers

292 rue Saint Martin, 75141 Paris

<sup>§</sup>Universite Pierre et Marie Curie (Paris 6)

4, place Jussieu, 75252 Paris

Emails: thibault.allart@gmail.com, guillaume.levieux@cnam.fr , michel.pierfitte@ubisoft.com,  
agathe.guilloux@upmc.fr , stephane.natkin@cnam.fr

**Abstract**—This paper proposes a method to help understanding the influence of a game design on player retention. Using Far Cry<sup>®</sup> 4 data, we illustrate how playtime measures can be used to identify time periods where players are more likely to stop playing. First, we show that a benchmark can easily be performed for every game available on Steam using publicly available data. Then, we introduce how survival analysis can help to model the influence of game variables on player retention. Game environment and player characteristics change over time and tracking systems already store those changes. But existing model which deals with time varying covariate cannot scale on huge datasets produced by video game monitoring. That is why we propose a model that can both deal with time varying covariates and is well suited for big datasets. As a given game variable can have a changing effect over time, we also include time-varying coefficients in our model. We used this survival analysis model to quantify the effect of Far Cry 4 weapons usage on player retention.

## I. INTRODUCTION

The amount of data collected by game companies about how their games are played is constantly growing. Indeed, consoles and PC are almost always connected to the internet, allowing Game Studio to virtually track every player interaction with the game. However, interpreting this massive amount of data is still a key challenge for game companies.

One of the goal pursued by game developer is not only to have many players, but to have each of them playing a lot. Players should spend more time than just trying the game for a few minutes : they should really enjoy the game and explore the whole content developers spent a long time to create. Having many players leaving the game in the first hours could really be considered as a design failure.

Game studios are thus often analyzing a game's *player retention*, that is to say, the proportion of remaining players after  $n$  hours of playtime. In this paper, we focus on using statistical analysis of tracking data to improve player retention. We need to help the designers to identify the design elements that seem to maintain players inside the game, and those who don't. By nature, game design is an incremental process and

games need to be tested to evaluate if players have fun playing it [17].

A lot can be learned by only monitoring playtime, including detecting when a player is about to leave the game. For instance, an issue with the tutorial could be highlighted by a player retention drop in the first minutes of the game. However most thing are not as easy to identify. An unbalanced weapon will have an unnoticeable impact in the quit rate curve, even if it causes lots of players leaving the game. Different players will use the given weapon at different times in the game leading to a wide spray on the time scale. A good way to detect such impact is to use statistical modeling. In order to inform designers on which modifications can be done, we need to investigate how in-game metrics are related to retention.

One of the most interesting aspects of video game tracking systems is that they allow us to get a large amount of accurate data on player behavior. In contrast, in the field of health care, the number of individuals is closer to hundreds than millions. Furthermore, in longitudinal studies, the patients might be monitored every week. In video games, we can track any action anytime in the virtual world. This deep tracking can help us to better understand the design's link with player retention but we have to adapt our model to be able to handle such a high amount of data.

## II. CONTRIBUTION AND MAIN RESULT

This paper proposes a method to extract actionable insight from tracking data. There exists many black box models, e.g. random forests or deep learning, that may be used to predict player departure, but, as our goal is to give actionable insight to designers, we are restricted to interpretable models.

First, we describe how a simple metric like playtime can be used to identify time periods when players stop playing the game. We also show that this simple metric can be calculated on publicly available data, and thus can be used by anyone to realize a benchmark and see how a given game differs from concurrent ones. Indeed, online gaming platforms, like Steam, are gathering data on their players, and some of these data

can be retrieved by anyone. This means that any researcher can nowadays analyze high level data on gamer's playing habits. However, this data are often limited to the time spent by players in each game, and do not provide any insight on the player's detailed behavior within the game.

Then, we develop how to model the link between the player's behavior and the player's departure. Thereby, we introduce a well-known survival analysis model and show how it can be used with constant-over-time data. Such model works well for covariates that don't change over time. However, using this model on time-varying data, which is often the case in video games, can lead to wrong conclusions.

Finally, we detail the complexity of time-varying data. We show how information can be visualized and explain why we need to take time-varying covariates into account. We introduce a new algorithm that is able to both analyze a high amount of tracked data and to take into account time-varying covariates. We show how it can be applied to analyze Far Cry 4 weapon usage, and what kind of design recommendations can be done.

### III. RELATED WORKS

A video game is an often-complex interactive multimedia system. As such, it can be described by many quantifiable variables, depending on the *gameplay* it provides as well as, for instance, technical aspects. Gameplay is a relationship between the player and the game system and thus gameplay variables may describe part of the game system, e.g. a gun's rate of fire, as well as the player's behavior, e.g. the preferred weapon. Technical variables can describe the position of a HUD's icon to the average amount of network lag. Many of these variables can be modified by the development team, and are often called game features when related to the gameplay. We will use the more abstract term of *game variables*.

Game analytics is a wide area including visualization [18], [4], clustering [2], [21] and prediction [12], [20]. In this work we focus on ways to understand the link between game variables and the risk that players quit the game. This topic has already been treated by some authors.

Weber[19] used a regression model to determine which game variable might have the most influence on player retention. In this research, Weber et al. study the case of an american football game in which the player has to perform multiple matches. Instead of directly predicting the game duration, the authors use the number of matches played as a target variable. Our approach extend their work for cases where we do not study a number of matches but a more general duration measure, as playtime.

Harrison et al[10] used N-gram models to dynamically attribute quests to players. They are thus able to select a sequence of quests that players are more likely to accept in order to increase player retention. However such models present an exponential complexity with the number of different actions available for the player, and might such be inapplicable to datasets with many game variables.

The largest study on playtime distribution was done by Bauckhage [3] on approximately 250 000 Steam players. They concluded that playtime seems to follow a Weibull distribution. However, using Ubisoft® tracking data, we observe that Ubisoft games seem to follow a log-Normal distribution. This can be explained by differences in the tracking system. Furthermore, playtime distribution of free-to-play games is quite different from paying games due to a high proportion of *droppers* (players who played only one day). Knowing that, using a parametric regression model (based on density distribution) could not be adapted to every games. We solve this problem using a semi-parametric approach.

Survival analysis has already been used in video game analysis. Chen[5] use survival analysis to quantify the effect of network quality on player retention in Online Games. They found that both network delay and network loss have a link with player retention. We propose to extend such work by including time-varying information in the model.

### IV. METHOD

In this section we propose a method to analyze a video game's retention, starting from a general, simple approach to a more complex modeling of the link between game variables and player retention.

First, we propose to analyze playtime and compute a *hourly survival curve*, that can both be used to get a first insight on the game's retention as well as to compare it to other games with publicly available data. Then we introduce a way to quantify the link between a given game variable and player retention, namely the Cox model. After what we point out that a lot of game variables change over time. This temporal evolution has to be included in the model, otherwise it can lead to wrong results. We propose a model that includes time-varying covariates in Cox model and can deal with huge dataset.

#### A. Playtime Analysis

Playtime is one of the most important metric in video game analysis. First, playtime allows us to know when the players stops playing. From this, we can derive and represent metrics like *retention*, *quit rate* and *hourly survival*.

1) *Retention*: Retention measures how many players remain in the game after a given time. We focus here on playtime retention, but it can be computed using different time measure (lifetime, played days, ...). Retention is represented as players survival curve. Let  $T$  being the random variable of time at which a players leaves the game, then we have :  $S(t) = \mathbb{P}(T > t)$

If we want to know when we lost the most players we would better look at quit rate.

2) *Quit Rate*: The hazard function, also called quit rate in video game industry, gives us the instantaneous rate at which players quit the game :

$$h(t) = \lim_{\Delta t \rightarrow 0} \frac{\mathbb{P}(t \leq T < t + \Delta t | T \geq t)}{\Delta t} \quad (1)$$

Retention and Quit Rate only needed a playtime measure to be computed.

A locally high value of quit rate may indicate a problem in your game.

3) *Hourly survival*: Using a one hour binning, quit rate can be interpreted as how many players stay from one hour to the next one. We call this *hourly survival* in the next section. This metric is less accurate than quit rate and is useful when we don't have a very precise measure of playtime, as it is the case in our benchmark data.

## B. Benchmark

Retention metrics can be analyzed both in an absolute and relative fashion. First, in an absolute fashion, any inflection in the curve might reveal a design problem and be investigated. But from a relative point of view, comparing the game's retention curve to other similar game curves can help to understand where the game strengths or weaknesses are. It is hard to tell whether a 85% first hour hourly survival is good, but if at some point, the game is much worse than many other similar games, then there might be a specific, hopefully fixable issue.

However, to perform such a comparative analysis, we need to know about many games playtime information, and many game companies do not share this kind of information. Hopefully, Valve's Steam online gaming platform provides an A.P.I that allow developers to query part of Steam users information, such as playtime information about the game titles owned by a specific player. Such an API may be used to regularly gather playtime information from a random set of players and then compute playtime statistics, as did for instance Bauckhage et al [3].

Moreover, this approach is used by the website SteamSpy [9] to give an approximation of playtime distribution for every Steam games. Thus, if we want to get high level information about playtime for Steam games, we do not need to directly query the Steam API but can rely on SteamSpy data instead. Currently, SteamSpy do not provide any API but data can easily be retrieved manually from the website.

Of course, data from SteamSpy are less accurate than internal data from the in game's tracking system. That is why we prefer to perform an hourly binning and thus look at the hourly survival curve in that case. Each value is the percentage of players who played for at least  $n$  hours and will continue playing for at least  $n+1$  hours. For instance we have computed the hourly survival curve for some of the most played games on Steam, based on SteamSpy data (fig. 1).

These curves can be interpreted as follows : among Counter Strike Global Offensive players who have played at least 4 hours, 98% of them have played at least 5 hours. Which means that 2% of the players who played at least 4 hours have left the game between 4 and 5 hours of playtime.

We can see that Dota 2 and Team Fortress 2 have around 80% of hourly survival in the first hour. That means that around 20% of players leave the game in the first hour of playtime. We may hypothesize that as both these games are free to play games, much more player will test them than if they had to pay for it. We may also speculate that these player

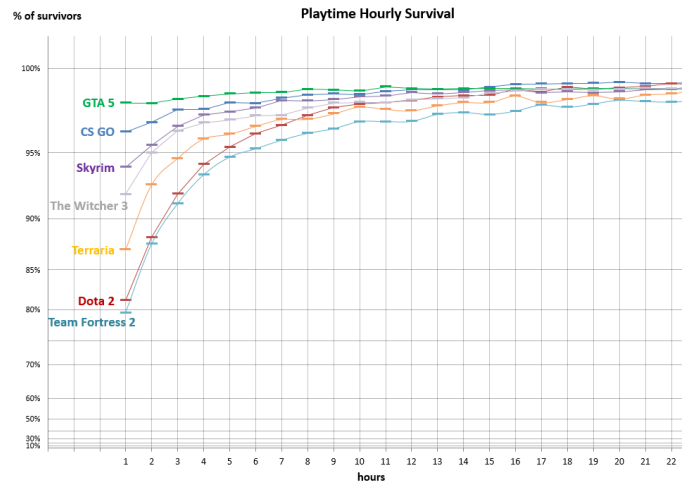


Fig. 1. Hourly survival of some games available on Steam using a logarithmic scale

have more chance to stop playing the game, thus leading to a low hourly survival during the first hours of the game.

On the contrary, Grand Theft Auto V has an exceptionally flat hourly survival curve. The game succeeds in retaining more than 98% of players hour by hour.

Looking at hourly survival curve and Quit rate may help to identify temporal windows where lots of players quit the game. In linear games it can be linked to a given game level. However, in non-linear game such as open-world games where the gameplay leaves much more freedom to the player, it is often impossible to relate a temporal measure like playtime to a spacial or a progression measure like player's position or player's progression in the missions.

Playtime analysis is thus inherently limited, and while it is a valuable first step, one will want to include other variables in the analysis and build a more thorough model.

## C. Cox Regression

There are many ways to model the effect of covariates on a given metric. In our case we are interested in the retention, which is a duration value. Modeling the player retention can be done by performing a *survival analysis*. One of the most used model in survival analysis is the Cox proportional hazard [6]. It considers that every player has a given risk to quit the game at time  $t$ . This risk depends on each player's individual characteristics, represented by a set of  $p$  game variables ( $X_1, \dots, X_p$ ) in the model. A player characteristics can be considered as a quantifiable variable, that can be measured for any player, and that gives us information about a game variable we want to study. For instance, if we want to know if the AI is not too strong on close combat, which might be the game variable "AICloseCombatDamage", we will calculate the player characteristic "NbDeathInCloseCombat", counting the number of deaths for each player while in close fight with an AI. The  $\beta_j$  related to "NbDeathInCloseCombat" might then help us tune the AI with regard to retention. We note  $X_{i,j}$  the value of the covariate  $j$  for a given player  $i$ . Coefficient  $\beta_j$

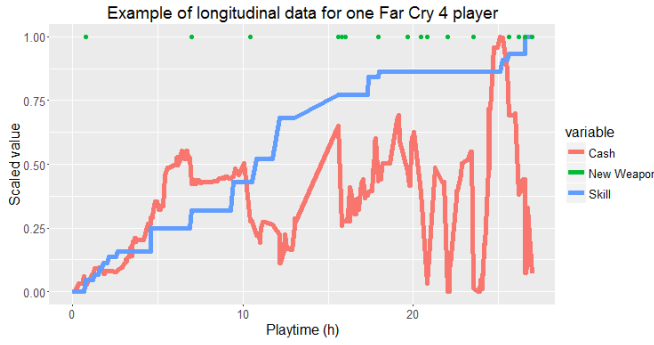


Fig. 2. Example of longitudinal data for one Far Cry 4 player. Blue curve represent the number of skills unlocked by the player. Red curve is the amount of cash own by the player. Green point indicate that player get a new weapon.

quantifies the effect of the covariate  $j$  on the hazard rate. The risk that this player stops playing at a given time  $t$  is modeled as

$$\lambda(t|X_i) = \lambda_0(t) \exp \left( \sum_{j=1}^p X_{i,j} \beta_j \right) \quad (2)$$

This model is called proportional hazard because the quit rate function  $\lambda$  of each player is proportional to a baseline quit rate function  $\lambda_0(t)$ .

Cox survival analysis will help us create a model of game variables on player retention. However, we must take into account the fact that while the player gender or age are constant, many game variables that we want to study vary a lot during the game. In a shooter game, the player does not use the same gun during the whole game. In a Role Playing Game, character skills change over time and the way a player uses them vary over time. Our model needs to take into account such time-dependent covariates.

#### D. Time Varying data

Covariates may change very often during a game session, like the number of enemies the player has killed, the amount of money he has won or spent, or his position in the world. Each player characteristic can thus be considered be as a time series curve (fig. 2).

A classical way to deal with time-varying data would be to add new covariates that try to summarize them, such as the  $mean(x(t))$ ,  $var(x(t))$  or to decompose them as basis functions (Wavelets).

However, summarizing time-varying covariate leads to a loss of information, and basis decomposition leads to a loss of interpretation. Furthermore, applying a model made only for constant data on aggregated time varying data leads to a scaling problem. Metrics of a player who played 5 minutes will be compared to those of a player who played 80 hours. One can try to create a ratio by dividing game variable by playtime, but often metric evolution over time is not linear in time. In fact metric evolution is related to design and game variable cannot simply be compared at different time.

To avoid all these problems we introduce a model that deals with time varying game variables  $X(t)$ . For almost the same complexity cost we also allowed the covariates effect to change over time  $\beta(t)$ , meaning that the same element can have various effect depending on time.

#### E. Modeling Time varying data

Extensions of Cox proportional hazard have been proposed to deal with time-varying covariates and coefficients [14].

$$\lambda(t|X_i) = \lambda_0(t) \exp \left( \sum_{j=1}^p X_{i,j}(t) \beta_j(t) \right) \quad (3)$$

However current implementations of such model are based on matrix inverse and iterative kernel smoothing that makes the model unable to deal with huge datasets. We propose a piece wise constant model that makes the minimization problem separable in the number of individuals. This allow us to use a stochastic gradient descent algorithm as Adagrad [7] or Adadelata [22] to solve the minimization problem. Technical part and theoretical guaranties are developed in [1]. Stochastic gradient algorithm allows us to load only few datapoints in main memory and thus to analyze datasets that are bigger than RAM capacity.

Our model has been implemented in a C++ library and interfaced with the R programming language [15] thanks to RCpp [8], to facilitate it usage by game analysts.

As it is the case with Cox proportional hazard, coefficient can be directly interpreted as an effect on retention. Time-varying coefficients cannot be summarized as a table of scalar value. Instead we need to plot the  $\beta(t)$  curves. A positive value of the coefficients means that the game variable related to this coefficient has a positive link with quit rate, meaning that player has higher chance to quit the game in a small time interval. Each curve can be interpreted as follows : for a given time  $t$ , if we set all features value to a constant except the one of interest  $X_j(t)$ , a one point increase in the feature value  $X_j(t)$  leads to multiplying the chance that players will leave the game in a short period of time by  $\exp(\beta_j(t))$ .

#### V. DATASETS

As Far Cry 4 is a shooter game, weapon usage can be considered as part of its core gameplay. Thus, we will focus our analysis on weapon usage but the method can be used with any covariates that may be tracked, time-varying or not.

##### A. Far Cry 4

Far Cry 4 is a first person shooter in which the player explores an open world named Kyrat. The game was released on November 18, 2014. During the game, the player discovers new weapons. He can carry four of them and switch between them to modify his strategy. We consider that weapon selection is highly related to play style and can thus give a lot of information on which part of the gameplay is experienced by the player. A player who wants to play stealthily could approach a camp using a long range weapon, as a sniper rifle equipped with a silencer, then a Bow, and finally get to his

objective using a short range silenced weapon. Another player could choose to fly over the camp with an helicopter armed with a one hand rocket launcher, shoot three or four rockets over the main guarded zone before landing inside and draw his shotgun to complete the mission, which will be a completely different play style and thus related to different part of the gameplay Far Cry 4 has to offer. As a result, if rocket users have a very different retention than silencer users, then it might give us a useful insight on which part of the gameplay the developers should modify.

### B. Metrics description

Far Cry 4 tracking system monitors the number of kills made with the following weapons :

- Assault including AK-47, STG-90, F1, MS16, P416 and A52
- Auto Crossbow
- Bait. Player can launch meat to bait animals who can attack enemies. The kill is then attribute to player.
- Bow including Recurve Bow and Hunter Bow
- Grenade
- Knife thrown
- Light Machine Guns including PKM, U100, MKG, MG42 and DshK
- Machete. Available only in Shanath Arena.
- Molotov cocktail
- Pistol including Mark IV, M-712, 1911, 6P9, A.J.M. 9, D50 and .44 Magnum
- Rocket launcher including M79, RPG-7, GL-94, GL-A87 and LK-1018
- Shotgun including D2, M133, 1887 and SPAS-12
- Submachine Guns including A2000, MP5, A99, Storpio, Vector .45 ACP, BZ19
- Sniper rifle including SVD, M-700, SA-50, Z93

Others game variables related to killing are :

- Alert : The number of kills made with AI in alert detection state.
- Animals : The number of animals killed by the player.
- Cautious : The number of kills made with AI in cautious detection state.
- Combat : The number of kills made with AI in combat detection state.
- Distance Kills : The number of kills made over 40 meters.
- Close Kills : The number of kills made under 40 meters.
- Headshot : The number of Headshot kills.
- Idle : The number of kills made with AI in idle (undetected) state.
- Silencer : The number of kills with a silencer equipped.
- TagCamera : The number of kills with enemy previously tagged with camera

### C. Visualizing time varying information

As metrics change over time for a given player, it is not appropriate to summarize a given covariate in terms of mean or standard deviation. In this case it is more informative to show how a given metric changes over time. We can represent

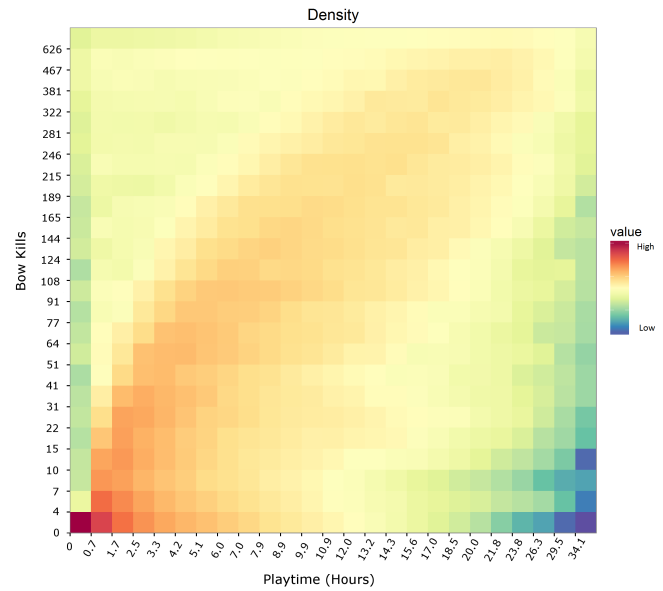


Fig. 3. Density of number of kill using bow over time

the time variation of a metric as a heatmap of players density (fig.3).

As different users can have a different tracking frequency in database, we have to be cautious. Each square of the heat map represents the number of player having the corresponding covariate value at the corresponding time. Thus we need to ensure that we take into account only one observation by player, if he is still playing at that time. By doing so, we mix players together and thus we are losing individual time-series, but it gives us a good way to see the main path followed by players.

Moreover, such plots are a good way to visually detect outliers. For instance, yellow squares at the top left of fig.3 show that some users have made hundreds of kills using a bow in less than a hour. Those are outliers that need to be removed before further analysis.

## VI. ANALYSIS

### A. Retention and Quit Rate

We see in fig. 4 that Far Cry 4 quit rate is mainly composed of two peaks : one between 1 and 10 hours of playtime, and one between 30 and 60 hours of playtime. The second peak corresponds to players having completed the game. However, game completion is a subjective notion. Different players need different playtime to complete the game not only because some of them reach a specific goal faster than the others but also because players have different play styles, and thus different goals [2]. Some players, mostly interested in the shooting gameplay, might spend a long time to complete all the side quests as long as they involve shooting on NPCs, while others may be mainly motivated by the main storyline and lose interest as soon as the main quest is complete.

The first peak between 1 and 10 hours corresponds to players who stop the game earlier. From such a high level



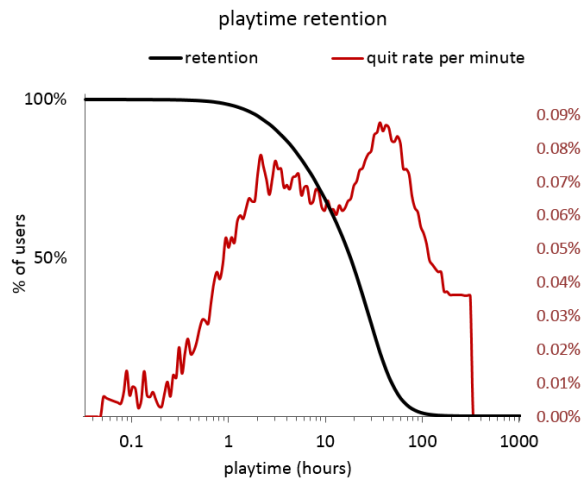


Fig. 4. Far Cry 4 retention in black and quit rate in red.

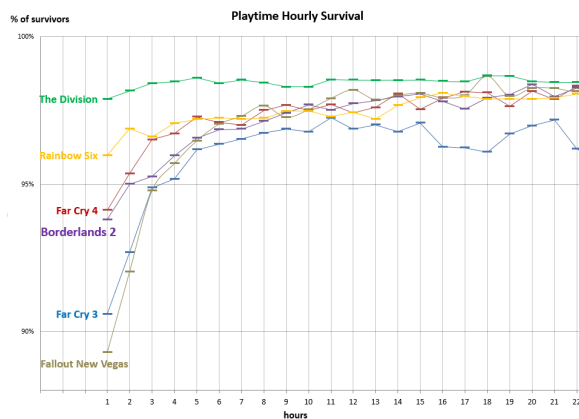


Fig. 5. Hourly survival curve of Far Cry 4 and some related games using a linear scale.

point of view, it is very hard to know why these players quit the game between 1 and 10 hours. But as we know that the quit rate is quite high it might be interesting to investigate more thoroughly in these time ranges, and realize a more detailed survival analysis.

### B. Benchmark

We realized a benchmark using SteamSpy data and compared the hourly survival curves as can be seen in (fig.5)

Far Cry 4 has a better hourly survival than Far Cry 3. That's true in the long term as in the first hours of the game. The biggest progress have been made in the first two hours of the game.

Borderlands 2 has quite similar hourly survival that Far Cry 4, but more players quit the game between two and five hours.

Tom Clancy's Rainbow Six® : Siege is also a first person shooter but the gameplay is more multiplayer oriented than Far Cry series. Even though Tom Clancy's Rainbow Six : Siege has almost the same long term hourly survival as Far Cry 4, we can see that Rainbow Six has a better retention in the first two hours than Far Cry 4.

Tom Clancy's The Division is an open world third-person shooter video game, who also includes action-RPG elements. It globally has a much higher retention than Far Cry 4.

We do not see any particular inflexion in Far Cry 4 curves. All the curves globally share a similar shape, and the game is globally among the other similar titles released by Ubisoft. As a result, it is very hard to derive a specific design insight from this high level point of view. In the next section, we will thus focus on a more detailed analysis using survival analysis.

### C. Time varying Cox model

We run our model on Weapon-related metrics. Estimated coefficients are plot in fig. 6 and 7. Recall that a positive coefficient means a positive correlation with player departure. As we prefer to talk about variable effect on retention, we should take the coefficient opposite value.

Three weapon usages have a highly positive link with player retention : Machete, Rocket Launcher and Shotgun. The user can only use the machete on the arena so this weapon usage gives us another information about a specific aspect of the gameplay. Playing arena mode allows the player to unlock some exclusive weapons. Players who invest time to unlock new weapons are likely to continue playing for a while.

Rocket and shotgun usage can be related to rough play. It seems that it is a well appreciated play style since players who do many kills with those weapons tend to play longer than others. On the other hand, some weapons related to stealth gameplay, like silencer and sniper rifle, tend to have no positive link on player survival.

Few covariates are still significant on long term - after 20 hours of playtime - except the number of kills made by grenade or machete. Part of this effect is due to the fact that as playtime increases the number of remaining player in the game decreases and hence confidence interval increases.

Auto crossbow is the only one that has a negative link with retention. Note that it has no effect before 10 hours of playtime because it is in average the time needed by players to unlock the weapon. The reason of this negative link could be due to multiple factors. It could thus interesting to perform a playtest in order to find why players who kill lots of enemies using auto crossbow are more likely to stop playing than others.

## VII. CONCLUSION

We've proposed a method to analyze the influence of design and players behavior on retention. First, we have defined player retention and have shown how we could use it to derive quit rate and hourly survival. Quit rate is very useful to detect design issues that are well localized in time, as a peak in the curve describes a sudden loss of players. Hourly survival is more adapted to data set where playtime measures are less accurate, as it is the case, for instance, with data extracted from SteamSpy website. Second, we have proposed a model based on survival analysis to evaluate the link between game variables and player retention, that extends Cox model to deal with time varying game variables and huge datasets.



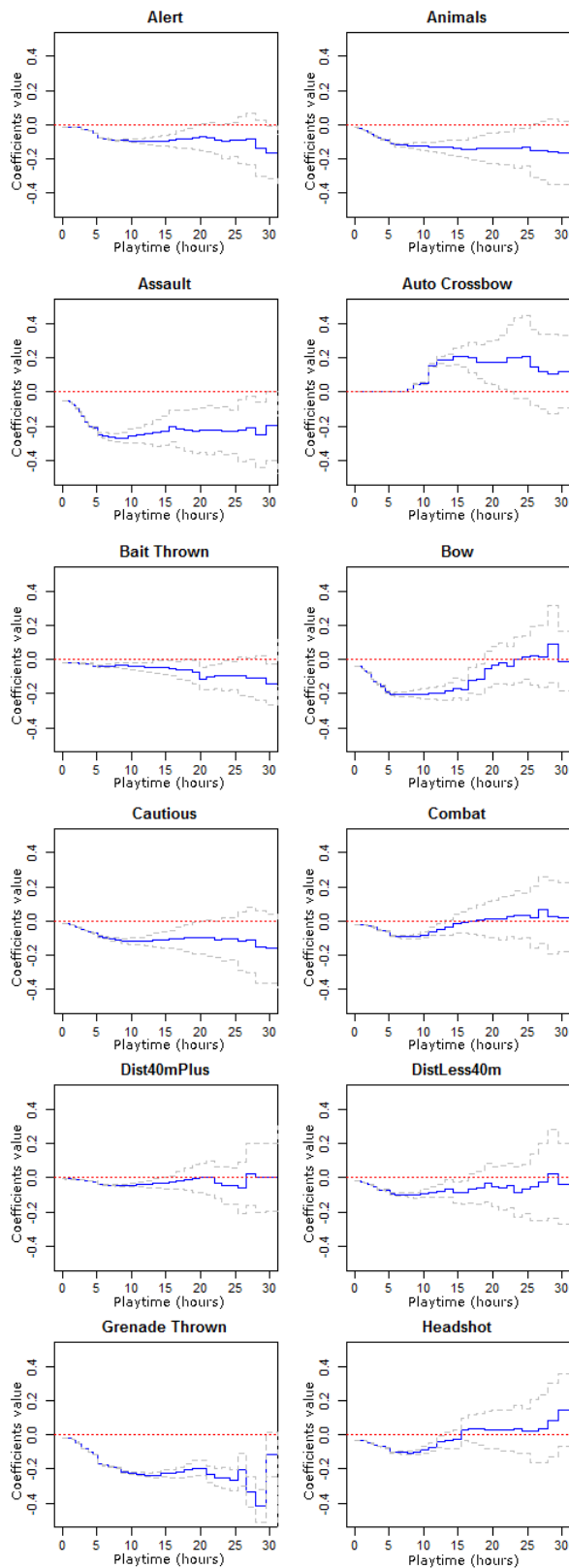


Fig. 6. Time varying coefficients (blue line) estimated on Far Cry 4 weapon usage dataset. Dashed grey line represent 95 % confidence intervals. Red dotted line at zero is the reference for null effect.

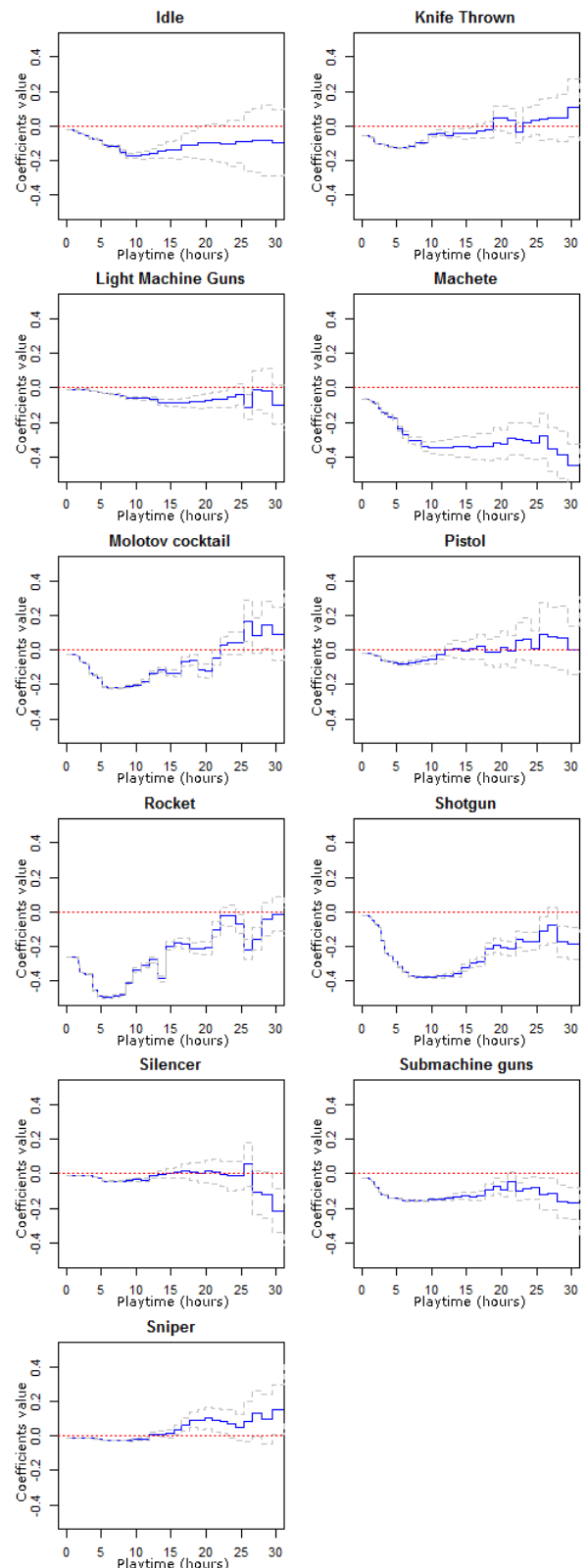


Fig. 7. Time varying coefficients (blue line) estimated on Far Cry 4 weapon usage dataset. Dashed grey line represent 95 % confidence intervals. Red dotted line at zero is the reference for null effect.

Then we used the proposed method to analyze Far Cry 4. First, we analyzed the Quit Rate curve, and found out that there was a first quit rate peak in the early phases of the game, between 1 and 10 hours of play, which might be worth of a more thorough investigation from the development team. Then, the Hourly Survival curve did not reveal any specific issue. This curve is the most easy to compute but also the most coarse grained metric.

We then investigated the game more thoroughly with regard to weapon related game variables. We selected these variable as they concern Far Cry 4 core gameplay. From these data, we found out that many weapons had a positive link with player retention, and only one weapon were found as having a negative link with retention. Globally, it seems that weapons related with stealth gameplay tend to have a less positive link with retention.

We've shown that our approach works well with low level data, like weapon usage. But to get more influent design feedback, we need to introduce higher level metrics.

Difficulty is one of the main factor of motivation [13]. We plan to investigate whether an objective measure of difficulty, as the one proposed by Levieux [11], might have a strong link with player retention.

Another research area that might be very interesting to investigate would be to transform a survey-based theory like Self-Determination Theory [16] into gameplay metrics. Having metrics related to autonomy, competence and relatedness might give interesting insight on player behaviors.

## REFERENCES

- [1] M. Z. Alaya, T. Allart, A. Guilloux, and S. Lemler. Time-varying high-dimensional aalen and cox models with change-points. *preprint*, 2016.
- [2] Richard Bartle. Hearts, clubs, diamonds, spades: Players who suit muds. *Journal of MUD research*, 1(1):19, 1996.
- [3] Christian Bauckhage, Kristian Kersting, Rafet Sifa, Christian Thureau, Anders Drachen, and Alessandro Canossa. How players lose interest in playing a game: An empirical study based on distributions of total playing times. In *Computational Intelligence and Games (CIG), 2012 IEEE conference on*, pages 139–146. IEEE, 2012.
- [4] Brian Bowman, Niklas Elmqvist, and TJ Jankun-Kelly. Toward visualization for games: Theory, design space, and patterns. *IEEE transactions on visualization and computer graphics*, 18(11):1956–1968, 2012.
- [5] Kuan-Ta Chen, Polly Huang, and Chin-Laung Lei. Effect of network quality on player departure behavior in online games. *Parallel and Distributed Systems, IEEE Transactions on*, 20(5):593–606, 2009.
- [6] D. R. Cox. Regression models and life-tables. *Journal of the Royal Statistical Society. Series B (Methodological)*, 34(2):pp. 187–220, 1972.
- [7] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.
- [8] Dirk Eddelbuettel and Romain François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011.
- [9] Sergey Galyonkin. Steamspy, 2016.
- [10] Brent Edward Harrison and David Roberts. Analytics-driven dynamic game adaption for player retention in a 2-dimensional adventure game. In *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2014.
- [11] Guillaume Levieux. *Mesure de la difficulté des jeux video*. PhD thesis, Conservatoire national des arts et metiers-CNAM, 2011.
- [12] Tobias Mahlmann, Anders Drachen, Julian Togelius, Alessandro Canossa, and Georgios N Yannakakis. Predicting player behavior in tomb raider: Underworld. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 178–185. IEEE, 2010.
- [13] Thomas W Malone. Toward a theory of intrinsically motivating instruction\*. *Cognitive science*, 5(4):333–369, 1981.
- [14] T. Martinussen, T. H. Scheike, Ib M. Skovgaard, and T. Matinerrsen. Efficient estimation of fixed and time-varying covariate effects in multiplicative intensity models. *Scandinavian Journal of Statistics*, 29(1):pp. 57–74, 2002.
- [15] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016.
- [16] Richard M Ryan and Edward L Deci. Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *American psychologist*, 55(1):68, 2000.
- [17] Katie Salen and Eric Zimmerman. *Rules of Play : Game Design Fundamentals*. The MIT Press, October 2003.
- [18] G Wallner and S Kriglstein. Visualization-based analysis of gameplay data—a review of literature. *Entertainment Computing*, 4(3):143–155, 2013.
- [19] Ben George Weber, Michael John, Michael Mateas, and Arnav Jhala. Modeling player retention in madden nfl 11. In *IAAI*, 2011.
- [20] Hanling Xie, Sam Devlin, Daniel Kudenko, and Peter Cowling. Predicting player disengagement and first purchase with event-frequency based data representation. In *Computational Intelligence and Games (CIG), 2015 IEEE Conference on*, pages 230–237. IEEE, 2015.
- [21] Nick Yee. Motivations for play in online games. *CyberPsychology & behavior*, 9(6):772–775, 2006.
- [22] M. D. Zeiler. Adadelta: An adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

# Guns and Guardians: Comparative Cluster Analysis and Behavioral Profiling in *Destiny*

Anders Drachen<sup>1</sup>, James Green<sup>2</sup>, Chester Gray<sup>3</sup>, Elie Harik<sup>4</sup>, Patty Lu<sup>5</sup>, Rafet Sifa<sup>6</sup>, Diego Klabjan<sup>7</sup>

<sup>1</sup>Aalborg University, Denmark, Email: andersdrachen@gmail.com

<sup>2,3,4,5,7</sup>Northwestern University, USA, Email: (jamesgreen2016;chestergray2016;elieharik2016;peiruolu2016,dklabjan)@u.northwestern.edu

<sup>6</sup>Fraunhofer IAIS, Germany, Email: rafet.sifa@iais.fraunhofer.de

**Abstract**—Behavioral profiling in digital games with persistent online worlds are vital for a variety of tasks ranging from understanding the player community to informing design and business decisions. In this paper behavioral profiles are developed for the online multiplayer shooter/role-playing game *Destiny*, the most expensive game to be launched to date and a unique hybrid incorporating designs from multiple traditional genres. The profiles are based on playstyle features covering a total of 41 features and over 4,800 randomly selected players at the highest level in the game. Four clustering models were applied (k-means, Gaussian mixture models, k-maxoids and Archetype Analysis) across the two primary game modes in *Destiny*: Player-versus-Player and Player-versus-Environment. The performance of each model is described and cross-model analysis is used to identify four to five distinct playstyles across each method, using a variety of similarity metrics. Discussion on which model to use in different circumstances is provided. The profiles are translated into design language and the insights they provide into the behavior of *Destiny*'s player base described.

## I. INTRODUCTION

The analysis of player behavior in digital games has, with the introduction of telemetry tracking and game analytics, become a cornerstone of game development. Behavioral analysis assists across all phases of a development cycle as well as after launch, and can help with a variety of tasks from balancing, experience evaluation, cheat detection, prediction, monetization and debugging [7]. One of the key behavioral analyses that has emerged in the nascent domain of game analytics in the past five years is behavioral profiling [6], [1].

Behavioral profiling is a technique known from a variety of data and information science domains, including web analytics and finance, and serves as a means for considering users or consumers in a non-abstract and quantifiable way. Behavioral profiling in digital games seeks to condense the often high-dimensional, high-volume and volatile behavioral datasets generated, notably typical for major commercial (AAA) titles, into a subset of well-described profiles that encapsulate player behavior and informs game developers and researchers about how people are playing the game under investigation. This location of patterns in the behavior of players is a major challenge in game analytics, as well as the construction of actionable models based on patterns.

Behavioral profiling in games has previously been performed using a variety of descriptive and statistical tech-

niques, as well as machine learning, the latter with an emphasis on unsupervised techniques due to the common explorative goal of profiling in games [8], [2], [22].

One of the most popular approaches has been clustering [10], [6]. Cluster models allow segments to be developed which can describe the behavior of players according to specific behaviors and are driven by specific research questions. For example, discovering major playstyles in a game. Clusters can be translated into descriptions of the different player segments, and the information contained within can inform the game design and optimization [1], [11]. However, there are a great number of cluster models available, each with specific strength and weaknesses, and it can be challenging to assess which algorithm to employ in a given situation.

The analysis of player behavior is important to any digital game but is notably important in persistent online games, as the success of these games rests on their ability to keep a player community engaged over extended periods. It is increasingly common for games to feature persistent gameplay, as it enables the application of sources of revenue not available under a retail-based model [12]. This is also the case for *Destiny*, a hybrid online game that combines elements from a number of game genres, with a primary emphasis on online shooter-type gameplay. The game was released in 2014 as a console-exclusive title. Players take on the role of a guardian, a warrior who works to fend off alien forces attacking Earth. In this paper four different cluster models are applied to detailed behavioral telemetry from *Destiny*, focusing on player performance, for the purpose of developing behavioral profiles for the game, as well as for evaluating the models themselves.

## II. CONTRIBUTION

In this paper, behavioral profiles are presented for the online multiplayer shooter game *Destiny*. The game forms a hybrid between a shooter game and Massively Multiplayer Online Game (MMOG), and thus presents a game format that has not previously been explored in game analytics. Behavioral profiles are developed based on cluster analysis for a subset of 41 behavioral features engineered from a dataset of over 1,400, focusing on performance and playstyle measures but including others. Profiles are developed for both of the two primary game modes: Player vs. Player

(PvP) and Player vs. Environment (PvE), and include more than 4,800 players, all randomly sampled from the pool of players at the maximum character level. Four clustering models are applied to the data: Archetype Analysis (AA), k-means, k-maxoids, and Gaussian mixture models (GMM), each resulting in prototype players or centroids that are representative of different clusters of behavior.

The results of each model are described and compared, and the strengths and weaknesses of each method described. This informs future work in industry that seeks to build behavioral profiles in games, and highlights the importance of balancing feature selection, choice of model and the interpretability and actionability of the resulting behavioral profiles. Feature engineering in an ultra-high dimensionality situation like *Destiny* is also discussed. Finally, the behavioral profiles developed are described in terms of design language, following the principles of Drachen et al. [8], and their insights into player behavior in *Destiny* discussed.

### III. DESTINY: GAMEPLAY

*Destiny* is a science-fiction themed game where players take on the role as Guardians, who defend the Earth in a future where there is only one safe city left, and the human race is threatened by alien powers. Players protect the city from these alien races, and are tasked with the overall goal of reviving a Moon-sized being called the Traveler, who protected the human civilization but now lays dormant. To do this the players must explore a variety of planets, complete various missions and help eliminate the alien threat.

*Destiny* is a hybrid digital game that blends features from a number of traditional game genres. *Destiny* is first and foremost an online first-person shooter (FPS) game, and the majority of the gameplay is focused on using any of thousands of different weapons to eliminate either computer-controlled or human opponents. Other examples of online FPS games include *Planetside 2*. As an FPS, *Destiny* features both single-player and multiplayer game modes, as well as both Player-vs-Environment (PvE) and Player-vs-Player (PvP) combat. PvE mode includes a variety of story missions, usually given by NPCs in the game's central quest hub "Tower", but also includes public events, raids and strikes, which require three or six players respectively to collaborate in "fireteams" of 3. When playing in fireteams, players can put themselves at risk and revive a teammate.

In terms of the MMOG elements, *Destiny* features a persistent, shared world in which the players interact with each other and Non-Player Characters (NPCs). The game has its own currencies, factions players can build reputation with, and a comprehensive item customization system, all common features in MMOGs. The game has missions/quests, delivered by NPCs, which covers a range of different activities. The game also features social interaction, although the communication options are more restrictive in *Destiny* as compared to MMOGs. As a Role-Playing Game (RPG), *Destiny* features character classes (Hunter, Warlock and Titan, each with three subclasses), experience points, character development and unlocking of more powerful items and

abilities as players progress from level 1 to 40, which is the current level cap. As a Multiplayer Online Battle Arena (MOBA) game, *Destiny* features team-based combat within arena-type environments, similar to e.g. *League of Legends*.

PvP play is done via the *Crucible*, the central hub for this game mode. There are a variety of different specific modes, including traditional deathmatch modes, take-and-control modes, and more. In PvP players can earn medals, points and in-game currency by accomplishing specific tasks or feats of skill. For example, a "First Blood" medal awards extra points to the player with the initiative to get the first kill in a match. In both PvE and PvP game modes, players are rewarded with new weapons and items through random drops or by completing specific tasks.

Another key aspect of *Destiny* in both the PvE and PvP modes is weaponry. *Destiny* features hundreds of different weapons (ranged and melee), which can fire a variety of energy types or projectiles, and be customized in innumerable ways. Together with armor and ships, guns are perhaps the focal point of development in the game. Weapons are divided into over a dozen different types or classes, each specialized for specific situations. For example, shotguns excel in close-quarters combat, dealing large amounts of damage with little need to precisely aim. Conversely, sniper rifles offer similar amounts of power but are rendered near useless when the target gets too close. Even at the optimal distance, however, they require skilled players who can aim precisely. Between game modes, the utility of these weapon types also varies; shotguns are typically a one-shot kill in PvP modes, but must be used with care in PvE modes, as enemies with large health pools can quickly strike back with devastating close range attacks. Players are given freedom to switch between any combination of weapon types, allowing for adjustment to in-game scenarios while at the same time reflecting individual behavior and preferences.

### IV. RELATED WORK

There are a number of challenges associated with behavioral telemetry data in digital games, notably that they are commonly large-scale, high-dimensional and volatile [6], [1]. This is also the case for *Destiny*, and is exemplified in the current dataset which contains over 1,400 baseline features, based on just one of a dozen JSON collections. While only a subset of the population is used here due to data size constraints, *Destiny* has dozens of millions of active players, requiring the adoption of random selection when defining samples. Given the constant changes in the design of the game via new content and tweaks to the mechanics, as well as the running turnover in the population of the players, any profiles generated will have a limited lifetime during which they are accurate representations of the underlying player base. Pattern recognition under the conditions of contemporary commercial game development can thus be difficult, but also potentially highly rewarding because such patterns directly inform the game development process, can be used for Game AI related purposes, or to personalize or adapt gameplay, assist matchmaking, and

identify valuable players [6], [8], [15], [1], [16], [17]. Cluster analysis [10] is one of the primary tools available for pattern recognition and has been readily applied across disciplines, and even in recent years adopted in game analytics for the purpose of finding patterns in the behavior of players. As an unsupervised method, it permits the exploration of data and can identify groups of players with similar behaviors or detect the features that constitute such behaviors [6], [8], [15], [1], [16], [17].

The popularity of cluster models in explorative evaluation of behavior is in part driven by the wide variety of models, which can be applied to reach specific outcomes, e.g. searching for extreme or central tendencies in the data [1], [6], [10].

The majority of previous work on behavioral profiling in games has focused on employing specific methods, with the only work specifically comparing multiple cluster models being Drachen et al. [2] who applied k-means, c-means, Non-negative Matrix Factorization (NMF), Archetype Analysis (AA) and Principal Components Analysis (PCA) for a dataset covering player character progression in *World of Warcraft*, and noted the different output produced by these models but did not discuss cross-model analysis. Bauckhage et al. [6] showed examples of multiple models and advised on their application. This paper directly extends on this previous work by comparing four cluster models and specifically targeting the problem of making decisions across models.

Behavioral profiling via clustering, and related methods, has been performed using a variety of models (the following are more or less in order of publication): The first paper to target the profiling problem was Drachen et al. [8], who used Self-Organizing Networks in combination with data from 1,365 players of *Tomb Raider: Underworld*. The authors documented that over 95% of the players could be allocated to one of four behavioral profiles. In a follow-up piece, by Sifa et al. [11] who explored how player profiles varied as a function of progress in the same game.

Shim and Srivastava [16] used segmentation and descriptive methods to examine the behaviors of *EverQuest II* players, focusing on behavioral profiling and efficiency in player behaviors. Thureau and Bauckhage [18] explored the evolution of guilds in *World of Warcraft*, across 18 million characters using matrix factorization. Thawonmas and Iizuka [17] used multidimensional scaling (CMDS) and KeyGraph to generate visualizations of player clusters working with the MMOG *Shen Zhou Online*. Drachen et al. [1] employed Simplex Volume Maximization (SIVM) and k-means on datasets from the MMOG *Tera: Online* and the team-based shooter *Battlefield 2*, developing behavioral profiles for these two games based on a range of behavioral features. Sifa et al. [19] analyzed more than 3,000 games and over 6 million players from the distribution platform Steam to investigate playtime patterns, and developed clusters of games via Weibull modeling. The work was followed up by Sifa et al. [20] who identified 11 clusters of players based on their relative playtime distribution across games on Steam.

Bauckhage et al. [21] adopted DEDICOM (Decomposition

into Directional Components) to cluster players of *Quake: Arena* and develop waypoint graphs for behavior-based partitioning. Normoyle and Jensen [22] used Bayesian Clustering on data from the multiplayer shooter game *Battlefield 3* covering over 500,000 matches. In the multiplayer esports game *DOTA 2*, Drachen et al. [24] clustered players according to spatio-temporal behavior and skill, using distance measures and k-means. Finally, Sifa et al. [23] adopted DEDICOM to investigate player churn behavior among multiple games.

## V. DATA AND PRE-PROCESSING

For this study, two distinct sets of data from the game *Destiny* were used: PvP, and PvE. The data was provided as a large JSON object, which was parsed and converted into a flat comma delimited file. The data is aggregated and exists on a static level, meaning for each character there is a slice of data only at the point in time when the data was pulled. If the character has progressed since (e.g. changed weaponry, made more kills, leveled up), the information is not reflected in the data. Similarly, the characters details when they started the game (e.g. weapon equipped in the first five minutes, number of kills in the first five minutes, level during the first five minutes), is also not reflected in the data.

### A. PvE Dataset

- Consists of 1,217 variables detailing encounters by 27,967 player characters in the game.
- Divided into five categories:
- Basic character information containing the account ID and character ID field, and the deleted flag, showing whether a character has been deleted or not. The information covers virtually any aspect of player behavior across performance, engagement, progression, etc.
- Game Progress variables detailing how many activities the character has participated in and completed.
- Personal bests of kills and deaths.
- Average statistics of kills and deaths.
- Total counts of kills and deaths.

### B. PvP Dataset

- Consists of 211 variables detailing encounters by 16,422 characters in the game.
- Divided into five categories:
- Basic Character information containing Account ID and Character ID, and the deleted flag, showing whether a character has been deleted or not.
- Game Progress variables detailing how many activities the character has participated in and completed.
- Personal bests of kills, deaths, and medals earned.
- Average statistics of kills, deaths, and medals earned.
- Total counts of various kills, deaths, and medals earned.

### C. Behavioral Features and Selection

Clustering analysis relies on being able to classify players into groups based on features. Conversely, the features used for clustering should easily explain the groupings that are found [8]. To this end, it is important to filter out highly specific, correlated, redundant, and dependent variables, which

may create noise and obfuscate the end results. In order to keep the clusters decipherable, feature selection, with an emphasis on finding the subset of variables that explain overall variation in the dataset is necessary. Feature selection was done based on classifying the available high-level variables into three categories: performance, progress, and playstyle. Here the focus is on developing high-level profiles and thus the playstyle features covering *Destiny's* main mechanics were used. The initial selection process yielded 41 features from the original over 1,400. Initially, about 90% of the PvE variables were removed due to redundancy. This paper focuses on high-level behaviors, so highly specified features are excluded, as the information is better contained and summarized by higher level aggregate features. For example, the original data kept track of how many times a player was killed by a specific enemy in a particular encounter of one mission in the game. Additionally, many attributes depended directly on a players time spent playing the game. To mitigate this effect, a subset of characters that had already reached the games level cap of 40 were considered in the analysis (4,800 players) (Fig. 1).

Features	PvP Mean	PvP St. Dev.	PvE Mean	PvE St. Dev.
Average Kill Distance	13.80	3.66	18.15	3.23
Proportion of Kills using Auto Rifle	12.25%	13.25%	16.1%	11.0%
Proportion of Kills using Fusion	4.57%	7.38%	3.6%	3.0%
Proportion of Kills using Grenade	6.49%	3.82%	7.1%	2.5%
Proportion of Kills using Hand Cannon	9.27%	9.86%	12.0%	9.3%
Proportion of Kills using Machine Gun	3.19%	3.30%	1.7%	1.2%
Proportion of Kills using Melee	11.40%	7.49%	12.5%	4.8%
Proportion of Kills using Pulse Rifle	7.44%	9.02%	9.4%	7.6%
Proportion of Kills using Rocket Launcher	2.73%	2.54%	3.4%	2.4%
Proportion of Kills using Scout Rifle	4.91%	7.92%	15.1%	10.4%
Proportion of Kills using Shotgun	12.25%	9.51%	3.6%	2.9%
Proportion of Kills using Side Arm	0.70%	2.64%	1.0%	1.4%
Proportion of Kills using Sniper	4.07%	5.24%	4.5%	2.8%
Proportion of Kills using Super	11.83%	6.48%	4.9%	2.7%
Proportion of Kills using Sword	1.01%	3.02%	1.3%	1.9%
Proportion of Kills using Relic	N/A	N/A	1.3%	1.3%
Proportion of Games earning First Blood	1.70%	1.28%	N/A	N/A
Proportion Offensive Kills	4.57%	2.97%	N/A	N/A
Proportion Defensive Kills	4.07%	2.55%	N/A	N/A
Avg Time Remaining after Quitting (sec)	33.50	320.76	N/A	N/A
Proportion of Games played in PvP	18.77%	14.06%	N/A	N/A
Max Proportion of Kills with a Weapon	N/A	N/A	26.4%	9.1%
Orbs Dropped Per Sec	N/A	N/A	0.01	0.002
Resurrections Performed / Received Ratio	N/A	N/A	1.1	0.5

Fig. 1. Mean and standard deviation of the primary features in the PvP and PvE datasets from *Destiny*.

An exploratory data analysis was performed on the remaining features, including histograms and correlation analysis. Correlation analysis revealed that many attributes exhibited large correlations with play time. For example, the raw number of recorded kills per weapon understandably increased over time. Weapon kills were time normalized after being converted to a proportion of total kills. Other variables that had no associated total amount with which to calculate a proportion were converted to a rate by dividing by the number of seconds played. Histogram plots revealed significant right skew in many of the attributes. To account for this, a logarithmic transformation was applied. Some skew still remained after applying these transformations, so zero mean standardization was applied to ensure that all variables were on the same scale. After correcting for the

forementioned skewness and time dependencies, the final selection of variables involved creating a set of attributes designed for all of the main game mechanics in either mode.

## VI. CLUSTER MODELS

Four different clustering methods were applied to the data. Each clustering methodology had a variety of parameters to choose from, such as number of clusters or archetypes, and cluster shapes. To account for the inherent differences in how the methods generate clusters, different evaluation metrics were chosen for each method to determine the optimal number of clusters. The optimal cluster solutions for each method were compared using adjusted mutual information. The four models are as follows:

### A. K-means Clustering

The k-means algorithms partitions the data in k different clusters such that all points in a given cluster are closest to the corresponding cluster center. K-means is a common clustering method in game analytics, for example in Drachen et al. [1] and Drachen et al. [2]. While more computationally efficient than the algorithms mentioned below, k-means focuses on the average behavior of players and does not identify more extreme behavior accurately. Additionally, resulting k-means clusters must be spherical in shape and the algorithm is biased to equal-sized clusters. However, the algorithm can still be used to observe and cluster the general average behavior of players in *Destiny*, serving as a baseline to support the findings of other methods, such as Archetype Analysis, that are designed to identify more extreme behavior. The k-means function in R was used to cluster PvP and PvE players based on the various playstyle metrics. In order to select the best number of clusters, a natural grouping was assumed to be homogenous within and heterogenous across, i.e. the solution should have a high between-cluster variance and a low within-cluster variance. The best solution, in terms of interpretability and using the aforementioned metric, resulted in a four cluster solution.

### B. Gaussian Mixture Models

Gaussian mixture models are conceptually related to k-means clustering, with a few distinct differences. First and foremost, k-means clustering assumes that each cluster is approximately equally sized and distributed—that is, there will be an approximately equal number of data points within each cluster, and each cluster will appear to be a sphere of equal proportions to the other clusters. Gaussian Mixture Models, on the other hand, relax this constraint; the user has control over whether to enforce equal sizes of clusters as well as whether or not to assume a spherical shape, as opposed to an ellipsoidal shape. When using ellipsoidal clusters, the ellipsoids can either all be oriented in the same direction, or always oriented along the coordinate axes; or, if necessary, the orientation can be in any direction. This allows for more flexible definitions of clusters. For more information on the math behind Gaussian mixture models, see [14].



For the purpose of this analysis, the R statistical software package Mclust was used thanks to its flexibility and available features [13]. Mclust was able to return the results of all possible model shapes as well as a range from one cluster to ten clusters. Based on the Bayesian Information Criterion (BIC), Mclust selects the best shape and number. The best result based on this criterion used four ellipsoidal, equally-sized, and equally-oriented clusters for PvP, and six ellipsoidal, equally-sized, and equally-oriented clusters for PvE. While Gaussian mixture models have not been known to be used in previous work in game analytics, it was included in the analysis for its similarity to k-means, and to provide an opportunity to examine the similarities between two centroid-based methods.

### C. Archetype Analysis

Archetypal Analysis seeks to identify points whose convex combinations can generally represent the population of the dataset. These archetype points are not necessarily observed, but exist as manifestations of extreme behavioral qualities. This means that archetypes typically exhibit more radical values than the typical observation. Each observed data point is then classified to its closest archetype, resulting in clusters. Archetype Analysis [4] has previously been used for game analytics by Sifa et al. [3]. In contrast to centroid-based clustering algorithms such as k-means and Gaussian mixture models, the clusters found using archetypal analysis are identified by prototypical points. This means that the contrast between various archetypes is magnified (more so than the mean of these clusters), as they represent more extreme values. Conversely, this means that the values that are less extreme get classified to these clusters. A Scree plot using residual sum of squares (RSS) found that the optimal data groupings were four clusters for PvP and five for PvE.

### D. K-Maxoids

Similar to archetype analysis, k-maxoids seeks to find cluster prototypes that represent the extremes of a data set rather than the modes. The maxoid of a set is defined as a data point that has the largest average distance from all other points in the same set. Bauckhage and Sifa [5] used the method in the context of game analytics to cluster players based on vehicle usage. Due to the extreme nature of the maxoids, the resulting cluster prototypes are generally more varied than those produced by centroid-seeking algorithms such as the Gaussian mixture models and k-means. To choose the optimal number of clusters, the average silhouette score was computed for cluster sizes ranging from three to eight. The four cluster and five cluster solutions were chosen as the best for PvE and PvP, respectively.

## VII. EVALUATING CLUSTER SIMILARITY

Each clustering method produced a somewhat different set of clustering classifications, which begs the question, how different are these clusters, and how does it affect the results? In order to understand how similar one method was to another, clustering results were compared using the Adjusted

Mutual Information (AMI) value [Vinh]. AMI ranges on a [-1,1] scale, where 1 is perfectly similar, 0 is no more similar than what would be expected in a random assignment, and -1 is perfectly dissimilar (less similar than a random clustering). Fig. 2 shows the AMI for each method, using cluster sizes of four, five, and six to account for different methods having different preferred numbers of clusters. This directly extends the comparative work of [2]

		<u>Gaussian Mixture Models</u>			<u>K-Means Clustering</u>			<u>K-Maxoids</u>			<u>Archetype Analysis</u>			PvE Similarities
		4 Clusters	5 Clusters	6 Clusters	4 Clusters	5 Clusters	6 Clusters	4 Clusters	5 Clusters	6 Clusters	4 Clusters	5 Clusters	6 Clusters	
<u>Gaussian Mixture Models</u>	4 Clusters	1	0.6	0.6	0.2	0.2	0.2	0.1	0.1	0.1	0.1	0.1	0.1	
	5 Clusters	0.2	1	0.7	0.2	0.2	0.2	0.1	0.1	0.1	0.1	0.1	0.1	
	6 Clusters	0.4	0.3	1	0.2	0.2	0.2	0.0	0.1	0.1	0.1	0.1	0.1	
<u>K-Means Clustering</u>	4 Clusters	0.1	0.2	0.2	1	0.6	0.5	0.1	0.2	0.2	0.3	0.3	0.3	
	5 Clusters	0.1	0.2	0.2	0.6	1	0.8	0.1	0.1	0.3	0.3	0.4	0.3	
	6 Clusters	0.1	0.2	0.2	0.3	0.5	1	0.1	0.1	0.3	0.3	0.4	0.4	
<u>K-Maxoids</u>	4 Clusters	0.1	0.1	0.1	0.1	0.1	0.1	1	0.6	0.2	0.1	0.1	0.1	
	5 Clusters	0.1	0.1	0.1	0.2	0.2	0.2	0.2	1	0.2	0.1	0.1	0.1	
	6 Clusters	0.1	0.1	0.1	0.2	0.2	0.2	0.2	0.6	1	0.2	0.3	0.2	
<u>Archetype Analysis</u>	4 Clusters	0.1	0.1	0.1	0.5	0.5	0.3	0.1	0.3	0.3	1	0.3	0.3	
	5 Clusters	0.1	0.2	0.2	0.4	0.4	0.4	0.2	0.3	0.3	0.4	1	0.3	
	6 Clusters	0.1	0.1	0.1	0.3	0.3	0.3	0.1	0.2	0.2	0.4	0.2	1	
PvP Similarities														

Fig. 2. Adjusted Mutual Information for Various Clustering Results. The upper triangle (yellow) represents PvE clustering AMIs, while the lower triangle (green) represents PvP clustering Adjusted Mutual Information values (AMI).

The results were not as expected based on the models employed: the expectation was to see a high degree of similarity between GMM and k-means, since the two methods share a centroid-based approach to clustering. Furthermore it was expected that k-maxoids and Archetype Analysis would be similar due to their mutual reliance on extrema of the dataset. However, from the above chart, the clusters had AMIs that were, for the most part, only slightly above 0 (where 0 implies they were no more similar than a random assignment of classifications). The one exception was between Archetype Analysis and k-means, which can reach as high as 0.6 for PvE and 0.5 for PvE, which suggests the two methods were producing more consistent results than a random assignment would. The lack of similarity amongst either the centroid-based models or the extrema-based models, as well as the moderate similarity between Archetype Analysis and k-means, were contrary to our expectations. A comparison using the Jaccard similarity coefficient yielded comparable results, so only one set of values is included here.

These results suggest that, while no method is necessarily more powerful than any other method, the expectation that each method is interchangeable with any other method is incorrect. As such, it is important to look beyond the quantitative comparisons of each clustering result, and instead focus on the characteristics of the clusters within each result to see which methods produce the most interpretable clusters.

### A. Comparing Models Using Cluster Interpretability

The key in any clustering exercise is that, first and foremost, the clusters produced must be valuable to the recipient of the analysis. Clusters may appear in the data but if there are no actionable insights to be gained, the knowledge of those clusters are unhelpful. Because the clustering methods so far have been shown to have low similarity, the expectation was that profiling those clusters (that is, looking at the defining characteristics of each cluster to create a more general set of terminologies by which to consider those clusters) would also return sets of profiles that are different from method to method. Table 1 shows an example heatmap used to identify clusters with Archetype Analysis.

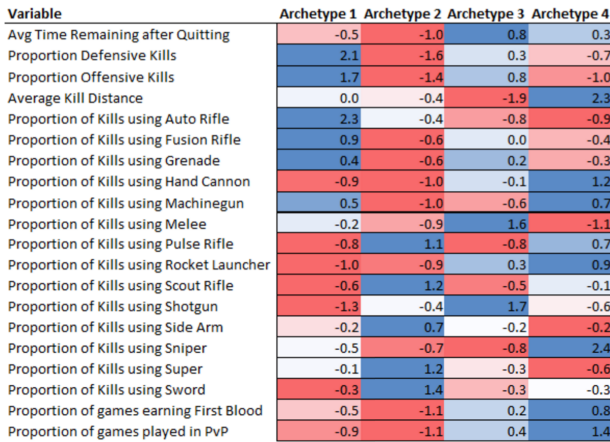


Fig. 3. Example heatmap used to identify clusters with Archetype Analysis. Each archetype has different prototypical values associated with each variable, which were used to differentiate and profile the archetypes. Red values indicate values below the mean, and blue values indicate values above the mean for each variable (the strength of the hue helps to show which clusters have the most extreme values for each parameter).

Each method, having produced a single best set of clusters based on the aforementioned criteria for cluster selection, was subjected to an exercise by which a title was assigned to each cluster based on its profile. Tables 1,2 show two sample results of for PvP, and tables 3,4 for PvE.

### B. PvP

For the PvP dataset, the two largest clusters identified by Archetype Analysis, using a four-clusters solution as shown in Table 1, are groups that are characterized by their use of specialized weapons: *Aggressive Close Range* players make effective use of Shotguns and complement their favorite weapon with melee blows in order to dismantle PvP opponents. The second largest group, *Marksmen*, make use of sniper rifles to take out opponents at a long range and make smart use of hand cannons to take out enemies that are at a close to medium range. *Objective Killers* are players that play a majority of Control games where the match consists of holding various bases, defending them and attacking enemy-controlled bases. Finally, *Casual PvPers* are individuals that play a higher proportion of PvE that are not focused on using any specific weapon type. Because these archetypes

are defined using the extreme values in the dataset, the differences between cluster values tend to be more dramatic. These more pronounced differences lead to clusters that are more distinguishable from one another.

TABLE I  
PvP PROFILES FOR ARCHETYPE ANALYSIS. %P = % OF PLAYERS

Title	%P	Characteristics
Objective Killers	20.1	Highest scores for proportion of offensive & defensive kills
Casual PvPer	15.9	Does not appear to play PvP much
Aggressive Close Range	35.1	Lowest average kill distance, highest melee and Shotgun kills
Marksmen	28.9	Highest average kill distance, highest Hand Cannon and Sniper usage; plays PvP the most

Table 2 shows the clusters identified by Gaussian mixture models, using a 4 cluster solution. However, only the long-range hardcore PvP cluster can be readily interpreted. The short-range kill distance with long-range weapons used cluster is counter-intuitive. There does not seem to be a logical explanation for why 14.2% of players prefer weapons intended for medium-long range to kill opponents at short range, but it may be an effect of less experienced players not yet proficient with weapon switching, possibly being recent lvl 40 characters. The two last clusters include almost 70% of all players included in the clustering analysis and represent what would be a balanced playstyle. Gaussian mixture models do not perform well here because the method identifies and groups players based on the average behavior of all individuals, a pitfall shared by k-means. The cluster results exist closer to each other at the center of the data, leading to prototypes with less pronounced differences.

TABLE II  
PvP PROFILES FOR GAUSSIAN MIXTURE MODELS. %P = % OF PLAYERS

Title	%P	Characteristics
Short-range with long-range mix	14.2	Lowest average kill distance, but with lower than average use of all typically short-range weapons, and higher than average use of scout rifle (the second-longest range weapon)
Long-range hardcore PvP	16.1	Higher than average fusion rifle usage but otherwise unremarkable; slightly more PvP play than average
Balanced I	37.4	Slightly stronger focus on PvP, with a preference for long-range weapons but also shotguns
Balanced II	28.9	Lower than average sidearm and sword usage, with slightly low PvP playtime, but otherwise unremarkable

### C. PvE

Clusters found by Archetypal Analysis in Table 3 were roughly equal in size and were the most readily interpretable in terms of in-game behavior. The archetypes are: *High DPS*, *Guerilla Warriors*, *Close Combatants*, *Sitting Duck*

*Snipers* and *Mobile Marksmen*. These groupings are primarily defined by damage output, distance from enemies and weaponry used. The largest cluster of players focused on using weapons with a high damage per second (DPS) output, and are named *High DPS*. Solely distance-based metrics defined the *Close Combatants*, who focus on a variety of close combat weaponry and have the shortest average kill distance. Solely variety of weaponry metrics defined the *Guerilla Warriors*, who are a family of players who are highly adaptable to changing situations, and have the highest variety of weaponry used. A combination of distance-based metrics and variety of weaponry metrics established the *Sitting Duck Snipers* and *Mobile Marksmen*. *Sitting Duck Snipers* are a group who prefer to shoot from a single location, utilizing snipers at first and switching weaponry as enemies come closer. In contrast, *Mobile Marksmen* are players who prefer to stick to a single weapon and move themselves as enemies get closer.

TABLE III

PvE PROFILES FOR ARCHETYPE ANALYSIS. %P = % OF PLAYERS

Title	%P	Characteristics
High DPS	23.5	Players who appear to focus on high DPS moves (such as specials)
Guerilla Warriors	16.7	Players who often switch weaponry to fit the occasion; highly adaptable, and play a lot of PvP
Sitting Duck Snipers	18.8	Players who are more prone to shoot from a single location and switch weaponry as the enemy closes in
Mobile Marksmen	18.1	Players who stick to using a weapon of choice (Pulse Rifles) and move around to maintain distance when fighting enemies

TABLE IV

PvE PROFILES FOR GAUSSIAN MIXTURE MODELS. %P = % OF PLAYERS

Title	%P	Characteristics
Pulse Rifle & Sword Reliant	5.5	High usage in pulse rifles and swords, but no other stand-out qualities
Auto Rifle Reliant	29.4	High usage of auto rifles, low scout rifles, but no other stand-out qualities
Mobile Marksmen	28.1	Highest average kill distance; high usage of scout rifle
High Variety	7.7	Lowest reliance on any single weapon type
Sitting Duck Snipers	21.2	High sniper but mid-range average kill distance suggest a player unwilling to adapt to surroundings
Close Combatants	8.2	Shortest kill distance and reliance on short-range weapons and melee

Table 4 shows prototypes derived via Gaussian mixture models. In this case, the commonalities are the *Mobile Marksmen*, *Sitting Duck Snipers*, and *Close Combatants*, though the proportions of each differ significantly. The combination of features defining the behavioral profiles here titled *Pulse Rifle & Sword Reliant*, *Auto Rifle Reliant* and

*High Variety* proved difficult to interpret in the context of the mechanics in *Destiny*. What the former three clusters have in common is that they all are defined by a set of weapons that seem to be consistent with the average kill distance, which creates an easily interpreted cluster. On the other hand, the other three clusters share the characteristic of being defined by only one or two weapons with nothing else that stands out in great detail.

## VIII. RESULTS AND DISCUSSION

*Destiny* provides a number of challenges to the task of developing behavioral profiles, including the sheer variety and volume of this player telemetry data which complicates the feature selection process. To align with the goal of creating high level behavioral clusters, overly specific features can be systematically excluded. A second level of filtering requires knowledge of *Destiny*'s mechanics, and targets features that cover primary gameplay. Lastly, care must be taken to isolate game modes with different gameplay, e.g. PvP and PvE.

The primary differentiators of character behavior fall into three dimensions: a) the usage frequency for different weapons, b) the average kill distance, and c) the time spent playing either PvP or PvE. The first of these typically align with each other; players getting more frequent kills with typically longer range weapons tend to have larger average kill distances. Given that *Destiny*'s main gameplay revolves round the collection and upgrading of weapons, the importance of kill frequency by weapon type is understandable, as players may latch onto certain weapon archetypes early in the game and develop a signature loadout. Some clusters, however, display a variety of weapon usage, suggesting that a portion of the playerbase is willing to adapt to various situations in different game modes by changing their weaponry. The game type dimension groups the cluster results into either PvP focused or PvE focused, with few players spending equal time in both. Within each game mode, other features serve as proxy measures for activity preferences. For example, offensive and defensive kills are exclusive to the control gametype, in which teams guard territory to earn points, so clusters with large values for these features may correspond to players that prefer objective-based gameplay. Players tend to focus on either only a few, or a variety of weapon types. Regardless of game mode, clusters of players emerge that prefer either extreme close range or long range playstyles. Long range players use scout and pulse rifles for primaries, and sniper rifles for secondaries. Short range players specialize in melee attacks and point-blank shotgun blasts. Players that vary their weapon choice also tend to include melee attacks, and special abilities such as grenades and super abilities. Player preference for PvE or PvP varies between two extremes, and within each game mode preferences for specific activities are revealed through average playtime and types of kills.

In addition to the three main dimensions described above, the results also demonstrate variability in features that are either more subtle or secondary to *Destiny*'s main gameplay goals of collecting items and defeating enemies. E.g. the

ratio of player resurrections performed to received was significantly above average for some clusters. In these cases, the values for the remaining features did not seem to follow any identifiable pattern. This could mean that some players are inherently more attuned to supportive roles, regardless of their preferences for certain weapon types. Another feature - the average time remaining in a PvP activity when a player quits - allows for inferences to be made about more nuanced player behavior. Some clusters show high values for this feature, suggesting that some players may be more likely to leave early if a match is not going their way. From the matrix and clusters we can conclude that each method gives varying results. If one specifically knows what kind of behavior needs to be analyzed, it is crucial that the appropriate method is used during the exercise, e.g. k-means for more general behavior or archetype analysis for more extreme behavior. Otherwise, it is important for a clustering project to include a variety of methods in order to evaluate a range of behaviors/scenarios, encompassing both general/typical- and more extremal behavior.

The developers of *Destiny* have designed an apparently well-balanced game when it comes to performance; a digital experience where players with different preferences can adopt diverging strategies in order to hit the level-cap and to continue the experience beyond that point. The aforementioned balance is indicated in the cluster analysis results, with varying styles colliding at the top-levels of the game. For developers, an analysis with similar results would serve as an evaluation of the design intent in delivering an experience that can be played in a variety of ways. The results indicate that there is no best method to examine how players form clusters, but that the choice should be determined by the goal of analysis and include multiple models. In essence, different clustering models are more or less suited for specific circumstances or for providing specific views on the data. The choice of clustering algorithm is important [6].

## IX. CONCLUSION AND FUTURE WORK

In this paper behavioral profiles were developed for *Destiny* across four cluster models. The results highlight patterns in the behavior of players in the game across PvP and PvE modes, with a focus on performance and playstyle measures. The challenges of operating with high-dimensional behavioral data and comparing results across cluster models has been described and discussed. Future work aims at building on the profiling results towards the creation of an item recommendation system for *Destiny*. The first step in this process will be extending across the character level range, and generate performance/playstyle clusters as a function of progression, adopting a more dynamic performance view. Secondly, the equipment held by each player can be incorporated into the analysis, providing insights into what weapon choices are preferred by the most skilled players for each playstyle at all levels of in-game progression. With this information, a recommender system for suggesting items to players can be developed. Additional work will also focus on methods such as agglomerative and divisive

hierarchical clustering. While the methods used in this paper were selected to cover centroid and extrema-based models that have previously been used in games, there are myriad other methods worth comparing in similar fashion.

## ACKNOWLEDGMENT

The authors would like to extend their sincere gratitude to Bungie for making telemetry from *Destiny* available.

## REFERENCES

- [1] A. Drachen, R. Sifa, C. Bauckhage, and C. Thureau, "Guns, swords and data: Clustering of player behavior in computer games in the wild", in Proc. IEEE CIG, 2012.
- [2] A. Drachen, and R. Sifa, and C. Thureau, and C. Bauckhage, "A Comparison of Methods for Player Clustering via Behavioral Telemetry", in Proc. FDG, 2013.
- [3] R. Sifa, and C. Bauckhage, and A. Drachen, "Archetypal Game Recommender Systems", Proc. KDML-LWA, 2014.
- [4] C. Bauckhage and C. Thureau, "Making Archetypal Analysis Practical", in Pattern Recognition, LNCS Springer, 2009, pp. 272-281.
- [5] C. Bauckhage and R. Sifa, k-Maxoids Clustering, Proc. KDML-LWA, 2015.
- [6] C. Bauckhage, A. Drachen and R. Sifa, "Clustering game behavior data," in Trans. Comp. Int. AI in Games, 7(3), pp. 266-278, 2015.
- [7] M. Seif El-Nasr, A. Drachen, and A. Canossa, "Game Analytics - Maximizing the Value of Player Data", Springer, 2013.
- [8] A. Drachen, A. Canossa, and G. Yannakakis, "Player Modeling using Self-Organization in Tomb Raider Underworld," in Proc. IEEE CIG, 2009.
- [9] A. Normoyle, and S.T. Jensen, "Bayesian Clustering of Player Styles for Multiplayer Games," in Proc. AAAI AIIDE, 2015.
- [10] C. Aggarwal and C. Reddy, Eds., "Data Clustering: Algorithms and Applications," Chapman & Hall/CRC, 2013.
- [11] R. Sifa, A. Drachen, and C. Bauckhage, Behavior Evolution in Tomb Raider Underworld, in Proc. IEEE CIG, 2013.
- [12] A. Drachen, C. Thureau, J. Togelius, G. Yannakakis and C. Bauckhage. *Game Data Mining*. In (M. S. El-Nasr, A. Drachen and A. Canossa) *Game Analytics: Maximizing the Value of Player Data*, Springer, 2013.
- [13] Fraley C, Raftery A.E., mclust Version 3 for R: Normal Mixture Modeling and Model-based Clustering. University of Washington, Department of Statistics, Technical Report 504, 2006.
- [14] I. Dinov, Expectation Maximization and Mixture Modeling Tutorial, Statistics Online Computational Resource, 2008.
- [15] O. Missura, and T. Grtner, "Player modeling for intelligent difficulty adjustment," in Proc. of the ECML Workshop From Local Patterns to Global Models, 2009.
- [16] K. Shim and J. Srivastava, Behavioral Profiles of Character Types in EverQuest II, in Proc. IEEE CIG, 2010.
- [17] R. Thawonmas, K. Yoshida, J.-K. Lou and K.-T. Chen, "Analysis of revisitations in online games," Entertainment Computing, 2(4), pp. 215-221, 2011.
- [18] C. Thureau and C. Bauckhage, "Analyzing the evolution of social groups in World of Warcraft." In Computational Intelligence and Games, pp. 170-177, 2010.
- [19] R. Sifa, C. Bauckhage and A. Drachen, "The Playtime Principle: Large-scale Cross-games Interest Modeling," in Proc. Computational Intelligence in Games, pp. 365-272, 2014.
- [20] R. Sifa, A. Drachen and C. Bauckhage, "Large-Scale Cross-Game Player Behavior Analysis on Steam," in Proc. Artificial Intelligence and Interactive Digital Entertainment, 2015.
- [21] C. Bauckhage, R. Sifa, A. Drachen, C. Thureau, and C. F. Hadji, "Beyond heatmaps. spatio-temporal clustering using behavior-based partitioning of game levels," in Proc. Computational Intelligence in Games, 2014.
- [22] A. Normoyle and S. T. Jensen, "Bayesian Clustering of PLayer Styles for Multiplayer Games," in Proc. of the AAAI Artificial Intelligence in Interactive Digital Entertainment, 2015.
- [23] R. Sifa, C. Ojeda, and C. Bauckhage, "User Churn Migration Analysis with DEDICOM," in Proc. ACM RecSys, 2015.
- [24] A. Drachen, M. Yancey, J. Maquire, D. Chu, Y. I. Wang, T. Mahlman, M. Schubert and D. Klabjan, "Skill-Based Differences in Spatio-Temporal Team Behaviour in Defence of The Ancients 2 (DotA 2)," in Proc. IEEE GEM, 2014.

# Towards The Automatic Optimisation Of Procedural Content Generators

Michael Cook  
Games Academy  
Falmouth University  
mike@gamesbyangelina.org

Jeremy Gow  
Computational Creativity Group  
Goldsmiths, University of London  
j.gow@gold.ac.uk

Simon Colton  
Games Academy  
Falmouth University  
simon.colton@falmouth.ac.uk

**Abstract**—Procedural generation is important to modern game development as both an artistic implement and an engineering tool. However, developing procedural generators and understanding how they work are both difficult tasks, and even more so for novice developers. In this paper we describe Danesh, a tool to help in analysing, changing and exploring procedural content generators. In particular, we describe several features in Danesh which help a user optimise their procedural generator towards a certain kind of output by automatically changing parameters and evaluating the effect it has on the generator. We compare different approaches to these tasks and describe our future intentions for Danesh's automated features.

## I. INTRODUCTION

Procedural content generation is an important part of modern game development and a well-known concept among gamers and critics [1], useful both as a tool for solving problems [2] and a paintbrush for expressing ideas [3], and often employed as both at once [4]. The ability to generate game content automatically opens up the potential for new kinds of game to be made possible [5], as well as easing the development of games by allowing abundance, serendipity and surprise to be added to a game in very simple ways [6].

In our experience, developing procedural generators is hard. Unlike many other common concepts in game development like physics engines [7] or particle systems [8], most modern tools for making games do not come with built-in support for content generation. Where they do offer such support, it is usually in the form of fixed-purpose generators of (usually decorative visual) content, such as Unity's tree generator [9]. In addition to this, a user wishing to learn more about procedural generation is mostly limited to tutorials about generating highly specific content through one method (such as Unity's cave generator tutorial [10]), which usually focus on the process of implementation rather than understanding and customising a generator after the fact. As an example, the tutorial in [11] ends by telling the reader: 'the final step is down to you: you must iterate over what you learned to create more procedurally generated content for endless replayability'.

Understanding procedural generators can also be difficult. Unlike traditional content creation, the user cannot see the entirety of what they are creating all at once. Instead, they typically view an example output from the generator (perhaps multiple examples in quick succession) to assess the approximate type of output the generator might produce. In a survey

we conducted of 53 game developers, 85% of them stated that their primary method of testing a generator involved changing parameter values and repeatedly viewing the output. There are many problems with such an approach: it doesn't capture the variance or distribution of the output; it can't detect outliers or anomalous results; it is extremely hard to do when parameters have nonlinear relationships with the output or interact with other parameters; and it doesn't provide feedback to the user as to whether their goal is even achievable.

In order to help new practitioners learn about procedural content generation as a skill and an artform, we need to build tools that focus on domain-independent analytical techniques, that provide as much help and feedback as possible in the process of learning what a generator does and understanding how to change it. There is also a need for such tools among experts too – better development tools could streamline the process of working with procedural generators and help make them more accessible to artists and designers [12]. This would make their use in modern game development easier, but also promote experimentation among expert communities [13].

In this paper we describe Danesh, an open-source tool for helping designers and developers of all experience levels explore, explain and experiment with procedural content generators. We focus here on describing our techniques for automatically optimising and analysing the generators, including identifying useful parameters for the generator to tweak and automatically configuring the generator towards producing a certain kind of output. In addition to being useful for both novice PCG developers and experts, we hope that by developing Danesh as an open-source tool we can promote crossover between procedural generation researchers, and provide an open platform for implementing analytical ideas about generative software.

The remainder of the paper is organised as follows: in section II we discuss existing work at the intersection of procedural generation and design tools; in section III we introduce Danesh and describe some of its features, including a suite of analysis tools; in section IV we describe the ways in which Danesh can automate some of its processes, and describe experimentation done to assess the best techniques; in section V we discuss the strengths and limitations of Danesh in its current form, and describe our intentions for future work on the system; finally, in section VI we summarise the paper's

results.

## II. BACKGROUND

In [14] Smith and Whitehead describe Tanagra, a tool for generating levels for platforming games using a rhythm-based system for labelling and slotting content together. In this paper they introduce the notion of *expressive range*, a way of evaluating a generator based on the qualities of its output. Hundreds of pieces of content from the generator are produced and then evaluated according to two metrics (in the paper these are the linearity of a level and leniency of its difficulty structure). The values are then plotted on a histogram, with bright colours or larger points indicating that more pieces of content fall in a particular area. This provides a neat visual summary of the current state of the generator, as the user can see information such as the spread of the generator's output (whether the cluster of points is large or small); the correlation between axis metrics (whether the points appear to correlate along a particular line); the quantity and nature of outlying points; and areas of the metric space which the generator does not currently cover. We have implemented expressive range in Danesh and this enables the user to generate histograms based on those described in [14].

Besides Tanagra, other attempts have been made to produce tools for generating content, typically tailored to one specific kind of game content such as maps or levels. Ropossum [15] uses a physics simulation to verify levels for the game Cut The Rope, and can either generate levels from scratch or co-create with a human user who has produced a partial level design. The Sentient Sketchbook [16] is a slightly more general tool aimed at developing maps – users can sketch a map at a low resolution and then upscale the map to a more detailed version automatically using the tool. Danesh is related to this work primarily because all of these tools are concerned with generative software – however, Danesh is a general analytical and developmental layer intended to be plugged into an existing generator, whereas the systems mentioned above are all generators themselves, trying to help a user solve a particular type of content generation problem.

In [17] the authors describe a procedural procedural level generator generator. This represents another attempt at higher-level descriptions of generators, by describing the properties of a system which then goes on to generate procedural generators. The work describes the notion of 'inner' and 'outer' generators, where the inner generation process is parameterised by the outer level, which the user interacts with. Similar to the tools we have mentioned, this is a bespoke tool aimed at a particular game domain, although the techniques are quite broadly applicable. One of the most interesting features of this work is how playful and divergent the system is – interacting with the generator generator is an enjoyable experience, and can produce unusual and unexpected results.

In [18] the authors present procedural generation from the perspective of those who use it, and highlight the different metaphors practitioners use when discussing it. One of the motivations for the work is to explore the possibility of a

'shared language' with which to talk about procedural content generation across different communities and areas of expertise. The four metaphors proposed are *Tool* (something which acts as an extension of its user to achieve a goal); *Material* (something that can be shaped or manipulated into a particular form); *Designer* (something that solves a design problem independently or collaboratively); and *Expert* (something that holds specific knowledge about a domain and can interpret data based on this knowledge).

The work presented in [18] not only reinforces how widely-used procedural generation is, by people of diverse backgrounds and experience, but also how important it is to provide different ways of engaging with this technology, and to assist people in getting the most out of these ideas by providing different ways to think about and explore them. We believe the work on Danesh outlined in this paper continues some of these ideas by trying to provide new tools to help people better understand procedural generation.

## III. THE DANESH TOOL

Danesh is a Unity plugin designed to help developers to analyse and improve generative software. It is being developed with the intention of making it as general as possible – it is currently not designed to evaluate a particular kind of content generator, and instead relies on modular subsystems to evaluate and visualise generated content while being agnostic to what exactly is being generated. This does not mean that Danesh is able to work with any kind of generator – there are limitations to the current version of the software, which we discuss later. However, it can handle a wide range of generator types and provides a suite of useful tools for analysing and improving them, which we briefly describe in this section.

Danesh is an open-source C# project and can be downloaded from GitHub<sup>1</sup>. We are writing tutorials for new features of the tool as they are developed – these, as well as further information about Danesh, can be found on the tool's website<sup>2</sup>.

### A. The Cellular Automata Cave Generator

Throughout the remainder of this paper we will be using a cave generator based on cellular automata [20] whenever an example generator is needed for explaining a feature of Danesh. The implementation of the generator is based on [19]. The generator produces two-dimensional grids of solid and empty tiles which describe a top-down view of a cave-like structure. The implementation of the generator in Danesh has four parameters: the *initial chance* a tile will be randomly assigned as solid when the algorithm begins; the number of *iterations* the algorithm is run for; and two numbers that define the conditions for a tile changing from solid to empty, or empty to solid (called the *birth* and *death limits*).

Danesh uses what we call *metric functions* to measure features of a generator's output. These metric functions must be provided by the user before starting to use the tool. They are written as methods which take a piece of generated content

<sup>1</sup><http://github.com/gamesbyangelina/danesh>

<sup>2</sup><http://www.danesh.procjam.com>



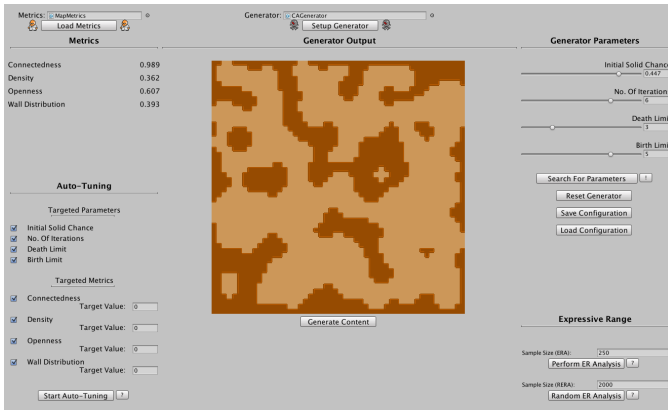


Fig. 1: A screenshot of Danesh working with a generator.

and return a number in the interval  $[0, 1]$ . We have written five metrics for the example cave generator:

- **Connectedness:** Measures the largest contiguous area of empty tiles, returns its area as a proportion of all empty tiles in the cave. (i.e. If the cave is one single contiguous area, returns 1.0).
- **Density:** Calculates the proportion of solid tiles.
- **Openness:** Calculates how many empty tiles are not adjacent to any solid tiles, as a proportion of all empty tiles.
- **Jaggedness:** Calculates how many empty tiles are ‘jagged’ (a tile is jagged if it is adjacent to two solid tiles on opposite sides), as a proportion of all empty tiles.
- **Wall Distribution:** Calculates how many empty tiles are adjacent to one or more solid tiles, as a proportion of all tiles in the cave.

The following sections will explain the use of these metrics and parameters in Danesh.

### B. Loading, Viewing, Changing

Before loading in a generator, the user tags sections of their code with custom C# Attributes that mark out methods used for generating content and for visualising that content. Danesh can currently visualise content either as text, or as a 2D image. The user is relied upon to write the visualisation function for their content, although we provide a small suite of utility methods to help them do this easily. In order to load a generator, three things must be tagged with attributes: a `Generator` method which Danesh can call to generate a new piece of content, a `Visualiser` method which Danesh can call to display a piece of generated content on-screen, and any fields of the generator that the user wishes to change using Danesh’s interface. When tagging a field, the user must also provide a simplified name to display in the tool, and minimum and maximum values for the field (where applicable).

Figure 1 shows a screenshot of the main interface after loading a generator. The display is split into three columns. In the central column is a piece of content that has been generated by Danesh and displayed using the visualiser. In the right-hand column is a series of sliders, each one referring to a field in the

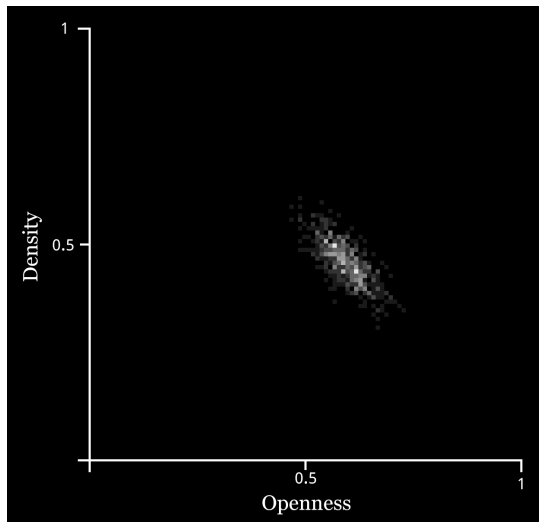
generator that was tagged by the user. The slider is limited by the minimum and maximum values set by the user. Changing the slider values directly adjusts the underlying generator, and the user can then click a button to generate and display a new piece of content using the changed parameters.

### C. Expressive Range Analysis

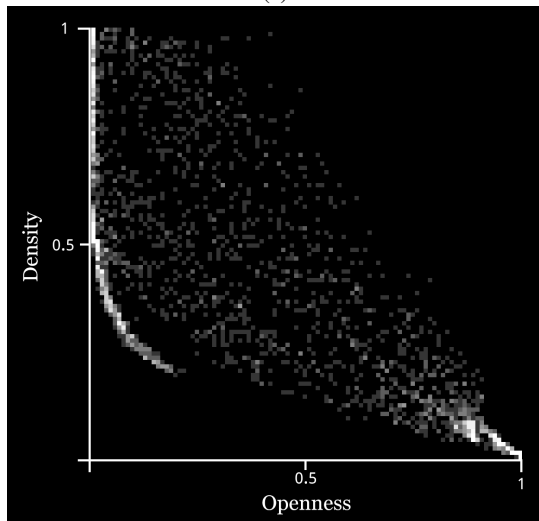
In the bottom-right corner of the interface in Figure 1 is a group of controls for the expressive range analysis functions of Danesh. Expressive range analysis (ERA) is a method of analysing a procedural generator according to the qualities of its output rather than those of its parameters. Suppose we have a generator with a set of fields and a zero-argument generation method `GenerateContent`. Additionally, suppose we have two metric functions `Metric1` and `Metric2` which take an output from the generation method and return a number in the interval  $[0, 1]$ . To produce an expressive range histogram, we repeatedly sample from the generator and calculate both metric values for each sampled output. We record each result by multiplying it by 100, flooring the result, and using the resulting integer to index a 100-element array, incrementing the indexed value. For example, if `Metric1` returns a value 0.87995, we calculate  $\lfloor (0.87995 * 100) \rfloor$ , which gives us 87, and then increment the value in the 87th element of the `Metric1` array.

To produce the visual ERA output we plot the data on a histogram with the two axes referring to `Metric1` and `Metric2`. If there are no samples at a particular co-ordinate, no colour is plotted. The higher the number of samples recorded at a point, the more intense the colour is plotted. This representation is derived from Smith and Whitehead’s original expressive range histograms in [14]. Figure 2a shows an expressive range analysis of the cellular automata generator, as seen in Figure 1. The x-axis records *Openness* – the proportion of tiles in a level which have no solid tiles adjacent to them – while the y-axis records *Density* – the proportion of tiles in a level which are solid (regardless of what they are adjacent to). From the ERA in Figure 2a, we can see that the generator produces levels which contain about 50% solid tiles, and of the open tiles, just over 50% are not touching solid tiles. This might mean that the space in the dungeon is more open rooms than narrow corridors. The user can get more information about a data point in the ERA by hovering their mouse over it to view an example piece of content.

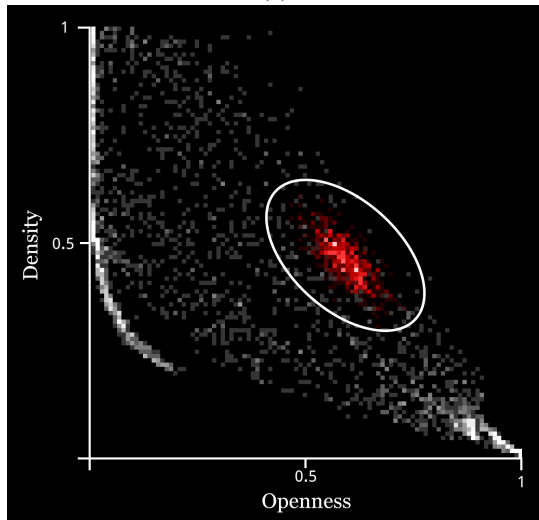
An ERA provides a visualisation of a single generator configuration, because it samples the generator with a set of fixed input parameters. This is useful when considering a particular configuration in detail, however it is not especially useful when the user wants to consider what might be possible with other configurations of the system. To this end, Danesh also provides a *randomised* expressive range analysis, or RERA. A RERA is performed very similarly to a standard ERA, except that each time Danesh samples the generator, the parameters are randomised within the minimum and maximum values set when the generator was loaded. The resulting histogram shows a broader picture of the generator’s potential generative space. Figure 2b shows a randomised expressive range analysis of the



(a)



(b)



(c)

Fig. 2: Top: An example of an expressive range histogram. Middle: An example of a random ER histogram. Bottom: The ERA from 2a overlaid in red on the RERA from 2b. We have circled the area for readers of a B&W version of the paper.

same procedural generator which produced the ERA in Figure 2a. Here we can see a much wider set of potential outputs from the generator, as well as seeing which areas are more dense with content than others – or even which parts of the metric space the generator does not appear to cover at all.

Figure 2c shows the standard ERA overlaid in red on top of the RERA. From this image, we can see the current configuration of the generator in the wider context of its potential. We can see, for example, that we can increase the openness or the density of the generated content, but not both (the top-right of the RERA is completely dark, indicating no generated content appeared in this area). This could be because the parameter intervals are not wide enough to explore. It could also be because there is some kind of conflict between the metric features (which is the case here – we cannot have extremely dense levels which also have a lot of open space, so it makes sense that we can't maximise both metrics). Most importantly, however, it might mean that the generative algorithm itself needs revising, because its structure means that content of this type cannot be generated currently.

RERAs provide the user with information about the potential of the wider space their generator exists within, while ERAs confirm the current state of the generator. Using the two in tandem, the user can make adjustments to their generator and then verify the effect of the changes by performing regular ERAs and examining the changes made. This is an improvement on simply generating and examining single examples, because the increased density of data contained in an ERA or RERA allows the user to understand the shape of the generative space better. They can also hover over the histogram to view points of interest such as outliers, or more dense/sparse areas, to better inform their decision-making.

For example, Figure 3 shows a RERA with two different ERAs superimposed for the purposes of illustration. In this histogram, the y-axis shows *Wall Distribution* – a similar concept to Openness, which we described earlier and showed in Figure 2a – and the x-axis shows *Density*. The first ERA, circled at top of the histogram, shows that the generator was producing content with a high *Wall Distribution* score. The user has tried to reduce this, and the second ERA (circled in the lower part of the histogram) shows they have been successful in doing so. We can also infer other small changes from the shape of the second ERA – the spread of the generator's output has increased slightly, and the average density has been slightly reduced. The user may wish to make further changes if this is not exactly what they wanted.

#### IV. AUTOMATING GENERATOR OPTIMISATION

So far we have described the basic investigative and analytical techniques of Danesh. These provide data and feedback to the user, but are primarily user-led and only enable people to perform adjustments and exploration of the generator. The user is able to tag parameters to identify areas for investigation, adjust those parameters manually in the tool, and then run ERA and RERA analyses to explore the space. This process of iteratively changing parameter configurations and running

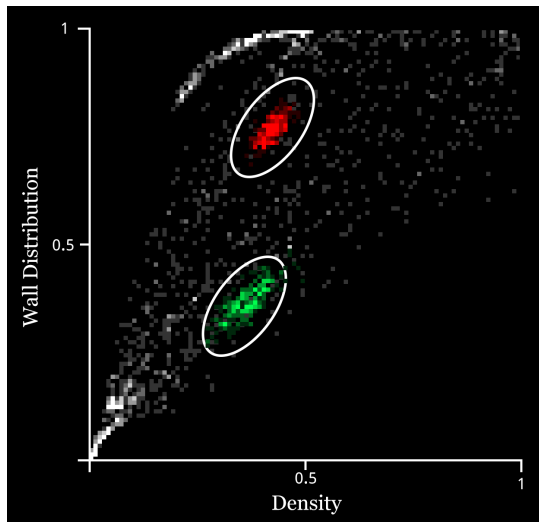


Fig. 3: An RERA with two ERAs superimposed (circled for B&W readers). The two ERAs show the state before and after adjusting the generator to reduce the y-axis metric score.

ERA analyses is time-consuming, however, and there may be hidden non-linear relationships between parameters or between parameters and metrics that make it hard to predict the impact of small adjustments on the output of the generator.

In this section, we describe features implemented in Danesh to automate parts of this process, and report on some analyses of the approaches we undertook. In the future we intend to provide automation support for other tasks the user may wish to undertake with the tool – we discuss these in Future Work.

#### A. Automatic Parameter Identification

Recall that in order to add a generator parameter to Danesh’s interface the user must add C# Attribute tags to their code to label each parameter individually. Adding parameters indiscriminately is not a good idea, since it clutters up the user interface and makes automation tasks more difficult. It can also confuse novice users who might need help focusing on parameters that have meaningful impact on the generator. Thus, deciding which parameters to add and which to leave out is a difficult decision to make. There are also many reasons why a user might mistakenly decide not to add a parameter to Danesh – perhaps they forgot the parameter was there, underestimated how useful it might be, or simply don’t want the burden of deciding which parameters to add. It would be helpful for the user if Danesh were able to intervene and automatically add useful generator parameters. To facilitate this, we have implemented a feature into Danesh which searches for fields in the generator code, tests their feasibility for influencing the generator, and then prompts the user to add them with upper and lower bounds and a name. The process by which a particular field is tested for feasibility is as follows.

First, before testing any fields, Danesh collects an average metric sample by generating several pieces of content (cur-

rently we use a sample size of 20) with the generator as-is and then computing the average metric score and standard deviation for each metric across all the generated content. We then use a metaprogramming technique called *reflection* to examine the generator’s class and extract a list of fields declared in it. For each of these fields (we currently only use numerical values and booleans) we set the field’s value to a series of test values, generate another sample of 20 pieces of content, compute the average metric score for this new sample and compare it to the baseline metric average. If any of the test values for the field produce an average metric score more than one standard deviation away from the original generator average, we consider the field to have had an effect on the generator and add it as a potential parameter.

The testing values we use for the field are pre-set values chosen via earlier experimentation to cover common parameter ranges. For boolean values, we simply test both values and see if there is a difference. For numerical values we test the following values:  $\{-1, 0, 1, -100, 100, \text{MAX\_VALUE}, \text{MIN\_VALUE}\}$ . If the numerical type is a floating-point value we also test  $\{-0.5, 0.5\}$ . We currently do not test string fields or other types, this is planned for future work.

Once this process is complete, Danesh presents the user with a list of potential parameters to add, along with three input fields to set a name for the field, a minimum value, and a maximum value (the same information given when a user manually tags a field using an attribute in their code). The user can then either confirm the addition of the parameter to Danesh, or delete the suggestion.

As an example, when running this procedure on our example cave generator it suggests both the width and height of the cave as parameters to the system, as well as a boolean value which sets whether the edge of the map is considered solid or empty (for the purposes of calculating tile births/deaths). The width and height parameters are useful for scaling the system, while the boolean field has a large impact on the expressiveness of the system - setting the edges of the map to be treated as empty changes the generated caves greatly, making them much sparser and fragmented.

#### B. Parameter Configuration Search

One of the tasks Danesh was designed to facilitate is the act of changing a generator’s parameters to achieve a different kind of output (in terms of the type of content being produced, how variable the content is, how likely outliers are to occur, what properties hold of the content, etc.). As we mentioned in the introduction, a small survey we conducted of 53 game developers indicated that 85% of them typically achieve this by adjusting parameters and then viewing individual output examples to assess whether the change was good or bad. We have already described in the previous section analytical tools in Danesh such as metric functions and expressive range analysis to help the user achieve some of these outcomes without the need for laborious output examination.

Danesh can also automate this process of parameter search entirely, allowing the user to simply specify a target outcome

in terms of desired metric scores and then let Danesh search the parameter space to find a parameterisation of the generator whose average metric score is as close to the user target as possible. To set up an automatic parameter search, the user must provide some information: first, they select which parameters they wish Danesh to search over. Fewer parameters results in a more efficient search, but more parameters covers a wider space of possibilities, so the user must decide how broad they want the search to be. Then, the user selects which metrics they wish to target for the search, and what values they wish the average generator output to be.

We implemented three algorithms for searching for parameterisations, in order to test different approaches: random search (as a baseline), hill climbing, and evolutionary search. There are conflicting goals here when considering the best algorithm for parameter search. A high-quality solution is desirable, since we want to get a result as close to the user's specified goals as possible. However, the time taken is also an important consideration – Danesh is designed to be an interactive application, so long periods of time searching for a solution is not desirable. In this section we describe the algorithms and some experiments performed to assess the relative performance of each one.

All of the algorithms use the same definition of fitness to assess the quality of a particular parameter configuration, which we define as follows. Given a set of  $n$  metric functions  $f_1 \dots f_n \in F$ , a target value  $t_i$  for each metric function  $f_i \in F$ , and a set of  $p$  generated content samples  $c_1 \dots c_p \in C$ , we define the **fitness** of a parameter configuration as follows:

$$m_i = \frac{\sum_{c_j \in C} f_i(c_j)}{|C|} \quad \delta_i = |m_i - t_i|$$

Where  $m_i$  denotes the mean value for metric function  $f_i$  on the set of generated content  $C$ , and  $\delta_i$  denotes the difference between the user's target value  $t_i$  for the metric and the observed mean value  $m_i$ . The score  $\Phi$  is then expressed as an average of these differences:

$$\Phi = \frac{\sum_{i=1}^n \delta_i}{|F|}$$

**Randomised search** randomly generates parameter values for the selected parameters, using the minimum and maximum values set by the user as limits. It then evaluates them by sampling from the generator and recording the average metric value for each metric. The random search terminates after a set number of iterations, at which point the best parameter set seen is returned. It can also terminate after a set amount of time has elapsed. The number of samples per iteration and the number of iterations are parameters to the system – in the experiments here we test 15 samples over 100 iterations.

**Hill climbing** randomly generates an initial set of parameter values, and then iteratively changes one of the parameters by a small fixed interval, ensuring that each time it picks the interval change that results in the biggest increase in fitness. If

it cannot increase its metric score any further, it has reached a local maxima. It records this maxima if it is higher than any it has found so far, and then randomly restarts. The hill climber terminates after a set number of iterations, or after a set amount of time has elapsed. As with randomised search, the number of samples per test and the number of iterations can be set as parameters to the system. In the experiments outlined here we use 15 samples per test, over 100 iterations.

**Evolutionary search** generates a population of random sets of parameter values and then performs an evolutionary search, crossing over parameter sets using one-point crossover on the parameter array and mutating a parameter value with a 5% probability. It terminates after a fixed number of generations have been completed, but can also terminate at the end of a generation cycle after a time limit expires. We use a population of size 15 evolved for 15 generations for these experiments.

1) *Experimentation:* We set up three auto-tuning scenarios of varying difficulty to test the algorithms, based on the example cave generator:

- **P1:** 2 target parameters, 2 target metrics.
- **P2:** 2 target parameters, 3 target metrics.
- **P3:** 4 target parameters, 4 target metrics.

Because responsiveness is important to the design of the Danesh tool, in our first experiment we were interested in the performance of each technique in a time-limited scenario. We ran each algorithm on each problem five times, and on each run we recorded the best fitness available at 2.5 seconds, 5 seconds, 10 seconds and 20 seconds. These were considered hard limits on time, so if the algorithm was computing an iteration at a time limit, we recorded the best fitness reported so far. Figure 4 details the results for each algorithm and problem case combination.

There are a few points worth making about the data available. First is that, in general, there is not a huge disparity between the three algorithms at the 20s mark. However, we can see that in many of the cases the hill climbing and evolutionary search algorithms perform badly at shorter timing marks – on P2 and P3, the evolutionary algorithm fails to complete processing a generation before the 2.5s mark. As these two algorithms progress, they improve in larger amounts. We believe that the primary reason for this disparity is the number of samplings and evaluations required to iterate the algorithm. Random search repeatedly chooses and evaluates new parameter configurations - meaning it can try several configurations across a wide search range in a few seconds. The evolutionary approach evaluates fifteen different configurations in each generation, however, and the number of evaluations the hill climber makes scales with the number of parameters (since it checks incremental changes to each parameter individually). This means that initial progress is slow, but rapid improvement is made once the algorithms progress.

To highlight this, we ran a second experiment which focused on minimum fitness. Instead of sampling fitness at preset timing cutoffs, instead we tested how long each algorithm took to reach a fitness of 0.9 and 0.95. This is to simulate Danesh finding a 'close enough' result, from which a user

	2.5s	5s	10s	20s
RS	0.892	0.909	0.959	0.964
HC	0.784	0.959	0.959	0.963
EVO	0.932	0.952	0.966	0.967

(a) Timing results for P1.

	2.5s	5s	10s	20s
RS	0.868	0.879	0.892	0.909
HC	0.738	0.806	0.892	0.911
EVO	-	0.887	0.897	0.917

(b) Timing results for P2.

	2.5s	5s	10s	20s
RS	0.889	0.906	0.916	0.942
HC	0.851	0.914	0.932	0.948
EVO	-	0.877	0.925	0.942

(c) Timing results for P3.

Fig. 4: Results showing best fitnesses recorded at fixed time intervals on three problem scenarios. RS: Random Search, HC: Hill Climb, EVO: Evolutionary Search. All results to 3 significant figures, an average of five samples.

could conceivably perfect or tweak the details to fine-tune the result. If the algorithm did not reach the fitness limit in 60 seconds, we recorded a failure. The results are shown in table 5 for each algorithm and problem case combination, as before.

First, note that the 0.95 target was not reached on P2 for any of the algorithms - this is because the targets set could not be reached closely enough in this problem scenario (in that it was slightly outside of the generator's expressive range). P2 is the hardest problem of the set because although it has fewer metric targets than P3, it also has fewer parameters it can change to reach those targets. The most important result here is that while the results are relatively close again, both hill climbing and evolutionary search far outperform the random search on the 0.95 fitness target for the hardest problem, P3. This shows that although random search can rapidly explore the state space to find relatively good results, the more intelligent techniques work better when prioritising high fitness.

We are still developing and refining techniques for automatic parameter configuration, but we believe that a hybrid approach may yield good results, where the first 5-10s is spent using random search, and then the best result is used to seed a hill climber. It is likely we will offer different techniques to the user depending on what tradeoff of speed versus quality they wish to have with the result.

## V. DISCUSSION

### A. Current Limitations Of The System

We are currently working on various additions and improvements to the Danesh tool. Boolean and string fields are currently unsupported, and other conveniences such as numeric intervals or arrays are not supported in a convenient way (it is possible to use them in Danesh but workarounds are required). As we release Danesh to more developers and complete the

	> 0.9	> 0.95
RS	2.61	4.29
HC	2.71	2.21
EVO	3.35	5.65

(a) Fitness-limited timing results for P1.

	> 0.9	> 0.95
RS	11.3	-
HC	11	-
EVO	11.6	-

(b) Fitness-limited timing results for P2.

	> 0.9	> 0.95
RS	2.06	30
HC	5.61	17.3
EVO	10.4	17.9

(c) Fitness-limited timing results for P3.

Fig. 5: Results showing time taken to reach a fitness of 0.9 and 0.95. RS: Random Search, HC: Hill Climb, EVO: Evolutionary Search. All results to 3 significant figures, an average of five samples.

user studies we are currently conducting, basic features like these will be implemented to round out Danesh's feature set.

Danesh's generality comes at the expense of placing a burden of implementation on the user. Currently, the user must write a visualisation method for their generator which makes it harder for novice users (although we provide a suite of utility functions to help with this, and text rendering requires almost no visualisation code). In the future, we hope to work on some automatic visualisation methods or a stock set of visualisers for common kinds of content (such as arrays representing tile-based levels, for example).

The other main implementation bottleneck for users is selecting and writing good metric functions. We plan to extend the automation of Danesh to cover this task, and provide other ways for users to express metrics, such as allowing the system to machine learn models of metrics. This will be conducted through an interface that allows the user to label positive and negative content examples and slowly define a metric function interactively. Good metrics are crucial to all of Danesh's more complex features, so this is an important area of future work.

Another potential limitation of the tool is that it is implemented as a plugin to Unity and written for C# and Javascript generators as opposed to being a general platform-agnostic tool. We do not see this as a limitation specifically, since we are primarily concerned with connecting with developer communities, and Unity is one of the most widely-used development tools today. Integrating with Unity and its asset store will help us contact developer communities directly and hopefully have a larger impact. However, we hope that the open-source nature of the tool will allow Danesh's techniques to be reimplemented into other languages, platforms and engines, should this be a barrier for other users.

## B. Towards Domain-Agnostic PCG

The development of Danesh comes partly in response to a feeling that most work in procedural content generation, both within and outwith academia, is highly fragmented. Generative systems in games are typically highly bespoke, and as such it is harder to make theoretical connections between them. This is possibly one reason why Super Mario has continued to be such a common domain for procedural generation research – it is one area where there is a lot of overlapping existing work that provides baselines, inspiration and complementary sources of code. Of course, there are many other reasons to work in this domain as well – it is well-defined, the game is popular and well-known among potential survey participants, it is a well-understood design space. Yet the density of work provides additional appeal – comparisons, competitions, engines, tools.

While the fragmentation of generative software and research is a cause of its vibrance and diversity, it also hampers the formation of strong theoretical work that is independent of a particular domain, genre or game. While we do not see Danesh as a panacea for this, we hope that more projects like it that aim to be less domain-specific will help shift the targets of procedural generation research and encourage more domain-agnostic theoretical work on generative software. Open-sourcing Danesh hopefully enables interested researchers to branch off or extend Danesh with their own work, contribute features to the tool, and as we expand the library of example generators, will allow them to perform experiments across a wide range of generator and content types. This could contribute to more abstract theories of content generation for larger classes of games or design scenarios.

## VI. CONCLUSIONS

In this paper we introduced Danesh, a tool for exploring, explaining and experimenting with procedural content generators. We described the basic functionality of the tool, and how it affords a richer way of visualising and interacting with generative spaces. We then discussed how Danesh can automate some aspects of its own process to greatly simplify the act of iteratively refining a generator's output. We evaluated several techniques for automating parameter configuration search, and then discussed the current limitations for the system and the potential future for domain-agnostic PCG tools.

We believe there exists a skills gap in game development concerning procedural generation, and that this gap is not being bridged by traditional tools. Writing procedural generators is already a difficult task, but understanding them well enough to tweak and adjust them to a designer's liking requires a lot of knowledge that is difficult to obtain. Other comparably complex (arguably even more complex) tasks such as writing graphics shaders have been made considerably easier thanks to intuitive and useful tools. We hope the same can be done for procedural generation, and that Danesh contributes towards this goal in some small way.

Procedural generation is often seen as a simple case of 'more unpredictable stuff', content that can be thrown into a game for endless replay value without much thought. But

generative techniques are increasingly a key tool in achieving certain design goals, expressing artistic ideas, and developing new genres of game. In order to promote this growth and diversity, we need to support developers, students, dabblers and novices of all kinds, to ensure this technology is as flexible and accessible as possible.

## ACKNOWLEDGMENTS

The authors wish to thank Gillian Smith for insight into her expressive range analysis work. We also thank our reviewers who provided a lot of useful feedback for the project.

## REFERENCES

- [1] D. Yu, "Spelunky," <http://www.spelunkyworld.com>, 2009.
- [2] J. Togelius, G. Yannakakis, K. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 172–186, 9 2011.
- [3] K. Compton and M. Mateas, "Casual creators," in *Proceedings of the Sixth International Conference on Computational Creativity (ICCC 2015)*, Brigham Young University. Park City, Utah: Brigham Young University, June - July 2015, p. 228–235.
- [4] S. Murray, "No Man's Sky," <http://www.no-mans-sky.com>, 2016.
- [5] M. Treanor, A. Zook, M. P. Eladhari, J. Togelius, G. Smith, M. Cook, T. Thompson, B. Magerko, J. Levine, and A. Smith, "Ai-based game design patterns," in *Proceedings of the 10 International Conference on Foundations of Digital Games, FDG*, 2015.
- [6] J. Togelius, N. Shaker, and M. J. Nelson, "Introduction," in *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*, N. Shaker, J. Togelius, and M. J. Nelson, Eds. Springer, 2015.
- [7] I. Parberry, *Introduction to Game Physics with Box2D*. CRC Press, 2013.
- [8] W. T. Reeves, "Particle systems—a technique for modeling a class of fuzzy objects," *ACM Trans. Graph.*, vol. 2, no. 2, pp. 91–108, Apr. 1983. [Online]. Available: <http://doi.acm.org/10.1145/357318.357320>
- [9] Unity Technologies, "Tree editor documentation," <http://docs.unity3d.com/Manual/class-Tree.html>, 2016.
- [10] S. Lague, "Procedural cave generation tutorial," <https://unity3d.com/learn/tutorials/projects/procedural-cave-generation-tutorial>, 2015.
- [11] Captain Kraft, "Create a procedurally generated dungeon cave system," <http://tinyurl.com/pcgtut>, 2013.
- [12] M. Ansari, *Game Development Tools*. Crc Press, 2011.
- [13] M. Cook, "Make something that makes something: A report on the first procedural generation jam," in *Proceedings of the Sixth International Conference on Computational Creativity*, 2015.
- [14] G. Smith and J. Whitehead, "Analyzing the expressive range of a level generator," in *Proceedings of the Workshop on Procedural Content Generation in Games*, 2010.
- [15] N. Shaker, M. Shaker, and J. Togelius, "Evolving playable content for cut the rope through a simulation-based approach," in *Proceedings of the Artificial Intelligence in Interactive Digital Entertainment Conference*, 2013.
- [16] A. Liapis, G. N. Yannakakis, and J. Togelius, "Designer modeling for sentient sketchbook," in *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, 2014.
- [17] M. Kerssemakers, J. Tuxen, J. Togelius, and G. N. Yannakakis, "A procedural procedural level generator generator," in *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*. IEEE, 2012, pp. 335–341.
- [18] R. Khaled, M. J. Nelson, and P. Barr, "Design metaphors for procedural content generation in games," in *Proceedings of the 2013 ACM SIGCHI Conference on Human Factors in Computing Systems*, 2013, pp. 1509–1518.
- [19] M. Cook, "Generate random cave levels using cellular automata," <http://tinyurl.com/pcgtut2>, 2013.
- [20] L. Johnson, G. N. Yannakakis, and J. Togelius, "Cellular automata for real-time generation of infinite cave levels," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM, 2010, p. 10.



# An Integrated Process for Game Balancing

Marlene Beyer\*, Aleksandr Agureikin\*, Alexander Anokhin\*, Christoph Laenger\*, Felix Nolte\*, Jonas Winterberg\*, Marcel Renka\*, Martin Rieger\*, Nicolas Pflanzl\*, Mike Preuss\*, Vanessa Volz†

\*Department of Information Systems  
Westfälische Wilhelms-Universität Münster, Germany  
Email: firstname.lastname@uni-muenster.de

†Computational Intelligence Group  
TU Dortmund University, Germany  
Email: vanessa.volz@tu-dortmund.de

**Abstract**—Game balancing is a recurring problem that currently requires a lot of manual work, usually following a game designer’s intuition or rules-of-thumb. To what extent can or should the balancing process be automated? We establish a process model that integrates both manual and automated balancing approaches. Artificial agents are employed to automatically assess the desirability of a game. We demonstrate the feasibility of implementing the model and analyze the resulting solutions from its application to a simple video game.

## I. INTRODUCTION

Game balancing is the process of finding a rule-set and corresponding parameters such that the resulting gameplay defined accordingly satisfies certain design goals, such as fairness. It seems obvious that this is a very important activity during the game design process. Balancing ensures that all game components work together smoothly and in the way intended by the designer. However, balancing is not necessarily constrained to the development phase. Especially content creation after game deployment as performed by professional developers or amateur users who create add-ons or mods may require a re-balancing as a reaction to the introduction of new game components or the modification of existing ones [8].

Balancing therefore has a significant effect on a player’s experience. Thus, if games have major balancing issues, the community will speak up. A recent example of this and how it can affect the game is the addition of the R8 Revolver weapon to *Counter-Strike: Global Offensive*, a first-person shooter published by Valve Corporation. The issue in this case was that the newly introduced pistol was able to kill players with one shot and had a near 100% accuracy while moving. The community’s feedback was harsh: players were unhappy with the new weapon and some even threatened to stop playing until the issue was resolved.<sup>1</sup> Valve was forced to react fast and just a few days after release the weapon was adjusted, reducing damage and increasing spread and cooldown.<sup>2,3</sup>

<sup>1</sup>Nathan Grayson (2015). "Counter-Strike Players Really Hate The Game’s New Gun" <http://steamed.kotaku.com/counter-strike-players-really-hate-the-games-new-gun-1747153436>

<sup>2</sup>Valve Corporation (2015). "Damage Control". <http://blog.counter-strike.net/index.php/2015/12/13366/>

<sup>3</sup>Reddit User SlothSquadron (10 December 2015). "In Depth Analysis of R8 Revolver Nerf (No other weapon changes included in update)." <https://www.reddit.com/r/GlobalOffensive/comments/3wbhmf/>

But how do we define balancing? Obviously, there is no undisputed, generally accepted definition. Nevertheless, one that is prominently featured on Wikipedia reads:

*In game design, balance is the concept and the practice of tuning a game’s rules, usually with the goal of preventing any of its component systems from being ineffective or otherwise undesirable when compared to their peers.*<sup>4</sup>

For this paper, we define game balancing slightly differently, emphasizing the process and its general aim [19, cf.]:

*Game balancing is the process of systematically modifying parameters of game components and operational rules in order to determine satisfactory configurations regarding predefined goals.*

This definition uses the terms *configurations* and *goals*. With configuration, we refer to parameters that flexibly define properties and/or behavior of game components. For example, a parameter could be the maximum hit points of a character or the damage an attack does. A goal stands for an explicit, checkable condition. Schell [14] provides a lot of possible goals for balancing, the most well-known of which may be fairness. But many other goals are relevant too, as we will see in the following. It may not always be easy to derive an explicit condition, but a formalization of the problem is necessary even in manual balancing if the reliance on intuitive conceptualization of balancing is to be minimized.

We deliberately use the plural for *configurations* and *goals*. One of the contributions of this paper is our conjecture that for most systems there are many satisfactory configurations, not just one. Consequently, even though designers usually aim to obtain a single good configuration, finding several good configurations has considerable advantages because it presents us with alternatives if the chosen solution is difficult to implement, fragile, or has other unwanted properties. Additionally, the solutions obtained with the different approaches can be used as starting solutions for the other one. We also conjecture that for obtaining several satisfactory configurations we need algorithmic assistance, but at the same time it makes

<sup>4</sup>Mark Newheiser (9 March 2009). "Playing Fair: A Look at Competition in Gaming". Strange Horizons. <http://www.strangehorizons.com/2009/20090309/newheiser-a.shtml>

no sense to rule out manual balancing efforts completely. We therefore define a process integrating both automated and manual balancing, but at the same time flexible enough to be adapted to a specific game. This integrated process is the main contribution of this paper and is formally depicted in Figure 1. It is instantiated for a simple game prototype, the Zombie Village Game (ZVG), for which we compare the results of automated and manual balancing attempts and discuss how the different approaches can profit from each other in the context of our integrated model and what lessons can be learned for more complex scenarios.

The remainder of this paper is structured as follows. Firstly, Section II will report on related work, after which Section III will present the process model that is proposed for integrated game balancing. Afterwards, Section IV will describe a concrete instantiation of this process by example of the aforementioned ZVG. The major results will be discussed in Section V. The paper concludes with a brief summary and outlook in Section VI.

## II. RELATED WORK

Despite the obvious practical importance of game balancing, the process has received little scientific attention until now. For this reason, in the following, we not only discuss directly related work, but also publications that contain parts of the addressed problem.

Volz et al. use artificial players (APs) to demonstrate the feasibility of automatic game balancing in terms of computational complexity and minimal achievable solution quality for the card game “top trumps” [19]. However, they focus on a multi-objective approach for situations where desirable characteristics can not be prioritized a priori. They also specifically address a multi-player game and consider a relatively large configuration search space, but restrict themselves to a simple game which is easy to simulate. In contrast, we use more complex games with real-time interaction and infinitely many possible player actions. Additionally, using APs that do not act like humans to evaluate a game configuration could potentially introduce a bias if the results are not thoroughly verified with humans. In order to tackle this problem, we use an interactive approach that is shown to be robust even in the face of necessary simplifications for modeling human player behavior.

Similarly, Mahlmann et al. choose card sets for the card game *Dominion* to optimize the gameplay regarding the decisiveness of a win in the game [10]. In contrast to parameter optimization, the balancing problem in this case is combinatorial in nature. The game is also multi-player and round-based, unlike the ZVG. Again, the authors do not look at the possible interaction between manual and automatic balancing.

Jaffe presents an approach to game balancing that is based on analyzing the win rates of different artificial players in a simulated game. The capabilities of these APs are restricted in different ways in order to investigate the satisfaction of balancing goals [7]. However, separate testing of different goals might lead to a result that is not easily integrated,

whereas our approach is able to handle a conclusive scenario. Again, the lack of interaction between the algorithm and humans could greatly bias the search.

Interactive, or so-called mixed-initiative approaches, that try to alleviate this, have mainly been used for level design. For example, in [17] the authors present a system that can propose levels for a side-scrolling 2-D platformer that are guaranteed to be playable. Similarly, Liapis et al. introduce a tool that is intended to suggest novel and playable levels to a designer and visualize them as support for gamedesign related decisions [9]. However, neither of these approaches expressively focuses on balancing or enables the designer to measure configurable design goals. Liapis et al. include a study on game designers that has an overall positive response to design support tools, which is encouraging for this work as well.

The automatic evaluation of a game has also been addressed in procedural content generation, specifically in the context of map or level generation. However, publications in this field mostly focus on specific simple criteria like playability [18], solvability [15], or diversity [6, 12].

Further related research areas include the dynamic adaptation of the difficulty in single-player games [5] and the generation of rules to conceive new games [2, 16]. Nelson et al. published a formalization of the latter in order to gain better insight into the game design process [11]. The process formalization we propose here, however, is intended as a guideline for practical work, specifically for automatic balancing rather than game generation.

## III. PROCESS MODEL

In this paper, we propose that game balancing should be envisioned as an *integrated process* that incorporates both automated as well as manual components. To describe this process, the *Business Process Model and Notation*<sup>5</sup> (BPMN) widely-used in the Business Process Management domain is used. This notation allows describing the order and conditions for the execution of a set of activities in order to accomplish a certain goal, which in this case lies in balancing a particular computer or video game. The main components of BPMN are *events* represented by circles, *activities* depicted as rounded rectangles, and *gateways* represented by diamond shapes. Arrows between these elements describe the *sequence flow* of the process, i.e., the order in which activities are executed.

Focusing on the top-level process model depicted in Figure 1, two events can be seen: one indicating the begin of the balancing process on the left, and one depicting its end on the right. Furthermore, two types of gateways are used: whereas a diamond containing an “x” represents an exclusive OR, a diamond containing an “o” represents an inclusive OR. All activities contain a small “plus” icon, meaning that they contain sub processes with additional details. Further elements used in the model are documents and databases which are involved in activity execution. More detailed information about BPMN can for instance be found in [3].

<sup>5</sup>See: <http://www.bpmn.org>. Last accessed: 2016/04/07

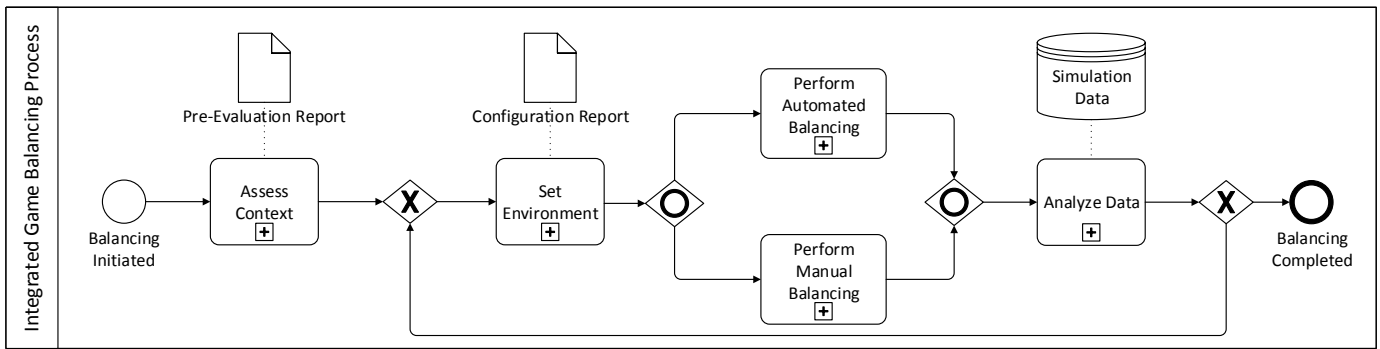


Fig. 1. Integrated Game Balancing Process modeled using BPMN.

The following paragraphs provide an in-depth description of the balancing process model presented in Figure 1. In particular, conceptual details about the activities shown in the model are discussed, including the required inputs as well as the outputs of these activities. The applicability and generalizability of the process is demonstrated by instantiating it for the ZVG in Section IV.

**Assess context.** This activity deals with the conceptual preparation of the balancing process. First, the *game* to be balanced needs to be selected. Furthermore, as a preparation for later balancing steps, a *stakeholder report* containing information about the target group of the game and other relevant parties must be compiled. Lastly, the *goals* of the balancing process must be defined. These goals encapsulate the knowledge and expertise of game designers about how the game is meant to be played and what kinds of emotional responses are intended. How well the intentions are met (i.e. how well balanced a game is) should be quantifiable so that these measures can be operationalized for algorithmic optimization.

**Set environment.** In this activity, the technical setup of the balancing process is carried out. This requires the specification of a *set of scenes to balance*, which can either be the entire game itself, or certain parts of it representing its important mechanics. Next, the choice of *game parameters* is performed. Each parameter, e.g., hit points or attack strength, represents a single decision variable that can be modified by an optimization algorithm to meet the balancing goals. Afterwards, an objective function (*fitness function*) that measures how well the game is currently balanced needs to be defined. This function should therefore accurately represent the balancing goals for the game. The final step lies in choosing an *optimization algorithm* that is capable of tweaking the game parameters to improve the resulting fitness. As indicated by the following gateway, after the environment has been set, manual balancing, automated balancing, or both are performed.

**Perform manual balancing.** To complement the automated balancing process, it is reasonable to also perform manual game balancing using already-established tools and techniques, such as spreadsheets, “doubling” and “halving”, and the game designer’s intuition. The results obtained from

manual balancing can be used to interact with automated balancing, for example for deriving an initial parameter configuration or for validation of its output.

**Perform automated balancing.** In this core step of the process, the game is balanced by applying an optimization algorithm that detects parameter configurations which result in good objective function values. For every parameter configuration, the prepared game scenes are successively simulated and the objective function values are measured. As this requires an (adapted or newly implemented) AP for representing player actions as well as an AI for the non-player characters, this activity starts with the configuration (and possibly even implementation) of the needed AIs. The optimization process is continued until a predefined stopping criterion is reached, e.g., a quality criterion (optimal or near-optimal solution), or a predefined time limit.

**Analyze data.** Both manual and automated balancing subprocesses produce simulation results as output, that is, pairs of parameter configurations and objective function values. This data is analyzed during the data analysis phase in order to assess the quality of results and recognize important structural patterns within the decision variables and objective values. Knowledge obtained in this step can be used, e.g., to eliminate some parameters from the balancing process. Based on the outcome of the analysis, the game designer decides if the process shall be continued or if the obtained configurations are already satisfactory. This step includes an important educational aspect of the integrated balancing process, as structural patterns help game designers to understand the actually implemented game mechanisms and the resulting gameplay.

#### IV. APPLICATION

This section demonstrates the applicability of the integrated game balancing process introduced in the previous section by applying it to an actual game prototype. For that purpose, it is subdivided into multiple subsections, each of which corresponds to a single activity in the balancing process.

##### A. Context Assessment

The proposed balancing process has been applied to a simple game prototype provided by BlueByte GmbH, named *Zombie Village Game* (ZVG). The prototype can be classified

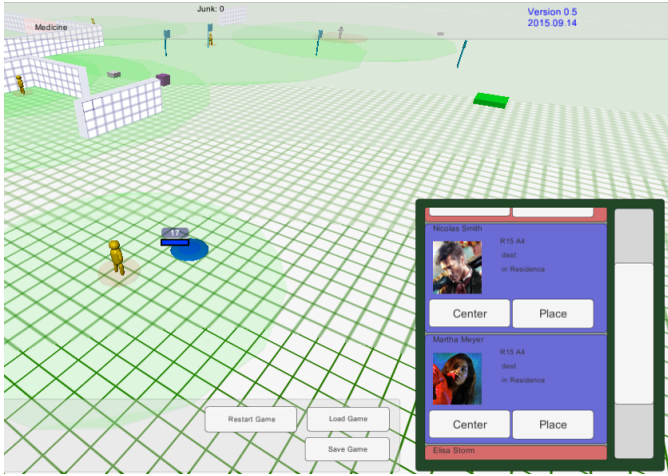


Fig. 2. Screenshot of the Zombie Village Game Prototype.

as a Tower Defense game, a sub-genre of Real-Time Strategy games. The basic idea of the concept is to defend a camp invaded by enemy zombies. The player commands units that may be freely placed on the map in order to combat the attacking zombies. Player and enemy (zombie) units attack each other automatically if they are within proximity. Resources are collected by player units from different locations and are consumed over time at a constant rate. Resource and enemy unit spawning spots are placed on the map by the level designer and are the core components of a level. Playing towards a given goal, a player has to fight the enemy units coming in large numbers. Simultaneously, they are forced to collect resources, keeping the consumption of resources over time in mind. The players are confronted with the decision of putting their limited number of units either into combat with enemy units or next to resource spots for gathering.

A gameplay example of the game is shown in Figure 2. In this screenshot, orange humanoid figures represent player units, the blue dot is a resource spot with the number of available resources displayed above it. Furthermore, a single zombie depicted as a blue figure can be seen in the top middle of the screen. The control panel on the right is used to manage (i.e. place and remove) player units on the map.

Regarding the target audience of the game, no particular focus was set. Therefore, the overall goal of the balancing process was defined *ex ante* as providing an exciting and challenging experience to all players. Consequently, players are intended to “barely” win the game most of the time, i.e., they should not be able to dominate too easily, but often come close to losing. To achieve an initial understanding of how this balancing goal focusing on “fair challenge” can be met, the game was manually played before the actual balancing activities to identify game parameters and estimate their relative influences on how the game plays out.

### B. Environment Setup

For the purpose of applying the balancing process and especially automated balancing to the ZVG, a game scene

TABLE I  
INITIAL GAME PARAMETERS CHOSEN AS CANDIDATES FOR BALANCING.

Parameter	Symbol	Range	Initial Value
Resource drain interval	$C_T$	[1, 5]	1
Resource drain amount	$C_A$	[1, 5]	1
Player Unit Start Health	$P_H$	[90, 100]	100
Player Unit Attack Power	$P_P$	[1, 20]	9
Player Unit Attack Distance	$P_D$	[1, 20]	5
Zombie Unit Start Health	$Z_H$	[90, 100]	100
Zombie Unit Attack Power	$Z_P$	[1, 10]	3

representing a very simple level was created. The scene exclusively consists of a number of resource spots and varying numbers of enemies that guard them. In order to successfully complete the scene, the player has to defeat all enemies and avoid running out of resources or losing all player units for a predefined time. The scene includes all main game mechanics and is viable for a publishable game. For a more complex game, one may have to resort to selecting several scenes for a more complete representation.

The initial set of parameters and their possible bounds were decided upon through expert knowledge gained from multiple game plays. The results of this step can be seen in Table I. Based on these parameters, we defined an *objective function* encapsulating the outlined gameplay goals as follows:

$$f = \beta_H * f_H + \beta_R * f_R \quad (1)$$

With different target audiences in mind, other choices for the components of  $f$  could be made. Here,  $f$  represents the total fitness (objective function value) of a playthrough, which is comprised of two non-conflicting components, namely the health-related fitness  $f_H$  and the resource-related fitness  $f_R$ . Furthermore, the linear weights  $\beta_H$  and  $\beta_R$  are used to influence the balance of  $f_H$  and  $f_R$ . The two individual fitness components are computed in the following manner:

$$f_H = |H^* - \sum_{i=1}^N h_i| \quad \text{and} \quad f_R = |F^* - F_r|.$$

Here,  $H^*$  and  $F^*$  represent the “optimal” target values for remaining health points and resources at the end of the game, respectively. Furthermore,  $N$  denotes the number of player units under control of the player and  $h_i$  the health of player unit  $i \in 1 \dots N$  when the scene is finished. Lastly,  $F_r$  is the remaining amount of resources in possession of the player at the end. Should the player loose, i.e., all player units die or the resources run out,  $f$  is set to a high penalty value.

The following values were chosen for the above equations:  $\beta_H = 1, \beta_R = 6, H^* = 50, F^* = 5$ . These choices were made after initial playing and manual balancing and incorporate game design expertise directed towards achieving the aforementioned gameplay goals. The best-possible fitness value of 0 is achieved if the player terminates the scene with a total sum of 50 health points and 5 units of resources.

With the objective function defined in eq. (1), we have defined a black-box minimization problem that can be tackled with different optimization algorithms. Random or grid search algorithms (or a more sophisticated experimental design as Latin Hypercube Sampling) could produce valuable initial insights into the game mechanics, especially in cases where the budget for evaluations (play-throughs) is very restrictive. In our case, the budget was in the order of several hundred up to some thousand evaluations. We thus relied on evolutionary algorithms (EAs) due to their flexibility and anytime character. We tested five EA versions with different strategies regarding parent selection, crossover, mutation and survival selection (called *algorithm configuration* in the following). As the parameters are mostly discrete, and in order to start with an easy-to-implement algorithm, we relied on relatively simple EA variants that are a mixture of genetic algorithm (GA) and evolution strategy (ES) concepts. The implemented EAs use a parent population of size  $\mu$  and generates a number of offspring  $\lambda$  in each generation, each from 2 parents out of  $\mu$ , selected in a fitness proportional manner with Goldberg's sigma-scaling [4, ch. 3]. Crossover is done uniformly random, mutation is (rounded) Gaussian perturbation and selection is done via truncation, letting the best  $\mu$  out of the pool of  $\mu + \lambda$  configurations survive to the next generation.

To decide on the concrete algorithm configuration, an adapted version of the F-RACE algorithm described by Birattari et al. [1] was used. The algorithm receives an initial set of algorithm configurations from which it determines the best one through iterative simulations and removal of configurations which have statistically significant higher cost. During the removal phase, every algorithm is assigned a cost value which expresses the anytime performance of the algorithm configuration. In our case, the initial set of algorithm configurations included 20 EA instances with varying parameters. As a cost function, we used the Area Under the Curve (AUC) of the mean fitness of the top three configurations found during the optimization process.

### C. Manual Balancing

Manual balancing was carried out following recommendations outlined by a game designer. It consisted of repeated playthroughs by two testers, the results of which were documented in a spreadsheet. At the beginning of this activity, the parameters as well as their bounds were already fixed. The parameters were set to the default values for the first two playtests in order to get a feeling for the game and get used to the controls. Then, a single parameter was adjusted using the “doubling and halving” method. The idea of the method is to understand the effects of a parameter on the game by making large changes to it. At the end of each playthrough, the fitness of the configuration was computed according to the function defined in the previous subsection. Each configuration was played five to six times to examine whether its results are reproducible. In accordance with the principles of a Tower Defense game, the playtesters employed a defensive playstyle focusing on unguarded resource spots first, small groups of

TABLE II  
CONFIGURATIONS FOUND BY MANUAL AND AUTOMATED BALANCING.

Configuration	$C_T$	$P_P$	$P_D$	$Z_H$	$Z_P$	$f$
Initial Values	2	9	5	100	3	>200
Manual Balancing	2	6	4	100	8	72
Auto-Bal. Solution 1	1	7	17	99	4	6
Auto-Bal. Solution 2	2	3	9	90	2	8
Auto-Bal. Solution 3	1	8	11	91	5	10

Zombies next, and larger groups of enemy units only at the end. Overall, 142 solutions were recorded by the testers over the course of several days. The “optimal” configuration that was eventually agreed upon by the testers is documented in Table II (row 2).

### D. Automated Balancing

As described in Section III, we propose to do automated balancing based on data gathered from playthroughs with APs. The AP we implemented for the ZVG was limited to the core actions: it is able to place units next to enemies to engage them in combat, place units next to resource spots to gather resources, and remove units that are below a certain health threshold from combat as well as move units that have depleted a resource into a combat. All possible actions are modeled in a single decision tree and chosen based on a set of predefined conditions, such as the amount of resources currently available to a player. These conditions are intended to ensure that the actions of the AP are reasonable within the context of the state of the game at any given point in time. Naturally, this simple AP can not entirely replicate the actions of a human player, but as long as its behavior is not detrimental to its progress in the game, it is satisfactory for our purposes.

Using this AP, the ZVG was successively simulated while collecting and documenting the results of each individual playthrough. Using the fitness function defined in Section IV-B, the quality of each parameter configuration was evaluated and the EA described in Section IV-B was employed to successively optimize the parameters. For each optimization run, we employed a termination criterion of about 400 simulations corresponding to roughly two hours of play-time. The best configurations were recorded, as seen in Table II. It must be noted that the configurations with the same fitness value often strongly resembled each other, and were thus summarized using the median value.

### E. Data Analysis

In the following, we describe some key observations we made on the results of the application of the proposed balancing process to the ZVG.

**Algorithm performance.** Multiple, automatically detected configurations were tested by human players to evaluate the algorithm in practice. An important aspect to analyze before assessing the algorithm itself is how well the fitness function expresses the human perception of the game and our playtesters stated that how well they enjoyed a playthrough

correlated with the obtained numerical fitness. The fitness function also seems to allow distinguishing between configurations that are playable and fun for humans (e.g. fitness values of 32 and 44) and those that are not (e.g. fitness values of 122 and 160) in many cases.

Two of the three solutions chosen for more detailed analysis were found to be challenging and enjoyable according to our playtesters, despite the games resulting from the successful configurations having a different pacing due to different values in  $P_P$  and  $Z_P$ . The last configuration did not provide a challenge at all and was easy to play despite having a good fitness. A likely explanation for this are the differences between the human and AP play styles. During the playtest, it became obvious that the player unit attack distance  $P_D$  can be exploited by the player if the player keeps placing their units just out of attack range for the zombies (*kiting*). The AP, however, was not programmed to do that and just places its units close to enemy units without even considering  $P_D$ . Considering this limitation, it is very interesting that viable results were produced and suggests that a simulation-based approach to balancing could be feasible even for complex games.

In terms of computational performance, the algorithm did reasonably well: one optimization run on one PC for the ZVG took 2 - 3 hours on average, depending on an initial seed and optimization algorithm. This is considerably faster than the several days it took two (admittedly inexperienced) game designers to find a single reasonable manual configuration.

**Fitness landscape.** 33 unique solutions with fitness value 6 were found automatically (0 would be optimal), 14 with fitness 8 and 108 with fitness 10, albeit all with only small variations. This supports the thesis of a complicated fitness landscape, since these values indicate small plateaus that a game designer would likely have trouble to identify manually. Additionally, we observed that although the solutions with the same fitness were very similar, solutions with even small differences in fitness (when compared to the standard deviation of approx. 72) differ noticeably. This further compounds the likeliness of a complex fitness landscape since there seem to exist multiple local optima.

**Parameter analysis.** To narrow down the set of game parameters that must be optimized in the balancing phase, the objective function was used to carry out an initial examination of their dependencies. This was achieved by repeatedly doing grid samples over two parameters while keeping all others constant. This enabled us to measure marginal effects of certain parameters to the fitness, as well as their correlation, without additional parameters complicating the observations. All possible pairs of parameters were simulated to construct fitness heat maps for the 2-dimensional slices of the search space. These heat maps were examined to identify parameters which are highly correlated or do not affect fitness at all.

Not surprisingly, it turned out that the parameter Player Unit Health has a huge impact on the objective values, rendering most other parameters negligible. In addition to that, parameters Time to Consume and Amount to Consume are

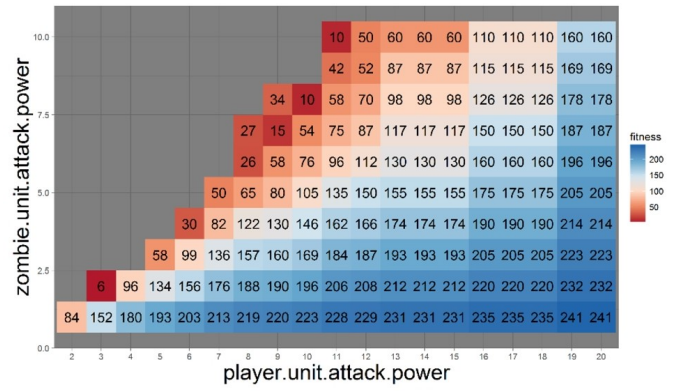


Fig. 3. Heat map of fitness values for various combinations of Zombie and Player Unit Attack Power, the rest of parameters are kept constant. Cells without fitness represent the outcomes where AP failed to win the game.

highly correlated (linear structure in heat maps). To the enable a more thorough investigation of the remaining parameters, the parameters Amount to Consume and Player Unit Health were eliminated from the balancing process and assumed to be fixed at values 1 and 100, respectively.

An exemplary heat map is depicted in Figure 3. It visualizes the interdependence of Zombie and Player Unit Attack Power. There is a dependency between the parameters, but it is not strictly linear. This figure outlines a pattern that can already be used for informed decisions on game balancing regarding these two parameters.

After eliminating the above mentioned parameters, we estimated the variable importance based on the  $R^2$  statistic of a non-parametric regression model using only one predictor against the intercept only null model as described in <sup>6</sup>. Among others, variable importance is expressed as a *Maximal Information Coefficient* that captures the relationship strength between a variable and fitness value. The results indicate that  $C_T$ ,  $Z_P$  and  $P_P$  are the most influential of the remaining variables. These are also the parameters that were constant between all solutions with the same fitness value (cf. Table II). Further related statistics (MAS, MEV, MCN) indicate that the relationship between the predictors and the outcome is relatively complex, thus suggesting that the choice to use an evolutionary optimization algorithm was justified.

## V. DISCUSSION

In the following, we discuss some findings and problems that may be generalizable to other games when applying the proposed framework. The main concerns in this discussion are three points already touched upon under the heading *Algorithm performance* in section IV-E for the ZVG, namely expressing human enjoyment, creating a believable AP and computational performance and costs.

Even with the obtained results, it is not clear whether human perception can be expressed by a fitness function in general. However, as was the case for the ZVG, there always seem to

<sup>6</sup><https://cran.r-project.org/web/packages/minerva/minerva.pdf>



be some indicators as remaining health, playtime and clicks per second that can be measured and that seem reasonable as an estimate of player enjoyment. The task of specifying their intentions in terms of these measures therefore remains for the game designer. This makes generalizing an approach for identifying a good fitness function difficult, for as long as current attempts of measuring flow and player engagement with physiological signals remain expensive.

In case that the chosen objectives are contradictory, the fitness function and selected algorithms would need to be adapted accordingly to enable multi-objective optimization. Additionally, with a multi-objective approach the designer does not need to specify any preferences in terms of objectives currently represented by the weights in our aggregated function. Besides the fitness function, another intricate part of the proposed process model is the AP. In some cases, it can be considered relatively difficult to build an AP that represents the behaviors of real testers, and even risky in terms of time and money investments. However, modern game AI algorithms are becoming more advanced and easier to implement, posing new opportunities in that area. There are also serious attempts at building general game playing AIs<sup>7</sup> that tackle this problem.

Additionally, with our experimental study we have shown that even a rough approximation of a player can be sufficient within the balancing process. About a third of the automatically detected configurations were considered to be enjoyable by a human player, in spite of the fact that the AP and the tester had different play styles.

Another point of concern are the relatively high upfront costs for the initialization of the process. The process starts to be productive only after all its components, including AP and Balancing Environment, are set up. Additionally, if, like for the ZVG, there is no clear indication of what optimization algorithm to use, a statistical approach to meta-optimization might be necessary. Thus, putting too much emphasis on such initialization tasks can result in having no time for the balancing itself. One way to cope with the risk is the development of a technological standard for game balancing like the process model we suggested. The standard should define modularized components for the process, thus some parts can be transferred across games, decreasing time and costs for the setup phase.

The amount of time necessary for the balancing process also heavily depends on the time it takes to simulate a game and the need to simulate one configuration multiple times in case of non-deterministic games. Besides exploiting the possibility to parallelize these simulation runs, a possible remedy for this could be to restrict the balancing process to parts of a game, e.g. the fighting. Another approach could be to integrate a simplification of the game, as already often done in industry, or other, more general, surrogate models in the optimization algorithm. In case of the relatively simple ZVG, the optimization runs proved to be much faster than the attempts at manual balancing.

<sup>7</sup><http://www.gvgai.net/>

Besides these possible concerns, the experiment was also able to highlight some advantages of automated game balancing. For the ZVG, the approach proved helpful during the phase of game identification, providing insights about the game and its parameters. Automated balancing delivered a larger set of satisfactory configurations in shorter time compared to manual balancing. Furthermore, as known from other applications of metaheuristic based optimization, automation often generates unsuspected solutions. In more than one instance, our play testers were convinced that a seemingly good parameter configuration from automated balancing would not be playable, but were then surprised that they were actually fun to play.

A great benefit of automated game balancing is that the setup is robust to any changes within the game, since the optimization process does not need to be adjusted when the game changes. It makes this approach interesting for long lasting games with continuous patches and releases. This characteristic could be successfully exploited during our work, when we added a new element (resources) to the game mechanics. After the AP and the goal were updated accordingly, all the functionality of the balancing environment was reused in order to find good parameter configurations.

## VI. CONCLUSION AND OUTLOOK

Game Balancing is a time-consuming, yet necessary process in order to make a satisfying game and keep players happy. Since there are many different interpretations of what balancing is, we proposed our own definition to align the term with our integrated balancing process. This process with its formal representation shown in Figure 1 is the main contribution of the paper, whereas its application to a simple game demonstrates its usefulness in practice. While we strive for an automated way of balancing, manual game balancing is incorporated in our approach. It is the de-facto standard of how balancing is done in the game industry and can not be disregarded when designing games for human players. Furthermore, both manual and automated balancing deliver data which can be analyzed and compared.

While there are still limitations regarding automated game balancing, the numerous resulting configurations can be used to supplement the manual balancing process as we have shown for the ZVG. The used player AIs are quite simple but also fast and easy to implement. Nevertheless, the results in our case were very satisfactory, making a strong case for the practicability of our integrated balancing approach.

Practitioners argued [13], that this approach may also be used as an educational tool for game designers. The produced data can help them identify relationships between different parameters. Until now, game designers often rely on "gut feeling" for this task. However, the acquired data may be used to formulate experience (or tacit knowledge) into explicit knowledge about a game. Furthermore, it was argued that the approach could first be used as a support tool for a game designer and later be extended, increasing the level of embeddedness into the game design process.

A potential first step for future research is to increase the complexity of the used game, i.e. increase the number of parameters used for optimization. At the same time, several fitness functions may be envisioned in order to allow for multi-objective optimization. With multiple balancing goals, different player types can be accounted for. Other research streams can go into the direction of developing more accurate player AIs and analyzing the processes in the game industry with regard to the general game development and different possible applications of automated balancing approaches.

The presented integrated process for game balancing is intended as a proof-of-concept, to be extended as discussed above, possibly including a more extensive user and expert (game designers) study.

#### REFERENCES

- [1] Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, and Thomas Stützle. “F-Race and Iterated F-Race: An Overview”. In: *Experimental Methods for the Analysis of Optimization Algorithms*. Ed. by T. Bartz-Beielstein, M. Chiarandini, Luis Paquete, and M. Preuss. Springer, 2010, pp. 311–336.
- [2] Cameron B. Browne. “Automatic Generation and Evaluation of Recombination Games”. PhD Thesis. Queensland University of Technology, Brisbane, Australia, 2008.
- [3] Michele Chinosi and Alberto Trombetta. “BPMN: An Introduction to the Standard”. In: *Computer Standards & Interfaces* 34.1 (2012), pp. 124–134.
- [4] David E. Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.
- [5] Guy Hawkins, Keith V. Nesbitt, and Scott Brown. “Dynamic Difficulty Balancing for Cautious Players and Risk Takers”. In: *International Journal of Computer Games Technology* 2012 (2012), pp. 1–10.
- [6] Aaron Isaksen, Dan Gopstein, Julian Togelius, and Andy Nealen. “Discovering Unique Game Variants.” In: *Computational Creativity (ICCC 2015)*. Ed. by H. Toivonen et al. Brigham Young University, Provo, Utah, 2015.
- [7] Alexander Jaffe. “Understanding Game Balance with Quantitative Methods”. PhD Thesis. University of Washington, Seattle, WA, 2013.
- [8] Markus Kemmerling and Mike Preuss. “Automatic Adaptation to Generated Content via Car Setup Optimization in TORCS”. In: *IEEE Conference on Computational Intelligence and Games*. IEEE Press, Piscataway, NJ, 2010, pp. 131–138.
- [9] Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. “Sentient Sketchbook: Computer-Aided Game Level Authoring”. In: *8th International Conference on the Foundations of Digital Games*. 2013, pp. 213–220.
- [10] Tobias Mahlmann, Julian Togelius, and Georgios N. Yannakakis. “Evolving Card Sets Towards Balancing Dominion”. In: *Proceedings of IEEE Congress on Evolutionary Computation (CEC)*. IEEE Press, Piscataway, NJ, 2012.
- [11] Mark J. Nelson and Michael Mateas. “Towards Automated Game Design”. In: *Artificial Intelligence and Human-Oriented computing*. Springer, Berlin, Germany, 2007, pp. 626–637.
- [12] Mike Preuss, Julian Togelius, and Antonios Liapis. “Searching for Good and Diverse Game Levels”. In: *IEEE Conference on Computational Intelligence and Games*. IEEE Press, Piscataway, NJ, 2014, pp. 381–388.
- [13] Reynald François. *Weld Process in Design Formulas*. presentation at WWU Münster, Germany, 2016.
- [14] Jesse Schell. “Game Mechanics Must be in Balance”. In: *The Art of Game Design: A Book of Lenses*. Morgan Kaufmann, Burlington, MA, 2008. Chap. 11, pp. 171–206.
- [15] Adam M. Smith, Eric Butler, and Zoran Popović. “Quantifying over Play : Constraining Undesirable Solutions in Puzzle Design”. In: *8th International Conference on the Foundations of Digital Games*. 2013, pp. 221–228.
- [16] Adam M. Smith and Michael Mateas. “Variations Forever: Flexibly Generating Rulesets from a Sculptable Design Space of Mini-Games”. In: *IEEE Conference on Computational Intelligence and Games*. IEEE Press, Piscataway, NJ, 2010, pp. 273–280.
- [17] Gillian Smith, Jim Whitehead, and Michael Mateas. “Tanagra: Reactive Planning and Constraint Solving for Mixed-Initiative Level Design”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.3 (2011), pp. 201–215.
- [18] Julian Togelius, Mike Preuss, Nicola Beume, Simon Wessing, Johan Hagelbäck, Georgios N. Yannakakis, and Corrado Grappiolo. “Controllable Procedural Map Generation via Multiobjective Evolution”. In: *Genetic Programming and Evolvable Machines* 14.2 (2013), pp. 245–277.
- [19] Vanessa Volz, Günter Rudolph, and Boris Naujoks. “Demonstrating the Feasibility of Automatic Game Balancing”. In: *Genetic and Evolutionary Computation Conference (GECCO 16)*. ACM Press, New York, NY, 2016, accepted for publication.

# Evolutionary Deckbuilding in HearthStone

Pablo García-Sánchez\*, Alberto Tonda†, Giovanni Squillero‡ Antonio Mora§ and Juan J. Merelo\*

\*Dept. of Computer Architecture and Computer Technology. University of Granada, Granada, Spain.

Email: pablogarcia@ugr.es, jjmerelo@geneura.ugr.es

†UMR 782 GMPA, INRA. Thiverval-Grignon, France. Email: alberto.tonda@grignon.inra.fr

‡Politecnico di Torino, Torino, Italy. Email: giovanni.squillero@polito.it

§Department of Software Engineering. University of Granada, Granada, Spain. Email: amorag@ugr.es

**Abstract**—One of the most notable features of collectible card games is deckbuilding, that is, defining a personalized deck before the real game. Deckbuilding is a challenge that involves a big and rugged search space, with different and unpredictable behaviour after simple card changes and even hidden information. In this paper, we explore the possibility of automated deckbuilding: a genetic algorithm is applied to the task, with the evaluation delegated to a game simulator that tests every potential deck against a varied and representative range of human-made decks. In these preliminary experiments, the approach has proven able to create quite effective decks, a promising result that proves that, even in this challenging environment, evolutionary algorithms can find good solutions.

## I. INTRODUCTION

Collectible Card Games (CCGs) have been part of the mainstream gaming culture since the 90s, when *Magic: the Gathering*<sup>TM</sup> first became popular. Such games have suffered a recent growth thanks to *HearthStone: Heroes of Warcraft*<sup>TM</sup> [1], a game that, thanks to a very effective Free-To-Play model, reached a record of 40 million registered accounts in 2016 [2].

The common objective in a wide set of turn-based card games is to beat the opponent by using on him different types of cards (such as *spells* or *minions*). In CCGs every player is asked to construct a specific deck before the actual match. As the cards include specific rules that deeply affect the interaction between players, building a deck promotes an interesting and rich game play.

These kind of games are an interesting test bed in AI research, as players need to deal with hidden information and randomness, with the combination of states, rules and cards that may imply complex or unpredicted reactions, such as *combos*, combination of card so explosive that probably have not been anticipated even by the creators of the game.

Several authors have applied diverse computational intelligence methods to a variety of problems related to this field. For example, Cowling et al. compared different Monte Carlo tree search methods to deal with the imperfect information of the *Magic: the Gathering* game, obtaining better results than an expert rule-based agent [3]. CCGs have also been used as an example of the application of a framework to automatically detect design issues of new games [4].

However, previous works dealt with the AI aspects of the game in terms of automatic playing and behavior design. Being the construction of the deck a very important part of this kind of games, where players may spend hundred of dollars in

buying cards, it is quite surprising the lack of works in the literature proposing computational methods for automatic deck generation and analysis of their effectiveness. The only work related with this topic is [5], where the authors evolved decks of the game *Dominion* in order find methods to balance the game. However, they were limited to select 10 cards for each deck, extracted from a pool of 25, while in *HearthStone*, each deck requires 30 cards from a collection more than 600.

These techniques can also be interesting for CCGs manufacturers or developers, as adding new sets of cards may unbalance the game. Balancing games is a complex task, as new cards can affect previous rules, as well as all the possible combinations of card effects [6].

This paper proposes a methodology to automatically create decks for CCGs using an *Evolutionary Algorithm* (EA), an optimization technique loosely inspired by natural evolution. In EAs, potential solutions are encoded in a suitable format, and an objective function called *fitness* is automatically optimized [7]. EAs have already been extensively used in AI generation for videogames [8], [9], [10].

EAs are commonly used in combinatorial problems, as they commonly produce very effective combinations of elements, yet quite different from what a human expert would do. In the current framework, it makes possible to obtain competitive decks from scratch, i.e. without adding human knowledge. The proposed approach encode the candidate decks as vectors. The fitness function used to drive the evolutionary process is based on a series of actual matches against properly selected opponents. The resulting statistics are then analyzed and parsed to obtain a numerical metric.

The rest of the paper is structured as follows: after some background in CCGs and Evolutionary Algorithms, the proposed approach is described in Section III. After the experimental setup (Section IV), the results are discussed in Section V. Finally, the Conclusions and future lines or work are addressed.

## II. BACKGROUND

In this section we present some preliminary concepts that will help the reader to better understand the work.

### A. Collectible card games

The field of CCGs, that exploded with *Magic: the Gathering* in 1993, over time developed a specific terminology. There is

a set of shared concepts in this field: *deckbuilding* (decks can be prepared by the player, following certain rules); *competitive play* (the objective is to defeat the opponent); *card costs* (players have limited resources available every turn, and playing each card consumes some of these resources). While there are many variations on CCGs, ranging from cooperative play to games with no card costs, the popular ones — namely *Magic*, *HearthStone*, and *Yu-Gi-Oh!* — include all these concepts.

1) *Deck types*: Some of these terms are referred to the type of decks:

- *Aggro*, short for “aggression”, is a deck driven by a relatively simple strategy: the player attempts to finish the game in its early stages, quickly consuming lots of resources to inflict the maximum possible damage to the opponent. Typically, if a player with an Aggro deck cannot end the game fast enough, he will eventually lose in the mid or late game.
- *Combo* is a deck where player’s main objective is to survive until he manages to draw all the necessary pieces of a combination. Combos usually include two or more synergistic cards that allow the player to unleash a considerable amount of damage (ideally lethal) over the span of a single turn, securing the game. Players with these decks may lose if the opponent is able to produce a significant attack before all the pieces of the combination are gathered, or if the opponent is prepared to somehow counter it.
- *Control* is a deck chosen to keep the opponent in check, neutralizing early-game threats to prolong the match until the late game, where they can finish off using high-cost, high-value cards. Players with Control decks risk losing if they cannot find good answers for the cheap, effective threats of Aggro decks, or if they fail to counter the lethal combinations of Combo decks.

2) *Metagame*: The term *metagame* is used to describe conceptually difficult activities associated with game play, perceived by players as ‘peripheral’ to the game itself, but important to the whole game experience. Concretely, in the context of CCGs, *metagame* indicates the types of decks that a player entering a specific competitive event (or ladder) is expected to find, in largest numbers. Or in other words, ‘what everyone else is playing’ [11].

## B. HearthStone

Launched in 2013, *Hearthstone: Heroes of Warcraft* is an online CCG, developed by Blizzard Entertainment. Players compete against each other, trying to reduce the enemy health from 30 to 0 points, building their decks from a pool of cards that is constantly increasing, when either expansion packs or single-player adventures that reward the player with collectible cards upon completion are published. Currently, there are 743 unique collectible cards in the game, with more planned to be added in the future through additional content. Every card has an associated probability to be obtained when the player buys an envelope, being related with its power: common, rare, epic and legendary.

Cards in *HearthStone* fall into two main categories: *spells* and *minions*. Spells are played, create an effect on the battlefield, and then are discarded. Minions, on the other hand, stay in play, and can be used to attack the enemy Hero or other minions. Each card has a cost, that is paid when the card is played, using *crystals* (also called *mana*), a resource that grows every turn. On their first turn, players can use a total of one crystal, on the second turn they are allotted two crystals, and so on, to a maximum of ten crystals for turns ten and later. The cost is used for balance: powerful cards have a higher cost, cheaper cards are not as effective. A deck has to feature cards of all costs, in order to be able to play effectively in the early, mid and late game.

In *HearthStone*, deckbuilding is further constrained by the *Hero* the player chooses: each Hero features a special power that can be activated during the game, and exclusive cards that can only be used for that Hero. There are currently 9 different types of Hero: Druid, Hunter, Mage, Paladin, Priest, Rogue, Shaman, Warlock and Warrior. Even if it is theoretically possible to build an Aggro/Combo/Control deck using each Hero, in practice most Heroes are more suited to a single deck type. For example, the Priest’s ability and exclusive cards make it a very powerful choice for Control (with several variations of Priest Control decks), but a poor one for Aggro.

Figure 1 shows a screenshot of a match confronting a Hunter versus a Mage.



Fig. 1. Screenshot of a *HearthStone* match.

## C. Evolutionary algorithms

Evolutionary algorithms (EAs) [7], [12] are bio-inspired meta-heuristics that can be effectively used to find nearly optimal solutions for optimization problems. Usually an EA starts by generating a set of random solutions, called *population*, following a user-defined description. Then, it evaluates each candidate solution, called *individual*, assigning it a *fitness* value, that describes how good the individual is, with regards to the target problem. New solutions are then generated by the application of *operators* that either mutate a single existing solutions or recombine different existing solutions.

After each iteration, called *generation*, the least fit individuals are removed, and the process continues until a user-defined stop condition is met.

What makes EAs particularly interesting is their ability to manipulate complex structures such as binary trees or graphs [13]; and their relying only upon the fitness values, that can be provided by black-box evaluation, with no need of assumptions of regularity or stochasticity of the search space. For all these reasons, EAs have been already successfully employed in game design, for example evolving the parameters of an agent that play RTS such as *Planet Wars* [8] or generating the strategy of a *StarCraft*<sup>TM</sup> bot [14], and even the automatic creation of card games [15].

### III. PROPOSED APPROACH

In this work, we propose to use an EA to optimize the deck for a specific metagame. The EA initially generates random decks, and then mutates and combines the most promising ones to generate new solutions. Candidate decks are evaluated using an AI capable of playing *HearthStone*, against a set of representative human-designed decks that define the target metagame. Their fitness is tied to the total number of victories obtained. We will go into details of the different aspects of the approach next.

#### A. Candidate solutions

Solutions in our problem are decks: following *HearthStone*'s rules, a deck has to be composed of exactly 30 cards, with no more than two copies of each, or exactly one copy in the case of Legendary cards. Decks can include both Neutral cards and those reserved to a single specific Hero.

#### B. Fitness function

As the evolutionary algorithm can freely manipulate decks, swapping any card for any other, crossing two decks and so on, it is possible that it will obtain decks that violate the rules of the game: for example, by having more than 2 copies of the same card, or more than 1 copy of a Legendary one. Also, while the total number of victories obtained is important, at the same time we desire a deck with a fair chance to win against all decks in the metagame, and not one that mercilessly slaughters specific opponents and loses badly against other ones. Also, and due to the stochastic nature of the game, a single execution of a game against a deck would not be statistically significant [16], so for each opponent at least 15 games should be played. For these reasons, the fitness function is divided into three parts, evaluated following a lexicographical order:

- 1) **Correctness:** this metric takes into account the number of errors in the decklist (repeated cards). decks that have this fitness value bigger than 0 are not evaluated further, and all their remaining fitness values are set to the lowest possible amount. This fitness value is to be minimized.
- 2) **Victories:** straightforwardly, this is the total number of victories obtained by the decklist played 16 times against each of the decks in the target metagame. This fitness value is to be maximized.

TABLE I  
PARAMETERS USED BY THE EA. THE ACTIVATION PROBABILITIES OF THE OPERATORS ARE SELF-ADAPTED. FOR MORE INFORMATION ON THE PARAMETERS, SEE [17] OR VISIT  
[HTTPS://SOURCEFORGE.NET/P/UGP3/WIKI/HOME/](https://sourceforge.net/p/ugp3/wiki/HOME/).

Parameter	Meaning	Value
$\mu$	Population size	10
$\lambda$	Operators applied	10
$\alpha$	Self-adapting inertia	0.9
$\sigma$	Initial mutation strength	0.9
$\tau$	Size of the tournament selection	[2-4]
$G$	Number of generations	50
$R$	Replacement mechanism	Generational
$e$	Number of parry decks	8
$t$	Number of games per parry deck	16
Operators used	singleParameterAlterationMutation onePointCrossover twoPointCrossover	

- 3) **Standard deviation:** this value is computed by evaluating the number of victories obtained against each opponent, and computing the standard deviation with regards to the number of victories against other opponents. If the deck obtains the same number of victories against all opponents, its standard deviation will be optimal. This fitness value is to be minimized.

This type of lexicographical fitness, using different parry opponents has been successfully used in previous works [14], [8].

### IV. EXPERIMENTAL EVALUATION

This section describes the algorithm used and the decisions taken into account to model the fitness function.

#### A. Evolutionary algorithm

The EA used in the experience is  $\mu GP$ , a general-purpose evolutionary framework [17], designed to easily implement different optimization problems out-of-the-box, thanks to its flexible definition of individual structure and external evaluator. The project is available on SourceForge<sup>1</sup>. During all the experiments,  $\mu GP$  has been configured with the parameters reported in Table I. The evolutionary operators collectively allow the EA to replace a card with any other card and cross over two decks.

#### B. MetaStone

MetaStone is an open-source *HearthStone* simulator<sup>2</sup>. It allows the manual creation of decks using the cards available in *HearthStone* and simulate games between decks, obtaining several statistics, such as turns taken or the damage done. Different heuristics can be selected for the AI engine, based on a score given to the actions that are evaluated in each turn, taking into account a combination of weights of the type of minions/spells used.

- Play Random: each turn the actions (moves) to play are selected randomly.

<sup>1</sup><http://ugp3.sourceforge.net/>

<sup>2</sup><https://github.com/demilich1/metastone>

- Greedy Optimize Move: in each turn the AI selects each move ordered by score.
- Greedy Optimize Turn: in each turn the AI selects the combination of all possible moves with the higher score.
- Flat MonteCarlo Tree: during a certain number of iterations the AI simulates random moves until possible ends of the match to calculate the score.

### C. Opponents decks

For the experimental evaluation, we consider the metagame of Season 18 of *HearthStone* competitive play, featuring the base set, the adventures *Curse of Naxxramas* and *Blackrock Mountain*, and the expansions sets *Goblin vs Gnomes* and *The Grand Tournament*, that overall include 694 cards. We have chosen this set of cards because it is the one used in the last season before the metagame changed to current (and still changing) one: just before the newest expansion (*League of Explorers*) appeared. This season has a good representation of different deckbuilding strategies, and we selected 4 representative human-designed Aggro decks (Hunter, Mage, Paladin, Shaman), 3 Control (Priest, Warrior, Warlock) and 2 Combo (Druid, Rogue).

The considered decks, summarized in Table II, have been taken from the website of Tempo Storm<sup>3</sup>, an American e-sports professional video game team, and selected among the ones able to reach the highest rank in the competitive ladder during season 18.

### D. Opponent decks analysis

In order to get an estimate of how well MetaStone can play the human-designed decks, we run a first tournament, where each deck was paired against every other for 256 games, using all combinations of the 4 possible AIs. Thus, 11520 games were played. From that results we discovered that the AI GreedyOptimizeTurn obtained the best percentage of victories, winning 4320 games out of the 11520 (37.5%). Therefore, we set this AI as the one to bet during the rest of the experiments. Focusing on the deck behavior using this AI, Table III shows the win ratio of each one.

### E. Experimental results

As the fitness evaluator requires a lot of computational time to simulate the large number of games for each individual, every execution of the algorithm requires several days. However, as this is a proof-of-concept, we have performed two preliminary experiments, limiting each run to a different set of cards. The first is aimed at evolving a Mage deck, normally played as a control deck, and the second is focused on a Hunter deck, usually played as aggro. In both experiments, each candidate decklist was played  $t$  times (16) against every human-designed deck in the metagame, with the exception of the deck featuring the same Hero ( $e=8$ ), so each individual is tested 128 times in each evaluation.

<sup>3</sup><https://tempostorm.com/articles/meta-snapshot-18-from-warrior-to-warrior>

TABLE II  
CONSIDERED DECKS FOR THE EVALUATION.

Deck name	Type	Description
MidRange Druid	Combo	Aims to stall for time in the early game, and slowly build a <i>ramp</i> that increase the resources faster than the opponent's to use a combination of 2 specific cards that inflict 14 to 30 damage.
MidRange Hunter	Aggro	Trades cheap cards for cost-effective minions that are harder to remove, and thus more difficult to deal with for Control decks.
Mage Tempo	Aggro	Uses cards that are able to improve one's progression, while at the same time slowing down the opponent, making enemy minions unusable for one or more turns.
Aggro Paladin	Aggro	Swarms the battlefield with a lot of weak but cheap minions.
Shadow Madness Priest	Control	Tries to switch between curing and dealing damage depending on board conditions, to keep the match under control.
Oil Rogue	Combo	Slowly builds a large hand of cards, to finally unleash lethal damage in one single turn.
Mech Shaman	Aggro	Uses minion of the type <i>mech</i> . They are harder for the opponent to deal with, and interact nicely with each other, as some Mechs provide bonuses to all other Mechs in play.
Warlock MalyLock	Control	Tries to obtain the card Malygos as soon as possible. This card increases the amount of damage dealt by all of the player's spells by a large quantity, allowing the Warlock to quickly win the game after it appears.
Warrior Control	Control	Tries to use Armor to survive the early game, removing the most pernicious threats, while waiting for powerful, expensive minions that will be extremely effective in the late game.

TABLE III  
NUMBER OF GAMES WON BY THE GREEDY OPTIMIZE TURN AI. EACH DECK SHOWN IN THIS TABLE PLAYED A TOTAL OF 256 MATCHES.

Deck name	Games Won	Games Lost	Win/Lose ratio
Aggro Paladin	182	74	0.7109
Mage Tempo	177	79	0.6914
Shadow Madness Priest	152	104	0.5937
Midrange Hunter	143	113	0.5585
Mech Shaman	119	137	0.4648
Oil Rogue	106	150	0.4140
Control Warrior	104	152	0.4062
Midrange Druid	85	171	0.3320
Warlock MalyLock	83	173	0.3242

## V. DISCUSSION

The proposed approach is proven able to discover decks with a satisfying win ratio against competitive human-designed decks in the target metagame. Figure 2 shows the evolution of victories of the best, the average and worst individuals in each generation, showing the fitness improvement during the evolution. In both cases (Mage and Hunter) the final win ratio outperformed the Mage Tempo and Midrange Hunter decks from season 18, respectively: the best evolved Mage wins 71.87% of the matches (vs Mage Tempo, 69.14%) and the best evolved Hunter wins 57.81% of the matches (vs Midrange Hunter, 55.85%).



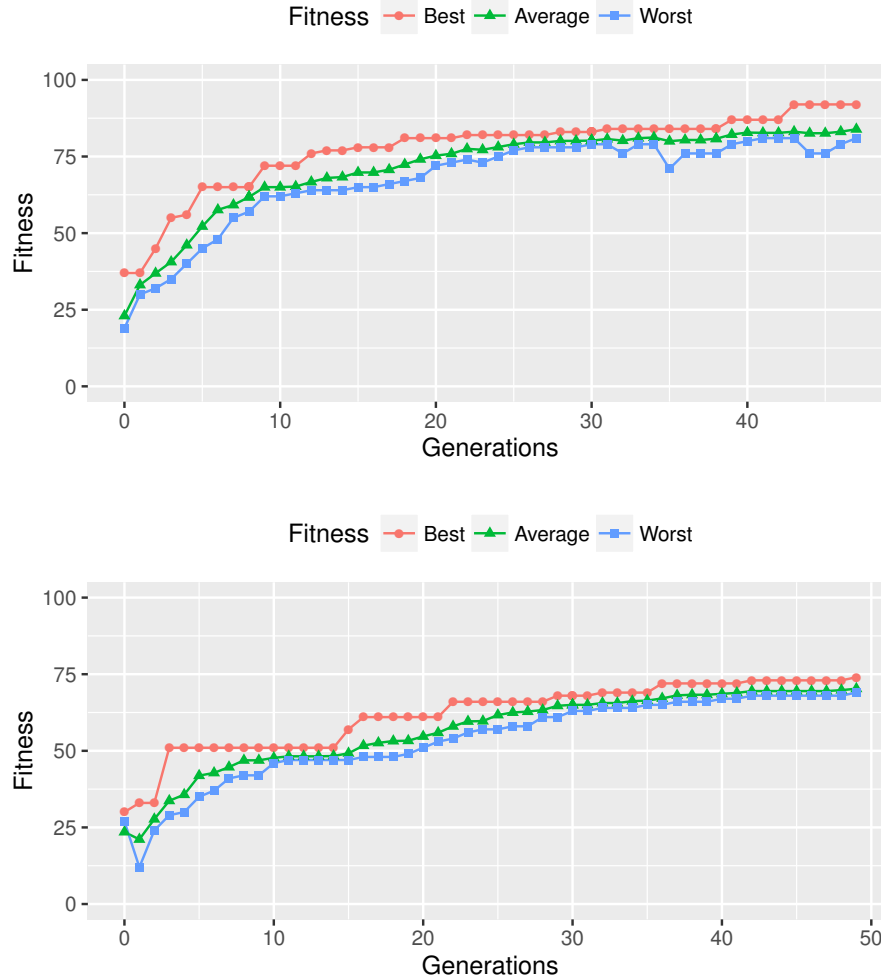


Fig. 2. Evolution of the number of victories in the population during the experiments for Mage (top) Hunter (bottom).

As the fitness evaluation is dependent on the MetaStone AI, however, our methodology might incur in *overfitting* with regards to the AI capabilities and playing style. By looking at the results of the preliminary tournament among the human-designed decks in Table III, it is immediately evident that MetaStone can use some decks (and some playing styles) better than others: for example, the *Mid-Range Druid* and *Malylock* decks, while pretty effective in the hand of an experienced human player, have relatively low performances when played by MetaStone. The human strategy for both decks relies upon waiting for specific cards (*Force of Nature*+*Roar* for the Druid, *Malygos* for the Warlock) and play them at the right moment, which is something that MetaStone might not be capable of. For this reason, we deem it useful to perform an expert card-by-card analysis of the deck found by the evolutionary approach, to understand whether the deck contains cards (and cards combinations) that are considered powerful by humans, or rather that MetaStone could play more effectively. The expertise comes from one of the authors, an average competitive HearthStone player, able to reach rank 10

in the season ladder (ranks 1-10 contain more or less the top 10% of the registered users [18]), that has played over 7,000 matches since the Open Beta of the game.

#### A. Evolved Mage deck

Figure 3 contains the best Mage decklist obtained at the end of the process. Mana curve (a histogram of the number of cards grouped by cost, shown in Figure 4) shows that the deck is clearly *Aggro*, using several small, effective minions as early threats (*Clockwork Gnome*, *Fallen Hero*, *Leper Gnome*, *Razorfren Hunter*); blocking the opponent's minions in the mid-game through so-called *freeze* spells (*Blizzard*, *Frost Nova*), that prevent hit minions from acting during the next turn, or directly wiping the board with *Flamestrike*; and finally attempting to finish off the game through large minions and powerful spells (*Baron Geddon*, *Fireball*, *War Golem*, *Pyroblast*).

There are a few remarkable properties of the evolved deck, that we are going to describe in more detail. First of all, the majority of cards appear in two copies, the maximum

TABLE IV

STATISTICS OF THE BEST INDIVIDUAL USING MAGE CARDS, AGAINST ALL THE HUMAN-DESIGNED DECKS (16 TIMES PER DECK). WIN RATES OF THE MAGE TEMPO SEASON 18 (MTS18) DECK IS ALSO SHOWN AS COMPARISON.

	Druid	Hunter	Paladin	Priest	Rogue	Shaman	Warlock	Warrior
% of wins of MTS18	90.625	59.375	34.375	53.125	78.125	78.125	87.5	71.875
% of wins of Evolved Mage	87.5	62.5	50	62.5	81.25	81.25	93.75	56.25
Damage Dealt	37.25	57.13	60.50	76.81	48.50	66.75	60.44	65.00
Healing Done	1.50	5.50	4.00	4.50	5.00	5.00	4.00	4.00
Mana Spent	22.94	31.31	25.69	46.06	37.56	39.06	41.69	44.69
Cards Played	9.06	11.63	11.31	16.69	12.94	15.44	16.69	16.38
Turns Taken	7.31	8.50	7.81	10.38	9.25	9.88	10.06	10.44
Cards Drawn	7.31	8.50	7.81	10.38	9.25	9.88	10.06	10.44
Minions Played	5.50	6.19	5.38	8.19	6.88	7.75	7.25	7.81
Spells Cast	2.06	2.94	2.81	5.19	3.63	4.00	5.19	4.69
Hero Power Used	1.50	2.50	3.13	3.31	2.44	3.69	4.25	3.88
Weapons Equipped	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

MINIONS	SPELLS
<b>Antique Healbot</b> (C)	<b>Blizzard</b> (R)
<b>Antique Healbot</b> (C)	<b>Blizzard</b> (R)
Argent Commander (R)	<b>Dragon's Breath</b> (C)
<b>Baron Geddon</b> (L)	<b>Fireball</b> (C)
Clockwork Gnome (C)	Flame Lance (C)
Clockwork Gnome (C)	<b>Flamestrike</b> (B)
Clockwork Knight (C)	<b>Flamestrike</b> (B)
<i>Coliseum Manager</i> (R)	<b>Mirror Image</b> (B)
Dancing Swords (C)	Polymorph Boar (R)
Fallen Hero (R)	Polymorph Boar (R)
Flesheating Ghoul (C)	<b>Pyroblast</b> (E)
<i>Gormok The Impaler</i> (L)	
Imp Master (R)	
<b>Leper Gnome</b> (C)	
<b>Leper Gnome</b> (C)	
<i>Razorfen Hunter</i> (B)	
<i>Razorfen Hunter</i> (B)	
War Golem (B)	
<b>Water Elemental</b> (B)	

Fig. 3. The best decklist obtained through the evolutionary approach for Mage. Cards that are considered particularly powerful by a human expert are highlighted in **bold**. Cards that are considered sub-optimal are in *italics*. The rarity of each card is also marked as (in decreasing order of rarity) Legendary (L), Epic (E), Rare (R), Common (C), Basic (B).

number allowed, even if there is no explicit pressure to have this configuration in the fitness function. The evolutionary algorithm autonomously discovered that possessing a higher number of copies of some cards is better, since it makes the deck more reliable. Secondly, a considerable percentage of cards appearing in the deck have been often used in the competitive ladder, with the exception of *Coliseum Manager*, considered a sub-optimal minion, *Razorfen Hunter*, that has several strong competitors in the same niche, and *Gormok the Impaler*, which is sometimes used but considered very circumstantial by the players.

Finally, the deck includes several interesting synergies. *Antique Healbot*, *Clockwork Gnome*, *Clockwork Knight*: these are all minions of type *Mech*, and *Clockwork Knight* is able to boost other *Mechs*. *Imp Master*, *Gormok the Impaler*, *Razorfen Hunter*: *Gormok* is a Legendary minion with a powerful ability that rarely activates, since it requires the presence of at least

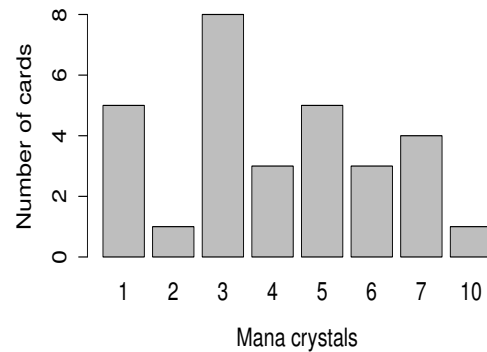


Fig. 4. Mana curve of the evolved Mage deck.

other 4 other minions on your side of the field; the other cards all spawn extra minions on the battlefield, making it easier to activate *Gormok*. *Flesheating Ghoul*, that increases its strength every time a creature on the board dies, works very nicely with spells able to wipe the board such as *Blizzard* and *Flamestrike*.

Table IV presents results and play statistics of the evolved decklist against each one of the human-designed decks. The deck is able to win reliably against most of the opponents, being particularly effective against Warlock, Shaman and even Paladin (the highest ranking deck in the preliminary evaluation). On the other hand, the toughest match-ups seems to be Hunter and Warrior: the former is probably often able to out-run the Mage deck in a damage race; the latter is more of a control deck relying on large, dangerous minions that are hard to deal with for the Mage deck.

#### B. Evolved Hunter deck

Figure 5 showcases the best Hunter decklist obtained at the end of the evolutionary process. Again, the deck is clearly Aggro (see Figure 6), but this time it exploits a relatively large selection of creature-removal spells, that can probably be used to control the field in the mid-game. Interestingly, the deck exploits either minions with a low cost (*Gadgetzan Joust*;

*Jungle Panther*, *Lance Carrier*), or with a large cost (*Gazlowe*, *King Krush*, *Piloted Sky Golem*, *Sneed's Old Shredder*), while featuring lots of spells with intermediate cost (*Multi-shot*, *Cobra Shot*, *Deadly Shot*, *Powershot*).

MINIONS	SPELLS
<b>Annoy-o-tron</b> (C)	<b>Animal Companion</b> (B)
<b>Annoy-o-tron</b> (C)	<b>Animal Companion</b> (B)
Blackwing Technician (C)	Arcane Shot (B)
<i>Captain Greenskin</i> (L)	<i>Bestial Wrath</i> (E)
<b>Defender Of Argus</b> (R)	Flare (R)
<b>Defender Of Argus</b> (R)	<b>Kill Command</b> (B)
<b>Fel Reaver</b> (E)	<b>Unleash The Hounds</b> (B)
<b>Fel Reaver</b> (E)	WEAPONS
<i>Gilblin Stalker</i> (C)	Gladiator's Longbow (E)
Goldshire Footman (B)	<b>Glaivezooka</b> (C)
Goldshire Footman (B)	
<i>Hungry Crab</i> (E)	
<b>Kezan Mystic</b> (R)	
<b>Loatheb</b> (L)	
Metaltooth Leaper (R)	
Piloted Sky Golem (E)	
Raging Worgen (C)	
<i>Ship's Cannon</i> (C)	
<b>Sylvanas Windrunner</b> (L)	
Timber Wolf (B)	
Twilight Guardian (E)	

Fig. 5. The best decklist obtained through the evolutionary approach for Hunter. Cards that are considered particularly powerful by a human expert are highlighted in **bold**. Cards that are considered sub-optimal are in *italics*. The rarity of each card is also marked as (in decreasing order of rarity) Legendary (L), Epic (E), Rare (R), Common (C), Basic (B).

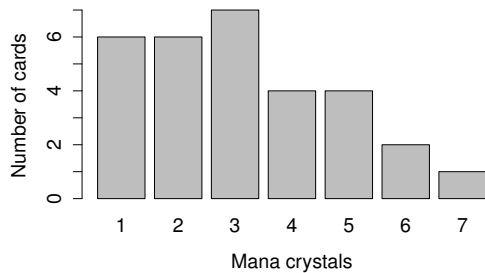


Fig. 6. Mana curve of the evolved Hunter deck.

Again, even without a specific pressure to do so, the algorithm found it useful to include double copies of several cards. And most of the cards have seen play in the competitive ladder: *Sylvanas Windrunner* and *Loatheb* are two Legendaries considered extremely powerful, *Animal Companion*, *Kill Command* and *Unleash the Hounds* are included in almost all Hunter decks, and the same can be said for one of the two weapons, *Glaivezooka*. *Annoy-o-Tron* and *Defender of Argus* are also pretty popular, albeit they are more used in decks featuring other Heros. *Fel Reaver* is perhaps the most surprising choice, being a large, cheap creature with a huge drawback: every time the opponent plays a card, *Fel Reaver* destroys the top three cards of the player's deck. The usual response to *Fel Reaver* is to play as many cards as possible, in

order to remove a huge part of its controller's deck; but maybe the MetaStone AI is not able to assess correctly the drawback, and thus in this environment *Fel Reaver* might be even more effective. The deck also features some questionable choices: *Hungry Crab*, *Goldshire Footman* and *Ship's Cannon* simply have too many better competitors in their respective niches; while *Bestial Wrath*'s effect is considered too circumstantial to be useful in competitive play.

Nevertheless, we can observe again some interesting synergies: *Kill Command* and *Bestial Wrath* are enhanced by minions of type Beast, and the deck has 4 of them (counting the spell *Animal Companion*, that puts a random Beast in play); *Metaltooth Leaper* boosts minions of type Mech, which the deck plays 6 of; *Flare* and *Kezan Mystic* are two cards that are particularly effective against spells used by Mage and Paladin decks, the top two of our preliminary evaluation; finally, even if not particularly cost-effective, *Captain Greenskin* can enhance both *Gladiator's Longbow* and *Glaivezooka*.

From the results in Table V, it is noticeable how the deck's performance is particularly good against Druid and Warlock, that are probably too slow to deal with the Hunter's aggression; while the worst match-ups are versus Priest, which is effective against low-strength minions, and Mage, another Hero whose cards are able to wipe the board, resulting in a big disadvantage for aggressive decks.

### C. Remarks

While the presented proof-of-concept seems promising, there are a few weak points that are worth discussing. The most evident issue lies in the fitness function: MetaStone is a good AI, but so far it cannot attain human-comparable levels of play, especially with the settings we used for the experiments, based on a greedy choice; thus, it is hard to tell whether the optimization process is discovering generally good decks, or good decks just *for this specific AI*. This issue is hard to solve, but the fact that the evolutionary process created a deck with cards considered good by human players is at least encouraging. In future works, we plan to perform a play-by-play analysis of selected games using the evolved decks, in order to better study the problem.

Another possible issue lies in our definition of the search space. Currently, the evolutionary algorithm is free to replace a card with any other card in the set, with a low chance of obtaining an improvement. It would probably be more sensible to include mutations able to transform a card into other cards with the same cost, or similar characteristics, as human players often do when considering modifications to a decklist. The presence of such mutations could potentially help smoothen the fitness landscape, driving the algorithm towards interesting areas more effectively.

## VI. CONCLUSIONS

In this paper we have presented a methodology for the automatic evolution of decks for collectible card games using *HearthStone* as a case study. An evolutionary algorithm is applied to the task. This EA uses as the structure of an

TABLE V

STATISTICS OF THE BEST INDIVIDUAL USING HUNTER CARDS, AGAINST ALL OF THE HUMAN-DESIGNED DECKS (16 TIMES PER DECK). WIN RATE OF THE MIDRANGE HUNTER SEASON 18 (MHS18) DECK IS ALSO SHOWN AS COMPARISON.

	Druid	Mage	Paladin	Priest	Rogue	Shaman	Warlock	Warrior
% of wins of MHS18	81.25	40.625	40.625	25	65.625	62.5	65.625	65.625
% of wins of Evolved Hunter	100	37.5	43.75	37.5	62.5	62.5	68.75	50
Damage Dealt	34.81	34.06	45.69	54.00	33.94	37.69	41.50	42.56
Healing Done	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Mana Spent	23.63	23.25	25.19	28.00	26.69	23.81	27.81	31.75
Cards Played	11.69	10.56	11.63	11.88	11.94	11.19	13.06	13.63
Turns Taken	7.81	7.44	7.56	7.94	7.81	7.56	8.25	8.75
Cards Drawn	7.94	7.69	7.75	7.94	7.69	7.69	8.69	9.13
Minions Played	6.00	6.00	6.13	6.94	6.44	6.00	6.75	7.31
Spells Cast	2.75	2.25	3.13	2.56	3.13	3.06	3.25	3.13
Hero Power Used	2.63	2.00	2.06	2.13	2.00	1.94	2.63	2.75
Weapons Equipped	0.31	0.31	0.31	0.25	0.38	0.19	0.44	0.44

individual a list of 30 cards, taken from the almost 700 available. The fitness function is the number of victories of the candidate deck against popular human-made competitive decks, performed through MetaStone, an AI able to play HearthStone. Two experiments have been conducted, and the proposed approach proved able to create a competitive Mage and Hunter deck for a specific real-world metagame, taken from Season 18 (the last one before the current, and still changing, metagame).

In future works, we plan to evolve decks for other Heroes, improve the evolutionary algorithm by adding context-aware mutations, and perform a play-by-play analysis of the decks, to try and assess the generality of our approach.

#### ACKNOWLEDGMENTS

The authors would like to thank github user @demilich1 for creating MetaStone. This work has been supported in part by TIN2014-56494-C4-3-P and TEC2015-68752 (Spanish Ministry of Economy and Competitiveness), MSTR (reference PRY142/14).

#### REFERENCES

- [1] "Hearthstone: Heroes of Warcraft," <http://eu.battle.net/hearthstone/>, accessed: 24 April 2016.
- [2] "Toucharcade: 'Hearthstone: Heroes of Warcraft' has more than 40 million registered players," <http://toucharcade.com/2016/02/11/hearthstone-heroes-of-warcraft-has-more-than-40-million-registered-players/>, accessed: 24 April 2016.
- [3] P. I. Cowling, C. D. Ward, and E. J. Powley, "Ensemble determinization in monte carlo tree search for the imperfect information card game magic: The gathering," *IEEE Trans. Comput. Intellig. and AI in Games*, vol. 4, no. 4, pp. 241–257, 2012.
- [4] J. C. Osborn, A. Grow, and M. Mateas, "Modular computational critics for games," in *Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE-13, Boston, Massachusetts, USA, October 14-18, 2013*, G. Sukthankar and I. Horswill, Eds. AAAI, 2013.
- [5] T. Mahlmann, J. Togelius, and G. N. Yannakakis, "Evolving card sets towards balancing dominion," in *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2012, Brisbane, Australia, June 10-15, 2012*. IEEE, 2012, pp. 1–8.
- [6] E. Ham, "Rarity and power: balance in collectible object games," *The International Journal of Computer Game Research*, vol. 10, no. 1, 2010.
- [7] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, ser. Natural Computing Series. Springer, 2015. [Online]. Available: <http://dx.doi.org/10.1007/978-3-662-44874-8>
- [8] A. M. Mora, A. Fernández-Ares, J. J. Merelo-Guervós, P. García-Sánchez, and C. M. Fernandes, "Effect of noisy fitness in real-time strategy games player behaviour optimisation using evolutionary algorithms," *J. Comput. Sci. Technol.*, vol. 27, no. 5, pp. 1007–1023, 2012.
- [9] N. Cole, S. J. Louis, and C. Miles, "Using a genetic algorithm to tune first-person shooter bots," in *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2004, 19-23 June 2004, Portland, OR, USA*. IEEE, 2004.
- [10] A. M. Mora, R. Montoya, J. J. Merelo-Guervós, P. García-Sánchez, P. A. Castillo, J. L. J. Laredo, A. I. M. García, and A. Esparcia-Alcázar, "Evolving bot AI in unreal<sup>tm</sup>," in *Applications of Evolutionary Computation, EvoApplications 2010: EvoCOMPLEX, EvoGAMES, EvoASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Istanbul, Turkey, April 7-9, 2010, Proceedings, Part I*, ser. Lecture Notes in Computer Science, C. D. Chio, S. Cagnoni et al., Eds., vol. 6024. Springer, 2010, pp. 171–180.
- [11] M. Carter, M. R. Gibbs, and M. Harrop, "Metagames, paragames and orthogames: a new vocabulary," in *International Conference on the Foundations of Digital Games, FDG '12, Raleigh, NC, USA, May 29 - June 01, 2012*, M. S. El-Nasr, M. Consalvo, and S. K. Feiner, Eds. ACM, 2012, pp. 11–17.
- [12] K. A. De Jong, *Evolutionary computation: a unified approach*. MIT press, 2006.
- [13] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. MIT press, 1992, vol. 1.
- [14] P. Garcia-Sanchez, A. Tonda, A. M. Mora, G. Squillero, and J. J. Merelo, "Towards automatic starcraft strategy generation using genetic programming," in *Computational Intelligence and Games (CIG), 2015 IEEE Conference on*, Aug 2015, pp. 284–291.
- [15] J. M. Font, T. Mahlmann, D. Manrique, and J. Togelius, "Towards the automatic generation of card games through grammar-guided genetic programming," in *International Conference on the Foundations of Digital Games, Chania, Crete, Greece, May 14-17, 2013*, G. N. Yannakakis, E. Aarseth, K. Jørgensen, and J. C. Lester, Eds. Society for the Advancement of the Science of Digital Games, 2013, pp. 360–363.
- [16] J. J. Merelo, F. Liberatore, A. Fernández-Ares, R. H. García-Ortega, Z. Chelly, C. Cotta, N. Rico, A. M. Mora, and P. García-Sánchez, "There is noisy lunch: A study of noise in evolutionary optimization problems," in *Proceedings of the 7th International Joint Conference on Computational Intelligence (IJCCI 2015) - Volume 1: ECTA, Lisbon, Portugal, November 12-14, 2015*, A. C. Rosa, J. J. Merelo-Guervós et al., Eds. SciTePress, 2015, pp. 261–268.
- [17] E. Sanchez, M. Schillaci, and G. Squillero, *Evolutionary Optimization: the  $\mu$ GP toolkit*. Springer Science & Business Media, 2011.
- [18] "Battle.net: Hearthside chat - you're better than you think!" <http://us.battle.net/hearthstone/en/blog/15955974/hearthside-chat-youre-better-than-you-think-9-18-2014>, accessed: 24 April 2016.

# Three Types of Forward Pruning Techniques to Apply the Alpha Beta Algorithm to Turn-Based Strategy Games

Naoyuki Sato

Japan Advanced Institute of Science and Technology  
Ishikawa, Japan  
Email: satonao@jaist.ac.jp

Kokolo Ikeda

Japan Advanced Institute of Science and Technology  
Ishikawa, Japan  
Email: kokolo@jaist.ac.jp

**Abstract**—Turn-based strategy games are interesting testbeds for developing artificial players because their rules present developers with several challenges. Currently, Monte-Carlo tree search variants are often utilized to address these challenges. However, we consider it worthwhile introducing minimax search variants with pruning techniques because a turn-based strategy is in some points similar to the games of chess and Shogi, in which minimax variants are known to be effective. Thus, we introduced three forward-pruning techniques to enable us to apply alpha beta search (as a minimax search variant) to turn-based strategy games. This type of search involves fixing unit action orders, generating unit actions selectively, and limiting the number of moving units in a search. We applied our proposed pruning methods by implementing an alpha beta-based artificial player in the Turn-based strategy Academic Package (TUBSTAP) open platform of our institute. This player competed against first- and second-rank players in the TUBSTAP AI competition in 2016. Our proposed player won against the other players in five different maps with an average winning ratio exceeding 70%.

## I. INTRODUCTION

Building competitive computer game players is one of the main themes in the field of artificial intelligence. As a result of much research, computer players are sufficiently competent to compete with professional human players in some traditional board games such as chess [1], Shogi (Japanese chess) [2], and Go [3]. On the other hand, there are games in which computer players are weaker than human players. Turn-Based Strategy (TBS) games constitute a genre of games that require additional research to increase the levels of competency of computer players.

TBS games require players to take turns moving their pieces (known as ‘units’ in TBS) when competing against each other in games such as chess and Shogi. However, TBS games contain rules that complicate the development of strong computer players.

For example, TBS games allow players to manipulate all their pieces (units) in whatever order they prefer in each turn. This rule provides players with a large number of possible actions per turn, which increases the number of nodes and edges for game tree search. Even the use of only six units, each of which has 10 possible actions, result in 720 million possible actions available to a player in a turn. As with other

examples, there are various game positions from which players start their games (known as ‘maps’ in TBS) and unit types have complex relationships to ensure competitiveness in combat. The rules make it difficult to design state evaluation functions with high precision and to apply supervised machine learning, which uses the game records of expert human players.

These difficulties are frequently overcome by adopting Monte-Carlo tree search variants for computer players in TBS games, whereas minimax search variants such as  $\alpha\beta$  search are rarely used. However, TBS games have basic game structures similar to those of chess and Shogi, for which  $\alpha\beta$  search is known to be effective. In addition, human players often look ahead by considering sequences of consecutive actions to decide their subsequent moves, as is the case with  $\alpha\beta$  search, by focusing only on the plausible actions for each position. Therefore, we tried to apply  $\alpha\beta$  search to TBS games and evaluated the performance.

We introduced three modifications to decrease the number of edges of the game tree in  $\alpha\beta$  search to allow us to increase the depth of the search sufficiently to achieve the desired performance. These modifications are as follows:

- **Fixing the order in which units are allowed to move**
- **Applying selective unit action generation**
- **Limiting the number of moving units in each search**

It is true that these modifications involve some risk of overlooking important or critical moves because these techniques are forward pruning. However, we believe that introducing an appropriate level of this pruning would result in an enhancement of the performance of artificial players in TBS games. Especially, the use of  $\alpha\beta$  search in TBS games often does not permit the player to look ahead by even one opponent move (i.e., the search cannot even reach 2 ply deep) in a practical amount of time positions without any pruning, because of the large number of possible actions in one turn. Thus, we expect pruning techniques to have a great effect on the performance, especially if they are capable of increasing the search depth to an extent that would enable them to consider the opponent’s future actions.

This paper is organized as follows. In Sect. II, we present work related to this research. Sect. III explains the TUBSTAP

platform we use in this research to evaluate the performance of the proposed method. Sect. IV describes the three types of modifications we adopt in this research. Sect. V to Sect. VII describe preliminary experiments to assess the respective effects of the three modifications. In Sect. VIII, we present the experiments in which our proposed computer player competes against a first- and second-rank player of the TUBSTAP AI competition in 2016. We discuss some future work in Sect. IX.

## II. RELATED WORK

### A. Minimax Tree Search and Pruning Techniques

Minimax tree search has been widely adopted to develop computer players in several games such as chess and Shogi. Sometimes developers adopted tree searches by removing certain branches of the tree to increase the search depth. Such techniques are known as pruning, and pruning techniques are categorized into two types: backward pruning and forward pruning. Backward pruning is pruning that does not affect the search result. The  $\alpha\beta$  algorithm is a well-known backward pruning technique [20].

On the other hand, forward pruning removes some game tree branches involving risks to affect the minimax value of the root node. For example, in a classic pruning method shown in a 1960s chess program, tapered n-best search searches [4] only the n best moves, which are decided by some move ordering techniques [5]. These best moves are considered and the other moves are pruned in the search.

More recent techniques often use the alpha/beta values in the  $\alpha\beta$  algorithm for forward pruning. For example, futility pruning [6] and null move pruning [7] estimate the upper/lower bounds of evaluation values obtained by moves. In this case, pruning discards moves with evaluation values that have little potential to exceed the alpha value or fall below the beta value. These techniques are used not only for chess but also for Shogi [9], thereby making it possible to search trees deeper than 10 plies.

### B. Turn-based Strategy

Considerable research has been performed on TBS games [12] [13] [14] for the “Cid Meier’s Civilization” series [10] or its clones. However, we consider the game system of this series to contain many complex factors other than combat by units, such as economy, research, and diplomacy. Thus, we focus on TBS games of which the systems concentrate on combat between units.

The following studies relate to TBS games of this type. A computer player with an evolutionary computational technique is proposed for the Advance Wars clone game [11]. A Monte-Carlo tree search algorithm [15] and upper confidence tree algorithm with fuzzy functions [16] were applied to the open platform TUBSTAP [17]. In this work, we also adopt the TUBSTAP platform for the application of our method.

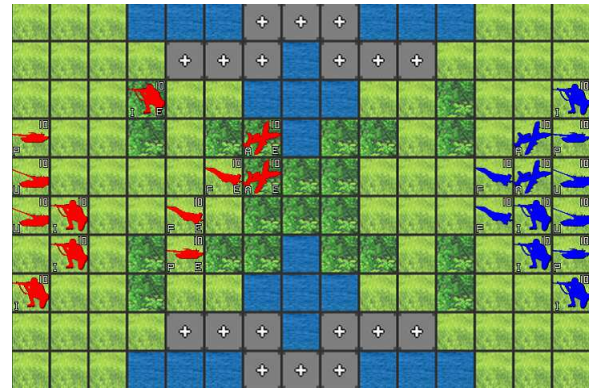


Fig. 1. Screenshot of the TUBSTAP platform.

TABLE I  
UNIT RELATIVE STRENGTH

Defense Attack	F	A	P	U	R	I
F	55	65	0	0	0	0
A	0	0	85	115	105	105
P	0	0	55	70	75	75
U	0	0	60	75	65	90
R	70	70	15	50	45	105
I	0	0	5	10	3	55

TABLE II  
LAND TYPE - PROTECTIVE EFFECT AND MOVEMENT COST

Land type	Mountain	Forest	Plain	Road	Sea
[Protective Effect]					
A, F	0	0	0	0	0
R, I, P, U	0.4	0.3	0.1	0	0
[Movement Cost]					
A, F	1	1	1	1	1
P, U, R	$\infty$	2	1	1	$\infty$
I	2	1	1	1	$\infty$

## III. TUBSTAP PLATFORM

TUBSTAP is an open platform for TBS game computer players. The game system is modeled after “Famicom Wars DS2” [18]. A screenshot of TUBSTAP is displayed in Fig. 1. AI player competitions are held annually using the platform, and the code produced by some of the participants and maps that are used for the competitions are freely available.

We provide an overview of the game rules of the platform. In each turn, each player can manipulate as many units as they like in whatever order the player prefers. Each unit can perform an attack action, a movement action, or an “attack after movement” action in a turn. Attack actions reduce the Hit-Points (HPs) value of the opponent unit at which the attack is directed. When the HP value of a unit is reduced to zero, the unit is excluded from the game. A player wins if the opponent player loses all of their units.

There are six types of units in this platform. That is, Fighter (“F”), Attack Aircraft (“A”), Panzer (“P”), Cannon (“U”), Anti-Air Tank (“R”), and Infantry (“I”). In addition, there are five types of land cells, that is, mountain, forest, plain,



road, and sea. Each unit type and each land type are assigned constant values as shown in Table I and II. The values “relative strength” and “protective effect” define the amount of damage by attack actions. Equation 1 shows the amount of damage, i.e., the amount by which the HP value is reduced by a unit carrying out an attack.

$$\text{damage} = \frac{(\text{relativestrength}) \times (\text{attackHP}) + 70}{100 + (\text{protectiveeffect}) \times (\text{defenseHP})} \quad (1)$$

For example, when a unit A with seven HPs attacks an opponent unit P with nine HPs on a forest cell, the amount of damage unit P experiences is calculated as  $\frac{85 \times 7 + 70}{100 + 3 \times 9} (= 5)$ .

Cells that can be reached by each unit in turn are defined as “movable capacity” of the unit and the sum of the movement costs the cells along the movement path impose on the unit. The “movable capacity” of F, A, P, U, R, and I is 9, 8, 6, 6, 6, and 3, respectively. The movement cost of each land cell is listed in Table I. A cell is reachable if the sum of the movement costs along the movement path is less than or equal to the unit movable capacity. Each unit cannot proceed through cells occupied by any opponent unit, whereas each unit can pass through cells occupied by friendly units (although, they cannot remain on the same cell).

Each unit except U can attack an opponent unit on an adjacent cell after one movement per turn. Attacks instantly prompt a counter attack by the opponent unit after an attack action. U can only attack more distant units, located in cells found at a Manhattan distance of 2 or 3. These distant attacks do not lead to any counter attack, but U cannot move and attack during the same turn.

This platform serves game systems simpler than those of existing commercial TBS games, although the platform covers essential rules that are found in most TBS games. Thus, we consider this platform as a suitable testbed for this research even though success in creating strong artificial players in this platform does not guarantee success in other TBS games.

#### IV. APPROACH

In this section, we explain the three modifications we adopt. We aim to reduce the computational costs for tree search by removing certain edges of the tree. These reductions are important not only because they can limit the computational time to a reasonable amount, but also because they might be able to sufficiently extend the search depth such that it enables them to look ahead at the opponent’s actions.

##### A. Fixing orders of unit to move

In TBS games, the order in which players manipulate their units in a turn is often important. However, there are many cases in which the difference between orders is irrelevant. In addition, there are also cases in which the order of only a few units affects the result (e.g., the case in which the result is affected only by whether a unit X moves before or after another unit Y, even though there are many other units in the position). Therefore, we try to reduce the amount of

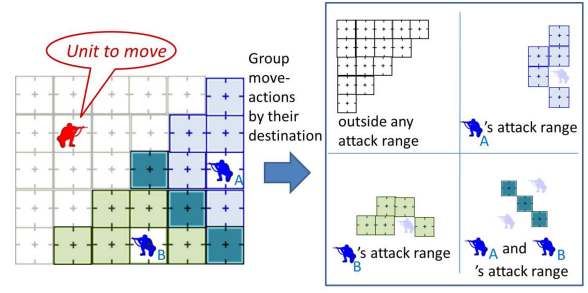


Fig. 2. Grouping movement actions. Only one movement action is generated per group.

computation by forcing each player to manipulate their units in some (or a single) fixed order.

In this work we attempted to use the fixed-order patterns listed below for all the units that are assigned IDs(1-N) randomly in a map.

- **Forward**  $1, 2, 3, \dots, N$ .
- **Backward**  $N, (N-1), (N-2), \dots, 1$ .
- **Cut-Forward**  $(\frac{N}{2} + 1), (\frac{N}{2} + 2), \dots, N, 1, 2, 3, \dots, \frac{N}{2}$ . Forward order with the former and the latter parts swapped.
- **Cut-Backward**  $(\frac{N}{2} - 1), (\frac{N}{2} - 2), (\frac{N}{2} - 3), \dots, 1, N, (N-1), \dots, \frac{N}{2}$ . Backward order with the former and the latter parts swapped.

Depending on the value of a parameter, our proposed player (which appears in the following sections) might consider only one of them, or some of them.

##### B. Selective unit action generation

At first we provide definitions for some of the specific terms used in this method. Strictly speaking, there are attack actions, movement actions, and “attack after movement” actions in TBS games, although we include “attack after moving” in the attack actions for convenience. Furthermore, we prune movement actions and attack actions differently.

##### [Movement action]

The number of possible movement actions per unit tends to be large. Thus, we group the movement actions of a unit according to the attack ranges of opponent units. This grouping procedure is illustrated in Fig. 2. Then, we select one movement action from each group, and discard the other actions. This approach seems to be important because other actions in the same group only serve the same opponent units that can attack the unit in the next turn. There are many ways to decide which action to pick from each group; however, we selected actions according to the priorities stated below (the first has higher priority).

- 1) The extent of protection the unit can take after the movement (the higher the better.)
- 2) The Manhattan distance after movement of the coordinates from the center of the whole units in the map (the shorter the better.)

If more than one candidate remains, choose only one at random.

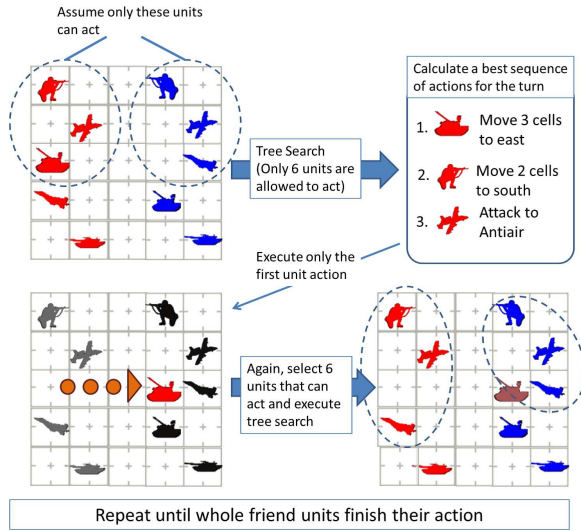


Fig. 3. Tree search with limited number of moving units. Instead of performing a tree search using a large depth, repeat a shallower tree search in which only a limited number of units can move.

### [Attack action]

Similarly, we group attack actions to partially prune them. We group the attack actions of a unit according to their target unit (note that there can be more than one attack action from a unit against another unit in a position). Then, we select only one action per group, and discard the others. The action to be chosen is selected by the number of opponent units that can attack the unit in the next turn. The action with the least value is selected. If there are multiple candidates, only one action is selected at random.

### C. Limiting the number of moving units

Before we start explaining the detailed issues, to avoid confusion, we provide definitions for words used frequently in this section.

- **Unit action:** Atomic action by a unit. Movement or attack.
- **Player action:** A sequence of multiple **unit actions** that one player can take in one turn. In case a player has  $N$  units, their player action consists of  $N$  unit actions (unless the player finishes the turn leaving some of their units unmoved).

In TBS games, the number of player actions increases exponentially according to the number of units. Given that a player has  $N$  units on their side, each of which can take  $M$  possible actions, a minimax tree requires  $N!M^N$  leaf nodes to look ahead of whole possible actions of the player in a turn (i.e., search 1 ply deep). This value easily exceeds a reasonable amount for practical use (e.g.,  $M = 30$  and  $N = 6$  result in about 520 billion nodes) as the value of  $N$  increases.

Therefore, we try to decrease the number of units involved in a tree search. The procedure is illustrated in Fig. 3. We repeat the two following operations until the whole the unit finishes its action, (a) search the best **player action** by tree

search in which only a limited number of units generates possible actions, (b) execute only the first **unit action** from the obtained best player action.

This procedure is able to decrease the exponential factor in the computation. In case each player has  $N$  units each of which  $M$  possible unit actions can be taken, the amount of nodes needed for a naive  $D$ -ply tree search is  $(N!M^N)^D$ . However, by applying this limitation technique (only  $N' (< N)$  units can generate actions), the number of nodes becomes fewer than  $N(N'!M^{N'})^D$ . The number of nodes decreases by a factor of  $\frac{1}{N}(\frac{N!}{N'}M^{N-N'})^D$ .

It should be considered how to select such ‘movable’ units (i.e., units capable of generating possible unit actions in a tree search) in this method. The measurements are as follows (the first has a higher priority):

- 1) HP value of the unit. A higher value is preferred.
- 2) Manhattan distance from the coordinates at the center of all the units in the map. A smaller distance is preferred.

By adopting these measurements, we ignore weak units (i.e., units incapable of causing a large amount of damage against opponents and can be destroyed easily), or units further from a hot spot. These units can be considered to be less influential on the battle situation.

## V. PRELIMINARY EXPERIMENT 1: FIXING UNIT ORDERS

In this section, we describe the experiment that was carried out to determine the influence of the fixing order technique on the performance. We prepared an  $\alpha\beta$  search player to which we apply the fixing order technique. (Hereafter we refer to the search player as the ‘proposed player’ to indicate players we built for experiments) This player has two parameter variables: the search depth of the game tree and the number of fixed-order patterns adopted by the player. The performance of the player is measured through battle trials.

### A. Design: Artificial Player

The proposed player’s search depth (the number of ‘player actions’ that need to be looked ahead) is 1 or 2. The unit orders considered by the player are {**Forward**} (see section IV-A), {**Forward, Backward**}, {**Forward, Backward, Cut-Forward, Cut-Backward**}, or whole possible orders. The state evaluation function adopted by our proposed player is described in the appendix of this paper.

### B. Experimental Setup

The proposed player competes against a naive UCT player (described in [15]) that uses 10,000 playouts per unit action generation. The maps illustrated in Fig.4 to 6 are used, and 200 matches are carried out for each map. The proposed player moves first in 100 games, after which the other player plays first in the next 100 games. A drawn match is counted as a 1/2 win for both players.



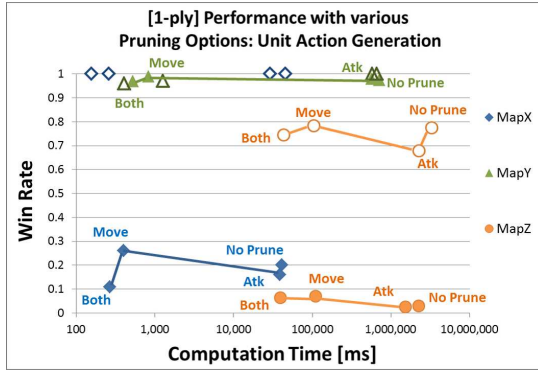


Fig. 9. Win rate against UCT player. 1 ply deep. Pruning options for selective action generation are varied. Filled/Outlined markers for matches in which the proposed player plays first/second.

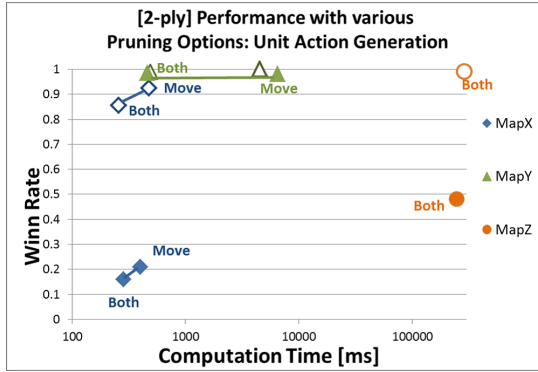


Fig. 10. Win rate against UCT player. 2 ply deep. Pruning options for selective action generation are varied. Filled/Outlined markers for matches in which the proposed player plays first/second. If the computation time exceeds the time limits (300 seconds), the plot is omitted; e.g., each of the ‘No-Prune’ options are omitted from the plot because their time limits exceed the specified value.

### C. Result

The results of the battles are shown in Fig. 9 and 10. The pruning attack actions seem to degrade the performance even though the differences are slight in some cases. On the other hand, pruning movement actions seem not to degrade the performance, although this type of pruning action decreases the computation time significantly by nearly a factor of 10.

Thus, we conclude that the approach involving selective generation is effective in this case, because it does not result in significant performance degradation even though it reduces the computation time.

## VII. PRELIMINARY EXPERIMENT 3: LIMITING THE NUMBER OF MOVING UNITS

The approach described in Sect. IV-C is examined in this section.

### A. Design: Artificial Player

As in the experiments in Sect. V and VI, we prepare an  $\alpha\beta$  search player of which the state evaluation function is provided in the appendix. The player employs a tree search with a 1-

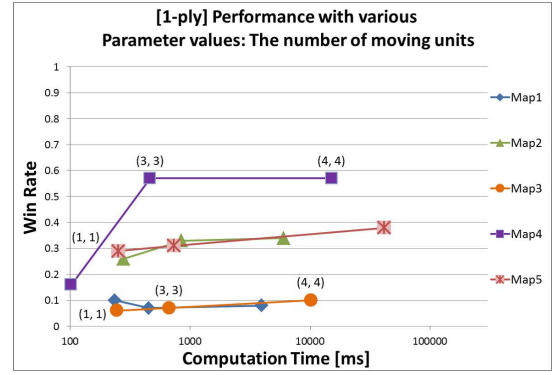


Fig. 11. Win rate against UCT player. 1 ply deep. The parameter value (M, N) represents that “Only M friendly units and only N opponent units can participate in the tree search”. Then, (1, 1), (3, 3), and (4, 4) are employed here.

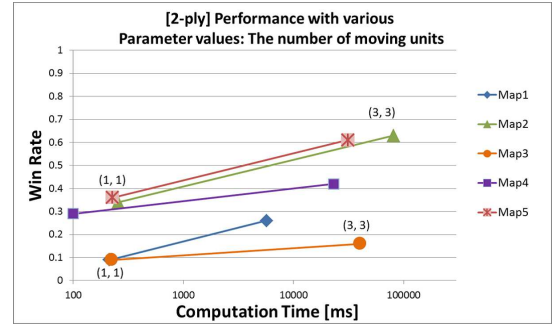


Fig. 12. Win rate against UCT player. 2 ply deep. The parameter value (M, N) represents that “Only M friendly units and only N opponent units can participate in the tree search”. Then, (1, 1), (3, 3), and (4, 4) are employed here, although (4, 4) exceeded the time limit (300 seconds).

or 2-ply search in which only the limited number of units can act.

### B. Experimental Setup

The number of friendly and opponent units that can act in the tree search performed by our player, are (1, 1), (3, 3), and (4, 4). A naive UCT player with 10,000 playouts is the opponent player.

We adopted maps different from those used in previous sections, because this pruning technique allows our player to function on larger maps. The five sample maps bundled with TUBSTAP platform ver. 1.07 (available on the website) are used. The number of red/blue units are (6, 6), (8, 4), (7, 7), (5, 5), and (7, 7), respectively. The number of matches per each map is the same as in Sect. V.

### C. Results

The results are shown in Fig. 11 to 12. When the parameter values are increased, the winning ratio and computation time are increased. Additionally, the performance improved with the deeper search except for the battles on map 4. The results show that this limiting approach might harm the performance in some cases, but it surely seems to decrease the computation time.



TABLE III  
WIN RATES AGAINST FIRST- AND SECOND-RANK AI PLAYERS (200 MATCHES FOR EACH MAP)

% wins against first-rank player "M-UCT" (with 95% C.I.)						
	Map-Fujiki	Map-Ishitobi	Map-Muto	Map-Sato	Map-Takahashi	Averaged
Move First	90 ( $\pm 5.9$ )	75 ( $\pm 8.5$ )	41 ( $\pm 9.6$ )	93 ( $\pm 5.0$ )	66 ( $\pm 9.3$ )	73 ( $\pm 3.9$ )
Move Second	94 ( $\pm 4.7$ )	65 ( $\pm 9.3$ )	26 ( $\pm 8.6$ )	98 ( $\pm 2.7$ )	92 ( $\pm 5.3$ )	75 ( $\pm 3.8$ )
Total	92 ( $\pm 3.8$ )	70 ( $\pm 6.4$ )	34 ( $\pm 4.2$ )	96 ( $\pm 1.7$ )	79 ( $\pm 3.6$ )	74 ( $\pm 2.7$ )
% wins against second-rank player "DLMC-PW55" (with 95% C.I.)						
	Map-Fujiki	Map-Ishitobi	Map-Muto	Map-Sato	Map-Takahashi	Averaged
Move First	89 ( $\pm 6.1$ )	65 ( $\pm 9.3$ )	82 ( $\pm 7.5$ )	97 ( $\pm 3.3$ )	63 ( $\pm 9.5$ )	79 ( $\pm 3.6$ )
Move Second	82 ( $\pm 7.5$ )	53 ( $\pm 9.8$ )	78 ( $\pm 8.1$ )	97 ( $\pm 3.3$ )	80 ( $\pm 7.8$ )	77 ( $\pm 3.7$ )
Total	86 ( $\pm 4.8$ )	59 ( $\pm 4.3$ )	80 ( $\pm 3.5$ )	97 ( $\pm 1.5$ )	72 ( $\pm 3.9$ )	78 ( $\pm 2.6$ )

## VIII. EXPERIMENT: PERFORMANCE EVALUATION

Next, we assess the performance of the proposed method against some advanced players. We prepared a player by applying all three the modifications. The proposed player competes against the winners of the TUBSTAP AI competition 2016 [19] we adopted as the opponent players in this experiment.

### A. Design: Artificial Player

We prepared an  $\alpha\beta$  player by employing the three forward pruning techniques. The parameter values that were used for pruning in the proposed player were decided by experiments that are omitted from this paper (because of space limitations). The parameters are:

- 2-ply deep search.
- Two fixed-order patterns for units' actions. (**Forward** and **Backward** in section V.)
- Each friendly unit generates pruned movement actions and pruned attack actions as its possible actions in the tree search.
- Each opponent unit generates no movement actions and pruned attack actions as its possible actions in the tree search.
- Five friendly units and 10 opponent units generate possible actions in the tree search.

### B. Experimental Setup

The proposed player competes against two players on five maps. The detailed conditions are as follows.

- Five maps are used. (These maps are the same as those used in the GAT2016 competition and are available on the website [19].)
- First-rank player "M-UCT" and second-rank player "DLMC-PW55" in the GAT2016 competition as opponent players. (These players are also available on the website.)
- Two-hundred matches per map for each opponent. One player plays first in 100 games, and the other player plays first in the next 100 games.
- A match that ends in a draw is counted as a 1/2 win for both players.

- Each player uses around 10 seconds to carry out its player action.

We also tested a player without any of the three pruning techniques, though, the player cannot search even 1-ply deep within one minute.

### C. Results

The result of the experiment is presented in Table III. The proposed player was significantly more competent than the first-rank player on four out of the five maps, and was significantly stronger against the second-rank player on all the maps. Additionally, the averaged win rates exceeded 70 % over the whole game against the first- and second-rank player, respectively. Thus, we conclude that the proposed player outperformed both of the winners of the 2016 competition. We attribute this performance to the success of our proposed methods in extending the search depth without negatively affecting the precision of the tree search to any significant extent.

## IX. CONCLUSION AND FUTURE WORK

We proposed three forward-pruning techniques to apply minimax search variants to TBS games. These methods allow artificial players to search quicker and deeper at the risk of overlooking some important moves.

The influence of these techniques on the respective players' performance was analyzed. These analyses showed that, although these techniques have a slightly harmful effect on performance, they allow the artificial player to search deeper, thereby resulting in the enhancement of the overall performance on the TUBSTAP platform.

Then, we introduced the techniques into an  $\alpha\beta$  search with appropriate parameter values and created an artificial player on the TUBSTAP platform. The experimental result showed that the created player significantly outperformed first- and second-rank players in the TUBSTAP AI 2016 competition.

However, we only evaluated our method on a platform of which the rules are simpler than those of existing commercial TBS game titles. On this platform, the benefits of using our method (reduction of the search space) are greater than the disadvantages (risks of overlooking critical moves). The benefits should be assessed in other TBS game environments because they may not be greater than the disadvantages.

Moreover, our methods have room for improvement because they contain unsophisticated architectures. For example, the method for selecting actions from whole legal moves is designed roughly. In addition, our method is presently unable to intentionally support movement actions that guard important friendly units.

#### ACKNOWLEDGMENT

The authors wish to thank Muto Kohsuke and Fujiki Tsubasa for releasing the AI players used in this paper, ‘M-UCT’ and ‘DLMC-PW55’.

#### APPENDIX

##### A. State Evaluate Function

We prepared artificial players by using an  $\alpha\beta$  search, and the search method needs a state evaluation function. Here, we explain the state evaluation function adopted over all the experiments in this paper.

A game position is scored as follows:

- 1) Each unit (owned by the player to move) carries out an attack action that causes the largest damage out of all possible attack actions. (However, these ‘possible attack actions’ are not precisely suggested. Whether a unit can attack another unit is roughly (and quickly) estimated by obtaining the Manhattan distance between the two units.)
- 2) Calculate the weighted sum of unit HPs as the evaluation value. The weight values are: 1 for the turn player’s infantry units, 4 for the turn player’s units except infantry, -1 for the other player’s infantry, -4 for the other player’s units except infantry.

#### REFERENCES

- [1] M. Campbell, A. J. Hoane and F. Hsu, “Deep blue,” *Artificial intelligence*, Vol.134, No.1, pp.57-83, 2002.
- [2] K. Hoki and T. Kaneko, “Large-Scale Optimization for Evaluation Functions with Minimax Search,” *Journal of Artificial Intelligence Research*, Vol.49, pp.527-568, 2014.
- [3] D. Silver, et al., “Mastering the game of Go with deep neural networks and tree search,” *Nature*, Vol.529, No.7587, pp.484-489, 2016.
- [4] R. D. Greenblatt, D. E. Eastlake III and S. D. Crocker, “The Greenblatt Chess Program”, *Fall Joint Computing Conf. Procs.*, pp.801-810, 1967.
- [5] chessprogramming - Move Ordering, [Online]. Available: <https://chessprogramming.wikispaces.com/Move+Ordering>, [Accessed: 2016-04-30].
- [6] J. Schaeffer, “Experiments in search and knowledge,” Ph.D. Thesis, Department of Computing Science, University of Waterloo, Canada, 1986.
- [7] G. M. Adelson-Velskie, V. I. Arlazarov, M. V. Donskoy, “Some methods of controlling the tree search in chess programs,” *Artificial Intelligence*, Vol.6, No.4, pp.361-371, 1975.
- [8] T. Romstat, An introduction to late move reductions. [Online]. Available: <http://www.glauringchess.com/lmr.html>, [Accessed: 30- Apr- 2016].
- [9] K. Hoki and M. Muramatsu, “Efficiency of three forward-pruning techniques in shogi: Futility pruning, null-move pruning, and Late Move Reduction (LMR),” *Entertainment Computing*, Vol.3, No.3, pp.51-57, 2012.
- [10] Sid Meier’s Civilization Beyond Earth, [Online]. Available: <https://www.civilization.com/jp/games/civilization-beyond-earth/>, [Accessed: 19- Feb- 2016].
- [11] HJ. M. Bergsma and P. Spronck, “Adaptive Spatial Reasoning for Turn-based Strategy Games,” *AIIDE*, Proc., pp.161-166, 2008.
- [12] W. Stefan and I. Watson, “Using reinforcement learning for city site selection in the turn-based strategy game Civilization IV,” *CIG*, Proc., pp.372-377, 2008.
- [13] C. Amato and G. Shani, “High-level Reinforcement Learning in Strategy Games,” *AAMAS*, Proc., pp.75-82, 2010.
- [14] P. Ulam, et al., “Using Model-Based Reflection to Guide Reinforcement Learning,” *IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games*, Proc., pp.107-112, 2005.
- [15] T. Fujiki, K. Ikeda and V. Simin, “A platform for Turn-Based Strategy Games, with a Comparison of Monte-Carlo Algorithms,” *CIG*, proc., pp.407-414, 2015.
- [16] K. Mutoh, J. Nishino, “Cutoff Using Fuzzy Evaluation on AI of Turn-based Strategy Games,” *IPSJ-GPW*, pp.54-60, 2015 (in Japanese).
- [17] TUBSTAP. [Online] (in Japanese). Available: [http://www.jaist.ac.jp/is/labs/ikeda-lab/tbs\\_eng/index.htm](http://www.jaist.ac.jp/is/labs/ikeda-lab/tbs_eng/index.htm) [Accessed: 1- May- 2016]
- [18] Nintendo, Advance Wars: Days of Ruin for Nintendo DS - Nintendo Game Details. [Online]. Available: <http://www.nintendo.com/games/detail/nLeg9iKpgq3fWBcqpDNWUJ4IvmaQBY>, [Accessed: 2- May- 2016].
- [19] TUBSTAP Game AI Tournament 2016. [Online] (in Japanese). Available: [http://www.jaist.ac.jp/is/labs/ikeda-lab/tbs/competition\\_menu.htm](http://www.jaist.ac.jp/is/labs/ikeda-lab/tbs/competition_menu.htm), [Accessed: 1- May- 2016]
- [20] E. D. James and T. P. Hart, “The alpha-beta, heuristic.” Massachusetts Institute of Technology, 1963.



# Voluntary Behavior on Cortical Learning Algorithm Based Agents

Ali Kaan Sungur  
Graduate School of Informatics  
Middle East Technical University  
Ankara, Turkey  
ksungur@metu.edu.tr

Elif Surer  
Graduate School of Informatics  
Middle East Technical University  
Ankara, Turkey  
elifs@metu.edu.tr

**Abstract**—Operating autonomous agents inside a 3D workspace is a challenging problem domain in real-time for dynamic environments since it involves online interaction with ever-changing decision constraints. This study proposes a neuroscience inspired architecture to simulate autonomous agents with interaction capabilities inside a 3D virtual world. The environment stimulates the operating agents based on their place and course of action. They are expected to form a life cycle composed of behavior chunks inside this environment and continuously optimize it around the stimulated reward. The architecture is composed of specialized units that run Cortical Learning Algorithm (CLA) which models functional properties of layers II and III as in six layer theory of neocortex. This work focuses on extending it with functional properties of layers IV, V and basal ganglia to obtain voluntary behavior that is suitable for an autonomous agent. Through experimental scenarios, the architecture is observed and evaluated in order to obtain an apparent learning process. The communication between layers and internal connectivity of embedded CLA units are able to capture sequential and causal relations from the environment and the first evaluation of the implementation has high potential for future directions.

**Keywords**—cortical learning algorithm; hierarchical temporal memory; autonomous agent; neocortex layers; artificial life

## I. INTRODUCTION

Building an autonomous agent that operates in a virtual 3D workspace is a complex task which can be approached from a variety of fields [1]. To name a few, the solution can be based on machine learning, artificial intelligence, logic models, signal processing principles, mathematical theories, nanotechnology and other related fields [2, 3]. Recently, a number of studies integrated neuroscience into their solutions. Hierarchical temporal memory (HTM) is one of the better known frameworks that inhabit neuroscientific properties [4]. The theory involves constructing a hierarchical structure composed of computational units that imitate the local regions of neocortex at a functional level. The computational units of this hierarchy runs on CLA [5]; a model that incorporates algorithmic principles of some individual layers in neocortex, namely layers II and III.

Cortical learning algorithm itself may seem to promise a domain independent solution on the surface with its high noise tolerance, semi-unsupervised approach and neuroscientific

algorithmic properties. However, in order to build an autonomous agent based on CLA, additional structures are required to produce voluntary behavior. Reward circuitry is a central structure in any autonomous agent for assessing options on a given situation. Novelty detection and internal experience replay mechanisms are also crucial to obtain a predictable, efficient and apparent learning process from a human-like perspective. These properties are obtained through various subcortical regions and brain parts in neuroscience.

This study identifies the related areas in the brain to incorporate the necessary properties into the model and extracts their algorithmic properties in order to come up with a model that is capable of simulating an artificial life inside a 3D virtual environment. After presenting the related background, the study continues with an overview of CLA and the proposed architecture. This is followed by the detailed functionality of these structures and how they are connected together. The focus then is on evaluating the model through the use of an autonomous agent operating in 3D environments with constraints. Implementation, performance metrics for these scenarios and information regarding the testing platform are provided. Finally, the model is discussed in detail to underline the strengths and weaknesses followed by conclusion and future works. This study implements the layers IV, V and basal ganglia based on current CLA foundations and introduces a novel prototype which further extends the capabilities of the existing models to simulate artificial life.

## II. BACKGROUND

The HTM model was developed based on the memory prediction theories that are presented On Intelligence by Jeff Hawkins [6]. A year after he published the book, he founded Numenta Inc. with Dileep George. The goal was to come up with a theory of how the brain works and a platform to implement it. Sparse distributed memory [7] was their starting point which represents information throughout the model.

Cortical learning has been around for a decade and it is utilized in various fields and for commercial purposes. Grok [8] is the commercial application of this model onto various IT analytics problems. It is mainly used as an anomaly detector in commercial applications such as geospatial tracking and stock predictions. The company also has an open source implementation Nupic [9] since 2013, in order to help advance

the theory and the implementation through an active community. The model is also utilized in a natural language processing domain by Cortical.io [10].

Computational architecture of neocortex layers were argued on [11] in an extensive research. In [12], the HTM theory extended to create a general solution for resembling the brain functioning. Hawkins and Subutai patented their neuroscience related research on cortical learning [5, 13]. These patents focused on the algorithmic properties and the correlations of the HTM theory with neuroscience. Building an autonomous agent based on CLA was attempted previously on a 2D virtual world where it is extended with emotions that would guide the learning [14]. The agent would then decide which grid to go depending on the output of the model.

### III. CORTICAL LEARNING ALGORITHM

Cortical learning algorithm is a neural classifier at its core. The aim is to model the basic learning principles of the neocortex at a functional level. The advancements on the theory of neocortex layers clarify the missing algorithmic properties of CLA. The structure of the model is similar to the mini and macro column theories in neuroscience [15]. Although this paper is not focused directly on CLA, it is necessary to have an understanding on its internal mechanism to grasp the proposed architecture. Therefore, basic functional mechanisms are described below.

The CLA structure consists of multiple columns of neurons in a grid, forming a rectangular prism of neural layers. The neurons of the same column have similar functionality and the whole column gets input from its proximal dendrites. In other words, all the neurons in the same column are stimulated with the same input fed into the CLA. The proximal dendrites have a receptive range on the input field. The individual neurons forming the columns have their own distal dendrites which are connected to other individual neurons in other columns.

#### A. Spatial Pooler

At any given state, the number of active columns excited by the input is sparsified by an inhibition system. Based on the experiments conducted by Numenta and the findings on neocortex, a sparsity of %2 is preferred [6]. This ensures that all the states are represented with a small set of columns. Therefore, the region utilizes sparse distributed representation to describe its state at all times. This representation has some very powerful capabilities and it is the backbone of CLA theory that satisfies vital properties to make it work [7].

#### B. Temporal Memory

The described system up to this point is just a general classifier with some neuroscientific properties. The same could be achieved with other models such as a variance of neural networks. However, the real capability of CLA is the contextual information that the individual neurons can provide. While the region is stimulated with online data, the patterns of active columns change according to the state. For example, let A be the set of active columns at time t. At t+1, the input changes along with the active columns and assume this new activation is B. The neurons belonging to the columns at B

subsample the neurons of the columns of A and start forming connections towards those neurons. With enough trials, the neural connections between different neurons become stable. This results in an ability to predict the activity of B when A is encountered. To emphasize, the connections are formed between neurons not the whole columns. A column is active if it contains any active neurons. The activation on a columnar level represents the state without the context. On the other hand, individual neurons of the columns represent the context of the current state, in other words, the sequence that this state is occurring. Therefore, the neural activation on a state C that came after A and B is different from a state C that came after D and E. Although the activated columns are the same, the individual neurons of the columns implicitly carry the information of the recent past through their connectivity.

### IV. ARCHITECTURE

The theory of Hierarchical Temporal Memory constitutes of multiple computational units that are functional imitations of cortical regions. The core computational units are modeled after CLA. The general approach is to obtain a framework that runs on the algorithmic principles of the six layer theory of neocortex. In order to do this, CLA units are used to represent one or more layers of a single region and these units have differing functions based on what layer they algorithmically imitate.

The current state of CLA mainly encapsulates the functionalities of layers II and III which are the layers that capture high order sequences. A CLA unit can also be used to emulate layer IV by configuring its parameters and connectivity. This layer takes the sensory/thalamic input into the region. It is assumed to be doing first order predictions in the model, meaning its predictions are based on the last sensory input. Although layer V is one of the most studied part of a region, because of its complexity, it is not yet fully understood and structural changes are necessary to CLA in order to implement it. Functionally, layer V is responsible for making motor neuron connections and it also is the source of the output coupled with very little understood layer VI. The general layer connectivity of the agent model can be seen in Figure 1 below.

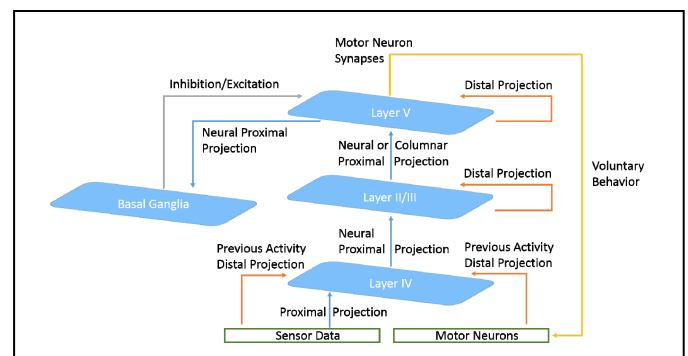


Fig. 1. Detailed infrastructure of the agent model.

#### A. Sensor

The proposed architecture is compatible with a variety of sensor types. However, in order to utilize the learning capability efficiently, certain properties of the input are crucial

to the functioning of the system. The HTM theory builds on sparse distributed representations and most of the properties below enforce the input to obey this representation type.

- Binary

For the computational simplicity and practical implementation, the sensory input is assumed to be a binary input vector. This vector could represent any type of data and encoded in any way. The input can be multi-dimensional as the topology of CLA allows it. A sensor encoding topological information from a virtual world significantly benefits having a multi-dimensional structure.

- Sparse

Sparsity directly affects the number of different patterns that can be described using a fixed sized input vector. It also dictates the range of semantic similarity between different input vectors. The preferred sparsity of HTM theory is between %10 and %1. The neurological evidence points towards the sparsity of %2 [6] and empirical experiments conducted using Nupic [9] show that neurological sparsity levels are efficient for the algorithm.

- Semantic Meaning

Every bit of the input must have some meaning in this architecture. This provides the structure for similarity or difference comparisons between vectors. Therefore, different inputs would have similar active bits corresponding to the semantic similarity.

Finally, there is no restriction on the size of the input and the layer connected to the sensory input can be mapped to any dimensionality or size.

#### B. Layer IV

Input is fed into cortical regions through layer IV. The input can be output of a region lower in the hierarchy or it can be sensory data from thalamus [16]. In this architecture, layer IV is represented as a modified CLA. The proximal dendrites of its columns are connected directly to the sensor of the system. The range of the receptive fields of the proximal dendrites should be according to the input structure. Studies point out that layer IV does first order processing which means that the predictions does not encapsulate high order sequences. Therefore, the distal dendrites of the neurons in CLA imitating layer IV are not connected to other neurons in the same CLA. They are connected to the previous state of the sensor. This would result in the predicting neurons of the CLA to be fired based on the context of the previous input.

#### C. Layer II/III

The original Cortical Learning Algorithm is mainly based on the functional properties of layers II and III combined. Therefore, the CLA imitating these layers is not modified structurally. The proximal dendrites of the columns are connected to individual neurons of the lower layer IV. The distal dendrites make connections horizontally with other neurons of the same CLA. This mechanism is capable of representing high order sequences as layers II and III. The

resulting predicting neurons encapsulate the context of the current input based on the current high order sequence.

#### D. Layer V

The output of the cortical region originates at layer V. The individual cells of this layer has indirect synaptic connections to the motor neurons in various places such as spinal cord and muscles. In order to voluntarily control any behavior, the layer has to learn which motor neurons are controlled with which neuron sets first. Therefore, before any voluntary behavior, the relation between motor neurons and layer V cells need to be constructed. This construction takes place as learning by association. At initialization, the synapses between motor neurons and layer V cell can be completely disconnected or there can be a preset of connectivity that could quicken learning the relationship between motor behavior and layer V states. At each iteration, the active neurons slowly connect to the motor neurons that are active at the same time. Therefore, it associates its internal states with motor activity.

Layer V can send signals for voluntary motor commands after the layer constructs a mapping between motor activity and its activation. The output of the region originating at this layer becomes the desired motor command itself. The detail of this command is relative to the place of the region place in the hierarchy. In the HTM theory, this output also feeds up the hierarchy of cortical regions. Simultaneously, the output is also sent to the corresponding part of the thalamus, basal ganglia. Therefore, the output also plays a critical role in corticothalamic circuit and reward circuitry involving basal ganglia.

The consensus around the workflow of neocortex is not completely unified. While it is clear that the input enters the region from its layer IV, projects into layer II and III, the preceding steps are argued upon. There are two views at this point. The layer III projects to layer V or the projections to layers II/III and V/VI happens simultaneously. The proposed architecture utilizes the former point of view [17].

This layer hierarchically sits on top of layer II/III and mainly gets excited by these layers. The proximal dendrites of the layer V CLA unit are connected to layer III. The connections can either be directly to the aligning columns for computational simplicity or to individual neurons of layer III. The distal dendrite connectivity of individual neurons is just as in default CLA unit; lateral connections among the neurons of layer V.

In [18], it is stated that the activation in layer V is modulated by layer VI which is the layer controlling the attention of a region. In terms of CLA functionality, this finding implies that layer V has connections to layer VI through its distal dendrites but this is currently not considered due to architecture scope. The architecture involves a single region. The layer VI is also responsible for feedback down the hierarchy.

The activation in the layer V represents the voluntary action created inside the region. Therefore, it is mandatory for this layer to be influenced by some kind of reward circuitry which is discussed in the next section. According to layer V is also

modulated by layer I through the apical tuft of its neurons. This allows subcortical regions such as basal ganglia to indirectly adjust the activation in layer V. In the architecture, this mechanism is implemented through direct modulation from basal ganglia to layer V bypassing layer I.

#### E. Basal Ganglia

The reward circuitry is a mandatory component of any autonomous agent. It guides the learning, prioritizes rewarding behaviors and therefore naturally results in the learning capacity being used for salient experiences. The reward mechanisms in mammalian brains involve a complex communication between neocortex, thalamus and subcortical regions. Basal ganglia are considered as the planning and the action selection center among these structures. The general consensus on its functionality is that basal ganglia do a type of reinforcement learning. It interferes with corticothalamic pathways and affects the cortex via increased or decreased reward signals [19, 20]. Although the layer I does not have an explicit place in this architecture, basal ganglia actually outputs to this layer [21]. This layer has dendritic tufts from layers II, III and V. Many subcortical structures and thalamus influence the region indirectly via these dendritic tufts. Basal ganglia is one of these structures and it indirectly performs excitation and inhibition on layer V. This mechanism guides the behavior by modulating the layer V activation and selectively picks an output based on its reward [22]. At the same time, it also gets rid of conflicting response patterns generated in this layer.

The complex reward mechanism in neocortex is simplified for practical modelling purposes in this architecture. Layer I does not have an explicit presence and basal ganglia are able to directly influence layer V. Having a single region in the proposed architecture instead of a hierarchy facilitates a more direct control over the individual region. As with every layer, basal ganglia are also emulated as a modified CLA unit. The proximal dendrites of the columns are connected to individual layer V neurons. For more efficient capacity usage, the activation sparsity is lowered into a few neurons becoming active per iteration. In the current architecture, basal ganglia do not store any sequential information therefore the columns can be single neurons and distal dendrites can be ignored. The functionality in general is similar to having pointers to layer V states. The activated neurons in basal ganglia also store the current stimulation value corresponding to the current layer V activation. This results in basal ganglia indirectly storing layer V states with their pleasure outcomes.

The voluntary action selection takes places as the last process of the whole architecture. It is important to note that the aim is to capture the algorithmic properties of the basal ganglia and the exact functionality can be achieved in a variety of ways. The predictive neurons of the layer V CLA unit actually represents all the possible actions for the next step. If these neurons are fed to basal ganglia simultaneously, the resulting activation inside the ganglia would represent all the possible actions and their reward outcomes. At this point, the ganglia can inhibit or excite these depolarized cells according to a given heuristic respecting the sparsity levels of layer V. In the end, some of the depolarized cells of layer V will become active and therefore project to motor neurons creating the

voluntary behavior. Obviously, the process involves numerous tweaks and trials to capture the desired behavior forming process.

#### F. Motor Neurons

Although the motor neurons are the output of the system, they are fed into layer IV along with the sensory input on each iteration. This allows the layer IV to represent sensory transitions in the context of motor activity. These neurons are excited by layer V and modulated by basal ganglia with inhibition mechanisms to cancel out alternative behaviors. This modulation is based on the previously experienced reward on similar layer V state and behavior combinations. In other words, at the end of each iteration layer V represents the union of possible behaviors with its predictive (depolarized) neurons. Then, basal ganglia selectively allow a set of this neural activation using a direct inhibition and excitation mechanism on these depolarized neurons.

### V. TEST ENVIRONMENT AND IMPLEMENTATION

The architecture is an ongoing implementation as a part of a game. Currently it is simulated real-time inside a 3D environment. The platform is an in-house game engine written in C++, powered with DirectX 11. The architecture is embedded onto agents that are controlled by physical forces simulated by the underlying physics engine; Nvidia PhysX. The environment is a complex 3D workspace and generated procedurally. One of the important design goals is that the architecture should be a feasible solution as a non-playable character intelligence. The target hardware for the game should not exceed 8 gigabytes of memory. A single copy of the simulation is required to be lower than 1 gigabyte at runtime, so that it could be ran on multiple agents simultaneously. It is designed to run on CPU utilizing a single core per simulation. Below is the exact specification of the platform that the experiments are conducted.

- Intel Core i7-3632QM CPU @ 2.20GHz
- 12 GB RAM
- Nvidia GT645M GPU

In this study, CLA implementation is based on the whitepaper by Numenta [4] and the patent by Hawkins et al. [13] which extends the algorithm by features that solidify neuroscience basis. It is written as a single core simulation for simplicity, debugging and implementation time (Figure 2a and 2b). However, the algorithm is highly compatible with parallel solutions. The performance data on the next section is obtained from this custom implementation. As a side note, there is a publicly available implementation of CLA called Nupic [9] that can also be utilized in this architecture. The public source code has C++ and Python variants. Although, these implementations could be modified according to the needs outlined in architecture overview, they are simply not able to perform in real-time.

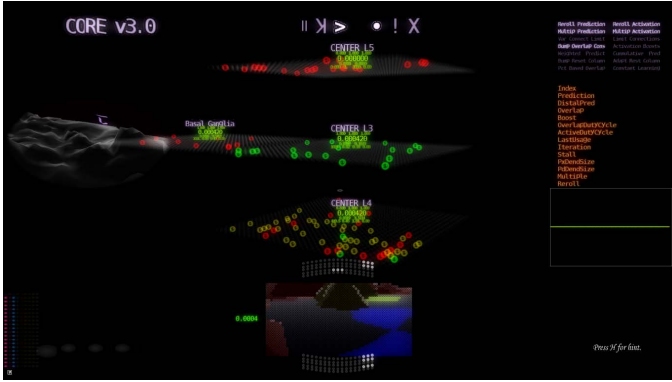


Fig. 2a. Columnar activity representation of the layers in the game.

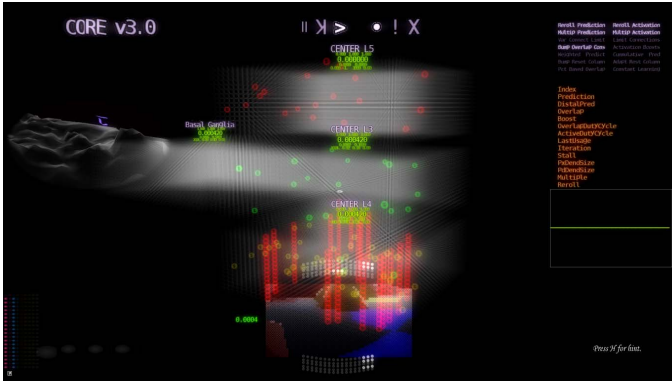


Fig. 2b. Neural activity representation of the layers in the game.

## VI. EVALUATION

### A. Performance Evaluation

Debugging and profiling the whole simulation is a complex task by itself. Being able to debug the CLA in real-time requires a highly sophisticated visualizer independent from the rendering of the game elements (Figure 3). Decoupling the total rendering task, game related computations and CLA simulations does not serve any purpose as the proposed architecture is supposed to run inside a game in real-time. In addition, the sensor of the architecture also utilizes the physics engine to obtain the information from the world. The motor neuron activations correspond to applying forces on the physical parts of the agent model.

Considering the intricate nature of the solution, profiling the architecture in isolation does not provide meaningful insight without simulating the environment and the agent. Still, the data on relative performance due to CLA unit size is given on Table 1. The comparison is based on CPU sampling detecting runtime occupations of functions related to the architecture. The number in the table represents CPU occupancy percentage of the architecture with respect to overall game. That said, the fundamental performance metric is the interaction responsiveness of the whole game. The interaction suffers as the frame rate of the whole simulation decreases. Obtaining around 30 frames per second is the ideal metric for a real-time solution with a lower bound of 20 frames per second on average.

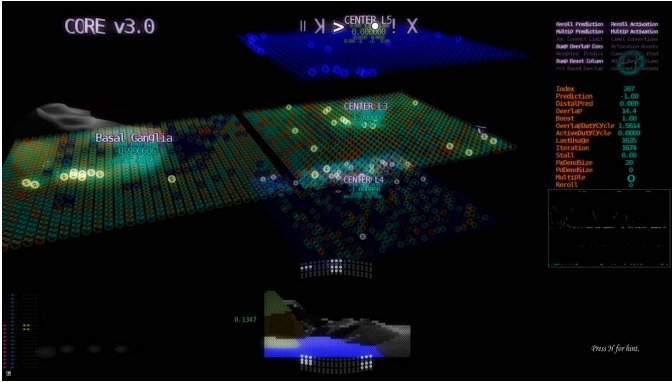


Fig. 3. Visual debugger showcasing the state of the architecture.

TABLE I. PERFORMANCE METRICS DUE TO CLA SIZE

Performance Comparison					
Columns per CLA	Neurons per Column	CLA Size	Model Size	Sampling Ratio	FPS
256	4	1024	4096	35,83	36,8
1024	8	8192	32768	58,66	26,2
1024	16	16384	65536	60,88	24,7

### B. Experimental Scenarios

The game environment is used for creating various scenarios which requires the agents to form behavior chunks that would gradually become better at obtaining pleasure. The scenarios can be generalized as navigating through an environment which stimulates the agent depending on the place and the course of actions. An example scenario is represented in (Figure 4). The environment is separated into polygons through creating a Voronoi diagram out of random points. The agent has a sensor obtaining information about these polygons. There is a set S of polygons that the agent is supposed to go that is predetermined. It obtains pleasure by travelling to these polygons that are not illuminated. The polygons that do not belong to the set S cause pain on the agent. The agent can also interact with these polygons by extracting resources if there are any available. A successful extraction provides pleasure for the agent. The agent uses a force actuated model and the force is controlled by the motor neurons of the architecture. It can interact with any polygon, rotate and travel in the direction it is pointing at. It does teleport to a preset location after a threshold of pain.

The scenario described above is a fairly complex one that involves a force actuated model navigating freely in a 3D environment and interacting with it. The agent and the architecture are both observed through every test session conducted. The information obtained through the accomplishments and the failures of the agent is discussed in the next section.



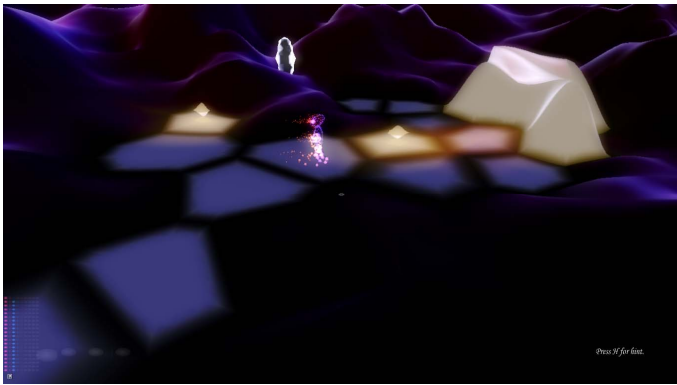


Fig. 4. Test scenario demonstrating the process of forming behaviors.

## VII. DISCUSSION

The expectation is to observe a humanly relatable learning process of an agent forming behaviors according to the stimulants of the environment. While the agent was able to form action sequences involving multiple steps, these were not consistent and did not create a complex life cycle involving similar behavior loops.

Throughout the test scenarios, layer IV captures the first order relations and this in turn lead to sparser representations that layer II/III can be fed with. In time, the prediction performance of layer II/III increases so that the relation between motor activity and layer V states stabilize. This results in layer V being able to associate motor activity with its internal states and forming connections from its neurons to motor neurons. The states of layer V are stored in basal ganglia simultaneously according to their respective outcomes. As soon as layer V generates predictions for the next step, basal ganglia interfere with those depolarized cells and through inhibition/excitation allows some of them to fire. These selected neurons of layer V create the voluntary motor activity that is going to change the sensory input in the best way the architecture can predict to obtain the most pleasure.

The main problem hindering the learning process is the trade-off between exploration and exploitation. If the agent is allowed to make optimized decisions for pleasure every time, it always performs the same course of action unless it predicts nothing. This hampers the exploration and no matter how small the reward is, the agent always picks the predictable route. This translates into doing short actions that end at the same state it started such as turning around itself, repeating rotation and movement patterns that each step is predicted. There are mechanisms implemented to reward novel actions and sensory inputs to guide the exploration but then the consistency of behaviors reduces dramatically. The connectivity of the CLA units build on statistical stability of the relation between sensory inputs and motor outputs. A sequence of action has to be repeated for multiple times in order for it to be captured. Therefore, the task of exploring while doing things in a frequency that could be learned is a great challenge.

There is also the problem of rare events. Most of the time, rewarding events happen infrequently. The architecture above in its current form is not suitable to learn from a few examples

as the synapse connections are made through repetitions. This presents a challenge in modelling those significant but rare events. When the learning speed is increased so that synapses form quicker connections, the system has problems with stabilizing. This leads to the problem of not being able to build upon the already learned relations between motor activity and sensory input. Every event no matter how insignificant has an impact on the connectivity including noise. There is a mechanism in place to weight the events based on their reward but this approach fails actually capturing the relations because the system needs already functioning predictions beforehand for this to work, like a preset basic knowledge.

## VIII. CONCLUSION

This study has proposed a neocortex inspired architecture to build an autonomous agent for a 3D virtual environment. To the authors' knowledge, there has not been a CLA-based working prototype that encapsulates the functionalities of layers IV, V and basal ganglia. The architecture is capable of forming simple voluntary behaviors if embedded onto an agent with an online sensor and controllable motor activity. The implementation performs in real-time as expected on the target hardware. Several scenarios have been prepared and tested on the autonomous agent on the gaming platform. The agent is able to form basic behavior chunks, however there are challenges such as exploration and exploitation conflicts, modeling infrequent yet salient events, stemming from the complexity of the task in hand. Finally, evaluation outputs have demonstrated that the functionality of the modeled neocortex layers are in line with their biological counterparts, thus making the proposed architecture a promising step towards forming more complex behaviors.

## IX. FUTURE WORK

There is a concept in neuroscience studies called hippocampal replay. The process involves both the neocortex and hippocampus. The short term episodic memory is stored in hippocampus. Internally, these memories are replayed forwards, backwards and with a variety of ordering during sleep and awake states. Hippocampal replay is believed to be playing an important role of modeling infrequent events that the neocortex cannot capture on its own without repetition. Based on the findings of this study, embedding algorithmic properties of this mechanism into the architecture may play a key role in creating the life cycle of the agents.

## REFERENCES

- [1] P. Maes, "Modeling adaptive autonomous agents", *Artificial Life* 1.1\_2:135-162, 1993.
- [2] P. Maes, "Artificial life meets entertainment: lifelike autonomous agents", *Communications of the ACM* 38.11:108-114, 1995.
- [3] T. D. Kulkarni, K. R. Narasimhan, A. Saeedi, and J. B. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation", *arXiv:1604.06057*, 2016.
- [4] J. Hawkins, A. Subutai, and D. Dubinsky, "Hierarchical temporal memory including HTM cortical learning algorithms", Technical report, Numenta Inc, Palo Alto, [http://www.numenta.com/htmoverture/education/HTM\\_CorticalLearningAlgorithms.pdf](http://www.numenta.com/htmoverture/education/HTM_CorticalLearningAlgorithms.pdf), 2010.



- [5] J. C. Hawkins, A. Subutai, G. Dileep, F. E. Astier, and M. I. Ronald, "Architecture of a hierarchical temporal memory based system", U.S. Patent 8,447,711, issued May 21, 2013.
- [6] J. Hawkins, and S. Blakeslee, "On intelligence", Macmillan, 2007.
- [7] P. Kanerva, "Sparse distributed memory", MIT press, 1988.
- [8] Grok for IT Analytics, (2016, May 2), Retrieved from: <http://numenta.com/grok/>.
- [9] Nupic: Numenta Platform for Intelligent Computing, (2016, May 2), Retrieved from: <http://numenta.org/>.
- [10] Cortical.io: Fast, precise, intuitive NLP, (2016, May 2), Retrieved from: <http://cortical.io/>.
- [11] D. Mumford, "On the computational architecture of the neocortex", *Biological cybernetics* 66, no. 3: 241-251, 1992.
- [12] D. George, "How the Brain Might Work: A Hierarchical and Temporal Model for Learning and Recognition", PhD thesis, Stanford University, 2008.
- [13] J. C. Hawkins, M. I. Ronald, A. Raj, and S. Ahmad, "Temporal memory using sparse distributed representation", U.S. Patent No. 9,189,745. 17 Nov. 2015.
- [14] M. Otahal, "Architecture of Autonomous Agent Based on Cortical Learning Algorithms", M.S. thesis, Faculty of Electrical Engineering, Department of Cybernetics. Czech Technical University in Prague, 2013.
- [15] G.J. Rinkus, "A cortical sparse distributed coding model linking mini- and macrocolumn-scale functionality", *Frontiers in Neuroanatomy*, vol. 4, no. 17, 2010.
- [16] K. D Miller, D. J. Pinto, and D. J. Simons, "Processing in layer 4 of the neocortical circuit: new insights from visual and somatosensory cortex", *Current opinion in neurobiology* 11, no. 4: 488-497, 2001.
- [17] X. Jiang, G. Wang, A. J. Lee, R. L. Stornetta, J. J. Zhu, "The organization of two new cortical interneuronal circuits", *Nat. Neurosci.* 16, 210–218, 2013.
- [18] S. Shipp, "Structure and function of the cerebral cortex", *Current Biology* 17, no. 12: R443-R449, 2007.
- [19] W. Schultz, P. Dayan, P. R. Montague, "A neural substrate of prediction and reward", *Science*, 275(5306), 1593-1599, 1997.
- [20] W. Schultz, "Predictive reward signal of dopamine neurons", *Journal of neurophysiology*, 80(1), 1-27, 1998.
- [21] W. J. Gao, and Z. H. Zheng, "Target-specific differences in somatodendritic morphology of layer V pyramidal neurons in rat motor cortex", *J. Comp. Neurol.* 476, 174–185, 2014.
- [22] M. Garcia-Munoz, and G.W Arbuthnott, "Basal ganglia—thalamus and the "crowning enigma", *Frontiers in neural circuits*, 9, 2015.

# Comparison of Rapid Action Value Estimation Variants for General Game Playing

Chiara F. Sironi and Mark H. M. Winands

Games & AI Group, Department of Data Science and Knowledge Engineering

Maastricht University, Maastricht, The Netherlands

Email: {c.sironi,m.winands}@maastrichtuniversity.nl

**Abstract**—General Game Playing (GGP) aims at creating computer programs able to play any arbitrary game at an expert level given only its rules. The lack of game-specific knowledge and the necessity of learning a strategy online have made Monte-Carlo Tree Search (MCTS) a suitable method to tackle the challenges of GGP. An efficient search-control mechanism can substantially increase the performance of MCTS. The RAVE strategy and its more recent variant, GRAVE, have been proposed for this reason. In this paper we further investigate the use of GRAVE for GGP and compare its performance with the more established RAVE strategy and with a new variant, called HRAVE, that uses more global information. Experiments show that for some games GRAVE and HRAVE perform better than RAVE, with GRAVE being the most promising one overall.

## I. INTRODUCTION

The aim of General Game Playing (GGP) is to develop agents that are able to play many arbitrary games at an expert level, only by being given their rules. As opposed to traditional game playing, GGP agents cannot rely on pre-coded game-specific knowledge because the game to be played is not known in advance. For the same reason, it is not possible to predetermine which search method is more suited for the game. The search method must be able to cope with a possibly infinite number of games. Moreover, the rules of the game are given to the agent only few seconds before the game starts, thus the agent has to learn the best playing strategy online. An extra challenge is posed by the fact that the agent usually has only few seconds per turn to choose a move.

A search technique that proved successful in GGP is Monte-Carlo Tree Search (MCTS) [1]–[3]. In its basic form MCTS is *ahuristic*, it does not require any game-specific knowledge, *anytime*, it can choose the move to be played within any time budget, and *selective*, it favours regions of the search tree that have the most promising moves, growing the tree asymmetrically [4]. Nowadays, all the best GGP agents are MCTS-based [5]–[9].

Other than GGP, MCTS has been successfully applied in many other domains. The most popular is the game of Go [1], for which MCTS represented a substantial step forward. Other examples are Hex [10], Havannah [11] and Lines of Action [12]. Moreover, the application of MCTS has not been limited to games, but also to other domains like combinatorial optimization problems, constraint satisfaction problems, scheduling problems, sample-based planning and procedural content generation [4].

Previous work [3], [13]–[19] has shown that good search-control mechanisms can consistently improve the overall performance of MCTS. Many enhancements have been proposed to improve different phases of the search. Some have been proposed for particular games as they rely on game-specific knowledge [3]. This makes them less interesting for GGP. Others, instead, are intrinsically domain-independent [13], [15]–[18] or are domain-independent modifications of game-specific methods [14], and are thus suitable for GGP.

Among domain-independent enhancements for the selection phase of MCTS we can find the Rapid Action Value Estimation technique (RAVE) [15], [20], [21]. RAVE has been successfully applied in different domains, like the game of Go [15], [20], and General Game Playing [21]. Recently, a generalization of RAVE has been proposed, the Generalized Rapid Action Value Estimation (GRAVE) [22]. This strategy has been shown to perform better than RAVE on some variants of Go and some other games. This and the fact that it does not necessarily need game-specific knowledge make GRAVE interesting to investigate further in the context of GGP.

The aim of this paper is to compare the performance and the robustness of GRAVE and RAVE for GGP. Moreover, we introduce another variant of GRAVE, called HRAVE, that uses the root history statistics. This enables to verify how performance is influenced by the use of information at different levels (from more local in RAVE to more global in HRAVE, with GRAVE being in between). In addition, we test how the performance of these RAVE variants is influenced by using a more informed play-out strategy instead of the one that chooses random moves. We do so by combining all the three strategies with MAST [13].

This paper is structured as follows. Section II gives an overview of MCTS and the MAST search-control mechanism. Section III describes the RAVE strategy and the variants that we are evaluating. Sections IV and V discuss the experimental setup and the obtained results, respectively. Finally, Section VI gives the conclusions and mentions possible future research.

## II. BACKGROUND

### A. Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) is a simulation-based search method that incrementally builds a tree representation of the search space for a game [1]–[3]. Each iteration of the algorithm performs a complete simulation of the game from

the root state to a terminal state, adding nodes to the tree after each simulation and collecting information about the game in every node. More precisely, each iteration of the MCTS algorithm consists of the following four phases:

- *Selection*: the algorithm descends the tree built so far until it reaches a node that needs expansion. At each node it uses a *selection strategy* to determine which move to visit next. The standard MCTS selection strategy is the Upper Confidence bounds applied to Trees (UCT) [2]. Given a state  $s$  and the set  $A(s)$  of all legal moves in  $s$ , it selects the most promising move  $a^*$  as follows:

$$a^* = \operatorname{argmax}_{a \in A(s)} \left\{ Q(s, a) + C \times \sqrt{\frac{\ln N(s)}{N(s, a)}} \right\} \quad (1)$$

$N(s)$  is the number of times node  $s$  has been visited during the search,  $N(s, a)$  is the number of times move  $a$  has been selected whenever node  $s$  was visited and  $Q(s, a)$  is the average result obtained for all the simulations in which move  $a$  was played in state  $s$ . The second term of the formula is used to balance the exploitation of the estimated best move and the exploration of less visited moves. The constant  $C$  controls this balance.

- *Expansion*: this phase controls the expansion of the tree. An *expansion strategy* chooses which node(s) must be added to the tree and when. A common strategy is the one that expands the first encountered node that has at least one unexplored move. The node corresponding to the state reached by playing this move is added to the tree. If there is more than one unvisited move, one of them is chosen randomly. Other strategies might add more than one node at a time or prefer the selection of a promising visited move even if there are unvisited moves in the node. Other expansion strategies are discussed in [23].
- *Play-out*: starting from the last node added to the tree, the algorithm plays the game until a terminal state is reached. In each state the algorithm uses a *play-out strategy* to choose the move to play. One of the basic play-out strategies consists in selecting a move uniformly at random among the legal moves in the considered state.
- *Backpropagation*: after reaching a terminal state, the result of the simulation is propagated back through all the nodes traversed in the tree. The information memorized in the nodes depends on what the simulation and play-out strategies need. Usually, for UCT, each node  $s$  memorizes the values used in Formula (1).

After a certain number of iterations, the best move in the root node is chosen to be played in the real game. The meaning of *best move* depends on the implementation. It could be, for example, the one with the highest number of visits or the one with the highest average score. In this paper we consider the one with the highest average score.

#### B. The MAST play-out strategy in GGP

The Move Average Sampling Technique (MAST) [13], [16] is among the successful domain-independent search-control

mechanisms proposed to guide the search during the play-out phase of MCTS. The main idea behind MAST is that a move that is good in one state is highly likely to be good also in other states. During the search this strategy memorizes for each move  $a$  a global average value  $Q_{MAST}(a)$  based on the results of all the simulations in which move  $a$  was played. The original version of MAST was using  $Q_{MAST}(a)$  in the Gibbs measure to compute a probability distribution over all the moves in a state and then select one of them according to this distribution. Later research [17], [18] has shown that an  $\epsilon$ -greedy strategy that chooses the move with highest  $Q_{MAST}(a)$  with probability  $(1 - \epsilon)$  and a random move with probability  $\epsilon$  performs significantly better in most of the tested games.

A variant of MAST, called NST, has been proposed by Tak *et al.* [17]. The NST play-out strategy keeps also track of statistics of sequences of moves. This strategy has been shown to outperform MAST in most of the tested games.

A characteristic of both MAST and NST is that they keep track of the collected statistics throughout all the game. Further gain in performance has been achieved by decaying such statistics [24]. As the game progresses old statistics might not be as reliable as they were before, they might refer to moves that are strong in some parts of the game but weak in others. Thus, it is desirable to reduce their influence over time.

### III. RAVE, GRAVE AND HRAVE

#### A. RAVE

The RAVE selection strategy has been proposed in order to speed up the learning process inside the MCTS tree [16], [20], [21]. The UCT algorithm bases the selection of a move in a node on the estimated value obtained by sampling this move in the node multiple times. However, especially when the state space is large, the algorithm needs many simulations before it can sample all the moves in a node and more simulations before it can accumulate enough samples for the moves to reduce the variance of their estimated scores. To overcome this issue, RAVE keeps track of other statistics, also known as *All Moves As First (AMAF)* values [25], [26]. In every node it memorizes for all legal moves the following values:

- The average result  $Q(s, a)$ , obtained from all the simulations in which move  $a$  is performed in state  $s$  (the same value used in Formula (1)).
- The average result  $AMAF(s, a)$ , obtained from all the simulations in which move  $a$  is performed further down the path that passes by node  $s$ .

This means that, when backpropagating the result of a simulation in a certain node  $s$  of the tree, the value  $Q(s, a)$  is updated for the move  $a$  that was directly played in the state, and the value  $AMAF(s, a')$  is updated for all the legal moves  $a'$  in  $s$  that have been encountered at a later stage of the simulation. In this way RAVE can collect more samples and use them to reduce the variance of the moves values estimates for the nodes that do not have many visits. Using the AMAF scores enables to gather more information faster, however this information is more global than the local  $Q(a, s)$  scores.

AMAF scores are useful for less visited nodes, but when the number of visits increases, the  $Q(s, a)$  scores become more reliable and the influence of the AMAF scores should progressively decrease. This is why the RAVE algorithm keeps track of the two scores separately and uses a weight  $\beta$  to reduce the importance of the AMAF score over time.

Different variants for the RAVE move evaluation formula and for the  $\beta$  parameter computation have been proposed [11], [15], [20]. This paper uses the same formula that has been first used in GGP by CADIAPLAYER [21]. RAVE selects a move according to (1), where the term  $Q(s, a)$  is substituted by:

$$(1 - \beta(s)) \times Q(s, a) + \beta(s) \times \text{AMAF}(s, a) \quad (2)$$

and the term  $\beta(s)$  is computed as follows:

$$\beta(s) = \sqrt{\frac{K}{3 \times N(s) + K}} \quad (3)$$

where  $N(s)$  is the number of times node  $s$  has been visited and  $K$  is the *equivalence parameter*, that indicates for how many simulations the two scores are weighted equal.

#### B. GRAVE

GRAVE [22] is a modification of RAVE that has been proposed to overcome one of its drawbacks. A problem of RAVE is that for the nodes close to the leaves of the tree not only the  $Q(s, a)$  scores are based on a low number of samples, but also the AMAF scores. In these nodes the estimates of the moves values have less accuracy.

To solve this problem, for the nodes that have a number of visits lower than a given *ref* value GRAVE uses the AMAF scores of an ancestor node. Each node in the tree memorizes its own AMAF scores, but keeps also a reference to the closest ancestor that has a sufficient number of visits for its AMAF scores to be considered reliable. When a node  $s$  has sufficient visits ( $N(s) > \text{ref}$ ), it starts using its own AMAF values instead of the ones of an ancestor, and the algorithm in that node starts behaving like RAVE. Note that, if  $\text{ref} = 0$ , GRAVE behaves exactly like RAVE from the beginning of the search. The GRAVE strategy enables to increase the accuracy of the estimates for the less visited nodes. However, the AMAF scores of an ancestor might be less relevant for its descendants, because these scores refer to a different game state.

Another aspect to be mentioned is the increased memory consumption of GRAVE with respect to RAVE. The latter needs only to store an extra statistic for each legal move in the node. With GRAVE, instead, the AMAF scores in a node might be used for other nodes lower in the tree that have a different set of legal moves. Therefore each node has to memorize the AMAF scores for all the moves that can be encountered at any lower level in the tree.

#### C. HRAVE

HRAVE is exactly the same as GRAVE, except that it always uses the AMAF scores of the current root of the tree (i.e. the *ref* parameter is set to infinity).

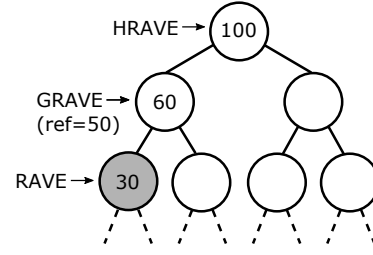


Fig. 1. Information used by RAVE, GRAVE and HRAVE for move selection in the highlighted tree node.

HRAVE shares similarities with the domain-independent selection strategy known as Progressive History [14]. This strategy adds to the UCT formula a bonus that depends on the *relative history* of the move being evaluated. This relative history is defined as the average result of all the simulations where the move was played. The influence of this bonus decreases over time as the number of visits of the node increases and the UCT estimate becomes more reliable.

In the case of HRAVE, the AMAF score of a move that is included in the UCT formula as shown in (2) can be compared to the Progressive History bonus. This is because both the AMAF score and the bonus are computed using the same statistics. In each turn of the game, the AMAF scores of the root of the tree correspond exactly to the history heuristic scores used by Progressive History. Moreover, like in Progressive History, the influence of the AMAF score decreases over time and makes the move evaluation formula converge to a pure UCT strategy.

A difference between HRAVE and Progressive History is that, for HRAVE, at the beginning of the search for a given turn, the root node already contains some statistics collected during previous turns. Progressive History, instead, starts each turn with an empty table. We decided to collect these statistics also during the previous turns to have a fair comparison of HRAVE with GRAVE and RAVE, because both of them, at every turn except the first, start the search already having some statistics in the AMAF tables of the nodes in the tree.

HRAVE can also be seen as the opposite of RAVE. While the latter uses the most local AMAF information, the former uses the most global one. GRAVE can be placed in between, it starts with more global AMAF statistics and then converges to the most local ones. Fig. 1 gives an example when these three heuristics are applied in MCTS. The number reported in each node is the number of node visits. For the selection of a move in the highlighted node, the figure shows in which node each algorithm looks for the AMAF statistics to use.

## IV. EXPERIMENTAL SETUP

#### A. Games

The discussed algorithms have been tested on 15 different games: *3D Tic Tac Toe*, *Breakthrough*, *Knightthrough*, *Skirmish*, *Battle*, *Chinook*, *Chinese Checkers* with three players, *Checkers*, *Connect 5*, *Othello*, *Quad* (the version played on a  $7 \times 7$  board), *Sheep and Wolf*, *Tic Tac Chess Checkers*

TABLE I  
CHARACTERISTICS OF THE GAMES USED FOR THE EXPERIMENTS

Game	Players	Simult.	Constant-Sum
GROUP I			
3D Tic Tac Toe	2	No	Yes
Breakthrough	2	No	Yes
Knightthrough	2	No	Yes
Skirmish	2	No	No
Battle	2	Yes	No
Chinook	2	Yes	No
Chinese Checkers 3P	3	No	No
GROUP II			
Checkers	2	No	Yes
Connect 5	2	No	Yes
Othello	2	No	Yes
Quad	2	No	Yes
Sheep and Wolf	2	No	Yes
TTCC4 2P	2	No	No
Zhadu	2	No	Yes
TTCC4 3P	3	No	No

Four (TTCC4) with two and three players, and Zhadu. Table I gives an overview of the main characteristics of these games specifying the number of players, if they are sequential or simultaneous move games, and if they are constant or variable sum games. This set of games has been chosen because it is heterogeneous and because most of the games have been used in previous experiments that applied RAVE to GGP [16], [21].

In the following experiments, the games in Group I of Table I have also been used for tuning the *equivalence parameter*  $K$  for the algorithms. The games in Group II, instead, have only been used for comparing the strengths of RAVE, GRAVE and HRAVE. The GDL description of all the considered games can be found on the GGP-Base repository [27].

### B. Setup

The aforementioned RAVE variants were implemented in the General Game Playing code base provided by the open-source GGP-Base project [7]. The code is implemented in Java and each agent tested in the experiments uses a reasoner based on Propositional Networks (PropNet, cfr. [28]).

In all the series of experiments, two agent types at a time are matched against each other. For each match, the PropNet of the game is generated in advance and both agents use the same so that none of them has any advantage for having a faster structure. Play clock and start clock are set to 1s, except for the experiments presented in Subsection V-B that are repeated also with start clock and play clock set to 10s.

For each game, if  $r$  is the number of roles in the game, there are  $2^r$  different ways in which 2 types of agents can be assigned to the roles [29]. Two of the configurations involve only the same agent type assigned to all the roles, thus are not interesting and excluded from the experiments. Each configuration is run the same amount of times until the desired number of matches have been played.

For each of the performed experiments, we report as results the average winning percentage of one of the two involved agents with a 95% confidence interval. For each match the agent that achieved the highest score is considered the winner.

When both agent types achieve the same score, the outcome of the match is considered a draw. In the first case, the winning player gets 1 point (full win) and the other player 0 points. In case of a draw both agent types get 0.5 points (half win).

As baseline to compare the different selection policies we have used an agent implementing the MCTS algorithm with UCT selection and random play-out strategy ( $P_{UCT}$ ) and an agent implementing the MCTS algorithm with UCT selection and MAST play-out strategy ( $P_{UCT-MAST}$ ). The UCT selection uses the formula given in (1), with  $C = 0.7$ . For the MAST strategy  $\epsilon$  is set to 0.4, because it is the value that overall performed better in [17]. Moreover, the MAST statistics are decayed after playing every move with a factor  $\gamma = 0.2$  (i.e. 20% of the statistics is kept for the next turn). This value is set lower than the one that was found to be the best in [24] because for each turn we have a higher number of simulations. This means that the number of collected statistics is higher and their influence needs to be decreased more strongly.

The aim of the first series of experiments is to tune the *equivalence parameter*  $K$  used to compute the weight  $\beta(s)$  in (3). The tested values for  $K$  are 10, 50, 100, 250, 500, 750, 1000 and 2000 and the parameter is tuned using the games in Group I shown in Table I. The agents  $P_{RAVE}$ ,  $P_{GRAVE}$  and  $P_{HRAVE}$  have been implemented and matched singularly against  $P_{UCT}$  for each value of  $K$  for at least 500 matches per game. As selection strategy they use the RAVE, GRAVE and HRAVE algorithm, respectively. They all use the random play-out strategy. All of them use the value 0.2 for the  $C$  constant because a lower value than the one used for the plain UCT algorithm empirically showed to achieve a better performance. For  $P_{GRAVE}$  the *ref* parameter is set to 50. For each of the three agents, the value of  $K$  that performed overall best in these series of experiments is also used in subsequent experiments.

In the second series of experiments, the agents  $P_{RAVE}$ ,  $P_{GRAVE}$  and  $P_{HRAVE}$  with the best value of  $K$  are matched against  $P_{UCT}$  on all the games in Table I. Testing the agents on a wider set of games enables to detect a potential overfitting of the  $K$  value to the games used for tuning. Moreover, it enables to check whether the tuned value works well also on other games. These experiments are performed with a start clock and play clock of 1s and then repeated with a start clock and play clock of 10s. This is to verify how an increased amount of time, and thus of simulations, influences the performance of the three RAVE variants. The minimum number of played matches per game is increased to 1000. This provides a more precise estimate of the average winning percentage of the agents, detecting with a higher confidence which of the algorithms performs best.

The aim of the third series of experiments is to verify the effect that the addition of the MAST play-out strategy has on the three variants of RAVE. For this series of experiments the random play-out strategy has been replaced with MAST to obtain the agents  $P_{RAVE-MAST}$ ,  $P_{GRAVE-MAST}$  and  $P_{HRAVE-MAST}$ . These agents have been matched only for the best value of  $K$  against  $P_{UCT-MAST}$  on all the games in Table I with 1000 matches per game. Each of these agents has the same

TABLE II  
WIN% OF  $P_{RAVE}$ ,  $P_{GRAVE}$  AND  $P_{HRAVE}$  AGAINST  $P_{UCT}$  FOR DIFFERENT VALUES OF  $K$  FOR THE GAMES IN GROUP I

Game	$K = 10$	$K = 50$	$K = 100$	$K = 250$	$K = 500$	$K = 750$	$K = 1000$	$K = 2000$
<b><math>P_{RAVE}</math> vs <math>P_{UCT}</math></b>								
3D Tic Tac Toe	68.7( $\pm 4.06$ )	70.2( $\pm 4.00$ )	72.3( $\pm 3.91$ )	80.3( $\pm 3.47$ )	75.0( $\pm 3.78$ )	81.9( $\pm 3.36$ )	79.5( $\pm 3.53$ )	74.4( $\pm 3.81$ )
Breakthrough	58.2( $\pm 4.33$ )	60.6( $\pm 4.29$ )	63.8( $\pm 4.22$ )	65.8( $\pm 4.16$ )	72.2( $\pm 3.93$ )	72.0( $\pm 3.94$ )	71.6( $\pm 3.96$ )	65.8( $\pm 4.16$ )
Knightthrough	68.4( $\pm 4.08$ )	70.8( $\pm 3.99$ )	71.4( $\pm 3.96$ )	73.6( $\pm 3.87$ )	70.6( $\pm 4.00$ )	71.2( $\pm 3.97$ )	71.2( $\pm 3.97$ )	70.8( $\pm 3.99$ )
Skirmish	64.7( $\pm 4.16$ )	57.1( $\pm 4.28$ )	53.5( $\pm 4.34$ )	49.3( $\pm 4.35$ )	41.2( $\pm 4.26$ )	41.9( $\pm 4.27$ )	40.8( $\pm 4.27$ )	39.7( $\pm 4.23$ )
Battle	58.0( $\pm 3.83$ )	60.2( $\pm 3.78$ )	54.3( $\pm 3.86$ )	57.2( $\pm 3.81$ )	55.8( $\pm 3.88$ )	58.0( $\pm 3.85$ )	54.0( $\pm 3.94$ )	52.5( $\pm 4.00$ )
Chinook	45.2( $\pm 4.04$ )	51.5( $\pm 4.06$ )	55.2( $\pm 4.10$ )	59.2( $\pm 4.05$ )	57.2( $\pm 4.09$ )	59.3( $\pm 4.01$ )	55.9( $\pm 4.12$ )	52.4( $\pm 4.11$ )
Chinese Checkers 3P	63.9( $\pm 4.20$ )	61.1( $\pm 4.26$ )	63.9( $\pm 4.20$ )	58.7( $\pm 4.30$ )	64.3( $\pm 4.19$ )	64.9( $\pm 4.17$ )	59.6( $\pm 4.28$ )	58.5( $\pm 4.31$ )
Robustness	5	6	6	6	5	5	5	3
Avg Win%	61.0	61.6	62.1	63.4	62.3	64.2	61.8	59.2
<b><math>P_{GRAVE}</math> vs <math>P_{UCT}</math></b>								
3D Tic Tac Toe	63.5( $\pm 4.21$ )	71.4( $\pm 3.96$ )	75.0( $\pm 3.78$ )	75.3( $\pm 3.78$ )	80.1( $\pm 3.49$ )	80.7( $\pm 3.45$ )	79.9( $\pm 3.51$ )	77.9( $\pm 3.61$ )
Breakthrough	52.8( $\pm 4.38$ )	58.4( $\pm 4.32$ )	61.2( $\pm 4.28$ )	65.0( $\pm 4.19$ )	67.8( $\pm 4.10$ )	68.6( $\pm 4.07$ )	65.2( $\pm 4.18$ )	62.2( $\pm 4.25$ )
Knightthrough	72.6( $\pm 3.91$ )	72.0( $\pm 3.94$ )	74.0( $\pm 3.85$ )	71.2( $\pm 3.97$ )	70.6( $\pm 4.00$ )	74.4( $\pm 3.83$ )	68.0( $\pm 4.09$ )	68.6( $\pm 4.07$ )
Skirmish	62.2( $\pm 4.20$ )	57.0( $\pm 4.28$ )	51.2( $\pm 4.34$ )	55.7( $\pm 4.30$ )	46.1( $\pm 4.31$ )	44.6( $\pm 4.27$ )	42.0( $\pm 4.28$ )	42.2( $\pm 4.28$ )
Battle	68.7( $\pm 3.38$ )	72.7( $\pm 3.33$ )	71.7( $\pm 3.29$ )	69.6( $\pm 3.36$ )	71.6( $\pm 3.31$ )	72.6( $\pm 3.25$ )	69.6( $\pm 3.46$ )	67.5( $\pm 3.46$ )
Chinook	55.0( $\pm 4.08$ )	64.4( $\pm 4.00$ )	66.6( $\pm 3.89$ )	67.3( $\pm 3.80$ )	69.6( $\pm 3.80$ )	70.5( $\pm 3.70$ )	66.5( $\pm 3.87$ )	64.2( $\pm 3.94$ )
Chinese Checkers 3P	63.9( $\pm 4.20$ )	67.5( $\pm 4.09$ )	63.3( $\pm 4.21$ )	63.6( $\pm 4.20$ )	60.0( $\pm 4.28$ )	64.9( $\pm 4.17$ )	62.5( $\pm 4.23$ )	57.7( $\pm 4.32$ )
Robustness	6	7	6	7	6	5	5	5
Avg Win%	62.7	66.2	66.1	66.8	66.5	68.0	64.8	62.9
<b><math>P_{HRAVE}</math> vs <math>P_{UCT}</math></b>								
3D Tic Tac Toe	66.5( $\pm 4.12$ )	63.1( $\pm 4.22$ )	71.9( $\pm 3.93$ )	76.1( $\pm 3.74$ )	76.1( $\pm 3.71$ )	75.4( $\pm 3.75$ )	77.0( $\pm 3.67$ )	68.5( $\pm 4.04$ )
Breakthrough	53.6( $\pm 4.38$ )	57.0( $\pm 4.34$ )	62.6( $\pm 4.25$ )	65.2( $\pm 4.18$ )	65.4( $\pm 4.17$ )	60.4( $\pm 4.29$ )	63.2( $\pm 4.23$ )	59.2( $\pm 4.31$ )
Knightthrough	74.0( $\pm 3.85$ )	72.4( $\pm 3.92$ )	74.0( $\pm 3.85$ )	77.4( $\pm 3.67$ )	74.0( $\pm 3.85$ )	73.8( $\pm 3.86$ )	70.4( $\pm 4.01$ )	68.2( $\pm 4.09$ )
Skirmish	62.6( $\pm 4.21$ )	57.4( $\pm 4.31$ )	52.0( $\pm 4.33$ )	48.9( $\pm 4.33$ )	46.2( $\pm 4.32$ )	41.8( $\pm 4.26$ )	44.2( $\pm 4.30$ )	37.0( $\pm 4.18$ )
Battle	72.3( $\pm 3.21$ )	75.7( $\pm 3.25$ )	73.2( $\pm 3.17$ )	69.2( $\pm 3.37$ )	70.9( $\pm 3.34$ )	67.4( $\pm 3.44$ )	73.5( $\pm 3.26$ )	69.4( $\pm 3.48$ )
Chinook	54.9( $\pm 4.10$ )	66.0( $\pm 3.89$ )	66.4( $\pm 3.94$ )	74.9( $\pm 3.54$ )	72.9( $\pm 3.58$ )	75.4( $\pm 3.55$ )	73.4( $\pm 3.63$ )	73.8( $\pm 3.61$ )
Chinese Checkers 3P	67.9( $\pm 4.08$ )	64.1( $\pm 4.19$ )	66.3( $\pm 4.13$ )	65.5( $\pm 4.16$ )	62.7( $\pm 4.23$ )	60.3( $\pm 4.28$ )	61.3( $\pm 4.26$ )	60.2( $\pm 4.27$ )
Robustness	6	7	6	6	6	5	5	5
Avg Win%	64.5	65.1	66.6	68.2	66.9	64.9	66.1	62.3

settings of the corresponding version without MAST and for the MAST strategy the settings are the same as  $P_{UCT}$ -MAST.

As a validation of the results obtained in the previous series of experiments, the last series of experiments matches  $P_{RAVE}$ ,  $P_{GRAVE}$  and  $P_{HRAVE}$  against each other two at a time and  $P_{RAVE}$ -MAST,  $P_{GRAVE}$ -MAST and  $P_{HRAVE}$ -MAST against each other two at a time. A total of at least 1000 matches per game have been played. All the experiments presented in the next sections were performed on a Linux server consisting of 64 AMD Opteron 6174 2.2-GHz cores.

## V. EMPIRICAL EVALUATION

### A. Parameter tuning

Table II shows the performance of  $P_{RAVE}$ ,  $P_{GRAVE}$  and  $P_{HRAVE}$  against  $P_{UCT}$  for different values of  $K$ . For each agent, the value of  $K$  that achieves the highest robustness is selected to be used in subsequent experiments. We compute the robustness of a certain  $K$  for an agent by summing 1 point for each game in which the agent with such  $K$  achieved a statistically significant improvement over  $P_{UCT}$  and subtracting 1 point for each game in which it obtained a statistically significant worsening of the performance. In case more values of  $K$  have the same robustness, we chose the one with highest average win percentage over all the games.

For  $P_{RAVE}$  none of the values of  $K$  reaches the maximum robustness, however, for more than one value the agent achieves a statistically significant improvement in all games but one.

TABLE III  
SIMULATIONS PER SECOND OF  $P_{UCT}$ ,  $P_{RAVE}$ ,  $P_{GRAVE}$  AND  $P_{HRAVE}$

Game	$P_{UCT}$	$P_{RAVE}$	$P_{GRAVE}$	$P_{HRAVE}$
3D Tic Tac Toe	3093	2831	2920	2877
Breakthrough	1453	1378	1430	1435
Knightthrough	2285	2100	2146	2210
Skirmish	106	105	104	106
Battle	2149	2001	1898	1916
Chinook	2178	2085	2150	2144
Chinese Checkers 3P	4995	4108	4235	4229
Checkers	532	518	511	518
Connect 5	1191	1160	1144	1148
Othello	39	39	39	38
Quad	2767	2617	2627	2684
Sheep And Wolf	2110	2071	2063	2097
TTCC4 2P	1124	1277	1321	1368
Zhadu	494	484	477	480
TTCC4 3P	2058	2207	2220	2257

Among these values,  $K = 250$  is chosen because it is the one with the highest average win percentage. For  $P_{GRAVE}$  the value  $K = 250$  is selected because among the values with highest robustness is also the one with highest average win percentage. Finally, for  $P_{HRAVE}$  the value  $K = 50$  is selected because it is the only one that reaches the highest robustness.

### B. Comparison of $P_{RAVE}$ , $P_{GRAVE}$ and $P_{HRAVE}$ with $P_{UCT}$

In this series of experiments,  $P_{RAVE}$ ,  $P_{GRAVE}$  and  $P_{HRAVE}$  are matched against  $P_{UCT}$ , both with 1s and 10s play clock.



TABLE IV

WIN% OF  $P_{RAVE}$ ,  $P_{GRAVE}$  AND  $P_{HRAVE}$  WITH BEST  $K$  AGAINST  $P_{UCT}$  WITH 1S PLAY CLOCK AND START CLOCK

Game	$P_{RAVE}$	$P_{GRAVE}$	$P_{HRAVE}$
3D Tic Tac Toe	<b>78.4</b> ( $\pm 2.54$ )	74.3( $\pm 2.70$ )	64.0( $\pm 2.97$ )
Breakthrough	66.6( $\pm 2.92$ )	<b>67.6</b> ( $\pm 2.90$ )	57.8( $\pm 3.06$ )
Knightthrough	73.0( $\pm 2.75$ )	<b>73.6</b> ( $\pm 2.73$ )	71.3( $\pm 2.81$ )
Skirmish	47.5( $\pm 3.07$ )	54.5( $\pm 3.05$ )	<b>59.3</b> ( $\pm 3.02$ )
Battle	57.0( $\pm 2.69$ )	69.7( $\pm 2.34$ )	<b>73.7</b> ( $\pm 2.29$ )
Chinook	59.6( $\pm 2.84$ )	<b>68.3</b> ( $\pm 2.71$ )	65.1( $\pm 2.74$ )
Chinese Checkers 3P	61.9( $\pm 3.00$ )	63.2( $\pm 2.98$ )	<b>64.4</b> ( $\pm 2.96$ )
Checkers	63.5( $\pm 2.83$ )	<b>70.8</b> ( $\pm 2.65$ )	60.7( $\pm 2.84$ )
Connect 5	70.8( $\pm 2.76$ )	<b>75.5</b> ( $\pm 2.62$ )	66.8( $\pm 2.89$ )
Othello	36.9( $\pm 2.96$ )	42.9( $\pm 2.99$ )	<b>57.4</b> ( $\pm 3.02$ )
Quad	<b>75.1</b> ( $\pm 2.67$ )	73.6( $\pm 2.72$ )	73.3( $\pm 2.73$ )
Sheep And Wolf	<b>66.0</b> ( $\pm 2.94$ )	62.9( $\pm 3.00$ )	56.7( $\pm 3.07$ )
TTCC4 2P	<b>72.9</b> ( $\pm 2.73$ )	71.2( $\pm 2.77$ )	62.3( $\pm 3.00$ )
Zhadu	69.3( $\pm 2.86$ )	67.4( $\pm 2.91$ )	<b>71.3</b> ( $\pm 2.80$ )
TTCC4 3P	52.1( $\pm 3.03$ )	52.6( $\pm 3.03$ )	<b>53.4</b> ( $\pm 3.05$ )
Robustness	11	12	<b>15</b>
Avg Win%	63.4	<b>65.9</b>	63.8

Table III reports for each game the average median number of simulations per second that each of the agents can perform.

Table IV shows the performance of  $P_{RAVE}$ ,  $P_{GRAVE}$  and  $P_{HRAVE}$  with the best  $K$  against  $P_{UCT}$  with 1s play clock and start clock.  $P_{HRAVE}$  is the only one that achieves a significant improvement over  $P_{UCT}$  in all games, despite not being the one with the highest average win percentage.  $P_{RAVE}$  and  $P_{GRAVE}$  still obtain a significant improvement in most of the games, only in *Othello* they are significantly outperformed by  $P_{UCT}$ .

Table V shows the results obtained by repeating the experiment with 10s play clock and start clock. The results of  $P_{HRAVE}$  with  $K = 50$  were noticeably lower (robustness = 7, average win percentage = 53.0) than the ones of  $P_{RAVE}$  and  $P_{GRAVE}$  with their best  $K$ . For this reason, the experiment for  $P_{HRAVE}$  was repeated with the value that produced the highest average win percentage in Table II,  $K = 250$ . Such results, being better than the ones of  $K = 50$ , are reported in Table V.

As can be seen, in most of the games the longer search time reduces the performance increase of  $P_{RAVE}$ ,  $P_{GRAVE}$  and  $P_{HRAVE}$  against  $P_{UCT}$ . In *Quad* it even makes the use of RAVE, GRAVE and HRAVE detrimental, substantially reducing the win percentage around 10%. In *Knightthrough* and *Othello* instead, it seems that more search time increases the performance of all the three RAVE variants.

### C. Comparison of $P_{RAVE-MAST}$ , $P_{GRAVE-MAST}$ and $P_{HRAVE-MAST}$ with $P_{UCT-MAST}$

Table VI shows the performance of  $P_{RAVE-MAST}$ ,  $P_{GRAVE-MAST}$  and  $P_{HRAVE-MAST}$  with the best  $K$  against  $P_{UCT-MAST}$ . For most of the games the addition of MAST as play-out strategy seems to benefit more  $P_{UCT-MAST}$ .  $P_{RAVE-MAST}$ ,  $P_{GRAVE-MAST}$  and  $P_{HRAVE-MAST}$  perform significantly better than  $P_{UCT-MAST}$  in most of the games. However, for many of these games the difference in performance achieved by  $P_{RAVE-MAST}$ ,  $P_{GRAVE-MAST}$  and  $P_{HRAVE-MAST}$  against  $P_{UCT-MAST}$  is not as high as the difference in performance achieved by  $P_{RAVE}$ ,  $P_{GRAVE}$  and

TABLE V

WIN% OF  $P_{RAVE}$ ,  $P_{GRAVE}$  AND  $P_{HRAVE}$  WITH  $K = 250$  AGAINST  $P_{UCT}$  WITH 10S PLAY CLOCK AND START CLOCK

Game	$P_{RAVE}$	$P_{GRAVE}$	$P_{HRAVE}$
3D Tic Tac Toe	<b>69.7</b> ( $\pm 2.81$ )	68.2( $\pm 2.86$ )	60.7( $\pm 2.99$ )
Breakthrough	<b>67.5</b> ( $\pm 2.90$ )	65.1( $\pm 2.96$ )	60.4( $\pm 3.03$ )
Knightthrough	<b>84.8</b> ( $\pm 2.23$ )	84.1( $\pm 2.27$ )	84.4( $\pm 2.25$ )
Skirmish	<b>60.2</b> ( $\pm 3.01$ )	60.0( $\pm 3.00$ )	55.8( $\pm 3.07$ )
Battle	<b>59.1</b> ( $\pm 2.19$ )	57.2( $\pm 2.29$ )	54.6( $\pm 2.35$ )
Chinook	39.8( $\pm 2.78$ )	56.6( $\pm 2.84$ )	<b>71.8</b> ( $\pm 2.52$ )
Chinese Checkers 3P	54.0( $\pm 3.08$ )	<b>54.7</b> ( $\pm 3.07$ )	49.7( $\pm 3.09$ )
Checkers	52.2( $\pm 2.77$ )	56.3( $\pm 2.73$ )	<b>61.8</b> ( $\pm 2.69$ )
Connect 5	<b>66.9</b> ( $\pm 2.36$ )	59.9( $\pm 2.50$ )	53.3( $\pm 2.47$ )
Othello	61.8( $\pm 2.97$ )	<b>62.0</b> ( $\pm 2.97$ )	60.6( $\pm 2.97$ )
Quad	<b>10.7</b> ( $\pm 1.87$ )	8.5( $\pm 1.68$ )	7.9( $\pm 1.64$ )
Sheep And Wolf	<b>69.6</b> ( $\pm 2.85$ )	69.0( $\pm 2.87$ )	67.2( $\pm 2.91$ )
TTCC4 2P	61.1( $\pm 2.90$ )	<b>66.4</b> ( $\pm 2.80$ )	65.7( $\pm 2.80$ )
Zhadu	63.4( $\pm 2.94$ )	66.5( $\pm 2.86$ )	<b>68.5</b> ( $\pm 2.82$ )
TTCC4 3P	54.4( $\pm 2.97$ )	<b>58.5</b> ( $\pm 2.95$ )	50.3( $\pm 3.02$ )
Robustness	10	<b>13</b>	11
Avg Win%	58.3	<b>59.5</b>	58.2

$P_{HRAVE}$  against  $P_{UCT}$ . Some examples are the games *3D Tic Tac Toe*, *Connect 5* and *TTCC4* with 2 players.

The game for which MAST has the highest benefit on  $P_{UCT}$  is *Quad*. In this game  $P_{RAVE}$ ,  $P_{GRAVE}$  and  $P_{HRAVE}$  were previously obtaining an improvement over  $P_{UCT}$ , while the corresponding agents with MAST are realizing a decrease in performance with respect to  $P_{UCT-MAST}$ .

Among the RAVE variants, the one that seems to benefit the most (in about half of the games) from the use of MAST is RAVE. This could be explained by considering that the AMAF scores used by RAVE in the nodes with a low number of visits only have a small number of samples. MAST can compensate the lack of local information near the leaf nodes of the tree. Using its global statistics, MAST steers the simulations towards more promising parts of the state space during the play-out improving its quality. The quality of a simulation for GRAVE and HRAVE, instead, is already improved near the leaf nodes by the use of the AMAF statistics of an ancestor.

TABLE VI

WIN% OF  $P_{RAVE-MAST}$ ,  $P_{GRAVE-MAST}$  AND  $P_{HRAVE-MAST}$  WITH BEST  $K$  AGAINST  $P_{UCT-MAST}$ 

Game	$P_{RAVE-MAST}$	$P_{GRAVE-MAST}$	$P_{HRAVE-MAST}$
3D Tic Tac Toe	64.9( $\pm 2.76$ )	<b>65.3</b> ( $\pm 2.75$ )	57.3( $\pm 2.89$ )
Breakthrough	<b>78.5</b> ( $\pm 2.55$ )	74.6( $\pm 2.70$ )	72.3( $\pm 2.78$ )
Knightthrough	<b>81.9</b> ( $\pm 2.39$ )	74.7( $\pm 2.70$ )	75.6( $\pm 2.66$ )
Skirmish	56.1( $\pm 3.04$ )	53.6( $\pm 3.04$ )	<b>64.9</b> ( $\pm 2.92$ )
Battle	72.5( $\pm 2.32$ )	76.9( $\pm 2.20$ )	<b>80.8</b> ( $\pm 2.03$ )
Chinook	32.2( $\pm 2.60$ )	<b>61.3</b> ( $\pm 2.85$ )	58.3( $\pm 2.91$ )
Chinese Checkers 3P	<b>58.7</b> ( $\pm 3.04$ )	57.5( $\pm 3.05$ )	56.1( $\pm 3.07$ )
Checkers	65.1( $\pm 2.80$ )	<b>67.1</b> ( $\pm 2.74$ )	59.1( $\pm 2.85$ )
Connect 5	<b>60.2</b> ( $\pm 2.25$ )	58.4( $\pm 2.29$ )	46.6( $\pm 2.41$ )
Othello	36.8( $\pm 2.94$ )	42.6( $\pm 3.00$ )	<b>50.1</b> ( $\pm 3.06$ )
Quad	<b>34.5</b> ( $\pm 2.80$ )	29.2( $\pm 2.65$ )	29.8( $\pm 2.67$ )
Sheep And Wolf	56.3( $\pm 3.08$ )	56.6( $\pm 3.07$ )	<b>57.3</b> ( $\pm 3.07$ )
TTCC4 2P	63.3( $\pm 2.92$ )	<b>66.2</b> ( $\pm 2.85$ )	46.6( $\pm 3.04$ )
Zhadu	<b>73.8</b> ( $\pm 2.73$ )	64.8( $\pm 2.96$ )	65.1( $\pm 2.96$ )
TTCC4 3P	<b>56.0</b> ( $\pm 2.98$ )	55.6( $\pm 2.98$ )	55.9( $\pm 3.01$ )
Robustness	9	<b>11</b>	8
Avg Win%	59.4	<b>60.3</b>	58.4

TABLE VII

WIN% OF ALL POSSIBLE COMBINATIONS OF AGENTS WITH AND WITHOUT MAST. THE WIN% ALWAYS REFERS TO THE FIRST OF THE TWO PLAYERS.

Game	P <sub>GRAVE</sub> vs P <sub>RAVE</sub>	P <sub>GRAVE-MAST</sub> vs P <sub>RAVE-MAST</sub>	P <sub>HRAVE</sub> vs P <sub>RAVE</sub>	P <sub>HRAVE-MAST</sub> vs P <sub>RAVE-MAST</sub>	P <sub>GRAVE</sub> vs P <sub>HRAVE</sub>	P <sub>GRAVE-MAST</sub> vs P <sub>HRAVE-MAST</sub>
3D Tic Tac Toe	50.4(±3.09)	51.1(±2.93)	40.1(±3.01)	41.9(±2.89)	63.0(±2.97)	57.1(±2.91)
Breakthrough	46.9(±3.09)	46.1(±3.09)	38.8(±3.02)	44.6(±3.08)	57.1(±3.07)	54.6(±3.09)
Knightthrough	52.8(±3.10)	38.7(±3.02)	49.6(±3.10)	40.3(±3.04)	48.7(±3.10)	45.6(±3.09)
Skirmish	52.3(±3.04)	54.0(±3.04)	62.6(±2.97)	59.5(±3.02)	40.7(±3.02)	44.2(±3.01)
Battle	66.8(±2.34)	54.5(±2.65)	68.4(±2.38)	62.6(±2.52)	50.0(±2.46)	48.4(±2.67)
Chinook	58.3(±2.76)	70.1(±2.41)	55.4(±2.76)	70.8(±2.38)	54.0(±2.79)	49.7(±2.84)
Chinese Checkers 3P	55.1(±3.07)	50.1(±3.09)	55.3(±3.08)	49.4(±3.10)	46.2(±3.09)	50.1(±3.10)
Checkers	53.4(±2.91)	53.1(±2.91)	46.5(±2.94)	43.0(±2.91)	54.3(±2.90)	58.5(±2.89)
Connect 5	57.9(±2.97)	47.3(±2.19)	51.0(±3.04)	39.6(±2.17)	58.7(±2.99)	57.8(±2.22)
Othello	52.8(±3.04)	54.5(±3.04)	65.0(±2.91)	65.0(±2.90)	37.5(±2.95)	36.4(±2.93)
Quad	50.5(±3.09)	42.4(±2.90)	48.9(±3.07)	44.4(±2.89)	52.5(±3.06)	53.4(±2.93)
Sheep And Wolf	51.0(±3.10)	49.2(±3.10)	43.8(±3.08)	48.8(±3.10)	57.2(±3.07)	49.8(±3.10)
TTCC4 2P	52.3(±3.03)	54.2(±2.96)	43.7(±3.03)	37.7(±2.92)	60.9(±2.96)	66.0(±2.85)
Zhadu	50.1(±3.10)	42.0(±3.06)	52.3(±3.10)	40.2(±3.04)	46.7(±3.09)	49.5(±3.10)
TTCC4 3P	48.6(±3.02)	50.0(±2.98)	52.8(±3.04)	50.4(±3.01)	48.7(±3.03)	48.5(±3.01)
Robustness	4	1	0	-4	3	3
Avg Win%	53.3	50.5	51.6	49.2	51.7	51.3

Thus, the addition of MAST in the play-out has less added benefit to the overall simulation quality.

#### D. Matching RAVE variants against each other

As a validation of previous results the agents based on the three RAVE variants have been matched two at a time against each other. Table VII shows the obtained results. For each pair of algorithms the table reports a column of results without MAST and a column with MAST.

These results are in line to what has been observed in previous experiments. P<sub>GRAVE</sub> performs better than P<sub>RAVE</sub> in some games and equally in others. The performance of P<sub>RAVE</sub> and P<sub>HRAVE</sub> is more game-dependent. In some games they perform equally, in games like *3D Tic Tac Toe* and *Breakthrough* P<sub>RAVE</sub> performs best and in games like *Skirmish* and *Battle* P<sub>HRAVE</sub> performs best. A similar game-dependent performance can be observed for P<sub>GRAVE</sub> and P<sub>HRAVE</sub>, but in this case there are more games in which P<sub>GRAVE</sub> performs best. When MAST is added to all the agents, the difference in their performance diminishes. P<sub>GRAVE-MAST</sub> and P<sub>RAVE-MAST</sub> perform similarly, one outperforming the other in a few games and vice-versa. MAST also benefits both P<sub>RAVE-MAST</sub> and P<sub>GRAVE-MAST</sub> against P<sub>HRAVE-MAST</sub>.

Finally, we can compare the results obtained for *Knightthrough* by P<sub>GRAVE</sub> against P<sub>RAVE</sub> with the ones in [22]. It can be noticed that we did not achieve the same performance increase. In [22] the player based on GRAVE achieves a win rate of 67.8% against the one based on RAVE when both players have a limit of 1,000 simulations per turn and a win rate of 67.2% when the limit is 10,000 simulations per turn. However, this might be due to the different formula that we use for  $\beta$  and to the fact that we do not limit the number of simulations per turn but the amount of time. Moreover, our implementation of RAVE is achieving a higher win rate against UCT than in [22], where the win rate of RAVE is 69.4% for 1,000 simulations per turn and 56.2% for 10,000. This, therefore, reduces the potential gain by GRAVE.

TABLE VIII

AVERAGE NUMBER OF MOVE STATISTICS PER NODE OF P<sub>RAVE</sub> AND P<sub>GRAVE</sub>

Game	P <sub>RAVE</sub>	P <sub>GRAVE</sub>
3D Tic Tac Toe	4.58	9.11
Breakthrough	3.49	21.14
Knightthrough	3.16	13.44
Skirmish	4.02	54.38
Battle	8.40	19.92
Chinook	2.46	13.81
Chinese Checkers 3P	2.59	16.01
Checkers	2.58	41.94
Connect 5	4.47	9.69
Othello	2.41	14.87
Quad	3.66	7.57
Sheep and Wolf	2.95	32.04
TTCC4 2P	2.27	28.82
Zhadu	2.73	23.12
TTCC4 3P	2.47	13.32

#### E. Memory usage

As mentioned in Section III-B, GRAVE needs to memorize in each node the AMAF statistics for all the actions that are encountered during every simulation that passes through the node. The RAVE algorithm, instead, only needs to memorize in each node the AMAF statistics for the moves that are legal in the corresponding game state.

Table VIII shows for RAVE and GRAVE the average number of AMAF move statistics that are memorized in each node for every game. These results give an idea of the difference between the algorithms in memory usage. The space required by GRAVE ranges between 2 (in *3D Tic Tac Toe*) to 16 (in *Checkers*) times the space required by RAVE.

## VI. CONCLUSION

In this paper the performance of the GRAVE strategy was compared to the one of the RAVE and the HRAVE strategies. GRAVE was also tested on a larger set of games than the one used in [22] to verify its applicability in the context of GGP.

When combined with a random play-out strategy, we may conclude that the performance of GRAVE is, in the worst case, comparable with the one of RAVE both when using 1s or 10s play clock. Not for all the tested games GRAVE was better than RAVE, but it never had an inferior performance, except in *Connect 5* when using a 10s play clock.

Regarding HRAVE, we may conclude that its performance is more game dependent when a random play-out strategy is used. In some games HRAVE is either better or comparable to RAVE and GRAVE, but there are some games where it performs worse. Moreover, when looking at the average win percentage, in none of the experiments its overall performance proved to be better than both RAVE and GRAVE.

When combined with the MAST play-out strategy, GRAVE still seems to be overall better than RAVE. However, it does not have the same advantage over RAVE that it has when both strategies are combined with the random play-out. MAST, apparently, compensates the lack of information near the leaf nodes for RAVE, closing the performance gap between RAVE and GRAVE. There are also a few games where the combination GRAVE-MAST actually performs worse than RAVE-MAST. Moreover, when using MAST, HRAVE is the strategy that appears to be the least beneficial among the three strategies.

As seen in the experiments, the difference in performance between RAVE, GRAVE and HRAVE is not large. Future research could investigate further the strengths of GRAVE over RAVE and HRAVE by tuning also its *ref* parameter. Moreover, the formula proposed more recently in [15] to compute the  $\beta$  parameter could be tested. According to their findings, with this formula the performance of the three RAVE variants could improve further. Moreover, in this paper we only tested the combination of these strategies with MAST. Other play-out policies might influence them in a different way. Testing the combination with the NST play-out strategy could be an idea for future research.

#### ACKNOWLEDGMENT

This work is funded by the Netherlands Organisation for Scientific Research (NWO) in the framework of the project GoGeneral, grant number 612.001.121.

#### REFERENCES

- [1] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," in *Computers and Games (CG 2006)*, ser. LNCS. Springer, 2007, vol. 4630, pp. 72–83.
- [2] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *Machine Learning: ECML 2006*, ser. LNCS. Springer, 2006, vol. 4212, pp. 282–293.
- [3] G. M. J.-B. Chaslot, M. H. M. Winands, H. J. van den Herik, J. W. H. M. Uiterwijk, and B. Bouzy, "Progressive strategies for Monte-Carlo tree search," *New Mathematics and Natural Computation*, vol. 4, no. 3, pp. 343–357, 2008.
- [4] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavenier, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte Carlo tree search methods," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 1, pp. 1–43, 2012.
- [5] Y. Björnsson and H. Finnsson, "CadiaPlayer: A simulation-based general game player," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 1, no. 1, pp. 4–15, 2009.
- [6] J. Méhat and T. Cazenave, "Ary, a General Game Playing program," in *Board Games Studies Colloquium*, 2010.
- [7] S. Schreiber, "The General Game Playing base package," <https://github.com/ggp-org/ggp-base>.
- [8] S. Darper and A. Rose, "Sancho GGP player," <http://sanchoggp.blogspot.nl/2014/07/sancho-is-ggp-champion-2014.html>.
- [9] R. Emsile, "Galvanise," [https://bitbucket.org/rxe/galvanise\\_v2](https://bitbucket.org/rxe/galvanise_v2).
- [10] B. Arneson, R. B. Hayward, and P. Henderson, "Monte Carlo tree search in Hex," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 2, no. 4, pp. 251–258, 2010.
- [11] F. Teytaud and O. Teytaud, "Creating an upper-confidence-tree program for Havannah," in *Advances in Computer Games*, ser. LNCS. Springer, 2010, vol. 6048, pp. 65–74.
- [12] M. H. M. Winands, Y. Björnsson, and J.-T. Saito, "Monte Carlo tree search in Lines of Action," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 2, no. 4, pp. 239–250, 2010.
- [13] H. Finnsson and Y. Björnsson, "Simulation-based approach to General Game Playing," in *AAAI*, vol. 8, 2008, pp. 259–264.
- [14] J. A. M. Nijssen and M. H. M. Winands, "Enhancements for multi-player Monte-Carlo tree search," in *Computers and Games*, ser. LNCS, H. J. van den Herik, H. Ida, and A. Plaat, Eds. Springer, 2011, vol. 6515, pp. 238–249.
- [15] S. Gelly and D. Silver, "Monte-Carlo tree search and rapid action value estimation in computer Go," *Artificial Intelligence*, vol. 175, no. 11, pp. 1856–1875, 2011.
- [16] H. Finnsson, "Simulation-based General Game Playing," Ph.D. dissertation, School of Computer Science, Reykjavik University, Reykjavik, Iceland, 2012.
- [17] M. J. W. Tak, M. H. M. Winands, and Y. Björnsson, "N-grams and the Last-Good-Reply policy applied in General Game Playing," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 2, pp. 73–83, 2012.
- [18] E. J. Powley, D. Whitehouse, and P. I. Cowling, "Bandits all the way down: UCB1 as a simulation policy in Monte Carlo tree search," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013, pp. 81–88.
- [19] E. J. Powley, P. I. Cowling, and D. Whitehouse, "Information capture and reuse strategies in Monte Carlo tree search, with applications to games of hidden information," *Artificial Intelligence*, vol. 217, pp. 92–116, 2014.
- [20] S. Gelly and D. Silver, "Combining online and offline knowledge in UCT," in *Proceedings of the 24th International Conference on Machine Learning*. ACM, 2007, pp. 273–280.
- [21] H. Finnsson and Y. Björnsson, "Learning simulation control in general game-playing agents," in *AAAI*, vol. 10, 2010, pp. 954–959.
- [22] T. Cazenave, "Generalized rapid action value estimation," in *Proceedings of the 24th International Joint Conference on Artificial Intelligence*. AAAI Press, 2015, pp. 754–760.
- [23] T. Yajima, T. Hashimoto, T. Matsui, J. Hashimoto, and K. Spoerer, "Node-expansion operators for the UCT algorithm," in *Computers and Games*, ser. LNCS, H. J. van den Herik, H. Ida, and A. Plaat, Eds. Springer, 2011, vol. 6515, pp. 116–123.
- [24] M. J. W. Tak, M. H. M. Winands, and Y. Björnsson, "Decaying simulation strategies," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 6, no. 4, pp. 395–406, 2014.
- [25] B. Brüggmann, "Monte Carlo Go," Max Planck Institute of Physics, München, Germany, Tech. Rep., 1993.
- [26] B. Bouzy and B. Helmstetter, "Monte-Carlo Go developments," in *Advances in Computer Games Many Games, Many Challenges*, ser. IFIP, vol. 263. Kluwer Academic, 2003, pp. 159–174.
- [27] S. Schreiber, "Games - Base repository," <http://games.ggp.org/base/>, 2016.
- [28] C. F. Sironi and M. H. M. Winands, "Optimizing propositional networks," in *Proceedings of the IJCAI Workshop on General Intelligence in Game-Playing Agents (GIGA)*, 2016, pp. 7–14.
- [29] N. Sturtevant, "A comparison of algorithms for multi-player games," in *Computers and Games (CG 2002)*, ser. LNCS. Springer Berlin Heidelberg, 2003, vol. 2883, pp. 108–122.

# Analyzing the Robustness of General Video Game Playing Agents

Diego Pérez-Liébana	Spyridon Samothrakis	Julian Togelius	Tom Schaul	Simon M. Lucas
University of Essex	University of Essex	New York University	Google DeepMind	University of Essex
Colchester CO4 3SQ	Colchester CO4 3SQ	715 Broadway, 10003	5 New Street Square	Colchester CO4 3SQ
United Kingdom	United Kingdom	New York	London EC4A 3TW	United Kingdom
dperez@essex.ac.uk	ssamot@essex.ac.uk	julian@togelius.com	schaul@google.com	sml@essex.ac.uk

**Abstract**—This paper presents a study on the robustness and variability of performance of general video game-playing agents. Agents analyzed includes those that won the different legs of the 2014 and 2015 General Video Game AI Competitions, and two sample agents distributed with its framework. Initially, these agents are run in four games and ranked according to the rules of the competition. Then, different modifications to the reward signal of the games are proposed and noise is introduced in either the actions executed by the controller, their forward model, or both. Results show that it is possible to produce a significant change in the rankings by introducing the modifications proposed here. This is an important result because it enables the set of human-authored games to be automatically expanded by adding parameter-varied versions that add information and insight into the relative strengths of the agents under test. Results also show that some controllers perform well under almost all conditions, a testament to the robustness of the GVGAI benchmark.

## I. INTRODUCTION: GAMES AND COMPETITIONS

Evaluation of algorithms using games and competitions is a common practice in the Game AI community, and to a certain extent in the wider AI community. Games provide parameterizable benchmarks that allow for fast experimentation with multiple approaches, while competitions establish a common framework and set of rules to guarantee that these algorithms are compared in a fair manner [1].

Recently, a new general framework for creating and playing video games was introduced [2], [3], [4], accompanied by a competition [5], [6]. This framework is called the General Video Game AI Framework, and the competition the General Video Game AI Competition; both are abbreviated “GVGAI”. A main feature of the framework is to allow for the creation of arbitrary games in a high-level game-specific language, which can then be used as benchmarks for artificial (and maybe real?) agents. A distinct advantage of GVGAI over other benchmarks is the possibility to generate/create new games in addition to using a pre-existing set of older, established games (as, for example, is done in the very popular Arcade Learning Environment [7]). Additionally, one can systematically vary certain qualities of the games involved and examine how different controllers react. One could even go a step further and design games that embody specific qualities that would advantage or disadvantage certain agent creation methods. Until now, this ability of the GVGAI framework has not been explored; we have not seen either carefully tuned games

aiming to portray different agent qualities, or any exploitation of the ability to modify any of the properties of the games.

It is well-known that some game-playing methods are more robust to imperfections in the sensors or forward model, noise or hidden information than others. For example, A\* can play Super Mario Bros near-optimally given linear levels, but tends to create “brittle” plans that rely on planned actions executing perfectly. Monte Carlo Tree Search, with its stochastic estimates of action values, struggles to keep up with A\*-based planning under normal conditions. However, when noise is introduced to the model the performance of A\* drops drastically whereas MCTS performs almost as well as before [8].

An important part of the justification for GVGAI in particular and general game playing in general is that the agents’ *general* intelligence is tested, as agent developers cannot tailor their performance to a particular game. That’s why agents are tested on unseen games, which are developed for each round of the competition. However, the developers of agents could still rely on certain assumptions about the GVGAI game engine, for example about the determinism of games and reliability of the forward model. Arguably, agents that are less dependent on such assumptions—less brittle—are more generally capable or “intelligent”. The obvious way to find out how brittle agents are is to vary all aspects of the game engine and see what happens to the performance of said agents.

This paper is an initial exploration of the effects of large-scale modification of game characteristics. The goal is to identify how robust game-playing algorithms are to particular changes in the reward structure and the existence of uncertainty in the form of noise.

While, to the best of our knowledge, this is the first time that such a systematic exploration is conducted in such a large number of games, with the explicit aim of testing robustness, there has been some work generating games using a parameter space and then using controllers that portrayed certain human-like qualities in order to better understand the resulting design parameter space [9]; once game-space is understood, it can be searched for game variants that differ from existing games while still being playable [10].

The rest of the paper is organised as follows; section II describes the framework used, while Section III introduces

a selection of controllers that we are going to use in our evaluation. Section IV discusses the original rankings obtained by the controllers presented previously in a subset of GVGA games. Section V describes the modifications done to the games and how each controller fared. We conclude with a short discussion in Section VI.

## II. GENERAL VIDEO GAME AI

### A. The Framework

The GVGA framework provides information about the game state via Java game objects. Its interface provides means to create agent controllers that can play in any game defined in the Video Game Description Language (VGDL [2], [4]). An agent implemented in this environment must be able to select moves in real-time, providing a valid action in no more than 40ms at each time step.

This controller receives information about the game state, including factors like the game status (winner of the game, score, time step), the player's state (position, resources gathered), and position of the different sprites (identified only by an integer id for its type) in the level. The dynamics of these sprites and the victory conditions are never given to the player. It is the agents responsibility to discover the game mechanics while playing. However, the agent is provided with a forward model to reason about the environment, a tool that allows the agent to simulate actions and roll the game forward to one of the possible next states of the game. The forward model is very fast and almost all successful agents simulate hundreds or thousands of game states for each decision taken. For more information about the interface and constituents of the framework, the reader is referred to [5].

### B. The Games

Four games (out of the 60 distributed with the framework) have been used in this study: *Aliens*, *Butterflies*, *Sheriff* and *Sequest*. These games have been chosen according to the following characteristics:

- High percentage of victories: Not even the best controllers submitted to the competition (by rankings, the ones used in this study) are able to achieve victories in all games distributed with the GVGA framework. Three of the games selected average a percentage of victories above 90%, with only *Sequest* averaging around 45%.
- Smooth scoring: All games provide small increments of score through their play (rather than having no score change but a point given or taken when the game is won or lost, respectively). Games that provide a different score landscape are left for future work.
- Different set of actions: Not all games in GVGA provide the same set of available actions. By choosing games with different sets, the experiments will permit an analysis on how this factor affects results after applying the different game modifications.
- They are all stochastic in nature.

These four games are described next:

- **Aliens:** Similar to the classic *Space Invaders*, this game features the player (avatar) moving along the bottom of the screen, shooting upwards at aliens, who fire back at the avatar. The avatar can use the actions *Left*, *Right* and *Use* (to shoot). The player loses if touched by an alien or its bullet, and wins if all aliens are destroyed. 1 point is awarded for each alien or protective structure destroyed by the avatar and 1 point is subtracted if the player is hit.
- **Butterflies:** The avatar must capture butterflies that move randomly. If a butterfly touches a cocoon, more butterflies are spawned. The player wins if it collects all butterflies, but loses if all cocoons are opened. 2 points are awarded for each butterfly captured. The avatar can use the actions *Left*, *Right*, *Up* and *Down*.
- **Sheriff:** The avatar is at the center of the screen and the objective is to kill all the bandits that move in circles along the level, shooting at the player. There are also some structures in the level that can be used as cover. 1 point is awarded for each bandit killed, and 1 point is subtracted if the avatar dies. The avatar can move in the four directions and shoot.
- **Sequest:** Remake of the Atari game by the same name. The player controls a submarine that must avoid animals whilst rescuing divers by taking them to the surface. Also, the submarine must return to the surface regularly to collect more oxygen, or the player loses. The submarine's capacity is 4 divers, and it can shoot torpedoes at the animals. 1 point is awarded for killing an animal with a torpedo, and 1000 points for saving 4 divers in a single trip to the surface. As in *Sheriff*, the avatar can move in the four directions and shoot.

### C. The Rankings

The GVGA Competition rankings system, which is also used in this paper, aims to reward those controllers that perform well across different games, rather than relying on differences of performance in particular games.

For each one of the games used, all controllers are sorted according to three criteria, in the following order of importance: percentage of victories, average of score achieved and time spent on the victories (the lower, the better). Then, controllers are awarded with points according to this game ranking, following the Formula 1 scoring system: {25, 18, 15, 12, 10, 8, 6, 4, 2, 1}, where 25 points are awarded to the best controller, 1 to the tenth, and no points beyond that rank. In order to determine the overall best, all points per game are added up and the controller with the highest sum is declared the winner. In case of a draw in points, the number of first positions in a game unties the ranking, proceeding to the highest number of second, third, etc. positions until the tie is broken.

## III. CONTROLLERS

This section describes the different controllers that have been used in this study. The first two, Sample Open

Loop Monte Carlo Tree Search (Sample OLMCTS, Section III-A) and Rolling Horizon Genetic Algorithm (RHGA, Section III-B), are sample controllers distributed with the framework. The third controller, Open Loop Expectimax Tree Search (OLETS, Section III-C), was the winner of the 2014 GVGAI competition. Finally, the last three controllers<sup>1</sup> were the winners of the three legs of the 2015 GVGAI Competition: YOLOBOT (GECCO 2015, Section III-D), Return42 (CIG 2015, Section III-E) and YBCRIBER (CEEC 2015, Section III-F).

#### A. Sample OLMCTS

Monte Carlo Tree Search (MCTS) [11] is a very popular tree search technique that iteratively builds an asymmetric tree in memory to estimate the value of the different actions available from a given state. Starting from the current state, the algorithm repeats the following steps in iteration until the time budget is over:

First, a *Tree Selection* process selects actions until reaching a state from which not all possible moves have been taken. These actions are selected according to a *Tree Policy*, like for instance the Upper Confidence Bounds (UCB1; see Equation 1 [12]), which balances between exploitation of the best actions found so far and exploration of the ones employed less often.

$$a^* = \arg \max_{a \in A(s)} \left\{ Q(s, a) + C \sqrt{\frac{\ln N(s)}{N(s, a)}} \right\} \quad (1)$$

where  $N(s)$  represents the number of times the state  $s$  is visited,  $N(s, a)$  is the number of times an action  $a$  is taken from  $s$ , and  $Q(s, a)$  indicates the empirical average of the rewards obtained when picking an action  $a$  from  $s$ . The exploration-exploitation balance can be tempered by the value of  $C$ : setting high values gives priority to exploration, while values closer to 0 reward those actions  $a \in A(s)$  with a higher expected reward.

The second step, *Expansion*, adds a new child to the node reached at the end of the previous one. Next, a *Monte Carlo Simulation* is performed from the new node until reaching the end game or a predetermined depth. This simulation picks actions on each state according to some *Default Policy*, which could select moves uniformly at random or biased by an heuristic based on the features of the state. Lastly, the *Back-propagation* phase uses the reward observed in the state reached at the end of the *Monte Carlo Simulation*, to update the  $Q(s, a)$  values of all nodes visited during the *Tree Selection* step.

The distinction between Open Loop and Closed Loop MCTS resides in using the forward model during the *Tree Selection* phase or not. In closed loop MCTS, the algorithm assumes that is stable to store game states on the nodes of the tree when *Expansion* is performed, and therefore the *Tree*

*Selection* step can simply navigate the tree without the need of calculating the new states. If randomness is encountered, instead of acting according to the tree policy, a random guess is made as to what state one might land after an action. This is a valid approach for all games, and indeed the only really “correct” one but it may lead to sub-optimal performance on stochastic scenarios (as the games used in this research work), where one might focus too much on exploring all future possible states, never having enough time to collect enough information to perform well. Another approach is to behave in an open-loop manner - Open Loop MCTS (OLMCTS) only stores the statistics on the tree nodes, and generates the next state using the forward model to average over the distribution of possible next states. Note that for deterministic settings open-loop and close-loop are the same. For more details about this distinction, the reader is encouraged to read [13].

For the experiments performed in this experiment, the number of moves performed on each iteration is set to 10, and  $C = \sqrt{2}$ .

#### B. RHGA

Rolling Horizon Genetic Algorithm (RHGA) employs a fast evolutionary algorithm to evolve a sequence of actions to be executed from the current game state. It is an open loop implementation of a minimalistic steady state genetic algorithm, known as a microbial GA [14].

Each individual receives a fitness equal to the reward observed in the state reached at the end of the action sequence, which has a length of 7. Two different individuals are selected and evaluated from a population, and the one that obtains the worse fitness is mutated randomly, with probability 1/7, whereas certain parts of its genome are recombined with parts from the other’s genome with probability 0.1.

Both OLMCTS and RHGA use the same function to evaluate a state. The procedure works as follows: the reward is the score of the game at that state plus a high positive (respectively, negative) number if the game is finished with a victory (resp. loss).

#### C. OLETS

Open Loop Expectimax Tree Search (OLETS), created by Adrien Couëtoux, is an algorithm inspired by Hierarchical Open-Loop Optimistic Planning (HOLOP, [15]). As OLMCTS, OLETS does not store the states in memory, but uses the sampled sequences to build a tree.

A first difference with OLMCTS is that OLETS does not use any roll-out and relies on the game scoring function to give a value to the leaves of the tree. Additionally, another important difference is that the empirical average of rewards obtained by performing simulations is not used in the UCB1 policy (see Equation 1). Instead, OLETS replaces  $Q(s, a)$  with the *Open Loop Expectimax* (OLE) value ( $r_M(n)$ ), as calculated in Equation 3).

$$r_M(n) = \frac{R_e(n)}{n_s(n)} + \frac{(1 - n_e(n))}{n_s(n)} \max_{c \in C(n)} r_M(c) \quad (2)$$

<sup>1</sup>To the knowledge of the authors of this paper, the descriptions of these controllers have not been published to date. All these controllers are accessible for download at the competition website, [www.gvgai.net](http://www.gvgai.net)



where  $n_s(n)$  the number of simulations that visited the node  $n$ ,  $n_e(n)$  the amount of them that end in  $n$ , and  $R_e(n)$  the accumulated reward from this last subset.  $C(n)$  the set of children of  $n$ , and  $P(n)$  the parent of  $n$ . For more details about this algorithm, please consult [5].

#### D. YOLOBOT

This controller, created by Tobias Joppen, Nils Schroeder and Miriam Moneke, was declared winner of the 2015 GVGAI championship, as they obtained the highest sum of scores across the three legs run that year. Their approach uses pathfinding to first identify those sprites that can be reached from the avatar's position, creating a list with the nearest reachable sprite of each type. At the same time, it also tries to identify if the game is deterministic or not, using the forward model to spot differences on states reached after applying the same action from a given state. This is done to choose which algorithm to use to try to discover how valuable these sprites are within the game. If the game is deterministic, YOLOBOT uses Best First Search (BFS) to navigate to the target sprite. In case the game is deemed as stochastic, the algorithm of choice is an open loop version of MCTS, in order to get closer to the aimed sprite without losing the game due to stochasticity.

#### E. Return42

This controller, created by Tobias Welther, Oliver Welther, Frederik Buss-Joraschek and Stefan Hbecker is a hyper-heuristic that combines different algorithms which are used depending on the type and state of the game. Initially, the games are differentiated by being deterministic or not, a feature checked using the forward model to determine if multiple states derived from the same original state are the same. If the game is deterministic, an A-Star algorithm is used to determine future states with high scores and possibly winning conditions. In case the game is stochastic, random walks are used to determine the best action based on a hand-crafted heuristic that considers score and changes on resources and NPCs in the game.

#### F. YBCRIBER

This controller was submitted by Ivan Geffner, Tomas Geffner and Felix Mirave. The algorithm is based on Iterative Width (IW [16]) with a dynamic look-ahead scheme. A previous version of this work can be found at [17]. YBCRIBER employs some basic statistical learning to save information about each sprite at each look ahead, which it then uses to select actions in stochastic games and to prune actions over the IW search. Additionally, a danger prevention mechanism minimizes the chances of the avatar being killed in close proximity of hazards.

### IV. DEFAULT RANKINGS

All controllers described in Section III have been executed 100 times in each one of the 5 levels of the 4 games detailed in Section II-B. Therefore, each controller plays 500 times on each game. The percentage of victories, average of scores and

time spent are recorded, and non-parametric Wilcoxon Signed Rank tests are computed to determine statistical significance ( $p$ -value  $< 0.05$ ). All experiments performed in this research have been carried out in this manner, for the default settings and for each one of the different environment configurations described in Section V.

Table I shows the results for the tested controllers in the games selected. The percentage of victories, average of scores and time steps used to complete the game are shown here with their respective standard error measures. First of all, it is worthwhile mentioning that there is no superior algorithm that achieves the best results in all games tested. Both in *Aliens* and *Butterflies*, three controllers achieve 100% of victories, the first metric in order of importance. Note that these controllers are not the same in both games. *Sheriff* is revealed to be a slightly more complicated game, as no controller achieves the maximum amount of victories. It seems, however, to be easier than *Sequest*, where the best controller obtained less than 70% of victories.

The variability of these games can also be observed in two factors: First, winners of some games can perform badly in others (i.e., YOLOBOT is the leader in *Aliens*, while achieving 0.20% of victories in *Sequest*; or like Return42, which is the best controller in *Sequest* but the worst one in *Sheriff*). Secondly, there is a high variance in the scores typically achieved on each game, as Table I shows.

Table II shows the rankings derived from these results. The controller that ranks first in this set of games is OLETS, closely followed by YBCRIBER. It is interesting to see how YBCRIBER ranks high albeit it does not perform the best in any game. This is due to its high general performance (ranking 2<sup>nd</sup> or 3<sup>rd</sup> in all games), a consequence derived from this ranking system, which rewards controllers that perform well across different games.

### V. EXPERIMENTS

This section describes the experiments performed for this paper. Each section details the changes and results obtained for each one of the different configurations tested.

#### A. Reward Penalization

In this setting, the GVGAI framework is modified so that every time an agent performs any action, the score in the game is reduced by 1 point. In principle, one could assume that controllers that are able to perform well using the minimum possible amount of moves would be rewarded in the rankings. These rankings are shown in Table III<sup>2</sup>.

All controllers seem to resist quite well the penalizations set to the actions performed, with the exception of Return42. This controller is specially affected by this change, as it is the one with the highest drop in percentage of victories (from 81.55% to 71.10%). The first and the second controller alternate positions compared to the original rankings (where OLETS was 1<sup>st</sup> and YBCRIBER was 2<sup>nd</sup>).

<sup>2</sup>To save space, no tables are reported for individual games and scores achieved, albeit some of those results are discussed.

TABLE I

PERCENTAGE OF VICTORIES AND AVERAGE OF SCORE ACHIEVED (PLUS STANDARD ERROR) IN 4 DIFFERENT GAMES. FOURTH, SIXTH AND EIGHTH COLUMNS INDICATE THE APPROACHES THAT ARE SIGNIFICANTLY WORSE THAN THAT OF THE ROW, USING THE NON-PARAMETRIC WILCOXON SIGNED-RANK TEST WITH P-VALUE < 0.05. BOLD FONT FOR THE ALGORITHM THAT IS SIGNIFICANTLY BETTER THAN ALL THE OTHER 5 IN EITHER VICTORIES OR SCORE.

Game	Algorithm	Victories (%)	Significantly better than ...	Scores	Significantly better than ...	Timesteps	Significantly better than ...
Aliens	A: YOLOBOT	100.00 (0.00)	B, D, E	70.56 (0.59)	B, C, D, E, F	434.55 (1.52)	B, C, D, E, F
	B: OLETS	98.80 (0.49)	∅	66.91 (0.64)	E, F	790.96 (3.65)	∅
	C: YBCRIBER	100.00 (0.00)	B, D, E	69.56 (0.60)	B, E, F	470.41 (1.82)	B, D, E, F
	D: Return42	97.40 (0.71)	∅	68.43 (0.68)	B, E, F	513.13 (8.50)	B, E, F
	E: OLMCTS	99.20 (0.40)	D	61.16 (0.52)	∅	613.02 (4.12)	B
	F: RHGA	100.00 (0.00)	B, D, E	64.99 (0.60)	E	599.38 (3.02)	B, E
Butterflies	A: YOLOBOT	95.60 (0.92)	E	27.80 (0.63)	C, D	528.03 (24.26)	E
	B: OLETS	100.00 (0.00)	A, E, F	26.13 (0.53)	C, D	170.53 (5.74)	A, E, F
	C: YBCRIBER	100.00 (0.00)	A, E, F	23.19 (0.43)	∅	59.45 (0.88)	A, B, D, E, F
	D: Return42	100.00 (0.00)	A, E, F	24.59 (0.48)	C	73.68 (1.38)	A, B, E, F
	E: OLMCTS	86.60 (1.52)	∅	31.44 (0.69)	A, B, C, D	728.91 (21.82)	∅
	F: RHGA	94.40 (1.03)	E	32.89 (0.75)	A, B, C, D	447.54 (17.48)	A, E
Sheriff	A: YOLOBOT	95.00 (0.97)	D	8.52 (0.09)	C, D, E, F	826.31 (13.63)	C, E
	B: OLETS	97.20 (0.74)	A, D, F	9.18 (0.07)	A, C, D, E, F	679.67 (11.08)	A, C, D, E, F
	C: YBCRIBER	96.80 (0.79)	D, F	6.09 (0.08)	∅	1006.00 (4.97)	D
	D: Return42	59.00 (2.20)	∅	6.49 (0.17)	C	1018.96 (28.82)	∅
	E: OLMCTS	97.40 (0.71)	A, D, F	6.56 (0.08)	C, D	1001.62 (4.69)	D
	F: RHGA	93.40 (1.11)	D	8.04 (0.08)	C, D, E	808.00 (13.47)	A, C, E
Seaquest	A: YOLOBOT	0.20 (0.20)	∅	97.54 (12.69)	∅	1572.78 (2.29)	∅
	B: OLETS	60.00 (2.19)	A, E, F	1309.77 (74.33)	A, C, E, F	1266.04 (16.90)	A, E, F
	C: YBCRIBER	60.60 (2.19)	A, E, F	452.48 (28.84)	A	1180.91 (11.55)	A, B, E, F
	D: Return42	69.80 (2.05)	A, B, C, E, F	2858.29 (123.42)	A, B, C, E, F	1170.54 (13.14)	A, B, C, E, F
	E: OLMCTS	46.60 (2.23)	A, F	508.48 (38.61)	A	1266.75 (12.99)	A, F
	F: RHGA	24.60 (1.93)	A	301.21 (27.40)	A	1384.76 (12.76)	A

TABLE II

RANKINGS TABLE FOR THE COMPARED ALGORITHMS ACROSS ALL GAMES. IN THIS ORDER, THE TABLE SHOWS THE RANK OF THE ALGORITHMS, THEIR NAME, TOTAL POINTS, AVERAGE OF VICTORIES AND POINTS ACHIEVED PER GAME, FOLLOWING THE F1 SCORING SYSTEM.

#	Algorithm	Points	Avg. Wins	G-0	G-1	G-2	G-3
1	OLETS	68	89.00	10	25	18	15
2	YBCRIBER	66	89.35	18	15	15	18
3	Return42	59	81.55	8	18	8	25
4	YOLOBOT	57	72.70	25	12	12	8
4	OLMCTS	57	82.45	12	8	25	12
5	RHGA	45	78.10	15	10	10	10

TABLE III

RANKINGS TABLE IN THE *Reward Penalization* SETTING.

#	Algorithm	Points	Avg. Wins	G-0	G-1	G-2	G-3
1	YBCRIBER	80	94.55	18	25	12	25
2	OLETS	61	89.45	10	18	18	15
3	YOLOBOT	58	72.95	25	10	15	8
4	OLMCTS	57	82.95	12	8	25	12
5	Return42	49	71.10	8	15	8	18
6	RHGA	47	80.15	15	12	10	10

Penalizations affect controllers differently, in different degrees, but the changes in performance are not extremely large in this setting. Regarding scores obtained, all controllers obtain now negative scores, but the cross comparison among them shows stability in the results, without major changes in performance in this metric.

TABLE IV

RANKINGS TABLE IN THE *Discounted Reward* SETTING.

#	Algorithm	Points	Avg. Wins	G-0	G-1	G-2	G-3
1	OLETS	78	80.70	10	25	18	25
2	OLMCTS	73	87.15	15	15	25	18
3	RHGA	58	81.15	18	18	12	10
4	YOLOBOT	56	63.90	25	8	15	8
5	Return42	44	56.45	12	12	8	12
6	YBCRIBER	43	55.95	8	10	10	15

### B. Discounted Reward

In this setting, the score returned by the forward model for a given state  $s$  is discounted depending on the depth of search ( $d$ ), according to the following scheme:

$$r_{disc}(s) = r_{raw}(s) \times D^d \quad (3)$$

where  $D$  is the discount factor, set to 0.9 to produce a significant (but not too damaging) effect on the controllers. The question that this modification poses is to verify how robust the controllers are to delayed rewards that are discounted in the future. The rankings for this configuration are shown in Table IV

This setting affects the controllers more than the previous one, although the first ranked controller is still the same (OLETS). In this configuration, YBCRIBER is the agent that suffers the most significant drop on the averages of victories, going from 89.35% to 55.95%.

TABLE V  
RANKINGS TABLE IN THE *Noisy World* SETTING.

#	Algorithm	Points	Avg. Wins	G-0	G-1	G-2	G-3
1	OLMCTS	83	74.75	25	8	25	25
2	Return42	63	50.70	10	25	10	18
3	YBCRIBER	53	49.00	15	18	8	12
4	RHGA	52	53.70	12	12	18	10
5	YOLOBOT	51	53.70	18	10	15	8
6	OLETS	50	49.75	8	15	12	15

It is interesting to note that reward discounting has such a surprisingly disruptive effect on the rankings. This modification affects the distinct agents tested in this study in different ways, and suggests as an open question whether it would be possible to identify concrete changes that would benefit particular controllers. In this scenario, the biggest impact happens in the game *Seaquest*, where the performances of YBCRIBER and Return42 plummet (the latter in percentage of victories, the former in both victories and score), OLMCTS increases slightly, and OLETS remains the same, enough to keep the first position in this game (and consequently, in the overall ranking).

Another interesting observation is that both sample controllers (OLMCTS and RHGA) are resilient to this setting, which allows them to climb to the 2<sup>nd</sup> and 3<sup>rd</sup> positions of the rankings, respectively.

#### C. Noisy World

In this modification, noise is added to the actions executed by the controller. Concretely, with a probability  $p$ , a different random action is chosen to be performed instead of the one intended by the controller.  $p$  was set to a high value, 0.25, in order to achieve a big impact in the controllers employed in this study. This noise is introduced both in the real game and in the forward model. The rankings obtained with this setting are shown in Table V.

This modification in the game engine and forward model produces a very important change in the rankings. The most significant is that a new controller gains the first position in the rankings: OLMCTS, with a relevant difference of points and percentage of victories with the second (Return42, 19 and 24.05%). Actually, it becomes the best controller in three out of the four games tested. On the other hand, OLETS, the best controller in the default setting, drops to the last position.

In general, all agents observe an important drop on the average of victories achieved (between 20% and 40%), with the exception of OLMCTS that, resilient to this modification, only suffers a drop of 7.7%. The differences on scores achieved are not large, with the exception of *Seaquest*, where all controllers achieved significantly lower scores. The loss in *Sheriff* and *Aliens* is smaller, and *Butterflies* experiences a slight increase. This change in *Butterflies* could be explained by the nature of the game (see Section II-B): higher scores are achievable only when less cocoons remain closed, but the game is lost when all cocoons open.

TABLE VI  
RANKINGS TABLE IN THE *Broken World* SETTING.

#	Algorithm	Points	Avg. Wins	G-0	G-1	G-2	G-3
1	OLMCTS	85	68.50	25	10	25	25
2	YBCRIBER	58	46.50	15	15	10	18
3	YOLOBOT	56	49.75	18	8	18	12
4	OLETS	55	40.30	10	18	12	15
5	Return42	49	34.00	8	25	8	8
6	RHGA	49	42.15	12	12	15	10

TABLE VII  
RANKINGS TABLE IN THE *Broken Forward Model* SETTING.

#	Algorithm	Points	Avg. Wins	G-0	G-1	G-2	G-3
1	OLMCTS	71	83.05	18	10	25	18
2	OLETS	63	63.80	10	18	10	25
3	YBCRIBER	60	77.15	15	12	18	15
4	YOLOBOT	58	64.05	25	8	15	10
5	RHGA	51	57.00	12	15	12	12
6	Return42	49	38.45	8	25	8	8

#### D. Broken World

In this setting, the same configuration as in the previous case was used, but in this case only the real game can introduce noise in the actions supplied, while the forward model is always accurate. Again,  $p = 0.25$  and the rankings are detailed in Table VI. The idea of this modification is to test how the agents can cope with a forward model that does not reproduce noise in the real game.

The new results obtained with this modification are similar to those achieved in the previous case. OLMCTS becomes the highest ranked entry achieving the best result in the same three games as shown in Section V-C, and drop in victory percentage happens across all controllers.

Note that the drop in percentage of victories is higher than in the previous scenario, where even OLMCTS loses 20.5 percentage points. This could be explained by the fact that inaccuracies are now introduced due to the noise included in the actions executed in the real game, but not in the forward model. However, it is interesting to note that again one of the sample (hence, simplest with regards to the value function) controllers suffers this effect the least.

Finally, regarding the games in particular, *Butterflies* still remains as the game where percentage of victories change the less (hence also being the game where OLMCTS does not rank the first).

#### E. Broken Forward Model

Finally, this setting proposes the complementary scenario to the one shown in the previous section. Noise with  $p = 0.25$  is introduced only in the forward model, while the actions supplied to the game are never altered. The rankings for this configuration are shown in Table VII.

In this final setting, OLMCTS achieves again the highest position in the rankings. It is worth noting, however, that in this case the difference with the second ranked entry (OLETS) is only of 8 ranking points. Additionally, it only achieves the

first position in one of the four games, and all controllers suffer a smaller loss in the percentage of victories than in the previous case.

An interesting observation that can be drawn from this results is that, when noise in the actions is only present in the real game instead of in the forward model, the agents have more difficulties to deal with this hazard. In other words, the algorithms tested are more robust to noise present in the forward model (when no noise is present in the real game) than vice-versa.

#### F. Overall Comparison

Figure 1 depicts the average of victories of all controllers in the four games tested, for the different six configurations experimented with in this research. This graphic summarizes well the findings of this study. It is clear that the latter modifications (adding noise in different parts of the framework) affect the controllers more than the first two changes in most of the games (*Butterflies* remains as an exception to this statement, where the loss in average of victories is smaller).

Concretely, it can be observed how the *Broken World* configuration produces the higher variance in the results: a forward model that is not able to simulate the noise on the actions that is present in the real game is not good enough for most controllers. However, this change does not equally affect all agents. The ones that use simpler value functions (with less domain knowledge, like *OLMCTS*) respond better to a noisy world without a noisy forward model.

It is also worth mentioning that a forward model that simulates noise, even at the high rate of executing a random action with  $p = 0.25$ , can cope with both a noisy and a non-noisy real game environment. In fact, in some occasions, results obtained in the *Broken Model* configuration are better than the ones from a *Noisy World*, which suggests that these techniques (especially *OLMCTS*) are robust to a noisy forward model even if the game itself is not noisy.

### VI. CONCLUSIONS AND FUTURE WORK

This paper described a study on the robustness of several good general video game AI controllers (concretely, the winners of the four legs of the previous competitions and some sample controllers from the GVGAI framework) when the conditions of the rewards and/or actions are changed in the environment. In this research, alterations in the rewards (introducing penalties for using certain actions, or discounting the game score) and in the action performed (either by including noise in the real game, or in the forward model, or both) are introduced to analyze how the rankings change. A key finding of the research is that some of these changes can dramatically alter the rankings of the agents, which provides a simple way to effectively expand the set of GVGAI games.

An interesting outcome of this study is that simpler controllers, those that utilize a state value function that only focuses on score and winning conditions, achieve better results when noise is introduced on the actions. The effect on the ranking differs significantly depending on the game and the agent

and the nature of the modifications. For instance, controllers that included an element of best-first search (*Return42* and *YOLOBOT*) seem to handle unexpected noise badly. This is consistent with earlier results where MCTS is able to handle the introduction of noise much better than A\* [8].

Furthermore, not all simple controllers perform well under noisy circumstances: *RHGA* is not able to climb in the rankings as much as *OLMCTS*, which becomes the 1<sup>st</sup> ranked entry in some scenarios or *OLETS*, which is able to keep the second position in these settings. Furthermore, results show that, in the noisy settings, a noisy forward model with a non-noisy real game makes the controllers behave better than introducing noise in the real game (either alone, or together with noise in the forward model). The latter condition (noisy model, deterministic world) is likely to most closely model non-game situations such as robot control.

The results shown in this paper leave us with multiple open questions for future investigation. A straightforward one could be to explore the parameter space (like the values of the noise probability  $p$  or the discount factor  $D$ ) to find out at which point they actually trigger the modifications observed in this paper. In other words, it is possible to analyze the continuum of values of  $p$  to identify at which point the amount of noise introduces a change in the rankings. It would also be possible to introduce other types of noise (like variations in the states observed) to analyze how does that modify the rankings, and study the effect of this in more games (especially in those omitted by the decisions explained in Section II-B).

For instance, given that the performance of the agents does also depend on the game used, a possible question to ask is if it is possible to identify or classify games with respect to what changes can make controllers go up or down in the rankings. For instance, what features make certain games be more indifferent to penalizations in the moves made? Could we infer some game design lessons from these categorizations?

As different controllers react differently to the changes made, it is worth investigating if it is possible to automatically find the parameters that will make some controllers behave better than others. In other words, could we find, maybe by evolution, the values of certain parameters that would permit us to have any ranking desired using a specific set of games? This would parallel previous work on evolving game maps to induce differential rankings between agents [18].

This research also proposes a new way of evaluating controllers: the same agents in a set of games can perform differently depending on the setting used. Therefore, it is at least thought provoking to consider if the best controller in a competition should be the one that resists such changes in the environment best.

Finally, it could be argued that we are not only testing the robustness of the controllers, but also the robustness of the competition itself, and thus its value as a benchmark. If the rankings of controllers only depended on the amount and type of noise, this would mean the benchmark would be rather brittle. However, as observed above, some controllers do better than others under all or almost all conditions. For

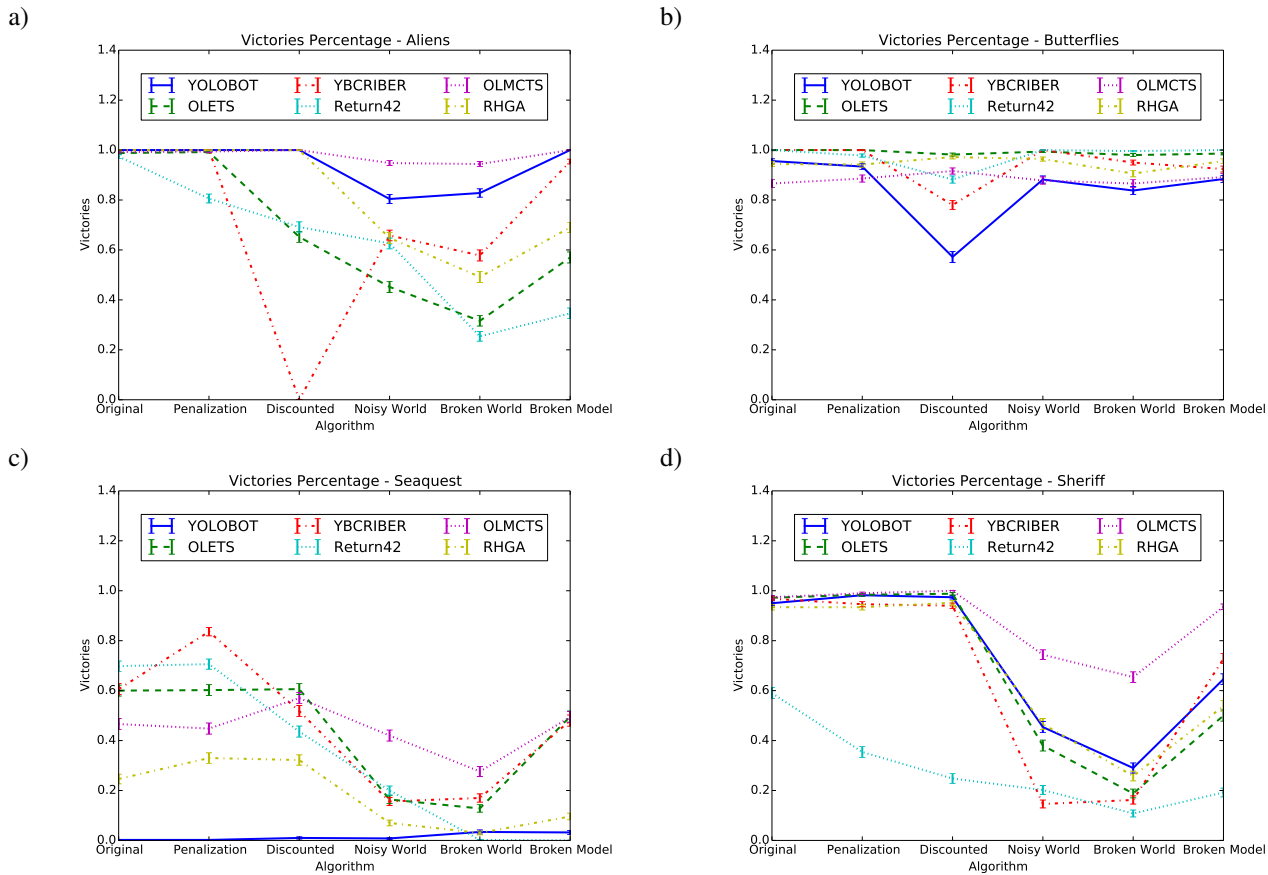


Fig. 1. Victory percentages per configuration and game.

example, OLMCTS always performs better than RHGA. It therefore seems that the underlying challenge of the GVGAI competition is fairly robust to perturbations.

#### ACKNOWLEDGMENTS

The authors would like to thank the participants of the several GVGAI competition rounds for their submissions.

#### REFERENCES

- [1] J. Togelius, "How to Run a Successful Game-Based AI Competition," *IEEE Trans. Comput. Intellig. and AI in Games*, vol. 8, no. 1, pp. 95–100, 2016.
- [2] T. Schaul, "A Video Game Description Language for Model-based or Interactive Learning," in *Proceedings of the IEEE Conference on Computational Intelligence in Games*, 2013.
- [3] M. Ebner, J. Levine, S. M. Lucas, T. Schaul, T. Thompson, and J. Togelius, "Towards a Video Game Description Language," *Dagstuhl Follow-Ups*, vol. 6, 2013.
- [4] J. Levine, C. B. Congdon, M. Ebner, G. Kendall, S. M. Lucas, R. Mikkilainen, T. Schaul, and T. Thompson, "General Video Game Playing," *Dagstuhl Follow-Ups*, vol. 6, 2013.
- [5] D. Perez-Liebana, J. Togelius, S. Samothrakis, T. Schaul, S. M. Lucas, A. Couetoux, J. Lee, C.-U. Lim, and T. Thompson, "The 2014 General Video Game Playing Competition," *IEEE Transactions on Computational Intelligence and AI in Games*, 2015.
- [6] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, and S. M. Lucas, "General Video Game AI: Competition, Challenges and Opportunities," in *AAAI*, 2016.
- [7] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The Arcade Learning Environment: An Evaluation Platform for General Agents," *Journal of Artificial Intelligence Research*, 2012.
- [8] E. J. Jacobsen, R. Greve, and J. Togelius, "Monte Mario: Platforming with MCTS," in *Proceedings of the Conference on Genetic and Evolutionary Computation*, New York, NY, USA, 2014, pp. 293–300.
- [9] A. Isaksen, D. Gopstein, and A. Nealen, "Exploring Game Space Using Survival Analysis," in *Foundations of Digital Games*, 2015.
- [10] A. Isaksen, D. Gopstein, J. Togelius, and A. Nealen, "Discovering Unique Game Variants," in *ICCC Games Workshop*, 2015.
- [11] C. Browne, E. J. Powley, D. Whitehouse, S. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *IEEE Trans. on Computational Intelligence and AI in Games*, vol. 4:1, pp. 1–43, 2012.
- [12] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo Planning," in *In: ECML-06. Number 4212 in LNCS*. Springer, 2006, pp. 282–293.
- [13] D. Perez-Liebana, J. Dieskau, M. Hunermund, S. Mostaghim, and S. M. Lucas, "Open Loop Search for General Video Game Playing," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2015, pp. 337–344.
- [14] I. Harvey, "The Microbial Genetic Algorithm," in *Advances in artificial life. Darwin Meets von Neumann*. Springer, 2011, pp. 126–133.
- [15] A. Weinstein and M. L. Littman, "Bandit-Based Planning and Learning in Continuous-Action Markov Decision Processes," in *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS, Brazil*, 2012.
- [16] N. Lipovetzky and H. Geffner, "Width and Serialization of Classical Planning Problems," in *Proceedings of the European Conference on Artificial Intelligence*, 2012, pp. 281–285.
- [17] T. Geffner and H. Geffner, "Width-based Planning for General Video-Game Playing," in *Proceedings of the IJCAI Workshop on General Intelligence in Game Playing Agents (GIGA)*, 2015.
- [18] D. Perez, J. Togelius, S. Samothrakis, P. Rohlfshagen, and S. M. Lucas, "Automated Map Generation for the Physical Traveling Salesman Problem," *Evolutionary Computation, IEEE Transactions on*, vol. 18, no. 5, pp. 708–720, 2014.

# Predicting Player Churn in Destiny: A Hidden Markov Models Approach to Predicting Player Departure in a Major Online Game

Marco Tamassia\*, William Raffe<sup>†</sup>, Rafet Sifa<sup>‡</sup>, Anders Drachen<sup>§</sup>, Fabio Zambetta<sup>¶</sup>, Michael Hitchens<sup>||</sup>

\* <sup>†</sup> <sup>¶</sup> School of Science, RMIT, Melbourne, Australia

Email: {marco.tamassia,william.raffe,fabio.zambetta}@rmit.edu.au

<sup>‡</sup>Fraunhofer IAIS, Sankt Augustin, Germany, Email: rafet.sifa@iais.fraunhofer.de

<sup>§</sup>Aalborg University, The Pagonis Network, Copenhagen, Denmark

Email: andersdrachen@gmail.com

<sup>||</sup> Department of Computing, Macquarie University, Sydney, Australia

Email: michael.hitchens@mq.edu.au

**Abstract**—*Destiny* is, to date, the most expensive digital game ever released with a total operating budget of over half a billion US dollars. It stands as one of the main examples of AAA titles, the term used for the largest and most heavily marketed game productions in the games industry. *Destiny* is a blend of a shooter game and massively multi-player online game, and has attracted dozens of millions of players. As a persistent game title, predicting retention and churn in *Destiny* is crucial to the running operations of the game, but prediction has not been attempted for this type of game in the past. In this paper, we present a discussion of the challenge of predicting churn in *Destiny*, evaluate the area under curve (ROC) of behavioral features, and use Hidden Markov Models to develop a churn prediction model for the game.

## I. INTRODUCTION

In this paper the problem of behavioral prediction in digital games is brought into the context of the most expensive game developed and released to date, the online shooter *Destiny*. Developed and released by Bungie in September 2014, the game cost over half a billion US dollars to develop, but sold more than that on its first day of retail, making it the biggest new franchise launch of all time in the game industry. According to Bungie, the game reached a billion US dollars in revenue around May 2015 [1].

*Destiny* is not only an example of the most expensive to develop and heavily marketed titles today, also referred to as “AAA” games, but also defines a new form of online game, mixing traditional individual/team-based shooter games with elements from Massively Multi-Player Online Games (MMOGs), and includes both Player-vs-Environment (PvE) and Player-vs-Player (PvP) elements, combining competition and collaboration. In terms of gameplay, *Destiny* is a complex title with a variety of different game modes and wide player freedom in terms of navigation and how to spend time in the game, wrapped in a traditional class-based progression system with missions and instances reminiscent of MMOGs such as *World of Warcraft*.

*Destiny* as a popular hybrid game title is in its own right worthy of study, but it is also of interest in a broader context: *Destiny* is representative of a trend among major commercial game titles, where publishers are moving away from the traditional retail fire-and-forget business model, and navigating towards a hybrid model which tries to take advantage of the revenue streams offered by persistent online games in the form of, for example, Downloadable Content (DLC), micro-transactions (In-App Purchases, IAPs) and similar tools. *Destiny*’s developers have experimented with a variety of different revenue opportunities already, including DLC, the sale of emotes and cosmetic items, weapon packs and more. However, with the change in business model comes also new requirements for analytics support. In a retail business model, there is no direct need to monitor the population of the players, but in a persistent game situation, the monitoring and forecasting of player behaviour becomes important to ensuring the revenue stream and operations of a game [2]–[5].

While smaller commercial games for mobile platforms, notably Free-to-Play (F2P) games have been the subject of prediction modelling recently [3]–[5], major commercial titles have received less attention. This is possibly partly due to a lack of accessible data, and partly due to the traditional non-persistent nature of such titles. In the types of AAA titles that are based on persistent game design, the situation is different however. For MMOGs, monitoring of the player community and prediction of their behaviour have been topics of considerable interest, notably from network balancing perspectives as these games have to balance a large population of players across multiple servers [6]–[8]. Similarly, within the domains of eSports – when computer games are played competitively – analytics support has received substantial interest, with behavioral analysis playing a similar role as in physical sports analytics [9].

The situation in *Destiny* compares with all of these related domains of inquiry but along different trajectories. Similar to F2P games, *Destiny* adopts micro-transactions as a source of



revenue and feature numerous in-game currencies and reward vectors (equipment, reputation, appearance, etc.). Similar to MMOGs, the game is highly persistent in nature, in essence you are never finished with the game, and there is a running stream of updates and new content being released. The game also supports a huge population of players operating within the same virtual environment. Finally, similar to eSports games, there is a substantial competitive and team-based play element in *Destiny*, exemplified by the *Crucible*, which is the framework for PvP play in the game. In essence, while *Destiny* as a title has not seen previous attention from game analytics, there is some related – but also very recent – work available which can form the basis for investigating churn prediction in the game.

In this paper the focus is on exploring the potential for predicting player churn in AAA titles like *Destiny*, with the game being used as the test case. Given the recent shifts in the revenue generation strategies in AAA games, the game with its hybrid design and large user base forms an ideal platform for investigating behavioral prediction. This is augmented by the availability of high-dimensional, time-series datasets about player behavior thanks to the telemetry tracking of Bungie. Access to data collected since the beginning of the history of the game is available through an API that is already used by the player community to inform the players, for example through services such as *destinytracker.com*. A similar pattern is observable for eSports games, where the importance of feeding behavioral data back to the community is essential to drive engagement in e.g. tournaments [9].

## II. CONTRIBUTION

In this paper time-series behavioral data from 10,000 randomly selected *Destiny* players are used as the basis for investigating churn in *Destiny*. It is the first time this kind of hybrid online game has formed the basis for behavioral analysis.

The contribution of this paper is threefold: a) we present the first behavioral analysis of *Destiny*, the to date most expensive AAA-level commercial title produced in the world; b) We present a churn prediction model for the game based on Hidden Markov Models (HMMs), chosen due to the time series nature of the data, and further due to their successful application in F2P mobile game contexts [3]–[5], [10], [11]. HMM results are benchmarked against other ML models; c) We present a thorough discussion of the kinds of behavioral features typically tracked of player performance in online AAA titles, and their relative usefulness in connection with churn prediction.

## III. RELATED WORK

Behavioral prediction in games is a relatively recent topic, but has a strong tradition outside games. One of the earliest successful churn models was presented by Mozer et al. [12], in the area of wireless communication, and churn has been investigated in e.g. in retail banking and insurance.

Churn prediction work in digital games has primarily taken place across four vectors: a) F2P mobile games [3], [4]; b) MMOGs [2], [8]; c) other games [13]–[15] and d) Game AI in general [16]. The latter approach is the least directly applicable to the current problem as the focus here – generally – is on artificial agents and mimicking the behavior of players, as compared to analyzing the behavior of the players. Due to space constraints the focus in this section will be on work directly related to the challenge of behavioral prediction in online persistent games.

### A. F2P mobile games

Recent work on prediction in F2P mobile games has covered a variety of machine learning models, and is focused on either predicting players leaving the game [4], [5], or conversely which players that will make a purchase in the game [3], [10], [11]. The churn problem in games was formally defined by Hadiji et al. [5], who also identified a number of behavioral features which are applicable across F2P game titles, including several temporally-bound features such as Number of Sessions, Avg. Time Between Sessions, Total Days Played, Current Absence Time, and Average Playtime per Session. The features identified by Hadiji et al. [5] were later used by Sifa et al. [3] and others, and several similar features occur in F2P prediction work such as Xie et al. [11], Rothenbuehler et al. [10] and others who focused on predicting IAPs. In general, evaluating the usefulness of these features in predicting churn across five F2P titles, the work in F2P games has highlighted the importance of behavioral features associated with playtime as a function of real-world time, e.g. the Number of Sessions, Number of Days Played and Avg. Playtime per Session were found to be the most important features. Interestingly, the duration of the time between play sessions have also been found to be important to churn prediction in MMOGs, see e.g. [8].

The methods applied range from pattern recognition and historical analysis, simple forecasting and multiple regression, to machine learning techniques. The latter notably includes Decision Trees [DTs] and variants such as Random Forest [3], [5], Support Vector Machines [11] and Hidden Markov Models [HMMs] [4], [10].

As yet deep learning methods have not been applied in behavioral prediction in games but forms a potentially interesting addition to the current arsenal of game analysts due to the ability of deep learning methods to handle sparse and imbalanced data, which are typical in behavioral telemetry situations [2], [3], [5]. There is as yet not enough publicly available knowledge to draw conclusions about which behavioral features provide the best result across different games, or which ML models work best for predicting player behavior, but commonly reported accuracies lie above 0.8, meaning that predicting player behavior in F2P games is definitely possible. It should be noted that these games are generally also much more restrictive in their design in terms of player agency than MMOGs and AAA-level titles such as *Destiny*. In essence, they are simpler games.

## B. MMOGs

The focus of behavioral modeling in MMOGs has from the onset had a quite different focus than work in the F2P space, namely that these lines of research originate in network science. Some of the earliest work includes Kawale and Srivastava [17] who investigated churn in the MMOG *EverQuest II* using social network analysis as the basis, proposing a churn model based on social influence among players. However, the precision and recall rates obtained were approximately 50%, which led Borbora et al. [18] to try out other classifiers, similarly using *EverQuest II* data and hybrid methods with a binary decision approach, defining churners as players who cancelled their subscription or been inactive for 2 months. Also with the focus on MMOGs, Nozhnin [19] focused on the first few minutes of gameplay in the game *Aion*, investigating triggers for churn. The author highlighted the challenge of feature selection in behavioral prediction in games.

Focusing on the two MMOGs *World of Warcraft* and *Warhammer Online*, Pittman and Gauthier [7] mined client-server streams from client servers measuring player distributions using data such as session length and high-level movements of the players, with the focus on informing MMOG server architecture. Feng et al. [8] applied traffic analysis to a three-year dataset from the MMOG *EVE Online*. The results indicated that churn rates in the game varied across the lifespan of the game, generally increasing with the age of the game. Furthermore, Thawonmas et al. [6] analyzed player revisitation in terms of returning to play the game, as well as returning to specific in-game areas, in the MMOG *Shen Zhou Online*. The primary behavioral features used were login time and login frequency.

## C. Other games

Outside the confines of MMOGs and F2P mobile games, behavioral prediction has been the topic of a few publications, across a variety of games. For example, one of the earliest investigations into behavioral prediction in AAA-level commercial games was performed by Mahlman et al. [20], who used Decision Trees to predict retention in the action-adventure game *Tomb Raider: Underworld*. Sifa et al. [21] built a tensor factorization based representation learning framework to incorporate the movement information of numerous players into the retention prediction process for the sandbox game *Just Cause 2*.

Within the genre of Multi-player Online Battle Arena (MOBA) games, Yang et al. [22] presented an approach for discovering and defining patterns in combat tactics among winning teams in the eSports title *DOTA 2*, based on graph representation. Schubert et al. [9] developed an algorithm for dividing eSports matches into encounters, which were then used as the basis for prediction models focusing on match outcome in *DOTA 2*.

Operating with data from the Steam game distribution and -hosting client, Bauckhage et al. [23] modelled players engagement to games using lifetime analysis across five major commercial titles. Modeling the players interest as a

hidden variable the authors extracted playtime information and showed how the interest can be represented in terms of lifetime distributions and their corresponding processes. This work formed the first attempt at an explanation of the power law pattern evident in many studies of playtime and retention in games to that point. The work was followed up by Sifa et al. [15] who found similar patterns across more than 3000 game titles, working with over five billion hours of play across more than six million players.

In summary, prediction has in the field of game analytics been focused on F2P games, with a deeper history in online games such as MMOGs. However, prediction is also finding uses within 3D navigation, progress prediction, or predicting what kind of problems specific players will encounter, etc. The vast majority of the current knowledge about these application areas rests within the industry, where the combined resources for behavioral research outranks academic research by at least a factor of ten, and only small glimpses of such business-sensitive knowledge is available through industry talks and presentations.

## IV. DESTINY: GAMEPLAY

*Destiny* is formally a science fiction-themed online first-person shooter game which has been blended with a number of features reminiscent of MMOGs, notably a persistent online world, as well as with Role-Playing Games (RPGs), notably character development along a number of trajectories, which also occur in many MMORPGs (Massively Multi-player Online Role-Playing Games). The game was developed by Bungie, and published by Activision in September 2014. The game is only available on major gaming consoles and requires the player to be always-online.

In terms of comparisons with earlier work on behavioral prediction in games, *Destiny* forms a unique case in that it shares design elements and mechanics that are found within the types of games this earlier work has focused on, without being similar to any previous game that has formed the basis for predictive analytics. For example, similar to F2P games, *Destiny* features microtransactions and purchasable content, but also has other features as noted above. Similar to prediction work on other types of games, *Destiny* features team-based combat reminiscent of MOBAs [9].

To understand these differences, and the impact this has on feature selection, it is necessary to explain how *Destiny* operates as a game.

The core mechanics of *Destiny* are those of a traditional FPS, and include run, jump, crouch and shoot as well as simple melee combat. The interface provides information such as ammunition, health, a mini-map and floating information text over enemies. Other mechanics more resemble RPGs, with character classes, attributes and levels based on earned experience points, all feeding into the complex damage system. Enemies typically take multiple hits to kill, although some weak enemies can be easily dispatched and headshots provide additional damage. *Destiny* also features an inventory system, a range of collectibles and crafting.

The setting is an extensive persistent game world, exploration driven by quests and available activities in the form of player-versus-environment (PvE) and player-versus-player (PvP) content. Single and multi-player elements both feature heavily, in the style of MMORPGs such as *Everquest* and *World of Warcraft*. There are also clear relationships to team-based FPS games, such as *Call of Duty* and *Team Fortress*, although the persistent game world sets it apart from those games. The overall game experience is one of fast paced combat action with players presented with multiple options in a large persistent world. One unusual feature for an online multi-player game, and one which has caused considerable controversy within the player base, is the limited in-game support for player to player communication.

Story missions direct the player through settings across the solar system, from Earth to the moon and out to the other planets and may be completed with other players but can typically be completed alone. Group activities in *Destiny* are based around a fireteam of three players. Strikes and the larger and more involved Raids are instanced co-operative group content for three and six players, respectively. Other co-operative group-based content, known as public events, take place in the persistent world, where all players that can reach the site of the event can participate.

PvP content (known as Crucible matches) takes place in instanced environments and involve one or two fireteams a side, for a maximum of twelve players. PvP modes include team and individual deathmatch, area control and some less well-known forms, although other familiar modes, such as capture the flag, are absent. The size of the teams gives the matches more of a feel of other small scale PvP content, such as *World of Warcraft* arena battles and small team battles possible in games such as *Counter-strike*, rather than the larger teams possible in many online FPS games that do not have a persistent world.

The restricted communication options, particular the lack of any text based chat channels, produces a different experience to many other MMO games, particularly those played on a PC. Voice communication was, at initial release, only possible between members of pre-formed fireteams, usually consisting of players who know each other outside the game. Recently added is the option of voice communication to players who are randomly put into teams by the matchmaking service in both PvE and PvP content. These voice-chat features are opt-in.

## V. DEFINITIONS

The term *churn* here refers to the process of a player leaving the game indefinitely and discontinuing to be a customer. While it is natural for most players to eventually turn away from a game over time, for games with hybrid revenue models such as *Destiny* holding onto players for as long as possible not only increases revenue but maintains a higher density of player interaction in the MMOG setting. Therefore, retaining players by either encouraging churned players to return or preventing current players from churning in the near future has a vast impact on the success of the game overall.

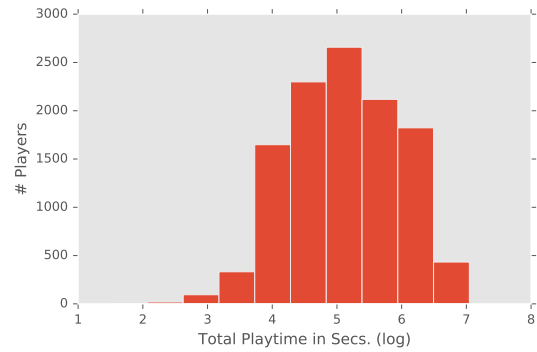


Fig. 1: Total playtime distribution in the log scale.

More specifically, the goal in this work is to identify players who are *about to churn* rather than those that have *already churned*. Enticing a churned player back to the game is less likely to be successful than encouraging a currently active player to continue playing [4]. By analyzing a player's time series data, we seek to predict whether that player is about to churn by identifying patterns of disengagement in in-game activity and how often they play. Additionally, we are predicting late churn of players who have been playing the game for at least a month; this is opposed to predicting early churn in players who have only recently begun playing.

## VI. MODELING

This section starts with an outline of the *Destiny* dataset that was used in the experiments in Section VII. It then gives a detailed account of how the data is pre-processed and labeled in preparation for classification. This pre-processing is focused on preparing the data for use with a HMM classifier. This section then concludes with the specifics of the features used, the setup of the HMM classifier, and the non-temporal classifiers that were tested for comparison purposes.

### A. Dataset

The dataset used in this study contains detailed daily behavioral information of more than 10000 *Destiny* players with 24118 characters that have been randomly sampled from all of the players that played the game at least for two hours. Grouped by the game modes, the dataset contains general metrics about in-game activities such as average scores per kills and number of deaths as well as very detailed information about the gameplay such as the suicides and the performed resurrections. The data covers 17 months of activity starting from September 2014 to January 2016 and the total playtime played by the players is 1,809,564 hours and the average per player is 158 hours. Figure 1 shows a log scaled histogram of total playtime.

### B. Data Pre-processing and Labeling

A subset of players was randomly sampled from all players of the game who played for at least two hours. This threshold is set to eliminate bias imposed by people who never migrate

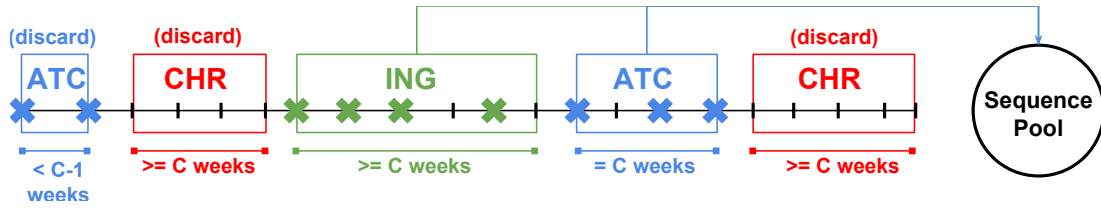


Fig. 2: Example of labeling sequences in time-series data of a player as churned (CHR), about-to-churn (ATC) or in-game (ING) with  $C = 4$ . Black notches on the timeline are weeks with no play data and each colored  $X$  marks a week where at least 120 seconds of play time was recorded. Labels below the timeline show the rule that determined the class assignment.

to becoming actual players of the game, e.g. people who install the game but never play it, or only play it very briefly. The exact placement of the threshold can of course be debated, but was in the current instance based on an investigation of the approximate playtime it takes to navigate the earliest step of the tutorial elements in the game.

The dataset provides daily snapshots of each players activities, performance, and achievements for every day that they played during the sample period. However, this data was further aggregated into weekly snapshots for each sampled player, where days that the player had no activity were filled with null values. If playtime was less than 120 seconds in a week, that week is considered to have no activity and is zeroed out. This threshold value was chosen through intuition but could be further explored through sensitivity analysis.

Aggregation is done because of irregular play behaviour between players within a week. For example, because Destiny play sessions require players to be heavily engaged with the game (unlike say with "casual" games), it is likely that a busy schedule will prevent them from playing for many days in a row. If a player is not active for a few days, it is unlikely that they have churned therefore leading to incorrect predictions.

For the same reason, we define a *churn window* ( $C$ ) such that if there is no data for a player during at least a  $C$  weeks period, then that player is considered to have churned during that time. Note that this means that it is possible for a player to churn for a period of time and then return. In all of our experiments, we set  $C = 4$ . The reason for this can be seen in Figure 3 where each point represents a sampled player, plotted against their playtime (in seconds) and their average absence from the game (in days) during the sample period. The density of the plotted player data is much higher for values of *average absence* less or equal to 28 (i.e. 4 weeks).

All of the classification techniques used to predict churn in this paper are trained through supervised learning and so requires the data to contain class labels. Sequences of weeks are labeled as either *churned* (CHR), *about-to-churn* (ATC), or *in-game* (ING). However, all CHR data is discarded and instead we use ATC for positive samples and ING for negative samples for classifier training and testing purposes.

The data is processed backwards through time. If there is no player activity for  $C$  weeks or more, then those weeks are grouped into a sequence and that sequence is given a label of CHR. Any weeks with activity that are within  $C$  weeks

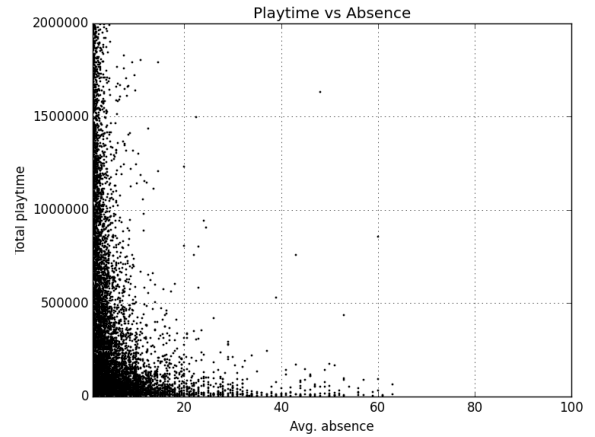


Fig. 3: Play time (in seconds) vs. average absence (in days) for each sampled player during the sampled time.

before the CHR sequence starts are again grouped together and given the label ATC. All other sequences of weeks with activity are given the label ING. Additionally, because players may return after  $C$  weeks of inactivity, it is possible to have more than one of each sequence type for each player. It is also possible for sequences to be less than  $C$  weeks long if, for example, an ATC period is surrounded by CHR periods and doesn't have activity that is at least  $C$  weeks apart. If these sequences are shorter than  $C - 1$  weeks long, then they are discarded. This means that both CHR and ING sequences can span  $C - 1$  or more weeks, while ATC sequences are either  $C - 1$  or  $C$  weeks long. Once these sequences are identified, they are added to a pool of sequences from all players. It is from this pool that training and test set data is drawn to be used with the classifiers.

This process can be seen in the example in Figure 2. The center black line indicates the timeline while each  $X$  marks a week where the player played the game. Here, the player plays for just two weeks before churning but then returns, plays consistently for a number of weeks, their play time becomes more sparse, and then they churn again. As the first two weeks of play are less than our  $C - 1$  threshold, this sequence is discarded along with the CHR sequences. The two central ING and ATC sequences are extracted and added to the global pool of sequences.

### C. Features

All the features below are discretised by comparing the current feature value with that of a rolling average and assigning a value of (0, 1, 2) corresponding to (*less*, *same*, *more*) tags. The rolling average for feature  $x$  at week  $t + 1$  is calculated as  $\sigma_{x,t+1} = \sigma_{x,t} + \alpha * (x_t - \sigma_{x,t})$ , where  $\alpha = 0.4$  is used as a weighting parameter for the experiments below. If the value for the current week is within 20% of the rolling average, it is given the discretised value for being the *same* as the average. Otherwise, it is given the value of *more* or *less* depending on whether it was greater or less than that of the rolling average beyond the 20% threshold.

In choosing features to use, we tested many combinations using experiments similar to those found in the experimental results (Section VII) below to find those that yielded the best HMM performance. The Destiny dataset provides 35 player performance statistics regarding what they did in the game and how well they did it. However, the best combination that we could find (and that we use in the experimental results below) only utilizes three of these, as well as three engineered temporal statistics regarding how often they played. Each week of recorded play is represented by a vector of the following feature values:

- Mean Lifespan: Total number of seconds played divided by the number of times the player has died since they started playing.
- Kill-Death Ratio: Total number of kills the player has divided by the total number of deaths since they started playing.
- Activities Completed Ratio: The ratio of activities that the player completed to the number of activities that they entered.
- Current Absence: Number of weeks the since the player last played. A week with less than 120 seconds of activity has a value of 1, the second consecutive week of no activity will have a value of 2, and so on. If a week contains at least 120 seconds of play time, then this value is 0.
- Current Absence to Mean Absence Ratio: Mean Absence takes into account all absence periods, excluding the current one.
- Weeks Present Ratio: Number of weeks this player has been active divided by the total number of weeks since they first registered to the game.

### D. Classifiers

As we are dealing with time series data, the main classifier that we examine is a multinomial Hidden Markov Model (HMM). The HMM models are learned using the *hmmlearn* Python library (<https://github.com/hmmlearn/hmmlearn>). Sequences of data are extracted in the pre-processing step as each constitutes a sequence of observations to be used by the HMM, with the features of each week making up a single observation.

Two HMM models were trained, one for the ATC class and one for the ING class. By passing the respective data to each

model, they were trained to recognize patterns corresponding to the specified class. During testing, the models take a sequence of observations and it returns the log likelihood that the sequence was produced by the given model, thus giving a probability of the sequence belonging to that class.

If an observation (a single week) in a test set sequence has not been seen by a multinomial HMM during the training phase, then that entire sequence will be unclassifiable. If the sequence is classifiable by one model but not the other, then the predicted class is that of the model that can classify it, even if the log likelihood is low. If the sequence is unclassifiable by both models, then the prediction defaults to ATC.

We also compare the performance of this HMM with that of several other classifiers, utilizing the *scikit-learn* Python library (<http://scikit-learn.org/stable/>). As these are non-time series models, each week in the dataset is given the class label of the sequence that it is a part of. Each week is then joined with  $C - 2$  other neighboring active weeks to create a single sample with  $n * (C - 1)$  features. For example, each sample will have three separate Kill-Deaths-Ratio features, one for each week. In our case, we use the  $n = 6$  features listed earlier and  $C = 4$ , giving 18 features per sample. This is done in order to provide at least some measure of temporal data to the classifiers. We also tested the classifiers by treating each week as a single sample with  $n = 6$  features but this performed worse than the  $n * (C - 1)$  setup. It is also worth noting that using the original feature values performed better than the pre-processed discretised values for all of these non-temporal classifiers and so these original features are used in the results shown below.

## VII. EXPERIMENTAL SETUP AND RESULTS

This section discusses the results of the HMM classifier versus the other non-temporal classifiers across the temporal data. As the HMM training process is stochastic in nature and can lead to different models even when provided with the same training data, the results for the HMM show the average performance of 15 training and testing cases using the same training and set sets. Meanwhile, all other classifiers use stratified 10-fold cross validation on the same training set as the HMM. The results for these classifiers show the performance of the best model identified by cross fold validation running on the same test set as the HMM.

The training set and test set were formed by splitting the sampled data on the date of the 20th of October, 2015. Any sequences with weeks entirely before or on this date were added to the training set, while those with weeks entirely after or spanning both sides of this date were added to the test set. In total there were 12086 ATC sequences and 3996 ING sequences in the training set. There were 3065 ATC and 927 ING sequences in the test set. In terms of individual weeks, there were 69287 ATC weeks and 49962 ING weeks in the training set and 33497 ATC weeks and 35235 ING weeks in the test set. Only 0.9% of sequences were unclassifiable by the ING HMM model while 0.4% were unclassifiable by the ATC model.

TABLE I: Results of the best models found for non-temporal classifiers and the mean HMM performance on the same training and test data sets. Bold values are best in column.

Classifier	Prec	Acc	Recall	F1	AUC
Theoretical Random Classifier	0.75	0.5	0.5	0.6	0.5
Bagging	0.54	0.55	0.49	0.51	0.56
Naive Bayes	0.55	0.57	0.70	0.61	0.61
Nearest Neighbor	0.52	0.53	0.57	0.54	0.54
Gradient Boosting	0.56	<b>0.58</b>	0.63	0.59	0.61
Decision Tree	0.55	0.57	0.60	0.58	0.59
Discriminant Analysis (Quadratic)	0.54	0.56	0.73	0.62	0.60
Discriminant Analysis (Linear)	0.54	0.56	0.75	<b>0.63</b>	0.61
Ada Boost	0.56	<b>0.58</b>	0.66	0.60	0.61
Logistic Regression	0.54	0.57	0.75	<b>0.63</b>	0.61
Random Forest	0.54	0.56	<b>0.76</b>	<b>0.63</b>	0.60
Hidden Markov Model	<b>0.92</b>	0.53	0.43	0.57	<b>0.77</b>

### A. Results

Table I shows the results of these tests, given as precision, accuracy, recall, F1 score (all with no bias in the binary classification threshold), and area under ROC curve (AUC). Bold values indicate the classifier that performed the best for the given metric. Theoretical results of a random classifier are also provided, calculate from the class distributions in the previous section. From these results we can see that the HMM far outperforms the other classifiers on precision, meaning that when it predicts that a player is about-to-churn (ATC), then it is highly likely that the player will churn. However, the HMM recall is the worst, which means the HMM is conservative in its predictions and is failing to identify at least half of ATC players. The fewer true positive predictions also gives the HMM the worst accuracy.

This result is opposite to many of the non-temporal classifiers, such as the Random Forest classifier, that has a high recall but a low precision. These classifiers are acting quite liberally in predicting that players are ATC. Overall the HMM performs best with respect to the AUC, suggesting a better balance between precision and recall overall with various prediction thresholds. Figure 4 shows the ROC curves of the HMM model pairs with the minimum, maximum, and median AUC values. This figure highlights that, especially for the best performing HMM, the recall (true positive rate) climbs rapidly as false alarm rate (false positive rate) increases. This suggests that by using an unbalanced binary prediction threshold to be slightly more generous with positive predictions, the HMM could increase recall to a reasonable level while not sacrificing too much precision.

### B. Discussion

There is a choice to be made in which classifier would be best to deploy to the real-world, based upon the results shown here. Let us assume, for example, that players identified as

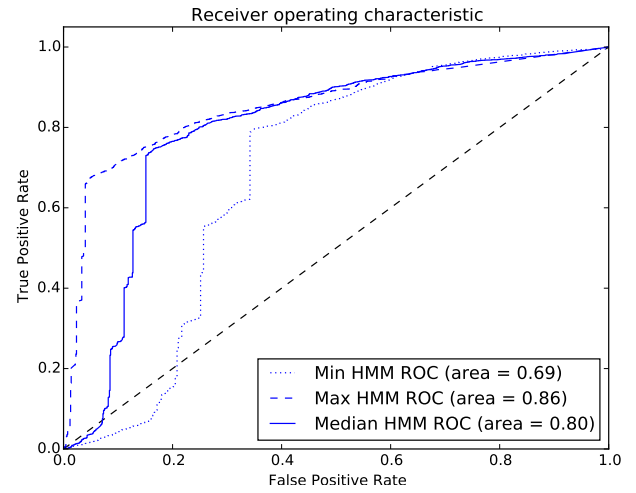


Fig. 4: The ROC curves for the best, worst, and median pair of HMM models found after 15 separate stochastic training runs.

ATC at any given time are offered an incentive by Bungie to continue playing *Destiny*. If this incentive is costly to Bungie or could hurt the in-game economy (such as giving players a rare in-game item), then precision is more important as the company would only want to give such an expensive incentive to those who are truly about to churn. However, if say the incentive is just a common in-game item corresponding to the players level then a higher recall is more important: it is less important if the non-churning player receives this bonus, as long as more players are encouraged to stay in the game. Overall, the HMM will also provide the best balance between these two simply by adjusting the binary classification threshold.

HMMs were chosen as the focus here as a representative of classifiers that have the ability to factor in additional temporal information from the in-game behavior of the players, in this case performance-based features. However, similar to work in F2P and MMOGs, the best performance relies on a mixture of in-game and temporal features, the latter describing how often the player was active. Of interest is that the combination of these features provided similar performance to that of [10] (based upon their provided ROC curve) who operated in a F2P game context, even though the authors only use specifically temporal features. There is a vast difference in the relative complexity of the design and mechanics between *Destiny* and F2P games. It is also interesting to see the importance of temporal features across these diverse game situations, which was speculated on in the work of Bauckhage et al. [23] and Sifa et al. [15], who hypothesized an underlying model of player interest in games from observing playtime distributions across more than 3000 games.

It may then be argued that the player performance features offer little benefit for churn prediction and that both the HMM and the non-temporal classifiers would be better served using only temporal features. Temporal features have previously



shown promise as a genre agnostic feature set for a wide variety of F2P games [5] and a single-player sandbox game [21]. However, when testing feature combinations similar to those used by [5], [21], we witnessed significantly poorer performance in both types of classifiers. This suggests that features used for successful churn prediction in F2P mobile and single player games may not be applicable in the context of MMO console games. This highlights the need for more research into potentially generalizable feature sets for game analytics. In the absence of those general features though, it remains important to consider the genre of the game and the context of the data when addressing the churn prediction problem.

## VIII. CONCLUSIONS AND FUTURE WORK

In this paper detailed time-series behavioral feature data from 10,000 randomly selected players from the hybrid FPS online game *Destiny* have been analyzed for the purpose of building a churn prediction model. Based on an application of Hidden Markov Models, a churn prediction model is presented. The HMM model has been benchmarked against an array of classifiers, and the relative performance of different approaches described and discussed. The results presented highlight the differences in the demands on the behavioral features used for prediction across game types, as is clear when comparing work across F2P mobile games, MMOGs, single-player games and now hybrid online games.

The work included here represents a first step towards building behavioral prediction models in *Destiny* and similar games. The results form a step on the way to developing robust predictive models for AAA-level commercial game titles, similar to the models currently available in F2P mobile games. With the increasing focus within major commercial game titles towards extending the interaction period, churn prediction models are an important first step in developing games that are user-responsive and able to adapt to prevent player disengagement, as is currently being explored in F2P mobile games [4] and Game AI [16].

Future work will focus on improving the precision and recall rates, for example by further feature engineering, moving beyond performance-based and temporal metrics, such as progression metrics (e.g. character level, faction reputation, missions accomplished). Future work will also investigate the potential for predicting other aspects of player behavior, notably related to monetization, social behavior and game content absorption. The latter forms an example of game-based behavioral predictions that directly target informing design, as compared to the monetization and retention focus common in prediction work in games, and forms a venue in game analytics that has not been well explored.

## ACKNOWLEDGMENT

The authors would like to extend their sincere gratitude to Bungie for making telemetry from *Destiny* available.

## REFERENCES

- [1] "Activision blizzard announces better-than-expected first quarter 2015 financial results," <http://bit.ly/1rLD9uK>, accessed: 01-05-2016.
- [2] M. Seif El-Nasr, A. Drachen, and A. Canossa, Eds., *Game Analytics – Maximizing the Value of Player Data*. Springer, 2013.
- [3] R. Sifa, F. Hadji, J. Runge, A. Drachen, K. Kersting, and C. Bauckhage, "Predicting Purchase Decisions in Mobile Free-to-Play Games," in *Proc. of AAAI AIIDE*, 2015.
- [4] J. Runge, P. Gao, F. Garcin, and B. Faltings, "Churn Prediction for High-value Players in Casual Social Games," in *Proc. of IEEE CIG*, 2014.
- [5] F. Hadji, R. Sifa, A. Drachen, C. Thureau, K. Kersting, and C. Bauckhage, "Predicting Player Churn in the Wild," in *Proc. of IEEE CIG*, 2014.
- [6] R. Thawonmas, K. Yoshida, J.-K. Lou, and K.-T. Chen, "Analysis of Revisitations in Online Games," *Entertainment Computing*, vol. 2, no. 4, pp. 215–221, 2011.
- [7] D. Pittman and C. GauthierDickey, "Characterizing Virtual Populations in Massively Multiplayer Online Role-playing Games," in *Proc. of the 16th Int. Conf. on Advances in Multimedia Modeling*, 2010, pp. 87–97.
- [8] W. Feng, D. Brandt, and D. Saha, "A Long-term Study of a Popular MMORPG," in *Proc. of the 6th ACM SIGCOMM Workshop on Network and System Support for Games*, 2007.
- [9] A. Schubert, M.; Drachen and T. Mahlman, "Esports Analytics Through Encounter Detection," in *Proc. of the 10th MIT Sloan Sports Analytics Conference*, 2016.
- [10] P. Rothenbuehler, J. Runge, F. Garcin, and B. Faltings, "Hidden Markov Models for Churn Prediction," in *Proc. of SAI IntelliSys*, 2015.
- [11] H. Xie, S. Devlin, D. Kudenko, and P. Cowling, "Predicting player disengagement and first purchase with event-frequency based data representation," in *Proc. IEEE Conference on Computational Intelligence and Games*, 2015, pp. 230–237.
- [12] M. Mozer, R. Wolniewicz, D. Grimes, E. Johnson, and H. Kaushansky, "Predicting Subscriber Dissatisfaction and Improving Retention in the Wireless Telecommunications Industry," *IEEE Trans. on Neural Networks*, vol. 11, no. 3, pp. 690–696, 2000.
- [13] B. Medler, "Play with Data – An Exploration of Play Analytics and Its Effect on Player Experiences," Ph.D. dissertation, Georgia Institute of Technology, 2012.
- [14] T. Mahlmann, A. Drachen, J. Togelius, A. Canossa, and G. N. Yannakakis, "Predicting Player Behavior in Tomb Raider: Underworld," in *Proc. of IEEE CIG*, 2010.
- [15] R. Sifa, C. Bauckhage, and A. Drachen, "The Playtime Principle. Large-Scale Cross-Games Interest Modeling," in *Proc. of the IEEE Computational Intelligence in Games*, 2014, pp. 139–146.
- [16] G. N. Yannakakis and J. Togelius, "A Panorama of Artificial and Computational Intelligence in Games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 7, no. 4, 2015.
- [17] J. Kawale, A. P., and J. Srivastava, "Churn Prediction in MMORPGs: A Social Influence Based Approach," in *Proc. of CSE*, 2009.
- [18] J. S. Z. Borbora, K. Hsu, and D. Williams, "Churn Prediction in MMORPGs Using Player Motivation Theories and an Ensemble Approach," in *Proc. of IEEE International Conference on Social Computing*, 2011.
- [19] D. Nozhnin, "Predicting Churn: Data-Mining Your Game," *Gamasutra*, 2012.
- [20] T. Mahlman, A. Drachen, A. Canossa, J. Togelius, and G. N. Yannakakis, "Predicting Player Behavior in Tomb Raider Underworld," in *Proc. Conference on Computational Intelligence in Games*, 2010, pp. 178–185.
- [21] R. Sifa, S. Srikanth, A. Drachen, C. Ojeda, and C. Bauckhage, "Predicting Retention in Sandbox Games with Tensor Factorization-based Representation Learning," in *Proc. of IEEE CIG*, 2016.
- [22] P. Yang, B. Harrison, and D. L. Roberts, "Identifying Patterns in Combat that are Predictive of Success in MOBA Games," in *Proc. of the Foundations of Digital Games*. FDG, 2014.
- [23] C. Bauckhage, K. Kersting, R. Sifa, C. Thureau, A. Drachen, and A. Canossa, "How Players Lose Interest in Playing a Game: An Empirical Study Based on Distributions of Total Playing Times," in *Proc. of the IEEE Computational Intelligence in Games*, 2012.

# Automated Game Balancing of Asymmetric Video Games

Philipp Beau

University of Amsterdam  
Intelligent Systems Laboratory  
Amsterdam, The Netherlands  
Email: philipp.beau@student.uva.nl

Sander Bakkes

Tilburg University  
Tilburg centre for Cognition and Communication  
Tilburg, The Netherlands  
Email: s.c.j.bakkes@uvt.nl

**Abstract**—Designing a (video) game such that it is balanced - i.e. fair for all players - is a prevailing challenge in game design. Perhaps counter-intuitively, games that are symmetric with respect to (board) design, starting conditions, and the employed action set, are not necessarily fair games. Indeed, perfect play from all players does not automatically lead to a draw, but may probabilistically favour e.g., the first player to move. Even more so, *asymmetric games* – in which the action set of one player is typically highly distinct from that of another player – are generally unbalanced unless meticulous care has been taken to ensure that the asymmetry in the design does not skew win probabilities. In this context, the present paper contributes a method for automatically balancing the design of asymmetric games. It employs Monte Carlo simulation to analyse the relative impact of game actions, and iteratively adjusts attributes of the game actions till the game design is balanced by approximation. To assess the effectiveness of the proposed method, experiments were performed with automatically balancing a set of tower-defence games. Preliminary experimental results revealed that the proposed method (1) is able to identify the principal component of a game's imbalance, and (2) can automatically adjust the game design till it is balanced by approximation.

## I. INTRODUCTION

It is generally acknowledged that a (video) game needs to be balanced in order to be enjoyable [1]. Informally speaking, this entails that every player, given they possess equal skill, should have the same probability of winning the game. In symmetrical games, where each player can always choose from the same action set and can always start from a position analogous to that of the opponent player(s), this is generally assumed by default (*cf.*, rock-paper-scissors). As such, derived from Herik et al. [2], we consider a game a balanced (i.e. fair) game if it is a game-theoretical draw, and both players have roughly an equal probability on making a mistake. In games containing asymmetrical choices however – like most multiplayer strategy games – each player typically starts with a set of actions that is highly distinct from that of other players, making balancing the game design particularly challenging [3]. Foremost, the challenge follows from attributing the relative impact of an action on the win probability. That is, the effectiveness of an individual action is not directly apparent, as it is dependent on the context in which it is executed.

As current-day video games typically contain dozens to hundreds of different types of (unit / building) actions, it has become highly challenging for a human designer to identify the precise action which causes the game design to be imbalanced.

Currently game designers rely on extensive and expensive human testing [4], requiring a considerable amount of time and effort which can even go far past the public release of a game. As such, developing a method which can automatically identify (and correct) the cause of an unbalanced game would accelerate the design process, and can be assumed to positively impact game design practise.

The contribution of the present paper, therefore is a method for automatically balancing the design of asymmetric games. It employs Monte Carlo simulation to analyse the relative impact of game actions, and iteratively adjusts attributes of the game actions till the game design is balanced by approximation.

## II. RELATED WORK

In related work, Jaffe [5] investigates the restricted-play balance framework, arguing for a mathematical formulation of game balance in which carefully restricted agents are played against standard agents. The work foremost contributes to the field of quantitative balance analysis, and is related to that of Nelson [6], who conceptually explores strategies for automatically extracting balance information from games. Also, Mahlmann et al. [7] have previously investigated evolving card sets to automatically balance the game of Dominion. Van Rozen et al. [8] have performed work on generating balanced tower defence games using *MicroMachinations*.

Indeed, the topic of automated game balancing is of general interest to the gaming community (*cf.* e.g., Elias et al. [9], Chapter 4.4). As such, Kim et al. [10] investigated a system to collect and visualise data from user studies, called TRUE. Their system analyses player deaths to find the cause of unintended difficulty artefacts introduced during development. Debeauvais et al. [11] uses aggregated data from the racing game *Forza Motorsport 4* to analyse how players use and customise driving assists; enabling them to balance the difficulty level of the game. Also, Lewis and Wardrip-Fruin [12] collected and analysed large quantities of game data from the popular MMORPG *World of Warcraft* which they used to investigate common player assumptions, such believed imbalances in specific game classes being more efficient for reaching the maximum character level. Indeed, while a plethora of research exists on dynamic difficulty adjustment (DDA), deep analysis of the balance of a game design – and its principal components – and the automated balancing thereupon, is still a relatively under-explored field.

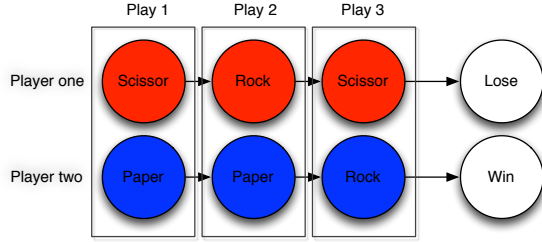


Fig. 1: Action history of a best of three rock, paper, scissor game.

### III. METHOD

Here we describe our method to automated game balancing of asymmetric video games. It consists of maintaining an administration of action histories (III.A) in the form of an action history tree (III.B). We give an example of an (im)balanced game, (III.C), and describe our procedure to identifying actions of high impact on a game's win probability (III.D); the method builds upon Monte Carlo tree search (MCTS) techniques.

#### A. Action history

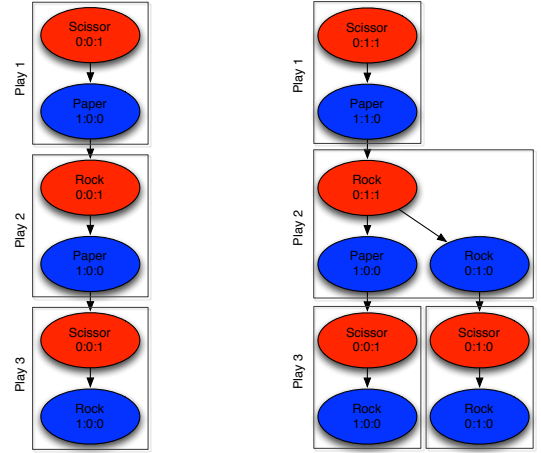
We consider the action history in a game to be the concatenation of actions taken by a player within the game, leading to a certain outcome. For instance, looking at a game of best of three rock-paper-scissors (RPS). If **player one** plays scissor, rock, scissors and **player two** plays paper, paper, rock this leads to player two winning the game and player one losing. The action history of this RPS game is thus  $(scissor \times paper), (rock \times paper), (scissor \times rock)$  which leads to a loss for player one and a win for player two.

#### B. The action history tree

The simulated action histories are stored in a tree structure. The root node of the tree is the first action of **player one**, and is succeeded by the first action of **player two**, then the second action of player one, etc. In every node the observed number of wins, draws, and loses subsequent to that action is stored; the metrics are derived from all simulated playouts from this node onward. Figure 2a shows the action history tree for the action history shown in Subsection III-A. Adding another action history to the tree, player one (**Scissor, Rock, Scissor**) vs. player two (**Paper, Rock, Rock**), results in Figure 2b.

#### C. Example of an (im)balanced game

Consider the – balanced – rock-paper-scissors (RPS) game; a zero-sum hand game usually played between two people, in which each player simultaneously forms one of three shapes with an outstretched hand. The action set for both players is identical, and the game is designed such that each player has an identical probability of winning the game. To create an imbalanced game on the basis of RPS, a fourth action called *Spock* can be introduced. The Spock action only loses against paper, and replaces rock of player one in the second play of the best of three RPS game (Table Ia). As a result the win ratio of the game becomes skewed towards player one (Table Ib) making RPS+Spock an imbalanced game. As such, the Spock action can be identified as an action with high impact on the game's win probability, in this case skewing the win probability in favour of player one.



(a) One action history (b) Two action histories

Fig. 2: Action history tree.

TABLE I: Results of RPS+Spock in the second play of the game (a), and the outcome of 10000 best of three RPS+Spock games (b).

(a)					(b)	
		Rock	P2 Paper	Scissor	Winner	Ratio
P1	Rock	Tie	P2	P1	Player one	42%
	Paper	P1	Tie	P2	Tie	25%
	Scissor	P2	P1	Tie	Player two	33%
	Spock	P1	P2	P1		

#### D. Discovering high impact actions in a game

In complex, actual video games, play-out information such as that in Table Ia and Ib is typically unavailable, among other reasons because game actions generally can not be directly pitted against each other in isolation. What is feasible, however, is observing the win ratio of simulated games, and monitoring the actions played to construct a means of analysis – such as building an action history tree.

Having an imbalanced game, the goal is to identify high impact actions which are leading to a win disproportionately more often than a loss. In an effort to calculate the impact of an action on a game one first has to look at the average win/draw/lose ratio (AWR, ADR, ALR) following an action of a player. In other words, does an action in general lead to a win more often than other actions of the same player.

$$\begin{aligned}
 AWR_{action} &= \sum_a^A a.win / \sum_b^A (b.win + b.draw + b.lose) \\
 ADR_{action} &= \sum_a^A a.draw / \sum_b^A (b.win + b.draw + b.lose) \\
 ALR_{action} &= \sum_a^A a.lose / \sum_b^A (b.win + b.draw + b.lose)
 \end{aligned}$$

where  $A$  = all nodes of *action* in the action history tree

In the example of RPS+Spock of Table IIb, spock outperforms all other actions of player one in AWR and ALR, with the AWR of spock (49%) being substantially above the overall win ratio of 42% (Table Ib). Second, we introduce  $AWR_{a,w}$ ,  $ADR_{a,w}$  and  $ALR_{a,w}$  which is defined as the AWR, ADR, ALR of an action  $a$  given the absence of action  $w$ ; it is calculated via Algorithm 1. That is, procedure *CalcWDLRatio* calculates the average win/draw/loss ratio of an action  $a$  in

TABLE II: Average win, draw and lose ratio after a specific action got played.

(a) Rock-paper-scissors				
		AWR	ADR	ALR
P1	Rock	0.37	0.26	0.37
	Paper	0.37	0.26	0.37
	Scissor	0.36	0.26	0.38
P2	Rock	0.38	0.25	0.37
	Paper	0.37	0.26	0.37
	Scissor	0.36	0.27	0.37

(b) Rock-paper-scissors-spock				
		AWR	ADR	ALR
P1	Rock	0.42	0.24	0.33
	Paper	0.40	0.25	0.34
	Scissor	0.39	0.25	0.35
	Spock	0.49	0.21	0.29
P2	Rock	0.32	0.24	0.44
	Paper	0.34	0.25	0.40
	Scissor	0.35	0.24	0.41

TABLE III: The calculated impact for each action of player one in RPS+Spock.

Action	Impact
Rock	-0.06
Paper	-0.04
Scissors	-0.03
Spock	+0.11

the game if the action  $w$  would not exist. If the average win/draw/loss ratio is affected,  $a$  can be considered dependent on  $w$ . Procedure *CalcInternal* is an internal subroutine of procedure *CalcWDLRatio*. The result of the algorithm is a vector containing

$$\begin{bmatrix} AWR_{a,w} \\ ADR_{a,w} \\ ALR_{a,w} \end{bmatrix}$$

Finally we define the *impact*  $I$  of an action  $b$  as how much does  $b$  affect other actions towards a higher win ratio when used in the same game. In other words, how big of a positive/negative impact does a specific action have on other actions and ultimately the game.

$$I(b) = \sum_a^A (AWR_a - AWR_{a,b}) + 0.5 \times (ADR_a - ADR_{a,b}) - (ALR_a - ALR_{a,b})$$

where  $A$  = all nodes of *action* in the action history tree

We surmise that in an imbalanced game, where the win probability is skewed towards one player (race / class, etc.), the action with the highest impact of the winning player is the source of the imbalance. In our example, having calculated the impact  $I$  of every action  $b$  with Algorithm 1, one observes that of all actions that player one can use in RPS+Spock (Table III), the imbalanced action spock of player one can be identified as the action with the highest impact on the game.

#### IV. SIMULATOR

For our experiments, we investigate a highly asymmetric type of strategy game, namely the popular tower defence (TD) games. A typical TD game (illustrated in Figure 3) is highly

**Algorithm 1** Calculate  $AWR_{a,w}$   $ADR_{a,w}$  and  $ALR_{a,w}$

```

1: procedure CALCWDLWITHOUT( $a, w$ )
2:   result = [0,0,0]
3:   for all node in actionHistoryTree.startingNodes do
4:     result += CalcInternal( $a, w$ , node, false)
5:   end for
6:   total = result[0] + result[1] + result[2]
7:   return result /= total
8: end procedure
9:
10: procedure CALCINTERNAL( $a, w$ , node, found)
11:   result = [0,0,0]
12:   if node.player != a.player then
13:     for all child in node.children do
14:       result += CalcInternal( $a, w$ , child, found)
15:     end for
16:     return result
17:   end if
18:   if node.action =  $a$  then
19:     result += node.result
20:     found = true
21:   else if node.action =  $w$  then
22:     if found then
23:       return -node.result
24:     else
25:       return result
26:     end if
27:   end if
28:   for all child in node.children do
29:     result += CalcInternal( $a, w$ , child, found)
30:   end for
31:   return result
32: end procedure

```

asymmetric, in that one player can spawn units that traverse a predetermined path (to attack the tower of the opponent), while the opponent player can construct buildings alongside the path that attack these units (to defend the tower). The offensive player wins if she succeeds in destroying the tower of the opponent, the defensive player wins if she can withstand these attacks. Indeed, the actions that players can take are highly distinct from each other, and balancing their effectiveness generally requires meticulous manual balancing.

We developed a simulator for real-time TD games, which – for rapid experimenting – provides the ability to decouple graphics from actual gameplay.<sup>1</sup> Indeed, TD is a genre which is popular within the gaming community and offers a variety of research opportunities such as dynamic difficulty adjustment, map generation, and player modelling [13], [14], [15]. Our hope is that the developed simulator may contribute to such related branches of research as well.

##### A. Experimental Implementation

The developed simulator contains two races, the *human race* and the *alien race*. Both races are distinguished by the towers they can build (IV-E3), while having the same selection of units and upgrades to choose from (IV-E). In this section we first give a brief overview of how the developed game works before explaining each design aspect in detail.

<sup>1</sup>The developed simulator is publicly available at Github <https://github.com/philiippi/multiplayer-balancing>



Fig. 3: Example 6 × 6 tower defence game field.

```

OOOSOO  O: Tower field
OXXXXO  X: Unit field
OXOOOO  S: Start unit field
OXXXXO  E: End unit field
OOOOXO
OEXXXO

```

Fig. 4: Representation of the example 6 × 6 game field in Figure 3.

In our simulated TD games, two players are playing against each other. The game is played simultaneously on two identical maps; one for the alien player, one for the human player. Each player builds towers on their own map. Also, each player sends creeps (units) to the other player's map. Each player can upgrade the units it will send to the other players map (once an upgrade was chosen, all consecutive sent units will have the improved attributes, units already sent stay the same). If a unit reaches the end of a player's map without getting 'killed' this player loses a live. If a player is at 0 or less lives, he/she loses. If that happens to both players at the same time it's a tie.

### B. Game field

The game field or map consists of a  $n \times n$  grid of fields. A field can be either a tower field or a unit field. The tower fields belong to the player playing on that map for placing towers, while the unit fields are where the opposing players units will be walking. Units get spawned at the start field and are walking towards the end field. Each tower field can be uniquely identified by its position starting with zero at the top left corner and ending at N-1 in the bottom right where N denotes the total amount of tower fields of the map (Figure 3).

New game maps can easily be created using the notation of Figure 4 inside a text file and placing it inside the root directory. The framework will automatically parse the file and convert it into a map if all of the following attributes are met. First, exactly one start field S and exactly one end field E must be present at the boundary regions of the map. Second, the start and the end fields must be connected via unit fields. Third, in every map, the way units walk must be uniquely identifiable. Meaning, a unit field must be adjacent to exactly two other unit fields with the exception of the start and end field which have to have exactly one adjacent unit field.

### C. Towers

In the simulator, races differ from each other with respect to the towers that are available to them. Every tower in the game

TABLE IV: Possible unit upgrades that the players may select.

Id	Attribute	Initial	Increase per upgrade
H	Health	1.0	+0.7
M	Movement	1.0	+0.4
A	Amount	1.0	+0.75

is exclusive to one race. Where the human race can build the fire, ice and archer tower, the alien race has the chain lightning, parasite and shock tower. To provide a realistic challenge to our method, every tower (action) is given two attributes, the damage dealt with each shot, and the distance that it can shoot. The distance or range of a tower is denoted in game fields reachable from the place the tower got build using horizontal, vertical or diagonal movement. In addition, most towers have a unique special ability, as detailed in Table V and Table VI.

### D. Game cycle

At the start of every game both players enter with the same amount of lives and an instance of the same  $n \times n$  map. Subsequently, the game cycles through the following three steps. (1) Both players choose one action which gets executed right away. An action can either be to send units, to upgrade all subsequent units in health, movement or amount or to build a tower on one of the free tower fields of the players map. (2) The already placed towers will pick a unit inside their range to shoot at. The decision which unit is picked at a given point is dependent on the type of tower choosing the target. Fire, archer and chain lightning towers will shoot at their last damaged target. If their last target is either dead or out of range, they choose a target at random. Ice, parasite and shock towers will preferably shoot at a unit currently not influenced by their special ability. If no such unit exists they will choose a target at random too. (3) All units walk appropriate to their movement attribute towards the end of the game field. If a unit walks out of the map the player playing on that map loses a life.

If after step 3 none of the three criteria are met, the game continues with step 1. The three end criteria are (1) a player has 0 or less lives (the other player wins), (2) both players have 0 or less lives (tie), or (3) game exceeded the maximum amount of cycles (tie).

### E. Actions

In the simulated tower defence game, two types of actions are available, that is (1) actions which are available to both races like upgrading and sending units, called *global actions*, and (2) actions only available to players of a certain race, called *race specific actions*, such as placing a specific tower.

1) *Upgrading units*: At every step of the game, a player can choose to (1) upgrade one of the attributes of the units, or (2) upgrade the amount of units that is spawned at starting position. A unit has two attributes, *health* and *movement*. Health describes the amount of damage a unit can take until they get removed from the map. Movement describes the amount of fields a unit walks along its path at every step.

2) *Sending units*: If at any step a player chooses to send units, the game will create units using the attributes of Table IV. Every attribute  $a$  will be set to:

TABLE V: Towers available to the Human race.

Id	Tower	Damage	Range	Special
F	Fire	0.6	1	Damages all units on one field
I	Ice	1	1	Reduces movement speed by 90% for 3 steps
A	Archer	1	3	-

TABLE VI: Towers available to the Alien race.

Id	Tower	Damage	Range	Special
C	Chain lightning	0.4	1	Damages 3 units less than 3 fields apart
P	Parasite	0	1	Lets the hit unit walk backwards for 2 steps
S	Shock	1	1	Reduces movement speed to 0 for 3 steps

$$I_a + t_a * U_a \quad (1)$$

where  $I_a$  is the initial value of attribute  $a$ ,  $t_a$  denotes the amount of times  $a$  was upgraded, and  $U_a$  denotes the increase per upgrade. While all three attributes hold floating point values, the amount attribute only takes the integer-part into account. E.g., if the amount is 2.5, then only 2 units will be spawned.

3) *Placing towers*: Next to actions available to both races, each race can build towers that are unique to its race (discussed in Section IV-C). Where the human race can build the fire, ice and archer tower, the alien race has the chain lightning, parasite and shock tower. Every tower has two attributes, the damage dealt with each shot and the distance it can shoot in game fields (Table V and Table VI).

## V. EXPERIMENTS

Here, we discuss two experiments that test our method to automated game balancing in an asymmetric tower defence video game. The first experiment evaluates to what extent our method can identify and automatically correct an imbalance in a reasonably well-balanced asymmetric game. The second experiment evaluates to what extent our method can identify and automatically correct an imbalance in a strictly imbalanced asymmetric game.

### A. Data generation

The generation of player data was performed with two Monte Carlo tree search (MCTS) [16] agents playing against each other. One MCTS agent was playing the alien race, the other MCTS agent was playing the human race. The two MCTS agents learned from 100.000 playouts of playing against each other (backpropagating rewards to learn effective behaviour), and the last 5.000 action histories were used to populate the action history tree.

The specific MCTS algorithm used by the agents is referred to as Upper Confidence Tree (UCT) [16] which is extending the Upper Confidence Bound algorithm by Auer et al.[17] to trees. UCT combines the exploration and the building of the tree. The tree starts at the root node, after which the algorithm

iterates through three phases: the bandit phase, the tree building phase, and the random walk phase.

*The bandit phase* starts in the root node where to agent continually chooses an action/child node until arriving in a leaf node. The decision which action is taken at every step is handled as a multi armed bandit problem. The set  $\mathcal{A}_s$  of possible actions  $a$  in a node  $s$  defines the child nodes  $(s, a)$  of  $s$ . The selected action  $a^*$  maximises the upper confidence bound:

$$\hat{r}_{s,a} + \sqrt{c_e \log(n_s) / n_{s,a}} \quad (2)$$

over all  $a$  in  $\mathcal{A}_s$  with  $\hat{r}_{s,a}$  describing the average reward accumulated by selecting action  $a$  in state  $s$ ,  $n_s$  the total number of times node  $s$  was visited and  $n_{s,a}$  the amount of times action  $a$  was taken from node  $s$ . The term  $c_e$  handles the exploration vs. exploitation trade-off where a high  $c_e$  favours exploration and a low  $c_e$  exploitation.

*The tree building phase* is entered upon arrival in a leaf node. An action is selected uniformly at random and added as a child node of  $s$ .

*The random walk phase* begins after the new child node was added to the tree. At every step an action is taken (uniformly or heuristically) until the game ends. At this point the acquired reward  $r_u$  is back propagated towards the root node and all nodes in this tree run are updated:

$$\hat{r}_{s,a} \leftarrow \frac{1}{n_{s,a} + 1} (n_{s,a} \times \hat{r}_{s,a} + r_u) \quad (3)$$

$$n_{s,a} \leftarrow n_{s,a} + 1; n_s \leftarrow n_s + 1 \quad (4)$$

Both MCTS agents were initialised with the same values for all experiments in the following sections. I.e.,  $\hat{r}_{s,a}$  initial is 0.5, and  $c_e$  is  $3 * \sqrt{2}$ . The terminal reward  $r_u$  for an agent is dependant on the outcome of the game:

$$r_u = \begin{cases} 10 & \text{if agent won} \\ L_{played} - L_{max} & \text{if draw} \\ -2 * L_{max} & \text{if agent lost} \end{cases}$$

with  $L_{max}$  denoting the maximum allowed game length and  $L_{played}$  the actual game length.

### B. Experiment 1

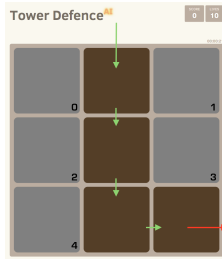
This first experiment evaluates to what extent our method can identify and automatically correct an imbalance in a reasonably well-balanced asymmetric game.

1) *Experimental setup*: All tower attributes were initialised manually with the designer's best intention to create a balanced game (Table VII). The game runs on two different maps, a three by three map (Figure 5a) and a four by four map (Figure 5b) where each player starts with 10 lives and the maximum game length is set to 100 steps. Every run was repeated three times and all numbers in the following section depict the average of those. After the agents have played against each other, the resulting win ratio is calculated, the attributes of the tower with the highest impact of the winning

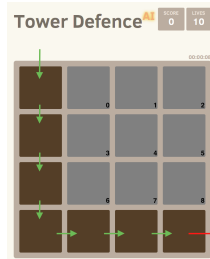


TABLE VII: Experiment 1 – Tower attributes.

Id	Tower	Damage	Range	Special
F	Fire	0.6	1	Damages all units on one field
I	Ice	1	1	Reduces movement speed by 90% for 3 steps
A	Archer	1	3	-
C	Chain lightning	0.5	1	Damages 3 units less than 3 fields apart
P	Parasite	0	1	Lets the hit unit walk backwards for 3 steps
S	Shock	0.5	1	Reduces movement speed to 0 for 3 steps



(a) Three by three map (3x3)



(b) Four by four map (4x4)

Fig. 5: Maps used in the two experiments.

faction will be decreased and the experimental trial is repeated until the difference in win ratio is smaller than 1% and the game considered balanced.

2) *Results on 3x3 map:* Analysis of the provided game design reveals a considerable difference in win ratio between the alien and the human player, with the alien player being 34% more likely to win the game against the human player on the 3x3 map (the human player wins 38%, the alien player 51%, the game ties 11%). This observation leads to the conclusion that this game is imbalanced in favour of the alien player. Calculating the impact of each tower (Figure 6) reveals the chain lightning tower has the largest impact on the game of the alien player.

Following the hypothesis that in an imbalanced game the action with the largest impact, is the action that is presumably causing the imbalance (cf. Section III), the damage of the chain lightning tower was gradually lowered from 0.6 to 0.3 (Table VIII). Lowering the damage output of the chain lightning tower ultimately resulted in a balanced game (Table VIII). Interestingly a direct correlation between the decline of the chain lightning tower's impact (Figure 7) and the decline of the alien players win ratio could be observed, indicating that, indeed, the chain lightning tower was the action causing the imbalance.

To verify these results, indeed, one could argue that the same effect would have been achieved lowering the attributes of the shock and/or parasite tower. To investigate this, the experiment was repeated using three different settings (Table IX), where the attributes of the parasite and/or shock tower were lowered. Looking at the results of each setting in Table X, even substantially lowering the attributes of any alien tower other than the chain lightning tower does not have the same effect on the game as lowering the attributes of the chain lightning tower.

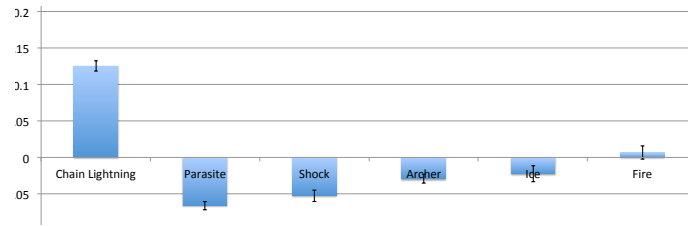


Fig. 6: Impact of every tower using the attributes in Table VII.

TABLE VIII: Win ratio human vs. alien agent depending on the damage of the chain lightning tower.

	0.6	0.5	0.4	0.3
Human	0.38%	40%	42%	44%
Tie	0.11%	11%	12%	12%
Alien	0.51%	49%	46%	44%

Calculating the impact of the towers over the course of the different settings (Figure 8) one can observe that neither settings has a significant impact on the values. The chain lightning tower remains the tower with the highest impact, supporting to the conclusion that it was in fact the chain lightning tower which caused the game to be imbalanced, and was as such corrected adjusted for establishing a balanced game design.

3) *Results on 4x4 map:* In contrast to the 3x3 map, the analysis of the proposed game design reveals a win ratio in favour of the human player, winning the game 17% more often than the alien player (the human player wins 48%, the alien player 41%, the game ties 11%). This indicates that this game is imbalanced in favour of the human player. To find the action responsible for the imbalance, an analysis of the impact of each tower was performed (Figure 9). It reveals that of all towers, the archer tower is the tower with the highest impact on the game.

To assert the correctness of the analysis, we employ the same process as in the previous section. First the damage of the archer tower was gradually lowered until a predictably balanced game was achieved (Table XI). When lowering the damage of the archer tower to 0.4, a balanced game is achieved. Again, a clear correlation between a lower damage and a decrease in impact of the archer tower can be observed (Figure 10).

To independently verify that the archer tower was indeed the source of the imbalance in the human race, we again try to achieve balance lowering the attributes of the other towers (fire and ice) (Table XII). As in the previous section, even substantially lowering the attributes of the other towers did not result in a balanced game (Table XIII). The win ratio changed slightly in favour of the alien player, but the human player still won the game around 7% more often. This leads to the conclusion, that the archer tower was in fact the source of the imbalance as suggested by its impact, and was correctly adjusted to achieve a balanced game design.

4) *Discussion of the results:* In this experiment, the proposed method was tested on two different maps using the same tower attributes. On both maps a balanced game was achieved automatically after adjustment of the correctly identified imbalanced action. On both maps a correlation between decrease

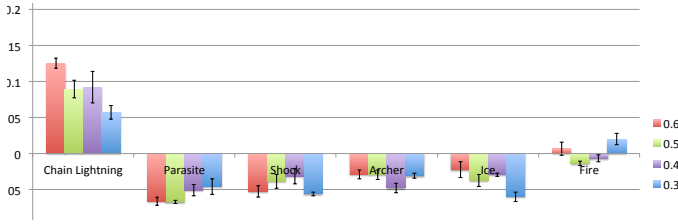


Fig. 7: Impact of the lightning tower, lowering its damage output from 0.6 to 0.3.

TABLE IX: The different settings the experiment was repeated with. Each setting uses Table VII as a foundation.

Setting	
A	Parasite effect reduced to 2 steps
B	Parasite and shock effect reduced to 1 step
C	Setting B and shock tower damage reduced to 0.25

of tower attributes and decrease of impact could be seen when balancing the highest impact tower (HIT), while the impact of it even increased if other towers attributes got lowered; this follows naturally, as the HIT will become more and more important as other towers get weaker.

### C. Experiment 2

This second experiment evaluates to what extent our method can identify and automatically correct an imbalance in a strictly imbalanced asymmetric game, regardless of initial values of the action attributes.

1) *Experimental setup*: To test the proposed method, the following system was setup to automatically balance the tower defence game. This happens in two steps, first the tower attributes are initialised at random using the ranges described in Table XIV. Second, the thus created game is balanced using three different methods subsequently while recording the balancing steps applied by each of them.

A balancing step is defined as the decrease of one attributes value of a chosen tower using the arithmetic in Table XIV to slowly lower the attribute action. The attributes get selected in turns. For example, if the shock tower gets selected three times, the damage attribute will be lowered first, followed by the duration attribute, followed by the damage attribute. The experiment was repeated 10 times on the same  $3 \times 3$  map used in experiment 1 where every player starts with 10 lives and the maximum game length is set to 100 steps. A game is considered balanced if the difference between the human and the alien win ratio is below 1%.

We employ three methods for automated balancing, (1) balance the tower with the highest impact of the winning faction, (2) balance a random tower of the winning faction, and (3) balance a random tower of the winning faction with the exception of the tower with the highest impact. If the impact attribute does accurately predict the unbalanced tower, then the first method should use considerably fewer balancing steps than the one choosing randomly while the method excluding the high impact action should use more.

2) *Result*: The experimental results reveal that using the impact to balance a game (method 1) – with an average of

TABLE X: Win ratio human vs. alien agent given setting A, B, or C.

	Setting A	Setting B	Setting C
Human	39%	40%	41%
Tie	11%	11%	11%
Alien	50%	49%	48%

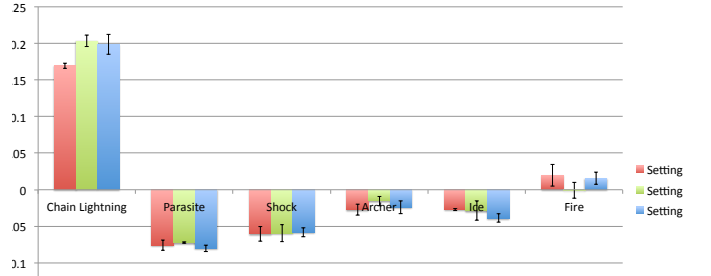


Fig. 8: Impact of every tower after applying setting A, B, or C.

8.1 balancing steps – it is by approximately 65% faster than choosing random towers (Table XV), outperforming method 2 in 10 out of 10 runs. Avoiding the high impact tower in method 3 is 35% worse than picking it randomly (excluding the runs it did not result in a balanced game) and over 2 times worse than using the impact to balance the game. Method 3 never outperforms method 1 but does outperform method 2 in 2 out of 10 times. Method 3 was the only method which did not result in a balanced game in 3 out of 10 times.

3) *Discussion of the results*: This second experiment indicated that the proposed method can identify and automatically correct an imbalance in a strictly imbalanced asymmetric game, regardless of initial values of the action attributes. Indeed, it may be argued that favourable initial parameter settings may render balancing the design a relatively straightforward task. As the present experiment showed, if this actually would have been the case, choosing random actions should have outperformed our method at least once in the course of this experiment. This however, was not the case. Balancing the action with the highest calculated impact consistently outperformed choosing randomly and was on average 65% faster in creating a balanced game. While the experiment should be considered a first step in our investigation, the experimental results indicated that the proposed automated balancing method is able to successfully balance a game design regardless of parameter values suggested by the game designer.

## VI. CONCLUSION

Creating a well balanced multiplayer game is a challenging and tedious task requiring a large amount of human player feedback. The challenge however can be significantly reduced by understanding which actions cause a game to be unbalanced. As such, the present paper contributed a method for automatically balancing the design of asymmetric games. It employs Monte Carlo simulation to analyse the relative impact of game actions, and iteratively adjusts attributes of the game actions till the game design is balanced by approximation. To assess the effectiveness of the proposed method, experiments were performed with automatically balancing a set of tower-defence games. Preliminary experimental results revealed that the proposed method (1) is able to identify the principal

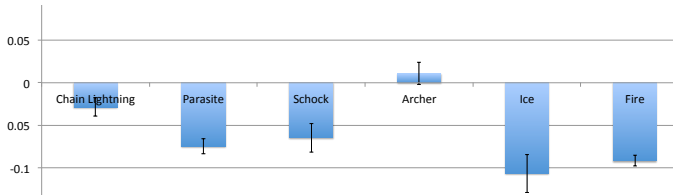


Fig. 9: Impact of every tower using the attributes in Table VII.

TABLE XI: Win ratio human vs. alien agent depending on the damage of the archer tower.

	1.0	0.8	0.6	0.4
Human	48%	47%	46%	44%
Tie	11%	11%	11%	11%
Alien	41%	42%	43%	45%

component of a game's imbalance, and (2) can automatically adjust the game design till it is balanced by approximation.

For future work, we will particularly investigate how the present linear computational effectiveness of the method may be enhanced further, and how the method may be embedded in mixed-initiative game-design toolkits.

## REFERENCES

- [1] A. Rollings and D. Morris, "Game architecture and design: a new edition," 2003.
- [2] H. J. Van den Herik, J. W. Uiterwijk, and J. Van Rijswijk, "Games solved: Now and in the future," *Artificial Intelligence*, vol. 134, no. 1, pp. 277–311, 2002.
- [3] K. Hullett, N. Nagappan, E. Schuh, and J. Hopson, "Empirical analysis of user data in game software development," in *ESEM*. IEEE, 2012, pp. 89–98.
- [4] R. Leigh, J. Schonfeld, and S. J. Louis, "Using coevolution to understand and validate game balance in continuous games," in *GECCO*, 2008, pp. 1563–1570.
- [5] A. Jaffe, A. Miller, E. Andersen, Y.-E. Liu, A. Karlin, and Z. Popovic, "Evaluating competitive game balance with restricted play," in *AIIDE*, 2012.
- [6] M. J. Nelson, "Game metrics without players: Strategies for understanding game artifacts," in *Artificial Intelligence in the Game Design Process*, 2011.
- [7] T. Mahlmann, J. Togelius, and G. N. Yannakakis, "Evolving card sets towards balancing dominion," in *IEEE CEC*, 2012, pp. 1–8.
- [8] R. van Rozen and J. Dormans, "Adapting game mechanics with micro-machinations," in *FDG*, 2014.
- [9] G. S. Elias, R. Garfield, K. R. Gutschera, and P. Whitley, *Characteristics of games*. MIT Press, 2012, 4:4.
- [10] J. H. Kim, D. V. Gunn, E. Schuh, B. Phillips, R. J. Pagulayan, and D. Wixon, "Tracking real-time user experience (true): a comprehensive instrumentation solution for complex systems," in *CHI*, 2008, pp. 443–452.
- [11] T. Debeauvais, T. Zimmermann, N. Nagappan, K. Carter, R. Cooper, D. Greenawalt, and T. Solberg, "Off with their assists: An empirical study of driving skill in Forza Motorsport 4," *FDG*, 2014.
- [12] C. Lewis and N. Wardrip-Fruin, "Mining game statistics from web services: a world of warcraft armory case study," in *FDG*, 2010, pp. 100–107.
- [13] S. K. Lo, H.-C. Keh, and C.-M. Chang, "A multi-agents coordination mechanism to improving real-time strategy on tower defense game," *Journal of Applied Sciences*, vol. 13, no. 5, p. 683, 2013.
- [14] P. Avery, J. Togelius, E. Alistar, and R. P. Van Leeuwen, "Computational intelligence and tower defence games," in *CEC*, 2011, pp. 1084–1091.
- [15] P. Rummell *et al.*, "Adaptive ai to play tower defense game," in *CGAMES*, 2011, pp. 38–40.
- [16] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European Conference on Machine Learning*, 2006, pp. 282–293.
- [17] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

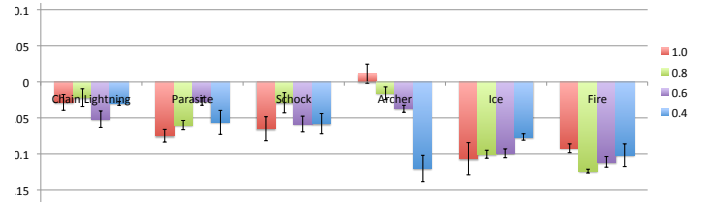


Fig. 10: Impact of the archer tower while lowering the damage from 1.0 to 0.4.

TABLE XII: The different settings the experiment was repeated with. Each setting uses Table VII as a foundation.

Setting	
A	Fire tower damage set to 0.4
B	Setting A and frozen effect reduced to 40%
C	Setting B and ice tower damage reduced to 0.5
D	Setting C and fire tower damage reduced to 0.2

TABLE XIII: Win ratio human vs. alien agent given setting A, B, C, or D.

	Setting A	Setting B	Setting C	Setting D
Human	47%	48%	46%	46%
Tie	11%	11%	11%	11%
Alien	42%	41%	43%	43%

TABLE XIV: Experiment 2 – Tower attributes.

Id	Tower	Attribute	Range	Balancing step
F	Fire	damage	0.0 - 1.0	*0.9
I	Ice	damage	0.0 - 1.0	*0.9
		hindrance	0.0 - 1.0	*0.9
		duration	1 - 9	–1
A	Archer	damage	0.0 - 1.0	*0.9
C	Chain lightning	damage	0.0 - 1.0	*0.9
		jumps	1 - 5	–1
		length	1 - 5	–1
P	Parasite	damage	0.0 - 1.0	*0.9
		duration	1 - 9	–1
S	Shock	damage	0.0 - 1.0	*0.9
		duration	1 - 9	–1

TABLE XV: Experiment 2 – The amount of times a tower's attribute had to be adjusted in order to achieve a balanced game. (\*) = Balance could not be achieved.

Method	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	Avg.
1.	3	5	18	5	2	8	11	14	1	14	8.1
2.	4	9	19	6	9	18	15	26	3	25	13.4
3.	14	41	23	24	*	9	13	*	3	*	18.1

# ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning

Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek & Wojciech Jaśkowski  
Institute of Computing Science, Poznan University of Technology, Poznań, Poland  
wjaskowski@cs.put.poznan.pl

**Abstract**—The recent advances in deep neural networks have led to effective vision-based reinforcement learning methods that have been employed to obtain human-level controllers in Atari 2600 games from pixel data. Atari 2600 games, however, do not resemble real-world tasks since they involve non-realistic 2D environments and the third-person perspective. Here, we propose a novel test-bed platform for reinforcement learning research from raw visual information which employs the first-person perspective in a semi-realistic 3D world. The software, called ViZDoom, is based on the classical first-person shooter video game, Doom. It allows developing bots that play the game using the screen buffer. ViZDoom is lightweight, fast, and highly customizable via a convenient mechanism of user scenarios. In the experimental part, we test the environment by trying to learn bots for two scenarios: a basic move-and-shoot task and a more complex maze-navigation problem. Using convolutional deep neural networks with Q-learning and experience replay, for both scenarios, we were able to train competent bots, which exhibit human-like behaviors. The results confirm the utility of ViZDoom as an AI research platform and imply that visual reinforcement learning in 3D realistic first-person perspective environments is feasible.

**Keywords:** video games, visual-based reinforcement learning, deep reinforcement learning, first-person perspective games, FPS, visual learning, neural networks

## I. INTRODUCTION

Visual signals are one of the primary sources of information about the surrounding environment for living and artificial beings. While computers have already exceeded humans in terms of raw data processing, they still do not match their ability to interact with and act in complex, realistic 3D environments. Recent increase in computing power (GPUs), and the advances in visual learning (i.e., machine learning from visual information) have enabled a significant progress in this area. This was possible thanks to the renaissance of neural networks, and deep architectures in particular. Deep learning has been applied to many supervised machine learning tasks and performed spectacularly well especially in the field of image classification [18]. Recently, deep architectures have also been successfully employed in the reinforcement learning domain to train human-level agents to play a set of Atari 2600 games from raw pixel information [22].

Thanks to high recognizability and an easy-to-use software toolkit, Atari 2600 games have been widely adopted as a benchmark for visual learning algorithms. Atari 2600 games have, however, several drawbacks from the AI research perspective. First, they involve only 2D environments. Second, the environments hardly resemble the world we live in. Third, they

are third-person perspective games, which does not match a real-world mobile-robot scenario. Last but not least, although, for some Atari 2600 games, human players are still ahead of bots trained from scratch, the best deep reinforcement learning algorithms are already ahead on average. Therefore, there is a need for more challenging reinforcement learning problems involving first-person-perspective and realistic 3D worlds.

In this paper, we propose a software platform, ViZDoom<sup>1</sup>, for the machine (reinforcement) learning research from raw visual information. The environment is based on Doom, the famous first-person shooter (FPS) video game. It allows developing bots that play Doom using only the screen buffer. The environment involves a 3D world that is significantly more real-world-like than Atari 2600 games. It also provides a relatively realistic physics model. An agent (bot) in ViZDoom has to effectively perceive, interpret, and learn the 3D world in order to make tactical and strategic decisions where to go and how to act. The strength of the environment as an AI research platform also lies in its customization capabilities. The platform makes it easy to define custom scenarios which differ by maps, environment elements, non-player characters, rewards, goals, and actions available to the agent. It is also lightweight – on modern computers, one can play the game at nearly 7000 frames per second (the real-time in Doom involves 35 frames per second) using a single CPU core, which is of particular importance if learning is involved.

In order to demonstrate the usability of the platform, we perform two ViZDoom experiments with deep Q-learning [22]. The first one involves a somewhat limited 2D-like environment, for which we try to find out the optimal rate at which agents should make decisions. In the second experiment, the agent has to navigate a 3D maze collecting some object and omitting the others. The results of the experiments indicate that deep reinforcement learning is capable of tackling first-person perspective 3D environments<sup>2</sup>.

FPS games, especially the most popular ones such as Unreal Tournament [12], [13], Counter-Strike [15] or Quake III Arena [8], have already been used in AI research. However, in these studies agents acted upon high-level information like positions of walls, enemies, locations of items, etc., which are usually inaccessible to human players. Supplying only raw visual information might relieve researchers of the burden

<sup>1</sup><http://vizdoom.cs.put.edu.pl>

<sup>2</sup>Precisely speaking, Doom is pseudo-3D or 2.5D.



of providing AI with high-level information and handcrafted features. We also hypothesize that it could make the agents behave more believable [16]. So far, there has been no studies on reinforcement learning from visual information obtained from FPS games.

To date, there have been no FPS-based environments that allow research on agents relying exclusively on raw visual information. This could be a serious factor impeding the progress of vision-based reinforcement learning, since engaging in it requires a large amount of programming work. Existence of a ready-to-use tool facilitates conducting experiments and focusing on the goal of the research.

## II. RELATED WORK

One of the earliest works on visual-based reinforcement learning is due to Asada et al. [4], [3], who trained robots various elementary soccer-playing skills. Other works in this area include teaching mobile robots with visual-based  $Q$ -learning [10], learning policies with deep auto-encoders and batch-mode algorithms [19], neuroevolution for a vision-based version of the mountain car problem [6], and compressed neuroevolution with recurrent neural networks for vision-based car simulator [17]. Recently, Mnih et al. have shown a deep  $Q$ -learning method for learning Atari 2600 games from visual input [22].

Different first-person shooter (FPS) video games have already been used either as AI research platforms, or application domains. The first academic work on AI in FPS games is due to Geisler [11]. It concerned modeling player behavior in *Soldier of Fortune 2*. Cole used genetic algorithms to tune bots in *Counter Strike* [5]. Dawes [7] identified *Unreal Tournament 2004* as a potential AI research test-bed. El Rhalib studied weapon selection in *Quake III Arena* [8]. Smith devised a RETALIATE reinforcement learning algorithm for optimizing team tactics in *Unreal Tournament* [23]. SARSA( $\lambda$ ), another reinforcement learning method, was the subject of research in FPS games [21], [12]. Recently, continuous and reinforcement learning techniques were applied to learn the behavior of tanks in the game *BZFlag* [24].

As far as we are aware, to date, there have been no studies that employed the genre-classical *Doom* FPS. Also, no previous study used raw visual information to develop bots in FPS games with a notable exception of the Abel's et al. work on *Minecraft* [2].

## III. ViZDOOM RESEARCH PLATFORM

### A. Why *Doom*?

Creating yet another 3D first-person perspective environment from scratch solely for research purposes would be somewhat wasteful [27]. Due to the popularity of the first-person shooter genre, we have decided to use an existing game engine as the base for our environment. We concluded that it has to meet the following requirements:

- 1) based on popular open-source 3D FPS game (ability to modify the code and the publication freedom),



Figure 1. *Doom*'s first-person perspective.

- 2) lightweight (portability and the ability to run multiple instances on a single machine),
- 3) fast (the game engine should not be the learning bottleneck),
- 4) total control over the game's processing (so that the game can wait for the bot decisions or the agent can learn by observing a human playing),
- 5) customizable resolution and rendering parameters,
- 6) multiplayer games capabilities (agent vs. agent and agent vs. human),
- 7) easy-to-use tools to create custom scenarios,
- 8) ability to bind different programming languages (preferably written in C++),
- 9) multi-platform.

In order to make the decision according to the above-listed criteria, we have analyzed seven recognizable FPS games: *Quake III Arena*, *Doom 3*, *Half-Life 2*, *Unreal Tournament 2004*, *Unreal Tournament* and *Cube*. Their comparison is shown in Table I. Some of the features listed in the table are objective (e.g., 'scripting') and others are subjective ('code complexity'). Brand recognition was estimated as the number (in millions) of Google results (as of 26.04.2016) for phrases "game <gamename>", where <gamename> was 'doom', 'quake', 'half-life', 'unreal tournament' or 'cube'. The game was considered as low-resolution capable if it was possible to set the resolution to values smaller than  $640 \times 480$ .

Some of the games had to be rejected right away in spite of high general appeal. *Unreal Tournament 2004* engine is only accessible by the Software Development Kit and it lacks support for controlling the speed of execution and direct screen buffer access. The game has not been prepared to be heavily modified.

Similar problems are shared by *Half-Life 2* despite the fact that the Source engine is widely known for modding capabilities. It also lacks direct multiplayer support. Although the Source engine itself offers multiplayer support, it involves client-server architecture, which makes synchronization and direct interaction with the engine problematic (network com-

Table I  
OVERVIEW OF 3D FPS GAME ENGINES CONSIDERED.

Features / Game	Doom	Doom 3	Quake III: Arena	Half-Life 2	Unreal Tournament 2004	Unreal Tournament	Cube
Game Engine	ZDoom[1]	id tech 4	ioquake3	Source	Unreal Engine 2 2004	Unreal Engine 4 not yet	Cube Engine 2001
Release year	1993	2003	1999	2004	2004	not yet	2001
Open Source	✓	✓	✓			✓	✓
License	GPL	GPLv3	GPLv2	Proprietary	Proprietary	Custom	ZLIB
Language	C++	C++	C	C++	C++	C++	C++
DirectX		✓		✓		✓	
OpenGL	✓ <sup>3</sup>	✓	✓	✓	✓	✓	✓
Software Render	✓						
Windows	✓	✓	✓	✓	✓	✓	✓
Linux	✓	✓	✓	✓	✓	✓	✓
Mac OS	✓	✓	✓	✓	✓	✓	
Map editor	✓	✓	✓	✓	✓	✓	✓
Screen buffer access	✓	✓	✓			✓	✓
Scripting	✓	✓		✓	✓	✓	✓
Multiplayer mode	✓	✓	✓		✓	✓	✓
Small resolution	✓	✓	✓	✓	✓	✓	✓
Custom assets	✓	✓	✓	✓	✓	✓	✓
Free original assets						✓	✓
System requirements	Low	Medium	Low	Medium	Medium	High	Low
Disk space	40MB	2GB	70MB	4,5GB	6GB	>10GB	35MB
Code complexity	Medium	High	Medium	-	-	High	Low
Active community	✓	✓	✓	✓		✓	
Brand recognition	31.5		16.8	18.7	1.0		0.1

munication).

The client-server architecture was also one the reasons for rejection of Quake III: Arena. Quake III also does not offer any scripting capabilities, which are essential to make a research environment versatile. The rejection of Quake was a hard decision as it is a highly regarded and playable game even nowadays but this could not outweigh the lack of scripting support.

The latter problem does not concern Doom 3 but its high disk requirements were considered as a drawback. Doom 3 had to be ignored also because of its complexity, Windows-only tools, and OS-dependent rendering mechanisms. Although its source code has been released, its community is dispersed. As a result, there are several rarely updated versions of its sources.

The community activity is also a problem in the case of Cube as its last update was in August 2005. Nonetheless, the low complexity of its code and the highly intuitive map editor would make it a great choice if the engine was more popular.

Unreal Tournament, however popular, is not as recognizable as Doom or Quake but it has been a primary research platform for FPS games [9], [26]. It also has great capabilities. Despite its active community and the availability of the source code, it was rejected due to its high system requirements.

Doom (see Fig. 1) met most of the requirements and allowed to implement features that would be barely achievable in other

games, e.g., off-screen rendering and custom rewards. The game is highly recognizable and runs on the three major operating systems. It was also designed to work in  $320 \times 240$  resolution and despite the fact that modern implementations allow bigger resolutions, it still utilizes low-resolution textures. Moreover, its source code is easy-to-understand.

The unique feature of Doom is its software renderer. Because of that, it could be run without the desktop environment (e.g., remotely in a terminal) and accessing the screen buffer does not require transferring it from the graphics card.

Technically, ViZDoom is based on the modernized, open-source version of Doom's original engine — ZDoom, which is still actively supported and developed.

#### B. Application Programming Interface (API)

ViZDoom API is flexible and easy-to-use. It was designed with reinforcement and apprenticeship learning in mind, and therefore, it provides full control over the underlying Doom process. In particular, it allows retrieving the game's screen buffer and make actions that correspond to keyboard buttons (or their combinations) and mouse actions. Some game state variables such as the player's health or ammunition are available directly.

ViZDoom's API was written in C++. The API offers a myriad of configuration options such as control modes and rendering options. In addition to the C++ support, bindings for Python and Java have been provided. The Python API example is shown in Fig. 2.

<sup>3</sup>GZDoom, the ZDoom's fork, is OpenGL-based.



```

1 from vizdoom import *
2 from random import choice
3 from time import sleep, time
4
5 game = DoomGame()
6 game.load_config("../config/basic.cfg")
7 game.init()
8
9 # Sample actions. Entries correspond to buttons:
10 # MOVE_LEFT, MOVE_RIGHT, ATTACK
11 actions = [[True, False, False],
12            [False, True, False], [False, False, True]]
13 # Loop over 10 episodes.
14 for i in range(10):
15     game.new_episode()
16     while not game.is_episode_finished():
17         # Get the screen buffer and and game variables
18         s = game.get_state()
19         img = s.image_buffer
20         misc = s.game_variables
21         # Perform a random action:
22         action = choice(actions)
23         reward = game.make_action(action)
24         # Do something with the reward...
25
26 print("total reward:", game.get_total_reward())

```

Figure 2. Python API example

### C. Features

ViZDoom provides features that can be exploited in different kinds of AI experiments. The main features include different control modes, custom scenarios, access to the depth buffer and off-screen rendering eliminating the need of using a graphical interface.

1) *Control modes*: ViZDoom implements four control modes: i) synchronous player, ii) synchronous spectator, iii) asynchronous player, and iv) asynchronous spectator.

In asynchronous modes, the game runs at constant 35 frames per second and if the agent reacts too slowly, it can miss some frames. Conversely, if it makes a decision too quickly, it is blocked until the next frame arrives from the engine. Thus, for reinforcement learning research, more useful are the synchronous modes, in which the game engine waits for the decision maker. This way, the learning system can learn at its pace, and it is not limited by any temporal constraints.

Importantly, for experimental reproducibility and debugging purposes, the synchronous modes run deterministically.

In the player modes, it is the agent who makes actions during the game. In contrast, in the spectator modes, a human player is in control, and the agent only observes the player's actions.

In addition, ViZDoom provides an asynchronous multiplayer mode, which allows games involving up to eight players (human or bots) over a network.

2) *Scenarios*: One of the most important features of ViZDoom is the ability to run custom scenarios. This includes creating appropriate maps, programming the environment mechanics ("when and how things happen"), defining terminal conditions (e.g., "killing a certain monster", "getting to a certain place", "died"), and rewards (e.g., for "killing a monster",



Figure 3. ViZDoom allows depth buffer access.

"getting hurt", "picking up an object"). This mechanism opens endless experimentation possibilities. In particular, it allows creating a scenario of a difficulty which is on par with the capabilities of the assessed learning algorithms.

Creation of scenarios is possible thanks to easy-to-use software tools developed by the Doom community. The two recommended free tools include Doom Builder 2 and SLADE 3. Both are visual editors, which allow defining custom maps and coding the game mechanics in Action Code Script. They also enable to conveniently test a scenario without leaving the editor.

ViZDoom comes with a few predefined scenarios. Two of them are described in Section IV.

3) *Depth Buffer Access*: ViZDoom provides access to the renderer's depth buffer (see Fig. 3), which may help an agent to understand the received visual information. This feature gives an opportunity to test whether the learning algorithms can autonomously learn the whereabouts of the objects in the environment. The depth information can also be used to simulate the distance sensors common in mobile robots.

4) *Off-Screen Rendering and Frame Skipping*: To facilitate computationally heavy machine learning experiments, we equipped ViZDoom with off-screen rendering and frame skipping features. Off-screen rendering lessens the performance burden of actually showing the game on the screen and makes it possible to run the experiments on the servers (no graphical interface needed). Frame skipping, on the other hand, allows omitting rendering selected frames at all. Intuitively, an effective bot does not have to see every single frame. We explore this issue experimentally in Section IV.

### D. ViZDoom's Performance

The main factors affecting ViZDoom performance are the number of the actors (like items and bots), the rendering resolution, and computing the depth buffer. Fig. 4 shows how the number of frames per second depends on these factors. The tests have been made in the synchronous player mode on Linux running on Intel Core i7-4790k. ViZDoom uses only a single CPU core.

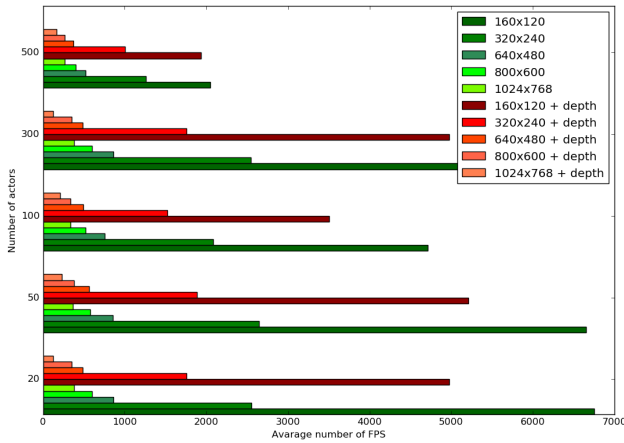


Figure 4. ViZDoom performance. “depth” means generating also the depth buffer.



Figure 5. The basic scenario

The performance test shows that ViZDoom can render nearly 7000 low-resolution frames per second. The rendering resolution proves to be the most important factor influencing the processing speed. In the case of low resolutions, the time needed to render one frame is negligible compared to the backpropagation time of any reasonably complex neural network.

#### IV. EXPERIMENTS

##### A. Basic Experiment

The primary purpose of the experiment was to show that reinforcement learning from the visual input is feasible in ViZDoom. Additionally, the experiment investigates how the number of skipped frames (see Section III-C4) influences the learning process.

1) *Scenario*: This simple scenario takes place in a rectangular chamber (see Fig. 5). An agent is spawned in the center of the room’s longer wall. A stationary monster is spawned at a random position along the opposite wall. The agent can strafe left and right, or shoot. A single hit is enough to kill

the monster. The episode ends when the monster is eliminated or after 300 frames, whatever comes first. The agent scores 101 points for killing the monster,  $-5$  for a missing shot, and, additionally,  $-1$  for each action. The scores motivate the learning agent to eliminate the monster as quickly as possible, preferably with a single shot<sup>4</sup>.

2) *Deep Q-Learning*: The learning procedure is similar to the Deep Q-Learning introduced for Atari 2600 [22]. The problem is modeled as a Markov Decision Process and Q-learning [28] is used to learn the policy. The action is selected by an  $\epsilon$ -greedy policy with linear  $\epsilon$  decay. The Q-function is approximated with a convolutional neural network, which is trained with Stochastic Gradient Decent. We also used experience replay but no target network freezing (see [22]).

##### 3) Experimental Setup:

a) *Neural Network Architecture*: The network used in the experiment consists of two convolutional layers with 32 square filters, 7 and 4 pixels wide, respectively (see Fig. 6). Each convolution layer is followed by a max-pooling layer with max pooling of size 2 and rectified linear units for activation [14]. Next, there is a fully-connected layer with 800 leaky rectified linear units [20] and an output layer with 8 linear units corresponding to the 8 combinations of the 3 available actions (left, right and shot).

b) *Game Settings*: A state was represented by the most recent frame, which was a  $60 \times 45$  3-channel RGB image. The number of skipped frames is controlled by the *skipcount* parameter. We experimented with skipcounts of 0-7, 10, 15, 20, 25, 30, 35 and 40. It is important to note that the agent repeats the last decision on the skipped frames.

c) *Learning Settings*: We arbitrarily set the discount factor  $\gamma = 0.99$ , learning rate  $\alpha = 0.01$ , replay memory capacity to 10 000 elements and mini-batch size to 40. The initial  $\epsilon = 1.0$  starts to decay after 100 000 learning steps, finishing the decay at  $\epsilon = 0.1$  at 200 000 learning steps.

Every agent learned for 600 000 steps, each one consisting of performing an action, observing a transition, and updating the network. To monitor the learning progress, 1000 testing episodes were played after each 5000 learning steps. Final controllers were evaluated on 10 000 episodes. The experiment was performed on Intel Core i7-4790k 4GHz with GeForce GTX 970, which handled the neural network.

4) *Results*: Figure 7 shows the learning dynamics for the selected skipcounts. It demonstrates that although all the agents improve over time, the skips influence the learning speed, its smoothness, as well as the final performance. When the agent does not skip any frames, the learning is the slowest. Generally, the larger the skipcount, the faster and smoother the learning is. We have also observed that the agents learning with higher skipcounts were less prone to irrational behaviors like staying idle or going the direction opposite to the monster, which results in lower variance on the plots. On the other hand, too large skipcounts make the agent ‘clumsy’ due to the

<sup>4</sup>See also [https://youtu.be/fKHw3wmT\\_uA](https://youtu.be/fKHw3wmT_uA)

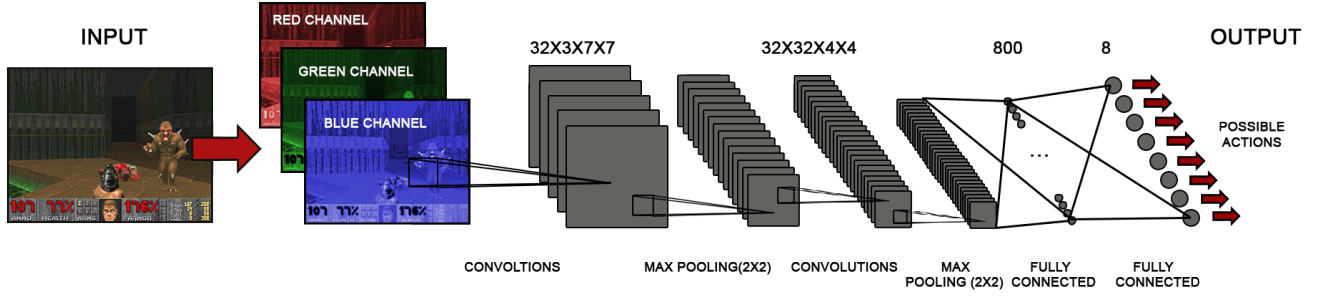


Figure 6. Architecture of the convolutional neural network used for the experiment.

lack of fine-grained control, which results in suboptimal final scores.

The detailed results, shown in Table ??, indicate that the optimal skipcount for this scenario is 4 (the “native” column). However, higher values (up to 10) are close to this maximum.

We have also checked how robust to skipcounts the agents are. For this purpose, we evaluated them using skipcounts different from ones they had been trained with. Most of the agents performed worse than with their “native” skipcounts. The least robust were the agents trained with skipcounts less than 4. Larger skipcounts resulted in more robust agents. Interestingly, for skipcounts greater than or equal to 30, the agents score better on skipcounts lower than the native ones. Our best agent that was trained with skipcount 4 was also the best when executed with skipcount 0.

It is also worth showing that increasing the skipcount influences the total learning time only slightly. The learning takes longer primarily due to the higher total overhead associated with episode restarts since higher skipcounts result in a greater number of episodes.

To sum up, the skipcounts in the range of 4-10 provide the best balance between the learning speed and the final performance. The results also indicate that it would be profitable to start learning with high skipcounts to exploit the steepest learning curve and gradually decrease it to fine-tune the performance.

### B. Medikit Collecting Experiment

The previous experiment was conducted on a simple scenario which was closer to a 2D arcade game rather than a true 3D virtual world. That is why we decided to test if similar deep reinforcement learning methods would work in a more involved scenario requiring substantial spatial reasoning.

1) *Scenario*: In this scenario, the agent is spawned in a random spot of a maze with an acid surface, which slowly, but constantly, takes away the agent’s life (see Fig. 8). To survive, the agent needs to collect medikits and avoid blue vials with poison. Items of both types appear in random places during the episode. The agent is allowed to move (forward/backward), and turn (left/right). It scores 1 point for each tick, and it is punished by  $-100$  points for dying. Thus,

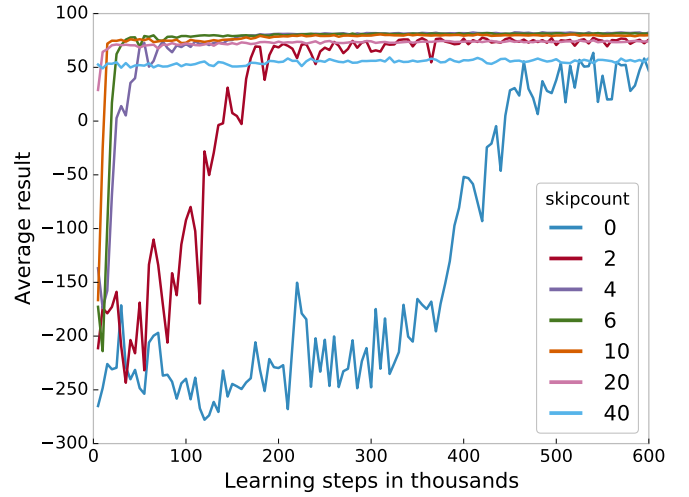


Figure 7. Learning dynamics depending on the number of skipped frames.

Table II  
AGENTS’ FINAL PERFORMANCE IN THE FUNCTION OF THE NUMBER OF SKIPPED FRAMES (‘NATIVE’). ALL THE AGENTS WERE ALSO TESTED FOR SKIPCOUNTS  $\in \{0, 10\}$ .

skipcount	average score $\pm$ stdev			episodes	learning time [min]
	native	0	10		
0	51.5 $\pm$ 74.9	51.5 $\pm$ 74.9	36.0 $\pm$ 103.6	6961	91.1
1	69.0 $\pm$ 34.2	69.2 $\pm$ 26.9	39.6 $\pm$ 93.9	29 378	93.1
2	76.2 $\pm$ 15.5	71.8 $\pm$ 18.1	47.9 $\pm$ 47.6	49 308	91.5
3	76.1 $\pm$ 14.6	75.1 $\pm$ 15.0	44.1 $\pm$ 85.4	65 871	93.4
4	<b>82.2 <math>\pm</math> 9.4</b>	<b>81.3 <math>\pm</math> 11.0</b>	76.5 $\pm$ 17.1	104 796	93.9
5	81.8 $\pm$ 10.2	79.0 $\pm$ 13.6	75.2 $\pm$ 19.9	119 217	92.5
6	81.5 $\pm$ 9.6	78.7 $\pm$ 14.8	76.3 $\pm$ 16.5	133 952	92
7	81.2 $\pm$ 9.7	77.6 $\pm$ 15.8	76.9 $\pm$ 17.9	143 833	95.2
10	80.1 $\pm$ 10.5	75.0 $\pm$ 17.6	<b>80.1 <math>\pm</math> 10.5</b>	171 070	92.8
15	74.6 $\pm$ 14.5	71.2 $\pm$ 16.0	73.5 $\pm$ 19.2	185 782	93.6
20	74.2 $\pm$ 15.0	73.3 $\pm$ 14.0	71.4 $\pm$ 20.7	240 956	94.8
25	73 $\pm$ 17	73.6 $\pm$ 15.5	71.4 $\pm$ 20.8	272 633	96.9
30	61.4 $\pm$ 31.9	69.7 $\pm$ 19.0	68.9 $\pm$ 24.2	265 978	95.7
35	60.2 $\pm$ 32.2	69.5 $\pm$ 16.6	65.7 $\pm$ 26.1	299 545	96.9
40	56.2 $\pm$ 39.7	68.4 $\pm$ 19.0	68.2 $\pm$ 22.8	308 602	98.6



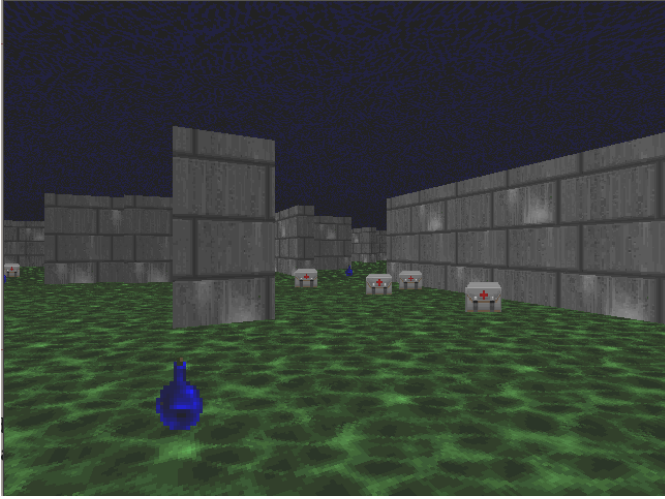


Figure 8. Health gathering scenario

it is motivated to survive as long as possible. To facilitate learning, we also introduced shaping rewards of 100 and  $-100$  points for collecting a medikit and a vial, respectively. The shaping rewards do not count to the final score but are used during the agent's training helping it to 'understand' its goal. Each episode ends after 2100 ticks (1 minute in real-time) or when the agent dies so 2100 is the maximum achievable score. Being idle results in scoring 284 points.

2) *Experimental Setup*: The learning procedure was the same as described in Section IV-A2 with the difference that for updating the weights RMSProp [25] this time.

a) *Neural Network Architecture*: The employed network is similar the one used in the previous experiment. The differences are as follows. It involves three convolutional layers with 32 square filters 7, 5, and 3 pixels wide, respectively. The fully-connected layer uses 1024 leaky rectified linear units and the output layer 16 linear units corresponding to each combination of the 4 available actions.

b) *Game Settings*: The game's state was represented by a  $120 \times 45$  3-channel RGB image, health points and the current tick number (within the episode). Additionally, a kind of memory was implemented by making the agent use 4 last states as the neural network's input. The nonvisual inputs (health, ammo) were fed directly to the first fully-connected layer. Skipcount of 10 was used.

c) *Learning Settings*: We set the discount factor  $\gamma = 1$ , learning rate  $\alpha = 0.00001$ , replay memory capacity to 10 000 elements and mini-batch size to 64. The initial  $\epsilon = 1.0$  started to decay after 4 000 learning steps, finishing the decay at  $\epsilon = 0.1$  at 104 000 episodes.

The agent was set to learn for 1000 000 steps. To monitor the learning progress, 200 testing episodes were played after each 5000 learning steps. The whole learning process, including the testing episodes, lasted 29 hours.

3) *Results*: The learning dynamics is shown in Fig. 9. It can be observed that the agents fairly quickly learns to get the perfect score from time to time. Its average score, however,

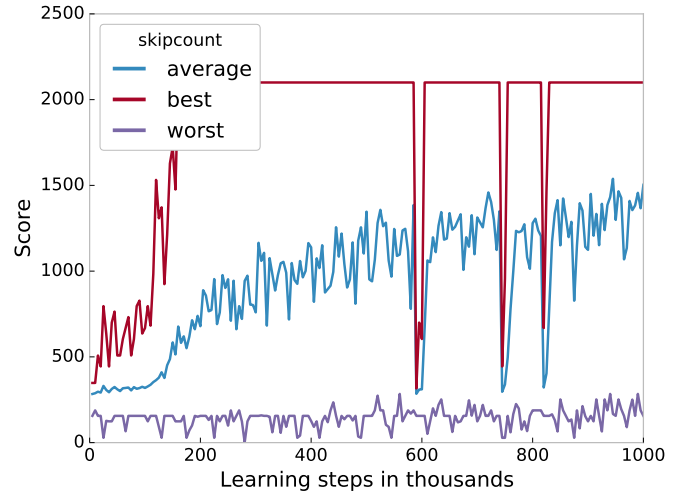


Figure 9. Learning dynamics for health gathering scenario.

improves slowly reaching 1300 at the end of the learning. The trend might, however, suggest that some improvement is still possible given more training time. The plots suggest that even at the end of learning, the agent for some initial states fails to live more than a random player.

It must, however, be noted that the scenario is not easy and even from a human player, it requires a lot of focus. It is so because the medikits are not abundant enough to allow the bots to waste much time.

Watching the agent play<sup>5</sup> revealed that it had developed a policy consistent with our expectations. It navigates towards medikits, actively, although not very deftly, avoids the poison vials, and does not push against walls and corners. It also backpedals after reaching a dead end or a poison vial. However, it very often hesitates about choosing a direction, which results in turning left and right alternately on the spot. This quirky behavior is the most probable, direct cause of not fully satisfactory performance.

Interestingly, the learning dynamics consists of three sudden but ephemeral drops in the average and best score. The reason for such dynamics is unknown and it requires further research.

## V. CONCLUSIONS

ViZDoom is a Doom-based platform for research in vision-based reinforcement learning. It is easy-to-use, highly flexible, multi-platform, lightweight, and efficient. In contrast to the other popular visual learning environments such as Atari 2600, ViZDoom provides a 3D, semi-realistic, first-person perspective virtual world. ViZDoom's API gives the user full control of the environment. Multiple modes of operation facilitate experimentation with different learning paradigms such as reinforcement learning, apprenticeship learning, learning by demonstration, and, even the 'ordinary', supervised learning. The strength and versatility of environment lie in

<sup>5</sup><https://www.youtube.com/watch?v=re6hkcTWWUY>

is customizability via the mechanism of scenarios, which can be conveniently programmed with open-source tools.

We also demonstrated that visual reinforcement learning is possible in the 3D virtual environment of ViZDoom by performing experiments with deep Q-learning on two scenarios. The results of the simple move-and-shoot scenario, indicate that the speed of the learning system highly depends on the number of frames the agent is allowed to skip during the learning. We have found out that it is profitable to skip from 4 to 10 frames. We used this knowledge in the second, more involved, scenario, in which the agent had to navigate through a hostile maze and collect some items and avoid the others. Although the agent was not able to find a perfect strategy, it learned to navigate the maze surprisingly well exhibiting evidence of a human-like behavior.

ViZDoom has recently reached a stable 1.0.1 version and has a potential to be extended in many interesting directions. First, we would like to implement a synchronous multiplayer mode, which would be convenient for self-learning in multiplayer settings. Second, bots are now deaf thus, we plan to allow bots to access the sound buffer. Lastly, interesting, supervised learning experiments (e.g., segmentation) could be conducted if ViZDoom automatically labeled objects in the scene.

#### ACKNOWLEDGMENT

This work has been supported in part by the Polish National Science Centre grant no. DEC-2013/09/D/ST6/03932. M. Kempka acknowledges the support of Ministry of Science and Higher Education grant no. 09/91/DSPB/0602.

#### REFERENCES

- [1] Zdoom wiki page. [http://zdoom.org/wiki/Main\\_Page](http://zdoom.org/wiki/Main_Page). Accessed: 2016-01-30.
- [2] David Abel, Alekh Agarwal, Fernando Diaz, Akshay Krishnamurthy, and Robert E. Schapire. Exploratory gradient boosting for reinforcement learning in complex domains. *CoRR*, abs/1603.04119, 2016.
- [3] Minoru Asada, Shoichi Noda, Sukoya Tawaratsumida, and Koh Hosoda. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. In *Recent Advances in Robot Learning*, pages 163–187. Springer, 1996.
- [4] Minoru Asada, Eiji Uchibe, Shoichi Noda, Sukoya Tawaratsumida, and Koh Hosoda. A vision-based reinforcement learning for coordination of soccer playing behaviors. In *Proceedings of AAIL-94 Workshop on AI and A-life and Entertainment*, pages 16–21, 1994.
- [5] Nicholas Cole, Sushil J Louis, and Chris Miles. Using a genetic algorithm to tune first-person shooter bots. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 1, pages 139–145. IEEE, 2004.
- [6] Giuseppe Cuccu, Matthew Luciw, Jürgen Schmidhuber, and Faustino Gomez. Intrinsically motivated neuroevolution for vision-based reinforcement learning. In *Development and Learning (ICDL), 2011 IEEE International Conference on*, volume 2, pages 1–7. IEEE, 2011.
- [7] Mark Dawes and Richard Hall. Towards using first-person shooter computer games as an artificial intelligence testbed. In *Knowledge-Based Intelligent Information and Engineering Systems*, pages 276–282. Springer, 2005.
- [8] Abdenour El Rhalibi and Madjid Merabti. A hybrid fuzzy ANN system for agent adaptation in a first person shooter. *International Journal of Computer Games Technology*, 2008, 2008.
- [9] A I Esparcia-Alcazar, A Martinez-Garcia, A Mora, J J Merelo, and P Garcia-Sanchez. Controlling bots in a First Person Shooter game using genetic algorithms. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8, jul 2010.
- [10] Chris Gaskett, Luke Fletcher, and Alexander Zelinsky. Reinforcement learning for a vision based mobile robot. In *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, volume 1, pages 403–409. IEEE, 2000.
- [11] Benjamin Geisler. *An empirical study of machine learning algorithms applied to modeling player behavior in a first person shooter video game*. PhD thesis, University of Wisconsin-Madison, 2002.
- [12] F G Glavin and M G Madden. DRE-Bot: A hierarchical First Person Shooter bot using multiple Sarsa( $\lambda$ ) reinforcement learners. In *Computer Games (CGAMES), 2012 17th International Conference on*, pages 148–152, jul 2012.
- [13] F G Glavin and M G Madden. Adaptive Shooting for Bots in First Person Shooter Games Using Reinforcement Learning. *Computational Intelligence and AI in Games, IEEE Transactions on*, 7(2):180–192, jun 2015.
- [14] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey J. Gordon and David B. Dunson, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, volume 15, pages 315–323. Journal of Machine Learning Research - Workshop and Conference Proceedings, 2011.
- [15] S Hladky and V Bulitko. An evaluation of models for predicting opponent positions in first-person shooter video games. In *Computational Intelligence and Games, 2008. CIG '08. IEEE Symposium On*, pages 39–46, dec 2008.
- [16] Igor V. Karpov, Jacob Schrum, and Risto Miikkulainen. *Believable Bot Navigation via Playback of Human Traces*, pages 151–170. Springer Berlin Heidelberg, 2012.
- [17] Jan Koutník, Jürgen Schmidhuber, and Faustino Gomez. Evolving deep unsupervised convolutional networks for vision-based reinforcement learning. In *Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 541–548. ACM, 2014.
- [18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.
- [19] Sascha Lange and Martin Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *IJCNN*, pages 1–8, 2010.
- [20] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning (ICML)*, 2013.
- [21] M McPartland and M Gallagher. Reinforcement Learning in First Person Shooter Games. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(1):43–56, mar 2011.
- [22] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharmash Kumar, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015.
- [23] Megan Smith, Stephen Lee-Urban, and Héctor Muñoz-Avila. RETALIATE: learning winning policies in first-person shooter games. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 1801. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.
- [24] Tony C Smith and Jonathan Miles. Continuous and Reinforcement Learning Methods for First-Person Shooter Games. *Journal on Computing (JoC)*, 1(1), 2014.
- [25] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSE: Neural Networks for Machine Learning, 2012.
- [26] Chang Kee Tong, Ong Jia Hui, J Teo, and Chin Kim On. The Evolution of Gamebots for 3D First Person Shooter (FPS). In *Bio-Inspired Computing: Theories and Applications (BIC-TA), 2011 Sixth International Conference on*, pages 21–26, sep 2011.
- [27] David Trenholme and Shamus P Smith. Computer game engines for developing first-person virtual environments. *Virtual reality*, 12(3):181–187, 2008.
- [28] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

# Artefacts: Minecraft meets Collaborative Interactive Evolution

Cristinel Patrascu

Center for Computer Games Research  
IT University of Copenhagen  
Copenhagen, Denmark  
Email: patrascu.cristinel@gmail.com

Sebastian Risi

Center for Computer Games Research  
IT University of Copenhagen  
Copenhagen, Denmark  
Email: sebr@itu.dk

**Abstract**—Procedural content generation has shown promise in a variety of different games. In this paper we introduce a new kind of game, called *Artefacts*, that combines a sandbox-like environment akin to Minecraft with the ability to interactively evolve unique three-dimensional building blocks. *Artefacts* does not only allow players to collaborate by building larger structures from evolved objects but also to continue evolution of others' artefacts. Results from playtests on three different game iterations indicate that players generally enjoy playing the game and are able to discover a wide variety of different 3D objects. Moreover, while there is no explicit goal in *Artefacts*, the sandbox environment together with the ability to evolve unique shapes does allow for some interesting gameplay to emerge.

## I. INTRODUCTION

In recent years there has been a growing interest in *procedural content generation* (PCG). This field includes algorithms and methods for generating a wide variety of different types of content (e.g. levels, three-dimensional objects, textures, stories, 3D caves etc.) that can be part of the virtual world of a video game [4, 5, 8, 10, 12, 19, 25]. One advantage of automatically generating game content is the reduced amount of work required by artists and game designers. Besides production cost reduction, games have also benefited from the novel gameplay emerging from PCG techniques [19]. Additionally, PCG can increase a game's replay value because content is constantly updated and varied throughout different play sessions.

A main inspiration for the game presented here is Minecraft<sup>1</sup>, which is a sandbox video game that allows players to build three-dimensional structures together with others from a selection of predefined cubes made out of different materials (e.g. stone, wood). Minecraft encourages players to play creatively by giving them a variety of different ways to play the game. While the cubes are predefined, Minecraft does employ a PCG-based approach to generate the 3D worlds for the players to explore.

In the new game presented here, called *Artefacts*, players can collaboratively build 3D structures in a sandbox environment similarly to Minecraft. However, in contrast to Minecraft, in which players only have a predefined number of cubes to choose from that all have about the same shape, *Artefacts* allows



Fig. 1. **Artefacts - The Video Game.** Players in *Artefacts* can collaboratively evolve unique 3D objects in an open physics sandbox and combine them to build larger structures.

players to create an unlimited variety of differently shaped 3D building blocks through an evolutionary computation (EC) approach. EC methods in particular have proven effective at automatically generating diverse content for games such as weapons in Galactic Arms Race (GAR [7]), levels for a competitive multiplayer FPS game [13], flowers in the social video game Petalz [16], or even complete games [5, 24].

The 3D objects in *Artefacts* are genetically encoded by a special kind of neural network called a *compositional pattern producing network* (CPPN; [3, 20]). The generative CPPN encoding enables players to breed an unlimited variety of different 3D objects with regularities such as symmetry or repetition. Importantly, the NEAT algorithm [21], which evolves the CPPNs in this paper, allows the 3D objects to become increasingly complex and more intricate over generations.

Players in *Artefacts* can guide evolution by choosing from a set of artefact seeds that spawn around a planted object. Importantly, players can collaborate in the breeding process by picking up seeds produced by others and continuing evolution from there. Moreover, players are able to manipulate

<sup>1</sup>Copyright (c) 2011 Mojang



the placement of objects in three-dimensional space and can express their creativity by building a wide variety of different structures using the evolved artefacts.

There are no explicit goals in Artefacts. The game was designed to encourage players to explore and to play with the ability to evolve and build with 3D objects, which means that players can use the artefacts in any way they see fit. For example, players can focus on building tall structures or on destroying other peoples' structures.

To investigate what type of new game affordances Artefacts offers, both quantitative and qualitative data from a series of playtests were collected and analyzed. The results from the initial playtests suggest that, while still in an early stage, the novel combination of evolved 3D objects in an open world is a promising game concept that offers many potential directions to expand upon.

## II. BACKGROUND

This section first discusses existing work combining PCG with video games and concludes by reviewing the technical building blocks of the PCG algorithm employed in Artefacts.

### A. Procedural Content Generation

When applied to games, PCG allows game elements (e.g. maps, textures, items, quests, etc.) to be generated algorithmically rather than through direct human design [5, 8, 25]. For example, the popular Diablo series<sup>2</sup> features procedurally generated dungeons that players explore as a central focus of the game. Like Diablo, many other PCG approaches similarly rely on a fixed set of parameters and randomness to generate content within a heavily constrained space of possibilities. However, a recent focus is to apply artificial intelligence approaches to enable more open-ended generation of PCG.

In particular, evolutionary computation and other search-based approaches [25] can limit the need for hand-designed rules, and may thus further save on PCG development costs. More interestingly, it also enables design of new content outside the scope of a fixed space of rules. One popular technique is interactive evolutionary computation (IEC [23]), in which the user in effect guides an evolutionary algorithm. An example of IEC applied to video games is provided by NeuroEvolving Robotic Operatives (NERO [22]), in which players guide the evolution of a team of fighting robots. In another example, called Galactic Arms Race (GAR [7]), weapons are evolved automatically based on user behavior, and in the social Petalz video game, players can evolve an unlimited variety of different flowers [16]. Further examples include Avery et al. [1], who evolved several aspects of a tower defense game, Shaker et al. [18] who evolved levels for the platform game Super Mario Bros, Olsted et al. [13] who interactively evolved levels for a competitive multiplayer FPS game, and Togelius and Schmidhuber [24], who experimented with evolving the rules of the game itself.

The particular evolutionary representation that is applied to represent evolved 3D objects in Artefacts, is reviewed next.

<sup>2</sup>Copyright Blizzard Entertainment, <http://blizzard.com/>

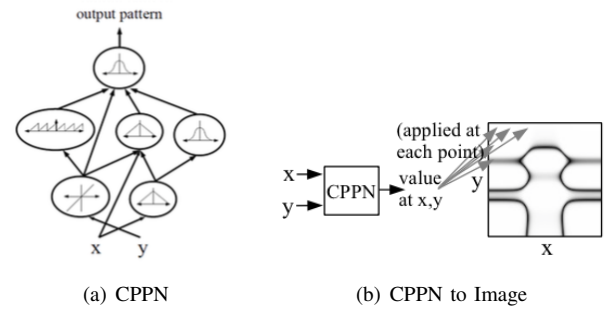


Fig. 2. **Compositional Pattern Producing Networks for 2D Images.** (a) CPPNs can use a variety of different functions like sigmoids, Gaussian, sine and many others in contrast to more traditional ANNs with sigmoid activation functions (b) The CPPN example in this figure inputs two arguments  $x$  and  $y$  that are interpreted as coordinates in two-dimensional space. Applying the CPPN to all the coordinates and drawing them with an ink intensity determined by its output results in a two-dimensional image.

### B. Compositional Pattern Producing Networks (CPPNs)

The 3D objects in Artefacts are generated by a variation of artificial neural networks (ANNs), called *compositional pattern producing networks* (CPPNs [20]), which differ in their set of activation functions and how they are applied. While ANNs often contain only sigmoid or Gaussian activation functions, CPPNs can include both such functions and many others. The choice of CPPN functions can be biased toward specific patterns or regularities. Additionally, unlike typical ANNs, CPPNs are usually queried across a space of possible input patterns to represent a complete image or pattern. Specifically, CPPNs produce a phenotype that is a function of  $n$  dimensions, where  $n$  is the number of dimensions in physical space. For each coordinate in that space, its level of expression is an output of the function that encodes the phenotype. Figure 2 shows how a two-dimensional phenotype can be generated by a function of two parameters that is represented by a network of composed functions. CPPNs in effect encode patterns at infinite resolution and can be sampled at whatever resolution is desired.

Successful CPPN-based applications include Picbreeder [17], MaestroGenesis [9], EndlessForms [3], the Galactic Arms Race (GAR) video game [7], folded wire robots [15], and virtual soft-body robots [2]. Clune and Lipson [3] introduced a modification to the general CPPN representation to produce 3D objects, which is the basis for the object representation in Artefacts. It is described in detail in Section III-B.

### C. Neuroevolution of Augmenting Topologies (NEAT)

Because CPPNs are ANNs, they can be evolved with the *Neuroevolution of Augmenting Topologies* (NEAT) algorithm [21], which is the standard neuroevolution algorithm for such purposes [6, 17, 20]. Neuroevolution in general has shown promise in a variety of different games [14].

NEAT begins with a population of simple neural networks or CPPNs and then *adds complexity* over generations by adding new nodes and connections through mutations. Novel topologies gradually accumulate, thereby allowing diverse and complex phenotype patterns to be represented. No limit

is placed on the size to which topologies can grow. New structures are introduced incrementally as structural mutations occur, and only those structures survive that are found to be useful (traditionally through fitness evaluations and through player selection in this paper). In effect, then, NEAT searches for a compact, appropriate topology by incrementally increasing the complexity of existing structure. A complete overview of NEAT can be found in Stanley and Miikkulainen [21]. For evolving content, complexification means that content (e.g. 3D objects in Artefacts) can become more elaborate and intricate over generations.

### III. ARTEFACTS – THE VIDEO GAME

Artefacts (publicly available at <https://cristi.itch.io/artefacts>) has been designed as an open world in which players can explore and interact with evolving objects. An important aspect of the game is the social multiplayer component, which allows players to collaborate in the evolution of the 3D objects but also – similarly to Minecraft – in the construction of larger structures. In other words, Artefacts is a construction game with a potentially infinite number and variety of resources. Players experience the game through a first-person perspective and can perform the standard first-person actions such as walking, running and jumping. The game aims to create an immersive experience in which the players feel as being part of the world they are creating.

#### A. Development and Multiplayer Framework

Players can easily host their own multiplayer games and play together with others in the same virtual space. The game and its multiplayer component were implemented using the Unity game engine<sup>3</sup> and its built-in networking framework. The CPPN implementation is based on UnityNEAT<sup>4</sup>, which is a port of the C# implementation of NEAT, called SharpNEAT<sup>5</sup>.

#### B. Generating 3D Artefact Objects

The algorithm to generate the 3D artefacts is based on the CPPN object representation introduced by Clune and Lipson [3]. Instead of CPPNs with two inputs that can generate two-dimensional images (Figure 2), CPPNs to generate 3D objects have three inputs  $x$ ,  $y$ , and  $z$ . The algorithm works by (1) inputting the coordinates of each point  $p$  (e.g.  $x=1, y=3, z=2$ ), of a three-dimensional voxel volume (e.g. a grid of  $5 \times 5 \times 5$  voxels) into the CPPN, (2) activating the network, and (3) determining if the voxel at that particular position  $p$  should be filled if the CPPN output is higher than some threshold, or empty otherwise. The coordinate input values are normalized within the  $[-1, 1]$  range before being passed into the CPPN.

The voxel array outputted by the CPPN is processed by the Marching Cubes algorithm [11], which generates a 3D mesh representation that can be easily rendered by common graphics APIs. After the polygonal surface is determined, the algorithm calculates the normal for each of the vertices.

<sup>3</sup><https://unity3d.com/>

<sup>4</sup><https://github.com/lordjesus/UnityNEAT>

<sup>5</sup><http://sharpneat.sourceforge.net/>

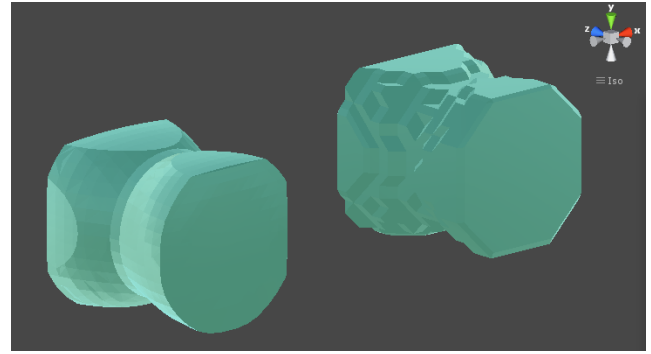
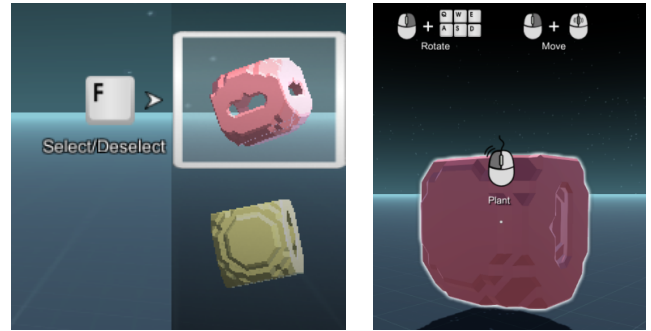


Fig. 3. **Artefact Generation.** 3D objects created with the representation introduced by Clune and Lipson [3] and with the “blockier” Artefacts modification (right).



(a) Inventory Selection

(b) Planting and Positioning

Fig. 4. **User Interface.** (a) Players can store seeds in their inventory and plant them anywhere in the virtual world. (b) Players can also position and rotate the artefacts before they are planted.

The voxel volume size in Artefacts, which is set to  $16 \times 16 \times 16$  units, tries to strike a balance between the level of detail of the generated 3D meshes and the time required by the Marching Cubes algorithm to create the mesh. As a bias towards rounded objects, the distance from the center of the workspace volume is given as an additional input to the CPPN.

In contrast to the approach by Clune and Lipson [3], the CPPN representation in this paper is slightly modified to create meshes with sharper edges that give the artefacts a “blockier” aesthetic. The CPPN output values are processed in the following way: (1) During the calculation of the output value for each coordinate, the algorithm keeps track of the minimum  $min$  and maximum  $max$  produced values. (2) The central value  $c$  between the minimum and maximum is calculated. (3) For each position  $p$ , a voxel is created if CPPN output  $m \geq c$ . In addition to the 3D mesh, the CPPN also determines RGB color values for each artefact through three additional outputs. Figure 3 shows an example of objects generated with the original representation (left) and the modified min/max representation (right).

#### C. Game Mechanics Overview

While exploring their environment, players can find and interact with artefacts of different shapes and colors evolved by themselves and other players. The user interface was created with the goal of making each available player action as intuitive as possible. Players have an inventory, which allows

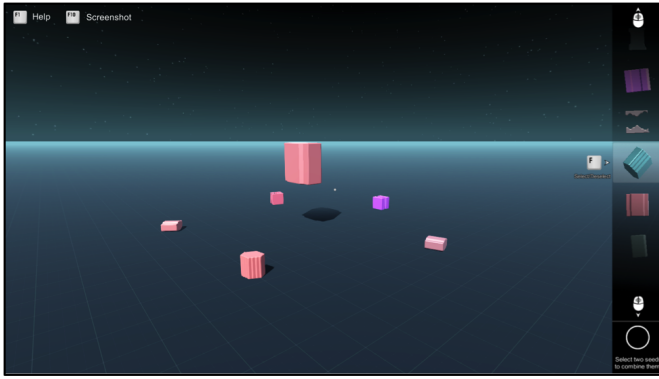


Fig. 5. **3D Artefact and Offspring.** When an artefact is planted it produces five smaller seed artefacts through mutation, which can be picked up by the players. Once planted, the seed produces a full-sized artefact. Mutations on the parent CPPN (e.g. adding new nodes and connections or changing the weight of a connection) create offspring that, while resembling the parent, vary in different ways. By selecting which seeds they prefer, players can guide evolution based on their personal taste.

them to store collected artefact seeds. They can select seeds by scrolling through them (Figure 4a) and plant selected seeds in the virtual world (Figure 4b). Planting a seed produces a full-sized artefact and spawns five offspring seeds surrounding the artefact (Figure 5). These offspring seeds are created by mutating the parent artefact, and while similar to their parents, they can vary in interesting ways. The created seeds can be picked up and planted by others, allowing multiple players to collaboratively influence the lineage of an evolving artefact. It is also possible to select two seeds from the inventory, thereby performing a crossover between them.

While planting artefacts, players have precise control over their position and rotation. By holding down the right mouse button, artefacts can be rotated about different axes through the keyboard, with a rotation speed of 100 degrees per second. Players can also move around while holding the artefact to position it anywhere in the virtual world. Once an artefact has been planted, it can be picked up and repositioned by other players. Players can also take screenshots of their creations, from which some are shown in the next section.

#### IV. PLAYTESTS AND ITERATIVE DEVELOPMENT

While developing Artefacts, an iterative development approach was chosen. New features were added progressively, tested and evaluated based on player questionnaires. Especially the user interface (UI) went through many iterations. Besides the UI, the controls to interact with the artefacts also changed significantly together with the way different artefacts physically interacted with each other. In the following sections we present the three game iterations in chronological order together with the results of the player questionnaires. Participants were not given concrete instructions on how to play and were only encouraged to explore the game's affordances.

##### A. Experimental Parameters

The available CPNN activation functions were Linear, Bipolar sigmoid, Gaussian and Sine, all with equal probability of being added. Offspring had a 45% probability of weight

TABLE I  
RESULTS OF FIRST ITERATION MULTIPLAYER TEST

Total number of players	7
Number of sessions	4
Number of players in each session	5
Average duration per session (in minutes)	16
Average # artefacts planted per session	158
Average # mutations per session	113
Average # crossovers per session	45
Average # seeds picked up per session	227
Average # player contributing per artefact	2.3
Max # player contributing per artefact	6
Average # artefacts planted per players	29
Total # artefacts planted	633
Total # of spawned seeds	$633 * 5 = 3,165$
Total # of collected seeds	908
Max generation	54
Total # of mutations	453
Total # of crossovers	180

mutation, 20% chance of node addition, 20% of adding a new connection, and a 15% probability of deleting a connection. The mutation probabilities were set to relatively high values to ensure that players see fast evolutionary progress while still producing offspring that resembles the parent artefacts.

##### B. First Version

Seven people participated in the playtest of the first game version on site at the IT University of Copenhagen. However, due to technical limitations, only five players could play the game at the same time. In the first iteration of the game, all artefacts were controlled by rigid-body physics, i.e. they were affected by gravity and could collide with each other. The testers played for approximately one hour (divided into four separate sessions with five players each) and filled out a questionnaire afterwards.

A summary of the results is shown in Table I. Players planted a total of 633 artefacts and collected 908 seeds. Not surprisingly, players seemed to plant more artefacts in less time as they got accustomed to the game mechanics and user interface. Figures 6a,b show some of the evolved artefacts, which come in a variety of shapes and colors, and a tall structure that was built by multiple players.

Picking up seeds evolved by others allowed users to continue evolution and collaborate on the design of other players. Up to six players contributed to the lineages of some artefacts<sup>6</sup>, with 2.3 players contributing on average per artefact. This suggests that the multiplayer component of the game allows meaningful interactions to emerge between players and the artefacts they create. Of the 3,165 spawned seeds, 908 were picked up by players, which is roughly 28%. The reason that players did not pick up every seed is likely due to the fact that (1) some of the produced offspring look similar to each other, and (2) players decide whether or not to pick up seeds based on their aesthetic preferences.

Interestingly, the placement of artefacts in the virtual world appears to form one or more clusters (i.e. a large number of

<sup>6</sup>While only five participants could play at the same time, players that left the game made room for others to join, making lineage contributions of more than five players possible.



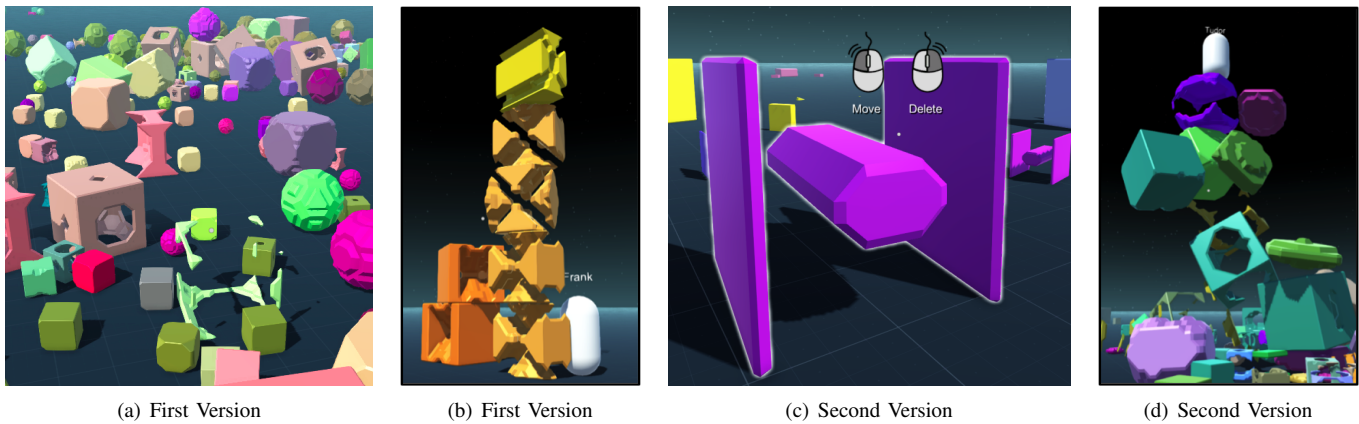


Fig. 6. **Artefacts Evolved by Players During the First and Second Version.** The CPPN-based representation allowed players to evolve a variety of different 3D objects (a). Players also tried to build taller structures together in the first version of the game (b), which proved quite difficult because it was not possible to permanently combine two artefacts. In the second version, players were able to glue artefacts together, thereby allowing the construction of a wider variety of different structures (c, d).

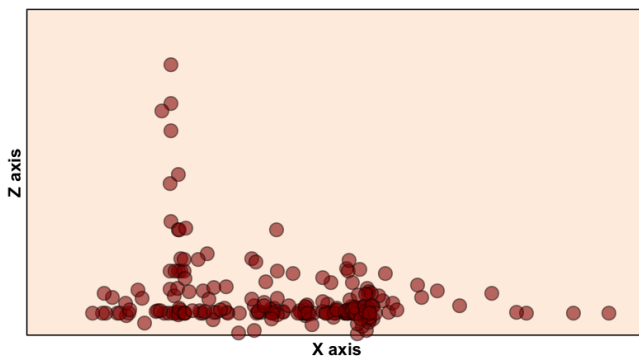


Fig. 7. **Artefact Placement.** The placement of the artefacts and observations during the playtest suggest that players often build structures together, thereby creating clusters of objects in the virtual world.

artefacts in a close distance from each other) and a number of artefacts spread across different directions (Figure 7). These results and observations from the playtest indicate that players often build structures with others or in their vicinity, instead of building structures by themselves in isolation.

1) *Questionnaire Results:* To analyze the players' subjective experience, they were asked to fill out a questionnaire after playing. A total of seven questionnaires were filled out. To characterize patterns in the players' responses, they were labeled with tags and then aggregated tags were created consisting of several related ones. The following is a list of tags for each of the answered questions from the first playtest:

- Most interesting part: interaction with other players (2), creating unique shapes (4), physical interactions (2), combining artefacts (1)
- Least interesting part: hard to build large structures (2), buggy interface (1), lack of more gameplay elements (2), structures getting destroyed by other players (1), interaction with other players (1)
- Could be better: having a way to easily connect (glue) artefacts (2), players flying around (1), buggy interface (1), more physical rules (3), more shapes in the beginning

(1), having some predefined goals (1)

- About evolving artefacts: difficult to predict (1), should have more initial variation (1), intuitive (2), player felt curious (3), breeding seeds should be more visible (1)
- More interesting in multiplayer than it would be in single-player: yes (6), the same (1)
- About combining (breeding) artefact seeds: outcome was sometimes unexpected (2), interesting outcome (2), fun/cool (2), could be more elaborate (1)
- About building structures: difficult (3), objects are too light (1), not so intuitive (1), more building controls (1), physics was a limiting factor (1), could be better with different sized artefacts (1)

The questionnaire answers and observations during the playtest indicate that players enjoyed (1) creating unique artefacts, (2) the physical interactions between artefacts and (3) the element of an open world, in which one can play together with others. Most players thought that the process of planting seeds was intuitive and clear. However, some players found it difficult to understand how combining seeds worked while others reported that it was difficult to predict the result of mutations and crossover. Additionally, some players would have preferred more variation in the seeds that are initially created to populate the world.

While the ability to create unique artefacts in a physics sandbox allowed some emergent gameplay (e.g. building the tallest structure), composing more complex structures proved challenging; objects would tend to easily knock each other down while the players were trying to place them next to each other.

For example, building the stacked structure in Figure 6b proved to be a very difficult task because players needed to place artefacts with extreme precision for the structure not to collapse. Players had to create an additional supporting structure that allowed them to climb high enough to place more artefacts on top of the already existing structure. Furthermore, some players were frustrated by the fact that anyone can interact and therefore destroy someone's constructions.

### C. Second Version

The first playtest provided valuable information about the players' experience and potential ways to improve it. In addition to minor bugfixes and interface improvements, more variation was added to the seeds initially present in the world by randomly evolving them for 10 up to 20 generations. Players now also had the ability to delete seeds and artefacts. The biggest change from the first version of the game was the added ability to attach or "glue" artefacts together by placing them so close to each other that they touch each other's bounding boxes. These modifications aimed to make it easier for the players to combine artefacts into larger and more complex structures.

Four participants that were new to the game took part in the second playtest. Because it focused on testing more specific game adjustments, performing two sessions (lasting 13 and 10 minutes) was deemed sufficient. Players evolved a total of 190 artefacts, 42% of those through crossover. Figure 6c,d show examples of structures built during the second user test: a large tower built by multiple players and a structure resembling a spaceship. While the new game modifications made it easier to build tall structures, the artefacts could still collide with each other, making it difficult to place them precisely next to each other.

1) *Questionnaire Results:* After playing the game, the participants were again asked to answer a questionnaire about their experience:

- Most interesting part: manipulating the evolution of artefacts (1), playing with other people (1), building structures (1), attaching artefacts to each other (1), variety of shapes (1)
- Least interesting part: attaching shapes was buggy (1), lack of more gameplay elements (1), artefacts do not evolve significantly enough (1), the flat plane environment (1)
- Could be better: hard to figure out how to combine seeds (1), more varied and complex shapes (4), attaching artefacts (1)
- About evolving artefacts: selection was counter-intuitive (1), felt repetitive (1), intuitive (1), artefacts look too much like boxes (1), player felt curious (1)
- More interesting in multiplayer than it would be in single-player: yes (3), equally interesting (1)
- About combining (breeding) artefact seeds: there should be more control over the outcome (2), some repeating archetypes (1), outcome could be more varied and complex (2)
- About building structures: many glitches (3), fun/cool (2), difficult (1), could be more interesting by having objects of different durability (1)

The answers from the questionnaire and observations during the playtest suggest that players enjoyed breeding artefacts and trying to control the direction in which they evolved. However, they felt that there could be more variation in the created artefacts. While the CPPN representation can produce different

3D objects, as shown in Figure 6, the volume in which the artefacts are generated in is always cube-shaped, resulting in many artefacts with flat sides that do not vary much in size. In the future it will be interesting to experiment with different 3D object encodings, allowing players to scale the artefacts, or to control the shape of the volume used to generate them.

In comparison to the first game iteration, the new modifications did in fact facilitate the construction of more complex structures. Players found it easy to attach artefacts together and to build on top of them. However, some issues remained that should be addressed to further enhance the experience of building structures. First of all, it was hard to align artefacts precisely with each other; artefacts move based on physical forces and synchronizing these physical simulations over the network was challenging. As a result, the artefacts could sometimes end up in a state in which the client-side objects failed to keep up with the server-side objects. Secondly, fitting artefacts together sometimes proved difficult; some had very different forms, not exactly fitting next to each other like the pre-made building blocks in games like Minecraft. Additionally, due to computational constraints, convex colliders were used on shapes that were concave, which meant that the colliders did often not match the exact shape of the object.

While some issues remained, the results from the second playtest suggested that the changes made after the first test did improve the players' experience. It also provided valuable information on how to further enhance the game experience.

### D. Third Version

The final playtest took place online instead of in a physical location. We allowed players to create their own servers that other players could join to play together. The game was made publicly available and was advertised for approximately two weeks. In that period the web page was visited 372 times, while the game was downloaded 35 times. However, only eight people that downloaded the game generated enough data for any analysis.

Based on the results of the first two playtests, physical interactions between artefacts were disabled (i.e. they could now intersect) and the artefacts themselves were not affected by gravity anymore. While the previous iterations showed that physical interactions between objects can allow for some interesting gameplay to emerge, the new modifications aimed to make it easier for players to build larger and more organic looking structures since the artefacts could now overlap. Additionally, the controls for placing the artefacts were fine-tuned, allowing for more precision and control.

A summary of the results of the final playtest are shown in Table II. Because of the small number of players for all eight games there was only one person playing the game at a time. The collected results suggest that players of the third game version found it much easier to control the placement of artefacts. Additionally, players were able to build structures faster than before, without spending too much time trying to work around the physical constraints of the previous versions. As Figures 8 and 1 show, players were able to more easily

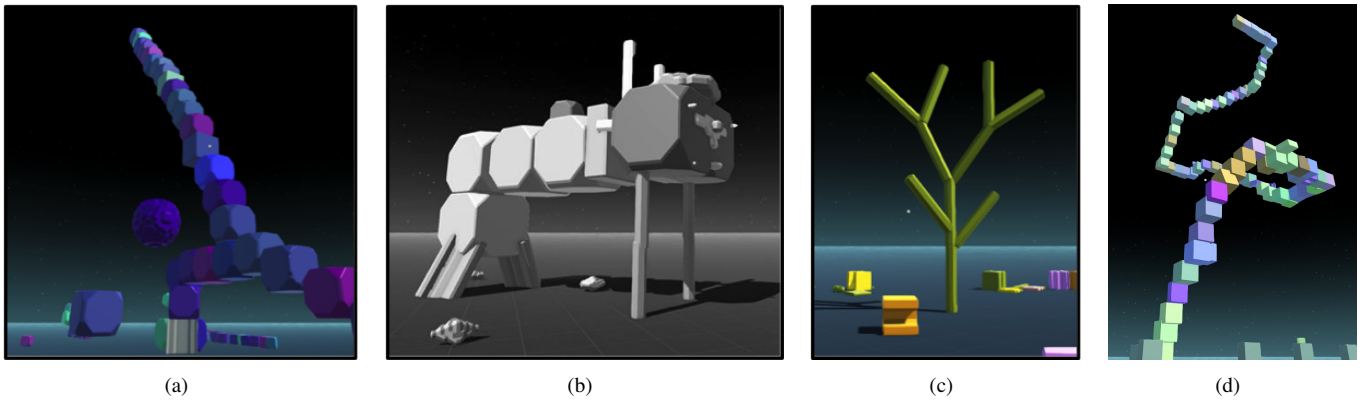


Fig. 8. **Artefacts Evolved in the Third Game Version.** By disabling physics and collisions between artefacts, players were able to more easily build a variety of different structures. Players bred a variety of 3D objects (e.g. long sticks, blocks) that allowed them to build different types of structures such as a tree (c) or a staircase (a).

TABLE II  
RESULTS OF THIRD ITERATION MULTIPLAYER TEST

Total number of players	8
Total number of artefacts planted	95
Max generation	51
Total number of mutations	90
Total number of crossovers	5
Average game duration (in minutes)	5
Average number of artefacts planted per player	12
Average number of mutations per player	11
Average number of crossovers per player	1
Average number of seeds picked up per player	16

build a variety of different structures such as a staircase, a robot, and a tree-like artefact. While disabling the physical interactions between artefacts might prevent some of the earlier emergent gameplay, it did allow players to have more creative freedom over the structures they were building. A video of some gameplay footage from the third version can be found at the project page: <https://cristi.itch.io/artefacts>.

While only a small number of people downloaded the game, the feedback received from the ones that did was mostly positive. Below are a number of quotes received from the players: “Fun, made a giant spiral staircase”, “Nice concept, keep it up!”, “Nice interface, easy to use. I quickly got frustrated trying to place objects together accurately. You may consider adding a “snap” so objects are flush against each other. Overall a nice sandbox, waiting to see how you expand on this”.

While the playtests would have ideally included a larger number of participants, even the tests with few players suggest that it is possible to create interesting and novel gameplay by evolving 3D objects in a sandbox video game.

## V. DISCUSSION AND FUTURE WORK

This paper presented a novel PCG-based game, which allows players to evolve 3D objects and use those objects to build larger physical structures. The results indicate that players enjoyed creating unique objects and were curious about the process of evolving them in an open world environment shared with others. The novel game mechanics in Artefacts allowed for some emergent gameplay, with players building structures

individually and collaboratively. Because players share the same physical space, they were able to collaboratively evolve artefacts and extend the lineages of artefacts evolved by others.

While the game concept shows promise, we imagine a variety of further studies and improvements that would make it more engaging in the future. Since our playtests were performed with a rather small number of players, an important next step is a larger multiplayer experiment. What type of objects could be evolved by thousands of players collaborating and what type of physical structures could they build? An important question in this context is if a game like Artefacts could allow players to express their creativity in ways similar to a game such as Minecraft. A step towards answering these questions is the creation of a dedicated Artefacts server that enables a persistent virtual world, allowing many players to join at the same time.

Based on the players’ questionnaire answers and observations during the playtests it became obvious that some would have enjoyed the addition of more gameplay elements. We imagine that in the future the game could have competitive elements that reward players for the unique structures they build or the objects they evolve. Additionally, the game could benefit from a resource-based system in which artefacts are limited and seeds have to be traded to get different variations. Furthermore, being able to interact in a more meaningful way with other players (e.g. talking to other players, trading artefacts etc.) and adding more physical rules (e.g. bouncing, springs etc.) could provide the player with a larger set of affordances. Giving players the means to share or sell the objects they evolved, similar to how players sell flowers in the marketplace in the Petalz video game [16], could not only allow the artefacts to create economic value but also increase the level of social interaction between players.

The current version of the game has a number of technical limitations. For example, the artefacts evolved in the game generally look very abstract and do not always resemble familiar shapes. In the future it might be possible to blend handmade content with generated artefacts. For instance, textures could be applied to the artefacts to create a variety of more natural looks. However, the biggest current limitation in the game is



the lack of a persistent world that players could join at any point. With the current implementation of the game such a world was not computationally feasible. The most expensive operation was the querying of the CPPNs to generate the 3D objects, which lead to too long waiting times when a player wanted to join a server with many existing objects. In the future, this process could be accelerated by incrementally querying the objects closest to the player or by executing the Marching Cubes algorithm on the GPU instead of the CPU.

## VI. CONCLUSION

Artefacts, a novel sandbox video game, allows players to interactively and collaboratively breed an endless variety of 3D objects. Importantly, players can build larger structures together with others by combining evolved objects. An iterative development approach was chosen, in which a total of three different game versions were tested. While the first physics-based iteration allowed some interesting gameplay to emerge, the final version in which physics and gravity were disabled, enabled players to build the greatest variety of different structures. Even though only a small number of people participated in the playtests, their feedback suggests potential for the game concept, and search-based PCG games in general. In the future it will be interesting to see what types of objects many players can evolve together in a persistent Artefacts world, and what structures they might build.

## ACKNOWLEDGMENT

Special thanks to the Artefacts testers.

## REFERENCES

- [1] P. Avery, J. Togelius, E. Alistar, and R. van Leeuwen. Computational intelligence and tower defence games. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 1084–1091. IEEE, 2011.
- [2] N. Cheney, R. MacCurdy, J. Clune, and H. Lipson. Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2013)*, New York, NY, 2013. ACM Press.
- [3] J. Clune and H. Lipson. Evolving 3D objects with a generative encoding inspired by developmental biology. In *Proceedings of the European Conference on Artificial Life (Alife-2011)*, volume 5, pages 2–12, New York, NY, USA, Nov. 2011. ACM.
- [4] K. Compton and M. Mateas. Casual creators. In *Proceedings of the Sixth International Conference on Computational Creativity June*, volume 228, 2015.
- [5] M. Cook and S. Colton. Multi-faceted evolution of simple arcade games. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 289–296. IEEE, 2011.
- [6] E. Hastings, R. Guha, and K. O. Stanley. Evolving content in the galactic arms race video game. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG-09)*, Piscataway, NJ, 2009. IEEE Press.
- [7] E. J. Hastings, R. K. Guha, and K. O. Stanley. Automatic content generation in the galactic arms race video game. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(4):245–263, 2009.
- [8] M. Hendriks, S. Meijer, J. Van Der Velden, and A. Iosup. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, 9(1):1, 2013.
- [9] A. K. Hoover, P. A. Szerlip, and K. O. Stanley. Generating a complete multipart musical composition from a single monophonic melody with functional scaffolding. In M. L. Maher, K. Hammond, A. Pease, R. P. Y. Perez, D. Ventura, and G. Wiggins, editors, *Proceedings of the 3rd International Conference on Computational Creativity (ICCC-2012)*, 2012.
- [10] A. Liapis, G. N. Yannakakis, and J. Togelius. Adapting models of visual aesthetics for personalized content creation. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(3):213–228, Sept 2012. ISSN 1943-068X. doi: 10.1109/TCIAIG.2012.2192438.
- [11] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM siggraph computer graphics*, volume 21, pages 163–169. ACM, 1987.
- [12] B. Mark, T. Berchet, T. Mahlmann, and J. Togelius. Procedural generation of 3d caves for games on the gpu. In *Foundations of Digital Games*, 2015.
- [13] P. T. Olsted, B. Ma, and S. Risi. Interactive evolution of levels for a competitive multiplayer fps. In *Evolutionary Computation (CEC), 2015 IEEE Congress on*, pages 1527–1534. IEEE, 2015.
- [14] S. Risi and J. Togelius. Neuroevolution in games: State of the art and open challenges. *IEEE Transactions on Computational Intelligence and AI in Games*, PP(99):1–1, 2015. ISSN 1943-068X. doi: 10.1109/TCIAIG.2015.2494596.
- [15] S. Risi, D. Cellucci, and H. Lipson. Ribosomal robots: Evolved designs inspired by protein folding. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2013)*, New York, NY, 2013. ACM.
- [16] S. Risi, J. Lehman, D. D’Ambrosio, R. Hall, and K. Stanley. Petalz: Search-based procedural content generation for the casual gamer. *IEEE Transactions on Computational Intelligence and AI in Games*, PP(99):1–1, 2015. ISSN 1943-068X. doi: 10.1109/TCIAIG.2015.2416206.
- [17] J. Secretan, N. Beato, D. D’Ambrosio, A. Rodriguez, A. Campbell, J. Folsom-Kovarik, and K. Stanley. Picbreeder: A case study in collaborative evolutionary exploration of design space. *Evolutionary Computation*, 19(3):373–403, 2011.
- [18] N. Shaker, G. N. Yannakakis, J. Togelius, M. Nicolau, and M. O’Neill. Evolving personalized content for Super Mario Bros using grammatical evolution. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2012)*, Menlo Park, CA, 2012. AAAI Press.
- [19] N. Shaker, J. Togelius, and M. J. Nelson. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2015.
- [20] K. O. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines Special Issue on Developmental Systems*, 8(2):131–162, 2007.
- [21] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10: 99–127, 2002.
- [22] K. O. Stanley, B. D. Bryant, and R. Miikkulainen. Real-time neuroevolution in the NERO video game. *IEEE Transactions on Evolutionary Computation*, 9(6):653–668, December 2005.
- [23] H. Takagi. Interactive evolutionary computation: Fusion of the capacities of EC optimization and human evaluation. *Proceedings of the IEEE*, 89(9):1275–1296, 2001.
- [24] J. Togelius and J. Schmidhuber. An experiment in automatic game design. In *Computational Intelligence and Games, 2008. CIG’08. IEEE Symposium On*, pages 111–118. IEEE, 2008.
- [25] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):172–186, 2011.

# Breeding a Diversity of Super Mario Behaviors Through Interactive Evolution

Patrikk D. Sørensen, Jeppe M. Olsen and Sebastian Risi  
Center for Computer Games Research  
IT University of Copenhagen  
Copenhagen, Denmark  
{pdyr, jmao, sebr}@itu.dk

**Abstract**—Creating controllers for NPCs in video games is traditionally a challenging and time consuming task. While automated learning methods such as *neuroevolution* (i.e. evolving artificial neural networks) have shown promise in this context, they often still require carefully designed fitness functions. In this paper, we show how casual users can create controllers for *Super Mario Bros.* through an *interactive evolutionary computation* (IEC) approach, without prior domain or programming knowledge. By iteratively selecting Super Mario behaviors from a set of candidates, users are able to guide evolution towards behaviors they prefer. The result of a user test show that the participants are able to evolve controllers with very diverse behaviors, which would be difficult through automated approaches. Additionally, the user-evolved controllers perform as well as controllers evolved with a traditional fitness-based approach in terms of distance traveled. The results suggest that IEC is a viable alternative in designing diverse controllers for video games that could be extended to other games in the future.

## I. INTRODUCTION

In recent years it has become more and more popular for video games to enable users to create and share game content. The examples are many and include Sony's puzzle game Little Big Planet<sup>1</sup>, Nadeos racing game TrackMania that allows users to build race tracks<sup>2</sup> and the recent Super Mario Maker, in which users can build their own Mario levels<sup>3</sup>. The user-created content most often comes in the form of levels; very few games let the user create or modify the behavior or underlying structure of the Non-Player-Characters (NPCs). Usually, the NPC behaviors are constructed by programmers and function in predetermined and static ways.

In this paper we show that casual users are able to create sophisticated behaviors for the *Super Mario Bros.* video game by using a simple interface to evaluate several candidate behaviors that is reiterated upon. The approach is based on *interactive evolutionary computation* (IEC) [19] and requires no prior knowledge of neither AI methods nor programming. The developed IEC framework presents users with a selection of GIFs that show short level playthroughs, from which they can then choose the parents of the next generation. Human-in-the-loop approaches, such as IEC, have shown promise in

a variety of different domains [9, 19, 21], because they allow human intuition to guide the search process.

The IEC results in this paper are compared to an automated fitness-based search. Interestingly, users were able to not only interactively evolve a variety of interesting and unique behaviors but also behaviors that perform competitively in comparison to the automated search. Additionally, and potentially more important, users reported that they (1) had fun while evolving Mario behaviors, and (2) felt that they had an impact on evolution. These results indicate that IEC could be a viable and entertaining approach to empowering players to create their own NPC behaviors.

This paper is organized as follows. Section II reviews the evolutionary computation methods this project builds upon, followed by a brief description of the Super Mario domain and game mechanics in Section III. In Section IV the domain representation, methods and core algorithms are described in detail. In Section V we give a description of our experiments, followed by the results in Section VI. Section VII highlights several pitfalls of the current approach and discusses future work that could improve it.

## II. BACKGROUND

This section gives a brief introduction to neuroevolution (NE) in general and the *NeuroEvolution of Augmenting Topologies algorithm* (NEAT), which evolves the Mario controllers in this paper. Lastly, we review several examples in which NE has been combined with IEC.

### A. The Mario Framework

The Mario framework has been used extensively for various AI related research projects and gameplay competitions<sup>4</sup>. Projects range from imitating player behavior [10], evolving behavior trees through grammatical evolution [12], or creating content based on player experience [11], to Mario controllers optimized through reinforcement learning [20].

### B. Neuroevolution

Neuroevolution (NE), the artificial evolution of artificial neural networks (ANNs), takes inspiration from the process that created our biological nervous system and has shown

<sup>1</sup><http://littlebigplanet.playstation.com>

<sup>2</sup><http://en.wikipedia.org/wiki/TrackMania>

<sup>3</sup>[http://en.wikipedia.org/wiki/Super\\_Mario\\_Maker](http://en.wikipedia.org/wiki/Super_Mario_Maker)

<sup>4</sup>For more info see: <http://www.marioai.org/gameplay-track>

promise in a variety of different domains [3], and video games [13]. Typically, a population of neural networks controlling an agent is tested in a given environment, in which those individuals with higher scores – based on some defined fitness criteria – will have a higher chance of being recombined and/or mutated to produce the next generation of solution candidates.

In this paper, the neural networks controlling Mario are evolved with the NEAT algorithm [17]. NEAT begins with a population of simple neural networks and then *adds complexity* over generations by adding new nodes and connections through mutations. By evolving networks in this way, the topology of the network does not need to be known a priori; NEAT searches through increasingly complex networks to find a suitable level of complexity. For a complete overview of NEAT see Stanley and Miikkulainen [17]. Most importantly, such complexification, which resembles how genes are added over the course of natural evolution, allows NEAT to establish high-level features early in evolution and then later elaborate on them.

### C. Interactive Evolutionary Computation

In *interactive evolutionary computation* (IEC), users make aesthetic decisions by rating individual candidates, thereby deciding which individuals breed and which ones die instead of relying on fitness functions designed by developers [19]. Several examples of combining NE and IEC exist, such as users evolving 2D pictures in Picbreeder [16], 3D models through EndlessForms [2] or sound timbres [6].

The combination of IEC and NE has also allowed the creation of new types of games [13]. In the Galactic Arms Race video game [4], weapons are evolving based on how frequently they are shot by the players, and in the social game Petalz [14, 15], players can breed their own unique flowers. Other examples include the NERO game, in which players breed an army of robots in order to fight a team evolved by other players [18]. Each robot in NERO is controlled by a neural network evolved by the NEAT algorithm. By designing different training exercises players can train their robots to behave in specific ways when encountering various challenges on the battlefield.

Recently, a video from Seth Bling showing the evolution of a neural network for Super Mario World, has gathered over two million YouTube hits<sup>5</sup>. The popularity of this work shows the potential of using video games to reach a broader audience. However, so far, none of the aforementioned approaches allow players to design NPCs in a totally user-driven process, which the approach in this paper will try to attempt.

## III. SUPER MARIO GAME MECHANICS

This section describes the game environment and the possible actions the Mario controlling ANN can perform. Super Mario is a world-famous game franchise, containing several games both in 2D and 3D. The game used in this paper is a modified version of the Infinite Super Mario Bros., originally

created by Markus Persson and modified by Julian Togelius and Sergey Karakovskiy for the Mario AI competition<sup>6</sup>. The Mario framework features many of the same enemies and types of levels as the original games, though the specific terrains are randomly generated.

### A. Level Terrain

In each level the avatar Mario has to overcome different challenges with the goal to ultimately reach his princess at the end of the game. To successfully navigate the levels, Mario needs to be able to notice changes in the terrain such as holes that he could fall into or obstacles. Several types of blocks, which can be interacted with, are scattered throughout the level, some of which will yield either coins or power-ups. Likewise, coins are dispersed and can be collected.

### B. Enemy Types

Mario is also facing a multitude of enemies. Most of them are either walking on the ground, jumping around or flying in the air. The majority can be killed by jumping on them, but some have spikes on their backs that prevents this kind of attack. All of them can be shot by fireballs as well. More uncommon are missiles shot from canons and flowers that continuously appear from and disappear into green tubes. The former can be destroyed by jumping on them, while the latter can only be shot by a fireball.

### C. Mario States

The character of Mario can exist in three different states: small, big and big with fire-shooting ability. Every time Mario is hurt by an enemy he regresses to a lower state, and dies if he is hurt in the small state. The previously mentioned power-ups can advance his state to a higher level.

### D. Actions

In the original game, players can control Mario through the Nintendo controller, with buttons right, left, down, up, A and B. Mario is able to move left or right, at a normal or fast speed, crouch, jump in the air and, if he is in the right state, shoot fireballs. In the Infinite Super Mario Bros. version the AI controlled Mario's have access to the same controls.

## IV. METHODS AND REPRESENTATION

This section details the implemented algorithms, neural network setup and IEC interface.

### A. Neural Network Setup

The ANN controlling Mario in this paper, receives a  $3 \times 3$  grid of cells centered around Mario as input, in which different cell values represent different terrain types (Figure 1). In detail, the specific values are as follows: 1.0 = ground, 0.2 = coin, 0.0 = unreachable ground or air, -0.2 = question mark box, -0.5 = breakable standard box, -0.75 = green tube, and finally -1.0 for a hill piece.

<sup>5</sup><https://www.youtube.com/watch?v=qv6UVOQ0F44>

<sup>6</sup><http://www.marioai.org>

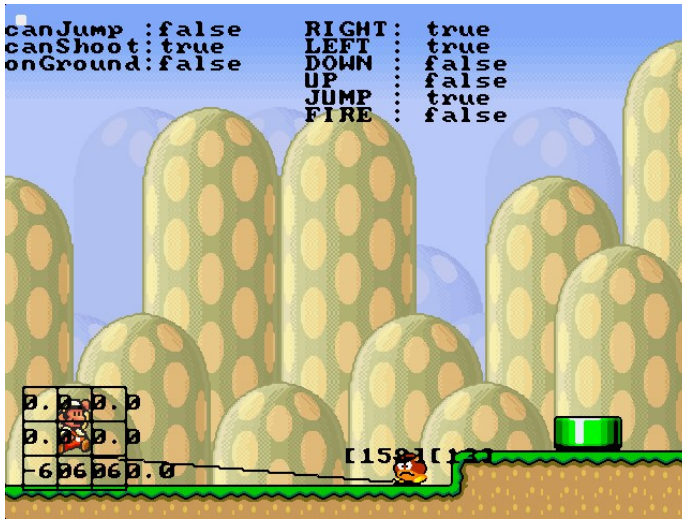


Figure 1. **Mario Representation.** The controlling ANN receives a  $3 \times 3$  grid as input together with information about the distance and angles to enemies, and conditional domain variables *canJump* and *onGround*. The ANN outputs (shown at the top-right corner) determine the action that Mario performs each tick.

Additionally, the ANN receives the distance and angle to the two nearest enemies relative to Mario; the enemy type is currently not provided to the network. The values for both angle and distance are normalized in the range  $[-1, 1]$ . The domain conditional inputs *canJump* and *onGround* are represented as either  $-1.0$  (false) or  $1.0$  (true).

The ANN has six outputs: right, left, up, down, jump, and fire. If the output value for an action is higher than a threshold of  $0.5$ , the particular button is pressed. Mario can perform multiple actions at the same time (e.g. shooting and jumping), except to run left and right. In case both left and right are chosen by the ANN, the one with the highest activation is performed; if both have the same value Mario will not move.

As the Mario framework implementation is written in JAVA, the framework in this paper is build on the NEAT and Java Genetic Algorithms Package (JGAP)<sup>7</sup> based framework *Another NEAT Java Implementation* (ANJI)<sup>8</sup>.

### B. The IEC Interface

The motivation for the IEC approach in this paper is the potential to enable casual users to create Mario controller without any technical skills. Figure 2 shows the developed IEC interface, which aims to accomplish this goal. While the user is watching, a total of nine controllers are playing – one after the other – through a small part of a Mario level. The number of presented controllers tries to strike a balance between giving the user enough variety to choose from while still allowing a reasonable quick evaluation of all behaviors shown.

During the playthroughs, GIFs are recorded for each of the different controllers that show Mario in action. Once the whole population has been played and recorded, a window

Table I  
EVOLUTIONARY APPROACHES.

Approach	Evaluation
Fitness-based Controller	Fitness function awarding cells passed
IEC Free Play Controller	User without any specific goal
IEC Competition Controller	User with the goal to pass as many cells as possible

with all nine recorded GIFs is shown to the user (Figure 2). The user is then able to evaluate and compare each individual in the population and select one preferred controller by simply clicking on the particular GIF. Based on the user's selection, the next generation of controllers is created through mutating the selected individual and the process starts again. That way users can guide the evolutionary search towards Mario behaviors they find interesting.

## V. EXPERIMENTS

To test whether users can evolve Mario behaviors, a user study was performed on site at the IT University of Copenhagen with a total of 20 participants. Each participant was asked to evolve controllers for 20 generations through the interface presented in Section IV. Additionally, we divided the participants in groups of ten and gave them different instructions in order to determine under what conditions certain behaviors evolve. The first ten participants were given no constraints as what to create, and were encouraged to evolve whichever behaviors they preferred. The other ten were told to evolve controllers that could travel as far as possible in the level. The second user study was set-up as a contest with a small prize for the best performing controller, to give some additional incentive for the players to do their best.

After the users evolved Mario controller for 20 generations, they were asked to answer two questions on a scale from 1–6 (where 6 is best). The questions read as follows:

- 1) *How much impact do you feel that your choices had on the evolutionary process?*
- 2) *How much fun was it to develop your AI this way?*

Controllers evolved through IEC were compared to controllers evolved with a traditional automated fitness-based approach. The fitness for the automated approach was the number of cells passed at the end of the simulation. The simulation was terminated if Mario reached the end of the level, he died or the time limit was reached. Additionally, to create more robust controllers, the starting position of the avatar was moved every four generations to a different location in the same level, for both the automated and IEC approach (Figure 3). Table I shows an overview of the three approaches.

### A. Experimental Setup and Parameters

The duration of GIFs shown to the users is initially set to 2,1 seconds, but increases gradually each generation, with a maximum of 4 seconds in the final generation. As it is often

<sup>7</sup><http://jgap.sourceforge.net>

<sup>8</sup><http://anji.sourceforge.net/>



Figure 2. **The IEC User Interface.** The user is presented with nine small playthroughs recorded as GIFs, from which the parent for the next generation can be chosen. The IEC approach offers casual users the ability to breed Mario controllers without requiring technical skills.

easier to eliminate inferior behaviors at earlier generations, the duration was limited in the beginning to reduce user fatigue.

The population size was set to nine for both the automated and IEC approach. The number of generations was set to 20. Offspring had a weight mutation chance of 0.55, 0.01 chance of node addition, and 0.01 chance of link addition.

## VI. RESULTS

The IEC results are based on the ten participants of each experiment and the fitness-based results are collected from ten independent evolutionary runs. Figure 4 depicts a general trend for all approaches to improve over generations. In particular, there is a significant increase in performance for all methods comparing first and last generations ( $p < 0.05$ ; Student's  $t$ -test). The pair-wise differences between the approaches are on the other hand not significantly different, which indicates that both automated and IEC approaches (free play and competitive) are able to evolve similar performing Mario controllers.

Not surprisingly, for all three approaches there are drops in performance when Mario's starting position is moved in generations 5, 9, 13 and 17. Not only does the level layout change at a new starting position, which might break less general controllers, later parts of the level often contain new object types (e.g. a question mark box) that the ANN controller first has to learn to respond to.

Additionally, the results show that the standard deviation for cells passed for the user-evolved controllers is often higher than for the automated approach. This difference indicates that a fitness-based approach tends to create more uniform controllers, while there is more variety in the type of behaviors evolved through IEC, which we examine next.

### A. IEC Evolved Behavior Examples

Indeed, the participants in the user tests were able to evolve controllers displaying a variety of different behaviors. While most users would first focus on creating controllers capable of jumping and moving to the right (a strategy often also



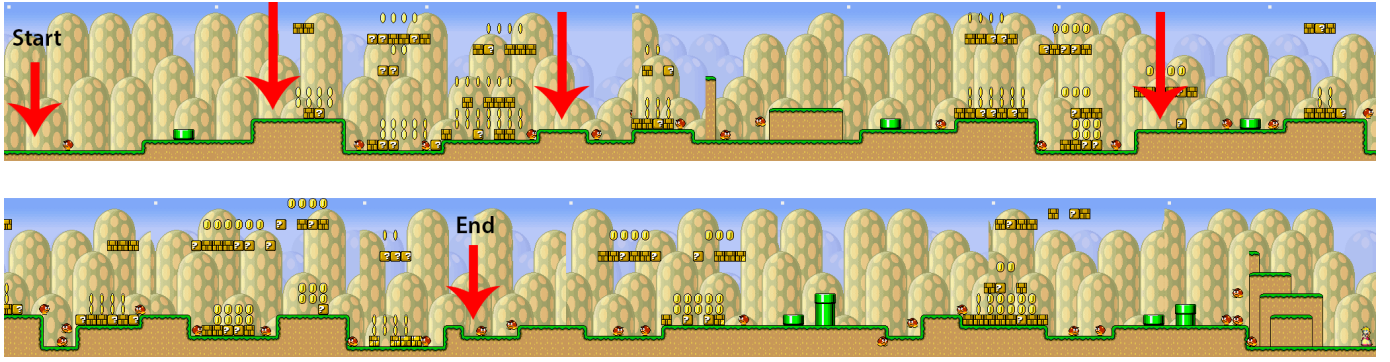


Figure 3. **Training Starting Positions.** To encourage the evolution of general behaviors, the starting position of Mario is moved every four generations to the next pre-defined position in the same level.

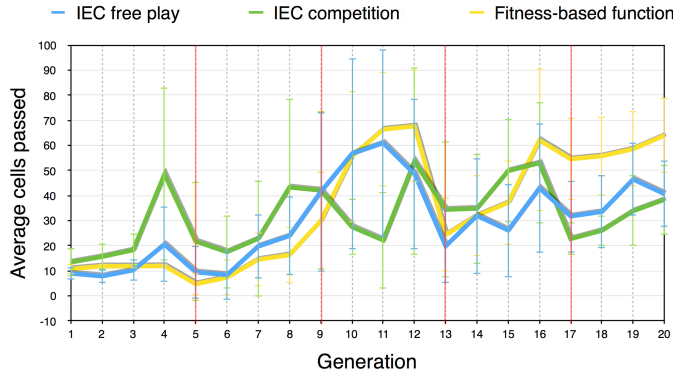


Figure 4. **Training Performance.** Shown are average cells passed during training over generations. The vertical red lines indicate the generations when Mario is set to a new position in the level.

discovered by the automated fitness-based search), some tried to evolve more specific and unique behavior.

Examples include behaviors that would try to collect as many coins as possible or rush through the level without paying attention to enemies or coins. Other controllers would act more cautiously, standing still and ducking when encountering enemies, and would only move once the enemies were behind Mario to his left.

Figure 5 shows a unique IEC created Mario behavior called *living-on-the-edge*. Mario would first try to run into enemies when in his normal state, and then try to keep a specific distance to the enemy in the small Mario state (see Section III-C for details on the different Mario states).

More aggressive Mario controllers such as *im-gonna-kill-everything-that-moves-mario*, which tries to jump on every encountered enemy while shooting fireballs in all directions, was also evolved in the IEC free play session. Other participants would try to evolve slightly more pacifistic Mario behaviors that used enemies as pads to jump higher, progressing more easily through difficult obstacles in the level (e.g. a high cliff). The reader is encouraged to take a look at the video accompanying this paper (available at <https://goo.gl/Ell82m>), to get a better sense of the types of controllers that were evolved and the IEC system in action.

Table II  
QUESTIONNAIRE RESULTS.

Questions	Average Rating	Standard Deviation
Q1: Impact on evolution	4.29	0.8
Q2: Amount of fun when evolving	4.88	1.0

### B. Questionnaire Results

The results from the questionnaire are shown in Table II. In general, the participants felt that they had significant impact on evolution with an average rating of 4.29 out of 6. However, there were no participants who felt that they had complete control, which could be due to the fact that (1) Mario's starting position was moved every fourth generation leading to drops in performance and unexpected behaviors, or that (2) IEC is simply a stochastic process.

Maybe slightly surprising, over 25% gave the maximum score when asked about the level of fun they had breeding Mario controllers, with a score of 4.88 on average. The user responses indicate that IEC-based interfaces have potential for other video games as well, which can benefit from casual users evaluating and evolving NPC behaviors. Similarly to games like NERO [18] and EvoCommander [5], the results in this paper suggest that the process of evolving NPC behaviors can be a novel and entertaining game mechanic.

### C. Generalization Test

NPC controllers employed in video games should be able to deal with variations in their environment and ideally generalize to situations they might not have encountered during training. To test the generality of the evolved behaviors in this paper, the selected IEC and fitness-based controllers from each generation are tested in ten variations of the level they were trained in. In each variation the level layout, amount and location of enemies, enemy types, tubes, coins and breakable boxes is changed randomly.

Figure 6 shows the generalization performance of the three approaches. IEC free play reaches the highest generalization performance, however, the pair-wise differences between the



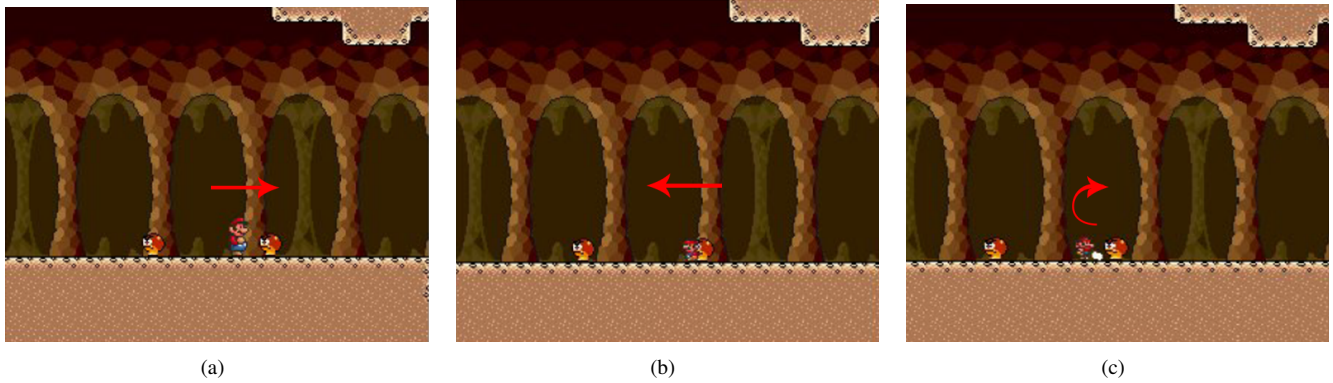


Figure 5. **Storyboard of the “living-on-the-edge” Mario controller.** The controller runs into an enemy (a) and then after converting to the smallest Mario state (b), walks as close to the enemies as possible without touching it (c).

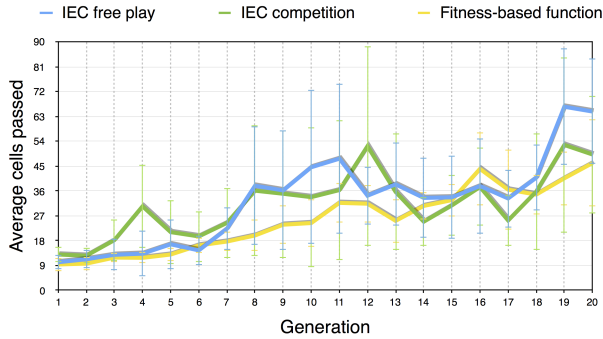


Figure 6. **Generalization Performance.** The controllers selected during training in each generation are tested on their ability to perform in levels they have not seen during training. Each controller is evaluated on ten different level layouts.

approaches are not significant. Similarly to the training performance, all approaches have a statistically higher generalization performance when comparing the first to the last generation ( $p < 0.05$ ). The performance of the controllers on the generalization test is mostly comparable to the performance in training, indicating that the evolved behaviors are able to deal with some variation in the layout of levels.

Overall the results suggest that casual users are able to evolve a variety of unique behaviors that often also perform on par to controllers evolved with a fitness-based approach. It is important to note that the main goal of this comparison is not to determine which method produces the highest performing controllers, but rather to establish a base-line that the IEC-based approaches can be compared to. Inevitably, some will argue that by giving the fitness-based approach more generations it would eventually outperform the IEC approach. While this is likely true, the main advantage of the IEC approach is that even in a game like Mario (i.e. a game whose main goal it is to just advance to the right), players can guide evolution towards unique and interesting behaviors that would not have been rewarded by a naive fitness-based approach (see Section VI-A for examples).

## VII. DISCUSSION AND FUTURE WORK

This section discusses the strengths and weaknesses of the presented approach to create Mario controllers. We also try to articulate suggestions for improvements and draw perspectives to other projects and methods.

### A. Representation

There exist a variety of different ways to represent the Mario world to an ANN-based controller, which differ in terms of the number of ANN inputs and level of sensory abstraction. The representation chosen in this paper tries to strike a balance between selecting the inputs that the controller needs in order to perceive its world, and keeping their number as small as possible; each additional input can lead to an increase in the evolutionary search space and greater demands on the information processing capabilities of the network.

The number of inputs for each object type (e.g. terrain grid size, number enemies) can also have a significant influence on the behavior of the agent. Representing too many enemies and having too small a grid, means the network will initially be highly sensitive to nearby enemies and less sensitive to the terrain. While the ANN can be trained to compensate for this or its inputs can be scaled, it will require additional training time or domain-dependent manual tweaking.

### B. User Fatigue

As IEC can be a time consuming process, user fatigue can be an issue. To produce a desired and usable result the process of iteratively selecting behaviors can take many hours. In the current system it takes approximately 60 seconds per generation, both to record the GIFs and to choose among them. The current implementation is not optimal as users have to wait for the recording of each GIF, and even though they can watch the controllers perform during this process, it increases the evaluation time significantly.

Initially, we experimented with combining IEC with an automated fitness-based search in between each IEC step to speed up the evolutionary process. The automated search would run for a few generations, rewarding Mario for moving towards the right. However, this addition sometimes introduced

an unwanted bias, leading evolution away from the direction sought by the user (e.g. a Mario controller that moves left).

User fatigue can also be negatively influenced if the controllers presented to the user are all very similar; unvaried controllers might not be very meaningful for the user to choose between. As proposed by Woolley and Stanley [21], a potential solution to this problem could be *novelty-assisted interactive evolution* (NA-IEC). By combining IEC with *novelty search* [7], a divergent evolutionary search method, the users would only choose from candidate solutions of *novel* behaviors, thereby accelerating the evolutionary search. Additionally, to gain insights into the quality of user-evolved versus automation-evolved *behaviors*, the IEC approach should be compared to a novelty search based approach. Another future extension to accelerate the IEC approach could be a rank-based IEC approach that was introduced by Liapis et al. [8] for content generation. The rank-based approach has shown advantages over the standard IEC approach with respect to speed of convergence.

### C. User Evaluation

A potential pitfall to the human evaluation step is the relative short duration of the GIF; the user only gets a small glimpse of Mario's behavior. Currently, the duration of the GIF is a compromise between how long the user should have to wait for the recording and how much gameplay the user needs to see in order to evaluate the controller properly. A solution could be to automatically learn a model of the user [1] and only show a few longer playthroughs to the user that the learned user model is uncertain about.

In the initial generations the ANNs often perform seemingly random behaviors. Instead of starting from random controllers, users could instead build upon the work of others, similar to how users collaborate in Picbreeder [16].

## VIII. CONCLUSION

The presented approach allows users, for the first time, to interactively evolve behaviors for Super Mario. The results show that controllers evolved with IEC perform similarly well compared to a fitness-based search in terms of distance traveled, but importantly display more varied strategies and behaviors. Moreover, the IEC users reported that they had fun while evaluating and evolving controllers. In the future, this system could be extended to other video games and to allow many users to evolve behaviors collaboratively online.

## REFERENCES

- [1] J. C. Bongard and G. S. Hornby. Combining fitness-based search and user modeling in evolutionary robotics. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 159–166. ACM, 2013.
- [2] J. Clune and H. Lipson. Evolving three-dimensional objects with a generative encoding inspired by developmental biology. In *Proceedings of the European Conference on Artificial Life*, pages 144–148, 2011.
- [3] D. Floreano, P. Dürri, and C. Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.
- [4] E. J. Hastings, R. K. Guha, and K. O. Stanley. Evolving content in the galactic arms race video game. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 241–248. IEEE, 2009.
- [5] D. Jallo, S. Risi, and J. Togelius. EvoCommander: A novel game based on evolving and switching between artificial brains. *IEEE Transactions on Computational Intelligence and AI in Games*, PP(99):1–1, 2016.
- [6] B. Jónsson, A. K. Hoover, and S. Risi. Interactively evolving compositional sound synthesis networks. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*, pages 321–328. ACM, 2015.
- [7] J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.
- [8] A. Liapis, H. P. Martinez, J. Togelius, and G. N. Yannakakis. Adaptive game level creation through rank-based interactive evolution. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pages 1–8. IEEE, 2013.
- [9] M. Löwe and S. Risi. Accelerating the evolution of cognitive behaviors through human-computer collaboration. In *Proceedings of the 2014 conference on Genetic and evolutionary computation*. ACM, 2016.
- [10] J. Ortega, N. Shaker, J. Togelius, and G. N. Yannakakis. Imitating human playing styles in super mario bros. *Entertainment Computing*, 4(2):93–104, 2013.
- [11] C. Pedersen, J. Togelius, and G. N. Yannakakis. Modeling player experience for content creation. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(1):54–67, 2010.
- [12] D. Perez, M. Nicolau, M. O'Neill, and A. Brabazon. Evolving behaviour trees for the Mario AI competition using grammatical evolution. In *Applications of evolutionary computation*, pages 123–132. Springer, 2011.
- [13] S. Risi and J. Togelius. Neuroevolution in games: State of the art and open challenges. *IEEE Transactions on Computational Intelligence and AI in Games*, 2015.
- [14] S. Risi, J. Lehman, D. B. D'Ambrosio, R. Hall, and K. O. Stanley. Combining search-based procedural content generation and social gaming in the Petalz video game. In *8th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2012.
- [15] S. Risi, J. Lehman, D. D'Ambrosio, R. Hall, and K. Stanley. Petalz: Search-based procedural content generation for the casual gamer. *IEEE Transactions on Computational Intelligence and AI in Games*, PP(99):1–1, 2015.
- [16] J. Secretan, N. Beato, D. B. D'Ambrosio, A. Rodriguez, A. Campbell, and K. O. Stanley. Picbreeder: evolving pictures collaboratively online. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1759–1768. ACM, 2008.
- [17] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.
- [18] K. O. Stanley, B. D. Bryant, and R. Miikkulainen. Evolving neural network agents in the nero video game. *Proceedings of the IEEE*, pages 182–189, 2005.
- [19] H. Takagi. Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation. *Proceedings of the IEEE*, 89(9):1275 – 1296, 2001.
- [20] J. Togelius, S. Karakovskiy, J. Koutník, and J. Schmidhuber. Super Mario evolution. In *Computational Intelligence and Games, IEEE Symposium on*, pages 156–161. IEEE, 2009.
- [21] B. G. Woolley and K. O. Stanley. A novel human-computer collaboration: combining novelty search with interactive evolution. In *Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 233–240. ACM, 2014.

# Evolving Micro for 3D Real-Time Strategy Games

Tyler DeWitt, Sushil J. Louis, and Siming Liu

Dept. of Computer Science and Engineering

University of Nevada, Reno

{tdewitt, sushil, simingl}@cse.unr.edu

**Abstract**—This paper extends prior work in generating two dimensional micro for Real-Time Strategy games to three dimensions. We extend our influence map and potential fields representation to three dimensions and compare two hill-climbers with a genetic algorithm on the problem of generating high performance influence map, potential field, and reactive control parameters that control the behavior of units in an open source Real-Time Strategy game. Results indicate that genetic algorithms evolve better behaviors for ranged units that inflict damage on enemies while kiting to avoid damage. Additionally, genetic algorithms evolve better behaviors for melee units that concentrate firepower on selective enemies to decrease the opposing army's effectiveness. Evolved behaviors, particularly for ranged units, generalize well to new scenarios. Our work thus provides evidence for the viability of an influence map and potential fields based representation for reactive control algorithms in games, 3D simulations, and aerial vehicle swarms.

## I. INTRODUCTION

Real-Time Strategy (RTS) games are a sub-genre of video games where players gather resources to build units to fight and defeat adversaries. Players collect resources to power up an economy that can then produce military units used to destroy opponent units and economy. RTS games thus incorporate elements of strategic economic development and tactical battle management that adds complexity to game play. As such, many interesting CI and AI research challenges exist within the game genre [1], [2], [3]. First, dynamic environments within RTS games mean that we need real-time planning on several levels - strategic, tactical, and reactive. Second, a “fog of war” hides enemy disposition and strategy, therefore players have to scout to gain information to formulate effective strategies. Third, players must learn and exploit their opponents’ playing “style” quickly in order to gain the advantage in future games. Fourth, players must employ spatial and temporal reasoning to coordinate effective unit formations and time-sensitive actions on a tactical and strategic level.

These challenges lead to two broad areas of RTS AI research that encapsulate game play elements in RTS games, macro and micro. Macro refers more to long term decision making dealing with resource management and creating a strong economy. A stronger economy enables more production of military units that battle the opponent. Micro refers to controlling small sets of such military units in combat. Good micro minimizes damage received by friendly units while maximizing damage dealt to enemy units. Good macro combined with good micro wins RTS games. Often, an RTS game may have multiple skirmishes between opposing groups of units and superior micro during a single skirmish may change the entire course of the game.

Influence maps (IMs) and potential fields (PFs) techniques

have been used, in the past, for spatial reasoning and unit maneuvering [4]. IMs map a number to each cell of a discretized game map and the number can indicate areas of interests within the level to the game AI [5], [6]. In this paper, we extend influence maps to 3D as a volumetric grid over 3D space. Each grid volume, or cell, contains a numerical value indicating the influence of nearby entities. Figure 1 shows the three-dimensional (3D) influence map of enemy units. Each IM cell value, computed by an IM function, depends on two parameters, a `weight`, corresponding to the influence of the entity occupying the cell and a `range` for this influence. The final value at a cell is the sum of the influences of all entities that have that cell within their range. Assume we design the IM function to provide low values for enemy influence and high values for friendly influence, a very low value in certain cells reveals that there is heavy enemy presence and therefore the area corresponding to those cells is dangerous for our units.

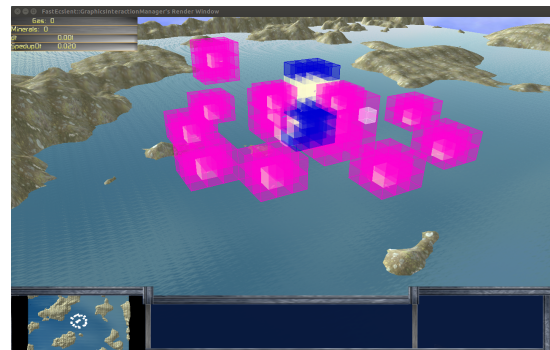


Fig. 1: A 3D influence map showing color coded influence over a map. Pink values are higher than blue which are higher than white. The white areas with the lowest values are thus areas to be avoided by opponents.

Potential field approaches from robotics have been extensively used in guiding group movement in RTS and other games [7], [8], [9]. They enable real-time, cohesive movement, with collision avoidance and have been used to generate good positioning for attack and defense. We extend prior work with 2D potential fields to three dimensions and evolve the (now) 3D parameters that define attractive and repulsive potential fields for game units.

In our experiments, we use a 3D influence map generated from enemy units to tell our units where to go and use two 3D potential fields to control unit navigation. Earlier work has shown that influence maps and potential fields provide representations that can be used by parameterized, but simple, reactive control algorithms to generate high performance 2D



micro [9]. In this work, good kiting, targeting, and fleeing behaviors evolved from the tuning of 3D IM, 3D PF, and 3D reactive algorithm parameters.

This paper extends Liu's 2D work to 3D [9]. Specifically, we use and compare genetic algorithms (GAs) and two hill climbers (HCs) on the problem of finding good 3D influence map, 3D potential field, and reactive control parameters that lead to high performance 3D micro. We generate and compare micro performance in our simulation environment with units similar to Zealots, a close-in, melee unit, and Vultures, a fast, ranged unit, in StarCraft. Our Zealots and Vultures have the ability to move in three dimensions, that is, units can fly in 3D space. We also report on micro performance on scenarios not used during search to investigate how our approach generalizes. Finally, we use FastEcslent, an open source game engine that supports full 3D unit movement in games [10]. We chose FastEcslent in place of the popular StarCraft: Brood Wars API (BWAPI) [11] in order to change the physics and enable full 3D movement. Not only do we want to move to 3D game-physics, but we would like to investigate the effect of more realistic physics on evolved "micro" performance for real-world unmanned aerial vehicles. Figure 2 shows a screen shot of in-game combat between two teams of units within FastEcslent.

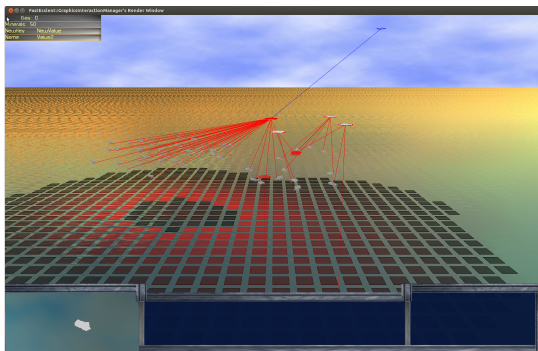


Fig. 2: Units are able to fly and fight enemies in 3D within FastEcslent. Note that although the influence map appears 2D, only the bottom layer of influence map cells is being rendered to provide an unobstructed view of the skirmish.

Preliminary results indicate that both GAs and HCs are able to evolve competent 3D micro. Ranged units (Vultures) learn to kite against melee units (Zealots) in three dimensions and spread firepower by moving to split the enemy unit group into smaller subgroups and thus avoid being surrounded. Ranged units evolve conservative behaviors that preserve health and do not engage in risky tactics. Melee units, when trained against evolved ranged units, learn to concentrate firepower on individual ranged units and diminish the damage dealt by the enemy unit group. Results also show that GAs more consistently produce higher quality solutions than HCs. That is, although hill-climbers occasionally generate high performance micro in shorter time, they do so unreliably. On the other hand, GAs may take more time to produce near-optimal solutions but do so more reliably.

The remainder of this paper is structured as follows. Section II discusses related work in RTS AI research, as well as common micro implementations and approaches. Section III

explains our simulation environment, the design of our AI player, and detailed IM and PF representation for generated 3D micro. Section IV shows preliminary results and compares the solutions generated by the search algorithms and investigates the applicability of solutions in new scenarios. The final section draws conclusions and explores future work.

## II. RELATED WORK

Numerous techniques have been used in the design of RTS AI players[3]. However, we focus on work related to micro management including spatial reasoning and unit movement. In this context, influence maps have been a popular technique for spatial reasoning in RTS and other games. Sweeter *et al.* designed a game agent that used IMs and cellular automata to model the game environment and assist the agent in decision-making within their EmerGEnT game world [5]. Their game agent was capable of pursuing a target while responding to both user actions and natural phenomenon. Bergsma and Spronck implemented IMs to produce adaptive AI for combat in turn-based strategy games [6]. Their adaptive AI evolved high level decision making using an evolutionary algorithm. Avery *et al.* used IMs to co-evolve tactics for a group of boats to move autonomously and attack enemy units in co-ordination [12]. Their technique generated an IM for each unit in order to produce different unit navigation; however, their technique was computationally intensive when increasing the number of units in-game. Preuss *et al.* generated group movement by using flocking in combination with IM path finding within the RTS game *Glest* [13], [14]. Their approach found improvements in group performances across each of their game scenarios. For our research we evolve and use an enemy IM to collect spatial information on enemy disposition and to direct friendly units to good locations from which to launch attacks. Potential fields then guide unit movement during attacks.

PFs were first introduced by Ossama Khatib as a computationally simple approach to real-time obstacle avoidance for mobile robots [7]. This method was then extensively used for collision avoidance among multiple entities [15], [16], [17]. In games, most work related to PFs involve spatial navigation and collision avoidance [18]. Multi-agent potential fields were used by Hagelbäck and Johansson for unit navigation and obstacle avoidance in RTS games [8]. Their research involved the development of an AI player that incorporated PFs at the tactical and reactive control level [19]. Early work in our lab applied spatial reasoning techniques with IMs to evolve a complete RTS game player [20]. More recent work combined IMs and PFs as a basis representation to generate micro position and movement tactics [9], [21], [4]. In this paper, we extend Liu's reactive control algorithm for micro to use 3D IMs and PFs.

Previous work by Uriarte and Ontañón implemented kiting using IMs for group positioning [22]. Their approach was incorporated into the NOVA bot, which competed in the annual StarCraft AI Competition. Gunnerud *et al.* developed a hybrid system that combines case-base reasoning and reinforcement learning which improves itself while playing an RTS game. The hybrid system learned effective targeting behaviors specific for a given scenario [23]. Wender *et al.* examined the suitability of reinforced learning algorithms for executing

effective micro in the RTS game StarCraft: Brood Wars [24]. Their results indicate that reinforcement learning algorithms are capable of developing strategies for winning small scale battle while learning actions such as “Fight”, “Retreat”, and “Idle” in combat. Limitations exist in their implementation however, as the default StarCraftBW AI was the opponent and performance was evaluated on a limited set of tasks. In this work, we extend and use Liu’s parameterized stateless distributed control algorithm which tries to maximize damage to the enemy while minimizing the amount of damage received by friendly units. With appropriate evolved parameters, the algorithm generated 2D kiting, targeting, and fleeing. We extend the algorithm to work in 3D to investigate the evolution of similar behaviors. Although many RTS games do not support full 3D movement, our work does, and as such may also be applicable to user interaction with, and control of, real 3D unmanned aerial vehicle swarms.

### III. METHODOLOGY

We used FastEcslent, an open-source and research-oriented game engine built on OGRE [25] that supports full 3D entity movement. Since graphics and user interaction run within a separate thread, FastEcslent can run without graphics or interaction, enabling easier integration with heuristic search algorithms. On the other hand, even when copying unit parameters from StarcraftBW, trying to replicate StarCraft movement and combat exactly is non-trivial. However, exact duplication is not necessary to evaluate our approach and to investigate whether we can evolve kiting and other well know micro behaviors during combat. With that caveat, our skirmish scenarios built within FastEcslent reflect combat found in StarCraftBW by replicating StarCraft units and their respective properties. In addition, we implement 3D physics and extend our influence map and potential field implementations to 3D. Figure 2 shows an in-game screen shot of a FastEcslent skirmish scenario between two opposing sides. In our scenario, each player controls a group of units initially positioned in opposite corners. In our experimental scenarios, the map does not contain obstacles or neutral entities. Second, FastEcslent entity properties reflect those of default StarCraft units, more specifically, Vultures and Zealots. A Vulture is a ranged unit with low hit-points but high movement speed, proving to be effective when outmaneuvering slower enemy units. A Zealot is a melee unit (low attack range) that has more hit-points than a Vulture but is comparatively slower. Table I details the parameters for both Vultures and Zealots in FastEcslent. Lastly, there is no fog of war since we are only looking at skirmishes, not a complete game. We also implemented a baseline opponent AI that behaves similar to the default StarCraft AI to control enemy Zealots. The maximum running time for our scenario is 6000 frames, approximately one minute at normal game speed. We created a skirmish scenario with four 3D moving Vultures on side RED (our side) and fifty 3D moving Zealots on side BLUE.

#### A. Influence Maps and Potential Fields

We represent group behavior as a combination of one enemy influence map, attractor and repulsor potential fields, and a set of reactive control parameters. The IM provides possible move-to locations and the PFs control movement to locations provided by the IM. Two parameters, the *weight*

TABLE I: Unit parameters defined in FastEcslent

Parameter	Vulture	Zealot
Hit-points	80	160
Size	$45 \times 10 \times 12$	$18 \times 3 \times 6$
MaxSpeed	64	40
MaxDamage	20	16
Weapon’s Range	256	224
Weapon’s Cooldown	1.1	1.24

and *range* specify the IM. Since computation time also depends on the number of IM cells in the map, we use a cell size of  $64 \times 64 \times 64$  pixels in the game map. If an enemy unit occupies a cell, the value of that cell and all neighboring cells in *range* get *weight* added to their current value. We call this the *SumIM* and *weight* and *range* are the evolvable parameters. Since we are evolving micro in a full 3D environment in this paper, influence maps and potential fields extend to three dimensions as well. However, extending IMs from 2D to 3D increases the computational complexity of their implementation. The original 2D IM was of  $O(MN)$  complexity where  $M$  is the number of IM cells on the x-axis and  $N$  is the number of IM cells on the y-axis. Considering the number of cells for the original 2D implementation of FastEcslent ( $64 \times 64$ ), 4096 cells updates were needed to update the IM. Since entities now move in three dimensions, the introduction of the z-axis increases the computational complexity to  $O(MNL)$ , where  $L$  is the number of IM cells on the z-axis. However, our IM implementation updates IM cells over multiple frames within a total of three seconds and does not noticeably slow down simulations or adversely affect unit behavior.

Equation 1 shows a standard potential field function, where  $F$  describes the potential force applied to the entity, with  $D$  being the 3D distance from the enemy entity. The force direction is in the direction of the vector difference from the enemy entity and  $c$  and  $e$  are evolvable parameters.

$$F = cD^e \quad (1)$$

We use one attractor PF and one repulsor PF of the form described by Equation 1 to control entity movement in-game. The attractor force guides a unit towards its target. The repulsor force repels units from other units or obstacles. Normally it is stronger than the attractor force at short distances while being weaker at long distances.

$$\vec{P}F = c_a D^{e_a} + c_r D^{e_r} \quad (2)$$

where  $c_a$  and  $e_a$  are attractor force parameters, and  $c_r$  and  $e_r$  parameters for the replusor force.

#### B. Reactive Controls

Along with group positioning and unit navigation, we represented our reactive control behaviors in a way that our search algorithms can tune. Our reactive control behaviors included micro behaviors frequently used by professional human

---

**Algorithm 1** Reactive Control Algorithm

---

```
UpdatePosition();
nearbyUnits ← FindNearbyUnits(enemies,  $R_{nt}$ );
highFocusUnit ← GetHighFocusUnit(nearbyUnits);

// Targeting
if lowUnit.healthPercentage <  $HP_{ef}$  then
    Target ← lowUnit
else if GetNumberOfAttackers(highFocusUnit) > 0 then
    Target ← highFocusUnit
else
    Target ← closestUnit
end if

// Kiting
if Weapon.cooldownTimer < ( $S_t$  * Weapon.cooldownRate) then
    return
end if
if Weapon.cooldownTimer ≤ 0 then
    MoveTowardsAndAttack(Target)
else
    KitingPos ← IM.GetKitingPos(position, Target.position,  $D_{kb}$ )
    if distanceFrom(Target) < (Target.Weapon.range +  $D_k$ ) then
        if Weapon.range > Target.Weapon.range then
            MoveTowards(KitingPos)
        else if BeingTargetedBy(enemies) and healthPercentage <  $HP_{fb}$  then
            MoveTowards(Target);
        end if
    end if
end if
```

---

players: kiting, targeting, and fleeing. Algorithm 1 specifies the algorithm with the targeting and kiting portions outlined.

Targeting selects and concentrates fire on a specific unit depending on candidate enemy unit hit-points and distance. Each of our *units* selects the nearest enemy unit  $t_{closest}$  as a possible target. Within a distance  $R_{nt}$  from  $t_{closest}$ , our *unit* will select a target based on prioritized criteria. The highest priority is  $t_{lowhp}$ , the enemy with the lowest hit-points below the evolvable threshold:  $HP_{ef}$ . Next in priority is  $t_{focus}$ , the enemy unit being targeted by the most friendly units within  $R_{nt}$  relative to  $t_{closest}$ . The third prioritized criteria is  $t_{closest}$ , the nearest enemy unit. Kiting serves as a useful hit-run-repeat tactic for units with higher speed and attack range. Units strike quickly and retreat back to avoid being attacked by the slower units. During kiting, our *unit* moves towards and attacks its *target* as soon as the *unit*'s weapon is ready, which is dependent on  $S_t$ . A *unit* will begin kiting if the *unit*'s weapon is not ready and if the distance between the *unit* and its *target* is less than  $D_k$ . The *unit* moves back (away from the target) to a *kitingPosition* which is computed by the function *getKitingPositionFromIM*( $D_{kb}$ ) from the SumIM.  $D_{kb}$  represents the number of cells away from the *target*'s cell. If  $Cell_t$  is the IM cell containing our target, *getKitingPositionFromIM*( $D_{kb}$ ) finds a neighboring IM cell with the lowest value. We set this new IM cell to  $Cell_t$  and then repeat this process of finding a new  $Cell_t$ ,  $D_{kb}$  times. The algorithm then uses  $Cell_t$  as the *kitingPosition* to move towards. Finally, fleeing to avoid further damage gets triggered when the unit's hit-points fall below a threshold, represented by  $HP_{fb}$ .  $HP_{fb}$  controls this "fleeing"

behavior.

We encode 12 micro parameters, consisting of 6 reactive control parameters as well as 6 IM and PF parameters, into a 51-bit binary string that represent a chromosome for our search algorithms. Our search algorithms then decode these encoded binary strings into a set of parameters as shown in [21]. FastEcslent receives this decoded set of parameters and uses them to run the skirmish. When finished, FastEcslent returns the resulting score and fitness to the calling search algorithm.

### C. Fitness Evaluation

The goal of our scenario is to maximize enemy unit damage while minimizing friendly unit damage. The evaluation function to compute fitness  $F$  reflects this:

$$F = TD_{eu} + (HP_{fu} \times 400) \quad (3)$$

where fitness is calculated at the end of the scenario.  $TD_{eu}$  represents the total damage given to enemy units, while  $HP_{fu}$  is the sum of remaining hit-points of all friendly units. According to our prior experiments, we use the scalar value 400 for multiplying  $HP_{fu}$  to give unit hit-points more weight than enemy unit damage as a means to encourage health conservation and more evasive kiting behaviors. This is somewhat arbitrary and an alternative approach in our current research, is to use a multi-objective evolutionary algorithms and treat damage done and damage received as two criteria in a pareto-optimization setting. Note also that the same fitness can be found in multiple ways. For example, a fitness of 7200 can describe an outcome of 45 enemy units destroyed with no friendly units remaining or an outcome of 40 enemy units destroyed and two friendly units alive with full hit-points. The fitness is used by our search algorithms to bias search.

### D. Hill-climbers

The Bit-Setting Optimization (BSO) hill climber searches a locally optimal solution in the search space by sequentially flipping each bit and saving the better fitness solution when it is found [26]. BSO searches a subset of the search space based on the initial point set by the random seed. In order to make results obtained from GAs and HCs comparable, the maximum number of evaluations made by all algorithms are set to the same number, 600. The BSO starts from the left again when it reaches the end of the chromosome. The Random Flip Optimization (RFO), an alternative hill-climber, randomly chooses a bit in the randomly generated initial chromosome and flips it. This is repeated 600 times.

### E. Genetic Algorithm

We use an elitist GA in our experiments. Assuming the population size is  $N$ , during selection our elitist GA selects the  $N$  best individuals from the combined parent and offspring populations ( $2N$ ) to create the next generation after recombination. We implemented a parallel version of this elitist GA where evaluations are done in parallel to significantly speed up our runs. For our scenarios, the population size was 20, run for 30 generations for a total of 600 evaluations. In order to relate our experiment to that of the original 2D implementation, we use the same crossover and mutation rate for our GA.



The probability of our two-point crossover is 88% and bit-flip mutation probability is 0.01. Standard roulette wheel selection is used to select chromosomes for crossover. These operator choices and GA parameter values were empirically determined to work well.

#### IV. RESULTS AND DISCUSSION

Unit AI behavior within FastEcslent is deterministic, meaning that a given set of parameters guarantees the same fitness every time a scenario runs. The FastEcslent game engine does not add any noise to picking a targeting, probability of hitting the target, or in the amount of damage done. We ran our scenarios with 30 random seeds for each search algorithm.

##### A. Search Algorithm Results

In the skirmish scenario with 4 friendly Vultures vs. 50 enemy Zealots, all three search algorithms were able to evolve high fitness 3D micro within 600 evaluations. Ranged units evolved kiting behaviors that were successful in destroying numerous enemy units while avoiding damage. According to the fitness function, the theoretical maximum score for our scenario is 9600. This is obtained when eliminating all of the enemy units (8000) with no friendly units receiving damage (1600). Figure 3 illustrates the average fitness from 30 runs.

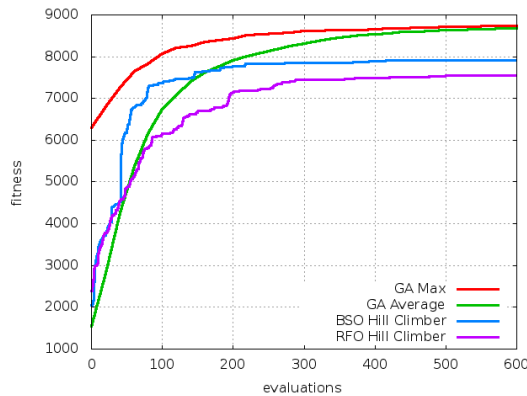


Fig. 3: Average performance of GA, BSO, and RFO for our scenario with 30 different random seeds. X-axis represents number of evaluations and Y-axis shows the average fitness over 30 runs.

The average fitness of the BSO shown in Figure 3 climbed quickly in the first 100 evaluations which seemed to indicate that the BSO quickly finds (local) optima. The final average score for BSO, 7904 was the second highest average score among the three tested algorithms. The best fitness over 30 runs was 9200, indicating that the BSO sometimes did very well and this solution destroyed 48 of the 50 enemy Zealots but did receive some damage. The average fitness of the RFO shown in Figure 3 did not climb as quickly as the other search algorithms and RFO usually did worse than the others. This was reflected in the final average score of 7556 attained over the 30 RFO runs, the worst among the three tested algorithms. On our scenario, RFO seemed less reliable than BSO. However, the highest fitness obtained by RFO was 9020 which indicated that RFO also has the potential to find relatively high quality

solutions. The solution with a score of 9020 destroyed 43 units but received more damage than the best solution found by the BSO.

The average fitness curves of the GA shown in Figure 3 rise smoothly and end higher than the averages of both HCs. The average over 30 runs of the maximum fitness in the GA population (GA Max) was also consistently higher than the quick climbing BSO. This indicated that the GA was more reliably and more quickly find high fitness solutions. The final average fitness for the GA was 8745.3 which was the highest average among the three tested algorithms. The highest fitness obtained by the GA was 9260 which was also the highest fitness found by any algorithm. This solution inflicted 7660 damage, destroying 42 of the 50 Zealots while not receiving any damage. The solution produced by the GA destroyed the least amount of Zealots out of all search algorithms, but displayed near-optimal kiting abilities by avoiding any damage whatsoever.

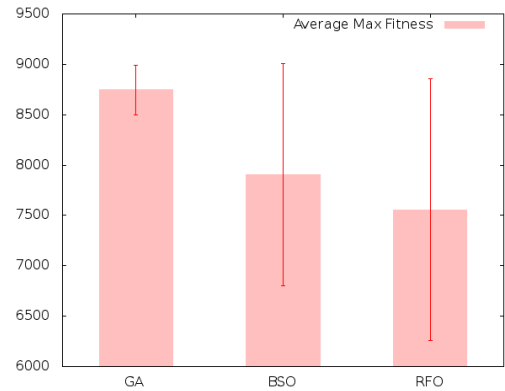


Fig. 4: For our scenario, the standard deviation showed by the error bars tells that GA on average produced more reliable solutions after 600 evaluations.

If we define a fitness of 8000 as our threshold for good performance, we can see that the GA performed better than either HC. The GA found solutions above 8000 every run. The BSO could only find solutions above a score of 8000 on 16 out of the 30 runs while the RFO found solutions above a score of 8000 on 13 of the 30 runs. The differences in final average fitness between the BSO and GA were statistically significant with a one-tailed  $P < 0.0001$ . Additionally, Figure 4 shows that the standard deviation of the GA's set of final fitnesses was 245.17, whereas the standard deviation of the BSO's final fitnesses was 1105.43. The differences in final average fitnesses between the GA and RFO were statistically significant with a one-tailed  $P < 0.0001$ . The standard deviation for RFO final solutions was 1298.26. These statistically significant results provide evidence that the GA more reliably produces higher quality 3D micro.

##### B. Evolved 3D Micro Behavior

We are also interested in the highest fitness 3D micro behavior generated by the search algorithms. The parameters for evolved Vultures in Table II produced by the GA results in a score of 9260, the highest score found. The behavior created

by these parameters enabled friendly Vultures to spread across the map and split enemy units into smaller subgroups, thus decreasing the concentrated firepower of the more numerous enemy group so our units do not become overwhelmed. Figure 5 shows a screen shot of the skirmish that illustrates our Vulture's 3D micro behavior. The parameters specifying PF values show that our units were strongly attracted towards enemy units with small repulsion, allowing friendly units to strike closely but remain out of the enemy unit's weapon range. A low freeze time ( $S_t$ ) also allows for units to kite more frequently and avoid becoming overwhelmed when at stand still. A maximum value of the  $HP_{ef}$  parameter demonstrates that the evolved Vultures did not prefer targeting previously damaged enemies and instead targeted units closest to them, preventing potentially dangerous chases through enemy groups and kiting in quick, successive intervals. Videos of evolved micro compared with initially generated micro can be found online at <http://www.cse.unr.edu/~tdewitt/>.

TABLE II: Best found solutions for both units.

Unit	IMs		PFs				Reactive control					
	$W$	$R$	$c_a$	$c_r$	$e_a$	$e_r$	$St$	$D_k$	$R_{nt}$	$D_{kb}$	$HP_{ef}$	$HP_{fb}$
Vultures	14	10	60	13	10	3	2	22	14	4	7	1
Zealots	12	9	55	27	9	2	6	21	10	6	5	7

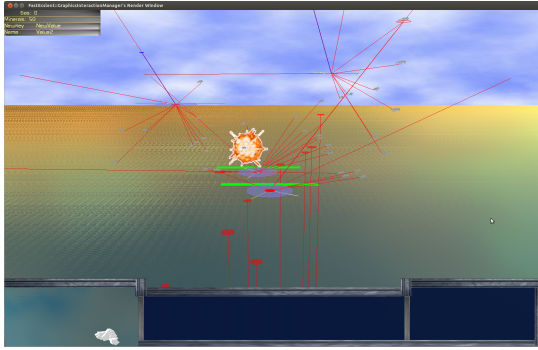


Fig. 5: Vultures with evolved micro fight smaller fragments of the enemy group for a higher chance of survival.

### C. Generalizability of Evolved 3D Micro Behaviors

We tested the generalizability of our evolved set of parameters for Vultures found in Table II by applying them to control vultures in new scenarios. For each side, at its corner, we randomly generated the unit positions of each side and averaged the fitnesses over 500 runs. Figure 6 (red bars) shows the fitness distribution of all 500 runs for this generalizability test. The average score out of 500 runs is 6391 which is 69.02% of the highest fitness evolved in the original scenario (9260).

We then further tested the generalizability of the evolved Vulture micro by randomly generating initial unit positions anywhere within the map and averaged the scores from each run. Figure 6 (blue bars) also shows the distribution of fitnesses over all 500 runs on this scenario. The average fitness is 6588 which is 71.14% of the highest fitness evolved in the original scenario. Although parameters were evolved on one specific scenario with fixed initial positions for all units, our

representation leads to behavior that is somewhat generalizable over other initial positions.

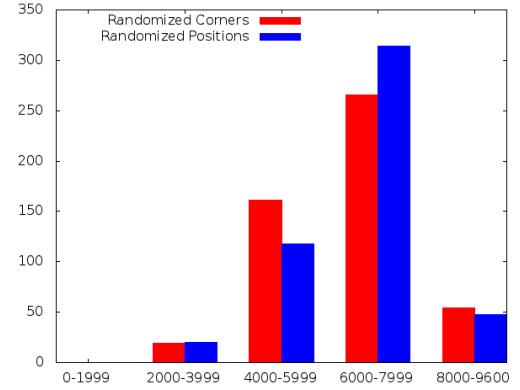


Fig. 6: The evolved 3D micro from our scenario of fitness 9260 generalizes well to new scenarios.





We also apply our solution to new scenarios with fixed initial positions. Table III details the design and results of the new scenarios used to test the kiting efficiency of our evolved Vultures. Scenario 1 starts our 4 evolved Vultures separated into two subgroups in opposite corners to surround 50 enemy Zealots placed in the center of the map. Vultures immediately split the Zealots into two groups with each group moving towards the closest Vulture subgroup. We lost 1 Vulture early in the scenario which decreased the overall fire power of our units for the remaining time duration, resulting in only 19 enemy units destroyed for a fitness of 6160. Scenario 2 places each of the 4 Vultures in each corner with 50 Zealots placed in the map's center. Zealots immediately split into four groups and began to move towards the Vulture closest to them. Vultures were already separated across the map and quickly engaged in kiting behaviors, eliminating 35 of the Zealots with no casualties. This scenario results in a fitness of 8060. Scenario 3 inverts the previous scenario by placing the Zealots in the corners and Vultures in the map's center. Our evolved Vultures split to individually fight Zealot subgroups at the beginning of the scenario. We lose 1 Vulture halfway through the scenario but it was alive long enough to eliminate multiple enemy units, therefore the loss in overall fire power was not as severe as if it had been eliminated early on. Our evolved Vultures still managed to eliminate 31 enemy units, resulting in a fitness of 7220. Scenario 4 doubles unit numbers and has 8 Vultures versus 100 Zealots. Vultures were able to handle Zealots well and although we lost 1 Vulture towards the end of the scenario, the Vultures destroyed 65 Zealots.

The evolved Vultures still engage in some risky behavior and casualties from this, result in a decrease in overall fire power for the entire Vulture group and lowers the group's tactical effectiveness by a considerable amount. However, Vultures still perform well by spreading across the map and kiting Zealot subgroups effectively.

### D. Evolving 3D Zealot Micro Behavior

Once we had good Vulture micro, we investigated evolving 3D Zealot micro against these previously evolved Vultures.

TABLE III: Screen shot of initial 3D unit positioning for four new scenarios.

Scenario	Description	Results
	4 Vultures in opposite corners versus 50 Zealots in center.	Fitness of 6160. 1 friendly destroyed, 19 enemies destroyed.
	4 Vultures in each corner versus 50 Zealots in center.	Fitness of 8060. 0 friendlies destroyed, 35 enemies destroyed.
	4 Vultures in center versus 50 Zealots divided into each corner.	Fitness of 7220. 1 friendly destroyed, 31 enemies destroyed.
	8 Vultures versus 100 Zealots.	Fitness of 14080 1 friendly destroyed, 65 enemies destroyed.

To do so we replicated the same set of experiments with the same map rules while swapping unit sides, therefore our new scenario now consists of 50 friendly Zealots fighting 4 enemy Vultures controlled by the highest performing micro in Table II found in our original scenario. We also modify our fitness function to better suite the objective of melee attack units in our new scenario. The new evaluation function is:

$$F = \begin{cases} (D_{eu} \times 100) + (N_{fu} \times 100), & \text{if } N_{eu} = 0 \\ (D_{eu} \times 100) + (N_{fu} \times 10), & \text{otherwise} \end{cases}$$

where  $D_{eu}$  represents damage, the sum of enemy unit casualties.  $N_{fu}$  and  $N_{eu}$  are the number of friendly units and number of enemy units remaining at the end of the scenario. With this fitness function there is only one way to achieve a particular fitness. For example, a fitness of 4300 indicates that all enemy units were destroyed with 39 friendly units remaining. This conditional fitness function is to guide search algorithms in evolving micro that eliminates all enemy units first in order to avoid evolving passive micro. Again, in future work, we plan to use multi-objective evolutionary algorithms that try to maximize damage done and minimize damage received as the two criteria to be pareto-optimized.

For this scenario, the theoretical maximum score for the scenario is 5400. This is obtained by eliminating all of the enemy units (400) and retaining all friendly units (5000). Figure 7 shows that we are able to evolve Zealots to fight and win against evolved Vultures in this specific scenario. The GA found solutions that were able to eliminate all enemy units in 22 of 30 runs for an average max fitness of 3451. The highest fitness obtained is 5000, which destroyed all 4 enemy Vulture units while losing 4 of 50 friendly Zealot units.

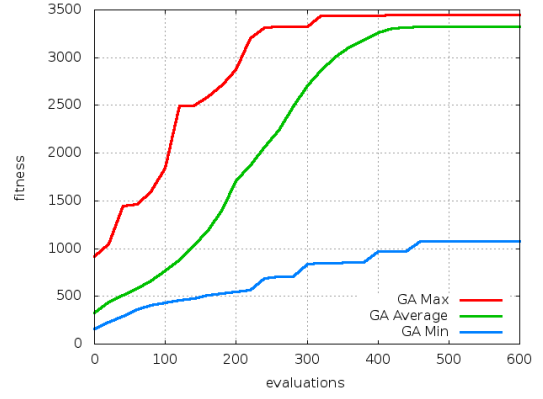


Fig. 7: Average performance of GA for our scenario with 30 different random seeds. X-axis represents number of evaluations and Y-axis shows the average fitness at that evaluation.

The GA was able to generate high quality solutions for melee versus ranged units by having Zealots concentrate firepower as a group on one enemy Vulture at a time, reducing the overall effectiveness of the enemy group with each unit eliminated. Instead of kiting, Zealots learn to rush one Vulture at a time as a means to overwhelm and quickly eliminate this Vulture. Zealots do not evolve kiting behaviors due to their inability to outrun enemy Vultures and instead develop more aggressive, risky behavior to destroy ranged enemy units. Rather than spacing out to fight smaller subgroups of enemy units, Zealots collectively form a condensed group and focus firepower on one enemy unit at a time. The fewer number of Vultures alive, the higher the number of Zealots that stay alive as the skirmish continues. This provides an incentive for Zealots to eliminate Vultures quickly to avoid prolonged skirmishes that lead to more Zealot casualties later in the scenario. A low repulsive PF evolves and allows Zealots to move into a more compact group which permits rushing with the concentrated firepower needed to eliminate ranged enemy units. Table II lists the evolved parameter values for the best evolved Zealot.

Further experiments in testing the generalizability of evolved 3D Zealot micro concludes that there exists limitations in our representation for evolving micro of this specific unit type. Kiting behaviors do not apply well to these melee units when fighting ranged units and our representation of micro does not incorporate effective melee micro parameters (i.e. flanking).

## V. CONCLUSION AND FUTURE WORK

This paper extends prior work in generating two dimensional micro for Real-Time Strategy games to three dimensions. We use influence maps and potential fields to coordinate group positioning and unit movement during skirmishes. Unit group behavior is represented as a set of parameters that define an influence map, an attractive and a repulsive potential field, and reactive controls while limiting the search space for our search algorithms to  $2^{51}$ . Results show that the genetic algorithm and two hillclimbers can find parameter values that lead to high fitness correlated with good micro. However, the genetic algorithm more reliably and more quickly finds

higher fitness parameter values. Both hill climbers find good solutions between 40% and 60% of the time, while the genetic algorithm finds high quality solution a 100% of the time. These results are statistically significant. For ranged versus melee unit combat, ranged units see higher effectiveness when the group becomes more spread and splits enemy firepower. Moreover, ranged units kite efficiently by attacking enemy units while avoiding being within enemy weapon's range. Conversely, evolved melee units can successfully eliminate ranged units by concentrating fire on one unit at a time, quickly reducing the overall effectiveness of the enemy group with each unit destroyed. Our evolved Vultures exploit opposing melee units (Zealots) in every scenario by slicing enemy units into smaller groups to avoid becoming overwhelmed and then kiting till skirmish-time runs out. Results also show that although our evolved 3D ranged unit (Vulture) micro generalize well to new scenarios, our evolved 3D Zealot micro does not generalize as well to other scenarios.

We are interested in evolving effective 3D micro for melee units against ranged units with appropriate representations of melee micro behaviors. We plan to investigate simplifying the fitness functions by turning to a multi-objective formulation of the problem and using multi-objective evolutionary algorithms. Techniques such as case-injection or other knowledge-based systems may be added to our system in future research to further investigate speed, quality, and generalizability of our representation and evolved solutions. We are also interested in co-evolving micro for rather than evolving micro against a fixed opponent. Essentially, we manually did one co-evolutionary cycle when evolving Zealot micro against our prior evolved Vultures. Finally, we plan to investigate evolving multi-unit micro with more complex unit interactions.

#### ACKNOWLEDGMENT

This material is based in part upon work supported by the National Science Foundation under Grant Number IIA-1301726. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. This work is also partially supported by the Office of Naval Research under Grant N00014-15-1-2015. We would also like to acknowledge Nathan Navarro-Griffin (navarrogriffin@gmail.com) who wrote the code that moved FastEcslent to 3D.

#### REFERENCES

- [1] M. Buro, "Call for ai research in rts games," in *Proceedings of the AAAI-04 Workshop on Challenges in Game AI*, 2004, pp. 139–142.
- [2] M. Buro and T. Furtak, "Rts games and real-time ai research," in *Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS)*, vol. 6370, 2004.
- [3] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game ai research and competition in starcraft," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 5, no. 4, pp. 293–311, 2013.
- [4] S. Liu, S. J. Louis, and M. Nicolescu, "Using cigar for finding effective group behaviors in rts game," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013, pp. 1–8.
- [5] P. Sweetser and J. Wiles, "Combining influence maps and cellular automata for reactive game agents," *Intelligent Data Engineering and Automated Learning-IDEAL 2005*, pp. 209–215, 2005.

- [6] M. Bergsma and P. Spronck, "Adaptive spatial reasoning for turn-based strategy games," *Proceedings of AIIDE*, 2008.
- [7] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The international journal of robotics research*, vol. 5, no. 1, pp. 90–98, 1986.
- [8] J. Hagelbäck and S. J. Johansson, "Using multi-agent potential fields in real-time strategy games," in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 2*, ser. AAMAS '08. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 631–638. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1402298.1402312>
- [9] S. Liu, S. J. Louis, and C. Ballinger, "Using ga for finding effective micro behaviors in rts game."
- [10] (2016) Evolutionary computing systems lab, unr. [Online]. Available: <http://ecsl.cse.unr.edu/>
- [11] M. Buro, "Real-time strategy games: A new AI research challenge," *Proceedings of the 18th International Joint Conference on Artificial Intelligence. International Joint Conferences on Artificial Intelligence*, pp. 1534–1535, 2003.
- [12] P. Avery and S. Louis, "Coevolving influence maps for spatial team tactics in a RTS game," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, ser. GECCO '10. New York, NY, USA: ACM, 2010, pp. 783–790. [Online]. Available: <http://doi.acm.org/10.1145/1830483.1830621>
- [13] M. Preuss, N. Beume, H. Danielsiek, T. Hein, B. Naujoks, N. Pi- atkowski, R. Stuer, A. Thom, and S. Wessing, "Towards intelligent team composition and maneuvering in real-time strategy games," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 2, no. 2, pp. 82–98, 2010.
- [14] H. Danielsiek, R. Stuer, A. Thom, N. Beume, B. Naujoks, and M. Preuss, "Intelligent moving of groups in real-time strategy games," in *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*. IEEE, 2008, pp. 71–78.
- [15] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215–233, 2007.
- [16] M. Egerstedt and X. Hu, "Formation constrained multi-agent control," *Robotics and Automation, IEEE Transactions on*, vol. 17, no. 6, pp. 947–951, 2001.
- [17] C. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4. ACM, 1987, pp. 25–34.
- [18] J. Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots," *Robotics and Automation, IEEE Transactions on*, vol. 7, no. 3, pp. 278–288, 1991.
- [19] J. Hagelbäck and S. J. Johansson, "The rise of potential fields in real time strategy bots," *Proceedings of Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2008.
- [20] C. Miles, J. Quiroz, R. Leigh, and S. Louis, "Co-evolving influence map tree based strategy game players," in *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, april 2007, pp. 88–95.
- [21] S. Liu, S. J. Louis, and M. Nicolescu, "Comparing heuristic search methods for finding effective group behaviors in rts game," in *Evolutionary Computation (CEC), 2013 IEEE Congress on*. IEEE, 2013, pp. 1371–1378.
- [22] A. Uriarte and S. Ontanón, "Kiting in rts games using influence maps," in *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2012.
- [23] M. J. Gunnerud, "A cbr/rl system for learning micromanagement in real-time strategy games," 2009.
- [24] S. Wender and I. Watson, "Applying reinforcement learning to small scale combat in the real-time strategy game starcraft: broodwar," in *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*. IEEE, 2012, pp. 402–408.
- [25] T. K. S. Ltd, "Ogre open source 3d graphics engine," February 2005. [Online]. Available: <http://www.ogre3d.org/>
- [26] S. W. Wilson, "Ga-easy doe not imply steepest-ascent optimizable," 1991.

# Semi-automated Level Design via Auto-Playtesting for Handheld Casual Game Creation

Edward J. Powley, Simon Colton, Swen Gaudl, Rob Saunders and Mark J. Nelson

The MetaMakers Institute, Games Academy, Falmouth University, UK

Email: {edward.powley, simon.colton, swen.gaudl, rob.saunders, mark.nelson}@falmouth.ac.uk  
metamakers.falmouth.ac.uk

**Abstract**—We provide a proof of principle that novel and engaging mobile casual games with new aesthetics, game mechanics and player interactions can be designed and tested directly on the device for which they are intended. We describe the Gamika iOS application which includes generative art assets; a design interface enabling the making of physics-based casual games containing multiple levels with aspects ranging from Frogger-like to Asteroids-like and beyond; a configurable automated playtester which can give feedback on the playability of levels; and an automated fine-tuning engine which searches for level parameterisations that enable the game to pass a battery of tests, as evaluated by the auto-playtester. Each aspect of the implementation represents a baseline with much room for improvement, and we present some experimental results and describe how these will guide the future directions for Gamika.

## I. INTRODUCTION

Mobile gaming is an important part of game culture and has become an everyday activity for a large fraction of our society. Within mobile gaming, casual games, categorised by relatively shallow learning curves, comparatively simple game mechanics and relatively short time investment for an engaging return, are extremely popular. Indeed, such games have broadened the popularity of gaming, contributing towards a recent shift in the demographic of game players, i.e., to a more gender-balanced state with a higher average age than a decade ago [1, pp.145]. Unlike music, photography, writing, abstract art and many other areas where handheld *casual creator apps* [2] have enabled consumers to become creative producers, the creative act of game-making is not yet fully supported on mobile devices. In section II, we partially categorise handheld applications that empower game creation and highlight limitations of existing tools.

A desire to reduce these limitations and further democratise game design has led us to build the Gamika iOS application. We believe it to be the first tool enabling entirely new casual games (containing multiple levels with novel aesthetics, game mechanics and player interactions) to be designed on a mobile phone without requiring coding. One of our main contributions here is a breakdown of a subset of physics-based casual games into a set of components so that games can be built by making choices for each part. In section III, we describe this breakdown, and provide a description of the Gamika software, focusing on how it enables the construction of game levels through a copy-and-tweak methodology, involving generative art assets, a mutation mechanism, drawing functionality, and fine-grained tuning of the game components.

Gamika, along with some other mobile apps, enables the production of clones of existing games, which are *safe* in the sense that the gameplay has been pre-tested. However,

Gamika also empowers designers to produce entirely new games with new game mechanics and *untried* gameplay. This increases the need to extensively tune and playtest game levels, a potentially tedious task that could reduce users' enjoyment of the app. Since we'd like the app to be fun to design as well as play games in, we have endeavoured to make tuning and testing of levels an entertaining experience. In section IV, we introduce a puzzle/reaction game called *Let it Snow*, made with Gamika, as a running example game for which playtesting and tuning is required for each level. To aid designers in rapidly producing levels, we have implemented an automated playtester in Gamika, which can be watched while playing and is configurable, as described in section V.

To further support designers in producing game levels, we have implemented an automated tuning facility, which can search for values of user-specified parameters which fix various failing aspects of a level design. This functionality is described via a case study in section VI. Here, a designer first designs an original level and then plays it to find values for the parameterisation of an imperfect playtester. Logs of both human and computer gameplay, in addition to a parameter sweep, highlight aspects of the game which aid the designer in making new levels. The designer then specifies a test suite that a game level must pass in order to be considered as part of a game design. Finally, the ten produces five new levels of *Let it Snow* by simply drawing one element of it in Gamika, and letting the software fine-tune various parameters to find a design which passes the tests.

The work here represents a proof of the principle that on-device casual game creation, beyond producing clones and not requiring coding, is possible. Our contribution is the whole pipeline and the AI functionality embedded in Gamika, which supports casual co-creation, rather than a focus on studying and optimising one particular aspect. Each part of the Gamika implementation is sub-optimal and can be improved via better user interaction design, faster search techniques and more sophisticated parameterisations, which will be informed by the results of the case study, as described in section VII.

## II. BACKGROUND

Compton and Mateas [2] introduce the term *casual creator* to describe a piece of software with which users can quickly and easily create artefacts such as musical compositions, artistic imagery such as filtered digital photos, abstract artworks and graphic designs, and texts such as stories and poems. We are interested in casual creators that work directly on handheld devices, rather than those which merely enable deployment to such devices, and in particular those which allow for the creation of digital games or game levels.



While there are many environments in which novice game designers can learn the craft, such as GameMaker: Studio (yoyogames.com), Scratch (scratch.mit.edu) and Stencyl (stencyl.com), these require PC-based development. Of the environments that enable on-device game/level creation, the following is a partial characterisation:

- Apps which require learning programming skills. Some, such as Scratch Jr (scratchjr.org) or HopScotch (gethopscotch.com), are designed to introduce children to coding.
- Apps which enable only the skinning of existing game templates, which allows for some level of creativity, but not the entire game making experience. Examples here include Coda Game (codarica.com) and Playr (playr.us).
- Apps which enable the authoring of fairly complex game levels for an existing game. An example here is Createrra 2 (incuvo.com), where levels of a side-scrolling platform game can be created on-device. These empower creative expression, limited to the provided characters, rules and game worlds.

Of particular note here is Sketch Nation (sketchnation.com), where the final two categories above are combined, i.e., users can create complex levels within a number of templates.

We defined a space of casual games by factoring Gamika games into a set of numerical parameters (elaborated in the next section). The idea of defining a space of games has similarities to systems such as VGDL [3] and PuzzleScript (puzzlescript.net). With these systems, games are mapped into a space of hierarchical code structures, whereas Gamika uses a space of numerical vectors. Gamika also differs in its use of simulated physics. That is, while those systems define explicit movement rules for in-game objects, Gamika specifies only the physical properties of the objects and the environment, from which movement emerges. This reliance on emergence changes how the space is navigated: on one hand, it reduces the ease of finding specific designs that users may have in mind, but on the other, it increases the chances of the parameters combining in unanticipated and serendipitous ways.

One of our ultimate aims is for Gamika to generate entire casual games automatically, and the work presented on automatically fine-tuning game levels is the first step towards this. Automatic generation of game levels has been looked at for both VGDL [4] and PuzzleScript [5] [6], in addition to a variety of systems generating Super Mario Bros. levels [10]. Nelson and Mateas [8] formally modularise recombinable game mechanics, so that users can define novel game variants and get automated feedback on properties such as playability, via automated theorem proving. Cook et al.'s ANGELINA system is very influential in automated game generation [11]. There has also been work on defining generative spaces of games in terms of the games' semantic and narrative content, mapping sprites to a relatively fixed set of mechanics [7], [9]; here we focus on the space of mechanics rather than theme or meaning, but in future work Gamika may branch out from abstractly themed games to include such elements.

As described in section V, Gamika provides a configurable automated playtester to aid designers in making game levels. For such playtesting, it would initially seem natural to choose a method that is capable of playing any general game (or at least any game expressible in the system). The results of the

General Video Game AI (GVGAI) Competition [12] suggest that game tree search approaches, particularly Monte Carlo Tree Search (MCTS), are strong in this area. As well as the grid-based games that have been the focus of the GVGAI competitions thus far, MCTS has been demonstrated to work well for physics-based games [13]. In the domain of game generation, the Mechanic Miner system by Cook et al. [14] uses breadth-first search as a playtester to evaluate generated game mechanics. Reinforcement learning methods such as deep  $Q$  learning have also been demonstrated to work well for general videogame playing [16].

One of our design goals is for the playtester's decision-making process to be transparent to the user so that they can design AI-bots to play their game level, and enjoy watching the bot play. Game tree search is conceptually simple, but rarely yields a satisfying explanation as to why the AI player chose a particular action. Reinforcement learning is even less comprehensible to the non-expert. For our purposes, easily explained tactics and strategies are preferable to a trained "black box". Another design goal is the ability to generate or tweak a game with respect to a fixed strategy. The success of search and learning based approaches in general game playing is precisely because they adapt to the game at hand, which puts them at odds with this aim. Thus, as described below, we opted for a simple rule-based player, whose rules are parameterised and exposed through the user interface. Whilst the playtester is not particularly sophisticated in computational intelligence terms, it is well-suited to our aims with the Gamika project.

### III. A BREAKDOWN OF PHYSICS-BASED CASUAL GAMES

The Gamika tool is an iOS application developed in the Swift programming language using the SpriteKit game development library's built-in 2D physics engine, a modified version of the well-known Box2D engine (box2d.org). Each Gamika game is an ordered list of game levels, where a level is a triplet of an optional text explaining the rules for players; an optional drawing represented as a vector graphic; and a list of numerical values for a set of 284 parameters which define how a set of game objects look, move and interact with each other and with the player's touches. Starting by analysing a number of classic arcade games like Frogger and Asteroids, and supplementing these with analyses of novel games, we have organically grown the set of parameters. Parameters were added until we were satisfied that the game levels they can define are sufficiently diverse, interesting and engaging.

There are three classes of physics object in a level: multiple *friend* objects, multiple *foe* objects, and the single *controller*. The naming of objects as friends/foes allows designers to attach meaning to game objects, to more easily manage the large number of game parameters, but the user is at liberty to ignore this. All objects have a rigid physics body and a collision shape. One option for the controller is for it to be a decorative abstract art image generated by the ELVIRA evolutionary art system [17] and the genomes and a set of thumbnails of 1,000 supplied images. The images cover different styles, giving a wide range of choices, and each genome can be mutated or more carefully varied, so designers have a good chance of expressing an aesthetic of interest to them. The first set of numerical game parameters define the mathematical functions, blurring regime and post hoc transformations which dictate



how the art asset is generated, as per [17]. Alternatively, the controller can be a hand-drawn shape created within the design interface, which can optionally be combined with an art image, which opens up many more aesthetic possibilities. The contour of the controller image is traced to determine its physical shape, hence the choice of the controller is not purely aesthetic, but can also be an important factor in gameplay. The numerical parameters control the following aspects:

- The **properties** of the friends/foes. These control the object size, shape, colour, sprite image, how the calculation of their boundary is performed, and some physical properties such as their restitution (bounciness), mass and damping.
- The **lighting effects** applied to the background and game objects. These control effects such as spotlights, ambient light, the calculation of a normal map for the background and controller image, and the lit appearance of the friend/foes.
- The **spawning** regime for the friends/foes. These control the spawning positions within time-varying ranges, spawn frequencies, total number of each object allowed, and some spatial constraints for the spawning, such as minimum/maximum distances from each other and being fixed on a grid.
- The **movement** of friend and foe objects both at the start of the level and during the game. These control the force fields acting on the objects via directions and strengths, with parameters for noise, friction and angular/linear drag on objects, speed limits, whether objects can rotate or not, and how joints such as pins, springs and sliders act on the objects.
- The **collisions** between friends, foes and the controller. These control whether objects stick, bounce, explode and/or change types on collisions, and timings for these, which screen walls are active and how bouncy they and the controller are, as well as how clusters form and when they explode.
- The **user interactions** which move the controller and affect the friends/foes. Tapping, dragging and swiping actions are caught, and the controller can be attached by springs, pins and horizontal/vertical sliders, and can be subject to movement and/or rotation by player touches. Player taps can also explode, halt, reverse or change the type of the friends/foes, and taps on the background can spawn more objects.
- The **progress calculations** which alter three counters: score, health and lives (naming suggestions the designer can choose to ignore). Each calculation adds up to five measures prescribed by events on friends/foes. Events include collisions, explosions, spawning, staying on screen, clusters being formed, and objects entering scoring regions.
- The **end-game criteria** which dictate how the progress calculations and/or game duration terminate the game. These control what constitutes a win or loss, how the overall score is calculated and whether high-scores are recorded.

Many games can be described in terms of an object with physical properties and spawning rules. Thus, they can be expressed in Gamika. As an illustration, a facsimile of the classic arcade game Frogger can be described as follows. Friends are spawned at the left and foes at the right of the screen and are attracted towards the opposite side by a force-field. The controller is drawn (possibly as a frog) at the bottom of the screen, and can be moved in the four

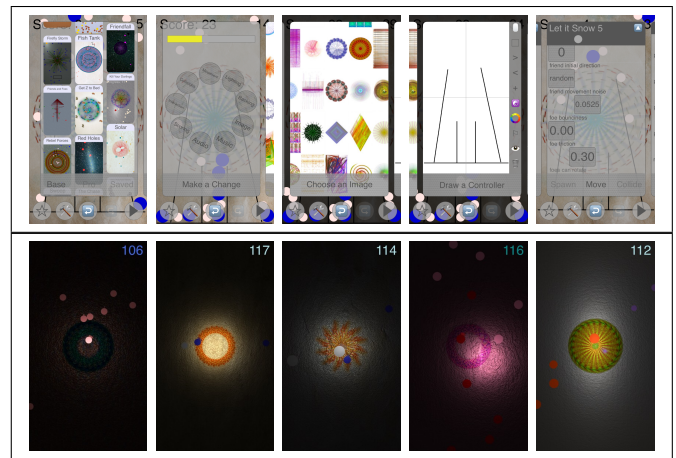


Fig. 1. (a) *basecamp*, mutation, art asset, drawing and parameters screens from the Gamika design interface (b) Five levels in the *Pendulands* game.

cardinal directions by the player swiping the screen. If the controller collides with a friend or a foe, the game is lost. If the controller reaches the top of the screen, the game is won. Likewise, a simple Asteroids version has the controller (ship) moving towards where the player taps. Friends (bullets) are spawned continuously at the controller's position and move in the direction in which the controller points. At the start, foes (asteroids) are spawned randomly on screen and move subject to a noise field. Friends and foes are destroyed upon collision. If the controller collides with a foe, the game is over.

The design interface includes a *basecamp* screen, from which the user can load one of several preset games, including new original games and homages to classic ones. The main mode of use for the app is that the user loads a *basecamp* game and modifies it, from a simple re-skinning to a complete change of game mechanics. We decided not to provide the option of creating a new game from scratch, to avoid the discouraging effect of the blank canvas. However, it is important to note that no part of the game design is concealed from the user, so any game that can be expressed in Gamika can be created by modifying any existing game.

The *basecamp* screen is the first showing in figure 1. The design interface also includes a mutation screen, shown second in figure 1. One advantage to having games decomposed into numerical parameters is that they can easily be mutated. The mutation screen has nine buttons arranged in a dial, with each button representing a different aspect of a game level, such as movement, collisions, lighting, etc. The chosen button and the degree by which the dial is rotated dictates which design aspects are varied, and by how much, with bespoke schemes used when the mutation is initiated. Figure 1 also shows the art asset selection screen where the chosen image is generated from an underlying genome as per [17], and the drawing screen, which has a number of editing facilities. Finally, figure 1 shows one of the screens where parameters can be altered directly via a slider interface.

Figure 1 shows five levels from a game called *Pendulands*. Here, the designer (second author) had the idea of having balls spawning from the sides and being attracted to the centre, with any pair of balls exploding if they collide. Players have to move the controller (a circular abstract art image) in order to be underneath the balls for long enough for them to stick to the

controller. Once stuck, these collected balls are then under the player's control and need to be protected. The designer used the mutation screen to explore player interaction and lighting design. Two mutations led to important aspects of the game: (i) the player interaction mutation added a spring to the game controller, so that when the player lets go, it springs back to the centre, usually causing collected balls to explode against uncollected ones, and setting progress back (ii) the lighting mutation reduced the ambient lighting significantly, and added a spotlight to the position of the player's finger.

Given that dragging the controller directly causes the player's finger to obscure it, the best way to play the game is to drag the controller off-centre. Dragging off-centre means that the controller is always in the shadows, as the only light in the game is at the player's finger, which is kept in contact with the screen at all times to avoid the controller springing back. The designer built on this aspect of gameplay with a grungy aesthetic, achieved via the use of normal maps. Once the first level of *Pendulands* was made, fifteen more levels were produced relatively quickly, sometimes with only 10 minutes of effort. Each level has similar elements, including a circular controller, dark/grungy aesthetic and the need to catch five balls on the controller to complete the level. However, each level includes a different game mechanic, achieved by tweaking the nature of the balls, where they spawn, what happens when they collide (bounce, stick or explode), how fast they are and how they move. *Pendulands* is one of around 30 games that have been developed with Gamika so far.

#### IV. RUNNING EXAMPLE: LET IT SNOW

For the remainder of the paper, we will use a running example of a particular game called *Let it Snow* to describe how advanced functionality in Gamika provided automated assistance to the designer (second author) when making the game. In *Let it Snow* levels, see figure 2, the controller is a drawn item, with the art asset being used purely as a backdrop. The player can drag the controller (which we call 'jiggling' it). Blue (rain/foe) and white (snow/friend) balls are spawned in random positions at the top of the screen and fall towards the bottom. Balls are spawned at a rate of three white and three blue per second until 20 of each colour are on-screen. Balls bounce off those of the opposite colour, the controller and the screen walls. When two balls of the same colour collide, they stick together. If a cluster of four or more balls of the same colour forms, the cluster explodes. An exploding ball causes a new ball of the same colour to spawn at the top of the screen.

Players gain one point for each white ball that explodes and lose one point for each blue ball that explodes. Players can tap blue balls, which causes the tapped ball to explode, tapping white balls has no effect. While tapping a blue ball will lose the player a point, it may cause a cluster of whites to form and explode which will score more than the loss. Jiggling the controller can unlock occasional stalemates. The aim of the game is to reach a score of 100 points in a time which beats the player's current personal best. If the level is completed within 60 seconds, the player can move on to the next level.

Let it Snow is a difficult, fast moving, reaction game, and it takes some practice to become proficient at it. Novice players tend to fail to get control of the game and scores can plummet.

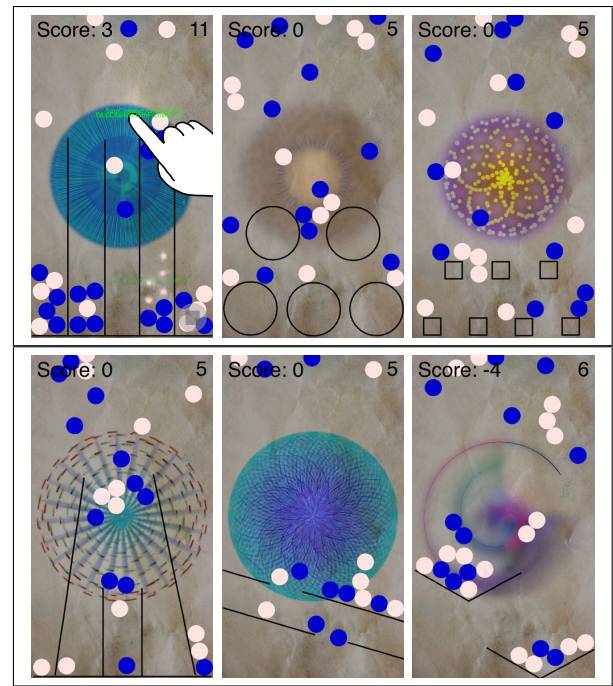


Fig. 2. Screenshots from *Let It Snow* level 1 showing the auto-playtester playing the level, and from un-fixed levels 2 to 6.

An interesting aspect of the gameplay is that, from time to time, an arrangement of balls arises where no more clusters are formed; hence the game comes to a halt as no new balls are spawned. In these quiescent moments, the game takes on a puzzle character, as the player can take their time to decide which blue ball to destroy in order to gain the most points through whites forming clusters. The designer discovered that a good strategy for players is to concentrate on getting to these quiescent moments, then carefully keeping control of the situation through selectively destroying blue balls. A strategy to make the game reach quiescent moments is to react to any potential cluster of four blues and stop it forming by destroying one of them. If the player is successful in this, eventually the blues and whites become distributed over the five columns of the controller, locked out of clusters by balls of the opposite colour.

A winning strategy is to get all the blue balls locked in singletons, pairs and triplets at the bottom of the screen. Then only white balls spawn and land on those exposed above the locked blues in such a way that they continuously form groups of four and thus allow more whites to spawn. At this stage, as only whites are being spawned, the game has the look of snowing (hence the game's name), and players can sit back and watch the score increase rapidly as it snows, although expert players may use a more proactive tactic. An expert player can usually reach 100 points within 60 to 90 seconds, with novices often taking more than three minutes to complete level 1. *Let it Snow* is quite an addictive and engaging casual game, but presented a number of difficulties for the designing of new levels. We explore how automation has helped address some of these difficulties in section VI, after we describe how automated playtesters can be configured to play the game.

## V. AUTOMATIC PLAYTESTING

To support designers in making casual games, we have implemented a configurable automated playtester which can be parameterised on the device to describe an AI-bot to play game levels. Currently, the exposed playtester design parameters are not as extensive as those for level design and are rather focused on playing *Let it Snow*; we intend to grow the parameter set in future work. The automated playtester “ticks” at a frequency of once per game frame (normally 60 per second), and analyses the state of the game. The designer chooses a number of *tactics* for the bot to employ, each formed of a condition or a pattern to check for, and an in-game action to execute if the pattern is found. The tactics are arranged in priority order, i.e., the  $(i + 1)$ th tactic takes effect only if the  $i$ th tactic does not. The following are the tactics used in a *Let it Snow* playtester, which through design and experimentation we have found to be a human-competitive player for the game.

- **Stop blue clusters from forming.** If two blue balls are close to one another, and they are not already in a cluster, and if the balls colliding would form a cluster of four or more, then tap the faster moving ball. “Close to one another” means that either the two balls are currently within a threshold distance of each other, or they are predicted to be before the next AI tick.
- **Pop blues blocking whites.** If a blue ball is touching two or more white balls which are in distinct clusters, and those clusters contain four or more balls in total, and the blue ball has met this criterion continuously for 1 second (a value which is changeable), then tap the blue ball.
- **Tap if quiescent.** If 2 seconds (a value which is changeable) have passed with all balls having zero velocity, then tap a blue ball that is touching a large number of white balls and is itself either unclustered or in one of the smallest clusters.
- **Jiggle if whites are stuck.** If 5 seconds have passed without a white ball being able to spawn, shake the board in a small random direction.
- **Do nothing.** If no above conditions are met, do nothing.

The “perfect” version of the playtester exhibits super-human accuracy and reaction speeds. We also consider an “imperfect” playtester, with a limit imposed on the number of actions per second and random noise added to its tap positions. We analysed logs from three games played by the first author, and identified his maximum number of taps per 1-second window was 3 and the average distance from the centre of a tapped ball was 10 pixels. Thus, we restrict the imperfect player to 3 taps per second and add random noise with magnitude  $2 \times 10 = 20$  to its taps. Additional experiments using a small random time delay in bot actions proved to be far too detrimental to the bot’s performance: as implemented currently, the “stop blue clusters from forming” tactic results in the bot waiting until the last possible moment to tap the incoming blue, so even a small delay results in a high miss rate. A tactic that tried to anticipate blue clusters further in advance would likely solve this problem. We acknowledge that the above is a basic attempt at mimicking the limitations of human players, but it does serve to dull the automated player’s super-human qualities. Creating playtesters that more convincingly try to model human players such as [15], and possibly the user in particular, is a subject for future work.

As per our desire to make interacting with all aspects of Gamika enjoyable, we have visualised the playtester so the designer can see exactly how the bot is playing the game, and enjoy the experience. The first screenshot in figure 2 shows the auto-playtester’s simulated hand interacting with the game. We have also added a slider to the design interface which forces the auto-playtester to play at 1, 2, 4 or 8 times normal speed. The speed up is achieved by increasing the physics world speed in SpriteKit which unfortunately means that the playtester is exposed to proportionally fewer ticks, and the bot design had to be altered to cope with this. At higher speeds, therefore, the playtester’s ability to play the game is reduced, but we have found that at four times normal speed, it still plays at human-comparable levels. The speed up allows designers to more quickly understand game levels through playtesting, and speeds up the automated fine tuning, as described below.

## VI. CASE STUDY

While the designer of both *Pendulands*, described above, and of *Let it Snow* is the same (the second author), the design process has been quite different. For *Pendulands*, the initial level took around 2 hours to perfect, and subsequent levels took much less time – for some, it involved making some changes to the game mechanic and aesthetics, then playing the game around 10 times to test whether it was difficult enough, but not too difficult. *Let it Snow* is perhaps a more interesting game, with simple rules yet high difficulty, and both a puzzle and reaction element, and potentially addictive qualities, which may drive players to want to improve their high score.

However, the designer found making levels for *Let it Snow* much more difficult than for *Pendulands*. In particular, the random nature of the spawning of the balls introduced quite a difficult design problem. That is, the designer found that the game is heavily, but not entirely, luck-based, and observed situations where an extremely lucky run of balls allows a novice player to achieve an expert-level score with hardly any interaction at all. Hence, it was difficult to tell whether it was possible to get better through practice, which is rather a pre-requisite for an enjoyable game of this type. It was for this reason that the designer configured the auto-playtester as per section V, and used it to analyse level 1, as described in the following subsection. Making new levels was also a challenge, and the designer relied on automated fine-tuning of 8 game parameters, guided by the auto-playtester, as described in the second subsection below.

### A. Automated Playtesting Analyses

Let it Snow level 1 is on the verge of being too affected by randomness to provide viable gameplay. Indeed, the designer struggled to understand whether the use of tactics helped get better scores, as it is possible to play quite skillfully using various tactics, yet on occasion still perform poorly due to bad luck. Hence, the designer first used Gamika to produce game logs of both a human player (author 1) and the “perfect” auto-playtester. These logs are presented in figures 3(a) and (b) respectively. We see that in the human log, the score (top blue graph) dips during the early stage of the game, as the player struggles to get control of the level. Control is initially achieved through a quiescent moment, and these are achieved throughout the game. This was a trend seen in multiple logs

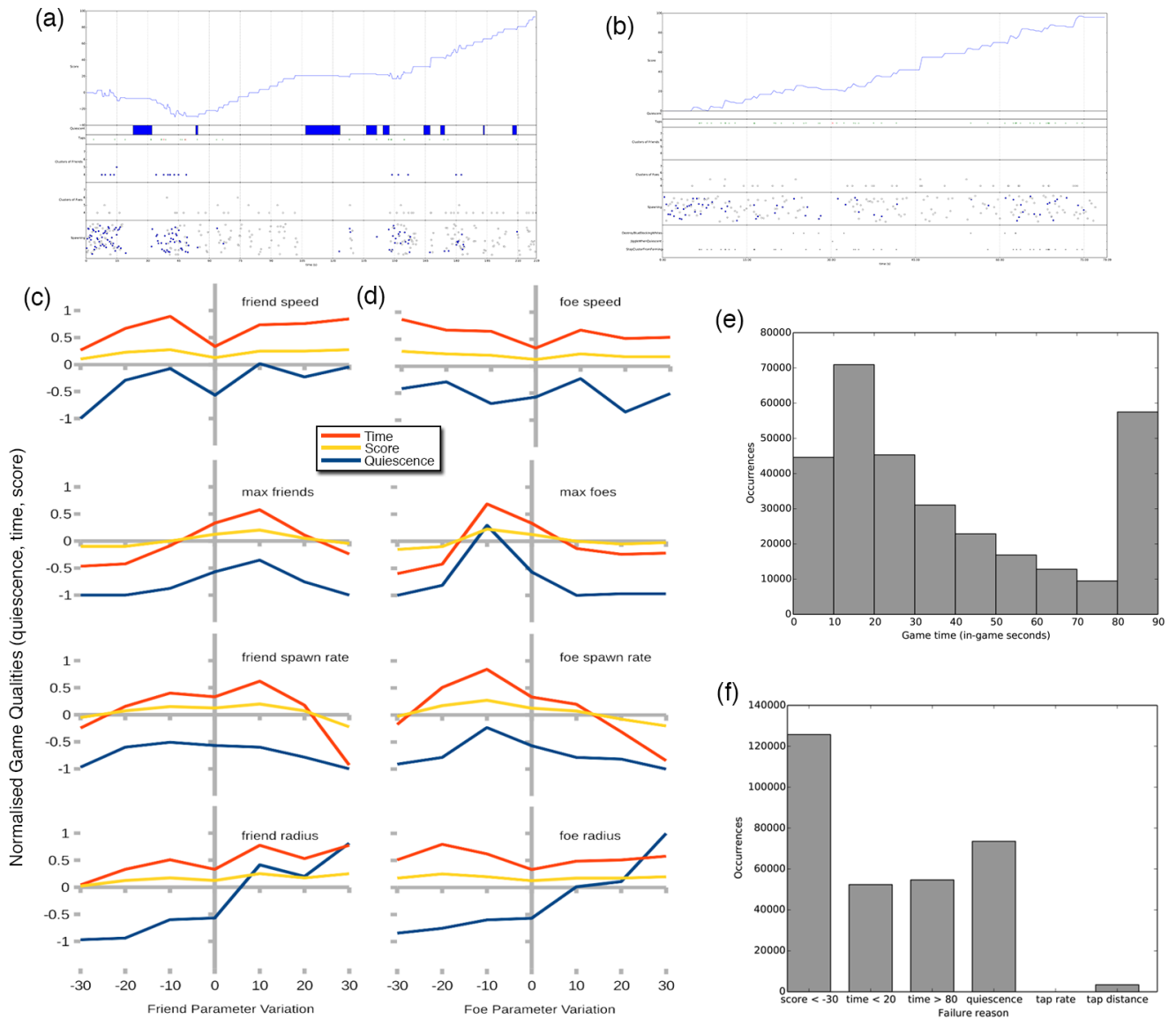


Fig. 3. (a) Log of human playing *Let it Snow* level 1 (b) Log of “perfect” auto-playtester playing *Let it Snow* level 1. (c)/(d) Analysis of the change in three normalised game qualities (quiescence, time and score) as four parameters for friend/foes are varied (e) Discretised game play time for the tuning sessions (f) Occurrences of each test failure in the tuning sessions

and confirmed the designer’s hunch that aiming for quiescence early on is a good tactic.

The auto-playtester logs show that the bot was able to complete the level with perfect application of the tactics described in section V above, i.e., without requiring the quiescent moments to reflect on the state of the game. Hence, the bot can play successfully in a different way to the human player, and it was encouraging that it played to a high standard (indeed, it completed level 1 in around 60 seconds on the second playing). The designer gained the biggest insight into the game when they compared on-screen the “imperfect” bot against the “perfect” one: double checking the logs confirmed that more skilful play does indeed lead to better completion times.

The designer decided that new levels would be created with just a change of drawn game controller and backdrop image, as in figure 2. The designer found with level 2, the way in which the balls bounced off the new controller (i.e.,

circles in level 2 rather than vertical lines in level 1), and the way the balls are collected in the gaps differently made the level feel sufficiently novel compared to the previous one. However, after designing level 2, he found that the game was even more difficult than level 1 because quiescent moments never happened. After some experimentation, the designer realised that to achieve quiescent moments, the number of balls allowed on-screen at any one time needed to be reduced. To the designer’s taste, this reduced the fun of the game, as it was less dynamic. On experimenting with other ball properties, namely the size, speed and spawning rate, the designer found that these properties could also be used to improve the game design.

The designer suspected that the relationship between the ball properties and aspects of the level such as completion time was non-linear. To investigate this, they returned to level 1 and used Gamika to perform a parameter sweep of the level. This was achieved by systematically altering the eight parameters (size, speed, spawning rate and maximum allowed on-screen



for both ball types) within the range  $-30\%$  to  $+30\%$  around their original value, using a  $10\%$  step-size and employing ten independent trials for each variation of a parameter. This resulted in a total of  $((8 \times 6) + 1) \times 10 = 490$  runs of each altered level, which took around 6 hours of processing on the device. For each of the parameters, Gamika calculated the mean value for the number of quiescent moments  $\mu$  (with a quiescent moment defined as a continuous stretch of time where no ball has a non-zero velocity), the score the player achieved  $s$  and the time until the level finished,  $\delta$ . The raw values were normalised as follows:  $s$  within the range of  $[-30, 50]$  points,  $\delta$  within the range of  $[0, 90]$  and  $\mu$  within the range of  $[0, \mu_{max}]$ , and mapped to the interval  $[-1, 1]$ .

Figures 3(c) and (d) show the results of the parameter sweep. As suspected, the non-linear nature in which the game design affects scoring, completion time and the number of quiescent moments is highlighted. The designer found that the parameter sweep was very informative and will no doubt help with future level design. In particular, the analysis highlighted certain sweet spots which can be exploited to fix faulty levels, e.g., if levels are too easy, increasing the foe speed by  $10\%$  would increase the average game completion time. One exception to non-linearity is the approximate positive correlation of ball size with the number of quiescent moments. This was surprising to the designer, as they had expected the opposite, as larger balls tend to have more collisions, hence more explosions and spawning and a more dynamic, less quiescent level. However, on investigation, it appears that smaller blue balls are less likely to get stuck in non-exploding clusters, as they have more room to move around in, which leads to clusters forming more often and less quiescence.

### B. Automated Fine Tuning of New Levels

We want to enable people to make experimental games with Gamika, without having to spend an inhibitive long time designing and playing them. To do so, we intend for the app to have intelligent search methods that can take a nearly-finished level and fine-tune various parameters so that the altered level passes a series of user-defined tests. In a baseline experiment to investigate the potential for this, the designer created levels 2 to 6 of *Let it Snow* as per figure 2. The levels were altered to end after 50 points, rather than the original 100 points, to improve testing speed, and a timeout of 90 seconds was applied. Using Gamika, the designer defined a test suite for new levels as follows: a level fails if (a) at any time, the score reaches  $-30$  or below, or (b) the bot completes a level in less than 20 seconds or more than 80 seconds, or (c) the game contained fewer than two quiescent moments, or (d) the tap rate was more than 3.5 per second, or (e) the average distance between consecutive taps was more than 200 pixels.

The designer specified that the size, speed, spawning rate and number-allowed parameters for both friends and foes should be the subject of random variation. These were chosen as the parameters that have the largest effect on gameplay, without having such a large effect as to make the game no longer recognisable as *Let it Snow*. For a given newly-designed level, Gamika randomly varies each of the eight parameters within  $-30\%$  to  $+30\%$  of its value for level 1. A trial involves a tweaked level being played three times at four times the normal speed by the “perfect” auto-playtester. If any of the

TABLE I. CURATION ANALYSIS FOR *Let it Snow* LEVELS 2 TO 6

Level	Trial	Mod.	Time	Prop.	Tot.
2	478	2	1h 19m	3/5	16
3	408	1	1h 04m	2/5	12
4	400	2	1h 00m	3/5	14
5	7531	3	17h 52m	2/5	6
6	402	1	0h 50m	3/5	11
av.	1843	1.6	4h 25m	2.4/5	11.8
av2	422	1.5	1h 3m	2.75/5	13.25

three playthroughs fails any of the above tests, the whole trial is discarded. If, however, all three playthroughs pass the tests, then the altered level is saved to the app, and a new alteration is sought, which continues until the user stops the process. We have found that these tests discard levels for being too easy or difficult, and also levels that cater somewhat to the super-human abilities of the playtester, which can be relentlessly accurate, where a person probably cannot. This testing set up is quite efficient, as the “perfect” bot is the fastest at completing a level, and the order of the tests means that many broken levels fail very fast, often within a few seconds.

We undertook the random approach to see whether successful tweaks are possible automatically and to set a baseline against which more intelligent search methods will be assessed in future. Not surprisingly, we found that the random approach was not particularly efficient, rejecting tens of thousands of tweaked games on average before finding one that passed the tests for three plays. Note that we ran these experiments overnight in parallel on a simulated desktop version of Gamika. Figure 3(e) portrays the proportion of games which finished in certain time bands. Note that those finishing between 80 and 90 seconds were those which timed out. We see that a large proportion of the games ended quickly, in 20 seconds or less (or 5 seconds at  $4\times$  physics speed). This is explained in figure 3(f), where the breakdown of the reasons why levels failed is given: the test to see whether a game is so difficult that the very able auto-playtester gets to a  $-30$  score is very effective, and this quickly rules out bad levels. The high proportion of games that fail due to lack of quiescence confirmed the designer’s view that producing a level to achieve such quiescent states was difficult. While the tap distance test was used to discard some trials, the tap rate was not.

To assess whether the random variation and auto-testing scheme works, the designer undertook a *curation analysis* (as introduced in [18]) of the tweaked *Let it Snow* levels. In particular, for each level, he played the first five output variations in the order in which they were produced. Each variation was played 10 times, and the designer decided whether the level was good enough to be added to the game, or should be discarded, recording the reasons for either decision. The *curation analysis* results are presented in table I. For level 2, the second (Mod)ified level shown to the designer, after 478 (Trial)s and a (Time) of 1 hour 19 minutes, was deemed the first one that was good enough. The (Prop)ortion of the first five modified levels shown to the designer which were deemed good was  $\frac{3}{5}$ , and these came from a (Tot)al crop of 16 generated over a 48 hour period. We see that the other levels have similar results, with level 5 being an exception. Here, only six levels passed the battery of tests in 48 hours and the first good one came after 17 hours 52 minutes. This level is clearly an outlier and suggests the controller drawn in this case admits a much smaller space of playable levels. Ignoring this outlier, the average waiting time for a good level

was around 1 hour, which, while clearly still too long to wait, is encouraging for a random search. The line marked av2 in table I provides averages for the four levels excluding level 5.

Of the levels that the designer saw but rejected, the main failure was that the level was too easy, encouraging too passive a playing style because the snowing moments happened too easily and the player was not actively able to improve their completion time. Some of the games were rated as very good by the designer. Sometimes this was because they had the feel of level 1, but for others, it was because they were quite novel, e.g., the second modified version of level 6 was relatively slow moving and had a greater emphasis on the puzzle element of the game, with very interesting quiescent moments.

## VII. CONCLUSIONS AND FUTURE WORK

With the work presented here, we believe that we have demonstrated in principle that novel, interesting and engaging casual games which are truly more than levels of an existing game world, or skinings of templates can be produced on a hand-held device without the requirement for coding. With the minor case study of the *Pendulands* game and the major case study of *Let it Snow*, we have shown that the Gamika app has much potential to help democratise game design so that broader sections of society can make digital games. *Let it Snow* is the kind of difficult to design game that requires automated assistance in producing levels. We showed that an automated playtester could be designed in Gamika that could play *Let it Snow* levels to a (super) human standard and that getting the AI-bot to play the levels on the device was (anecdotally) entertaining, and could elucidate valuable insights into game levels. We further showed that a random search for tweaks of game parameters, with levels tested by the playtester, could be used in semi-automated game design, where Gamika fine-tunes game mechanics to find suitable modifications of levels. The results from this approach for four of five new levels of *Let it Snow* were encouraging.

While we have shown in principle that the approach is viable, there is much work on all aspects which will need to be carried out before Gamika can be released commercially, which is our intention. In particular, we are currently undertaking the following improvements to Gamika:

- Further expanding the space of games available through the app, by implementing more intelligent behaviours in the friends and foes, possibly via the on-device specification of fitness functions and behaviour trees.
- Improving the user interface to the game design parameters, employing more drag-and-drop functionality and expanding what can be defined through the drawing interface.
- Making the automated playtester much more generic so that it can be configured to play a wide range of games. We also plan to experiment with an approach to training the playtester through play coupled with structured answering of questions, which will hopefully be entertaining, in a pedagogic way.
- Implementing more intelligent fine-tuning search techniques. The random approach has provided a baseline, but it takes far too long to find viable solutions. Hence, we have started work on a hill-climbing method and we will also investigate evolutionary and constraint solving approaches.

We agree with Liapis et al. [19] that videogame design is a killer application for Computational Creativity research [18], and we are very interested in Gamika becoming a creative game designer, much like the ANGELINA system [11]. Moreover, we believe that Computational Creativity is ready to have an impact on gaming culture, and we hope to help bring this about through the Gamika app, which will co-create games with designers from all backgrounds, complimenting in many people the joy of game playing with the joy of game design.

## ACKNOWLEDGMENTS

This work is funded by EC FP7 grant 621403 (ERA Chair: Games Research Opportunities). We are grateful to the many people who tested Gamika and provided valuable feedback.

## REFERENCES

- [1] J. Juul, *A Casual Revolution: Reinventing Video Games and their Players*. MIT Press, 2009.
- [2] K. Compton and M. Mateas, "Casual creators," in *Proceedings of the Sixth International Conference on Computational Creativity*, 2015.
- [3] T. Schaul, "An extensible description language for video game," in *IEEE Trans. Comp. Intell. AI Games*, 2014.
- [4] T. Nielsen, G. Barros, J. Togelius, and M. Nelson, "Towards generating arcade game rules with VGDL," in *Proceedings of the IEEE Conference on Computational Intelligence in Games*, 2015.
- [5] C.-U. Lim and F. Harrell, "An approach to general videogame evaluation and automatic generation using a description language," in *Proceedings of the IEEE Conference on Computational Intelligence in Games*, 2014.
- [6] A. Khalifi and M. Fayek, "Automatic puzzle level generation: A general approach using a description language," in *Proceedings of the First Workshop on Computational Creativity and Games*, 2015.
- [7] M. J. Nelson and M. Mateas, "An interactive game-design assistant," in *Proc. of the Intl. Conf. on Intelligent User Interfaces*, 2008.
- [8] M. J. Nelson and M. Mateas, "Recombinable game mechanics for automated design support," in *Proc. of the AIIDE Conference*, 2008.
- [9] M. Treanor, B. Blackford, M. Mateas, and I. Bogost, "The micro-rhetorics of Game-o-Matic," in *Proceedings of the Procedural Content Generation Workshop*, 2012.
- [10] B. Horn, S. Dahlsgog, N. Shaker, G. Smith, and J. Togelius, "A comparative evaluation of level generators in the Mario AI framework," in *Proceedings of the Foundations of Digital Games Conference*, 2014.
- [11] M. Cook, S. Colton, and J. Gow, "The ANGELINA videogame design system, parts I and II," *IEEE Trans. Comp. Intell. AI Games*, 2016.
- [12] D. Perez, S. Samothrakakis, J. Togelius, T. Schaul, S. Lucas, A. Couetoux, J. Lee, C.-U. Lim, and T. Thompson, "The 2014 general game playing competition," *IEEE Trans. Comp. Intell. AI Games*, 2015.
- [13] D. Perez, E. J. Powley, D. Whitehouse, P. Rohlfshagen, S. Samothrakakis, P. I. Cowling, and S. M. Lucas, "Solving the Physical Travelling Salesman Problem: Tree Search and Macro-Actions," *IEEE Trans. Comp. Intell. AI Games*, 2013.
- [14] M. Cook, S. Colton, A. Raad, and J. Gow, "Mechanic miner: Reflection-driven game mechanic discovery and level design," in *Proceedings of the EvoGames Workshop*, 2013.
- [15] S.E. Gaudl, J.C. Osborn, and J.J. Bryson, "Learning from Play: Facilitating Character Design Through Genetic Programming and Human Mimicry," in *Progress in Artificial Intelligence: EPIA 2015*, 2015.
- [16] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, 2015.
- [17] S. Colton, M. Cook, and A. Raad, "Ludic considerations of tablet-based evo-art," in *Proceedings of the EvoMusArt Workshop*, 2011.
- [18] S. Colton and G. Wiggins, "Computational Creativity: The final frontier?" in *Proceedings of the 20th ECAI*, 2012.
- [19] A. Liapis, G. Yannakakis, and J. Togelius, "Computational game creativity," in *Proceedings of the Fifth International Conference on Computational Creativity*, 2014.



# Evaluating Real-Time Strategy Game States Using Convolutional Neural Networks

Marius Stanescu, Nicolas A. Barriga, Andy Hess and Michael Buro

Department of Computing Science

University of Alberta, Canada

{astanesc|barriga|athess|mburo}@ualberta.ca

**Abstract**—Real-time strategy (RTS) games, such as Blizzard’s StarCraft, are fast paced war simulation games in which players have to manage economies, control many dozens of units, and deal with uncertainty about opposing unit locations in real-time. Even in perfect information settings, constructing strong AI systems has been difficult due to enormous state and action spaces and the lack of good state evaluation functions and high-level action abstractions. To this day, good human players are still handily defeating the best RTS game AI systems, but this may change in the near future given the recent success of deep convolutional neural networks (CNNs) in computer Go, which demonstrated how networks can be used for evaluating complex game states accurately and to focus look-ahead search.

In this paper we present a CNN for RTS game state evaluation that goes beyond commonly used material based evaluations by also taking spatial relations between units into account. We evaluate the CNN’s performance by comparing it with various other evaluation functions by means of tournaments played by several state-of-the-art search algorithms. We find that, despite its much slower evaluation speed, on average the CNN based search performs significantly better compared to simpler but faster evaluations. These promising initial results together with recent advances in hierarchical search suggest that dominating human players in RTS games may not be far off.

## I. INTRODUCTION

The recent success of AlphaGo [1], culminating in the 4-1 win against one of the strongest human Go players, illustrated the effectiveness of combining Monte Carlo Tree Search (MCTS) and deep learning techniques. For AlphaGo, convolutional neural networks (CNNs) [2], [3] were trained to mitigate the prohibitively large search space of the game of Go in two ways: First, a policy network was trained, using both supervised and reinforcement learning techniques, to return a probability distribution over all possible moves, thereby focusing the search on the most promising branches. Second, MCTS state evaluation accuracy was improved by using both value network evaluations and playout results.

In two-player, zero-sum games, such as Chess and Go, optimal moves can be computed by using the minimax rule that minimizes worst case loss. In theory, these games can be solved by recursively applying this rule until reaching terminal states. However, in practice completely searching the game tree is infeasible, the procedure must be cut short, and an approximate evaluation function must be used to estimate the value of the game state. Because states closer to the end of the game are typically evaluated more accurately, deeper search produces better moves. But as game playing agents

often have to make their move decision under demanding time constraints, great performance gains can be achieved by improving the evaluation function’s accuracy.

The size of the state space of the game of Go, although much larger than that of Chess, is tiny in comparison to real-time strategy (RTS) games such as Blizzard’s StarCraft. In the game of Go, at every turn, a single stone can be placed at any valid location on the 19×19 board and the average game length is around 150 moves. In RTS games, each player can simultaneously command many units to perform a large number of possible actions. Also, a single game can last for tens of thousands of simulation frames, with possibly multiple moves being issued in each one. Moreover, RTS game maps are generally much larger than Go boards and feature terrain that often affects movement, combat, and resource gathering. Therefore, for RTS games, good state evaluations and search control, such as using policy networks, plays an even greater role.

CNNs are adept at learning complex relationships within structured data due to their ability to learn hierarchies of abstract, localized representations in an end-to-end manner [3]. In this paper we investigate the effectiveness of training a CNN to learn the value of game states for a simple RTS game and show significant improvement in accuracy over simpler state-of-the-art evaluations. We also show that incorporating the resulting learned evaluation function into state-of-the-art RTS search algorithms increases agent playing strength considerably.

## II. RELATED WORK

Search based planning approaches have had a long tradition in the construction of strong AI agents for abstract games like Chess and Go, and in recent years they have progressively been applied to modern video games, especially the RTS game StarCraft. This is a difficult endeavor due to the enormous state and action spaces, and finding optimal moves under tight real-time constraints is infeasible for all but the smallest scenarios. Consequently, the research focus in this area has been on reducing the search space via different abstraction mechanisms and on producing good state evaluation functions to guide this search effort.

In this section we briefly discuss some of these attempts, starting with various methods used for state evaluation in

RTS games. We then present recent research on deep neural networks and their use in game playing agents.

#### A. State Evaluation in RTS Games

Playing RTS games well requires strategic as well as tactical skills, ranging from building effective economies, over deciding what to build next based on scouting results, to maneuvering units in combat encounters. In RTS game combat each player controls an army consisting of different types of units and tries to defeat the opponent's army while minimizing its own losses. Because battles have a big impact on the result of RTS games, predicting their outcome accurately is very important, especially for look-ahead search algorithms.

A common metric for estimating combat outcomes is LTD2 [4], which is based on the lifetime damage each unit can inflict. LTD2 was used, in conjunction with short deterministic playouts, for node evaluation in alpha-beta search to select combat orders for individual units [5]. A similar metric was later used as state evaluation, this time combined with randomized playouts [6], [7].

Likewise, Hierarchical Adversarial Search [8] requires estimates of combat outcomes for state evaluation and uses a simulator for this purpose. However, because simulations become more expensive as the number of units grows, faster prediction methods are needed. For instance, a probabilistic graphical model trained on simulated battles can accurately predict the winner [9]. This model, however, has several limitations such as not modeling damaged units and not distinguishing between melee and ranged combat. Another model, based on Lanchester's attrition laws [10], does not have such shortcomings. It takes into account the relative strength of different unit types, their health and the fact that ranged weapons enable units to engage several targets without having to move, which causes a non-linear relationship between army size differences and winning potential. After learning unit strength values offline using maximum likelihood estimation from past recorded battles, this improved model has been successfully used for state evaluation in a state-of-the-art RTS search algorithm [11].

All mentioned approaches focus on a single strategic component of RTS games, i.e. combat, and lack spatial reasoning abilities, ignoring information such as unit positions and terrain. Global state evaluation in complex RTS games such as StarCraft has been less successful [12], likely due to the limited expressiveness of the linear model used.

#### B. Neural Networks

In recent years deep convolutional neural networks (CNNs) have sparked a revolution in AI. The spectacular results achieved in image classification [3] have led to deep CNNs being effectively applied to a wide range of domains. For vision tasks, CNNs have been applied to object localization [13], segmentation [14], facial recognition [15], super-resolution [16] and camera-localization [17] to name just a few examples, all the while continuing to make further progress in image classification [18]. Deep CNNs have also

been successfully applied to tasks as diverse as natural language categorization [19], [20], translation [21] and algorithm learning [22].

Deep CNNs owe their success to their ability to learn multiple levels of abstraction, each one building upon abstractions learned in previous layers. More specifically, deep CNNs learn a hierarchy of spatially invariant, localized representations, each layer aggregating and building upon representations in previous layers toward the combined goal of minimizing loss [13].

There is a long history of using simple linear regression and shallow neural networks to construct strong AI systems for classic board games such as Backgammon and Othello [23]. However, scaling up state evaluations to more complex games such as Go only became possible when it was discovered how to effectively train weights in deep neural networks, which can be considerably more expressive than shallow networks with the same number of weights [24].

Since then CNNs have been successfully used to play Atari video games with a policy network trained by supervised learning, using training data generated by a slow but strong UCT player [25]. Similar networks have been trained with reinforcement learning [26], [27]. Most remarkable, however, is the recent 4-1 win of AlphaGo [1], a deep CNN based Go playing program, over one of today's best Go players Lee Sedol. AlphaGo combines MCTS with deep CNNs for state evaluation and move selection that were trained by supervised and reinforcement learning.

This historic accomplishment sparks hope that CNNs can also be used for even more complex tasks, such as playing real-time games with imperfect information — a domain still dominated by human players.

### III. A NEURAL NETWORK FOR RTS GAME STATE EVALUATION

In this section we describe the dataset, the neural network structure and the procedure used for training a state evaluation network for  $\mu$ RTS<sup>1</sup>, a simple RTS game designed for testing AI techniques.  $\mu$ RTS provides the essential features of an RTS game: it supports four unit and two building types, all of them occupying one tile, and there is only one resource type. The game state is fully observable.  $\mu$ RTS supports configurable map sizes, commonly ranging from 8×8 to 16×16 in published papers. The game user interface and details about the unit types are shown in Figure 1.  $\mu$ RTS comes with a few basic scripted players, as well as search based players implementing several state-of-the-art RTS search techniques [6], [28], [7], making it an useful tool for benchmarking new AI algorithms.

The purpose of the neural network we describe here is to approximate the value function  $v^*(s)$ , which represents the win-draw-loss outcome of the game starting in state  $s$  assuming perfect play on both sides.

<sup>1</sup><https://github.com/santiontanon/microrts>

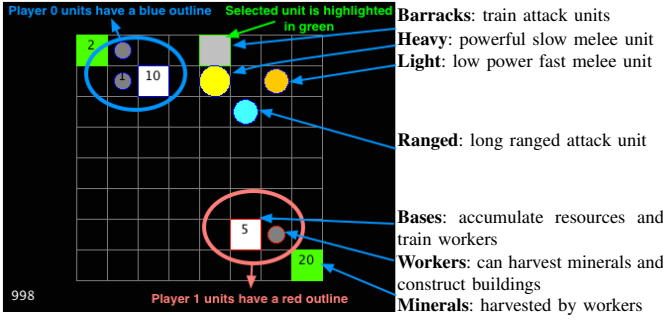


Fig. 1. Screenshot of  $\mu$ RTS, with explanations of the different in-game symbols.

In practice we have to approximate this value function with  $v_\theta$ , for instance by using a neural network with weights  $\theta$ . These weights are trained by regression on state-outcome pairs  $(s, w)$ , using stochastic gradient descent to minimize the mean squared error between the predicted value  $v_\theta(s)$  and the corresponding outcome  $w$ . The output of our network will be the players' probabilities of winning the game when starting from the input position.

#### A. Data

The dataset used for training the neural network was created by playing round-robin tournaments between 15 different  $\mu$ RTS bots, 11 of which are included in the default  $\mu$ RTS implementation. The other 4 are versions of the Puppet Search algorithm [11]. Each tournament consists of  $(15 \times 14)/2 = 105$  matches. One  $8 \times 8$  map was used, with 24 different initial starting conditions. All scenarios start with one base and one worker for each player, but with different, symmetric, initial positions. These tournaments were played under four different time limits: maximums of 100ms, 200ms, 100 playouts and 200 playouts per search episode. In total  $105 \times 24 \times 4 = 10\,080$  different games were played from which draws were discarded ( $\approx 8\%$ ).

Predicting game outcomes from data consisting of complete games leads to overfitting because while successive states are strongly correlated, the regression target is shared for the entire game. To mitigate the problem, the authors of AlphaGo [1] add only a single training example  $(s, w)$  to the dataset from each game. Because we have significantly less data (10 thousand vs. 30 million episodes), we chose to sample 3 random positions from each game. As a result, for game  $i$  we add  $\{(s_{i1}, w_i), (s_{i2}, w_i), (s_{i3}, w_i)\}$  to the dataset, and slightly over 25 000 positions are generated.

The dataset was split into a test set (5 000 positions) and a training set (the remaining 20 000 positions). Finally, the training set was augmented by including all reflections and rotations of each position for a total of 160 000 positions.

#### B. Features

Each position  $s$  is preprocessed into a set of  $8 \times 8$  feature planes. These features correspond to the raw board representation and contain information about each tile of the  $\mu$ RTS

TABLE I  
INPUT FEATURE PLANES FOR THE NEURAL NETWORK.

Feature	# of planes	Description
Unit type	6	Base, Barracks, worker, light, ranged, heavy
Unit health	5	1, 2, 3, 4, or $\geq 5$
Unit owner	2	Masks to indicate all units belonging to one player
Frames to completion	5	0–25, 26–50, 51–80, 81–120, or $\geq 121$
Resources	7	1, 2, 3, 4, 5, 6–9, or $\geq 10$

map: unit ownership and type, current health points, game frames until actions are completed and resources.

All integers, such as unit health points, are split into  $K$  different  $8 \times 8$  planes of binary values using the one-hot encoding. For example, five separate binary feature planes are used to represent whether an unit has 1, 2, 3, 4 or  $\geq 5$  health points. The full set of feature planes is listed in Table I.

#### C. Network Architecture & Training Details

The input to the neural network is an  $8 \times 8 \times 25$  image stack consisting of 25 feature planes. There are two convolutional layers that pad the input with zeros to obtain a  $10 \times 10$  image. Each then is convolved with 64 and respectively 32 filters of size  $3 \times 3$  with stride 1. Both are followed by leaky rectified linear units (LReLU) [29], [30]. A third hidden layer convolves 1 filter of size  $1 \times 1$  with stride 1, again followed by an LReLU. Then follow two fully connected (dense) linear layers, with 128 and 64 LReLU units, respectively. A dropout ratio of 0.5 is applied to both fully connected layers. The output layer is a fully connected layer with two units, and a softmax function is applied to obtain the winning probabilities for player 0 and player 1 ( $P(p_0)$  and  $P(p_1)$ ). All LReLU units have negative slope of  $\alpha = -1/5.5$ . The resulting architecture is shown in Figure 2.

Our architecture was motivated by current trends toward the use of small filter sizes ( $\leq 3 \times 3$ ), few (or no) pooling layers, and same-padded convolution (multiple layers of the same width and height, each layer padded with zeros following convolution) [31], [1]. We were also guided by the principle of gradually decreasing the dimension of internal representations as one moves from input toward task; one example being the reduction from 64 to 32 filters, another being the use of  $1 \times 1$  convolutions for dimensionality reduction [18]. This principle can also be seen in the fully connected layers. LReLU units were used following suggestions from [29] and [30].

Before training, we used Xavier random weight initialization [32] which equalizes signal variance. During training, the stepsize alpha was initialized to 0.00001 and was multiplied by 0.2 every 100K training steps. We used adaptive moment estimation (ADAM) with default values of  $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$  as suggested in [33]. The network was trained for 400K mini-batches of 64 positions, a process which took approximately 20 minutes on a single GPU to converge.

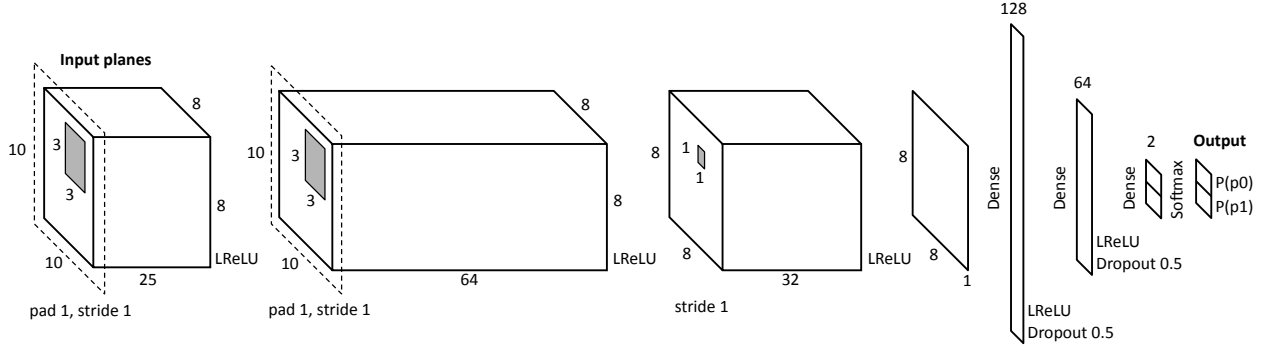


Fig. 2. Neural network architecture.

For training, we used the Python (2.7.6) interface to Caffe [34], utilizing CUDA<sup>2</sup> version 7.5 and cuDNN<sup>3</sup> version 4. The machine used for training the neural network had an Intel(R) Pentium(R) CPU G2120 3.10GHz processor, 8 GB RAM and one GeForce GTX 760 GPU (1152 cores and 4 GB memory) running Linux Mint 17.3.

#### IV. EXPERIMENTS AND RESULTS

All experiments that are reported below were performed on Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz with 8 GB RAM machines running Ubuntu 14.04. The test machines do not have CUDA capability 3 and the neural network computations were run solely on the CPU.  $\mu$ RTS software is implemented in Java and compiled and run with JDK 8u74.

##### A. Winner Prediction Accuracy

In the first set of experiments we compare the speed and accuracy of our neural network for evaluating game states with a Lanchester model [10] and a simple evaluation function that takes into account the cost and health points of units and the resources each player has. This is the default evaluation function that the  $\mu$ RTS search algorithms use. In equation 1 player indices are either 0 or 1:  $player \in \{0, 1\}$ .

$$eval(player) = E_{player} - E_{1-player} \quad (1)$$

In equation 2,  $R_p$  is the amount of resources a player currently has,  $W_p$  is a player's set of workers,  $R_u$  is the amount of resources each worker unit is carrying,  $C_u$  is the cost of unit  $u$ ,  $HP_u$  the current health points of unit  $u$ , and  $MaxHP_u$  its maximum health points.  $W_{res}$ ,  $W_{work}$ ,  $W_{unit}$  are constant weights.

$$E_p = W_{res}R_p + W_{work} \sum_{u \in W_p} R_u + W_{unit} \sum_{u \in p} \frac{C_u HP_u}{MaxHP_u} \quad (2)$$

Two versions of this simple evaluation functions were used: one with  $\mu$ RTS's default weights, and one optimized via

logistic regression on the same training set used for the neural network. The Lanchester model keeps the two resource terms of the simple evaluation function but revises the army's impact. While the contribution of the buildings is similar to equation 2, a new term is added for combat units:

$$E'_p = W_{res}R_p + W_{work} \sum_{u \in W_p} R_u + W_{base} \frac{HP_{base}}{MaxHP_{base}} + W_{barracks} \frac{HP_{barracks}}{MaxHP_{barracks}} + N_p^{(o-1)} \sum_{u \in p} \alpha_u \frac{HP_u}{MaxHP_u} \quad (3)$$

In equation 3,  $\alpha_u$  is a strength value unique to each unit type,  $N_p$  is the total number of units of player  $p$  and  $o$  is the Lanchester attrition order. For  $\mu$ RTS, our experiments suggest that an attrition order of  $o = 1.7$  works best on average if we had to choose a fixed order for all possible encounters. The four  $W$  and four  $\alpha$  constants (one for each unit type) are optimized with logistic regression.

Figure 3 shows the position evaluation accuracy of the neural network, compared to the default  $\mu$ RTS evaluation function, the optimized version and the Lanchester model, on the previously described test set of 5000 positions sampled from bot games. A scripted playout evaluation was also tested, in which the position is played until the end using the WorkerRush script, described in section IV-B, to generate moves for both players. Values of 1, 0 or -1 are returned, corresponding to a player 0 win, draw or loss, respectively. The WorkerRush script is the strongest of the four scripts described in the next section, and produces the most accurate winner prediction function, though slightly worse than the simple evaluation function. A random playout was also tried, but it performed even worse.

The neural network is consistently more accurate during the first half of the game. At the beginning of the game before any unit has been built, the simple evaluation function and the Lanchester model mostly predict draws, because they do not take positional information into account. During the second half of the game army balance is more relevant, and both the neural network and the Lanchester model perform better than the simple evaluation functions.

<sup>2</sup><https://developer.nvidia.com/cuda-toolkit>

<sup>3</sup><https://developer.nvidia.com/cudnn>

The average time needed for a single simple evaluation is  $0.012\mu s$ , the Lanchester model takes  $0.087\mu s$ , while a full network evaluation on the CPU takes  $147\mu s$ . This time includes processing the games state into feature planes, sending the data to a Python thread (on the same CPU core as the search algorithm), running a forward pass on the network and returning the outcome. The network evaluation takes close to two thirds of the time, around  $102\mu s$ . We tested the speed of the network evaluation on a GPU as well. On a mid-range NVIDIA GTX 760, the time is slightly shorter than the CPU-only version ( $118\mu s$ ).

However, processing only one position at a time does not take advantage of the pipelined GPU architecture. To measure potential gains of evaluating positions in parallel, we ran batches of 256 positions whose evaluation took  $10\,707\mu s$ , of which  $9\,985\mu s$  was spent on the CPU (feature planes) and  $722\mu s$  on the GPU, for an average of  $2.8\mu s$  of GPU time per evaluation. A search algorithm — like AlphaGo's — that can perform leaf evaluations asynchronously would benefit greatly from doing state evaluations on the GPU.

### B. State Evaluation in Search Algorithms

A second set of experiments compares the performance of four game tree search algorithms —  $\epsilon$ -Greedy MCTS, Naïve MCTS, AHTN-F and AHTN-P, described below — when using the simple evaluation function, the optimized evaluation function, the Lanchester model or the neural network for state evaluation.

The sixteen resulting algorithms played against the following eleven opponents provided by the  $\mu$ RTS implementation, all using default parameters and the simple  $\mu$ RTS evaluation function:

**WorkerRush:** a hardcoded rush strategy that constantly produces workers and sends them to attack.

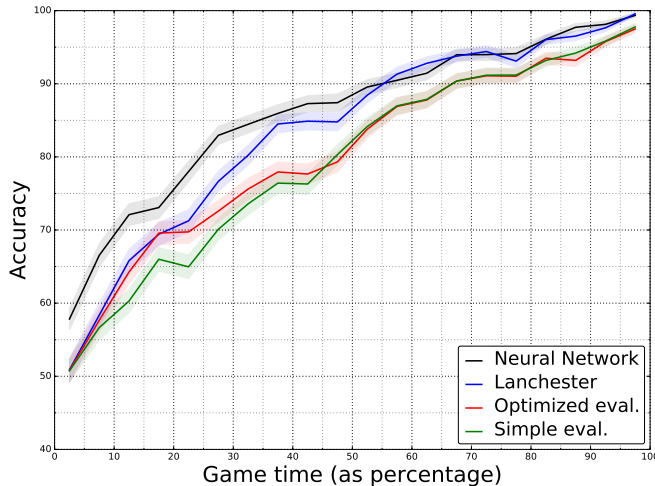


Fig. 3. Comparison of evaluation accuracy between the neural network,  $\mu$ RTS's built-in evaluation function, its optimized version and the Lanchester model. The accuracy at predicting the winner of the game is plotted against the stage of the game, expressed as a percentage of the game length. Results are aggregated in 5% buckets. Shaded area represents one standard error.

**LightRush:** builds a barracks, and then constantly produces *light* military units to attack the nearest target (it uses one worker to mine resources).

**RangedRush:** is identical to **LightRush**, except for producing *ranged* units.

**HeavyRush:** is identical to **LightRush**, except for producing slower but stronger *heavy* units.

**MonteCarlo(MC):** a standard Monte Carlo search algorithm: for each legal player action, it runs as many simulations as possible to estimate their expected reward.

**$\epsilon$ -Greedy MC:** Monte Carlo search, but using an  $\epsilon$ -greedy sampling strategy.

**Naïve MCTS:** Monte Carlo Tree Search algorithm with a sampling strategy specifically designed for games with combinatorial branching factors, such as RTS games. This strategy, called *Naïve Sampling*, exploits the particular tree structure of games that can be modeled as a Combinatorial Multi-Armed Bandit [6].

**$\epsilon$ -Greedy MCTS:** like NaïveMCTS, but using an  $\epsilon$ -greedy sampling strategy.

**MinMax Strategy:** for a set of strategies (WorkerRush, LightRush, RangedRush and Random), playouts are run for all possible pairings. It approximates the Nash equilibrium strategy using the minimax rule, whereby one player (Max) maximizes its payoff value while the other player tries to minimize Max's payoff [35].

**AHTN-P:** an Adversarial Hierarchical Task Network, that combines minimax game tree search with HTN planning [7]. In this AHTN definition the main task of the game can be achieved only by three non-primitive tasks (abstract actions that decompose into actions that agents can directly execute in the game). The tasks are three rushes with three different unit types.

**AHTN-F:** a more elaborate AHTN with a larger number of non-primitive tasks for harvesting resources, training units of different types, or attacking the enemy.

All search based algorithms (bottom seven in the list above) evaluate states by running a short playout of 100 frames. The playouts are performed using a random policy in which non-move actions (harvest, attack, build) have a higher probability than moves. The only exception is MinMax, whose playouts are 400 frames long, because it only does 16 playouts — one for each pair of strategies — and uses its fixed set of strategies instead of the random policy. The resulting states are evaluated with the simple evaluation function in equation 1, the optimized function, the Lanchester model or the neural network.

Every player has a computational budget of either a given time duration or a maximum number of state evaluations per game frame. Moreover, players can split the search process over multiple frames; for example, if the game state does not change during 10 game frames before a player needs to issue an action, then players have ten times the budget to issue actions. We call this consolidated budget a *search episode*.

In the tournament each of the 176 matchups consists of 24 games played on an  $8\times 8$  map, with different but symmetric

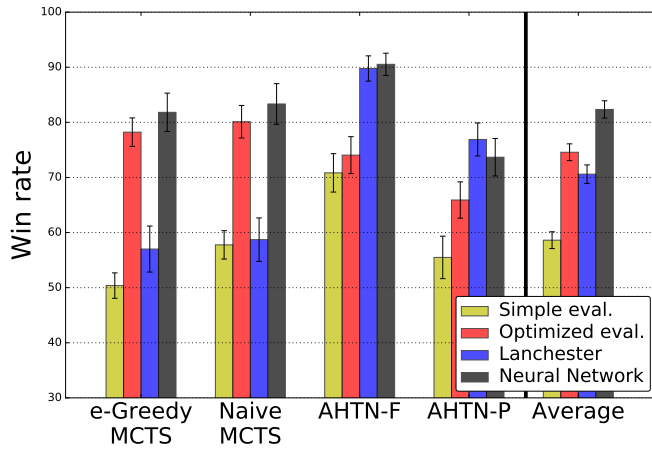


Fig. 4. Average win rate against all opponents when using the simple evaluation function described in equations 1 and 2, the same function with optimized weights, the Lanchester model or the neural network described in section III. Each algorithm has **200 milliseconds** of search time per frame. Error bars show one standard error.

starting positions. To compute the score, every win is worth 1 point, and if the game reaches 3000 frames, it is considered a draw, and awarded 0.5 points.

Figure 4 summarizes the average win rate against all opponents when using the different evaluation methods. On average, the neural network shows over 10% higher win rates than the other methods. Moreover, the performance of the neural network is consistent across all four algorithms, while the results of the optimized evaluation and the Lanchester model fluctuate depending on the underlying search algorithm type.

Table II shows the average number of nodes expanded per search episode. The average length of a search episode in the tournament games was around seven frames. Slow search algorithms such as AHTNs are less affected by a slow state evaluation, as most of their computational effort is expended in the tree phase. As a result, the AHTNs perform better when using the most accurate functions, regardless of their speed. The balance on the faster MCTS algorithms is more delicate, with both the fast optimized evaluation function and the neural network outperforming the relatively accurate and fast Lanchester model. The  $\sim 1\%$  accuracy increase in the first quarter of the game between the optimized simple

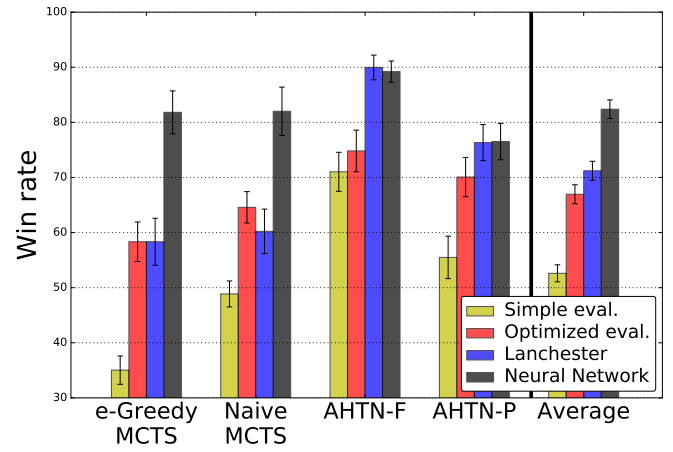


Fig. 5. Average win rate against all opponents when using the simple evaluation function described in equations 1 and 2, the same function with optimized weights, the Lanchester model or the neural network described in section III. Each algorithm is allowed to expand **200 nodes** per frame. Error bars show one standard error.

evaluation and Lanchester is not enough to offset the  $\sim 15\%$  less nodes per second. However, the  $\sim 7\%$  accuracy gain of the neural network more than makes up for its  $\sim 93\%$  speed loss. Improving the accuracy of the evaluation function at the beginning of the game is important, as early game decisions likely have a large impact on the game outcome.

Figure 5 shows a summary of a similar tournament using a limit of 200 state evaluations per frame, rather than 200 milliseconds. The fastest simple evaluation function shows significantly worse performance on the MCTS algorithms, because in this experiment only the evaluation accuracy is relevant, not the speed.

To scale the neural network to larger map sizes and more complex games, the size of the network will likely have to increase, both in the size of each layer and in the number of layers. This expansion will lead to slower evaluation times. However, we have shown that a small increase in evaluation accuracy is able to compensate for several orders of magnitude in speed reduction. Furthermore, running the network in batches on a GPU rather than the CPU should counteract most of the lost speed. MCTS algorithms can readily be modified to perform state evaluations in batches, as done for AlphaGo [1], which would result in several orders of magnitude speed improvements.

## V. CONCLUSIONS AND FUTURE WORK

In this paper we have used deep CNNs to evaluate RTS game states. We have shown that the evaluation accuracy is higher than current alternatives in  $\mu$ RTS. This new method performed better evaluating early game positions which led to stronger gameplay when used within state-of-the-art RTS search algorithms.

While CNNs might not perform significantly better in all cases (for instance compared to Lanchester when used in AHTN-P and AHTN-F, see Figures 4 and 5), the game playing

TABLE II

NODES EXPANDED PER SEARCH EPISODE, WHEN RUNNING WITH A MAXIMUM TIME LIMIT OF 200MS PER FRAME.

AI Algorithm	Average # nodes expanded per search episode		
	Simple Evaluation	Lanchester	Neural Network
$\epsilon$ -Greedy MCTS	16834	14682	1069
Naive MCTS	16654	13876	1122
AHTN-F	937	969	507
AHTN-P	134	125	123



agents based on them were stronger on average. Evaluating our CNN is several orders of magnitude slower than the other evaluation functions, but the accuracy gain far outweighs the speed disadvantage.

With these promising results, coupled with the fact that modern CNNs have shown excellent results on large problem sets [3], we are confident that the presented methods will scale up to more complex RTS games. StarCraft maps are similar in size to the images these networks are usually applied to. Using an MCTS implementation based on game abstractions similar to  $\mu$ RTS, that allows for asynchronous state evaluations on multiple GPUs can aid in tackling these larger problems while meeting real time constraints. Moreover, policy networks may also be trained to return probability distributions over the possible moves which can be used as prior probabilities to focus MCTS on the most promising branches.

Unlike Go, however, even RTS games with professional leagues such as StarCraft do not make replays of competition games publicly available. Without a large number of high quality records, reinforcement learning techniques will likely need to be considered in future work.

## REFERENCES

- [1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, “Mastering the game of go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
- [2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.
- [4] A. Kovarsky and M. Buro, “Heuristic search applied to abstract combat games,” *Advances in Artificial Intelligence*, pp. 66–78, 2005.
- [5] D. Churchill, A. Saffidine, and M. Buro, “Fast heuristic search for RTS game combat scenarios,” in *AI and Interactive Digital Entertainment Conference, AIIDE (AAAI)*, 2012.
- [6] S. Ontañón, “The combinatorial multi-armed bandit problem and its application to real-time strategy games,” in *AIIDE*, 2013.
- [7] S. Ontañón and M. Buro, “Adversarial hierarchical-task network planning for complex real-time games,” in *IJCAI*, 2015, in press.
- [8] M. Stanescu, N. A. Barriga, and M. Buro, “Hierarchical adversarial search applied to real-time strategy games,” in *Tenth Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2014.
- [9] M. Stanescu, S. P. Hernandez, G. Erickson, R. Greiner, and M. Buro, “Predicting army combat outcomes in StarCraft,” in *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2013.
- [10] M. Stanescu, N. A. Barriga, and M. Buro, “Using Lanchester attrition laws for combat prediction in StarCraft,” in *Eleventh Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2015.
- [11] N. A. Barriga, M. Stanescu, and M. Buro, “Puppet Search: Enhancing scripted behaviour by look-ahead search with applications to Real-Time Strategy games,” in *Eleventh Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2015.
- [12] G. Erickson and M. Buro, “Global state evaluation in StarCraft,” in *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2014.
- [13] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.
- [14] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3431–3440.
- [15] F. Schroff, D. Kalenichenko, and J. Philbin, “Facenet: A unified embedding for face recognition and clustering,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 815–823.
- [16] C. Dong, C. C. Loy, K. He, and X. Tang, “Learning a deep convolutional network for image super-resolution,” in *Computer Vision—ECCV 2014*. Springer, 2014, pp. 184–199.
- [17] A. Kendall, M. Grimes, and R. Cipolla, “Posenet: A convolutional network for real-time 6-dof camera relocalization,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2938–2946.
- [18] C. Szegedy, S. Ioffe, and V. Vanhoucke, “Inception-v4, inception-resnet and the impact of residual connections on learning,” *arXiv preprint arXiv:1602.07261*, 2016.
- [19] R. Johnson and T. Zhang, “Effective use of word order for text categorization with convolutional neural networks,” *arXiv preprint arXiv:1412.1058*, 2014.
- [20] X. Zhang and Y. LeCun, “Text understanding from scratch,” *arXiv preprint arXiv:1502.01710*, 2015.
- [21] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *arXiv preprint arXiv:1409.0473*, 2014.
- [22] L. Kaiser and I. Sutskever, “Neural gpus learn algorithms,” *arXiv preprint arXiv:1511.08228*, 2015.
- [23] J. Frnkranz and M. Kubat, *Machines that learn to play games*. Nova Publishers, 2001.
- [24] Y. LeCun and M. A. Ranzato, “Deep learning,” Tutorial at ICML, 2013. [Online]. Available: [www.cs.nyu.edu/~yann/talks/lecun-ranzato-icml2013.pdf](http://www.cs.nyu.edu/~yann/talks/lecun-ranzato-icml2013.pdf)
- [25] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang, “Deep learning for real-time atari game play using offline monte-carlo tree search planning,” in *Advances in Neural Information Processing Systems*, 2014, pp. 3338–3346.
- [26] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” in *NIPS Deep Learning Workshop*, 2013.
- [27] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [28] A. Shleyfman, A. Komenda, and C. Domshlak, “On combinatorial actions and cmabs with linear side information,” in *ECAI*, 2014, pp. 825–830.
- [29] B. Xu, N. Wang, T. Chen, and M. Li, “Empirical evaluation of rectified activations in convolutional network,” *CoRR*, vol. abs/1505.00853, 2015. [Online]. Available: <http://arxiv.org/abs/1505.00853>
- [30] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1026–1034. [Online]. Available: [http://www.cv-foundation.org/openaccess/content\\_iccv\\_2015/html/He\\_Delving\\_Deep\\_into\\_ICCV\\_2015\\_paper.html](http://www.cv-foundation.org/openaccess/content_iccv_2015/html/He_Delving_Deep_into_ICCV_2015_paper.html)
- [31] —, “Deep residual learning for image recognition,” *arXiv preprint arXiv:1512.03385*, 2015.
- [32] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *International conference on artificial intelligence and statistics*, 2010, pp. 249–256.
- [33] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: <http://arxiv.org/abs/1412.6980>
- [34] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” *arXiv preprint arXiv:1408.5093*, 2014.
- [35] F. Sailer, M. Buro, and M. Lanctot, “Adversarial planning through strategy simulation,” in *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*. IEEE, 2007, pp. 80–87.

# Improved LinUCT and Its Evaluation on Incremental Random-Feature Tree

Yusaku Mandai

Graduate School of Arts and Sciences  
The University of Tokyo  
Tokyo, Japan  
Email: mandai@graco.c.u-tokyo.ac.jp

Tomoyuki Kaneko

Graduate School of Arts and Sciences  
The University of Tokyo  
Tokyo, Japan  
Email: kaneko@graco.c.u-tokyo.ac.jp

**Abstract**—UCT is a standard method of Monte Carlo tree search (MCTS) algorithms, which have been applied to various domains and have achieved remarkable success. This study proposes a family of *Leaf-LinUCT*, which are improved LinUCT algorithms incorporating LinUCB into MCTS. LinUCB outperforms UCB1 in contextual multi-armed bandit problems, owing to a kind of online learning with ridge regression. However, due to the minimax structure of game trees, ridge regression in LinUCB does not always work well in the context of tree search. In this paper, we remedy the problem and extend our previous work on LinUCT in two ways: (1) by restricting teacher data for regression to the frontier nodes in a current search tree, and (2) by adjusting the feature vector of each internal node to the weighted mean of the feature vector of the descendant nodes. We also present a new synthetic model, *incremental-random-feature tree*, by extending the standard incremental random tree model. In our model, each node has a feature vector that represents the characteristics of the corresponding position. The elements of a feature vector in a node are randomly changed from those in its parent node by each move, as the heuristic score of a node is randomly changed by each move in the standard incremental random tree model. The experimental results show that our Leaf-LinUCT outperformed UCT and existing LinUCT algorithms, in the incremental-random-feature tree and a synthetic game studied in [1].

## I. INTRODUCTION

Monte Carlo Tree Search (MCTS) [2] is a search algorithm that tries to identify the best move by performing a number of simulations. It has achieved remarkable success in the game of Go [3], general game players [1], real-time games [4], and several others. For games, MCTS constructs a search tree and performs a *playout* involving Monte Carlo simulation at each time step to improve the empirical estimation of the reward at each node in its search tree. In a limited computational budget, MCTS needs to estimate the rewards as precisely as possible.

UCT [5] is the most successful and widely used MCTS variant. For each playout of UCT, a leaf node is selected in a best-first manner by descending the most promising node with respect to the upper confidence bound of the reward, UCB1 [6]. When UCT reaches the leaf node, it performs Monte Carlo simulation and observes the reward of a simulation that consists of random moves. One of the advantages of UCT is that it works effectively without domain knowledge

in contrast to alpha-beta search [7], which requires a decent evaluation function.

Although UCT does not explicitly require heuristics, many studies have incorporated domain knowledge such as progressive widening, prior knowledge, and PUCB [8]–[10] into UCT to improve the convergence speed or playing strength. Moreover, in recent years, many studies have tried to exploit the domain knowledge by using convolutional neural networks [11], [12] to incorporate it into MCTS, especially in the game of Go.

In this paper, we propose a new MCTS algorithms, Leaf-LinUCT. Instead of UCB1, it employs the LinUCB algorithm [13], [14] which has been studied in contextual bandit problems. While UCB1 maintains past rewards of each arm separately, LinUCB generalizes past episodes by ridge regression of rewards on feature vectors. Thus, LinUCB is an alternative means to incorporate domain knowledge in an online manner. However, as shown in our previous work [15], ridge regression in LinUCB does not always work well in the context of tree search, due to the minimax structure of game trees. In this paper, we describe how we remedied the problem, and extend our previous work on LinUCT in two ways: (1) by limiting teacher data for regression in the frontier nodes in a current search tree, and (2) by adjusting the feature vector of each internal node to the weighted mean of the feature vector of the descendant nodes. We also present a new synthetic model, *incremental-random-feature tree*, by extending the standard incremental random tree model. In our model, each node has a feature vector that represents the characteristics of the corresponding position. The elements of a feature vector in a node are randomly changed from those in its parent node by each move, just as the heuristic score of a node is randomly changed by each move in the standard incremental random tree model.

The experimental results in incremental-random-feature tree indicated that our Leaf-LinUCT outperformed UCT and existing LinUCT algorithms, especially when the branching factor was relatively large.

In addition, we conducted experiments in a synthetic game that Finnsson and Björnsson proposed in the literature [1]. Throughout the game we observed the properties of our proposed Leaf-LinUCT algorithms.

A part of this work was supported by JSPS KAKENHI Grant Number 25330432 and 16H02927.

The rest of this paper is organized as follows. The next section briefly reviews methods of MCTS and LinUCB. The third section introduces our new algorithm Leaf-LinUCT and the fourth section presents the synthetic model of a game tree. The fifth and sixth sections present our experimental results, and the last section concludes the paper.

## II. BACKGROUND AND RELATED WORK

In this section, we briefly review the background of our study, particularly the Monte Carlo Tree Search method and work related to it, as well as contextual bandit problems by comparison with multi-armed bandit problems.

### A. Multi-armed Bandit Problem

The Multi-armed Bandit Problem is a well-studied reinforcement learning problem. In the problem, an agent is supposed to choose an action from a set of actions. The action in this problem is typically called an *arm*. When the agent pulls an arm number  $i$ , he receives a reward that depends on a fixed but unknown distribution with an expected value  $\mu_i$ . In the conventional setting of the multi-armed bandit problem, the aim of the agent is to accumulate as much reward as possible in  $T$  trials. The aim can be considered as to minimize the *cumulative regret*, defined as

$$R_T = \sum_{t=1}^T (\mu^* - \mu_{I_t}),$$

where  $\mu^*$  is the mean reward of the optimal (i.e., maximum) arm, and  $\mu_{I_t}$  is the mean reward that the agent pulled at the  $t$ -th trial. A strategy of the bandit problem, or *bandit algorithm*, is optimal if the cumulative regret of the algorithm can be restricted to  $O(\log T)$  [16]. Some use the *average regret* per turn as the quality of bandit algorithms, defined as  $R_T/T = \mu^* - \sum_{t=1}^T \mu_{I_t}/T$ .

**UCB1** [6] is one of the most well-known bandit algorithms. Through the trials, the UCB1 algorithm calculates the upper confidence bound of rewards of each arm, which is defined as:

$$\bar{X}_{i,T_i(t)} + \sqrt{\frac{2 \ln t}{T_i(t)}},$$

where  $T_i(t)$  is the number of times the algorithm pulled arm  $i$  up to time  $t$ , and  $\bar{X}_{i,T_i(t)}$  is the empirical mean reward of arm  $i$  over  $T_i(t)$  trials.

### B. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) [2] is a tree search algorithm that employs Monte Carlo simulation to evaluate states in a tree. MCTS is an iterative search algorithm, and its iteration consists of four steps: (1) selection, (2) expansion, (3) simulation, and (4) back-propagation. Given a root state, MCTS selects recursively the most urgent child node until it reaches a leaf state in its search tree. Several implementations of MCTS use a strategy for multi-armed bandit problems as a selection-criterion to evaluate the urgency of each child state. In UCT [5], the selection-criterion is UCB1. A leaf is

expanded (i.e., the subsequent states from the leaf are added to the search tree) if the number of visit-counts reaches a certain threshold. The threshold is varied by the implementation: however, we assume the threshold is two, which is the least reasonable threshold. From the leaf state, the remaining states are simulated until it reaches a terminal state. Typically the simulation is performed uniformly randomly. In a terminal state, a reward (e.g., win, loss or draw in games) of the state is observed and back-propagated from the leaf to the root in its search tree to update the urgency of each node in the path. MCTS terminates the iteration steps when a certain computational budget (e.g., time) runs out. Then it returns the most promising child of the root, typically the most visited child [17].

### C. Contextual Bandit Problem

One of the interesting applications of the multi-armed bandit problem is the contextual bandit problem, in which an agent has to choose one arm in a set of arms in each iteration. The difference from the plain multi-armed bandit problem is that the agent can observe  $d$ -dimensional feature vectors (context vectors) associated with each arm and the current iteration. The agent can exploit these contexts along with the rewards of past trials to make better predictions.

Many contextual bandit algorithms exist, and these can be split into two categories: linear-classifier algorithms and nonlinear-classifier algorithms. We introduce here the most popular linear-classifier algorithm called LinUCB.

**LinUCB** [13], [14] assumes a linear dependency between the expected reward of an arm and its feature vector. In other words, the expected reward of arm  $a$  is the following formula:

$$\mathbf{E}[r_{t,a} | \mathbf{x}_{t,a}] = \mathbf{x}_{t,a}^\top \boldsymbol{\theta}_a^* \quad (1)$$

The LinUCB algorithm estimates  $\boldsymbol{\theta}_a^*$  as  $\hat{\boldsymbol{\theta}}_a$  through the trials. With the  $\hat{\boldsymbol{\theta}}_a$ , the prediction error can be evaluated with a probability of at least  $\delta$  as follows [13], [18]:

$$|\mathbf{x}_{t,a}^\top \hat{\boldsymbol{\theta}}_a - \mathbf{E}[r_{t,a} | \mathbf{x}_{t,a}]| \leq \alpha \sqrt{\mathbf{x}_{t,a}^\top \mathbf{A}_a^{-1} \mathbf{x}_{t,a}}, \quad (2)$$

where  $\alpha = 1 + \sqrt{\ln(2/\delta)/2}$  and  $\mathbf{A}_a$  represents a variance-covariance matrix of arm  $a$ . Equation (2) gives a reasonably tight UCB for the expected payoff of arm  $a$ . From (2), the LinUCB algorithm chooses

$$a_t = \operatorname{argmax}_{a \in \mathcal{A}_t} \mathbf{x}_{t,a}^\top \hat{\boldsymbol{\theta}}_a + \alpha \sqrt{\mathbf{x}_{t,a}^\top \mathbf{A}_a^{-1} \mathbf{x}_{t,a}} \quad (3)$$

at each trial from the available arm set  $\mathcal{A}_t$ . Algorithm 1 describes the LinUCB algorithm.

### D. LinUCT

LinUCT is an MCTS algorithm incorporating LinUCB, which we proposed in our earlier work [15], rather than UCB1. As in LinUCB, LinUCT observes the feature vector for each node and predicts the reward of the node. Considering its application to two-player games, we address the case where the feature vectors of nodes are stationary (i.e.,  $\forall t \mathbf{x}_{t,a} = \mathbf{x}_a$ ). In

```

1: Inputs:  $\alpha \in \mathbb{R}_+$ 
2: for  $t = 1, 2, 3, \dots$  do
3:   Observe features of all arms  $a \in \mathcal{A}_t$ :  $\mathbf{x}_{t,a} \in \mathbb{R}^d$ 
4:   for all  $a \in \mathcal{A}_t$  do  $\triangleright \mathcal{A}_t$  is a set of available arms at  $t$ 
5:     if  $a$  is new then
6:        $\mathbf{A}_a \leftarrow \mathbf{I}_{d \times d}$   $\triangleright d$  dimensional identity matrix
7:        $\mathbf{b}_a \leftarrow \mathbf{0}_{d \times 1}$   $\triangleright d$  dimensional zero vector
8:        $\hat{\boldsymbol{\theta}}_a \leftarrow \mathbf{A}_a^{-1} \mathbf{b}_a$ 
9:        $p_{t,a} \leftarrow \mathbf{x}_{t,a}^\top \hat{\boldsymbol{\theta}}_a + \alpha \sqrt{\mathbf{x}_{t,a}^\top \mathbf{A}_a^{-1} \mathbf{x}_{t,a}}$ 
10:     $a_t \leftarrow \underset{a \in \mathcal{A}_t}{\operatorname{argmax}} p_{t,a}$  with ties broken arbitrarily
11:    Observe a real-valued payoff  $r_t$ 
12:     $\mathbf{A}_{a_t} \leftarrow \mathbf{A}_{a_t} + \mathbf{x}_{t,a_t} \mathbf{x}_{t,a_t}^\top$ 
13:     $\mathbf{b}_{a_t} \leftarrow \mathbf{b}_{a_t} + r_t \mathbf{x}_{t,a_t}$ 

```

Algorithm 1: LinUCB [13]

addition, we assume that the expected reward of each *terminal* node  $s$  in a game tree is  $\mathbf{x}_s^\top \boldsymbol{\theta}^*$  and under the control of a common  $\boldsymbol{\theta}^*$ , while LinUCB assumes there are different  $\boldsymbol{\theta}_a^*$  for each arm. The assumption seems to be reasonable in deterministic games since algorithms employing static evaluation functions are dominant in various real games [12], [19], [20]. Hereafter, the algorithms use a common  $\boldsymbol{\theta}^*$ ,  $\mathbf{A}$  and  $\mathbf{b}$  (without subscription).

Even when the reward at a leaf can be predicted by  $\mathbf{x}_s^\top \boldsymbol{\theta}^*$  similarly to the LinUCB setting, such an assumption is not valid for internal nodes in a game tree. Therefore, the main challenge in LinUCT is the prediction of the reward in internal nodes. In this paper, we present two techniques to achieve this. In our previous work, LinUCT algorithms regress the rewards on the feature vectors of all nodes in the path from the root to the leaf in each iteration. However, regressions of rewards at an internal node tend to make the estimation of  $\boldsymbol{\theta}^*$  less accurate. Therefore, we limit the teacher data to feature vectors of leaf nodes. Moreover, we present weighted propagation to stabilize the feature vector of an internal node. We show that these differences achieved a significant improvement in empirical convergence in the experiments.

### III. LEAF-LINUCT

Leaf-LinUCT is a family of MCTS algorithms incorporating LinUCB instead of UCB1, which is an enhanced version of our previous work of LinUCT. In the following subsection, we first present Leaf-LinUCT<sub>PLAIN</sub>, which is a straightforward application, for reference. Then, Leaf-LinUCT<sub>WP</sub>, which is the main contribution, is presented. Also, Leaf-LinUCT<sub>FP</sub>, which is presented in our earlier work, is introduced for comparison.

#### A. Leaf-LinUCT<sub>PLAIN</sub>

Leaf-LinUCT<sub>PLAIN</sub> is a straightforward application of LinUCB to MCTS. Leaf-LinUCT<sub>PLAIN</sub> employs LinUCB in the

```

1: procedure INITIALIZEREGRESSION( $\lambda$ )
2:    $\mathbf{A} \leftarrow \lambda \mathbf{I}_{d \times d}$ 
3:    $\mathbf{b} \leftarrow \mathbf{0}_{d \times 1}$ 
4: procedure UPDATEREGRESSION( $s, r$ )
5:    $\mathbf{A} \leftarrow \mathbf{A} + \mathbf{x}_s \mathbf{x}_s^\top$ 
6:    $\mathbf{b} \leftarrow \mathbf{b} + r \mathbf{x}_s$ 
7: procedure BACKPROPAGATION(path,  $\Delta$ )
8:    $s_l \leftarrow$  leaf node
9:   UPDATEREGRESSION( $s_l, \Delta_s$ )

```

Algorithm 2: Supplementary Procedures in Leaf-LinUCT

selection step of MCTS. The urgency of child state  $s_i$  is defined as

$$\mathbf{x}_{s_i}^\top \hat{\boldsymbol{\theta}} + \alpha \sqrt{\mathbf{x}_{s_i}^\top \left( \frac{T_{s_i}(t)}{N_{\text{update}}} \mathbf{A} \right)^{-1} \mathbf{x}_{s_i}}.$$

where  $\mathbf{x}_{s_i}$  is a feature vector of state  $s_i$ ,  $\hat{\boldsymbol{\theta}}$  represents the current estimate of  $\boldsymbol{\theta}^*$ , which can be calculated by  $\hat{\boldsymbol{\theta}} = \mathbf{A}^{-1} \mathbf{b}$ ,  $\mathbf{A}$  means the variance-covariance matrix,  $\mathbf{b}$  is the corresponding response vector (i.e., it has the corresponding simulation rewards, see algorithm 2) and  $\alpha$  is a constant.  $N_{\text{update}}$  is the number of times that LinUCT<sub>PLAIN</sub> updates its guess of  $\boldsymbol{\theta}^*$ , and  $T_{s_i}(t)$  is the visit-count of state  $s_i$  so far.

In the same way as UCT, LinUCT<sub>PLAIN</sub> descends its search tree to a leaf state. After the selection step, it performs a simulation of the rest of the game and observes the outcome of the simulation. In this paper, we assume this simulation consists of uniform random moves. In the back-propagation step,  $\hat{\boldsymbol{\theta}}$  is updated by the following formulae:

$$\mathbf{A} = \mathbf{A} + \mathbf{x}_{s_j} \mathbf{x}_{s_j}^\top, \quad \mathbf{b} = \mathbf{b} + r \mathbf{x}_{s_j}, \quad \hat{\boldsymbol{\theta}} = \mathbf{A}^{-1} \mathbf{b},$$

where  $\mathbf{x}_{s_j}$  is the feature vector of  $s_j$ , which is the leaf state in the search tree reached in the selection step, or equivalently, the start state of a random simulation in the iteration.

#### B. Leaf-LinUCT<sub>WP</sub>

In MCTS, the estimation of the reward of a node is expected to converge to the minimax score of its sub-tree as the iteration grows. However, a serious problem of Leaf-LinUCT<sub>PLAIN</sub> is that it cannot estimate the reward of an internal node, because the assumption in (1) does not hold for an internal node in the tree.

To remedy this problem, we proposed the Leaf-LinUCT<sub>FP</sub> algorithm, in which the feature vector of each node  $s$  is modified so that  $\mathbf{x}_s \boldsymbol{\theta}^*$  converges to the minimax score of its sub-tree in a certain assumption.

At each iteration, the feature vector of LinUCT<sub>WP</sub> is updated to the weighted average of feature vectors of its child nodes by its number of visit-counts so far. At time  $t$ , the feature vector of each internal node is

$$\mathbf{x}_s = \frac{\sum_{s_i \in \text{children}} (T_{s_i}(t) + 1) \cdot \mathbf{x}_{s_i}}{\sum_{s_i \in \text{children}} (T_{s_i}(t) + 1)}$$

```

1: procedure UPDATEFEATURE-LINUCTWP( $s$ )
2:   if  $s$  is leaf node then return
3:    $S \leftarrow$  children of  $s$ 
4:    $\mathbf{x}_s \leftarrow \sum_{s_i \in S} (T_{s_i}(t) + 1) \cdot \mathbf{x}_{s_i} / \sum_{s_i \in S} (T_{s_i}(t) + 1)$ 
5:   procedure BACKPROPAGATION(path,  $\Delta$ )
6:      $s_l \leftarrow$  leaf node
7:     UPDATEREGRESSION( $s_l, \Delta_s$ )
8:     # supplementary procedure of LinUCT-WP
9:     for  $s \in$  path do ▷ From bottom to root
10:      UPDATEFEATURE-LINUCBWP( $s$ )

```

Algorithm 3: Back-propagation process of LinUCT<sub>WP</sub>

The number of visit-counts is increased by one to avoid division by zero. Note that the feature vector of each leaf in a search tree is not modified. Algorithm 3 describes the back-propagation procedures of LinUCT<sub>WP</sub>.

The evaluation using the averaged feature vector is close to the empirical average of rewards as long as the expected reward at each leaf follows  $\mathbf{x}_s^\top \boldsymbol{\theta}^*$  as assumed in Eq. (1):

$$\begin{aligned}
\mathbf{x}_s^\top \boldsymbol{\theta}^* &= \left( \frac{\sum_{s_i \in \text{children}} (T_{s_i}(t) + 1) \cdot \mathbf{x}_{s_i}}{\sum_{s_i \in \text{children}} (T_{s_i}(t) + 1)} \right)^\top \boldsymbol{\theta}^* \\
&\approx \sum_{s_i \in \text{children}} \frac{T_{s_i}(t)}{T_s(t)} \cdot (\mathbf{x}_{s_i}^\top \boldsymbol{\theta}^*) \\
&= \sum_{l_i \in \text{leaves in sub-tree under } s} \frac{T_{l_i}(t)}{T_s(t)} \cdot (\mathbf{x}_{l_i}^\top \boldsymbol{\theta}^*)
\end{aligned}$$

Hence, it is reasonable to modify the feature vectors based on the feature vectors of the children according to their visit-counts.

### C. Leaf-LinUCT<sub>FP</sub>

Leaf-LinUCT<sub>FP</sub> is an alternative of Leaf-LinUCT<sub>WP</sub> and adaptation of LinUCT<sub>FP</sub> [15] so that the regression is only performed in the leaf node at each iteration. In Leaf-LinUCT<sub>FP</sub>, the feature vector of each node in the path in the search tree in each iteration are slightly modified to incorporate the vector of the next node as

$$\mathbf{x}_s \leftarrow (1 - \gamma) \mathbf{x}_s + \gamma \mathbf{x}_{s_c},$$

where  $s_c$  is the child node of  $s$  in the path and  $\gamma \in (0, 1)$  represents the learning rate of feature vectors. Note that LinUCT<sub>FP</sub> propagates the feature vector no matter what simulation rewards are so as to take both positive/negative rewards in the simulations into account. The purpose of this modification is to make a feature vector similar to that of the principal variation leaf as the number of iteration grows large. Algorithm 4 describes the procedures of LinUCT<sub>FP</sub> in the back-propagation phase. However, the experiments show that our new Leaf-LinUCT<sub>WP</sub> is stable.

## IV. TREE MODEL

Incremental random trees (or P-game) [21] have served as domain-independent test sets for evaluation of various search

```

1: procedure UPDATEFEATURE-LINUCTFP( $s, \gamma$ )
2:   if  $s$  is root node then return
3:    $s_p \leftarrow$  parent of  $s$ 
4:    $\mathbf{x}_{s_p} \leftarrow (1 - \gamma) \mathbf{x}_{s_p} + \gamma \mathbf{x}_s$ 
5:   procedure BACKPROPAGATION(path,  $\Delta$ )
6:      $s_l \leftarrow$  leaf node
7:     UPDATEREGRESSION( $s_l, \Delta_s$ )
8:     # supplementary procedure of LinUCT-FP
9:     for  $s \in$  path do ▷ From bottom to root
10:      UPDATEFEATURE-LINUCBFP( $s, \gamma$ )

```

Algorithm 4: Back-propagation process of LinUCT<sub>FP</sub>

algorithms [5], [22]–[25]. The advantages of using this model are that 1) the search space can be controlled easily via the width and height of the tree and 2) a correlation between the heuristic score of a node and that of its descendants is produced, which is expected in real games.

We extend the existing incremental random tree model and present incremental-random-feature tree so that the feature vector is assigned for each node. Our previous work presented a similar extension [15]: however, there are two significant differences: (1) the previous model can only handle binary features, while the new model can handle arbitrary values, and (2) the previous model has an unpleasant property in that each element of the feature vector of a node in a generated tree converges to zero when the depth of the tree gets larger [26]. The expected value and variance of each element of a feature vector is well designed in the new model.

### A. Incremental Random Feature Tree

We introduce the extended model of the incremental random tree model with feature vectors, called *incremental-random-feature tree* (IRF-tree). In IRF-tree, each node  $s$  has a feature vector (state vector)  $\mathbf{x}_s \in \mathbb{R}^d$ . A state vector is computed by adding a parent state vector and a feature vector associated with the move (move vector) as depicted in fig. 1. Move vectors are  $d$ -dimensional vectors whose elements are drawn from uniform random distribution  $\mathcal{U}(-w, w)$ . To model two-player zero-sum games, a state vector for a min player is  $-\mathbf{x}_s$  if the corresponding state vector for a max player is  $\mathbf{x}_s$ .

Heuristic scores of nodes are defined as:

$$\mathbf{x}_s^\top \boldsymbol{\theta}^*, \quad (4)$$

where  $\boldsymbol{\theta}^*$  is a hidden vector generated for each tree and cannot be observed by algorithms. The elements of  $\boldsymbol{\theta}^*$  are also drawn from uniform random distribution  $\mathcal{U}(-\sigma, \sigma)$ .

### B. Properties of Incremental Random Feature Tree

1) *Expected Vector*: Given a state vector  $\mathbf{x}_s$ , we can compute the expected state vector of its descendant through a path  $S = \{s_1, s_2, \dots, s_n\}$  as follows:

$$\mathbf{E} \left[ \mathbf{x}_s + \sum_{i=1}^n \mathbf{x}^{s_i} \right] = \mathbf{x}_s + \sum_{i=1}^n \mathbf{E} [\mathbf{x}^{s_i}] = \mathbf{x}_s,$$

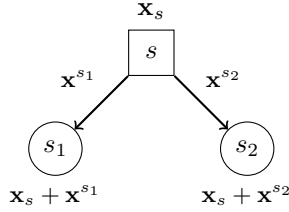


Fig. 1: Example of Incremental Random Feature tree: the move vector associated with the move for state  $s_1$  ( $s_2$ ) is  $\mathbf{x}^{s_1}$  ( $\mathbf{x}^{s_2}$ ) and the state vector of  $s_1$  ( $s_2$ ) is  $\mathbf{x}_s + \mathbf{x}^{s_1}$  ( $\mathbf{x}_s + \mathbf{x}^{s_2}$ )

since the elements of move vectors are drawn from a uniform random distribution centered at 0. Consequently, the score of the descendant, computed by the inner product between its state vector and a tree coefficient vector  $\theta^*$ , is:

$$\mathbf{E} \left[ \left( \mathbf{x}_s + \sum_{i=1}^n \mathbf{x}^{s_i} \right)^\top \theta^* \right] = \mathbf{x}_s^\top \theta^*$$

2) *Distribution of Rewards*: We show some properties of the distribution of the rewards for the leaves under node  $s$  in a tree. Obviously the expected mean of the distribution is  $\mathbf{x}\theta^*$ . The variance of the distribution can be calculated as

$$\text{Var} \left[ \left( \mathbf{x}_s + \sum_{i=1}^n \mathbf{x}^{s_i} \right)^\top \theta^* \right] = \frac{w^2 \sigma^2}{9} n d. \quad (5)$$

An approximation of the summation of uniform random variables by the normal distribution shows that the distribution of rewards follows the normal distribution centered at 0 with the variance shown in (5). In the later experiments, we set the parameters  $w$  and  $\sigma$  to 1,  $1/\sqrt{6d}$  respectively, so that each element is in a suitable range. We confirmed that distributions of heuristic scores (i.e., rewards for MCTS algorithms) are almost identical to the normal distribution, whose mean is 0 and variance is (5), with various configurations of tree-depths and branching factors and approximately 99% of the rewards is in range  $[-1, 1]$ .

## V. EXPERIMENTS IN IRF-TREE

We conducted experiments to compare the performance of our Leaf-LinUCT algorithms, LinUCT, and standard UCT. Throughout the experiments, we used incremental-random-feature tree as a test-bed, and each tree had a uniform branching factor and depth. The dimensions of feature vectors and coefficient vectors were fixed to 16 because we observed similar results in the experiments with varying dimensions.

Each MCTS algorithm iteratively constructs its search tree until it covers all nodes of the tree instance by incorporating the children of a leaf at the second visit. Simulations of MCTS from the leaf of the search tree consisted of 1) choosing the next node randomly until it reached a terminal of the tree instance and 2) calculating the reward of the terminal node according to (4) without error or noise.

For UCT, the rewards should be in a limited range, typically  $[0, 1]$ . We applied two types of transformation functions: linear

TABLE I: Parameters of algorithms

algorithm	parameter	value
LinUCT	$\lambda$	1.0
LinUCT	$\alpha$	1.0
LinUCT-FP	$\gamma$	0.01

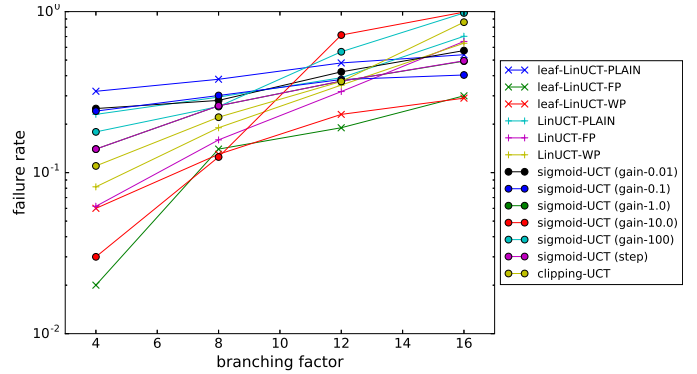


Fig. 2: Failure rate (vertical axis) versus branching factor of trees (horizontal axis) at  $t = 10\,000$

transformation with clipping  $\max(0, \min(1, x/2 + 0.5))$  and sigmoid function  $1/(e^{-ax} + 1)$ . Because almost all rewards are in range  $[0, 1]$  as described in Sect. IV-B2, clipping is rarely needed in the former function. Several values for the slope of the sigmoid function,  $a = 0.01, 0.1, 1.0, 10, 100$ , and  $a = \infty$  (step function), were evaluated in the experiments. Throughout the experiments, we set the parameters of LinUCT as described in table I in accordance with our previous work [15].

### A. Performance of Algorithms

First, we observed the *failure rates* and average regrets of these algorithms following the experiments in the literature [5]. The failure rate represents the frequency of choosing a move other than the optimal one. The optimal move of each tree instance was identified by the minimax tree search in advance. We generated 100 tree instances of IRF-tree, and each algorithm ran on each tree 100 times. Note that tree instances that contained multiple optimal moves were removed.

Fig. 2 and Fig. 3 plot the failure rates and average regrets of each algorithm with the branching factor of trees varying from 4 to 16. We can see that Leaf-LinUCT<sub>WP</sub> and Leaf-LinUCT<sub>FP</sub> are the two best algorithms, with few exceptions, in the failure rate for trees with a small branching factor. With regard to failure rate, UCT with sigmoid transformation occasionally performed better than the LinUCT algorithms. However, the best value of the slope is not stable with the branching factor of trees. Leaf-LinUCT algorithms are generally better than LinUCT algorithms. Additionally, Leaf-LinUCT<sub>PLAIN</sub> did not as well as Leaf-LinUCT<sub>WP</sub> or Leaf-LinUCT<sub>FP</sub>. These observations suggest the presented algorithms are effective.

### B. Accuracy of Evaluation

In order to analyze the difference in performance of the algorithms, we evaluated the correlation between the (game-



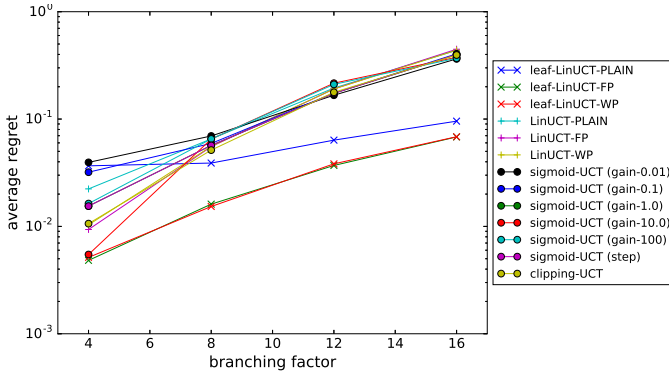


Fig. 3: Average regret (vertical axis) versus branching factor of trees (horizontal axis) at  $t = 10\,000$

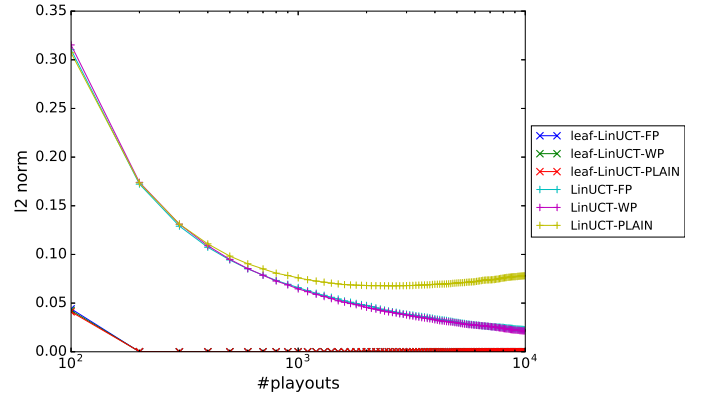


Fig. 5:  $\|\theta^* - \hat{\theta}\|_2$  as a function of  $t$  (log-scale, depth = 4, branching factor = 4)

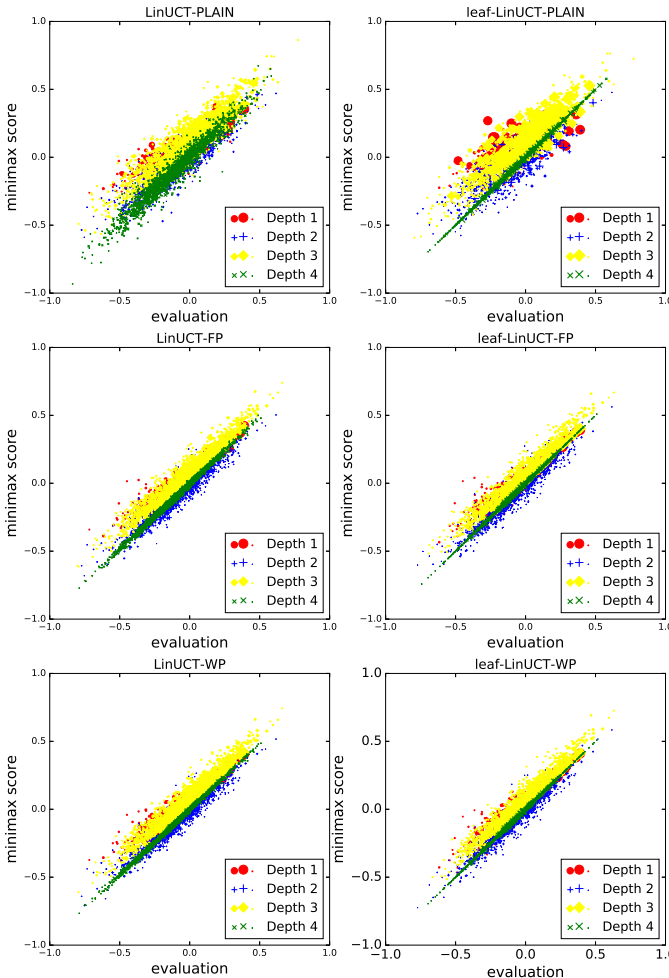


Fig. 4: Scatter plot of static evaluation (horizontal axes) versus minimax score (vertical axes) at 10 000 playouts. Nodes more than 100 visits are plotted

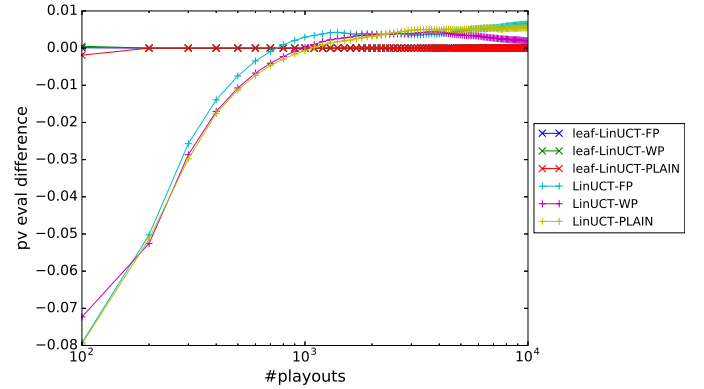


Fig. 6:  $\mathbf{x}_s^\top \theta^* - \mathbf{x}_s^\top \hat{\theta}$  as a function of  $t$  (log-scale, depth = 4, branching factor = 4)

theoretical) minimax score identified by a thorough minimax search and the score evaluated by (Leaf) LinUCT algorithms for each node. The scatter plots of these scores after the 10 000 playouts are shown in Fig. 4. Nodes visited more than 100 times are plotted. We can see that LinUCT (upper left) cannot evaluate nodes correctly even for terminal nodes at depth-4.

This result suggests the difficulty of estimation by plain LinUCB in tree search, because the minimax score does not suit the assumption in (1) for an internal node.

In contrast, Leaf-LinUCT<sub>PLAIN</sub> (upper right) successfully estimated the minimax scores of terminal nodes at depth 4. This can be explained by the fact that the problem was mitigated by applying regression only at frontier nodes.

The evaluations by LinUCT-FP and LinUCT-WP (middle and bottom left) are significantly more accurate than those by (Leaf) LinUCT<sub>PLAIN</sub>. Therefore, modifying the feature vectors of internal nodes according to the search progress contributes to the accuracy of the evaluation.

Also, the evaluations by Leaf-LinUCT<sub>FP</sub> and Leaf-LinUCT<sub>WP</sub> (middle and bottom right) are also significantly better than those by LinUCT<sub>PLAIN</sub>, especially for non-terminal nodes (depth  $\leq 3$ ).

We discuss the accuracy of the estimation of  $\theta^*$ , by showing

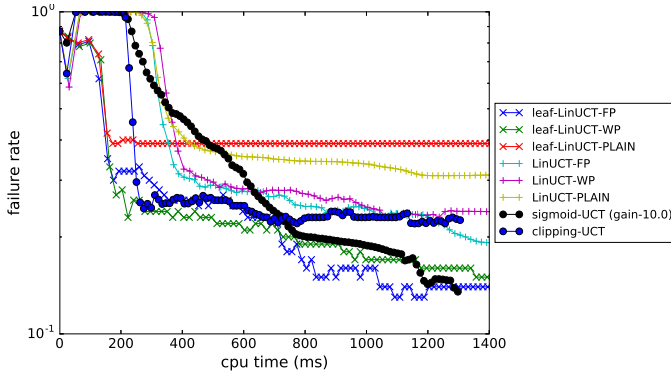


Fig. 7: Failure rate (vertical axis) versus branching factor of trees (horizontal axis), depth = 4, branching factor = 8

the  $L_2$  norm of the difference between  $\theta^*$  and  $\hat{\theta}$  in Fig. 5. Also, Fig. 6 plots the average difference between the minimax value of PV leaf  $x_l$  determined by minimax search in advance and its score  $x_l^T \hat{\theta}$  evaluated by each algorithm, as a function of playouts  $t$  in the trees with branching factor 4 and depth 4. It is clear that Leaf-LinUCT algorithms estimate  $\theta^*$  more accurately and rapidly than LinUCT algorithms.

### C. Run-time Efficiency of Algorithms

Finally, we compared the run-time efficiency of each algorithms. The failure rates as the CPU time increases in trees with depth 4 and branching factor 8 is shown in Fig. 7.

We can see from the figure that Leaf-LinUCT still worked better than UCT even with their time complexity of  $O(d^2)$ , where  $d$  is the dimension of feature vectors. We should note that dimension 16 in this experiment is smaller than the dimension of typical evaluation functions in real games. To handle higher dimensions in Leaf-LinUCT algorithms, it might be effective to incorporate acceleration techniques such as fLinUCB-GD [27].

## VI. EXPERIMENTS IN PROGRESSION GAME

We also compared the performance of our Leaf-LinUCT algorithms and UCT algorithm in the progression games, which Finnsson and Björnsson proposed in [1].

### A. Finnsson and Björnsson's Progression Game

Finnsson and Björnsson introduced a set of synthetic two-player games to discuss what properties of games affect performances of UCT. One of them is *Progression game* [1]. Fig. 8 depicts the initial state of the game. Each player chooses one of his pieces (*active* or *inactive* runners) in his turn. If the chosen runner is active, the runner moves forward (White) or downward (Black) by one square. The objective is to locate one of the runners in the goal square. Obviously the optimal strategy is continually choosing one of the active runners.

They introduced the *depth factor* that limits the maximum number of moves the players can. In a board with 20 lanes  $\times$  10 rows, the minimum number of moves the game ended is 18, if the Black (second) player's strategy is optimal. They

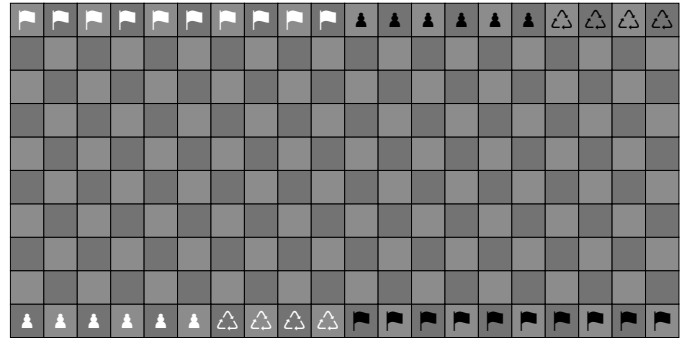


Fig. 8: Initial state of Progression game by Finnsson and Björnsson [1] (Pawn: active, Circular arrows: inactive, Flag: goal)

TABLE II: Result of Progression game

Algorithm	Feature Vector	#Wins	Win%
leaf-LinUCT-PLAIN	ratio-feature	2 000	100
leaf-LinUCT-FP		2 000	100
leaf-LinUCT-WP		1 342	67
leaf-LinUCT-PLAIN	raw-feature	55	3
leaf-LinUCT-FP		30	2
leaf-LinUCT-WP		1 677	84
UCT		801	40

modified the upper limit of moves in fixed step size of 18 (e.g., total number of moves for both players is 36 for depth factor 2). In our experiments, however, the depth factor was set to 100 to avoid the effect that simulations produce unreliable results. Throughout the following experiments, the board size and the number of active runners were set to  $20 \times 10$  and 1 respectively and the algorithms matched against the Black optimal player, following the experiments in [1]. The number of simulations per turn was set to 1 000.

### B. Feature vector for Leaf-LinUCT

We designed feature vector for Leaf-LinUCT two patterns: (1) ratio feature and (2) raw feature.

In ratio feature vector, the vector has 20 elements and each of them is proportional to the row of the piece in each lane. This feature vector contains sufficient information that enables Leaf-LinUCTs to predict the outcome of games perfectly. The other one, raw feature vector represents a board as it is. The dimension of the feature vectors is 200 and each element of a feature vector has positive constant if corresponding square of a board has a runner (regardless of the type). Note that these feature vectors are regularized to satisfy the condition  $\|x\| \leq 1$ .

### C. Experiments

TABLE II shows results of the progression game experiments. For ratio-feature algorithms, the performances are superior than UCT algorithm, especially leaf-LinUCT-PLAIN and leaf-LinUCT-FP. While these two algorithms won all matches, leaf-LinUCT-WP sometimes failed to choose the

correct runner. In contrast, for raw-feature algorithms, leaf-LinUCT-PLAIN and leaf-LinUCT-FP performed poorly and leaf-LinUCT-WP worked better. Because every children tend to be favorable for a root player in ratio-feature, leaf-LinUCT-WP evaluates states by average of child nodes. Consequently it performed worse. For leaf-LinUCT-FP, it needs feature vectors of the terminal states in this game to predict reward correctly. Hence, it requires deeper and narrower search: however, it could not search that manner without any support such as progressive widening.

## VII. CONCLUSION

We described in this paper our application of LinUCB to MCTS and presented our Leaf-LinUCT algorithms. The new algorithms are improved compared to the existing LinUCT algorithms: (1) by restricting teacher data for regression to the frontier nodes in a current search tree, and (2) by adjusting the feature vector of each internal node to the weighted mean of the feature vector of the descendant nodes. We also proposed a new synthetic tree model, called Incremental Random Feature tree, in which one can evaluate algorithms that use feature vectors for heuristic evaluation. The empirical experiments in incremental-random-feature trees showed that Leaf-LinUCT algorithms perform significantly better than existing algorithms including UCT, especially in trees with large branching factor. Empirical analyses on failure rate, average regret, prediction accuracy, and run-time efficiency were also presented. In addition, experiments in the game Finnsson and Björnsson proposed implied properties of our proposed algorithms.

Theoretical analysis on the convergence of Leaf-LinUCT algorithms would be an interesting topic in future work.

## REFERENCES

- [1] H. Finnsson and Y. Björnsson, "Game-tree properties and mcts performance," in *IJCAI*, vol. 11, 2011, pp. 23–30.
- [2] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 1, pp. 1–43, March 2012.
- [3] S. Gelly, L. Kocsis, M. Schoenauer, M. Sebag, D. Silver, C. Szepesvári, and O. Teytaud, "The grand challenge of computer go: Monte carlo tree search and extensions," *Commun. ACM*, vol. 55, no. 3, pp. 106–113, Mar. 2012.
- [4] P. Perick, D. L. St-Pierre, F. Maes, and D. Ernst, "Comparison of different selection strategies in monte-carlo tree search for the game of tron," in *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, Sept 2012, pp. 242–249.
- [5] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18-22, 2006, Proceedings*, ser. Lecture Notes in Computer Science, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds., vol. 4212. Springer, 2006, pp. 282–293.
- [6] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.
- [7] D. E. Knuth and R. W. Moore, "An analysis of alpha-beta pruning," *Artificial Intelligence*, vol. 6, no. 4, pp. 293–326, 1975.
- [8] G. M. J.-B. Chaslot, M. H. Winands, H. J. van den Herik, J. W. Uiterwijk, and B. Bouzy, "Progressive strategies for monte-carlo tree search," *New Mathematics and Natural Computation*, vol. 4, no. 03, pp. 343–357, 2008.
- [9] S. Gelly and D. Silver, "Monte-carlo tree search and rapid action value estimation in computer go," *Artificial Intelligence*, vol. 175, no. 11, pp. 1856–1875, 2011.
- [10] C. Rosin, "Multi-armed bandits with episode context," *Annals of Mathematics and Artificial Intelligence*, pp. 1–28, 2011.
- [11] Y. Tian and Y. Zhu, "Better computer go player with neural network and long-term prediction," *CoRR*, vol. abs/1511.06410, 2015.
- [12] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot et al., "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
- [13] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 661–670.
- [14] W. Chu, L. Li, L. Reyzin, and R. E. Schapire, "Contextual bandits with linear payoff functions," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011*, ser. JMLR Proceedings, G. J. Gordon, D. B. Dunson, and M. Dudík, Eds., vol. 15. JMLR.org, 2011, pp. 208–214.
- [15] Y. Mandai and T. Kaneko, "Linucb applied to monte-carlo tree search," in *Advances in Computer Games - 14th International Conference, ACG 2015, Leiden, The Netherlands, July 1-3, 2015, Revised Selected Papers*, ser. Lecture Notes in Computer Science, A. Plaat, H. J. van den Herik, and W. A. Kusters, Eds., vol. 9525. Springer, 2015, pp. 41–52.
- [16] T. Lai and H. Robbins, "Asymptotically efficient adaptive allocation rules," *Advances in Applied Mathematics*, vol. 6, no. 1, pp. 4 – 22, 1985.
- [17] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," in *Computers and Games, 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers*, ser. Lecture Notes in Computer Science, H. J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers, Eds., vol. 4630. Springer, 2006, pp. 72–83.
- [18] T. J. Walsh, I. Szita, C. Diuk, and M. L. Littman, "Exploring compact reinforcement-learning representations with linear regression," in *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, ser. UAI '09. Arlington, Virginia, United States: AUAI Press, 2009, pp. 591–598.
- [19] M. Buro, "Improving heuristic mini-max search by supervised learning," *Artificial Intelligence*, vol. 134, no. 12, pp. 85 – 99, 2002.
- [20] K. Hoki and T. Kaneko, "Large-scale optimization for evaluation functions with minimax search," *Journal of Artificial Intelligence Research*, pp. 527–568, 2014.
- [21] R. E. Korf and D. M. Chickering, "Best-first minimax search," *Artificial Intelligence*, vol. 84, no. 12, pp. 299 – 337, 1996.
- [22] S. J. Smith and D. S. Nau, "An analysis of forward pruning," in *AAAI*, 1994, pp. 1386–1391.
- [23] T. Furtak and M. Buro, "Minimum proof graphs and fastest-cut-first search heuristics," in *Proceedings of the 21st IJCAI*, C. Boutilier, Ed., 2009, pp. 492–498.
- [24] K. Yoshizoe, A. Kishimoto, T. Kaneko, H. Yoshimoto, and Y. Ishikawa, "Scalable distributed monte-carlo tree search," in *the 4th SoCS*, 2011, pp. 180–187.
- [25] T. Imagawa and T. Kaneko, "Enhancements in monte carlo tree search algorithms for biased game trees," in *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, Aug 2015, pp. 43–50.
- [26] Y. Mandai and T. Kaneko, "Linucb applied to monte carlo tree search," *Theoretical Computer Science*, 2016.
- [27] N. Korda, P. L. A., and R. Munos, "Online gradient descent for least squares regression: Non-asymptotic bounds and application to bandits," *CoRR*, vol. abs/1307.3176, 2013.

# Detection and Labeling of Bad Moves for Coaching Go

Kokolo Ikeda, Simon Viennot and Naoyuki Sato

School of Information Science

JAIST

Japan

Email: kokolo@jaist.ac.jp, sviennot@jaist.ac.jp, satonao@jaist.ac.jp

**Abstract**—The level of computer programs has now reached professional strength for many games, even for the game of Go recently. A more difficult task for computer intelligence now is to create a program able to coach human players, so that they can improve their play. In this paper, we propose a method to detect and label the bad moves of human players for the game of Go. This task is challenging because even strong human players only agree at a rate of around 50% about which moves should be considered as bad. We use supervised learning with features largely available in many Go programs, and we obtain an identification level close to the one observed between strong human players. Also, an evaluation by a professional player shows that our method is already useful for intermediate-level players.

## I. INTRODUCTION

Computer programs to play games have improved a lot this last decade, with the use of machine-learning and Monte-Carlo Tree Search (MCTS). The last success was the defeat of a professional Go player by AlphaGo in 2016[6], by using a combination of deep convolutional neural networks and MCTS. The strength of other programs for the game of Go is now expected to increase quickly, so we can consider that creating a strong Go program is not as challenging as before.

However, there are still other difficult and interesting problems for computer intelligence in the area of games, especially the game of Go. A first one is the creation of entertaining programs. Computer programs are still too frequently boring for human players, because they tend to use similar strategies repeatedly. Another interesting problem is the ability to coach human players, by showing them their mistakes, and explaining them how to improve.

For entertainment or coaching purposes, programs need some new abilities usually not considered when strength is the only target. For example, for entertaining humans, a control of the position is needed, so that both players keep a reasonable chance of winning. It

can be achieved with intentionally gentle - but natural - moves. The thinking time used for each move or the resign timing are also important. For coaching humans, bad moves need to be detected, and some explanation is also needed, either with figures, text or speech.

In this research, we consider the problem of coaching Go players. In the case of the game of Go, it is frequent for human players to review their own games with a stronger player and seek advice about which moves were bad. So, an ideal coaching computer program should be able to detect the bad moves, to label them with the type of mistake, and finally to give a more detailed explanation. Also, a figure showing a better move with its consequences would be useful. In this research, we consider only the problem of detecting and labeling the bad moves.

Deciding which moves should be considered as bad is a challenging task. In a preliminary experiment, we asked strong players to show the bad moves in game records of intermediate-level players. The strong players only agreed at a rate of around 50%. Also, we will show that a naive approach like using only the drop of winning ratio from the point of view of a strong computer program does not work well. Many bad moves from the point of view of humans are locally non-optimal moves (for example a bad shape), but the loss in terms of winning chances is in fact small. In this paper, we propose to use machine-learning to address this problem.

In Section II, we give some more details about coaching Go, and how it is usually done between humans. In Section III, we discuss some related work. Then, in Section IV, we describe our approach based on supervised machine learning. Section V describes our main experiments, with a machine learning for detecting bad moves, and a separate machine learning for labeling them. The result is evaluated with a professional Go player.

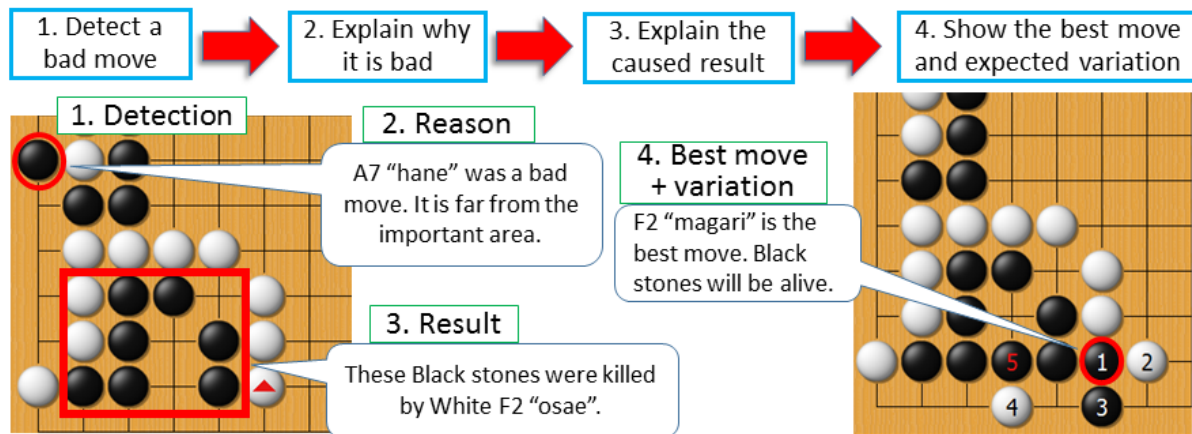


Fig. 1. Complete process for helping players to correct their bad moves.

## II. COACHING GO

Go is an ancient game, especially popular in Asia, with a rich variety of sub-problems and strategies. For this reason, many Go players find their satisfaction not just in playing the game, but in trying to improve their play and becoming stronger.

There are many ways to improve at the game of Go, such as replaying professional games, solving local life and death problems (tsumego), or reading books about common tactics and patterns. But it is often considered that one of the best way to improve is to play a game with a stronger player, and to review the game with him.

Coaching Go (Shido-Go in Japanese) is a special type of game, where amateur players pay some professional or semi-professional player to play and review a game. There is a strong demand from amateur players for such games, but it can be expensive. Also, intermediate-level players are often reluctant to invest the money or the time in such coaching games, because they feel that their level is still too low for that. If a computer player could perform the same kind of coaching, it would be of great help for many amateur players, especially from beginner to intermediate level.

We surveyed in Go clubs in Japan how strong players teach intermediate-level players about their bad moves. It usually follows the process shown in Figure 1. First, a bad move is detected (1). Some reason (2) is given on why it is a bad move. We call this step "labeling" in this paper. Then, a more detailed explanation is given on what happened as a result (3) of the bad move. Finally, a better move is shown, with the expected best variation (4).

In this paper, we address only the first two steps, i.e. the detection of bad moves, and the labeling with some "reason". Our goal in the future is to create a computer program able to perform all 4 steps.

## III. RELATED WORK

Entertaining and coaching players is a developing area of research on board games. In 2013, Ikeda et al. (authors of this paper) proposed a computer Go program able to entertain players by using various strategies and controlling the board position [3]. In 2015, Kameko et al. used machine-learning to generate comments in natural language about Shogi positions [4]. Also in 2015, Ikeda et al. (authors of this paper) used machine-learning to learn the natural language names usually used by humans to refer to moves in the game of Go [5]. This is an important part for a coaching Go program, since moves are usually referred by shape names and not by coordinate positions in the game of Go. For example, in Figure 1, A7 is called "Hane".

## IV. APPROACH

The ultimate goal of our research is to make a coaching computer player who plays gently against human players, corrects bad moves and explains how to think/play. It would encourage human players to continue playing while their skills are improving. As a first step towards this goal, we try (1) to detect bad moves from a game record and (2) to associate an explanation label to each bad move. Then, the computer player could output something like "The 17th move at D4 was not good, because the local shape is bad. D5 is better."



The definition of “bad move” is not trivial. If the set of best moves can be defined and calculated, though theoretically it is possible, it may be possible to say that the other moves are all bad moves. However, for coaching intermediate players, usually only *fairly* bad moves are pointed out because such players will be confused or depressed if too many moves are corrected.

For selecting *fairly* bad moves, it will be effective to refer to the “winning ratio” computed by computer players. Now, we assume that White is played by an MCTS player, and Black is played by an intermediate human player. An MCTS program calculates not only the next best move but also many statistics such as the expected winning probability (winning ratio). When the ratio for White is increased, for example from 30% to 50%, it means that Black played a fairly bad move which loses a big advantage, 20%. Usually, a Black move which loses 20% winning ratio should be pointed out, before any other move that loses only 2%.

But it should be noted that winning ratio will be not sufficient to select bad moves for effective coaching. Human teachers often point out and correct some kind of bad moves even when the loss of winning ratio is not so serious. One example is shown in Fig. 2. The shape of Black move A is bad, and B should be played. We think almost all Go teachers will point out this move, even if the difference of winning ratio of A and B is only about 1 – 2%. To detect such a move, “shape goodness” should be computed and referred. Another example can be considered in end games. Assume that Black is almost surely winning, and the territory advantage is 12 points. When Black played a bad move which loses 4 points, it will be pointed out, even if the winning ratio is only slightly changed from 99% to 98%. To detect such a move, “territory advantage” should be computed and referred. So, we calculate and use several values as input features, for accurate detection of bad moves. The employed features are explained in V-A.

The detection and labeling of bad moves are done separately. The whole procedure is as follows:

- 1) Many handicap games are done. Game records are collected.
- 2) Bad moves are selected by strong human players, with only 5 to 10 moves selected per game.
- 3) Also, one type (why the move is bad) is labeled on each bad move, from some candidates.
- 4) Many features are calculated by a computer program, for each move. We obtain a set of items (feature1, feature2, ..., bad/good, type).
- 5) A supervised learning is executed by using all items

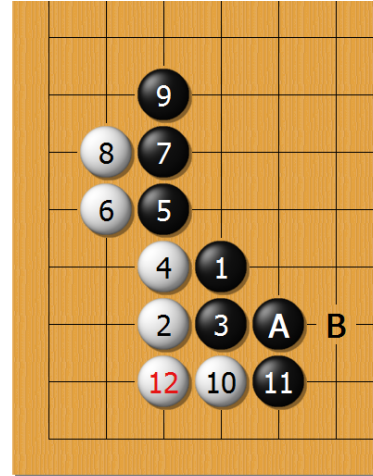


Fig. 2. An example of bad shape, A is bad, B is good. This A will be pointed though the loss of winning ratio is not so big.

where “bad/good” is the output. The result is the “detection system”.

- 6) Another supervised learning is executed by using “bad” items where type is the output. The result is the “labeling system”.

## V. EXPERIMENTS

In this section, we show four series of experiments. The brief content is as follows:

- 1) Preliminary experiment to show that the winning ratio is not a sufficient feature for detecting bad moves
- 2) Learning of bad move detection system, and comparison with human’s decision
- 3) Learning of bad move labeling system, and comparison with human’s decision
- 4) Evaluation of the detection and labeling systems by a professional player

### A. Preparation

As described at the end of Section IV, we need to gather many handicap games, to let strong human players select bad moves and label them, and to calculate many feature values for each move. We employed our computer Go program “Nomitan”. It is ranked 3d on the KGS server, which is not so strong, but not so weak. First, we asked 8 intermediate-level human players (from about 7k to 1d) to play against Nomitan, using a 13 × 13 board and with 2 to 4 handicap stones, as they want. Totally, 108 games were collected.

Next, we asked three strong human players (about 4d to 7d on KGS) to select bad Black moves and select a



type label for each bad move. It was requested to ignore bad White moves, and to select about 5 to 10 bad moves per game. A type label for each bad move is a brief reason explaining why the move is bad. It was selected from the following 10 candidates. Since some types were rarely labeled, they are integrated into 5 groups.

- **Group-1**
  - Local shape is bad.
- **Group-2**
  - Gain is small.
  - The move is too defensive or fearing a risk, then the gain is small.
- **Group-3**
  - The move is far from the hot area.
  - The move is far from the hot area, White stones should be attacked.
  - The move is far from the hot area, Black stones should be defended.
- **Group-4**
  - The player seems to do a reading mistake (i.e. a tactical error when considering what happens a few moves ahead).
- **Group-5**
  - The move is too passive. It seems to be only responsive to the last White move.
  - The move helped White stones to be stronger.
  - Other reasons.

For 102 in 108 games (set denoted by  $G_{102}$ ), only one of the three human players did this selection and type labeling of the bad moves. For 6 games (set denoted by  $G_{common6}$ ), 244 Black moves, all of the three human players did this work. This allows us to compare the selection result between humans. For example, Table I shows the difference in bad move selection between two strong players A and B.

TABLE I  
BAD MOVE SELECTION BY STRONG PLAYERS A AND B

	B good	B bad
A good	180	27
A bad	24	13

Out of 244 moves, player A selected 37 moves as bad moves, but only 13 of these 37 moves are also selected by player B. This result shows that bad move detection is not a simple work even for strong human players.

Totally, 4836 Black moves were collected from 108 games. For each of these moves, we calculated 29 feature values to be referred in supervised learning. Here, some

important features are explained, and the other ones are explained in Appendix A. Please note that such features are not specific to our program. They can be easily calculated by most MCTS programs.

- **handi**, the number of handicap stones.
- **move**, the number of moves played.
- **wrbefore**, **wrafter**, **wrdiff**, expected winning ratio before the move, after the move, and its difference.
- **trbefore**, **trafter**, **trdiff**, expected territory advantage before the move, after the move, and its difference.
- **shaperate**, **shapelog**, shape goodness calculated by Bradley-Terry model [2], relative value and absolute log value.
- **dist1b**, Euclidean distance between the actual Black move and the estimated best move.
- **ownbefore**, **ownafter**, **owndiff**, ownership of the position before the move, after the move, and its difference. High ownership means that the area is occupied by Black, i.e. the Black stones in the area are strong, or the White stones in the area are weak.

#### *B. Preliminary Experiments: feature selection for good/bad detection*

In this section, binary supervised learning experiments about “detection system” are shown, to prove that many features should be used for detecting bad moves accurately.

We have 3963 good move instances and 873 bad move instances. Since such unbalance among the numbers of instances is not preferable in classification, 2000 good move instances are randomly removed in this experiment. Since there are a lot of candidate methods for binary classification, we employ Multilayer Perceptron in a free machine learning platform, Weka version 3.6.11 [1].

For evaluating the performance, the F-measure, the mixed value of precision and recall, is used. For example, in the case of Table I, if we assume that the decision of B is always true, the precision of A about bad moves is  $\frac{13}{24+13} = 0.351$ , the recall of A about bad moves is  $\frac{13}{27+13} = 0.325$ , and the F-measure about bad moves is  $\frac{2 \cdot 0.351 \cdot 0.325}{0.351 + 0.325} = 0.338$ .

When only **wrdiff** (how the winning ratio is changed by a Black move) is used as the input, the F-measures are 0.812/0.299/0.654 (F-measure about good moves / F-measure about bad moves / weighted average for good and bad moves). These values are the averages of 10-folding validation. It is not strange that the F-measure about good moves is better than that about bad moves,

because the number of good move instances (1963) is still bigger than the number of bad move instances (873).

We tried to improve the performance by adding other features. Table II shows the result. By adding one or two features, the total F-measure is increased by 0.008 to 0.031. It is clear that shape goodness and territory advantage should be considered for accurate detection. Maybe it is interesting to see that **move** or **wrbefore/wrafter** fairly improve the performance. This is because usually bad moves in early stage frequently affect the game consequence, and bad moves after losing game (for example winning ratio is under 30%) are not pointed by human coaches.

Finally, when using 9 features **wrdiff**, **wrbefore**, **wrafter**, **shapelog**, **trdiff**, **trbefore**, **trafter**, **move**, **owndiff**, the F-measure about bad moves is significantly improved from 0.299 to 0.444. We can conclude that, not only the winning ratio, but also many other features are needed for an accurate detection of bad moves.

TABLE II  
GOOD/BAD DETECTION RESULTS. USED FEATURES AND F-MEASURES

features	F-measures	gain
wrdiff only	0.812/0.299/0.654	-
+wrbefore, wrafter	0.815/0.361/0.675	0.021
+shaperate	0.814/0.326/0.664	0.010
+shapelog	0.812/0.357/0.672	0.018
+trdiff	0.809/0.381/0.677	0.023
+trbefore, trafter	0.817/0.389/0.685	0.031
+handi	0.810/0.333/0.663	0.009
+move	0.812/0.378/0.678	0.024
+dist1b	0.813/0.322/0.662	0.008
+owndiff	0.812/0.330/0.664	0.010
+8 features	0.826/0.444/0.709	0.055

### C. Machine-Learning for Detection

In Section V-B, we observed that 8 additional features are effective to improve the detection accuracy, and there 10-folding self validation is used. In this section, the learning set and the test set are manually separated, and the performance for test data is compared to the performance between human strong players.

As shown in Table I, decisions are fairly different from each other, even among strong players. Table III shows F-measures of each player for another player, we can see A for B is relatively far, B for C is relatively similar. The simple averages are **0.892/0.435/0.820**. We try to achieve these values by machine learning.

$G_{common6}$  is used as the test set, including 609 good moves and 117 bad moves.  $G_{102}$  is used as the training set, including 3354 good moves and 756 bad moves. In order to balance the numbers of good moves and

TABLE III  
F-MEASURES OF GOOD/BAD DETECTION

	F-measures
player A for B	0.876/0.338/0.794
player B for C	0.907/0.525/0.844
player C for A	0.895/0.442/0.821
average	0.892/0.435/0.820
MP for player A,B,C	0.875/0.409/0.800

bad moves, we clone each bad moves of the training set from one to three, then 3354 good moves and 2268 bad move instances are used for training. We think this cloning method is better for obtaining a good detection system than deleting 2000 good moves, but it should be noted that 10-folding self validation becomes unfair when using this cloning method, then another way was used in Section V-B.

Multilayer Perceptron (MP) in Weka is used, and the same 9 features shown in Section V-B are referred. The achieved F-measures were **0.875/0.409/0.800**. They are slightly worse than the average among strong human players, but better than those of player A for B. We can guess that the decisions (detected bad moves) are not so strange compared to those from strong human players.

### D. Machine-Learning for Labeling

The second step is to label a type on each detected bad move. We have totally 873 labelled (bad move) instances, the numbers of instances of 5 groups are 228, 228, 212, 98 and 107.

Like the experiments shown in Section V-B, we did some preliminary experiments to select a classification method and select the referred features. After comparing several methods available in Weka, such as J4.8, LADTree, SMO or Multilayer Perceptron, we selected “Logistic” as the classification method.

The total F-measure (averaged by 10-folding self validation) is 0.406 when using the full set of 29 features. We tried to improve the F-measure by removing some features to avoid overfitting. In almost all cases the F-measure is decreased by removing features, this suggests that more complex features are effective in the labeling system compared to the detection system. The F-measure is slightly increased when removing some of 7 features, and finally the F-measure is 0.434 when removing all the 7 features.

Next, as in the experiments shown in Section V-C, we separated 873 instances in a learning set and a test set, and compared the F-measure to that among human players. The learning set contains 756 instances from  $G_{102}$ , which are selected as bad moves. The test

set contains 69 instances from  $G_{common6}$ , which are selected as bad moves, **by two or three of strong human players A,B,C**. 48 instances of  $G_{common6}$  are selected as bad moves by only one of the three players, then it is impossible to compare whether the labeled types are the same or different.

The total F-measure among human players is shown in Table IV. The averaged F-measure is 0.483, which means that two players frequently gave different labels to a bad move. The achieved F-measure by Logistic is 0.499, this is better than the average. Since the number of test set is only 69, we think this is just a lucky case. In fact, when using other sophisticated classifiers, the F-measure is only in the range from 0.35 to 0.42. The labeling system (the second step classification) will be more difficult than the detection system (the first step classification), because the number of output classes is bigger, 5 instead of 2, and because the size of the learning set is smaller, 756 instead of 4110. We guess the performance will be fairly improved when increasing the size of the learning set.

TABLE IV  
F-MEASURES OF BAD MOVE TYPE LABELING

	F-measure
player A for B	0.482
player B for C	0.436
player C for A	0.531
average	0.483
Logistic for player A,B,C	0.499

#### E. Evaluation by a Professional Player

In Sections V-C and V-D, mainly F-measure values are used for evaluation, and they are compared to the average F-measure between strong human players. However, F-measures cannot evaluate whether terrible decisions exist or not, for example “a really bad move is not detected” or “the definitely-best move is detected as a bad move”. Then, an absolute evaluation by a professional player is done.

$G_{common6}$  are the games for which all three strong human players selected bad moves and labeled their types. At first, the detection system (Multilayer Perceptron) employed in Section V-C was used for  $G_{common6}$ , then 46 bad moves were selected. Next, the labeling system (Logistic) employed in Section V-D was used for these bad moves, then we obtained 6 game records where bad moves are selected and labeled. A game labeled by our method is shown in Appendix B.

Totally 24 game records were sent to a 6d professional player in a blind manner, and we asked him to give a

score for each game record, about (1) How well the bad move detection is done, and (2) How reasonable the type labeling is done. The scoring criterion we asked was as follows:

- 100 points: at the same quality of human professional coaches
- 90 points: at the same quality of human 6d amateur coaches
- 70 points: there are some problems, but still sufficiently valuable for intermediate players.
- 50 points: there are many or serious problems, then not so valuable even for intermediate players.

Table V shows the points given for (1) bad move detection, and Table VI shows the points given for (2) type labeling. The average scores of players A, B and C are similar, about 80 points, but individual scores are not so stable, from 60 points to 100 points. Please note that 90 points are not achieved even though they are about 6d amateur players. The average scores of our systems are worse than that of strong human players, by about 6 points, but better in some games. Total average 74.2 and 76.7 are not bad, clearly better than 70 points level, “sufficiently valuable for intermediate players”.

We consider that our method is promising or even already useful, and we can expect the performance to improve if we collect more games as training data.

TABLE V  
EVALUATION SCORES BY A PROFESSIONAL, FOR BAD MOVE DETECTION

handicap game ID	4 stones		3	2 stones			average
	1	2	3	4	5	6	
player A	75	60	90	85	90	90	81.7
player B	90	75	80	75	75	90	80.8
player C	80	85	90	65	85	80	80.8
our method	70	70	80	75	70	80	74.2

TABLE VI  
EVALUATION SCORES BY A PROFESSIONAL, FOR TYPE LABELING

handicap game ID	4 stones		3	2 stones			average
	1	2	3	4	5	6	
player A	70	70	80	90	80	90	80.0
player B	90	75	80	70	75	100	81.7
player C	95	95	70	70	90	85	84.2
our method	70	70	90	80	70	80	76.7

## VI. CONCLUSION AND FUTURE WORK

Since strong computer players can be implemented for many games, entertaining and/or coaching computer players have become a new target of computer intelligence. In this paper, we design a system for detecting

and labeling bad moves played by human players, in the game of Go. It was shown that such a system is not easy to obtain, since there is around a 50% mismatch even among strong human players, and many input features are needed for making adequate decisions. We collected 4110 moves labeled by strong players, calculated 22 features, and employed a two-step supervised learning. The qualities of detection and labeling were evaluated by a professional Go player. It was shown that both of them were clearly at a useful level, though slightly worse than the level of strong human players.

As future work, the number of learning data should be increased because these supervised learning problems are a difficult task. Many features are needed and then much learning data is needed to avoid overfitting. Also, we would like to tackle some other aspects of game coaching. Especially in the case of correcting bad moves in the game of Go, after the detection and labeling of the bad moves, it would be useful to explain the result of each bad move, and also to show the best move with its consequences. Playing various games with an understanding of abstract concepts is now not such a difficult task for computer intelligence, but coaching human players with an explanation of such abstract concepts is still a challenging task.

#### REFERENCES

- [1] J. R. Quinlan, C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, 1993
- [2] Remi Coulom, "Computing Elo Ratings of Move Patterns in the Game of Go", *ICGA Workshop*, (2007)
- [3] Kokoro Ikeda and Simon Viennot, "Production of Various Strategies and Position Control for Monte-Carlo Go - Entertaining human players", Proceedings of the 2013 IEEE Conference on Computational Intelligence and Games (CIG), pp. 145-152 (2013)
- [4] Hirofumi Kameko, Shinsuke Mori and Yoshimasa Tsuruoka, "Learning a Game Commentary Generator with Grounded Move Expressions", Proceedings of the 2015 IEEE Conference on Computational Intelligence and Games (CIG), pp. 177-184 (2015)
- [5] Kokoro Ikeda, Simon Viennot and Takanari Shishido, "Machine-Learning of Shape Names for the Game of Go", 14th International Conference Advances in Computer Games, (2015)
- [6] David Silver, Aja Huang et al. "Mastering the game of go with deep neural networks and tree search". *Nature*, 529(7587) pp. 484-489 (2016)

#### APPENDIX A.

In Section V-A, 14 of 29 features are explained. Here the other 15 features are briefly explained. Please note that the additional cost for calculating these 29 features for each Black move is not so expensive in fact, because almost all feature values can be calculated within the procedure that the program decides the White move.

- **trstdbefore, trstdafter, trstddiff**, standard deviation of territory advantages, before/after the move

and its difference. They are calculated with **trbefore**, **trafter**, **trdiff**, and representing how unclear the game result is.

- **dist01, dist02, dist21, dist0b, dist2b**, Euclidean distances between two of { the last White move (0), the next White move (2), the actual Black move (1), and the estimated best move (b) }.
- **own2before, own2after, own2diff**, averaged ownership of Black stones on  $3 \times 3$  area neighboring the Black move. Values before/after the move, and its difference.
- **bdecav, wdecav**, average ownership decreases of all Black/White stones, by the next White move. When **bdecav** is high, it means Black stones are weakened by the next White move, because of losing chance to defend.
- **bdec30, wdec30**, the number of Black/White stones which their ownerships are decreased by 0.3, by the next White move.

In this paper, 7 of them, **own2before, own2after, own2diff, dist0b, dist2b, bdecav, wdecav** were removed after a test described in Section V-D and not used in the last evaluation. However, if more learning set and/or stronger program can be used, it may be better to use these 7 and more features.

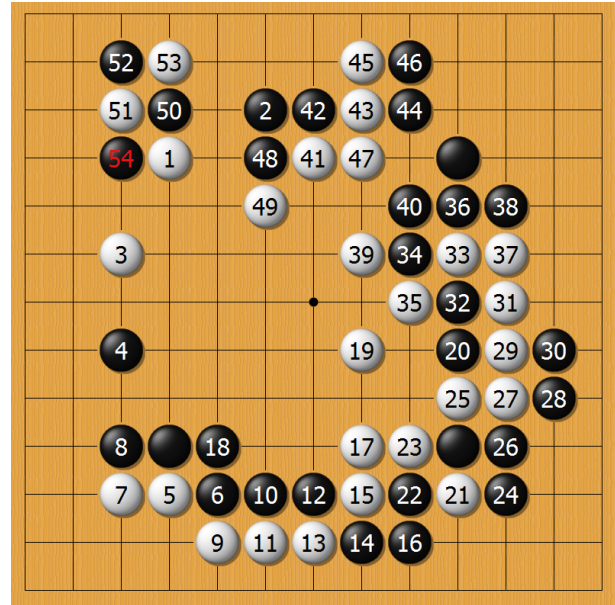


Fig. 3. Game-3 labeled by our method, up to the 54th move.

#### APPENDIX B.

Figure 3 shows the 1st to 54th moves of game-3, evaluated in Section V-E. In fact, the Black player

resigned after White 77, but only 54 moves are shown for readability. The bad moves detected by our method, and comments by the professional are as follows.

- 8th, group-1 (bad shape). OK.
- 14th, group-1. OK.
- 18th, group-1. OK.
- 24th. This move is not good, but not detected.
- 30th, group-3 (far from hot area). This move is not so bad and another type would be better.
- 32nd, group-3. OK.
- 38th, group-1. This move is not so bad.
- 46th, group-1. Another type will be better.
- 54th. This move is fairly bad, but not detected.
- anyway, detection and labeling are at a useful level.

# Investigating Vanilla MCTS Scaling on the GVG-AI Game Corpus

Mark J. Nelson  
The MetaMakers Institute  
Falmouth University  
Penryn, Cornwall, UK  
Email: mjn@anadrome.org

**Abstract**—The General Video Game AI Competition (GVG-AI) invites submissions of controllers to play games specified in the Video Game Description Language (VGDL), testing them against each other and several baselines. One of the baselines that has done surprisingly well in some of the competitions is `sampleMCTS`, a straightforward implementation of Monte Carlo tree search (MCTS). Although it has done worse in other iterations of the competition, this has produced a nagging worry to us that perhaps the GVG-AI competition might be too easy, especially since performance profiling suggests that significant increases in number of MCTS iterations that can be completed in a given time limit will be possible through optimizations to the GVG-AI competition framework. To better understand the potential performance of the baseline vanilla MCTS controller, I perform scaling experiments, running it against the 62 games in the public GVG-AI corpus as the time budget is varied from about 1/30 of that in the current competition, through around 30x the current competition's budget. I find that it does not in fact master the games even given 30x the current time budget, so the challenge of the GVG-AI competition is safe (at least against this baseline). However, I do find that given enough computational budget, it manages to avoid explicitly *losing* on most games, despite failing to win them and ultimately losing as time expires, suggesting an asymmetry in the current GVG-AI competition's challenge: not losing is significantly easier than winning.

## I. INTRODUCTION

The General Video Game AI Competition (GVG-AI) [1], [2] is a recurring videogame-playing competition intended to stimulate progress in general videogame AI (as distinguished from AI written for a specific game) by testing submitted AI agents against previously unseen videogames. Games used in the competition are written in the Video Game Description Language (VGDL) [3], a domain-specific language designed to capture the variety of arcade-style games in which the rules take the form of sprite movement and interaction on a 2d grid. In the current competition, games are all single-player, and may be deterministic or stochastic. The set of unpublished test games on which agents are scored is periodically replaced with a fresh set of games, to keep agents from having the opportunity to even inadvertently become specialized to a specific reused test set. When a new test set is added, the old one is released publicly. Therefore a corpus of VGDL games has slowly grown, currently at 62 public games (as of April 2016).

The 62 VGDL games distributed with the GVG-AI competition framework do represent a specific subset of games—

there is nothing here like chess, Starcraft, or even Tetris. But within the style of arcade games that VGDL targets, they cover a fairly wide range of challenges and characteristics, from twitch-type action games to puzzle games, games with NPCs and without, games with counter-based, spatial, or time-based win conditions, and so on.<sup>1</sup> This makes the corpus useful as a testbed for investigating differences between algorithms and games.

The purpose of this paper is to take an extended look at the performance scaling of one specific algorithm across this GVG-AI corpus: how the play of vanilla Monte Carlo tree search (MCTS) improves, or doesn't, on these 62 games as its computation budget is increased. The goal of doing so is to better understand both sides of the pairing: to use the GVG-AI corpus to look at how MCTS scales with performance across a range of videogames, and also to use the MCTS performance curves as a way of characterizing the nature of the challenges found in the current public set of GVG-AI competition games.

One specific question motivating this scaling experiment was whether the GVG-AI competition might be too easy. In the first GVG-AI competition at CIG 2014, the `sampleMCTS` controller implementing a vanilla MCTS search, included with the SDK and intended as baseline, somewhat surprisingly came in 3rd place, achieving a win rate of about 32%. In the competitions since then it has not done as well, as both its competitors and the test games have gotten more challenging; at CEEC 2015 it did particularly badly, placing 31st with a win rate of only 12%.<sup>2</sup>

The poorer recent results do not entirely dispel the worry about difficulty. The competition tests agents with only one specific time limit, 40 milliseconds per move. That leaves open the possibility that the challenge might be mostly posed by the short time budget. But as hardware gets faster, and the GVG-AI framework becomes more optimized,<sup>3</sup> many more MCTS iterations will be possible to complete within the same 40 milliseconds. Would the competition *then* become too easy, with the `sampleMCTS` agent making quick work of

<sup>1</sup>A feature matrix comparing the first 30 games is given in Table 1 of [1].

<sup>2</sup>Past competition results are taken from the online rankings at <http://www.gvgai.net/>.

<sup>3</sup>Profiling suggests that a large portion of the computation time in the GVG-AI framework's forward model is taken up by Java collections bookkeeping; significant speedups are likely possible by reworking the code here.



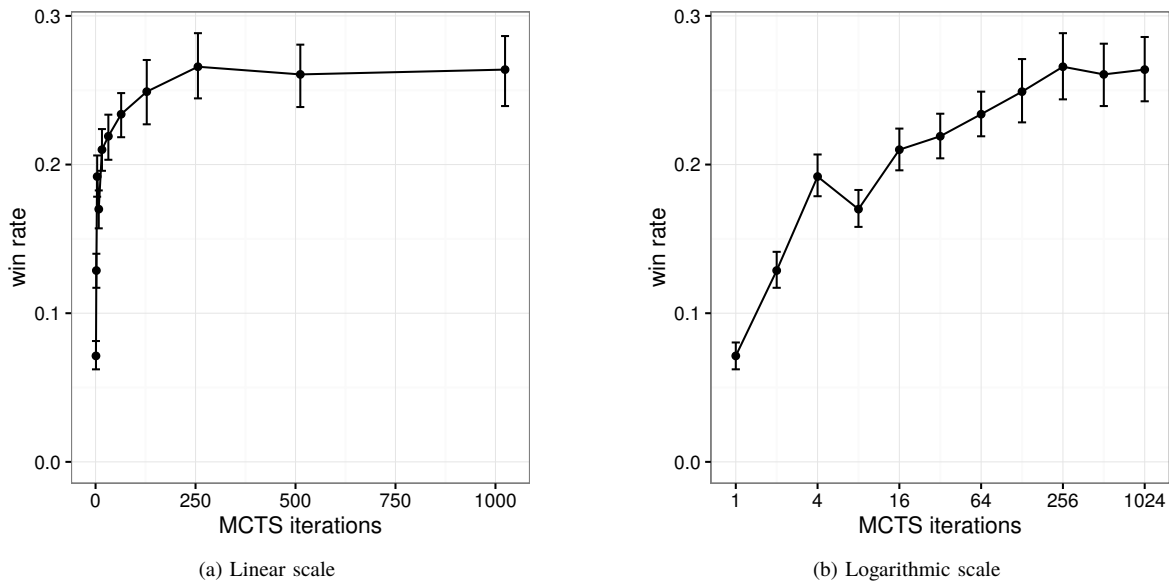


Fig. 1. Number of MCTS iterations versus overall win rate across all trials of the 62 games, shown with both (a) linear and (b) logarithmic x axes. Error bars, here and in subsequent figures, represent 95% confidence intervals, estimated via a nonparametric bootstrap.

the games?

The experiments here find that the answer is no. Performance on the current corpus of 62 games for the `sampleMCTS` agent does improve as computation time is increased from the current limit. But it plateaus at a win rate of 26% when given about 10x the current competition’s computation budget, and doesn’t further improve beyond that (tested out to around 30x). Therefore we can conclude that even after an order of magnitude increase in speed of the forward model (whether through hardware improvements or optimizations), the GVG-AI competition will still pose challenges requiring something more than vanilla MCTS to tackle them.

I do however find that as MCTS’s computation budget is increased, the way it plays in the games it loses changes considerably. In many of the GVG-AI games, failing to achieve a win within the maximum number of timesteps is a loss. This is in addition to more explicit ways of losing, such as dying due to collision with an enemy. As its computation budget increases, vanilla MCTS manages to avoid most of its explicit losses—only to end up losing via timeout instead. This makes sense if we think of many classic arcade games as consisting of two layers of challenge: avoid dying, and while doing so, achieve a goal. Vanilla MCTS becomes much better at the first challenge as it is given time to perform more iterations, but only slightly better at the second one. This suggests that much of the future challenge in the GVG-AI competition lies in the goal-achievement part; vanilla MCTS can mostly avoid dying, but it often nonetheless can’t win.

## II. BACKGROUND AND RELATED WORK

MCTS [4] is an anytime search algorithm: it has a core iteration loop that can be stopped when it runs out of computation budget, returning the best estimate it has come up with so far.

Given more time, it will generally perform better, but existing research has found mixed results regarding what that scaling curve looks like.

Since search trees grow exponentially with depth, there is some heuristic reason to believe that performance will scale with the logarithm of computation time, meaning each doubling of computation time will produce a linear increase in playing strength. This is indeed what some researchers have reported, for example on the game Hex [5]. But other experiments, on Go, have reported diminishing returns with repeated doubling of the computational budget, as performance plateaus rather than continuing to increase linearly [6]. This paper supplements the existing knowledge on MCTS performance scaling by adding results on a fairly large set of single-player games to the existing studies that have focused on specific two-player games.

Besides explicit tests of scaling, which are clearly the most relevant related work, investigating algorithm performance curves has been used for several other purposes in games. Togelius and Schmidhuber [7] propose using the performance curve of a genetic algorithm repeatedly playing a game as a fitness function to measure game quality: good games, they hypothesize, are learnable at a moderate pace, leading to mastery neither too quickly nor too slowly. This is a reinforcement learning formulation rather than a planning formulation, so the x-axis of their performance curve represents numbers of games played, rather than time budget within a single game as here, but nonetheless the ideas are closely related.

Several other researchers have recently used ratios between different algorithms’ performance on a game, called a relative algorithm performance profile (RAPP), as part of a game-quality fitness function [8], [9], [10]. This is based on a hypothesis that the strength difference between good and bad

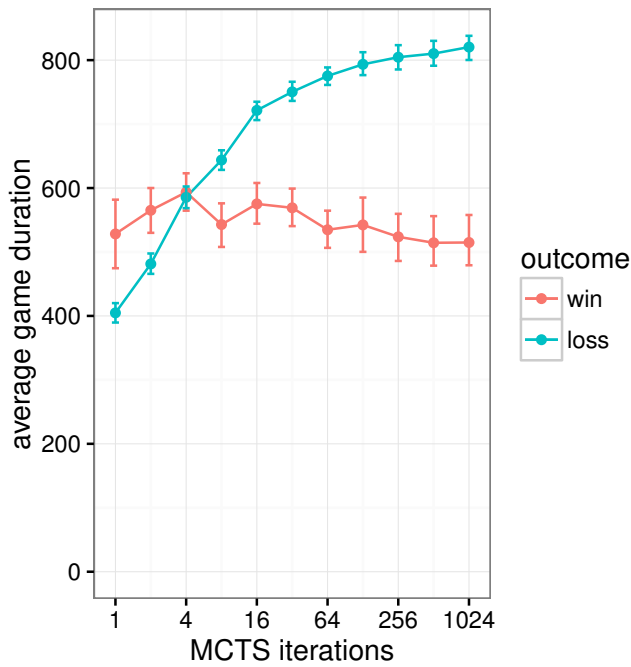


Fig. 2. Average game duration by number of MCTS iterations and win-loss outcome.

agents playing a game can be used as a proxy for game quality, or at least as a filter of bad-quality games: a game on which a random agent performs as well as an agent employing a smarter strategy likely lacks interesting depth. The single-algorithm performance curves I investigate here can be seen as a generalization of RAPPs to look at not only ratios between algorithms with specific settings (such as random play vs. a specific MCTS configuration), but also curves with parameter variation of a single algorithm. Although generating games is not the purpose of the present paper, properties of an MCTS performance curve may be interesting to use as part of a game-generation fitness function. For example, games where MCTS performance plateaus early, versus late, versus increases linearly, may pose different types of challenges (at least, as such challenge is viewed by the MCTS agent).

### III. METHODOLOGY

Since I'm investigating the scaling curve of MCTS as its computation budget is increased, the basic methodology is simple: have the agent repeatedly play each game in the GVG-AI corpus starting from a small computation budget, then double the computational budget and re-run the same trials on all the games.

Tests were run using a minor variant of the GVG-AI framework, starting from the April 4, 2016 revision in its GitHub repository.<sup>4</sup> I modified the framework to use a limit on MCTS iterations, rather than a time limit, as the method of time budgeting. This change was made in order to make the tests more reproducible. A time limit of, for example, 40

milliseconds or 80 milliseconds per move only produces comparable data if run on exactly the same hardware, with system load and other factors held constant. Since I'm running a large number of trials, it was convenient to use cloud-computing resources to carry out the experiments, an environment where it's not possible to assume that every trial will be run on identical hardware with identical system load. A time budget of, for example, 32 or 256 MCTS iterations, on the other hand, is completely reproducible on any hardware.

As a point of reference to anchor the MCTS iteration counts in this paper with the wall-clock time limit of 40 milliseconds in the GVG-AI competition, in my tests I found that 40 ms is enough time to run around 30 MCTS iterations on average, although this varies significantly depending on the specific game and hardware. Therefore, since the tests in this paper run from 1 through 1024 iterations, they represent a time budget starting at about 1/30 of the current competition's time budget, and going up to about 30x.

The specific implementation of vanilla MCTS I test here is the one included with the GVG-AI SDK as the `sampleMCTS` controller, and described in [1]. It uses an exploration-exploitation constant of  $\sqrt{2}$  and a play-out depth of 10 moves, with cutoff states evaluated by giving them a large positive score if a win, large negative score if a loss, and otherwise the current point value. I chose this controller because it is already used as a baseline in the GVG-AI competition, widely available to every GVG-AI competition participant, and has reported competition results going back several years; therefore what happens to its performance if the current competition's time budget were significantly increased serves as a useful baseline. There are of course many other things about this baseline that could be varied; here I vary only the number of iterations, not any other parameter choices, though doing so would be interesting future work.

The GVG-AI game corpus includes 62 games, each of which comes with 5 levels. I run 10 trials of each level, i.e. 50 of each game, for the tests with MCTS iteration limits of 1 through 64, and (to reduce computational resources needed) 5 trials of each level, or 25 per game, for the 128-iteration through 1024-iteration experiments. The difference is visible in the slightly larger error bars on the higher-iteration-count data points in Figures 1–3; the data is otherwise completely comparable. For each of these trials, I record whether the outcome was a win or loss, and the number of timesteps to end of game. Games are allowed to run for a maximum of 1000 timesteps. The GVG-AI framework also defines a point score, but we don't use it here, since the MCTS controller is mainly trying to maximize its wins vs. losses rather than playing for points, and comparing points across games is often not very illuminating in any case.

### IV. RESULTS

The top-level result is shown in Figure 1, plotting win rate (across all trials of all games) versus MCTS iteration limit. This shows a rapid increase in win rate initially, as the iteration limit is increased from its very low starting point of 1, followed

<sup>4</sup><https://github.com/EssexUniversityMCTS/gvgai>

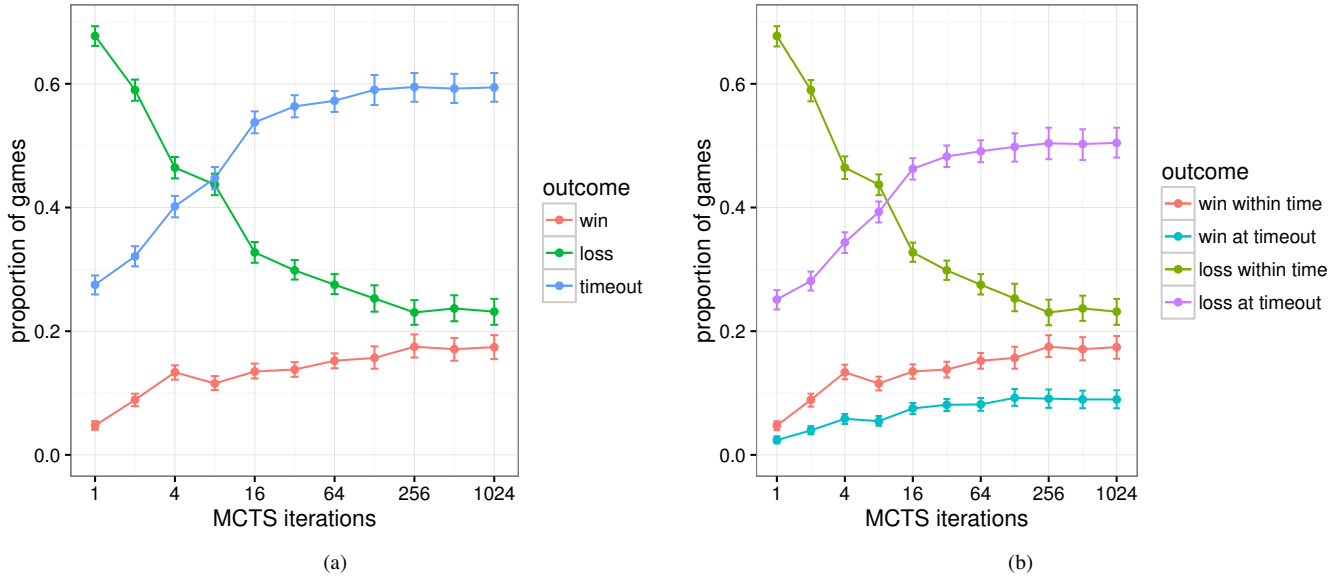


Fig. 3. Same data as shown in Figure 1, but reinterpreted as either 3- or 4-valued outcome rather than the 2-valued win/loss outcome. In (a), the three outcomes are win, loss, or timeout. In (b), whether a game is a win or loss is crossed with whether it was a timeout or not, producing 4 possible outcomes.

by diminishing returns up towards a plateau of about 26% past the 256-iteration trials.

#### A. Overall scaling

The version of the win-rate curve with a logarithmic x axis (b) gives a more detailed look at the scaling properties. The general expectation that MCTS should scale logarithmically with increased iterations is partly confirmed: the scaling does look logarithmic, i.e., linear on the log-axis graph, up until the plateau past 256, although with some anomalies. There is a higher slope from 1-4, followed by a lower slope from 16 to 256, interrupted by a strange but significant dip in performance at 8. The reason for this dip is not entirely clear, but is visible across a number of specific games as well, when looking at the results broken down per-game in Figure 4.

The main takeaway from the top-level results is therefore that overall performance of the `sampleMCTS` agent would improve only modestly if the GVG-AI competition's time budget allowed for more than the current 30 or so MCTS iterations, either through increasing time budget or (more likely) optimizing the code for rollouts: from around 22% to 26% overall win rate. Therefore, my initial hypothesis that the GVG-AI competition's challenge might be somewhat illusory, due mainly to the small time budget and unoptimized code, is not confirmed: the baseline `sampleMCTS` agent would not start dominating the competition even if its time budget were increased by 10x+. And based on the clear plateau in performance, it is unlikely to do so even if it were given still more time than that.

#### B. Types of wins and losses

While running these experiments, I noticed that experiments with higher iteration limits were taking much longer to complete, by more than would be expected just from the increased

number of MCTS iterations. Investigating further, the reason appears to be that when given more computation time, the MCTS agent plays much longer games. And specifically, its losses drag out for longer, while its wins stay at about the same length, as shown in Figure 2.

In fact not only is the MCTS agent playing longer games in its losses as its time budget increases, but an increasing proportion of losses come right at the maximum length limit of 1000 timesteps. This suggests that significant qualitative changes of play are taking place that are somewhat masked what looking only at the win-rate results (since long losses and short losses are still losses). One way of making these changes more visible in the overall results is to change from scoring games on a 2-outcome scale of win-loss, to a 3- or 4-outcome scale that treats timeouts when the maximum game length is reached differently.

Figure 3 reinterprets the overall win-rate data using two alternate ways of treating timeouts. On the left (a), a 3-outcome scoring is used, with games resulting in either a *win* (within time), a *loss* (within time), or a *timeout*, treated as different from either a win or loss. From this it can be seen that while wins slowly increase, the biggest swing in overall performance is that, given more computation time, the MCTS agent manages to convert losses into timeouts. On the right (b), the timeouts are further broken down into wins at timeout and losses at timeout, which shows that the biggest swing is specifically from losses within time, to losses at timeout.

From this view of the data, the MCTS agent can actually be said to come close to mastering the GVG-AI games if the goal were *not to lose*, with timeouts treated as non-losses: while its win rate plateaus at a not-that-impressive 26%, it brings its overall loss-within-time rate down to a mere 22%, i.e. it manages to avoid explicitly losing (except by timeout)

in 78% of games. This quite large gap in two ways of looking at what constitutes good performance leads to a hypothesis that the current GVG-AI competition has two distinct types of challenges, and a vanilla MCTS agent can master one but not the other: first, avoid losing, and then, figure out how to win. As one of this paper’s anonymous reviewers aptly pointed out, however, what a timeout means varies by type of game, sometimes indicating partial mastery and other times, especially in puzzle games, not indicating much success at all: “It’s an accomplishment to stay alive in Pac-Man even if you don’t eat all the dots, but it’s not an accomplishment in Sokoban if you put the blocks in an unwinnable state and then run out the clock”.

Further study would be needed to clarify the nature of the challenges posed by the different games. The fact that controllers do get better at avoiding explicit losses suggests that there is challenge involved in doing so, but it may be that in most of the GVG-AI games, winning requires a more complex policy than avoiding explicit losses does, which the controller is unable to find. For example, avoiding explicit losses in some of the games requires only short-term reactive behavior such as avoiding an enemy, while winning requires putting together a sequence of steps to achieve a goal. The sampleMCTS controller’s 10-depth search cutoff would further make it entirely unable to find winning plans in games requiring longer sequences of action. This finding also suggests that the choice of whether a GVG-AI game should result in a win or loss at timeout has a significant effect on the challenge posed, if judged by the headline win–loss rate; most of the current GVG-AI games result in a loss at timeout, but a few result in a win.

The different insight into performance given by looking at only wins and losses, as in Figure 1, and by treating timeouts separately, as in Figure 3, suggests that future analyses of algorithm performance on the GVG-AI corpus may want to report both measures, in order to provide a fuller view of the algorithm’s playing strengths and weaknesses.

### C. Per-game results

Figure 4 breaks the results down for each of the 62 games, showing win rate, represented by brightness of the table entry, as MCTS iterations increase. The table is ordered from top to bottom by the sum of win rates across all trials for that game, i.e. games at the top are overall won by the MCTS agent more often than those at the bottom.

A few noticeable aspects are worth pointing out. First, a relatively small number of games, about a dozen, are the only ones that really differentiate performance of the lower-iteration and higher-iteration MCTS agents. Most games are insensitive to iterations: in more than half, the MCTS agent rarely wins, regardless of time budget, and in a few, it mostly wins even with small time budgets. Only a few games, such as *sequest* and *racebet2*, seem to present the MCTS agent with a difficulty curve, where it performs better as it computes more. And a few games are completely mastered with increasing iterations: the most stark cliff is for *intersection*, which reaches an 100% win rate at mere 4 iterations, but is

not so easy that it can be mastered with only 1 or 2 iterations. The odd decrease in performance at 8 iterations observed in the overall results can also be seen in a number of individual games here, such as *plaqueattack* and *sheriff*.

The rather few number of games that show performance differentiation has implications for uses of algorithm performance profiles as a fitness function in game generation. Algorithm performance profiles are only a meaningful stand-in for difficulty curves or challenge depth if we can blame a lack of performance differentiation in a specific game on the *game* rather than the *algorithm*. The many GVG-AI games on which vanilla MCTS fails to achieve any win rate at all implies it may not be a great candidate for use in such fitness functions, since it will reject many games which do have challenge depth but which it is simply not able to play.

## V. CONCLUSIONS AND FUTURE WORK

The experiments in this paper investigate scaling the baseline vanilla MCTS controller used in the GVG-AI competition from about 1/30 of the current competition’s time budget through about 30x of the current competition’s time budget. The two purposes of doing so were to better understand MCTS scaling properties, using the 62 games in the GVG-AI corpus as the testbed, and to better understand the challenge posed by the GVG-AI competition.

Regarding scaling, I found performance increased roughly with the logarithm of increased computation time, although with some anomalies in the slope, up to a plateau at around 256 MCTS iterations, after which performance did not increase further. The plateau implies that even massive amounts of computational power would be insufficient to solve the challenges in the current competition using the baseline MCTS agent. This may be due to the 10-depth cutoff in the Monte Carlo rollouts, or it may be due, as Perez *et al.* [1] hypothesize, to suboptimal estimates produced by closed-loop MCTS on stochastic games.

The initial skeptical hypothesis regarding the GVG-AI competition’s medium-term challenge is refuted by these results. I had worried that the competition might really be too easy; that if rollouts were optimized so that agents were given, say, 10x the effective time budget they have now, the competition might become trivial, with the baseline controller mastering the games. However, I find that is not the case. Instead, performance plateaus past 256 MCTS iterations, which is roughly 10x the time budget of the current GVG-AI competition, and furthermore plateaus at only a 26% win rate. Therefore the GVG-AI competition would remain interesting, at least relative to the current baseline, even if orders of magnitude more computation time were given to the agents entering the competition, through either an increase in time budget or optimizations to the framework.

I do however find that quite a lot of change in gameplay is happening beneath the headline win-rate data. As time budget is increased, MCTS takes longer to lose the games that it’s going to lose. And, it often loses them in a qualitatively different way, by hitting the maximum game length (which

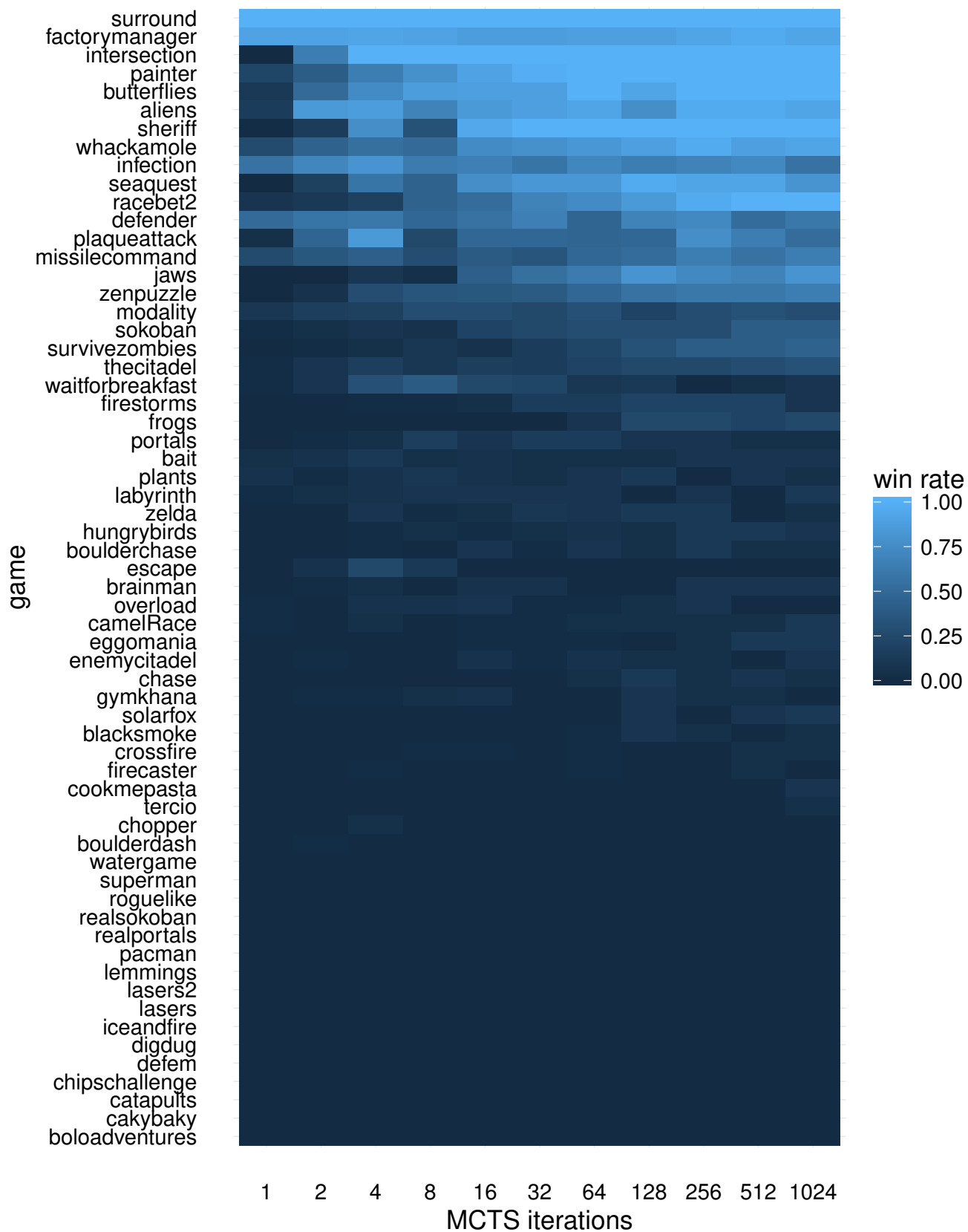


Fig. 4. Win rates for each of the 62 individual GVG-AI games, as number of MCTS iterations is increased. Brighter table entries represent higher win rate. Games are sorted by sum of win rates across MCTS iterations.

in many GVG-AI games is a loss through timeout), rather than succumbing to one of the explicit loss conditions, such as colliding with an enemy. When the overall results are reinterpreted using a 3-valued outcome of win, loss, or timeout, the vanilla MCTS agent does manage to avoid losses in the majority of games, at the larger time budgets, bringing its “non-loss” rate up to 78%—much more impressive than its 26% win rate—with the majority of games ending in a timeout.

This suggests to us two conclusions. First, it may be helpful to look beyond the headline win rate in the GVG-AI competition when comparing agents, and treat games that timeout differently from those where a win or loss is recorded within time. This provides a fuller view of what the agent is doing, and which kinds of challenges it is succeeding or failing at. Secondly, it suggests that in the current corpus of GVG-AI games, the challenge is structured so that not-losing is easier than winning. Further work might help clarify why this is; for example, it may be that in the majority of the games a relatively simple controller can avoid losing, perhaps with even a very simple reactive policy like “move away from enemies”, but more complex planning may be needed to achieve win conditions. One approach to confirming whether this is the case could be to solve for optimal policies for several of the games and investigate their structure.

Future work should look at more algorithms beyond the one baseline whose scaling properties I test here. In this paper, only the time budget in the baseline MCTS controller is varied, but there are many variations of MCTS and quite a few other parameters that can be varied. MCTS can also be compared on this corpus with the scaling of other algorithms, such as traditional full-width search.

#### ACKNOWLEDGMENT

This work was supported in part by the European Union FP7 grant 621403 (ERA Chair: Games Research Opportunities). Thanks to the reviewers for helpful comments and suggestions.

#### REFERENCES

- [1] D. Perez, S. Samothrakis, J. Togelius, T. Schaul, S. Lucas, A. Couëtoux, J. Lee, C.-U. Lim, and T. Thompson, “The 2014 general video game playing competition,” *IEEE Transactions on Computational Intelligence and AI in Games*, 2015.
- [2] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, and S. Lucas, “General video game AI: Competition, challenges and opportunities,” in *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, 2016, pp. 4335–4337.
- [3] T. Schaul, “An extensible description language for video games,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 4, pp. 325–331, 2014.
- [4] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, “A survey of Monte Carlo tree search methods,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.
- [5] B. Arneson, R. B. Hayward, and P. Henderson, “Monte Carlo tree search in Hex,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, pp. 251–258, 2010.
- [6] A. Bourki, G. Chaslot, M. Coulm, V. Danjean, H. Daghmen, J.-B. Hoock, T. Hérault, A. Rimmel, F. Teytaud, O. Teytaud, P. Vayssière, and Z. Yu, “Scalability and parallelization of Monte-Carlo tree search,” in *Proceedings of the 7th International Conference on Computers and Games*, 2010, pp. 48–58.
- [7] J. Togelius and J. Schmidhuber, “An experiment in automatic game design,” in *Proceedings of the 2008 IEEE Symposium on Computational Intelligence and Games*, 2008, pp. 111–118.
- [8] T. S. Nielsen, G. A. B. Barros, J. Togelius, and M. J. Nelson, “General video game evaluation using relative algorithm performance profiles,” in *Proceedings of the 18th Conference on Applications of Evolutionary Computation*, 2015, pp. 369–380.
- [9] —, “Towards generating arcade game rules with VGDL,” in *Proceedings of the 2015 IEEE Conference on Computational Intelligence and Games*, 2015, pp. 185–192.
- [10] J. Kowalski and M. Szykuła, “Evolving chess-like games using relative algorithm performance profiles,” in *Proceedings of the 19th Conference on Applications of Evolutionary Computation*, 2016, pp. 574–589.



# Modeling Player Decisions in a Supply Chain Game

Yifan Sun\*, Chisheng Liang\*

Northeastern University  
Boston, MA

Email: \*{yifansun, cliang, kaeli}@ece.neu.edu

Steven Sutherland†

University of Houston-Clear Lake  
Houston, TX

†sutherland@uhcl.edu

Casper Harteveld‡, David Kaeli\*

Northeastern University  
Boston, MA

‡c.harteveld@neu.edu

**Abstract**—Player decision modeling can provide useful guidance to understand player performance in serious games. However, current player modeling focuses on high-level abstraction of player behavior rather than decision-level player modeling, and is predominantly applied to entertainment games. In this paper, we describe an approach from game design to data mining and data analysis to determine detailed player decision patterns. We illustrate this approach with *VistaLights*, a supply chain game we developed based on a recent oil spill event in Houston. With this game, we set up a within-subjects experiment to study decision making under varying circumstances, specifically to consider whether/how a recommendation system can improve human decisions. Using a series of data analysis techniques we built a coarse-grained decision model as well as a fine-grained model to compare players' actions on the game outcomes. The results confirm the need for decision-level modeling and show an ability of our approach to both identify the good and bad decision patterns among players.

## I. INTRODUCTION

Player modeling has become more popular and essential for game design to appeal to a broad audience. Player modeling techniques aim to abstract player behavior patterns and have been successfully applied to game development [1], [2], self-adaptive games [3], [4], and agent design [5]. These techniques would be useful for serious games too; however, thus far applications have been rather limited [6], [7].

Unfortunately, the existing work in player modeling typically classify players based on their high-level behavioral statistics. No matter which machine learning or data mining approaches are used, aggregate properties such as play time or number of actions tend to be used to categorize players into a limited number of classes. Although general clustering based on player features can be useful, this likely does not provide the depth and accuracy needed to understand the dynamics of any game, whether a serious or entertainment game. For example, with this high-level approach it is difficult to determine what the key decisions are that lead to a poor or good performance. Exactly this kind of information is critical for improving the effectiveness of serious games. Therefore, we argue that the existing work needs to be complemented with decision-level modeling and decision-by-decision evaluation. The motivation for this paper is to demonstrate the usefulness of this low-level approach for serious games.

In order to explore how to study human decision-making behaviors in serious games, including how player decisions can be improved by providing help in the form of decision aids, we have developed a supply chain game from scratch

called *VistaLights*. In this simplified but realistic simulation game, players manage a port by prioritizing ships and dealing with disruptions, specifically oil spills. The game is inspired by the recent oil spill event at the Port of Houston [8]. Although a simplification of reality, this game provides a complex dynamic decision making environment where optimization techniques cannot find the optimal solution strategy and simple analytical techniques do not reveal why certain players performed as they did. By developing it ourselves, we have complete control over what happens in the game, allowing us to systematically study player decisions. In this paper we report the findings of our initial pilot study, where we set up a within-subjects experiment with three levels that vary in the use of a recommendation system (no recommendation, recommendation, and recommendation with justification).

Our contributions are threefold. First, we illustrate through a detailed description of designing and evaluating *VistaLights* how to develop and study a serious game for decision making, and demonstrate how to assess a well balanced design, which is needed to analyze player performance. Second, we detail a generalized approach not limited to *VistaLights* to understand the impact of player decisions on game outcomes from a high-level and decision-level perspective, and highlight the limitations of clustering techniques. Third, we report our findings from analyzing player behavior in *VistaLights*, including how players engaged with the recommendation systems.

## II. BACKGROUND

### A. Serious Games

Serious games, games with a non-entertainment purpose, are increasingly used in various fields, from health to business, to study and improve human behavior [9]. As all games are essentially about making decisions, it is key to identify how players make decisions to increase the effectiveness of serious games. First, by identifying player behavior the design can be adjusted accordingly to maximize the impact the game is attempting to achieve. For example, when players make poor decisions, the game can recognize this and provide personalized feedback or adjustments. Second, players themselves can then identify which types of players they are and how they need to improve their decision making.

Typically, player decisions are evaluated according to a normative model, and then players receive feedback accordingly to improve their behavior. Such evaluations would still benefit from player modeling to be able to personalize the game,

and the limited work where player modeling has been applied to serious games has exactly done this, by modeling players according to normative models [6], [7]. However, in complex dynamic games such as *VistaLights* this typical approach will not be sufficient because it is difficult to determine upfront what the key decisions are. But even in simpler games unexpected behavior may happen—actions not identified by the normative models—and identifying how these actions impact the game outcomes will be beneficial. We argue that this type of identification requires a different approach to player modeling, one that considers decision-level analysis.

### B. Player Modeling

In terms of player modeling approaches, machine learning and data mining techniques, especially clustering techniques, have been widely accepted [10], [11]. For example, Drachen et al. [1] use Emergent Self-Organizing Maps to cluster high-level player behavior features, such as completion time and number of deaths. The clustering result is used to improve the game and determine whether the player is following the game designer's intention. However, even with advanced machine learning algorithms, modeling human player decisions can be difficult due to the large data dimensions and the uncertainty of human behavior [12]. Other approaches have also been applied to better understand player decisions. Holmgård et al. [2] use generative agents as personas to characterize and discriminate human players. They show that a high-level abstraction of human decisions is possible. In our research, we use similar approaches to cluster human decisions, as well as some novel solutions to identify a more fine-grained analysis, and apply these in the context of a serious game.

## III. DESIGN

In this section, we describe the design of *VistaLights*<sup>1</sup>, which we use as a research environment to study player decisions and the role of decisions aids. We discuss the context for the development, how to play it, and how the scores are calculated.

### A. Context

We modeled the game after the Port of Houston. Based on discussions with stakeholders, we understood that when a disruption occurs, representatives from different industries in addition to several authorities discuss what actions to take. The different industries concern: breakbulk, dry bulk, and liquid bulk. The port authority takes care of container ships but also of special ships such as cruise ships. Actions basically involve prioritizing certain ships and implementing mitigation strategies. The resiliency to bounce back from a disruption is of interest to everyone involved because the port is a shared infrastructure on which everyone's productivity depends.

In building the first game version, we focused on disruptions caused by oil spills. Specifically, we used the oil spill that happened in March 22, 2014 as an example. At that date, a collision occurred between an oil barge and a ship at a critical node in the network (blocking the Houston Ship Channel

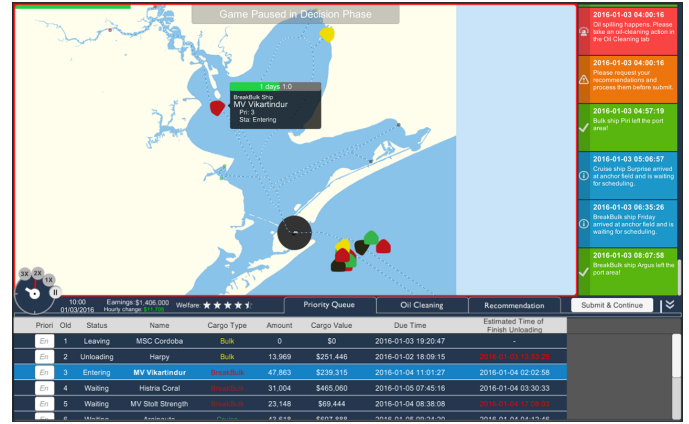


Fig. 1: A screen shot of *VistaLights*. The main screen shows the map of the port with the ships; on the right are the messages; at the bottom is the priority queue and control panel.

where all ships must pass to enter and exit the port), causing an almost complete standstill in the port. Interestingly, a cruise ship was required to wait right outside of the port and could not enter due to the spill. The decision was made to let the cruise ship go through the oil spill before cleaning it up. We were also inspired by a decision aid and monitoring system that is used to schedule ships. This system is essentially a combination of a visualization of the entire port as well as priority queue of ships and their characteristics. It formed the basis for the game's interface and gameplay.

### B. Gameplay

Like any simulation game, in representing the object of interest we made simplifications. For example, the current version is a single player game where the player single-handedly decides what actions to take. The player's goal is to manage the port by prioritizing ships and dealing with disruptions when they occur. In managing the port there are two benchmarks to consider: earnings and welfare. Earnings is a quantitative economic score based on the efficient use of the port's infrastructure; welfare is a qualitative composite score that considers the environment and reputation.

In the game, players see a map of the Port of Houston with a network of channels that ships use to navigate to the docks to unload their cargo. Players cannot directly control the ships; they can only change the priorities in the priority queue, which lists all of the ships with their names, current status, industry type, cargo amount and value, due time, and estimated time of unloading (Fig. 1). Each ship is assigned a unique priority value and ships with higher priority will be scheduled first. Lower priorities will be scheduled when no conflicts exist. In prioritizing ships, players will need to maximize the occupancy rate of the shipping lanes and docks while minimizing penalties that result from ships being overdue.

Players must further decide how to respond to the oil spill when it occurs. Other than the null-option (leaving the oil spill alone), players have three options to clean up the oil: burning,

<sup>1</sup>For game and source code, see <http://hdl.handle.net/2047/D20213074>

dispersants, and skimmers. Each option varies in cost, clean up time, impact on traffic, and impact on welfare. Additionally, with burning and skimmers the traffic can only resume after the spill has been cleaned up; with the null-option and dispersants the traffic can continue at a lower speed. Players can postpone their decision to allow time-critical ships to go into port but at the cost of a welfare penalty.

The game is divided into two phases: the decision and simulation phase. During decision phases, players can take actions. Although the simulation time is paused during these phases, players need to submit their decisions within a certain time. During the simulation phases players cannot take action; however, they can retrieve information about the ships that would be useful to make informed decisions during the next decision phase. The game spans several days in the port and decision phases occur every six hours of port operation (i.e., one simulation phase covers six hours of port operation). To help players be aware of what is happening in the port, they receive messages categorized on events throughout (Fig. 1). Players can change the speed of the simulation time during the simulation phases with a control panel.

#### C. Score Calculation

The player's goal is to maximize the economic score without reducing the overall welfare. The economic score is increased by unloading cargo, and calculated by multiplying the amount of cargo by the cargo value. It is decreased by due time penalties if ships fail to unload on time in addition to cargo maintenance costs: the longer the cargo stays on the ship, the higher the cargo maintenance cost will be. The cost for oil cleaning will also be subtracted from the earnings made. We express the economic score as the total earnings generated after a number of days and as the average earnings per hour.

Welfare is a qualitative score with a value between zero and five (represented as stars). It is negatively affected by overdue cruise ships and how the oil spill is handled. When a cruise ship is overdue, it continues to decrease proportionate to the number of passengers until all passengers have left the ship. With the oil spill, it continues to decrease proportionate to the amount of oil until it is cleaned up. An additional penalty is applied for the chosen solution because solutions such as burning and dispersants have further implications for the environment. The welfare score recovers at a constant but slow rate; however, if it becomes zero, players lose the game.

### IV. METHODS

Our goal for the pilot study with *VistaLights* was to explore decision making in a serious game. For the study we implemented a within-subjects experimental design; however, the scope of this paper is specific to evaluating modeling player behavior within this space, not to evaluate the manipulations themselves, which were intentionally implemented to observe decision making under varying circumstances.

#### A. Participants

Participants were recruited at Northeastern University and University of Houston-Clear Lake. At the first University

primarily students in Computer Engineering volunteered to participate ( $N = 26$ ); at the second University it concerned solely students in Psychology who participated for credit ( $N = 11$ ). No demographic information was collected.

#### B. Materials

The game *VistaLights* that is provided as the material for the study has four levels. The first level is a tutorial that explains step by step how to play the game with a pop-up screen. The level itself is a short level that ends in two days of simulation time. The other three levels have been designed according to the experimental design described in detail in the following section. These levels end in five days of simulation time. Based on playtests we roughly estimated that it would take an hour to play all levels if players would make use of increasing the simulation speed at times. We set the maximum time during the decision phases at two minutes. In the remainder of the paper we refer to the first level as the tutorial and to the other three levels as Challenge 1, 2, and 3, respectively.

We varied the four levels in terms of a number of level characteristics and manipulations, both which we discuss in detail below. These controlled variation allow us to maintain some consistency over the types of decisions the players must make and to identify how players learn to modify their decisions across the game, while allowing enough variability to isolate the impact of specific variables on player decisions.

1) *Level Characteristics*: We varied the levels in terms of ships, oil spill location, and goals. First, we populated each level with 30 different ships. We randomized all the ship values between realistic values for one level first. For example, we calculated the arrival time by multiplying a random number between zero and one and multiplying this by three days. For the other levels we kept the same values except for industry type, arrival time, and due time. For those characteristics we calculated a new random value. We made these variations to make sure players experience different scenarios, and are therefore not inclined to take the exact same decisions.

Second, we varied the location of the oil spill between three locations for Challenge 1, 2, and 3; no oil spill occurred during the tutorial. The three locations were chosen such that they would have the same impact on the game; however, it gives players the illusion that there is variation and that they cannot predict what will happen. Every oil spill happens around the same time, after two days, with a few hours difference between each level. Unlike the oil spill, we used the exact same network with the same number and type of docks for all four levels. For each industry type (breakbulk, dry bulk, liquid bulk, and cruise ships) we included two docks and mapped them to how these industry types are located in the Port of Houston.

Third, we varied the earnings and welfare goals between levels. At the start of each level, players receive a message that specify the earnings and welfare targets that they have to obtain. We determined realistic target goals for both the revenue and welfare based on prior playtests. For example, the first level is much harder and so we set the target goals lower than for the subsequent levels.

2) *Level Manipulations*: Our manipulations pertaining to the levels are related to the provision of a decision aid that provides recommendations regarding how to prioritize the ships. In addition to a level where no decision aid is provided, we settled for this initial study on two variations: a recommendation without and with justification. These manipulations allowed us to explore how players make decisions when confronted with a complex, unfamiliar task under varying circumstances. This manipulation resulted in the following: players received no recommendations for Challenge 1; they received recommendations regarding prioritizing ships with no justification for the recommendation for Challenge 2; and they received recommendations regarding ship prioritization and justifications for those recommendations for Challenge 3.

In both recommendation systems, the player receives up to three recommendations in each decision phase. For each level (other than the tutorial) there are 20 decision phases across the five days. Recommendations are based on the ships that are at the time of the decision phase waiting outside of the port; ships that are moving and unloading are ignored. Both systems will first check if any cruise ship is going to be overdue or is already overdue. Then they will check on overdue or nearly overdue ships. From there recommendations involve prioritizing ships with the highest total cargo value.

Recommendations with suggested priority values are made in the order described above and in the format of “Consider to prioritize ship <ship\_name> to priority <x>.” We decided for both systems to recommend a specific priority because in that way we can determine whether players comply with the advice. The difference between the two recommendation systems concerns the justification. Justifications are short explanations such as “Because this ship has a high cargo value.”

The three recommendations are provided but only after players requested the advice. As players cannot progress without requesting advice, we essentially required them to do this. This seems unnatural but was implemented to ensure that players would consider the recommendations, which is the manipulation they are exposed to, and not play the game without the recommendations provided. Once the recommendations are requested, players then need to accept or reject each one of them before they can progress to the priority queue to make their changes. In this priority queue, they see the old value as well as the suggested value by the recommendation system.

We designed the recommendation system to be imperfect and purposely did not inform players about its logic. For example, players could prioritize moving ships whereas the recommendation system does not include these. Therefore, a better performance is possible by not completely relying on the recommendations.

### C. Procedure

We implemented a within-subjects experiment where every participant experiences every condition. There are three conditions: • *Challenge 1* (no recommendation), • *Challenge 2* (recommendation), and • *Challenge 3* (recommendation with justification). The tutorial level was included to make sure

that players first learn how to play the game before starting the experiment and to minimize the practice effect from Challenge 1 to Challenge 2. We did not vary in the order of the conditions because seeing the recommendation, and most certainly the recommendation with justification, would likely affect further play. A consideration for just three conditions was a possible fatigue effect. We requested that all players finish the experiment in one session to minimize any possible bias from contextual factors for when and how players engage with the game. Although the within-subjects design creates the possibility of learning effects and behavior constancy, it was necessary to determine whether prior performance in the game predicted future compliance with the recommendations.

The implementation was different at both Universities. At the first University the game was distributed with instructions to play over e-mail. Participants were requested to take an hour and complete all levels in one sitting. At the second University players participated in person in a lab setting. After the facilitator briefed them about the purpose of the study and how to play, they were assigned to a computer where the game was installed. This variation in play context was not intentional. It was pragmatic and based on the infrastructures in place for the researchers involved. For both locations selected participants were contacted for a debriefing interview to understand what strategies they used in the game.

### D. Data Analysis

We applied several different data analysis techniques to evaluate player behavior. We first analyzed the distribution of the players’ earnings and welfare results to examine if the game is well balanced. If the game is too difficult or too easy, it becomes harder to distinguish players, and what decisions can be considered good or bad. We then examined the role of the recommendation systems by considering the relationship between compliance rates with player performance. We also compared a typical poor and top player from our sample with a hypothetical player who does not take any actions (“No Action Player”) and one who complies with everything that the recommendation systems suggest but does nothing else (“Compliance Player”). We performed these analyses to show if the recommendation systems are useful and if players can perform better than the imperfect recommendation systems.

To discover the decision patterns, we performed a coarse-grained analysis and a fine-grained analysis. For the coarse-grained analysis, we linked the players’ decisions on the oil cleaning solutions and the ship priorities with their results on earnings and welfare. For the ship priorities we categorized different groups of priority level changes and counted the changes per group. We then performed a clustering analysis and used the resulting clusters of player decisions to predict players’ win/lose probability. The clustering was done by first whitening [13] the number of actions of each priority group category, and then using Ward method [14] as the criterion to perform hierarchical clustering [15] analysis.

For each of the above analyses, all 37 players’ data are considered; however, for the fine-grained analysis we focused

on only two players to illustrate our proposed method of analyzing specific player decisions in serious games. By comparing the cycle-by-cycle decisions of two similar players, we explored how individual decisions impacted the performance trajectory of each player, and tried to infer what strategies the players had used. To complement our strategy inferences, and see if players intentionally took certain actions, we compared this analysis with our interview notes of selected players who articulated their strategies to us.

## V. RESULTS

In this section, we discuss first the overall player performance and then the role of the recommendation system, followed by the influence of the oil cleaning decisions and the priority change decisions. Finally, we illustrate the play trajectories of two players and show how making similar decisions can still lead to drastically different results in a dynamic environment such as *VistaLights*.

### A. Player Performance

The final results on earnings and welfare are depicted for each player in each level in Figure 2. The horizontal lines represent the earnings target in each subfigure; the vertical lines represent the welfare target. The resulting quadrants show if players won the game (upper right quadrant), lost because they failed to meet the earnings goal (lower right quadrant), lost because they failed to meet the welfare goal (upper left quadrant), or lost because they failed to meet both earnings and welfare goals (lower left quadrant). Except for Challenge 3, where few players lost on both earnings and welfare, players were well-distributed over the quadrants, suggesting that the targets were fair and that the game had a reasonable difficulty.

There was no immediate clear pattern for previous player performance predicting future performance. A player who did well in Challenge 1 did not necessarily do well in Challenges 2 and 3. The likelihood that a player got the same result in Challenges 1 and 2, Challenges 1 and 3, and Challenges 2 and 3 were 51%, 62%, 48%, respectively. This finding was also illustrated by the win percentage: the percentages of players who won were 58% (21 of 36), 29% (10 of 34), and 60% (18 of 30) for Challenges 1, 2, and 3, respectively, suggesting that Challenge 2 may have been more difficult than the other challenges. Of the 30 participants who completed all levels, five players won every challenge and five lost every challenge. Losing two or winning two was split almost equally as well.

### B. Recommendation System

In Challenges 2 and 3, we provided players with an imperfect recommendation system. However, this recommendation system was most certainly beneficial. Table I illustrates the performance of the recommendation system by modeling a player who complies with all provided recommendations and does not do anything else. We compared this performance by modeling a player who does not take any action at all and a typical poor and top player from our sample. This table shows that even the poorest players made some good decisions but

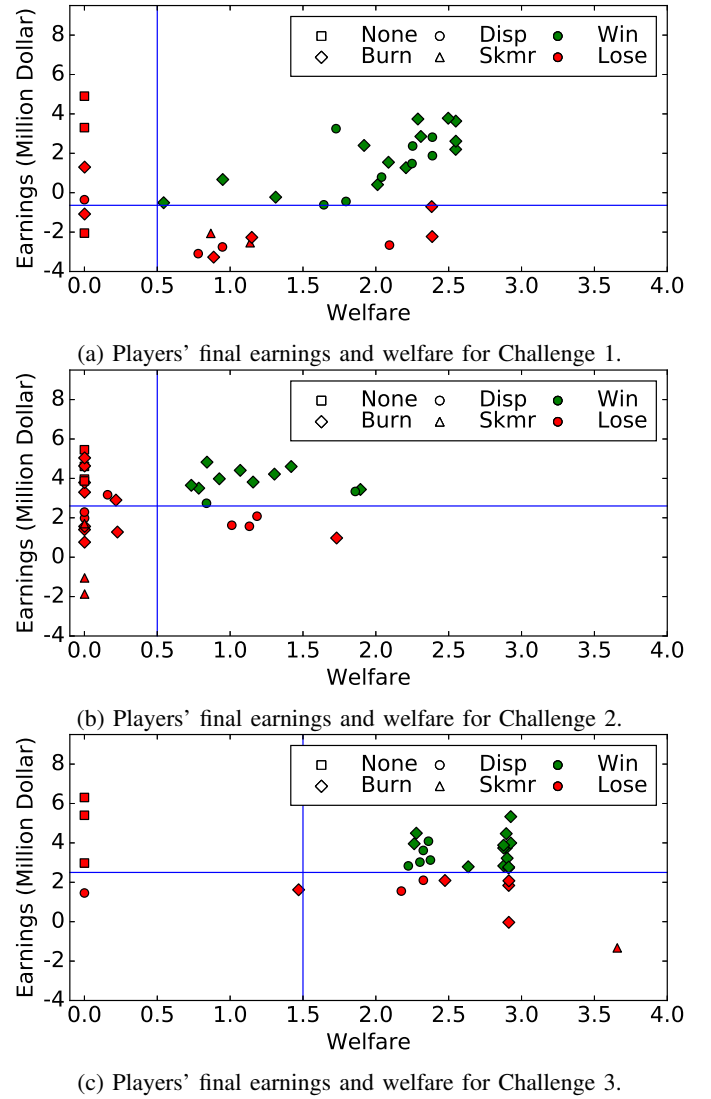


Fig. 2: Distribution of the players' final results. The blue lines are the target earnings and welfare. The shapes represent the oil spill solutions players chose.

that if they only complied with the recommendation system, doing nothing else, their final results would have been better. In fact, if a player simply accepted all the recommendations and chose dispersants or burning as their oil spill solution, they would have exceeded the targets and won both challenges. The table also shows that players can outperform the recommendation system. The system only suggested three priority changes and only for ships that were waiting outside of the port. Therefore, room existed for human decision making to outperform simple recommendation compliance in the game.

In terms of the usage of the recommendation systems, we calculated the rate with which players accepted the recommendations and then the rate that they complied by actually implementing the advice. A strong correlation existed between the acceptance rate and the compliance rate for both Challenge 2,  $r = .96$ ,  $p < .001$ , and Challenge 3,  $r = .93$ ,  $p < .001$ .



TABLE I: The performance comparison between a hypothetical player not taking any action, a typical poor player, a hypothetical player who complies with every recommendation and nothing else, and a typical top player. For the hypothetical players we chose dispersants when an oil spilling happens.

Player	Item	C2	C3
No Action Player	Earnings	-3.89M	-0.76M
	Welfare	0.00	1.98
Poor Player	Earnings	-1.87M	1.45M
	Welfare	0.00	0.00
Targets for Players	Earnings	2.60M	2.50M
	Welfare	0.50	1.50
Compliance Player	Earnings	3.13M	3.15M
	Welfare	1.48	2.30
Top Player	Earnings	4.60M	5.33M
	Welfare	1.42	2.93

.001. Therefore, when players accepted the advice, they also complied by implementing the advice. The actual acceptance rates were similar across challenges: in both challenges, a small majority of the advice was implemented ( $M_2 = .55$ ,  $SD_2 = .29$ ;  $M_3 = .55$ ,  $SD_3 = .33$ ). The rates were, in fact, similar because reliance on the recommendation system in Challenge 2 was a strong predictor of reliance in Challenge 3, accounting for 81% of the variability ( $R^2 = .81$ ). Knowing that the recommendations do help, it seems that more players could have benefited from an increased reliance. However, we did not find any relationship in the data between the compliance rates and performance, suggesting that any differences we found between the challenges in terms of performance could not be explained by the use of the recommendation systems. Therefore, other factors determined how well players performed.

Additionally, performance on previous challenges (earnings, welfare, and whether or not the player won the challenge) did not predict whether players would rely on the recommendations for later challenges. It should be expected that players who had seen that they were unsuccessful would have been more likely to comply with the recommendations, but this was not the case. Because compliance did not predict performance and previous performance did not predict future compliance, despite the fact that simply following the recommendations and doing nothing else would lead to success, we argue that players were not able to effectively decide when the recommendations were beneficial and when they were not. This result may have been due to players not having received immediate feedback about their decisions to comply. The sum of their decisions was reflected as a final score at the end of the challenge, making it difficult to identify which decisions should be changed.

### C. Oil Cleaning Decisions

The response to the oil spill was one of the key player decisions, and our results confirmed this. Figure 3 shows the distribution of the earnings and welfare for each challenge by different oil cleaning solutions. The trend of how solutions impacted earnings was clear and similar from challenge to challenge. The null-option did not cost anything and did not

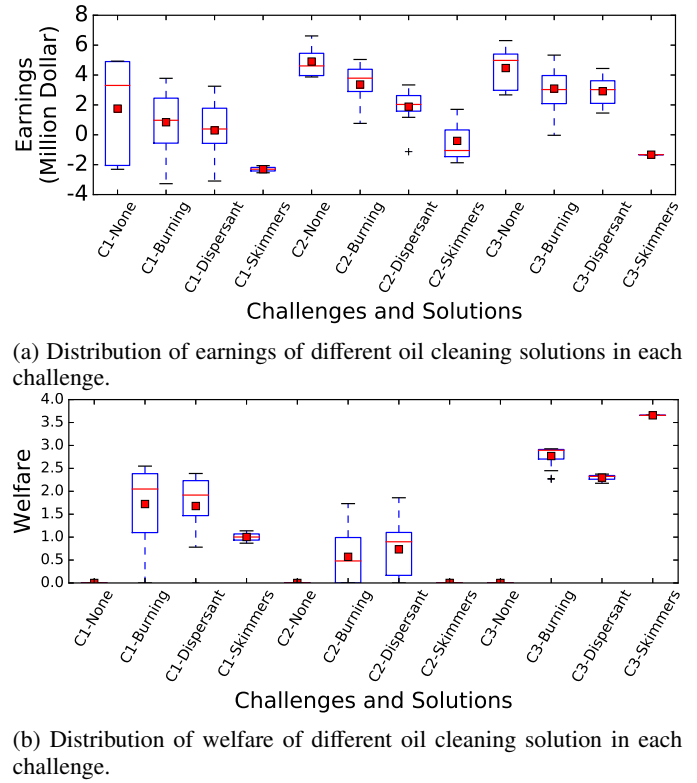


Fig. 3: Oil cleaning solution impact on the final result.

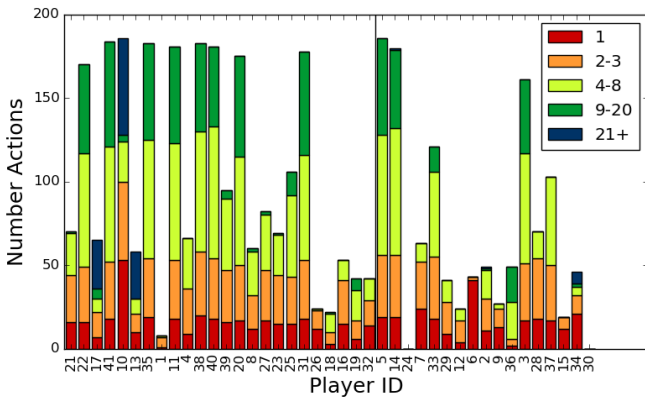
cause traffic to stop. Burning did stop traffic but had the advantage that it cleared the oil relatively quickly. Few players chose skimmers and the figure highlights that it may not have been the best solution for earnings. Its advantages were likely overruled by the penalties for overdue cargo.

The solutions were also related to welfare. The null-option was a guarantee for losing the game. However, the patterns for the effects across the challenges were dissimilar. Further investigation revealed that this had to do with the arrival of the cruise ships around the oil spill. With Challenge 2, two cruise ships arrived shortly after the oil spill, explaining why so many players lost that challenge due to welfare. In contrast, there was little variation in Challenge 3 because no cruise ship arrived during the oil cleaning period. The variance of burning was also a result of cruise ship scheduling. Those that implemented burning after letting the cruise ships pass first, did better on welfare.

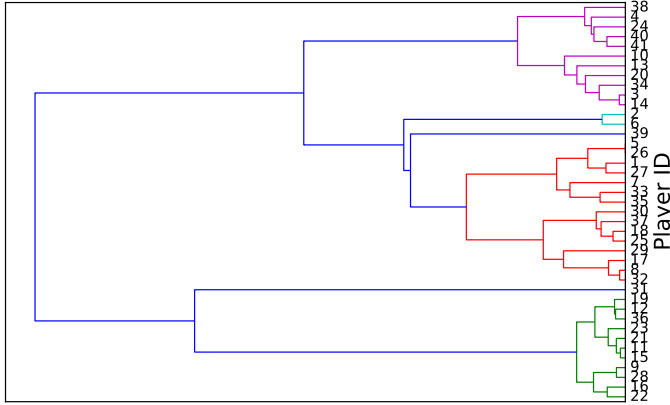
### D. Priority Change Decisions

In addition to the one-time oil-cleaning decision, players were tasked with changing the ship priorities. As shown in Figure 4a, we counted the number of priority change actions and what kind of priority changes players made in Challenge 1. We divided priority changes into the following groups: priority 1, priorities 2 and 3, priorities 4 to 8, priorities 9 to 20, and priorities higher than 21. In this figure we further placed those who won to the left side of the vertical line and those who lost to the right side. This figure shows that players who won Challenge 1 made more changes, and that the number of





(a) The number of actions for assigning new priorities in Challenge 1, categorized by type of priority change. To the left of the vertical line are the players that won. In both the win and lose groups, players are sorted by their earnings in descending order.



(b) Cluster analysis on the player's priority changes

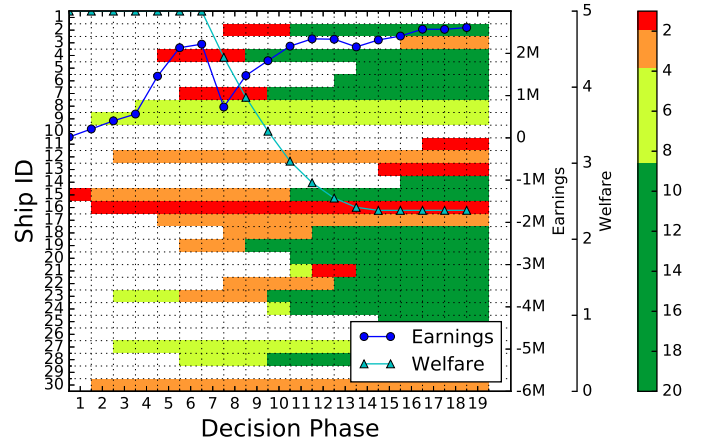
Fig. 4: Analysis of players decision pattern

actions predicted performance earnings with 26% variability ( $R^2 = .26$ ). This prediction was not evident in Challenges 2 and 3, and the number of actions did not increase due to the recommendation systems. Therefore, it may be that players who put in more effort in Challenge 1 were able to change the priorities of critical ships, whether intentionally or by chance.

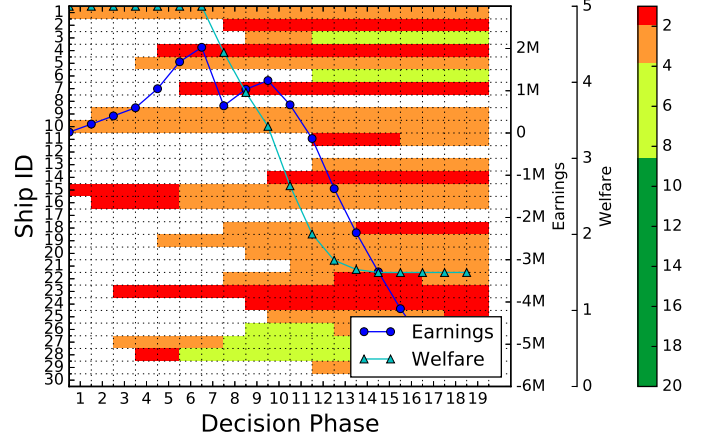
To better understand player decision patterns, we conducted a cluster analysis, and the result is depicted in Figure 4b. There were three major clusters, and from the top to bottom, they can be characterized as the medium amount of priority change, the low amount of priority change, and the high amount of priority change. An outlier was player 31 who had a large amount of actions that set priority to 1 and to a value of higher than 20. A related subgroup included Players 2 and 6 who also moved ships to a priority higher than 20. By explicitly moving the ship to the end of the priority queue, they freed up the main channel and let more urgent ships enter.

#### E. Detailed Play Trajectories

From these results we were unable to get a fine-grained decision evaluation, such as determining if a particular decision was good or bad. For example, Players 12 and 15 made



(a) The actions and status of each decision phase of Player 2.



(b) The action and the status of each decision phase of Player 34.

Fig. 5: Analysis of individual play trajectories. The trajectory includes the scores on earnings and welfare across the decision phases as well as the priority changes for all 30 ships. The priority changes are color coded with the priority groups.

almost the same number of priority changes and they both used dispersants as oil cleaning solutions. However, they ended up with different results. To perform a more fine-grained decision evaluation we selected two seemingly similar players, Players 2 and 34, and analyzed their play trajectories. Their trajectories are illustrated in Figure 5. This figure shows their earnings and welfare over time, in addition to what changes they made to the 30 ships over the 20 decision phases.

According to the earnings curve, there was a turning point at decision Phase 9. After Phase 9, the earnings of Player 2 kept increasing while the earnings of Player 34 sharply plunged until the end of the game. Player 34 must have made some critical decision just before or during this phase that caused the avalanche effect. The first difference was that player 34 moved Ship 3 to Priority 1 at Phase 9 and this ship was a breakbulk ship. When we recreated the game according to the player's log file, we noticed that by that time, the breakbulk docks were already heavily overloaded and Ship 3 was going to be overdue according to the system estimation. During that decision phase, Player 34 made the natural decision to move Ship 3 to a higher

priority. As a consequence, ships at breakbulk docks and ships that were already in the port had to wait for Ship 3 to move in. Dock utilization was then significantly reduced by this action, and other ships may have, as a result, become overdue. Similar actions were made by Player 34 repeatedly throughout the game, including prioritizing another breakbulk ship, Ship 26, at Phase 9. We consider such decisions to be bad decisions.

Another key difference between Player 2 and Player 34 was that Player 2 moved Ships 4 and 19 to a low priority. By explicitly moving ships to the end of the priority queue, Player 2 let the ships that had already unloaded wait close to the dock before moving out of the port, which prevented outgoing ships to occupy the channel. We consider such actions as good decisions and Player 34 did not take these.

An interview with Player 2 confirmed that the strategy was intentional. He stated: "I gave lower priority to the unloading ships and higher priority to cruise and cargo with high value." Others also articulated this strategy. However, Player 14 did not use that strategy and still outperformed Player 2 in Challenge 1. From the interview it becomes clear that this player has a well thought out strategy for playing, and is not a top player by chance: "First, I usually set high priority to the overdue cargo or cargo that is going to be due. Second, I watched the docks closely, each color represents a particular type of industry and I always tried to keep the docks busy all the time and if I see any dock is empty, I will give higher priority to a proper ship to enter that dock. Third, I will also keep an eye on the values. The more expensive the ship's cargo, the higher priority the cargo will have." Player 14's strategy of watching the under utilized docks was only used by the high-performing players.

## VI. CONCLUSION

In this paper we presented a supply chain game called *VistaLights* that we developed to model human decisions. The results from a within-subjects experimental pilot study with 37 participants highlighted that this is a fair and valid environment to study decision making: the participants were reasonably well distributed in terms of their performance and participants had to make the right decisions to perform well. Our results illustrate that straightforward analyses do not illuminate what happens in these complex dynamic decision-making environments and that fine-grained decision models are needed in addition to data mining techniques.

Key insights are that certain critical bad decisions can negatively impact the outcomes. Therefore, it is of importance to identify when people are about to make such bad decisions. Likewise, critical good decisions can positively impact the outcomes. Identification and recommendation of such good decisions would help improve the effectiveness of serious games for training, and may even impact the actual workplace. The results also highlight that participants may need to rely more on recommendation systems, even if they are imperfect, especially when their own performance suggests an inability to succeed. Finally, when it comes to the recommendation

systems we show that some people are simply more willing to rely on these than others, as illustrated by the only significant predictor of future recommendation compliance having been past compliance. This signifies the importance of player modeling as it identifies individual differences.

Our work can serve as an example of how to design for games to model player decisions and then how to analyze these decisions. We acknowledge there are limitations to our game and our analysis approach. To understand how the recommendation system can help in the decision making process, we plan to perform studies with more players, and compare the performance with additional players that do not have recommendations in any challenge. We plan to expand this pilot work with additional design variations and by developing analytical techniques that will help to analyze fine-grained decisions on a larger scale. The analytical model should also be implemented together with the game to give players real-time guidance as part of the recommendation system.

## ACKNOWLEDGMENT

We thank the Greater Houston Port Bureau for their input.

## REFERENCES

- [1] A. Drachen, A. Canossa, and G. N. Yannakakis, "Player modeling using self-organization in Tomb Raider: Underworld," in *Computational Intelligence and Games*, 2009, pp. 1–8.
- [2] C. Holmgård, A. Liapis, J. Togelius, and G. N. Yannakakis, "Generative agents for player decision modeling in games," *Foundations of Digital Games*, 2014.
- [3] O. Missura and T. Gärtner, "Player modeling for intelligent difficulty adjustment," in *Discovery Science*, 2009, pp. 197–211.
- [4] J. Togelius, R. De Nardi, and S. M. Lucas, "Making racing fun through player modeling and track evolution," in *SAB Workshop on Adaptive Approaches for Optimizing Player Satisfaction in Computer and Physical Games*, 2006.
- [5] P. Spronck and F. den Teuling, "Player modeling in Civilization IV," in *AIIDE*, 2010, pp. 180–185.
- [6] S. Göbel, V. Wendel, C. Ritter, and R. Steinmetz, "Personalized, adaptive digital educational games using narrative game-based learning objects," in *International Conference on Technologies for E-Learning and Digital Entertainment*, 2010, pp. 438–445.
- [7] Y.-G. Cheong, R. Khaled, C. Grappiolo, J. Campos, C. Martinho, G. P. Ingram, A. Paiva, and G. Yannakakis, "A computational approach towards conflict resolution for serious games," in *Foundations of Digital Games*, 2011, pp. 15–22.
- [8] H. Rice, A. Hassan, and L. Olsen, "Ship channel remains closed after oil spill," 2015. [Online]. Available: <http://goo.gl/L3nmZm>
- [9] C. Harteveld, *Triadic game design: Balancing reality, meaning and play*. London, UK: Springer, 2011.
- [10] B. G. Weber and M. Mateas, "A data mining approach to strategy prediction," in *Computational Intelligence and Games*, 2009, pp. 140–147.
- [11] A. Drachen, C. Thureau, R. Sifa, and C. Bauckhage, "A comparison of methods for player clustering via behavioral telemetry," in *Foundations of Digital Games*, 2013, pp. 245–252.
- [12] A. Drachen, R. Sifa, C. Bauckhage, and C. Thureau, "Guns, swords and data: Clustering of player behavior in computer games in the wild," in *Computational Intelligence and Games*, 2012, pp. 163–170.
- [13] A. Kessy, A. Lewin, and K. Strimmer, "Optimal whitening and decorrelation," *ArXiv e-prints*, Dec. 2015.
- [14] J. H. Ward Jr, "Hierarchical grouping to optimize an objective function," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 236–244, 1963.
- [15] S. C. Johnson, "Hierarchical clustering schemes," *Psychometrika*, vol. 32, no. 3, pp. 241–254, 1967.

# Sonancia: a Multi-Faceted Generator for Horror

Phil Lopes  
Intitute of Digital Games  
University of Malta  
Msida, Malta

Email: louis.p.lopes@um.edu.mt

Antonios Liapis  
Intitute of Digital Games  
University of Malta  
Msida, Malta

Email: antonios.liapis@um.edu.mt

Georgios N. Yannakakis  
Intitute of Digital Games  
University of Malta  
Msida, Malta

Email: georgios.yannakakis@um.edu.mt

**Abstract**—Fear and tension are the primary emotions elicited by the genre of horror, a peculiar characteristic for media whose sole purpose is to entertain. The audience is often lead into tense and fearful situations, meticulously crafted by the authors using a narrative progression and a combination of visual and auditory stimuli. This paper presents a playable demonstration of the Sonancia system, a multi-faceted content generator for 3D horror games, with the capability of generating levels and their corresponding soundscapes. Designers can also guide the level generation process, by defining an intended progression of tension, which the level generator and sonification will adhere to.

## I. INTRODUCTION

Digital games can be defined as a synthesis of different types of content, such as sound, visuals and level architecture, which provide different interactive entertaining experiences for players [1]. Procedural content generation (PCG) systems have often concentrated on the level architecture facet of creativity [2], however more recently various efforts have been made for procedurally generating playable experiences that blend a variety of different faceted content such as: audio and gameplay [3]; level architecture and gameplay [4]; and audio, level architecture and visuals in earlier studies of the Sonancia generation [5]. The orchestration of audiovisual content can augment and provide meaningful experiences as digital games are primarily an interactive audio-visual activity. This is especially true in the survival horror genre, where the effective use of lighting, audio and labyrinths provide tense and frightening experiences [6].

This paper showcases a playable demonstration of Sonancia, a multi-faceted procedural content generator for the horror genre. Sonancia can generate multi-faceted levels by first evolving the level's layout, lighting, enemy positioning and 3D diegetic audio placement. Once a structure has been created, the system will then pick and allocate background audio assets within the level (i.e. sonification) for the creation of its soundscape. Sonancia targets the survival horror genre as a specific case study for the concepts of multi-faceted blending, due to the genre's heavy reliance on the audio and visual facets [6]. It also follows a specific game progression scheme based on tension, where the focus is to lead the audience into uncomfortable and fearful situations. Sonancia attempts to simulate this scheme by generating levels through a *tension frame*, allowing designers to define the intended rise and fall of tension guiding the level generation process,

which will adhere to the defined frame. Sonification will also take into account the tension frame by proxy through the level generation. It allocates background audio assets by following the level's tension progression and a crowdsourced ranking annotation attributed to each background audio asset available in the Sonancia library. The methodologies used in this demo are detailed in [5], while the autonomous generation of levels using dramatical tropes are described in [7]. In this demo, however, several features are added to the evolutionary blending component of Sonancia, including the placement of lighting and 3D audio (not to be confused with background audio) within the level layout. For demonstration a play-through of the demo can be found here<sup>1</sup> and downloaded here<sup>2</sup>.

## II. SONANCIA

The generated game consists of a 3D first-person survival horror game, taking place in a dark underground dungeon. These dungeons consist of multiple rooms that are interconnected through doorways. Each dungeon contains an ancient statue that players must reach in order to complete the level, acting as the objective. These dungeons also host several monsters that will run after the players, if they are in their line of sight. If the player gets hit more than three times by a monster, they will be defeated and the level will restart. Players can not directly fight back, but they can run away and use stealth to get past and progress through the level.

### A. Level Generation

Sonancia generates levels using a genetic algorithm without recombination, and a human or machine defined framing of tension. Due to spatial constraints, this paper will briefly describe the algorithm, but for the interested reader a more detailed description can be found in [5].

The level structure in the genotype consists of an array of integers that represent a tile within the level and what room it belongs to. Mutations will shift walls, divide rooms and move, place or remove monsters, lights, doors, and 3D audio in rooms, if possible. The level generation currently applies the following constraints: than one type of the same object (i.e. monster, light or 3D audio) may be placed in the same room; rooms must have at least 5 tiles; and a path between the start and objective must exist.

<sup>1</sup><http://goo.gl/2RQDG6>

<sup>2</sup><http://goo.gl/jcXcBu>

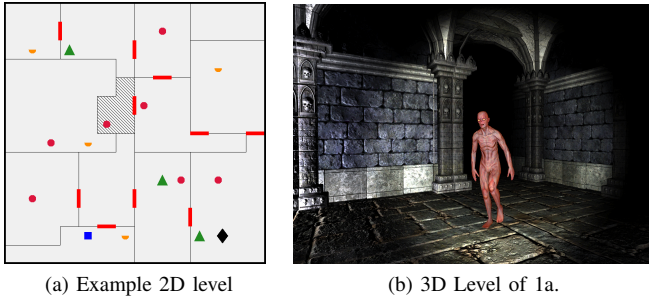


Fig. 1: Figure 1a depicts a 2D illustration of a dungeon, consisting of rooms and interconnecting doors. Thin black lines represent walls, while thick red lines represent doors. Monsters are represented as green triangles, the objective is represented as a blue square, lights are circles, 3D audio are half circles, and the black diamond is the player position in Fig. 1b. The darker room is the player's starting room. Figure 1b consists of gameplay footage of the level represented in Fig. 1a.

The frame represents the intended progression of the rise and fall of tension that the level generator must adhere to. More specifically, a frame consists of a line graph where the y-axis represents a numerical representation of tension, while the x-axis consists of a specific room in the level. This frame is then used as the fitness function for the level generator, which informs the optimal number of rooms for the level, and how monsters, lights and 3D audio will be distributed throughout the level. For example, adding a monster or a 3D audio (to a lesser degree) raises the tension of that room; adding lights decreases the tension slightly as it is less dark compared to other rooms, allowing players to orient themselves better. Sequences of rooms that do not increase in tension will continuously suffer a tension decay, in order to simulate the player relaxation after a stressful event.

### B. Sonification

Level sonification consists of the selection and allocation of *background audio* assets that will loop during gameplay. Two constraints are enforced by the sonification algorithm: only one background audio piece may be allocated per room, to avoid cacophony; and audio pieces may not repeat in the same level. Sonification will also select and position *3D audio* assets within the rooms, which were defined by the previous level generation process.

*Background Audio* is allocated according to the characteristics of a generated level, such as the distribution of monsters, lighting and which rooms the 3D audio assets were placed. More precisely it follows the tension that was obtained through the generation process (the actual tension progression), and not the previously defined tension frame. This allows sonification to more closely adapt a soundscape that follows the exact characteristics of the level, instead of the conceptual specifications defined by the tension frame.

All background audio assets in the library (40 in total) are ranked based on human-annotated tension preferences. The *global order* of sound tension is derived through the pairwise preference test statistic [8], which is used by the sonification algorithm to select which sounds are better suited for a specific section of the level based on both the tension progression of the generated level and the annotated audio piece. For the interested reader a video demonstrating some examples of background audio is available here<sup>3</sup>.

*3D Audio* consists of audio cues that play only once, when players trigger them within the environment. Although the level generation specifies these sounds should be located in, the sonification system places the audio trigger event within the room itself, by calculating the mid-point of the shortest path between two doors of the room. If a room contains only one door, the trigger is placed at the room's centre. For the interested reader a video demonstrating examples of 3D audio is available here<sup>4</sup>.

## III. FUTURE WORK

Future work will include enhancements to the sonification system such as the creation of a data-driven model of tension, allowing the system to automatically annotate and more accurately choose between which audio assets in comparison to others. Extensive user testing will also be conducted, for both validating and improving the multi-faceted game content generation algorithms developed.

### ACKNOWLEDGMENT

This work was supported, in part, by the FP7 Marie Curie CIG project AutoGameDesign (project no: 630665) and the Horizon 2020 project CrossCult (project no: 693150).

### REFERENCES

- [1] A. Liapis, G. N. Yannakakis, and J. Togelius, "Computational game creativity," in *Proceedings of the International Conference of Computational Creativity*, vol. 4. Springer, 2014, pp. 71–78.
- [2] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *Conference on Computational Intelligence and Games*, vol. 3, no. 3, pp. 172–186, 2011.
- [3] A. K. Hoover, W. Cachia, A. Liapis, and G. N. Yannakakis, "Audioinspace: Exploring the creative fusion of generative audio, visuals and gameplay," in *Proceedings of the EvoMusArt conference*. Springer, 2015, pp. 101–112.
- [4] M. Cook and S. Colton, "Multi-faceted evolution of simple arcade games," in *CIG*, 2011, pp. 289–296.
- [5] P. Lopes, A. Liapis, and G. N. Yannakakis, "Targeting horror via level and soundscape generation," in *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.
- [6] I. Ekman and P. Lankoski, "Hair-raising entertainment: Emotions, sound, and structure in silent hill 2 and fatal frame," *Horror video games: Essays on the fusion of fear and play*, pp. 181–199, 2009.
- [7] P. Lopes, A. Liapis, and G. N. Yannakakis, "Framing tension for game generation," in *Proceedings of the International Conference on Computational Creativity*, 2016.
- [8] G. N. Yannakakis and J. Hallam, "Ranking vs. preference: a comparative study of self-reporting," in *Affective computing and intelligent interaction*. Springer, 2011, pp. 437–446.

<sup>3</sup><https://youtu.be/dtcV6v5fHHA>

<sup>4</sup><https://youtu.be/FUhzM6UiudI>

# Ms. Pac-Man Versus Ghost Team

## CIG 2016 Competition

Piers R. Williams\*, Diego Perez-Liebana<sup>†</sup> and Simon M. Lucas<sup>‡</sup>

School of Computer Science and Electronic Engineering,

University of Essex, Colchester

CO4 3SQ, UK

Email: {pwillic\*, dperez<sup>†</sup>, sml<sup>‡</sup>}@essex.ac.uk

**Abstract**—This paper introduces the revival of the popular Ms. Pac-Man Versus Ghost Team competition. We present an updated game engine with Partial Observability constraints, a new Multi-Agent Systems approach to developing Ghost agents, and several sample controllers to ease the development of entries. A restricted communication protocol is provided for the Ghosts, providing a more challenging environment than before. The competition will debut at the IEEE Computational Intelligence and Games Conference 2016. Some preliminary results showing the effects of Partial Observability and the benefits of simple communication are also presented.

### I. INTRODUCTION

Ms. Pac-Man is an arcade game that was immensely popular when released in 1982. An improvement on the original Pac-Man game; Ms. Pac-Man added better graphics, additional mazes and new Artificial Intelligence (AI) behaviour for the ghosts. The primary difference that interests academics and researchers is the ghost AI. In Pac-Man the ghosts behaved in a deterministic manner. Ms. Pac-Man added a semi-random element to the ghost behaviours making them non deterministic. This non determinism vastly increased the challenge in creating an effective agent for Ms. Pac-Man.

Ms. Pac-Man has been the focus of two previous competitions. The Ms. Pac-Man screen capture competition [15] which periodically provided the agents with a pixel map of the game and requested the direction of travel. This competition only allowed the entrants to submit agents for the Ms. Pac-Man character. The second Ms. Pac-Man competition was the Ms. Pac-Man Vs Ghost Team competition [22] which was based on a simulator that mimicked the original game reasonably closely. Entrants had to submit a controller for either the Ms. Pac-Man agent or the ghost team.

This new competition adds Partial Observability (PO) to Ms. Pac-Man. PO greatly increases the challenge in creating good AI controllers. Limited information about the ghosts makes it more difficult for Ms. Pac-Man to plan effectively. Limited information about Ms. Pac-Man forces the ghosts to search and communicate effectively in order to trap Ms. Pac-Man and capture her.

Computational Intelligence (CI) has a long history of using competitions to galvanise research in game agent development. These competitions typically focus on trying to develop the strongest AI for a particular scenario although some exceptions such as the BotPrize [11] competition that focuses on developing Unreal Tournament agents that are human-like. There are

many competitions currently active in the area of games. The Starcraft competition [17] runs on the original *Starcraft: Brood War* (Blizzard Entertainment, 1998). Starcraft is a complex Real-Time Strategy (RTS) game with thousands of potential actions at each time step. Starcraft also features PO, greatly complicating the task of writing strong AI. The General Video Game Artificial Intelligence (GVGAI) competition [19] runs a custom game engine that emulates a wide variety of games, many of which are based on old classic arcade games. The Geometry Friends competition [20] features a co-operative track for two heterogeneous agents to solve mazes, a similar task to the ghost control of Ms. Pac-Man.

Previous competitions have been organised that focused on games or scenarios with PO. An early example is the classic Iterative Prisoner Dilemma (IPD) [12], a game featuring a small amount of PO in the form of the simultaneous actions of the two prisoners.

Section II contains a review of the research into the domain of Ms. Pac-Man. Section III contains a description of the alterations to the problem domain since the previous competition. Section IV concludes the paper and describes some possible future work for the competition.

### II. RECENT RESEARCH

A large amount of research has been put into both Ms. Pac-Man and the ghost teams, which is covered in depth in this section.

#### A. PacMan AI

Gallagher and Ledwich [8] investigated a simplified version of the game including for the majority of their experiments using only a single near deterministic ghost and no power pills.

Lucas [14] explored using a simple Evolutionary Algorithm (EA) in  $(N + N)$  form with  $N = 1$  or  $10$ . The EA was used to train the weights for a Neural Network.

Robles and Lucas [21] investigate writing a simple tree search method for writing an agent to play Ms. Pac-Man. This was performed on the actual game using screen capture and a simulator. The tree was formed as every possible path through the maze (depth limit 10) with information in the nodes about ghosts and pills encoded. A simulator of the game was used to evaluate the state of the game in future ticks. The simulator was not identical to the game actually played, leading to some



possible errors in judgement. The authors tried a few heuristics and found that some performed better than others.

Burrow and Lucas [3] compared two different approaches to learning to play the game of Ms. Pac-Man. The paper uses a Java implementation of the game Ms. Pac-Man that allowed easy integration of existing machine learning implementations. The two techniques used were Temporal Difference Learning (TDL) and EA. These techniques were used to train a Multi-Layer Perceptron (MLP) that was then evaluated within the game. The EA was subsequently shown to be superior to TDL.

Handa and Isozaki [10] used Fuzzy logic tuned by a 1+1 EA. The rules were tuned with the EA and consisted of a series of predefined rules about avoidance and chasing as well as pill collecting.

Wirth and Gallagher [27] used Influence Maps to drive a Ms. Pac-Man agent. Positive influence was exerted by pills and edible ghosts, whilst ghosts exerted a negative influence upon the map. The map is then checked in the four cardinal directions that Ms. Pac-Man can move in and the maximum influence is chosen.

Alhejali and Lucas [1] [2] studied the use of Genetic Programming (GP) for evolving heuristics to control Ms. Pac-Man. Some care was needed to prevent the agent from focusing too much on ghost eating instead of pill clearing by using multiple mazes.

Samothrakakis et al [24] used a 5 player  $max^n$  tree with limited tree search depth. The paper experimented with both Monte-Carlo Tree Search (MCTS) for Ms. Pac-Man and for the Ghosts. Schrum and Miikulainen [25] investigated the use of modular neural networks to control Ms. Pac-Man. The agent was developed for the same simulator as used in the Ms. Pac-Man Versus Ghost Team competition.

Flensbak and Yannakakis [5] describe their solution to the Ms. Pac-Man competition of WCCI 2008. This controller was a largely hand coded agent based around pill hunting and ghost avoidance as its primary tactics. The agent avoids ghosts within a 4x4 grid around Ms. PacMan and then collects pills. With this approach, less time is spent in danger from the ghosts before the maze is reset.

Pepels et al [18] describe their work in creating an entrant to the Pac-Man Versus Ghost Team competition (WCCI'12 and CIG'12). A MCTS agent is described in detail containing a number of enhancements and alterations designed to improve performance specifically in Ms. Pac-Man. Emilio et al [4] worked with Ant Colony Optimisation (ACO) to design an agent for Ms. Pac-Man. Two objectives are chosen to drive the agent. The first is to maximise pill collecting. The second is to minimise being eaten by ghosts. This leads to two types of ants used in the system, the *collector ants* maximising pill collecting and the *explorer ants* minimise death.

Foderaro et al [7] [6] used a tree search technique after abstracting the maze into a connected graph of cells.

### B. Ghost Control

Nguyen and Thawonmas [16] present their agent that was entered into the CEC 2011 Ms. Pac-Man vs Ghost Team Competition, subsequently winning. The agents used for this

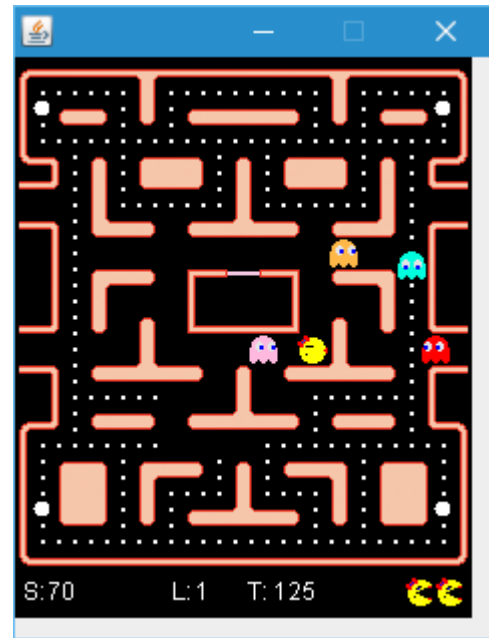


Figure 1. A view of the basic Ms. Pac-Man game



Figure 2. The various characters of the game, Left to Right: Blinky, Inky, Pinky, Sue and Ms. Pac-Man

controller were to control Pinky, Sue and Inky with MCTS whilst using a completely rule based approach for Blinky.

Wittkamp et al [28] investigate using an online learning technique - Neuro-Evolution Through Augmenting Topologies (NEAT) - to evolve the controllers for the ghosts team. Each ghost evolves separately but shares the score of the team.

Liberatore et al [13] look into the use of Swarm Intelligence (SI) to control the ghost team.

## III. THE COMPETITION

### A. The game

The game that we have based the competition on is the Ms. Pac-Man arcade game. This game consists of 5 agents, a single Ms. Pac-Man and 4 Ghost agents. The world is a maze environment, with peachy coloured walls that are non traversable. There is a ghost lair in the center, where the ghosts start and also respawn after being eaten. Pills are placed in the corridors for Ms. Pac-Man to collect as well as larger Power Pills that allow Ms. Pac-Man to consume the ghosts and score additional points. A view of the game is shown in Figure 1. The various characters in the game are shown in Figure 2. Eating a pill earns Ms. Pac-Man 10 points and eating ghosts earn 200 points for the first ghost but doubling each time up to 1600 points for the fourth ghost. The maximum points  $s$  for a maze where  $n$  is number of pills in the maze is  $s = 10n + 4 \times (200 + 400 + 800 + 1600)$ .



### B. Partial Observability

PO is the impairment of the ability of an agent to completely observe the world that it is situated within. PO in Ms. Pac-Man can vary in its implementation. We consider some simple methods that could be used, before explaining why we chose the final implementation. First we consider the approach taken in another paper, followed by some simple methods.

1) *pacman*: PO has been applied to the game of Ms. Pac-Man previously by Silver and Veness [26]. This work covered a small number of domains, one of which was *pacman* - a PO Ms. Pac-Man clone.

In *pacman*, the main agent had to navigate a 17x19 maze and eat the food pellets randomly distributed across the maze. Four ghosts roamed the maze with a simple strategy controlling them. The 4 power pills are also present, allowing *pacman* to eat the ghosts upon contact for 15 steps instead of being eaten in the usual manner.

The ghosts operate randomly unless they are within Manhattan distance 5 of Pacman in which case they chase or evade based on if he is under the effect of a power pill. This implies that the simulation does not support the usual method of ghosts respawning as non-edible before the edible time is up and could be eaten again. This is a major difference to the game that would require some serious modification to the framework in order to replicate. An alternative to replicating this is to make the minor alteration to the ghost AI that it runs away if it is edible and attacks Pacman if it is not edible.

The *pacman* agent receives a reward to his score each tick from Table I. For example, if the agent moves across an empty square it will receive a reward of -1. This should help force agents to try to finish levels as quickly as possible.

Table I: Table of rewards for *pacman*

Reason	Reward
Eating Pellet	+10
Eating Ghost	+25
Dying	-100
None of above	-1

*pacman* has a particular type of observability in which he is sent 10 observation bits. The first four refer to each cardinal direction and are high if a ghost is present in that directions Line-of-Sight (LOS). The fifth observation bit tells him if he can hear a ghost which occurs when at least one ghost is within Manhattan distance of 2. Four more observation bits refer to the presence of a wall at distance 1 in each of the cardinal directions. The final observation bit is set to high if he can smell food which occurs when a pill is adjacent or diagonally adjacent to him.

2) *Line-of-Sight*: LOS is where the agents can see in straight lines up to a limit unless there is an obstacle in the way. Obstacles are considered to be the walls in the maze. Ghosts and pills don't count as obstacles. This applies to both Ms. Pac-Man and the Ghosts and means that they can see both forwards, backwards and sideways. This method is simple to implement as well as fairly realistic based on how light travels. Agents cannot see around corners just like real people. This

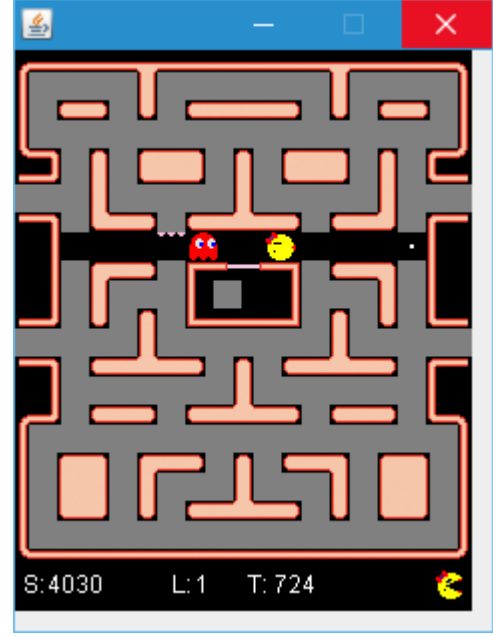


Figure 3. A view of the basic Ms. Pac-Man game with Partial Observability enabled

is similar to the standard first person view although we allow full backwards, left and right sight as well.

3) *Forward Facing Line-of-Sight*: This is an additional restriction on LOS where the agent can only observe in the direction they are currently travelling.

4) *Radius based Partial Observability*: Radius based PO is a simple technique where we consider anything within a distance  $d$  from the agent is considered visible. This technique provides a circular vision when Euclidean Distance is used, and a diamond shaped vision when Manhattan Distance is use. This allows agents to view other agents that are around corners or behind walls. This is not particularly realistic but does provide more information to the agent than LOS.

5) *Partial Observability Implementation*: The method of PO that has been implemented in Ms. Pac-Man is based on LOS. This we felt was the most realistic without being overly restrictive. The game supports a range limit to the sight - allowing some level of customisation, although at present it is larger than the longest corridor. The view generated by this restriction for Ms. Pac-Man is shown in Figure 3.

With the PO constraints, the ghost entrants must now submit a controller for each ghost. These controllers will be given a 40ms shared time budget, equal to the original competition. The ghosts will be called sequentially in order: (Blinky, Pinky, Inky and Sue). This will allow the flexibility to adjust how much time is spent on each ghost in each tick. This flexibility is useful due to the game rules forcing ghosts to have no actual decision ability when not at a junction in the maze. Locking each ghost into 10ms each would be potentially wasteful for ghosts in tunnels and doesn't leave as much time free for ghosts at junctions.

### C. Messaging

Communication is the cornerstone of teamwork and vital to the creation of co-operative agents. In the competition, the communication will be heavily controlled by the game in order to force agents to share information rather than attempt to control the actions of each other. The communication component is composed of two main parts - the messenger and the message. The messages allowed are presented in Table II.

The messages allowed will have a large impact on the ability and even potentially the design of the controllers. In early versions of the messaging system there were more messages planned, allowing the controllers to ask other controllers where they were and where they were heading. Logically it was clear that most controllers would ask for that information every tick and would receive a reply every tick (once enough time has passed). It therefore made sense to simply remove the messages asking and allow the controllers to pass on the information spontaneously. Logically the game could simply provide the information - however the effects of information delay would be lost if that were the case.

The data variable is a single integer at present due to the internal structure of the Ms. Pac-Man simulator. Locations are represented as indices of the node graph, with only a single integer required to show a location in the map. If the messaging system is expanded to include more complex messages, then a more complex system can be used. The extension to a more complex data type for messages would allow even harder messages to use instead of the perfect information messages previously discussed. A more difficult version of the game would only allow an AI to transmit that they can see Ms. Pac-Man or that they can only see Ms. Pac-Man in a certain direction. Imperfect information would lead to more possibilities such as triangulation between reports from multiple ghosts being used to improve the data received.

Messages can be either sent to a single recipient or broadcast to all ghosts on the map. The Java interface for the Messages is presented in Listing 1.

Table II: Table of messages allowed in PacMan

Message Type	Description
<b>Pacman Seen</b>	A message informing others that PacMan has been seen.
<b>I Am</b>	A message informing others where the sender is currently located.
<b>I Am Heading</b>	A message informing others where the sender is currently heading.

Other potential messages could be allowed to be passed. Some simple user defined messages for example could allow agents to declare the current strategy they are using. Ghosts could be interested in not just declaring their current position but also their state. Once a powerpill is eaten no ghost knows who has been eaten and who hasn't. An edible ghost could travel towards a chasing ghost for protection if it knew more information.

The messenger system will deliver messages at the time specified by a simple formula. The time it takes to deliver a message for the implementation can be calculated using

Equation (1). This allows a level of configurability in how quick the messages get delivered. Each message type has its own cost, the  $\delta_m$ , and the system has both a multiplier to that  $\delta_x$  and a constant delay applied equally to all messages  $\delta_c$ . This allows for example all messages to be delivered equally ( $\delta_m = 0$ ).

The messenger system at present makes no charge for delivering its messages. This allows AI agents to use as many messages as they wish. Introducing the notion of cost to a message would force the algorithms to become more careful with messages and decide whether it is worth sending the message at all. This would increase the level of difficulty for the AI agents as effective and thrifty strategies for messaging would need implementing.

The Java interface for this system is in Listing 2.

	Table III: Explanation of terms in Equation (1)
$t_d$	The tick a message will be delivered.
$t_a$	The tick a message arrives in the system.
$\delta_c$	The constant delay added to all messages equally.
$\delta_m$	The individual message delay.
$\delta_x$	The constant delay multiplier applied to message delay.

$$t_d = t_a + \delta_c + (\delta_x \times \delta_m) \quad (1)$$

Listing 1. Message Interface

```
public interface Message {

    //Gets the sender of the message
    public GHOST getSender();

    //Gets the intended recipient of the message
    public GHOST getRecipient();

    //Gets the message type of the message
    public MessageType getType();

    //Gets the data associated with the message
    public int getData();

    //Gets the tick that the message was created
    public int getTick();

    //Contains information about message delays
    public enum MessageType {
        PACMAN_SEEN(2),
        I_AM(1),
        I_AM_HEADING(1);

        private int delay;

        private MessageType(int delay) {
            this.delay = delay;
        }

        public int getDelay() {
            return delay;
        }
    }
}
```

Listing 2. Messenger Interface

```

public interface Messenger {

    //Gets a deep copy of the messenger object
    Messenger copy();

    //Updates the messenger
    void update();

    //Adds a message to the messenger
    //to be delivered as soon as it can be
    void addMessage(Message message);

    /**
     * Get all messages that are due to
     * be delivered to me this tick
     * @param querier The agent doing the querying.
     * @return The messages due (could be empty).
     */
    ArrayList<Message> getMessages(GHOST querier);
}

```

#### D. Sample Controllers for Ms. Pac-Man vs Ghosts

Having implemented PO for both Ms. Pac-Man and the Ghosts an initial trio of new controllers were also implemented that could function within PO. These controllers required the minimum amount of modification to the original basic controllers from the previous competition. The controllers, along with the original starter controllers, will be described next.

1) *StarterPacMan (COP)*: This is the original basic controller for the previous competition and works only in Complete Observability (CO) environments. This controller follows a very basic algorithm with some simple sequential rules as shown in Algorithm 1. The controller will avoid ghosts that are too close, chase ghosts that are edible or travel to the nearest pill.

---

**Algorithm 1** StarterPacMan basic algorithm

---

```

function GETMOVE()
    distanceLimit ← 20
    nearestGhost ← GETNEARESTCHASINGGHOST(limit)
    if nearestGhost ≠ NULL then
        return NEXTMOVEAWAYFROM(nearestGhost)
    end if
    nearestGhost ← GETNEARESTEDIBLEGHOST(limit)
    if nearestGhost ≠ NULL then
        return NEXTMOVETOWARDS(nearestGhost)
    end if
    nearestPill ← GETNEARESTPILL()
    return NEXTMOVETOWARDS(nearestPill)
end function

```

---

2) *StarterGhosts (COG)*: This is the original basic controller for the previous competition to control the four ghosts. It is a puppet master style algorithm, meaning it is a single block of logic that generated moves for all of the ghosts. The controller follows some basic strategies if a ghost is allowed to make a move as shown in Algorithm 2. The ghosts will run away from Ms. Pac-Man if she is able to eat the ghost, or near a power pill (Potential to eat ghost). If the previous rule doesn't apply then the ghost will 90% of the time chase Ms. Pac-Man and 10% of the time move randomly.

---

**Algorithm 2** StarterGhosts basic algorithm

---

```

function GETMOVE()
    if GAME.DOESREQUIREACTION() = False then return
    NULL end if
    pacman ← GETPACMANINDEX()
    if ISEDIBLE() OR PACMANCLOSETOPILL() then
        return NEXTMOVEAWAYFROM(pacman)
    end if
    if NEXTFLOAT < 0.9 then
        return NEXTMOVETOWARDS(pacman)
    else
        return NEXTRANDOMMOVE()
    end if
end function

```

---

3) *POPacMan (POP)*: This is a modification of the StarterPacMan where each strategy is followed if it is possible as shown in Algorithm 3.

---

**Algorithm 3** POPacMan basic algorithm

---

```

function GETMOVE()
    limit ← 20
    nearestGhost ← GETNEARESTCHASINGGHOST(limit)
    if nearestGhost ≠ NULL then
        return NEXTMOVEAWAYFROM(nearestGhost)
    end if
    nearestGhost ← GETNEARESTEDIBLEGHOST(limit)
    if nearestGhost ≠ NULL then
        return NEXTMOVETOWARDS(nearestGhost)
    end if
    nearestPill ← GETNEARESTPILL()
    if nearestPill ≠ NULL then
        return NEXTMOVETOWARDS(nearestPill)
    end if
    return NEXTRANDOMMOVE()
end function

```

---

Other than modifying the original strategies with guards against null, it was clear that a new default strategy was needed. This is because within the PO game, it was possible to proceed through the previous strategies without returning a move. This new default strategy was to simply return a random move.

4) *POGhosts (POG)*: This is a modification of the StarterGhosts where each strategy is followed if it is possible in the PO case. If there is no information available to the ghost, then the ghost will behave randomly at intersections as shown in Algorithm 4.

5) *POCommGhosts (POGC)*: This is a modification of the POGhosts but attempts to communicate each tick in order to improve its chances. If this ghost can see Ms. Pac-Man then it will send a message to everyone else. If it can't see Ms. Pac-Man then it will check if anybody else has seen it. If someone else has seen Ms. Pac-Man then it pretends it can see Ms. Pac-Man and follows the original POGhosts strategy outlined above. This controller will presumably lose capability as the message delay increases due to the reduced accuracy. The pseudo code for this is shown in Algorithm 5.

The threshold used to determine when to forget Ms. Pac-Man's location needs tuning. Every value from 0 to 200 were put to a test on 4000 games against the COP agent and 33,300 games against the POP agents. The results are

**Algorithm 4** POGhosts basic algorithm

---

```

function GETMOVE()
  if DOESREQUIREACTION() = False then return NULL end
if
    pacman ← GETPACMANINDEX()
    if pacman ≠ NULL then
      if ISEDIBLE() OR ISPACMANCLOSETOPILL()
    then
      return NEXTMOVEAWAYFROM(pacman)
    end if
    if NEXTFLOAT < 0.9 then
      return NEXTMOVETOWARDS(pacman)
    end if
  else
    return NEXTRANDOMMOVE()
  end if
end function

```

---

**Algorithm 5** POCmmGhosts basic algorithm

---

```

function GETMOVE()
  currentTick ← GETCURRENTTICK()
  if currentTick = 0 || currentTick - tickSeen
    ≥ TICKTHRESHOLD then
    lastPacmanIndex ← -1
    tickSeen ← -1
  end if
  pacman ← GETPACMANINDEX()
  messenger ← GETMESSENGER()
  if pacman ≠ -1 then
    lastPacmanIndex ← pacman
    tickSeen ← currentTick
    if messenger ≠ NULL then MESSENGER.ADDMESSAGE(...)
  end if
  if pacman = -1 AND messenger ≠ NULL then
    for message in MESSENGER.GETMESSAGES(ghost) do
      if MESSAGE.GETTYPE() = PACMANSEEN then
        if MESSAGE.GETTICK() > tickSeen then
          lastPacmanIndex ← MESSAGE.GETDATA()
          tickSeen ← MESSAGE.GETTICK()
        end if
      end if
    end for
  end if
  if pacmanIndex = -1 then pacmanIndex ← lastPacmanIndex
end if
  if DOESREQUIREACTION() = False then return NULL end
if
    pacman ← GETPACMANINDEX()
    if pacman ≠ NULL then
      if ISEDIBLE() OR PACMANCLOSETOPILL() then
        return NEXTMOVEAWAYFROM(pacman)
      end if
      if NEXTFLOAT < 0.9 then
        return NEXTMOVETOWARDS(pacman)
      end if
    else
      return NEXTRANDOMMOVE()
    end if
  end function

```

---

displayed in Figure 4 and show that the value of 50 is a good value against these two agents. Interestingly the data against the POP algorithm is significantly noisier than COP. This is presumably due to COP being deterministic and POP being

non-deterministic.

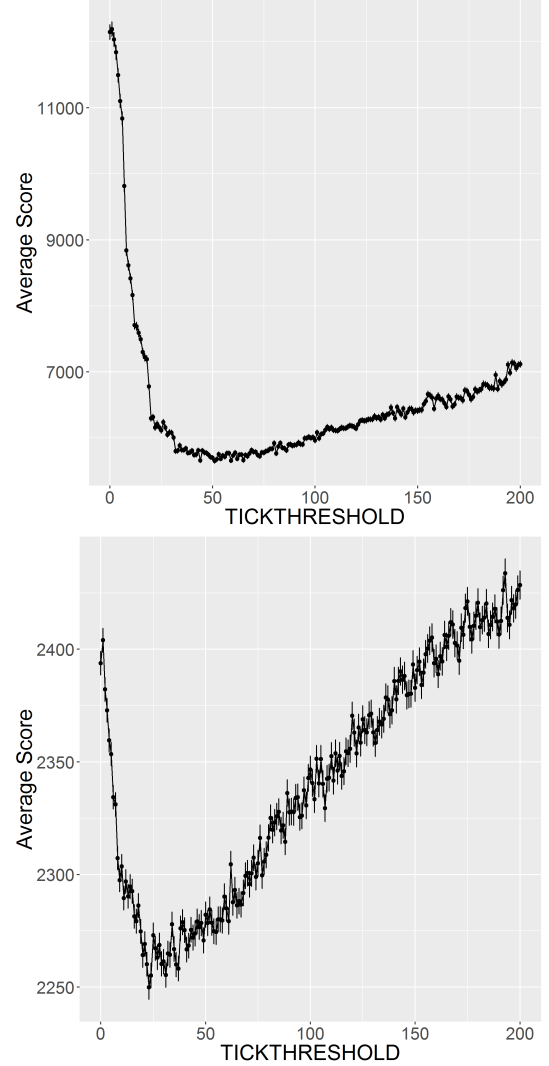


Figure 4. Tuning results of POGC against COP(Top) and POP(Bottom) both with error bars.

### E. Sample Controllers Results

These basic controllers may not represent the best agents for the game, but they do provide simple comparisons between them as they rely on the same strategy. The only difference between them is the addition of PO and the effects of this are apparent. Running these controllers over 1000 runs in each combination provides the results displayed in Table IV. It is clear that for the same strategies, PO is a large handicap to the agent. Against COG, adding PO to Ms. Pac-Man caused the score to drop to only 45% of previous performance. Adding communication abilities to the PO Ghosts allowed CO Ms. Pac-Man to achieve only 33.43% of her previous score. This is a huge difference between two very simple algorithms and clearly shows the benefits of communication in this scenario.

### F. Competition Tracks

The competition was originally run with two main tracks. The first track allowed participants to submit code to control

Table IV: Table of results after 1000 runs of different controllers

Agents	Mean Score	Std. Error
COP Vs COG	3895.67	48.23
COP Vs POG	17257.24	280.49
COP Vs POGC	5769.30	77.41
POP Vs COG	1753.52	26.97
POP Vs POG	2708.15	37.98
POP Vs POGC	2349.34	30.32

the Ms. Pac-Man character. The second track allowed participants to submit a single class to control the Ghosts.

The revived competition will also feature two tracks. The first track will allow participants to submit code to control Ms. Pac-Man but they will be operating within PO constraints. The second track will allow participants to submit 4 controllers - one for each ghost - that will be operating under PO constraints.

#### G. Entrant Ranking

While the number of entrants remains low, a round robin tournament will be used for simplicity. If this process begins to take too long, entrants will be assigned scores using the Glicko2 rating algorithm [9] as recommended for competitions [23] similar to this. These will be used to calculate matches in the competition periodically, with these matches updating the scores. The final results will be calculated with a full round robin of the top 10 ghost and top ten Ms. Pac-Man controllers before being announced at IEEE Computational Intelligence and Games Conference (CIG).

#### IV. CONCLUSIONS AND FUTURE WORK

In this paper we presented a major update to the Ms. Pac-Man Vs Ghost Team competition that will be running at CIG. We presented the PO constraint that has been added to the environment and studied the effect that the ability to communicate has on the ghosts performance when there is incomplete information available. Finally we presented the two tracks: PO ghosts and PO Ms. Pac-Man. The controllers used in this paper were of a very basic nature, and there is a great deal more potential to be realised. This competition aims to explore PO in real time games and communication within PO constraints.

There is a lot of potential work to be done in the future. The current competition still only has 4 mazes for the entrants to play on. Additional mazes can be created and included. The competition has maintained the original balance of one Ms. Pac-Man and 4 ghosts. The competition could be extended to allow for modifications to this balance, with more or less of either type of player.

At present, there is only the one method of sight for observing the environment. The competition could be extended to include additional observations such as hearing or smell. These would be less precise than sight but have the potential to provide information that sight can't.

#### V. ACKNOWLEDGMENTS

This work was funded by the EPSRC Centre for Doctoral Training in Intelligent Games & Game Intelligence (IGGI) [EP/L015846/1]

#### REFERENCES

- [1] Atif M Alhejali and Simon M Lucas. Evolving Diverse Ms. Pac-Man Playing Agents Using Genetic Programming. In *Computational Intelligence (UKCI), 2010 UK Workshop on*, pages 1–6. IEEE, 2010.
- [2] Atif M Alhejali and Simon M Lucas. Using Genetic Programming to Evolve Heuristics for a Monte Carlo Tree Search Ms Pac-Man Agent. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pages 1–8. IEEE, 2013.
- [3] Peter Burrow and Simon M Lucas. Evolution Versus Temporal Difference Learning For learning to Play Ms. Pac-Man. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 53–60. IEEE, 2009.
- [4] Martin Emilio, Martinez Moises, Recio Gustavo, and Saez Yago. Pac-mAnt: Optimization based on ant colonies applied to developing an agent for Ms. Pac-Man. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 458–464. IEEE, 2010.
- [5] Jonas Flensburg and Georgios N Yannakakis. Ms. Pacman AI Controller. 2008.
- [6] Greg Foderaro, Ashleigh Swingle, and Silvia Ferrari. A model-based approach to optimizing ms. pac-man game strategies in real time.
- [7] Greg Foderaro, Ashleigh Swingle, and Silvia Ferrari. A model-based cell decomposition approach to on-line pursuit-evasion path planning and the video game ms. pac-man. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pages 281–287. IEEE, 2012.
- [8] Marcus Gallagher and Mark Ledwich. Evolving Pac-Man Players: Can We Learn From Raw Input? In *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, pages 282–287. IEEE, 2007.
- [9] Mark E Glickman. Example of the glicko-2 system. *Boston University*, 2012.
- [10] Hisashi Handa and Maiko Isozaki. Evolutionary fuzzy systems for generating better ms. pacman players. In *Fuzzy Systems, 2008. FUZZ-IEEE 2008. (IEEE World Congress on Computational Intelligence). IEEE International Conference on*, pages 2182–2185. IEEE, 2008.
- [11] Philip Hingston. A new design for a turing test for bots. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games, CIG 2010, Copenhagen, Denmark, 18-21 August, 2010*, pages 345–350, 2010.
- [12] Graham Kendall, Xin Yao, and Siang Yew Chong. *The iterated prisoners' dilemma: 20 years on*. World Scientific Publishing Co., Inc., 2007.
- [13] Federico Liberatore, Antonio M Mora, Pedro A Castillo, and Juan Julián Merelo Guervós. Evolving evil: optimizing flocking strategies through genetic algorithms for the ghost team in the game of Ms. Pac-Man. In *Applications of Evolutionary Computation*, pages 313–324. Springer, 2014.
- [14] Simon M Lucas. Evolving a Neural Network Location Evaluator to Play Ms. Pac-Man. In *CIG*. Citeseer, 2005.
- [15] Simon M Lucas. Ms pac-man competition. *ACM SIGEVOlution*, 2(4):37–38, 2007.
- [16] Kien Quang Nguyen and Ruck Thawonmas. Applying Monte-Carlo Tree Search To Collaboratively Controlling of a Ghost Team in Ms Pac-Man. In *Games Innovation Conference (IGIC), 2011 IEEE International*, pages 8–11. IEEE, 2011.
- [17] Santiago Ontañón, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill, and Mike Preuss. A survey of real-time strategy game AI research and competition in starcraft. *IEEE Trans. Comput. Intellig. and AI in Games*, 5(4):293–311, 2013.
- [18] Tom Pepels, Mark HM Winands, and Marc Lanctot. Real-time monte carlo tree search in ms pac-man. *Computational Intelligence and AI in Games, IEEE Transactions on*, 6(3):245–257, 2014.
- [19] Diego Perez, Spyridon Samothrakis, Julian Togelius, Tom Schaul, Simon Lucas, Adrien Couëtoux, Jeyull Lee, Chong-U Lim, and Tommy Thompson. The 2014 General Video Game Playing Competition.
- [20] Rui Prada, Phil Lopes, Joo Catarino, Joo Quirio, and Francisco S. Melo. The Geometry Friends Game AI Competition. In *CIG2015 - IEEE Conference on Computational Intelligence and Games*. IEEE CIG, pages 431–438, Tainan, Taiwan, August 2015. IEEE Computer Society.

- [21] David Robles and Simon M Lucas. A Simple Tree Search Method For Playing Ms. Pac-Man. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 249–255. IEEE, 2009.
- [22] Philipp Rohlfshagen and Simon M Lucas. Ms Pac-Man versus Ghost Team CEC 2011 Competition. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 70–77. IEEE, 2011.
- [23] Spyridon Samothrakis, Diego Perez, Philipp Rohlfshagen, and Simon Lucas. Predicting dominance rankings for score-based games. 2014.
- [24] Spyridon Samothrakis, David Robles, and Simon Lucas. Fast Approximate Max-n Monte Carlo Tree Search for Ms Pac-Man. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(2):142–154, 2011.
- [25] Jacob Schrum and Risto Miikkulainen. Discovering multimodal behavior in ms. pac-man through evolution of modular neural networks.
- [26] David Silver and Joel Veness. Monte-Carlo planning in large POMDPs. In *Advances in neural information processing systems*, pages 2164–2172, 2010.
- [27] Nathan Wirth and Marcus Gallagher. An Influence Map Model for Playing Ms. Pac-Man. In *Computational Intelligence and Games, 2008. CIG’08. IEEE Symposium On*, pages 228–233. IEEE, 2008.
- [28] Markus Wittkamp, Luigi Barone, and Philip Hingston. Using NEAT for continuous adaptation and teamwork formation in Pacman. 2008.



# Beyond Computational Intelligence to Computational Creativity in Games

Dan Ventura  
Computer Science Department  
Brigham Young University  
Provo, UT 84602  
Email: ventura@cs.byu.edu

**Abstract**—This paper argues that computational creativity is the logical next step in the evolution of game design; briefly overviews what is meant by computational creativity and suggests some ways in which it could augment contemporary games; explores some initial ideas for its incorporation into the future of gaming and game design; and argues for increased cross-pollination and collaboration between the computational intelligence and games research community and the computational creativity research community.

## I. INTRODUCTION

Computational intelligence has become the (near) future of game design and development, and it is interesting to ask then what might be the next logical step for the longer-term. Here, it is argued that step is *computational creativity* (CC).

The computational intelligence and games (CiG) community has facilitated the wide-spread adoption of procedural content generation (PCG) and artificial intelligence (AI) into game development and experience. Procedural generation of content for games is increasingly allowing designers to focus on higher-level concerns, while automatic generators produce lower-level content such as textures, landscapes, buildings, layouts, music, simple dialogues, etc. [1], [2]. Artificial intelligence is being used most commonly in the form of non-player characters and other types of agents, but it has been used in many other ways as well, including dynamic difficulty balancing [3], [4], player experience modeling [5], [6], and datamining of user behavior [7], [8].

Recently, it has been proposed that the games domain may be the “killer app” for the nascent field of computational creativity [9], and the arguments supporting this position are rather compelling, enough so to suggest a reciprocal kind of relationship—if games are the “killer app” for computational creativity, then perhaps computational creativity is the future of games. The beginnings of this idea have, in fact, been suggested elsewhere [10], and the purpose here is to argue for this evolution in the extreme. As the games community has embraced computational intelligence as an integrated augmentation device for the designer’s intelligence, this suggests that the next possible step is to do the same thing for the designer’s creativity. By embracing computational creativity, game developers can build computational collaborators that take real creative responsibility as a member of a team, and,

in the extreme, may build fully autonomous content- or game-creation systems that are legitimate designers themselves.

The integration of CC philosophies and techniques into the game design process is a natural next step in the computational intelligence and games evolution, allowing for greater personalization of the gaming experience, novel and adaptable system behaviors, and the transitioning of intelligence into higher-level components of game creation. It can produce a more immersive game-play experience (e.g. more robust and believable NPC intelligence, personalized and adaptive content), allow for the automatic creation of additional types of game content (and eventually even complete complex games), and make game creation more scalable by relieving the human-creator bottleneck.

## II. WHAT IS COMPUTATIONAL CREATIVITY?

Because the concept of creativity is very difficult, if not impossible, to formalize and is likely, in fact, an *essentially contested concept* [11], the field as a whole has agreed not to argue (any longer) about what exactly it is but instead focuses on building computationally creative systems. While this may seem disingenuous, it should be noted that the broader field of AI has dealt with a similar crisis of agreement cf. the meaning of intelligence, and yet has still made remarkable discoveries and inventions that everyone agrees are useful progress and likely even constitute “intelligence” in some sense. It is in this same hopeful spirit that the CC field has, in the last few years adopted a circular description suggested by Colton and Wiggins [12] as its stand-in for a proper definition:

[Computational creativity is] *the philosophy, science and engineering of computational systems which, by taking on particular responsibilities, exhibit behaviours that unbiased observers would deem to be creative.*

So, the goal is to build systems that do things that are creative. This is pretty wide open, as almost any domain of endeavor can be argued to require creativity to meet at least some of its challenges, from the artistic to the scientific to the mundane. Indeed, creativity is certainly an aspect of intelligence and is likely at least part of the reason that human intelligence is so robust. While, to date, computationally intelligent systems have demonstrated and continue to demonstrate some remarkable abilities, they are, at the same time, distinctly fragile, in

the sense that their domain of expertise is almost vanishingly narrow. In contrast, humans demonstrate an impressively broad general intelligence, and at least one thing that differentiates human and computational intelligence seems to be creativity.

Perhaps the defining difference between computational intelligence and computational creativity is in the types of problems to which they are applied; or possibly the difference can be captured in their approaches to solving these problems. In the case of computational intelligence, problems are usually framed as some type of optimization—the system is trying to attain the highest accuracy, or maximize area under the curve, or minimize losses, or maximize discounted future reward, or win. Actually, perhaps the concept of winning could be argued to be not quite optimization, in the sense that there may be more than one winning strategy, and in fact, given the case of an opponent strategy, the winning strategy may be dynamic, but fixing opponent strategy, it is still largely a type of optimization, or, possibly it is a form of satisficing, which is simply a weaker form of optimization—it is just “optimizing” for “good enough”. In contrast, the kinds of problems that computational creativity addresses are of an entirely different class. There is no such thing as a best song, or best theorem or best design. One cannot maximize a piece of visual art or a recipe or a poem. There are many interesting songs, theorems, designs, paintings, recipes and poems, and the goal is to find one or more of these. What constitutes a “solution” for these types of “problems” might also be dynamic based on environment, but even if such environmental factors can theoretically be held constant, this is still nothing like an optimization problem, at least in the traditional sense.

Instead, success is qualified (and sometimes possibly quantified) with notions of *novelty* and *utility*. Whatever the domain in question, a computationally creative system should produce artifacts<sup>1</sup> that are novel and useful. Here, these qualities will be defined as follows:

**novelty:** the quality of being new, original or unusual; this is relative to the population of artifacts in the domain in question and can apply in the personal or historical sense.

**value:** the importance, worth or usefulness of something; this would typically be ascribed by practitioners of the domain in question.

Note that these qualities are most commonly addressed with respect to product; however, creativity often refers at least as much to process as it does to product, and, in particular, it has been argued that the (perception of the) process by which a computational system produces artifacts is likely at least as important as the artifacts produced [13].

To summarize, computational creativity deals with constructing artificial agents that produce artifacts that are judged novel and useful by those that understand the domain in which the agent works.

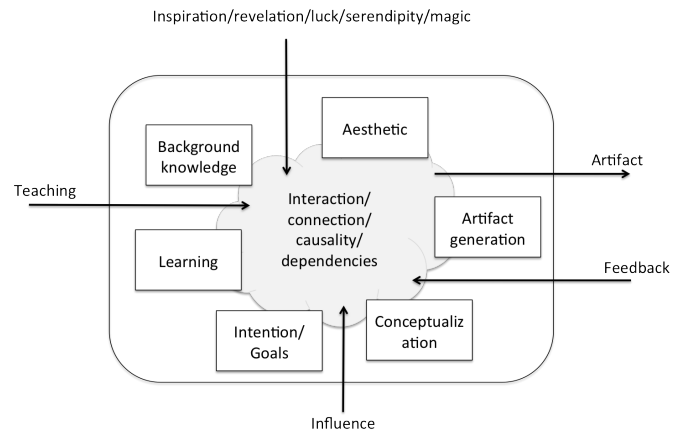


Fig. 1. An abstraction of a creative agent. The component internal mechanisms are meant to be likely necessary though possibly not sufficient. No attempt here is made to accurately visualize the dependencies and communication between these mechanisms. The agent communicates with the environment in several ways, represented by labeled arrows entering or leaving the agent (borrowed from [14]).

#### A. An Abstraction of the Creative Agent

Figure 1 offers a gross visualization of an abstract, archetypal CC agent. Such an agent is composed of multiple internal mechanisms/processes, some of which include *background knowledge*, an ability to *learn*, *intentionality*, an ability to *conceptualize*, a sense of *aesthetic* and some method of *generating* artifacts.

1) *Background knowledge*: can be encoded in a variety of forms, including rules, associations, semantic networks, iconic representations, prototypical artifacts, a model or set of models (e.g. built from training examples), a database, statistical information, etc.

2) *Learning*: typically happens by some appropriate machine learning technique such as training (deep or recurrent) neural networks, building a (variable order) Markov model or other form of graphical model, inducing decision trees, forests or other forms of rules, nearest neighbor methods, etc.

3) *Intentionality*: can be effected by giving the system goals such as the communicating of a concept, innovation (e.g., with respect to its background knowledge), utility (e.g., with respect to some objective function), the accomplishment of some task or the realization of some state, etc. More advanced systems will typically have fewer, more abstract goals and will generate their own concrete subgoals.

4) *An aesthetic*: can be encoded explicitly (e.g. as a fitness function or a probability distribution over possible outcomes) or implicitly (e.g. in the generative mechanism, etc.), should likely be correlated with both background knowledge and what is learned and should be dynamic in the sense of changing over time in response to changes in either the internal or external environment.

<sup>1</sup>Note that the use of artifact here is abstract, and that, in particular, the artifact produced might, in fact, itself be a process.

5) *Conceptualization*: includes the ability to represent, manipulate and invent concepts in the domain. What makes a reasonable representation is certainly domain specific, with examples including things like rule sets, statistical models, vectors and mixed modalities. Conceptual representations should be grounded in the domain, and can be quite specific, with a simple mapping to an element of the domain (e.g., like the relationship between the genotype and phenotype in evolutionary computation), or they can be quite abstract, mapping to large subsets of the domain.

6) *Generation*: of artifacts can be accomplished with a random process, an evolutionary mechanism, a grammar, a generative model, etc. It is important to note that the use of the term *artifact* here is meant in a very abstract sense, so that, in particular, an artifact could be a concrete product, such as a game level, a weapon, an NPC monster, etc.; a more complex product such as a puzzle, a game mechanic, a narrative, etc.; an abstract product like a strategy to be used by an NPC monster, a quest to be assigned to a player, a level theme, etc., or even a process, such as a method for generating a game level, an aesthetic by which to evaluate a potential weapon design, or a new goal to drive the system's behavior.

Of course, in reality there are rarely crisp boundaries that clearly differentiate these mechanisms/components, and some of these are already very developed (in a computational sense) in the computational intelligence and games community, but some are still almost exclusively the purview of the human designer. It is in the consideration, development and incorporation of these others by which the games community might take a quantum leap forward.

In addition to these internal mechanisms, because the agent exists in an environment, it interacts with the environment in multiple ways, including being *taught*, *presenting* artifacts, being *inspired*, receiving *feedback* and other *influences*.

7) *Teaching*: often comes in the form of a supervisory signal derived from either structured or unstructured data resources, many of which are now freely available on the web; additional sources of supervisory signal can be obtained through human-labeled examples, human or computational reaction to queries, etc.

8) *Presentation*: of created artifacts is typically constrained by domain (e.g., visual images, written or performed music, a set of rules defining a strategy, a theorem and accompanying proof, a recipe, a process, a game); if the output is intended for human consumption, it obviously must be presented in a form appreciable by humans. In addition, the presented artifact can (and often should) be accompanied with some form of framing information (e.g., a title, a backstory, instructions, context, etc.)

9) *Inspiration*: is an ill-defined concept that encompasses environmental stimuli that directly or indirectly affect the agent's creative process. The most obvious source of inspiration would typically be example artifacts produced by others (e.g., for a musician, another musician's composition); however, inspiration may be found in artifacts from other domains, outside advice, etc.

10) *Feedback*: may take the form of immediate and direct (positive or negative) reinforcement with respect to a presented artifact, or, it may take less direct forms, such as a collection of survey responses, sale/resale value, citation or adoption data and so on.

11) *Other influences*: might include things like feedback from the environment about others' work (when publicly available), the behaviors, opinions, preferences and aesthetics of other creators or consumers, current events, etc.

This abstract notion of a creative agent has been instantiated in many different forms, and a variety of systems of varying degrees of sophistication and efficacy have been built by the CC community for creating artifacts in a broad range of domains, including culinary recipes [15], [16], language constructs such as metaphor [17] and neologism [18], visual art [19], [20], poetry [21], [22], humor [23], [24], advertising and slogans [25], [26], narrative and story telling [27], [28], mathematics [29] and music [30], [31].

At least some of these kinds of systems, and possibly any of them, could be incorporated directly in games, or the techniques they use could be repurposed for use in creating games or components of games. However, in the long run, the most useful takeaway for the CiG community is the set of meta-level ideas driving this research independent of a particular domain. Subscribing to these can have a lasting impact in extending the autonomy/responsibility that can be given to game systems.

The meta-level ideas referred to here are those discussed above in the treatment of an abstract creative agent and should be considered necessary, though possibly not sufficient, for a (sophisticated) CC system. Indeed, because creativity is here assumed to be an essentially contested concept, it is likely impossible to establish a sufficiency condition for computational creativity. Instead, analogous to scientific theories, a system likely may only conclusively be discredited as *not* creative, and this list is a starting point for avoiding the obvious ways this might happen (cf. some of the arguments in [13]). The avoidance of such a discrediting may seem like a dubious goal to which to aspire; however, a system that cannot easily be argued to be uncreative is likely a quite impressive one.

#### B. How is this different than PCG?

It is possible to consider this proposal of computational creativity as just procedural content generation on steroids, and in one sense, that may not be completely incorrect.

However, the idea of content should be expanded beyond the low-level (though certainly important and by no means trivial) components currently being tackled (e.g., maps, levels, skins and other visuals, some music, simple dialogue, etc.) to include much higher-level constructs such as complete quests, complex mechanics, goals and objectives, governing rules, ludic considerations, full narratives and eventually complete games (cf. Smith's position on the future of PCG [32]).

Perhaps the most significant improvement over traditional PCG approaches is the consideration of agent *intentionality*—CC research targets the building of systems that demonstrate

deliberation or purposiveness in their creating. Accomplishing this is nontrivial, to be sure, and is a topic of ongoing research and debate. One way in which this might be approached is by building systems that share a perceptual grounding with users. Such shared grounding facilitates successful communication of intention between the system and those with which it interacts by providing a common medium for motivating and interpreting system actions. Games are an ideal domain for this kind of research, because it is possible to forge that shared grounding through the game experience (which may even facilitate types of grounding and types of intentionality not possible in the real world).

Of course, CiG research is already beginning to explore some of these possibilities, and that's why this discussion is timely—the computational creativity perspective can lend a transformative momentum to this trend.

### III. IMAGINING THE POSSIBILITIES

Several classic and recent games provide good examples of how CC ideas might be incorporated; here, the games are categorized as research or commercial, with the allowance that others may see the dichotomy somewhat differently.

#### A. Research Games

These games have typically been used as research platforms over many years, are well-understood and are relatively simple, in the sense that one might be able to envision an automated system for inventing something like them in its entirety.

For example, consider the class of platform- or level-based puzzle games. *Super Mario Bros.*<sup>2</sup> is certainly the most studied of these, likely because of both its original popularity and because of its conservative nature—it is easy to generate playable levels. Examples like *Ms. Pac-Man*<sup>3</sup> and *Spelunky*<sup>4</sup> complexify the genre by adding ghosts and bombs, respectively, that make both level design and play more challenging. Is it possible to abstract a complete description of such games in such a way that comparable new, cohesive and interesting games of the genre could be created automatically?

Or consider card-based games such as *Hearthstone*<sup>5</sup> and *Lords of War*<sup>6</sup>. Because these games lack mechanics, it is perhaps even simpler to imagine abstracting the genre and building a system that creates new complete games. The challenge is the invention of diverse card packs and their coherent incorporation into a set of gameplay and ludic rules.

Racing games like *TORCS*<sup>7</sup> provide another relatively simple class of games to consider for abstraction. The mechanics are well-defined and immutable (though, an interesting variation would allow mutation of the mechanics in coherent ways). Vehicle types, tracks, obstacles, race conditions, skins, and soundtracks could all be the subject of CC intervention, and

while the basic ludic principle is to finish first, perhaps even this could be tampered with in interesting ways.

First person shooters like the very popular *Unreal Tournament*<sup>8</sup> and strategy games like *Starcraft*<sup>9</sup> and their variants and expansions introduce multi-player, tactical decision making, and real-time considerations as well as scale and additional complexity issues. Perhaps the most obvious focus here is the development of sophisticated and believable AI NPCs, but not using traditional tricks that allow NPCs game-knowledge not available to normal players (cf. the comment below about *The Flame in the Flood*<sup>10</sup>).

Games like *Galactic Arms Race*<sup>11</sup> and *Neuro-Evolving Robotic Operatives (NERO)*<sup>12</sup> are interesting in that they might be described as exploring some of the themes taken up here in a non-cognitive way—by using neuroevolution to procedurally generate content that dynamically responds to the game environment.

Other types of research games exist, of course, and the interactive fiction game *Faade*<sup>13</sup> deserves a special mention as a very different kind of “game”, that is in some ways much more complex than those mentioned above and is also in some ways a true pre-cursor to the idea of CC in games. (A more recent version of this idea, *Versu*<sup>14</sup>, introduces some groundbreaking work on social modeling and how it affects narrative and in particular the progression of a group conversation.) Imagine the core engine for *Faade* coupled with the ability to potentially invent any human-drama-based interactive fiction, based on current events, online novels, movies, etc.

#### B. Commercial Games (AAA and Indie)

Commercial games have a high “cool” factor, but their implementations are usually opaque, and therefore it is difficult to evaluate their process—there is always the possibility of smoke and mirrors, but if it works, it works—their goals are different than a research community's (though hopefully they are informed by such goals). Given that, what follows are some potential CC contributions to such games.

*The Flame in the Flood* (a river journey) and *No Man's Sky*<sup>15</sup> (a mind-bogglingly large universe) and, to a lesser extent *Secret Habitat*<sup>16</sup> (a series of art galleries), offer impressive PCG of the main game content. However, in the first case, NPC AIs are supported by dynamic markup of generated terrain, which would be considered “cheating” in the CC sense, and in the latter two, there are no NPCs. CC techniques might facilitate the introduction of believable NPCs that inhabit the PCG worlds and thereby provide a cohesive plot (of sorts).

At the other end of the spectrum are games with very strong NPC AIs, with examples including *Incognita* from *Invisible*,

<sup>2</sup>Nintendo, 1985

<sup>3</sup>Bally/Midway Manufacturing, 1982

<sup>4</sup>Mossmouth, 2008

<sup>5</sup>Blizzard Entertainment, 2014

<sup>6</sup>Black Box Games Publishing, 2012

<sup>7</sup>Eric Espi   and Christophe Guionneau, 1997

<sup>8</sup>Epic Games and Digital Extremes, 1999

<sup>9</sup>Blizzard Entertainment, 1998

<sup>10</sup>The Molasses Flood, 2016

<sup>11</sup>Evolutionary Games, 2010

<sup>12</sup>Neural Networks Group, CS Dept., UT Austin, 2005

<sup>13</sup>Michael Mateas and Andrew Stern, 2005

<sup>14</sup>Richard Evans and Emily Short

<sup>15</sup>Hello Games, 2016

<sup>16</sup>Strangethink, 2014

*Inc.*<sup>17</sup>, the Xenomorph from *Alien: Isolation*<sup>18</sup> and Elizabeth from *Bioshock: Infinite*<sup>19</sup>. These AIs might be made even more believable with a focus on intentional novelty and utility.

Adventure games such as *Assassin's Creed*<sup>20</sup>, *Sunset Overdrive*<sup>21</sup> and *Dying Light*<sup>22</sup> are hailed for their visuals, their combat systems, their mechanics, etc. while at the same time being criticized for their relatively weak stories. Of course all games do not need to be all things to all people, but here is another example of a subcomponent of the game that might be co-opted by a CC system to positive effect.

Other games of interest here might include *Total War: Rome II*<sup>23</sup> for its impressive incorporation of Monte Carlo tree search (discussed in the next section) and *Cities: Skylines*<sup>24</sup> a building/planning simulation game that might be made more game-like if it could pose interesting problems to be solved.

#### IV. COMPUTATIONAL CREATIVITY IN GAMES

As mentioned above, the field of computational creativity is already beginning to produce working systems that themselves produce artifacts that could be used as components of games. However, the bigger vision is, of course, the building of a system that can contribute significantly to the creation of complete games (or, in the extreme, autonomously create complete games). A good deal of recent work has begun to explore various approaches to this for simple games [33]–[38], with the most explicit example probably being Cook's ANGELINA system [39], [40], so this is of course not a novel idea; however, the widespread adoption of CC ideas will help move this effort forward significantly.

The first necessity is some way to describe a game in the abstract, and many attempts at this exist, including the classic *Zillions of Games* [41], the game description language [42] used in the AAAI general game playing competition, the game description language used in the Ludi system [33], the recently developed puzzlescript [43] and a general video game description language [44].

Now, imagine something like these description languages that admit the description of a space of possible games in an abstract, hierarchical manner, as suggested in Figs 2, 3, and 4. Each level of abstraction consists of a set of design choices, and the hierarchical organization allows the exploration of the space by traversal of a tree, as in Figure 5. Making choices at each node in the tree corresponds to design decisions, both general and specific for creating a particular game, and the leaf nodes in the tree correspond to completely specified games, given the representation/description language. One obvious possibility for exploring such a tree would be some variation on Monte Carlo tree search (MCTS), which has become

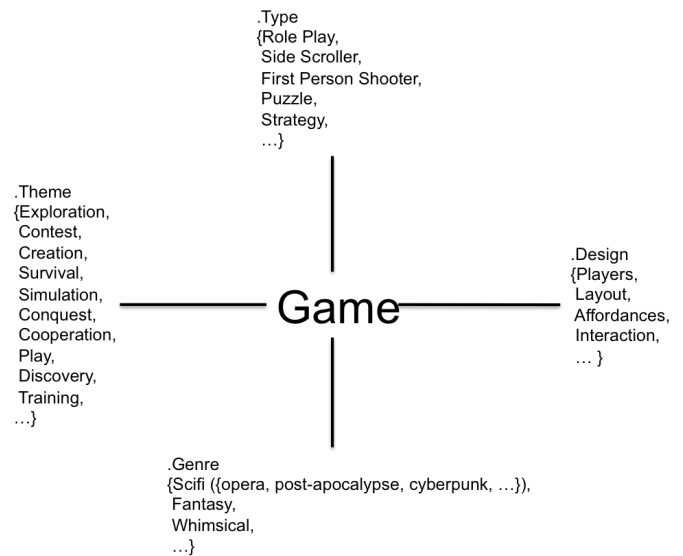


Fig. 2. An abstract (representative, incomplete) model of a *game*. A game has *theme*, *genre*, *type* and *design* elements (among other things), and each of these elements can take different values, each of which can be expanded with further detail at a lower level of the abstraction (see Fig. 3).

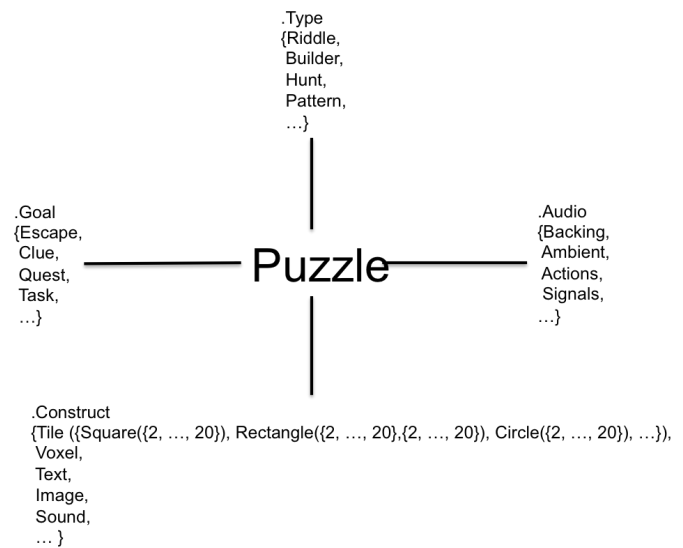


Fig. 3. An abstract (partial) model of a *puzzle-game*. A second level of abstraction specializes the game-type puzzle, which itself has associated elements *goal*, *type*, *audio* and *construct*. Again, each of these elements can be expanded with further detail at a lower level of the abstraction (see Fig. 4).

widely adopted in the CiG community [45]. (In fact, something like this has been done very recently on a limited scale, for the creation of platformer levels [46].)

In traditional MCTS, nodes in the tree represent game states, and leaf nodes represent completed games, with an outcome (win or lose, or possibly draw). A path from a node to a leaf represents a sequence of moves, each resulting in a new game state further down the tree, until the leaf is reached and the game is completed. Following one of these paths is called a *playout*, and the search involves making many playouts and

<sup>17</sup>Klei Entertainment, 2015

<sup>18</sup>Creative Assembly, 2014

<sup>19</sup>Irrational Games, 2013

<sup>20</sup>Ubisoft, 2007

<sup>21</sup>Insomniac Games, 2014

<sup>22</sup>Techland, 2015

<sup>23</sup>Creative Assembly, 2013

<sup>24</sup>Colossal Order, 2015

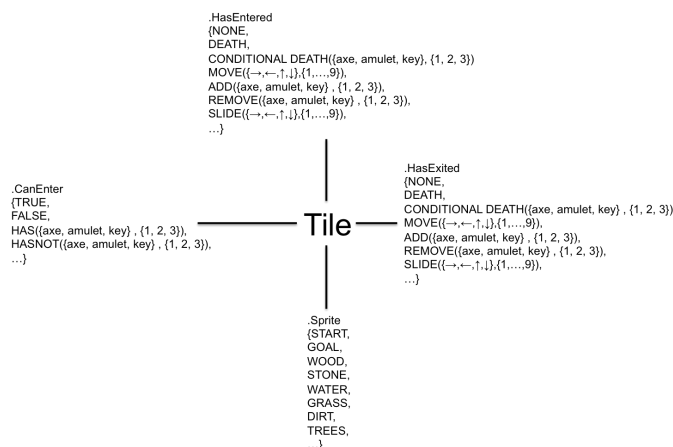


Fig. 4. An abstract model of a *tile* for a tile-based puzzle-game. This third level of the abstraction provides concrete detail that can be implemented—a tile has three parameterized functional characteristics: *.CanEnter*, *.HasEntered* and *.HasExited* and an associated *sprite*, which can take any of several visual values (of course in general, the levels of abstraction could continue).

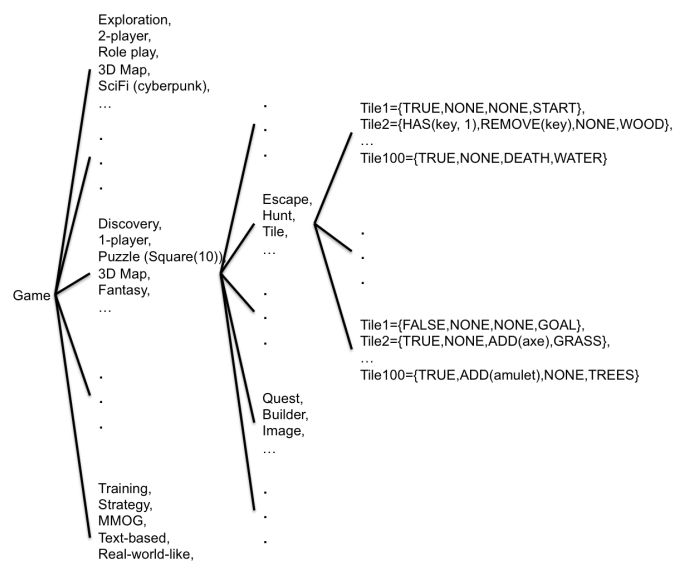


Fig. 5. Search tree for exploring the hierarchical space of possible games (partially) described by the abstractions of Figs 2, 3, and 4. Each node in the tree represents a partial game design, and each branch represents a particular design decision that further specifies the game. Leaf nodes represent fully specified games.

collecting statistics on their results (wins/losses/draws) and backpropagating those statistics up the tree. Eventually, based on these collected statistics, a move is actually made, and the game progresses. In this search, the objective is to win the game, and the search explores different paths down the tree in an effort to find a “good” move to make, which increases the chances of meeting the objective. Each time the search reaches a leaf, the result is known and the relative quality of the moves in the path can be updated.

In searching the proposed game creation tree of Fig. 5, the objective is to create a “good” game, and the search explores

different paths down the tree in an effort to find one of these “good” games. Each node in the path is a design decision, with the idea being to make “good” decisions that lead to the creation of a “good” game, in the same way that making good moves leads to winning a game. In other words, the game creation process can be thought of as a meta-game of sorts; however, it is not a competitive game—there is no explicit opponent nor explicit concept of winning—another example of the difference between traditional AI problems and CC problems. Since there is no objective concept of winning, how can the leaf nodes of this tree be evaluated? How can it be determined when one of them represents a “good” game (or piece of a game, in the case of creating a soundtrack or mechanics or visuals, etc.)?

This, it turns out, is a really difficult question, that can be posed in a variety of ways. For example, consider the idea of using an inductive logic program (ILP) to represent a quest and the (pre- and post-) conditions that apply to various stages of that quest. It is not perhaps that difficult to think about how one might do this (though for a complex quest, this may not be trivial, either), and this might, in fact, be an interesting way to approach the problem of realizing a game quest. Having taken that step, it might not be (too) difficult to then take the next step—considering how to build a system that could generate an ILP to represent a quest. However, what determines if the generated quest is a good one? Is there some way to formalize an aesthetic for (ILP-encoded) quests, so that the system has a method for evaluating its own output? This is a difficult question, of course, and the discovery and implementation of useful aesthetics for various domains is one of the key points of study in the field. However, even this is not yet satisfactory. For, given a formalized aesthetic for “good” quests, the system now can (in principle) create such quests, but it will not create other quests that may be considered “good” under some other defensible aesthetic. In other words, what is really wanted is a system that can (also) create an aesthetic, and for that, what is required is an aesthetic for aesthetics (about quests in this particular example).

This meta-level reasoning has its finger prints all over computational creativity, and once recognized, it immediately begs the question, “what about an aesthetic for aesthetics?” Ad infinitum. This is a fair question, but for now a satisfactory accounting for one additional level of meta will constitute serious progress. It is also worth mentioning here that it is not clear that human creators are capable of this type of higher-order meta-evaluation either, so if artificial systems are limited to “only” one meta-level of evaluation, they may be still in good company.

Now, assume the existence of a system for exploring the (tree-structured) space of possible games, that the system knows what constitutes a “good” game, and even that it knows how to “change its mind” about what makes a game “good” in a defensible way. There is *still* work to do, in the sense that the search space has been given to the system in the first place and is immutable from the system’s point of view. What if the system could invent new branches for the tree? This could be



something as simple as adding an additional item to associate with the *Tile.CanEnter.HAS* property of Fig. 4; or as complex as adding a new *Game.Type* in Fig. 2; or even inventing an additional element, *Game.NewElement*, analogous to the existing elements: *Game.Theme*, *Game.Type*, *Game.Design*, and *Game.Genre*. This is another form of meta-creativity in which the space to be explored is the space of all possible trees that define spaces of game representations. But then, what kind of aesthetic would guide *that* search?

Of course, MCTS may not be the right approach to this general problem of exploring the space of possible games. The search tree envisioned here is possibly relatively shallow, and possibly very broad (even likely infinitely so). Is MCTS the best approach for this shape of tree, let alone this kind of search problem? There are certainly other traditional forms of search that could also be adapted as a possible mechanism for exploring this space, and it is possible that the shape of the tree might be changed significantly by changing the abstraction. Perhaps a tree isn't even the appropriate structure for describing this space. Perhaps it is even possible to search the space of abstractions for good ones for searching for games. Many of these considerations are related to interesting foundational work by Wiggins [47]; however, this is likely beyond the realm of (immediate) interest for the CiG community.

One additional suggestion that deserves further exploration is the idea of building games in which computational creativity is the main feature of the game. It is not yet clear what this might entail, but a similarly intriguing idea has been suggested for games featuring AI [48], and as argued there, it seems likely that taking full advantage of CC as the main event will require re-thinking at least some accepted ideas about games and may open the way for entirely new types of game.

#### A. Evaluation

As with any creative endeavor, it is not sufficient that the creator believe that the result is novel and useful, though this is certainly necessary; other creators or consumers or other “gatekeepers” of the field must also attribute these values to the result. This sort of external feedback can be measured in any number of ways, and certainly gross measures such as sales ranking, hours played and other adoption/popularity metrics represent something of a bottom line when it comes to games. However, there is another sense of external measurement that is also critical to the advancement of computational creativity as a field and is somewhat more difficult to assess—the “creativity” of the system. This is a difficult question, made the more so by the lack of a concrete definition for creativity. Still, some progress has been made and varying proposals suggest ways of dealing with the problem, including suggesting metrics for quantifying various qualities of system output [49]; an abstract ontology of behaviors potentially demonstrable by a system [50]; a proposal for a standardized evaluation methodology that uses case-specific requirements-based testing [51]; and a spectrum of prototypical abstract landmark algorithms that characterize varying levels of system intentionality in producing novelty and utility [52].

## V. CONCLUSION

This paper argues that the logical next step for computational intelligence and games is the incorporation of computational creativity in games. It gives a necessarily brief overview of the field of computational creativity, imagines some initial uses for it in contemporary games, and explores the beginnings of a few ideas for its incorporation into the next generation of games and beyond. Many more questions have been raised than have been answered, with the goal being to arouse interest in the CiG community reciprocal to that which has begun to grow in the CC field. For those whose interest is piqued, a good resource for all things CC, and in particular an extensive and growing bibliography can be found at [www.computationalcreativity.net](http://www.computationalcreativity.net).

Computational creativity is itself basically domain agnostic. However, since it is very difficult to make an effective study in the abstract, it is typical to settle for being agnostic in the statistical sense of “averaging” over many domains. This provides interesting opportunities for collaboration between CC researchers and researchers in a particular domain to which CC may be applied, and that, in turn, can strengthen research agendas in both fields. Here the call by Liapis, et al. for such a collaboration between the CiG and CC communities is reciprocated as being perhaps the most natural of possible collusions. Though the extreme possibility of a fully autonomous system that creates complete games has been considered, in reality most games are now so complex that they are built by (often large) teams of creative individuals, and so the more likely positive outcome would be a system or systems that can participate in that collaborative process as true co-creative members of such a team.

## ACKNOWLEDGMENT

Thanks to Jim Whitehead, Michael Mateas, Michael Cook, Antonios Liapis and Julian Togelius for many useful suggestions that helped guide the development of this manuscript.

## REFERENCES

- [1] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, “Search-based procedural content generation: A taxonomy and survey,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 172–186, 2011.
- [2] M. Hendriks, S. Meijer, J. V. D. Velden, and A. Iosup, “Procedural content generation for games: A survey,” *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 9, no. 1, p. 1, 2013.
- [3] R. Hunicke and V. Chapman, “AI for dynamic difficulty adjustment in games,” in *AAAI Workshop on Challenges in Game Artificial Intelligence*, 2004.
- [4] G. Chaneil, C. Rebetez, M. Betrancourt, and T. Pun, “Emotion assessment from physiological signals for adaptation of games difficulty,” *IEEE Transactions on Systems Man and Cybernetics, Part A*, vol. 41, no. 6, pp. 1052 – 1063, 2011.
- [5] A. Drachen, “Crafting user experience via game metrics analysis,” presented at NORDICHI 2008, Lund, Sweden, 2008.
- [6] G. N. Yannakakis and J. Togelius, “Experience-driven procedural content generation,” *IEEE Transactions on Affective Computing*, vol. 2, no. 3, pp. 147–161, 2011.
- [7] C. Bauckhage, A. Drachen, and R. Sifa, “Clustering game behavior data,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 7, no. 3, pp. 266–278, 2015.

- [8] N. Ducheneaut, N. Yee, E. Nickell, and R. J. Moore, "Alone together?: Exploring the social dynamics of massively multiplayer online games," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2006, pp. 407–416.
- [9] A. Liapis, G. N. Yannakakis, and J. Togelius, "Computational game creativity," in *Proceedings of the 5th International Conference on Computational Creativity*, 2014, pp. 46–53.
- [10] K. Compton and M. Mateas, "Casual creators," in *Proceedings of the 6th International Conference on Computational Creativity*, 2015, pp. 228–235.
- [11] W. B. Gallie, "Essentially contested concepts," *Proceedings of the Aristotelian Society*, vol. 56, pp. 167–198, 1956.
- [12] S. Colton and G. Wiggins, "Computational creativity: The final frontier?" in *Proceedings of the 20th European Conference on Artificial Intelligence*, 2012, pp. 21–26.
- [13] S. Colton, "Creativity versus the perception of creativity in computational systems," *Creative Intelligent Systems: Papers from the AAAI Spring Symposium*, pp. 14–20, 2008.
- [14] D. Ventura, "The computational creativity complex," in *Computational Creativity Research: Towards Creative Machines*, T. R. Besold, M. Schorlemmer, and A. Smaill, Eds. Atlantis Press, 2015, pp. 65–92.
- [15] R. Morris, S. Burton, P. Bodily, and D. Ventura, "Soup over bean of pure joy: Culinary ruminations of an artificial chef," in *Proceedings of the 3rd International Conference on Computational Creativity*, 2012, pp. 119–125.
- [16] L. Varshney, F. Pinel, K. Varshney, A. Schorgendorfer, and Y.-M. Chee, "Cognition as a part of computational creativity," in *Proceedings of the 12th IEEE International Conference on Cognitive Informatics and Cognitive Computing*, 2013, pp. 36–43.
- [17] T. Veale and Y. Hao, "Comprehending and generating apt metaphors: A web-driven, case-based approach to figurative language," *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, vol. 2, 2007.
- [18] M. R. Smith, R. S. Hintze, and D. Ventura, "Nehovah: A neologism creator nomen ipsum," in *Proceedings of the 5th International Conference on Computational Creativity*, 2014, pp. 173–181.
- [19] D. Heath, D. Norton, and D. Ventura, "Conveying semantics through visual metaphor," *ACM Transactions on Intelligent Systems and Technology*, vol. 5, no. 2, p. article 31, 2014.
- [20] S. Colton, "The Painting Fool: Stories from building an automated painter," in *Computers and Creativity*, J. McCormack and M. d'Inverno, Eds. Springer-Verlag, 2011, pp. 3–38.
- [21] J. M. Toivanen, H. Toivonen, A. Valitutti, and O. Gross, "Corpus-based generation of content and form in poetry," in *Proceedings of the 3rd International Conference on Computational Creativity*, 2012, pp. 175–179.
- [22] H. G. Oliveira, "PoeTryMe: a versatile platform for poetry generation," in *Proceedings of the ECAI 2012 Workshop on Computational Creativity, Concept Invention, and General Intelligence*, 2012.
- [23] O. Stock and C. Strapparava, "HAHAcronym: Humorous agents for humorous acronyms," *Humor - International Journal of Humor Research*, vol. 16, no. 3, pp. 297–314, 2003.
- [24] K. Binsted and G. Ritchie, "A symbolic description of punning riddles and its computer implementation," in *Proceedings of the Association for the Advancement of Artificial Intelligence*, 1994, pp. 633–638.
- [25] C. Strapparava, A. Valitutti, and O. Stock, "Automatizing two creative functions for advertising," in *Proceedings of 4th International Joint Workshop on Computational Creativity*, 2007, pp. 99–105.
- [26] G. Özalp, D. Pighin, and C. Strapparava, "BRAINSUP: Brainstorming support for creative sentence generation," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, 2013, pp. 1446–1455.
- [27] M. O. Riedl and R. M. Young, "Narrative planning: Balancing plot and character," *Journal of Artificial Intelligence Research*, vol. 39, no. 1, pp. 217–268, 2010.
- [28] R. Pérez y Pérez and M. Sharples, "Three computer-based models of storytelling: BRUTUS, MINTREL and MEXICA," *Knowledge-Based Systems*, vol. 17, no. 1, pp. 15–29, 2004.
- [29] S. Colton, A. Bundy, and T. Walsh, "HR: Automatic concept formation in pure mathematics," in *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, 1999, pp. 786–791.
- [30] B. Houge, "Cell-based music organization in Tom Clancy's EndWar," in *Proceedings of the 1st International Workshop on Musical Metacreation*, 2012.
- [31] F. Pachet and P. Roy, "Non-conformant harmonization: the real book in the style of Take 6," in *Proceedings of the 5th International Conference on Computational Creativity*, 2014, pp. 100–107.
- [32] G. Smith, "The future of procedural content generation," in *Experimental Artificial Intelligence in Games: Papers from the AIIDE Workshop*, 2014, pp. 53–57.
- [33] C. Browne and F. Mairé, "Evolutionary game design," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 1, pp. 1–16, 2010.
- [34] C.-U. Lim and D. F. Harrell, "An approach to general videogame evaluation and automatic generation using a description language," in *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2014, pp. 1–8.
- [35] T. S. Nielsen, G. A. B. Barros, J. Togelius, and M. J. Nelson, "Towards generating arcade game rules with VGDL," in *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2015, pp. 185–192.
- [36] A. Khalifa and M. Fayek, "Automatic puzzle level generation: A general approach using a description language," in *Computational Creativity and Games Workshop*, 2015.
- [37] D. M. LeBaron, L. A. Mitchell, and D. Ventura, "Intelligent content generation via abstraction, evolution and reinforcement," in *Experimental AI in Games: Papers from the AIIDE 2015 Workshop*, 2015, pp. 36–41.
- [38] J. Gow and J. Corneli, "Towards generating novel games using conceptual blending," in *Experimental AI in Games: Papers from the AIIDE 2015 Workshop*, 2015, pp. 15–21.
- [39] M. Cook, S. Colton, and J. Gow, "The ANGELINA videogame design system, part I," *IEEE Transactions on Computational Intelligence and AI in Games*, 2016, to appear.
- [40] —, "The ANGELINA videogame design system, part II," *IEEE Transactions on Computational Intelligence and AI in Games*, 2016, to appear.
- [41] J. Mallett and M. Lefler, "Zillions of games," 1998. [Online]. Available: <http://www.zillions-of-games.com>
- [42] N. Love, T. Hinrichs, D. Haley, E. Schkufza, and M. Genesereth, "General game playing: Game description language specification," Stanford, LG-2006-01, 2006.
- [43] S. Lavelle, "Puzzlescript." [Online]. Available: <http://www.puzzlescript.net/>
- [44] T. Schaul, "A video game description language for model-based or interactive learning," in *Proceedings of the IEEE Conference on Computational Intelligence in Games*, 2013, pp. 1–8.
- [45] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte Carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.
- [46] A. Summerville, S. Philip, and M. Mateas, "MCMCTS PCG 4 SMB: Monte Carlo tree search to guide platformer level generation," in *Experimental AI in Games: Papers from the AIIDE 2015 Workshop*, 2015, pp. 68–74.
- [47] G. A. Wiggins, "A preliminary framework for description, analysis and comparison of creative systems," *Knowledge-Based Systems*, vol. 19, pp. 449–458, 2006.
- [48] I. Horswill, "Game design for classical AI," in *Experimental Artificial Intelligence in Games: Papers from the AIIDE Workshop*, 2014, pp. 28–34.
- [49] G. Ritchie, "Some empirical criteria for attributing creativity to a computer program," *Minds and Machines*, vol. 17, pp. 67–99, 2007.
- [50] S. Colton, J. Charnley, and A. Pease, "Computational creativity theory: The FACE and IDEA descriptive models," in *Proceedings of the 2nd International Conference on Computational Creativity*, 2011, pp. 90–95.
- [51] A. Jordanous, "A standardised procedure for evaluating creative systems: Computational creativity evaluation based on what it is to be creative," *Cognitive Computation*, vol. 4, no. 3, pp. 246–279, 2012.
- [52] D. Ventura, "Mere generation: Essential barometer or dated concept?" in *Proceedings of the 7th International Conference on Computational Creativity*, 2016, to appear.

# Enhancements for Real-Time Monte-Carlo Tree Search in General Video Game Playing

Dennis J. N. J. Soemers, Chiara F. Sironi, Torsten Schuster, and Mark H. M. Winands

Department of Data Science and Knowledge Engineering, Maastricht University

d.soemers@gmail.com, t.schuster@student.maastrichtuniversity.nl, {c.sironi,m.winands}@maastrichtuniversity.nl

**Abstract**—General Video Game Playing (GVGP) is a field of Artificial Intelligence where agents play a variety of real-time video games that are unknown in advance. This limits the use of domain-specific heuristics. Monte-Carlo Tree Search (MCTS) is a search technique for game playing that does not rely on domain-specific knowledge. This paper discusses eight enhancements for MCTS in GVGP; Progressive History, N-Gram Selection Technique, Tree Reuse, Breadth-First Tree Initialization, Loss Avoidance, Novelty-Based Pruning, Knowledge-Based Evaluations, and Deterministic Game Detection. Some of these are known from existing literature, and are either extended or introduced in the context of GVGP, and some are novel enhancements for MCTS. Most enhancements are shown to provide statistically significant increases in win percentages when applied individually. When combined, they increase the average win percentage over sixty different games from 31.0% to 48.4% in comparison to a vanilla MCTS implementation, approaching a level that is competitive with the best agents of the GVG-AI competition in 2015.

## I. INTRODUCTION

General Video Game Playing (GVGP) [1] is a field of Artificial Intelligence in games where the goal is to develop agents that are able to play a variety of real-time video games that are unknown in advance. It is closely related to General Game Playing (GGP) [2], which focuses on abstract games instead of video games. The wide variety of games in GGP and GVGP makes it difficult to use domain-specific knowledge, and promotes the use of generally applicable techniques.

There are two main frameworks for GVGP. The first framework is the Arcade Learning Environment (ALE) [3] for developing agents that can play games of the Atari 2600 console. The second framework is GVG-AI [4], which can run any real-time video game described in a Video Game Description Language [5], [6]. This paper focuses on the GVG-AI framework.

The GVG-AI framework is used in the GVG-AI Competition [4], [7]. Past competitions only ran a *Planning Track*, where agents were ranked based on their performance in single-player games. In 2016, it is planned to extend this with a *2/N-Player Track*, a *Learning Track*, and a *Procedural Content Generation Track*. This paper focuses on the Planning Track.

Monte-Carlo Tree Search (MCTS) [8], [9] is a popular technique in GGP [10] because it does not rely on domain-specific knowledge. MCTS has also performed well in GVGP in 2014 [4], which was the first year of the GVG-AI competition, but was less dominant in 2015 [7]. This paper discusses and evaluates eight enhancements for MCTS to improve its

performance in GVGP: *Progressive History*, *N-Gram Selection Technique*, *Tree Reuse*, *Breadth-First Tree Initialization*, *Loss Avoidance*, *Novelty-Based Pruning*, *Knowledge-Based Evaluations* and *Deterministic Game Detection*.

The remainder of the paper is structured as follows. Section II provides background information on the GVG-AI framework and the GVG-AI competition. MCTS is discussed in Section III. In Section IV, the enhancements for MCTS in GVGP are explained. Section V describes the experiments to assess the enhancements. Finally, the paper is concluded in Section VI and ideas for future research are discussed.

## II. GVG-AI FRAMEWORK AND COMPETITION

In the GVG-AI competition [4], [7], agents play a variety of games that are unknown in advance. Agents are given 1 second of processing time at the start of every game, and 40 milliseconds of processing time per *tick*. A tick can be thought of as a turn in an abstract game. Every tick, the agent can choose an action to play, and at the end of the tick the chosen action is played and the game state progresses. Every game has a duration of at most 2000 ticks, after which the game is a loss. Other than that, different games have different termination conditions, which define when the agent wins or loses. Every game in GVG-AI contains at least an *avatar* object, which is the “character” controlled by the agent. Games can also contain many other types of objects. Games in GVG-AI are *fully observable* and can be *nondeterministic*.

Agents can perform searches and attempt to learn which actions are good using the *Forward Model*, consisting of two important functions; *advance* and *copy*. Given a game state  $s_t$ , the *advance*( $a$ ) function can be used to generate a successor state  $s_{t+1}$ , which represents one of the possible states that can be reached by playing an action  $a$ . In deterministic games, there is only one such state  $s_{t+1}$  for every action  $a$ , but in nondeterministic games there can be more than one. The *copy*( $s_t$ ) function creates a copy of  $s_t$ . This function is required when it is desirable to generate multiple possible successors of  $s_t$ , because every call to *advance* modifies the original state, and there is no *undo* function. Because the framework supports a wide variety of different games, it is not optimized as well as any framework dedicated to a specific game would be. This means that the *advance* and *copy* operations tend to be significantly slower than equivalent functions in individual game implementations.

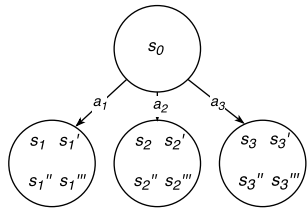


Fig. 1. Example open-loop game tree. Nodes other than the root node can represent multiple possible states in nondeterministic games.

### III. MONTE-CARLO TREE SEARCH

Monte-Carlo Tree Search (MCTS) [8], [9] is a best-first search algorithm that gradually builds up a search tree and uses Monte-Carlo simulations to approximate the value of game states. To handle nondeterministic games with probabilistic models that are not exposed to the agent, an “open-loop” [11] implementation of MCTS is used. In an open-loop approach, the root node represents the current game state ( $s_0$ ), every edge represents an action, and every other node  $n$  represents the set of game states that can be reached by playing the sequence of actions corresponding to the path from the root node to  $n$ , starting from  $s_0$ . See Figure 1 for an example.

MCTS is initialized with only the root node. Next, until some computational budget expires, the algorithm repeatedly executes simulations. Every simulation consists of the following four steps [12], depicted in Figure 2.

In the *Selection* step, a *selection policy* is applied recursively, starting from the root node, until a node is reached that is not yet fully expanded (meaning that it currently has fewer successors than available actions). The selection policy determines which part of the tree built up so far is evaluated in more detail. It should provide a balance between *exploitation* of parts of the search tree that are estimated to have a high value so far, and *exploration* of parts of the tree that have not yet been visited frequently. The most commonly implemented selection policy is *UCB1* [8], [13], which selects the successor  $S_i$  of the current node  $P$  that maximizes Equation 1.  $S_i$  and  $P$  are nodes, which can represent sets of states.

$$UCB1(S_i) = \bar{Q}(S_i) + C \times \sqrt{\frac{\ln(n_P)}{n_i}} \quad (1)$$

$\bar{Q}(S_i) \in [0, 1]$  denotes the normalized average score backpropagated through  $S_i$  so far (as described below),  $C$  is a parameter where higher values lead to more exploration, and  $n_P$  and  $n_i$  denote the visit counts of  $P$  and  $S_i$ , respectively.

In the *Play-out* step, the simulation is continued, starting from the last state encountered in the selection step, using a (semi-)random *play-out policy*. The most straightforward implementation is to randomly draw actions to play from a uniform distribution until a terminal game state is reached. In GVGP, this is typically not feasible, and a maximum play-out depth is used to end play-outs early.

In the *Expansion* step, the tree is expanded by adding one or more nodes. The most common implementation adds one node to the tree per simulation; the node corresponding to the first action played in the play-out step. In this paper, the

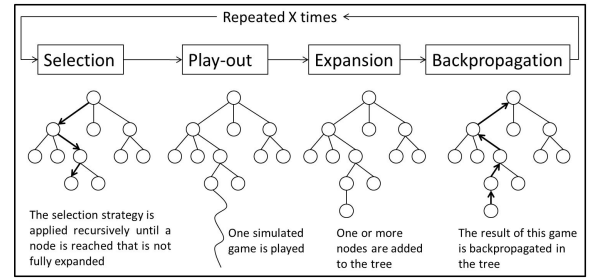


Fig. 2. The four steps of an MCTS simulation. Adapted from [12].

tree is simply expanded by adding the whole play-out to the tree. The number of simulations per tick tends to be low enough in GVG-AI that there is no risk of running out of memory. Therefore, to keep all information gathered, all nodes are stored in memory.

In the *Backpropagation* step, the outcome of the final state of the simulation is backpropagated through the tree. Let  $s_T$  be the final state of the simulation. Next, an evaluation  $X(s_T)$  of the state is added to a sum of scores stored in every node on the path from the root node to the final node of the simulation, and the visit counts of the same nodes are incremented. Because it is not feasible to let all simulations continue until terminal states are reached in GVG-AI, it is necessary to use some evaluation function for non-terminal states. A basic evaluation function that is also used by the sample MCTS controllers included in the GVG-AI framework is given by Equation 2.

$$X(s_T) = \begin{cases} 10^7 + \text{score}(s_T) & \text{if } s_T \text{ is a winning state} \\ -10^7 + \text{score}(s_T) & \text{if } s_T \text{ is a losing state} \\ \text{score}(s_T) & \text{if } s_T \text{ is a non-terminal state} \end{cases} \quad (2)$$

$\text{score}(s_T)$  is the game score value of a state  $s_T$  in GVG-AI. In some games a high game score value can indicate that the agent is playing well, but this is not guaranteed in all games.

Finally, the action leading to the node with the highest average score is played when the computational budget expires.

### IV. MCTS ENHANCEMENTS FOR GVGP

There is a wide variety of existing enhancements for the MCTS algorithm, many of which are described in [14]. This section discusses a number of enhancements that have been evaluated in GVGP; Progressive History, N-Gram Selection Technique, Tree Reuse, Breadth-First Tree Initialization, Loss Avoidance, Novelty-Based Pruning, Knowledge-Based Evaluations, and Deterministic Game Detection. Some are known from existing research, and some are new.

#### A. Progressive History and N-Gram Selection Technique

*Progressive History* (PH) [15] and *N-Gram Selection Technique* (NST) [16] are two existing enhancements for the selection and play-out steps of MCTS, respectively. The basic idea of PH and NST is to introduce a bias in the respective steps towards playing actions, or sequences of actions, that performed well in earlier simulations. Because the value of playing an action in GVG-AI typically depends greatly on the current position of the avatar, this position is also

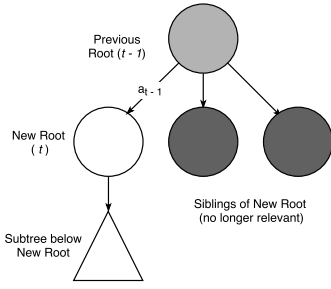


Fig. 3. Tree Reuse in MCTS.

taken into account when storing data concerning the previous performance of actions. For a detailed description of these enhancements we refer to the original publications [15], [16].

### B. Tree Reuse

Suppose that a search tree was built up by MCTS in a previous game tick  $t-1 \geq 0$ , and an action  $a_{t-1}$  was played. The entire subtree rooted in the node corresponding to that action can still be considered to be relevant for the new search process in the current tick  $t$ . Therefore, instead of initializing MCTS with only a root node, it can be initialized with a part of the tree built in the previous tick, as depicted in Figure 3. This was previously found to be useful in the real-time game of *Ms Pac-Man* [17]. This idea has also previously been suggested in the context of GVGP [11], but, to the best of our knowledge, the effect of this enhancement on the performance of MCTS in GVGP has not yet been evaluated.

In nondeterministic games, it is possible that the new root (which was previously a direct successor of the previous root) represented more than one possible game state. In the current tick, it is known exactly which of those possible states has been reached. Therefore, some of the old results in this tree are no longer relevant. For this reason, all the scores and visit counts in the tree are decayed by multiplying them by a decay factor  $\gamma \in [0, 1]$  before starting the next MCTS procedure. Tree Reuse (TR) with  $\gamma = 0$  completely resets the accumulated scores and visit counts of nodes (but still retains the nodes, and therefore the structure of the generated tree), and TR with  $\gamma = 1$  does not decay old results.

### C. Breadth-First Tree Initialization and Safety Prepruning

In some of the games supported by the GVG-AI framework, the number of MCTS simulations that can be executed in a single tick can be very small; sometimes smaller than the number of available actions. In such a situation, MCTS behaves nearly randomly, and is susceptible to playing actions that lead to a direct loss, even when there are actions available that do not directly lose the game.

Theoretically this problem could be avoided by adjusting the limit of the play-out depth of MCTS to ensure that a sufficient number of simulations can be done. In practice, this can be problematic because it requires a low initial depth limit to ensure that it is not too high at the start of a game, and this can in turn be detrimental in games where it is feasible and beneficial to run a larger number of longer play-outs.

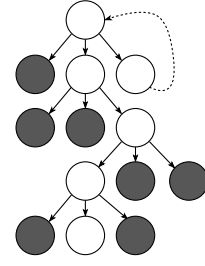


Fig. 4. Example search tree. Dark nodes represent losing game states, and white nodes represent winning or neutral game states.

We propose to handle this problem using Breadth-First Tree Initialization. The idea is straightforward; before starting MCTS, the direct successors of the root node are generated by a 1-ply Breadth-First Search. Every action available in the root state is executed up to a number  $M$  times to deal with nondeterminism, and the resulting states are evaluated. The average of these  $M$  evaluations is backpropagated for every successor with a weight equal to a single MCTS simulation. MCTS is only started after this process. When MCTS starts, every direct successor of the root node already has a prior evaluation that can be used to avoid playing randomly in cases with an extremely small number of simulations. The  $M$  states generated for every successor are cached in the corresponding nodes, so that they can be re-used in the subsequent MCTS process. This reduces the computational overhead of the enhancement.

*Safety prepruning*, originally used in an algorithm called Iterated Width [18], has been integrated in this process. The idea of safety prepruning is to count the number of immediate game losses among the  $M$  generated states for each action, and only keep the actions leading to nodes with the minimum observed number of losses. All other actions are pruned.

### D. Loss Avoidance

In GVGP, many games have a high number of losing game states that are relatively easy to avoid. An example of such a game is *Frogs*, where the avatar is a frog that should cross a road and a river. The road contains trucks that cause a loss upon collision, but can easily be avoided because they move at a constant speed. The river contains logs that also move at a constant speed, which the frog should jump on in order to safely cross the river.

An example of a search tree with many losing states is depicted in Figure 4. In this example, the rightmost action in the root node is an action that brings the agent back to a similar state as in the root node. In the *Frogs* game, this could be an action where the frog stays close to the initial position, and does not move towards the road.

The (semi-)random play used in the play-out step of MCTS is likely to frequently run into losing game states in situations like this. This leads to a negative evaluation of nodes that do in fact lead to a winning position. This is only corrected when sufficient simulations have been run such that the selection step of MCTS correctly biases the majority of the simulations towards a winning node. With a low simulation count in GVG-

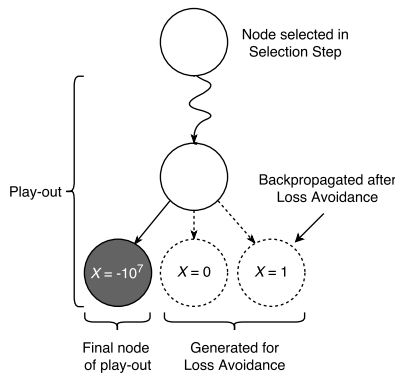


Fig. 5. Example MCTS simulation with Loss Avoidance. The  $X$  values in the last three nodes are evaluations of game states in those nodes. The dark node is a losing node.

AI, MCTS is likely to repeatedly play the rightmost action in Figure 4, which only delays the game until it is lost due to reaching the maximum game duration.

This problem is similar to the problem of *traps* [19] or *optimistic moves* [20] in (two-player) adversarial games. In those cases, MCTS has an overly optimistic evaluation of some states, whereas in the cases discussed here it has an overly pessimistic evaluation of some states. In [21], it was proposed to integrate shallow minimax searches inside some of the steps of MCTS to improve its performance in game trees with *traps* or *optimistic moves*. Using minimax searches to *prove* wins or losses is difficult in GVGP because games can be nondeterministic, but a similar idea can be used to get less pessimistic evaluations.

In this paper, an idea named *Loss Avoidance* (LA) is proposed for GVGP. The idea of LA is to try to ignore losses by immediately searching for a better alternative whenever a loss is encountered the first time a node is visited. An example is depicted in Figure 5. Whenever the play-out step of MCTS ends in a losing game state, that result is not backpropagated as would commonly be done in MCTS. Instead, one state is generated for every sibling of the last node, and only the evaluation of the node with the highest evaluation is backpropagated. All generated nodes are still added to the tree, and store their own evaluation in memory.

LA causes MCTS to keep an optimistic initial view of the value of nodes. This tends to work well in the single-player games of GVG-AI, where it is often possible to reactively get out of dangerous situations. It is unlikely to work well in, for instance, adversarial games, where a high concentration of losses in a subtree typically indicates that an opposing player has more options to win and is likely in a stronger position.

In an open-loop implementation of MCTS, LA can have a significant amount of computational overhead in game trees with many losses. For instance, in the *Frogs* game it roughly halves the average number of MCTS simulations per tick. This is because the node prior to the node with the losing game state does not store the corresponding game state in memory, which means that *all* states generated in the selection and play-out steps need to be re-generated by playing the same action sequence from the root node. In nondeterministic games this

process can also lead to finding a terminal state before the full action sequence has been executed again. To prevent spending too much time in the same simulation, the LA process is not started again, but the outcome of that state is backpropagated.

### E. Novelty-Based Pruning

The concept of *novelty tests* was first introduced in the Iterated Width algorithm (IW) [18], [22]. In IW, novelty tests are used for pruning in Breadth-First Search (BrFS). Whenever a state  $s$  is generated in a BrFS, a novelty measure (described in more detail below)  $nov(s)$  is computed. This is a measure of the extent to which  $s$  is “new” with respect to all previously generated states. States with a lower measure are “more novel” than states with a higher measure [22]. The original IW algorithm consists of a sequence of calls to  $IW(0)$ ,  $IW(1)$ , etc., where  $IW(i)$  is a BrFS that prunes a state  $s$  if  $nov(s) > i$ . In GVGP, it was found that it is only feasible to run a single  $IW(i)$  iteration [18]. The best results were obtained with  $IW(1)$ , and a variant named  $IW(\frac{3}{2})$  (see [18] for details).

The definition of the novelty measure  $nov(s)$  of a state  $s$  requires  $s$  to be defined in terms of a set of boolean features. An example of a boolean feature that can be a part of a state is a predicate  $at(cell, type)$ , which is true in  $s$  if and only if there is an object of the given type in the given cell in  $s$ . Then,  $nov(s)$  is defined as the size of the smallest tuple of features that are all true in  $s$ , and not all true in any other state generated previously in the same search process. If there is no such tuple,  $s$  must be an exact copy of some previously generated state, and  $nov(s)$  is defined as  $n + 1$ , where  $n$  is the number of features that are defined. For example, suppose that in  $s$ ,  $at((x, y), i) = true$ , and in all previously generated states,  $at((x, y), i) = false$ . Then,  $nov(s) = 1$ , because there is a tuple of size 1 of features that were not all true in any previously generated state.

$IW(1)$  prunes any state  $s$  with  $nov(s) > 1$ . In this paper, *Novelty-Based Pruning* (NBP) is proposed as an idea to prune nodes based on novelty tests in MCTS. The goal is not to prune *bad* lines of play, but to prune *redundant* lines of play.

MCTS often generates states deep in the tree before other states close to the root. For instance, the last state of the first play-out is much deeper in the tree than the first state of the second play-out. This is an important difference with the BrFS used by IW. It means that the novelty measure  $nov(s)$  of a state  $s$  should be redefined in such a way that it not necessarily uses *all* previously generated states, but only a specific set of states, referred to as the *neighborhood*  $N(s)$  of  $s$ .

$N(s)$  is the union of four sets of states. The first set consists of the siblings on the “left” side of  $s$ . The ordering of the states matters, but can be arbitrary (as in a BrFS). The second set contains only the parent  $p(s)$  of  $s$ . The third set consists of *all* siblings of  $p(s)$ . The fourth set is the neighborhood of  $p(s)$ . More formally, let  $s_i$  denote the  $i^{th}$  successor of a parent  $p(s_i)$ . Then,  $N(s_i)$  is defined as  $N(s_i) = \{s_1, s_2, \dots, s_{i-1}\} \cup \{p(s_i)\} \cup Sib(p(s_i)) \cup N(p(s_i))$ , where  $Sib(p(s_i))$  denotes the set of siblings of  $p(s_i)$ . For the root state  $r$ ,  $N(r) = Sib(r) = \emptyset$ . An example is depicted in Figure 6.



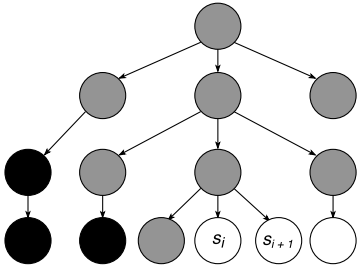


Fig. 6. States used for NBP in MCTS. The grey states are the neighborhood of  $s_i$  in MCTS. For  $s_{i+1}$ ,  $s_i$  is also included. The black states would be included for the novelty tests in IW, but not in MCTS.

Using the above definition of  $N(s)$ ,  $nov(s, N(s))$  is defined as the size of the smallest tuple of features that are all true in  $s$ , and not all true in any other state in the set  $N(s)$ . The novelty tests are used in MCTS as follows. Let  $n$  be a node with a list of successors  $Succ(n)$ . The first time that the selection step reaches  $n$  when it is fully expanded, all successors  $Succ(n)$  are novelty tested based on a single state generated per node, using a threshold of 1 for the novelty tests (as in IW(1)). The same boolean features are used to define states in GVG-AI as described in [18]. Nodes are marked as not being novel if they fail the novelty test. Whenever *all* successors of a node are marked as not novel, that node itself is also marked as not novel. There are a few exceptions where nodes are not marked. If a state has a higher game score than the parent, it is always considered to be novel. Additionally, states transitioned into by playing a movement action are always considered to be novel in games where either only horizontal, or only vertical movement is available (because these games often require moving back and forth which can get incorrectly pruned by NBP otherwise), and in games where the avatar has a movement speed  $\leq 0.5$  (because slow movement does not result in the avatar reaching a new cell every tick, and is therefore not detected by the cell-based boolean features).

In the selection step of MCTS, when one of the successors  $Succ(n)$  of  $n$  should be selected, any successor  $n' \in Succ(n)$  is ignored if it is marked as not novel, unless the average normalized score  $\bar{Q}(n) < 0.5$ . In such cases, the situation is considered to be dangerous and all alternatives should be considered to see if a better position can be found. For the final selection of the move to play in the real game, non-novel nodes are also only considered if the best novel alternative has a normalized average score  $< 0.5$ .

When the successors  $Succ(n)$  have been novelty tested, every node  $n_i \in Succ(n)$  stores a set of tuples of features that were all true in the states generated for the purpose of novelty testing for the nodes  $\{n\} \cup Succ(n)$ . This means that the tuples of features that are true in the neighborhood  $N(s)$  of a state  $s$  can be reconstructed relatively efficiently by traversing the path from  $s$  back to the root, and collecting the tuples in the stored sets. This is the main reason for defining  $N(s)$  as described above. Including more states (for instance, the black states in Figure 6) would require also traversing back down the tree to collect more sets of tuples. This could increase the number of nodes that NBP marks as not being novel, but

would also be more expensive computationally. This is not a problem in the BrFS of IW, because it can simply store all tuples of features that are all true in any generated state in the same set for the entire search process.

Novelty measures are assigned to nodes based on only one state per node. Therefore, given two identical open-loop game trees in nondeterministic games, it is possible that a node in one tree is pruned and the equivalent node in the other tree is not pruned. For this reason, when combining NBP with Tree Reuse, the results of novelty tests on nodes in the first ply below the new root node are reset when reusing the previous tree. This does not entirely remove the influence of nondeterminism on NBP, but close to the root that influence is at least reduced.

#### F. Knowledge-Based Evaluations

An important problem with MCTS in GVG-AI is that it is often infeasible to find any terminal states, or even states with a change in game score. This means that the evaluation function in Equation 2 often returns the same value for all states generated in the same tick, and MCTS explores the search space and behaves randomly. In this paper, a heuristic evaluation function is proposed that uses knowledge collected during simulations, and distances to objects that could potentially be interesting, to distinguish between states that have identical evaluations according to Equation 2. The basic idea is not new; some agents in the competition of 2014 used distance-based evaluation functions [4]. A similar idea is also described in [23], and extended in [24]. The idea discussed here is based on the same intuition, but a number of implementation details are different. Another related idea is described in [25], where MCTS is used to learn which objects are interesting, and a pathfinding algorithm is used to move towards a selected goal.

Let  $X(s_0)$  denote the evaluation of the current game state  $s_0$ , and let  $X(s_T)$  denote the evaluation of the final state  $s_T$  of a play-out. If  $X(s_T) = X(s_0)$ , a heuristic evaluation  $Eval_{KB}(s_T)$  is computed and added to  $X(s_T)$ . For every object type  $i$  observed in a game, let  $d_0(i)$  denote the distance from the avatar to the closest object of type  $i$  in  $s_0$ , and let  $d_T(i)$  denote the distance from the avatar to the closest object of type  $i$  in  $s_T$ . These distances are computed using the A\* pathfinding algorithm [26]. The pathfinding algorithm takes objects of the *wall* type into account as obstacles. Many games can also contain other objects that block movement, or portals that can be used for teleportation. These objects are not taken into account, because the agent would first need to learn how these objects influence pathfinding. For every object type  $i$ , a weight  $w_i$  is used to reward or punish the agent for moving to objects of that type. This is done by computing  $Eval_{KB}(s_T)$  as given by Equation 3, normalizing it to lie in  $[0, 0.5]$ , and adding it to  $X(s_T)$  if otherwise  $X(s_T) = X(s_0)$ .

$$Eval_{KB}(s_T) = \sum_i w_i \times (d_0(i) - d_T(i)) \quad (3)$$

Object types  $i$  with a small absolute weight ( $|w_i| < 10^{-4}$ ) are ignored, to save the computational cost of pathfinding.

The weights  $w_i$  are determined as follows. To motivate exploration, all weights are initialized with positive values (0.1 for NPCs, 0.25 for Movables, and 1 for Resources and Portals), and incremented by  $10^{-4}$  every game tick. States  $s_t$  generated during the selection or play-out steps of MCTS are used to adjust these weights. Let  $s_{t-1}$  denote the predecessor of  $s_t$ . Whenever such a state  $s_t$  is generated, it is used to update some of the weights  $w_i$ . The intuition is that, if  $X(s_t) \neq X(s_{t-1})$ , it is likely that some interesting collision event occurred in the transition from  $s_{t-1}$  to  $s_t$  that caused the change in score. The framework provides access to a set  $E(s_t)$  of collision events that occurred in that transition. Every event  $e \in E(s_t)$  is a collision event between two objects, where one object is either the avatar, or an object created by the avatar (for instance, a missile fired by the avatar), and the other object is of some type  $i$ . Let  $\Delta = X(s_t) - X(s_{t-1})$  denote the observed change in score. For every object type  $i$ , a sum  $\Delta_i$  is kept of all changes in scores observed in state transitions where collision events with objects of type  $i$  occurred. Additionally, a counter  $n_i$  of event occurrences is kept for every type  $i$ , such that the average change in score  $\bar{\Delta}_i = \frac{\Delta_i}{n_i}$  for collisions with every type can be computed. Whenever an event with an object of type  $i$  is observed,  $w_i$  is updated as given by Formula 4.

$$w_i \leftarrow w_i + (\bar{\Delta}_i - w_i) \times \alpha_i \quad (4)$$

$\alpha_i$  is a learning rate that is initialized to 0.8 for every type, and updated as given by Formula 5 after updating  $w_i$ .

$$\alpha_i \leftarrow \max(0.1, 0.75 \times \alpha_i) \quad (5)$$

This idea is similar to using gradient descent for minimizing  $|\bar{\Delta}_i - w_i|$ . The main reason for not simply using  $\bar{\Delta}_i$  directly is to avoid relying too much on the knowledge obtained from a low number of observed events.

### G. Deterministic Game Detection

The idea of *Deterministic Game Detection* (DGD) is to detect when a game is likely to be deterministic, and treat deterministic games differently from nondeterministic games. At the start of every game,  $M$  random sequences of actions of length  $N$  are generated. Each of the  $M$  sequences is used to advance a copy of the initial game state  $s_0$ , with  $R$  repetitions per sequence. If any of the  $M$  action sequences did not result in equivalent states among the  $R$  repetitions for that sequence, the game is classified as nondeterministic. Additionally, any game in which NPCs are observed is immediately classified as nondeterministic. Any other game is classified as deterministic. In this paper,  $M = N = 5$  and  $R = 3$ .

Many participants in previous GVG-AI competitions [7] used a similar idea to switch to a different algorithm for deterministic games (for instance, using Breadth-First Search in deterministic games and MCTS in nondeterministic games). In this paper, DGD is only used to modify MCTS and the TR and NBP enhancements in deterministic games. The  $\bar{Q}(S_i)$  term in Equation 1 (or the equivalent term in the formula of PH) is replaced by  $\frac{3}{4} \times \bar{Q}(S_i) + \frac{1}{4} \times \bar{Q}_{max}(S_i)$ , where  $\bar{Q}_{max}(S_i)$  is the maximum score observed in the subtree rooted in  $S_i$ .

TABLE I  
WIN PERCENTAGES (BENCHMARK AGENTS, 1000 RUNS PER SET)

Sets	SOLMCTS	MCTS	IW(1)	YBCRIBER
Set 1	34.5 $\pm$ 2.9	41.7 $\pm$ 3.1	55.8 $\pm$ 3.1	68.8 $\pm$ 2.9
Set 2	33.4 $\pm$ 2.9	33.5 $\pm$ 2.9	47.0 $\pm$ 3.1	65.0 $\pm$ 3.0
Set 3	13.2 $\pm$ 2.1	23.0 $\pm$ 2.6	17.8 $\pm$ 2.4	40.3 $\pm$ 3.0
Set 4	28.3 $\pm$ 2.8	30.5 $\pm$ 2.9	30.6 $\pm$ 2.9	43.5 $\pm$ 3.1
Set 5	19.7 $\pm$ 2.5	28.9 $\pm$ 2.8	17.5 $\pm$ 2.4	42.6 $\pm$ 3.1
Set 6	30.1 $\pm$ 2.8	28.6 $\pm$ 2.8	32.8 $\pm$ 2.9	54.4 $\pm$ 3.1
Total	26.5 $\pm$ 1.1	31.0 $\pm$ 1.2	33.6 $\pm$ 1.2	52.4 $\pm$ 1.3

This is referred to as *mixmax* [27], [28]. Additionally, TR and NBP are modified to no longer decay or reset any old results.

## V. EXPERIMENTS

### A. Setup

The enhancements discussed in this paper have been experimentally evaluated using the following setup. Every experiment was run using six sets that are available in the framework, of ten games each, for a total of sixty different games per experiment. Table VI lists the names of the games for every set. Average results are presented for every set of games, and for the total of all sixty games combined. For every game, five different levels were used, with a minimum of fifteen repetitions per level per experiment (leading to a minimum of 750 runs per set). 95% confidence intervals are presented for all results. All games were played according to the GVG-AI competition rules<sup>1</sup>, on a CentOS Linux server consisting of four AMD Twelve-Core OpteronT 6174 processors (2.2 GHz).

### B. Results

In the first experiment, the following benchmark agents are compared to each other; SOLMCTS, MCTS, IW(1), and YBCRIBER. SOLMCTS is the Sample Open Loop MCTS controller included in the framework. MCTS is our baseline implementation of MCTS, based on the MAASTCTS [29] agent, which has a number of differences in comparison to SOLMCTS. MCTS expands all nodes for states generated in simulations (as opposed to one node per simulation),  $C$  is set to 0.6 in the UCB1 equation (as opposed to  $C = \sqrt{2}$ ), it simulates up to ten actions after the selection step (as opposed to ten steps from the root node), it uses the 1 second of initialization time for running the algorithm (as opposed to not using that time), and it plays the action with the maximum average score (as opposed to the maximum visit count). IW(1) is the Iterated Width-based agent, as described in [18]. YBCRIBER is an IW-based agent with a number of other features, which won the GVG-AI competition at the IEEE CEEC 2015 conference. The results are given in Table I. The experimental data reveals that the baseline MCTS agent outperforms SOLMCTS. IW(1) performs slightly better than MCTS overall, and YBCRIBER performs much better than the other benchmark agents.

In Table II, our MCTS implementation with Breadth-First Tree Initialization and Safety Prepruning (BFTI) is compared

<sup>1</sup>Revision 24b11aea75722ab02954c326357949b97efb7789 of the GVG-AI framework (<https://github.com/EssexUniversityMCTS/gvgai>) was used.

TABLE II  
BREADTH-FIRST TREE INITIALIZATION WITH SAFETY PREPRUNING

Sets	Win Percentage		% of Losses $t < 2000$	
	BFTI	MCTS	BFTI	MCTS
Set 1	43.3 $\pm$ 3.5	41.7 $\pm$ 3.1	42.6 $\pm$ 4.7	52.8 $\pm$ 4.1
Set 2	33.1 $\pm$ 3.4	33.5 $\pm$ 2.9	50.8 $\pm$ 4.4	51.1 $\pm$ 3.8
Set 3	21.2 $\pm$ 2.9	23.0 $\pm$ 2.6	0.0 $\pm$ 0.0	16.1 $\pm$ 2.6
Set 4	30.3 $\pm$ 3.3	30.5 $\pm$ 2.9	73.4 $\pm$ 3.8	76.8 $\pm$ 3.1
Set 5	23.1 $\pm$ 3.0	28.9 $\pm$ 2.8	72.4 $\pm$ 3.6	73.7 $\pm$ 3.2
Set 6	29.2 $\pm$ 3.3	28.6 $\pm$ 2.8	69.3 $\pm$ 3.9	72.3 $\pm$ 3.3
Total	30.0 $\pm$ 1.3	31.0 $\pm$ 1.2	51.0 $\pm$ 1.7	56.7 $\pm$ 1.5

TABLE III  
WIN PERCENTAGES (PH AND NST, 750 RUNS PER SET)

Sets	BFTI	PH	NST	NST+PH
Set 1	43.3 $\pm$ 3.5	43.2 $\pm$ 3.5	45.1 $\pm$ 3.6	43.5 $\pm$ 3.5
Set 2	33.1 $\pm$ 3.4	34.5 $\pm$ 3.4	36.5 $\pm$ 3.4	38.0 $\pm$ 3.5
Set 3	21.2 $\pm$ 2.9	23.3 $\pm$ 3.0	23.1 $\pm$ 3.0	24.1 $\pm$ 3.1
Set 4	30.3 $\pm$ 3.3	29.5 $\pm$ 3.3	29.7 $\pm$ 3.3	32.3 $\pm$ 3.3
Set 5	23.1 $\pm$ 3.0	23.9 $\pm$ 3.1	30.0 $\pm$ 3.3	28.0 $\pm$ 3.2
Set 6	29.2 $\pm$ 3.3	30.0 $\pm$ 3.3	31.1 $\pm$ 3.3	33.1 $\pm$ 3.4
Total	30.0 $\pm$ 1.3	30.7 $\pm$ 1.3	32.6 $\pm$ 1.4	33.2 $\pm$ 1.4

to the MCTS implementation without BFTI. The results for MCTS are based on 1000 runs per set, and the results for BFTI on 750 runs per set. BFTI appears to lower the win percentage slightly, but the 95% confidence intervals overlap. The two columns on the right-hand side show the percentage of lost games where the game was terminated before  $t = 2000$  (where  $t = 2000$  is the maximum duration of a game in GVG-AI). BFTI reduces this percentage significantly. Even though it may slightly decrease win percentages, the quality of play in lost games can be considered to be improved; the agent delays a significant number of losses. This may leave more time for other enhancements to find wins. Therefore, BFTI is included in the baseline MCTS agent for the following experiments that evaluate other enhancements individually. This is followed by an experiment with more enhancements combined.

Table III shows the win percentages obtained by adding Progressive History (PH), N-Gram Selection Technique (NST), or both to the BFTI agent. PH and NST appear to increase the average win percentage, but the confidence intervals overlap. The two combined result in a statistically significant increase.

Figure 7 depicts 95% confidence intervals for the win percentage of the BFTI agent with Tree Reuse (TR), for six different values of the decay factor  $\gamma$ . The confidence interval for BFTI is shaded in grey. TR with  $\gamma \in \{0.4, 0.6, 1.0\}$  significantly improves the win percentage of BFTI.

Table IV shows the win percentages of adding either Knowledge-Based Evaluations (KBE), Loss Avoidance (LA) or Novelty-Based Pruning (NBP) to the BFTI agent. All three individually show an increase in the average win percentage over BFTI, with KBE giving the largest increase.

Table V shows the win percentages of a number of variants of MCTS with multiple enhancements combined. “No DGD” is an agent with all enhancements discussed in this paper, except for Deterministic Game Detection (DGD). “No BFTI” is an agent with all enhancements except for BFTI. This is added to test the assumption made earlier that the ability of BFTI

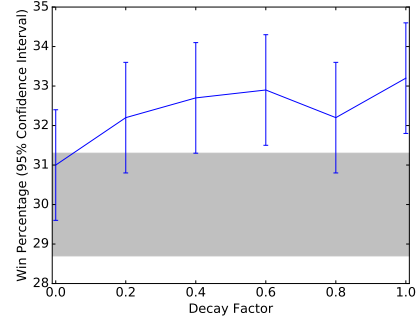


Fig. 7. 95% confidence intervals for win percentages of BFTI with Tree Reuse (TR) for different values of the decay factor  $\gamma$ . The area shaded in grey is the confidence interval for the win percentage of BFTI without TR.

TABLE IV  
WIN PERCENTAGES (KBE, LA AND NBP, 750 RUNS PER SET)

Sets	BFTI	KBE	LA	NBP
Set 1	43.3 $\pm$ 3.5	50.4 $\pm$ 3.6	52.0 $\pm$ 3.6	49.6 $\pm$ 3.6
Set 2	33.1 $\pm$ 3.4	52.3 $\pm$ 3.6	34.0 $\pm$ 3.4	34.5 $\pm$ 3.4
Set 3	21.2 $\pm$ 2.9	19.1 $\pm$ 2.8	23.3 $\pm$ 3.0	23.5 $\pm$ 3.0
Set 4	30.3 $\pm$ 3.3	30.1 $\pm$ 3.3	29.6 $\pm$ 3.3	32.0 $\pm$ 3.3
Set 5	23.1 $\pm$ 3.0	31.3 $\pm$ 3.3	31.9 $\pm$ 3.3	23.9 $\pm$ 3.1
Set 6	29.2 $\pm$ 3.3	33.2 $\pm$ 3.4	28.8 $\pm$ 3.2	34.8 $\pm$ 3.4
Total	30.0 $\pm$ 1.3	36.1 $\pm$ 1.4	33.3 $\pm$ 1.4	33.0 $\pm$ 1.4

to delay games may enable other enhancements to find more wins. The last agent contains all enhancements. In combination with all the other enhancements, DGD significantly improves the win percentage. DGD was found not to provide a significant increase in win percentage when applied to the BFTI, TR ( $\gamma = 0.6$ ) or NBP agents without other enhancements (those results have been omitted to save space). Additionally, BFTI appears to increase the win percentage in combination with all other enhancements, whereas Table II shows it appears to *decrease* the win percentage when other enhancements are absent, but these differences are not statistically significant.

## VI. CONCLUSION AND FUTURE WORK

Eight enhancements for Monte-Carlo Tree Search (MCTS) in General Video Game Playing (GVGP) have been discussed and evaluated. Most of them have been shown to significantly (95% confidence) increase the average win percentage over sixty different games when added individually to MCTS. All the enhancements combined increase the win percentage of our basic MCTS implementation from  $31.0 \pm 1.2$  to  $48.4 \pm 1.5$ . This final performance is relatively close to the win percentage of the winner of the IEEE CEEC 2015 conference; YBCRIBER, with a win percentage of  $52.4 \pm 1.3$ .

Many of the discussed enhancements have parameters, which so far have only been tuned according to short, preliminary experiments. These parameters can likely be tuned better in future work to improve the performance. Loss Avoidance (LA) and Novelty-Based Pruning (NBP) as proposed in this paper have binary effects, in that LA backpropagates only one result from multiple generated siblings and NBP classifies nodes as either novel or not novel. Perhaps these can be improved by making them less binary. The overall performance of the agent can also likely be improved by incorporating more features that are commonly seen among the top entries in past

TABLE V  
WIN PERCENTAGES (ENHANCEMENTS COMBINED, 750 RUNS PER SET)

Sets	BFTI	No DGD	No BFTI	All Enhanc.
Set 1	43.3 ± 3.5	62.7 ± 3.5	62.7 ± 3.5	62.8 ± 3.5
Set 2	33.1 ± 3.4	56.4 ± 3.5	55.7 ± 3.6	59.3 ± 3.6
Set 3	21.2 ± 2.9	22.1 ± 3.0	28.5 ± 3.2	28.7 ± 3.2
Set 4	30.3 ± 3.3	32.7 ± 3.4	47.1 ± 3.6	48.1 ± 3.6
Set 5	23.1 ± 3.0	37.2 ± 3.5	39.6 ± 3.5	42.1 ± 3.5
Set 6	29.2 ± 3.3	38.3 ± 3.5	49.2 ± 3.6	49.1 ± 3.6
Total	30.0 ± 1.3	41.6 ± 1.4	47.1 ± 1.5	48.4 ± 1.5

TABLE VI  
NAMES OF THE GAMES IN EVERY SET

Set 1	<i>Aliens, Boulderdash, Butterflies, Chase, Frogs, Missile Command, Portals, Sokoban, Survive Zombies, Zelda</i>
Set 2	<i>Camel Race, Digidug, Firestorms, Infection, Firecaster, Overload, Pacman, Sequest, Whackamole, Eggomania</i>
Set 3	<i>Bait, BoloAdventures, BrainMan, ChipsChallenge, Modality, Painter, RealPortals, RealSokoban, TheCitadel, ZenPuzzle</i>
Set 4	<i>Roguelike, Surround, Catapults, Plants, Plaque-Attack, Jaws, Labyrinth, Boulderchase, Escape, Lemmings</i>
Set 5	<i>Solarfox, Defender, Enemy Citadel, Crossfire, Lasers, Sheriff, Chopper, Superman, WaitForBreakfast, CakyBaky</i>
Set 6	<i>Lasers 2, Hungry Birds, Cook me Pasta, Factory Manager, Race Bet 2, Intersection, Black Smoke, Ice and Fire, Gymkhana, Tercio</i>

competitions, such as the use of influence maps [30]. Finally, some of the new enhancements for MCTS, such as LA and NBP, can be evaluated in domains other than GVG-AI.

#### ACKNOWLEDGEMENT

This work is partially funded by the Netherlands Organisation for Scientific Research (NWO) in the framework of the project GoGeneral, grant number 612.001.121.

#### REFERENCES

- [1] J. Levine, C. B. Congdon, M. Ebner, G. Kendall, S. M. Lucas, R. M. anad T. Schaul, and T. Thompson, "General Video Game Playing," in *Artif. and Comput. Intell. in Games*, ser. Dagstuhl Follow-Ups, S. M. Lucas, M. Mateas, M. Preuss, P. Spronck, and J. Togelius, Eds. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013, vol. 6, pp. 77–83.
- [2] M. Genesereth, N. Love, and B. Pell, "General Game Playing: Overview of the AAAI Competition," *AI Magazine*, vol. 26, no. 2, pp. 62–72, 2005.
- [3] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The Arcade Learning Environment: An Evaluation Platform for General Agents," *Journal of Artif. Intell. Research*, vol. 47, pp. 253–279, 2013.
- [4] D. Perez, S. Samothrakakis, J. Togelius, T. Schaul, S. M. Lucas, A. Couëtoux, J. Lee, C.-U. Lim, and T. Thompson, "The 2014 General Video Game Playing Competition," *IEEE Trans. on Comput. Intell. and AI in Games*, 2016, To appear.
- [5] M. Ebner, J. Levine, S. M. Lucas, T. Schaul, T. Thompson, and J. Togelius, "Towards a Video Game Description Language," in *Artif. and Comput. Intell. in Games*, ser. Dagstuhl Follow-Ups, S. M. Lucas, M. Mateas, M. Preuss, P. Spronck, and J. Togelius, Eds. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013, vol. 6, pp. 85–100.
- [6] T. Schaul, "A Video Game Description Language for Model-based or Interactive Learning," in *Proc. of the IEEE Conf. on Comput. Intell. in Games*. Niagara Falls: IEEE Press, 2013, pp. 193–200.
- [7] D. Perez-Liebana, S. Samothrakakis, J. Togelius, S. M. Lucas, and T. Schaul, "General Video Game AI: Competition, Challenges and Opportunities," in *Proc. of the Thirtieth AAAI Conf. on Artif. Intell.* AAAI Press, 2016, pp. 4335–4337.
- [8] L. Kocsis and C. Szepesvári, "Bandit Based Monte-Carlo Planning," in *Machine Learning: ECML 2006*, ser. LNCS, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds. Springer Berlin Heidelberg, 2006, vol. 4212, pp. 282–293.
- [9] R. Coulom, "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search," in *Computers and Games*, ser. LNCS, H. J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers, Eds., vol. 4630. Springer Berlin Heidelberg, 2007, pp. 72–83.

- [10] Y. Björnsson and H. Finnsson, "CadiaPlayer: A Simulation-Based General Game Player," *IEEE Trans. on Comput. Intell. and AI in Games*, vol. 1, no. 1, pp. 4–15, 2009.
- [11] D. Perez, J. Dieskau, M. Hünemann, S. Mostaghim, and S. M. Lucas, "Open Loop Search for General Video Game Playing," in *Proc. of the Genetic and Evol. Computation Conf.* ACM, 2015, pp. 337–344.
- [12] G. M. J.-B. Chaslot, M. H. M. Winands, H. J. van den Herik, J. W. H. M. Uiterwijk, and B. Bouzy, "Progressive Strategies for Monte-Carlo Tree Search," *New Mathematics and Natural Computation*, vol. 4, no. 3, pp. 343–357, 2008.
- [13] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time Analysis of the Multiarmed Bandit Problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, 2002.
- [14] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *IEEE Trans. on Comput. Intell. and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.
- [15] J. A. M. Nijssen and M. H. M. Winands, "Enhancements for Multi-Player Monte-Carlo Tree Search," in *Computers and Games (CG 2010)*, ser. LNCS, H. J. van den Herik, H. Iida, and A. Plaat, Eds. Springer Berlin Heidelberg, 2011, vol. 6515, pp. 238–249.
- [16] M. J. W. Tak, M. H. M. Winands, and Y. Björnsson, "N-Grams and the Last-Good-Reply Policy Applied in General Game Playing," *IEEE Trans. on Comput. Intell. and AI in Games*, vol. 4, no. 2, pp. 73–83, 2012.
- [17] T. Pepels, M. H. M. Winands, and M. Lanctot, "Real-Time Monte Carlo Tree Search in Ms Pac-Man," *IEEE Trans. on Comput. Intell. and AI in Games*, vol. 6, no. 3, pp. 245–257, 2014.
- [18] T. Geffner and H. Geffner, "Width-Based Planning for General Video-Game Playing," in *Proc. of the Eleventh Artif. Intell. and Interactive Digital Entertainment International Conf.*, A. Jhala and N. Sturtevant, Eds. AAAI Press, 2015, pp. 23–29.
- [19] R. Ramanujan, A. Sabharwal, and B. Selman, "On Adversarial Search Spaces and Sampling-Based Planning," in *20th International Conf. on Automated Planning and Scheduling*, R. I. Brafman, H. Geffner, J. Hoffmann, and H. A. Kautz, Eds. AAAI, 2010, pp. 242–245.
- [20] H. Finnsson and Y. Björnsson, "Game-Tree Properties and MCTS Performance," in *IJCAI'11 Workshop on General Intelligence in Game Playing Agents (GIGA'11)*, Y. Björnsson, N. Sturtevant, and M. Thielscher, Eds., 2011, pp. 23–30.
- [21] H. Baier and M. H. M. Winands, "MCTS-Minimax Hybrids," *IEEE Trans. on Comput. Intell. and AI in Games*, vol. 7, no. 2, pp. 167–179, 2015.
- [22] N. Lipovetzky and H. Geffner, "Width and Serialization of Classical Planning Problems," in *Proc. of the Twentieth European Conf. on Artif. Intell. (ECAI 2012)*, L. De Raedt, C. Bessière, D. Dubois, P. Doherty, P. Frasconi, F. Heintz, and P. Lucas, Eds. IOS Press, 2012, pp. 540–545.
- [23] D. Perez, S. Samothrakakis, and S. Lucas, "Knowledge-based Fast Evolutionary MCTS for General Video Game Playing," in *Proc. of the IEEE Conf. on Comput. Intell. and Games*. IEEE, 2014, pp. 68–75.
- [24] J. van Eeden, "Analysing and Improving the Knowledge-based Fast Evolutionary MCTS Algorithm," Master's thesis, Utrecht University, Utrecht, the Netherlands, 2015.
- [25] C. Y. Chu, H. Hashizume, Z. Guo, T. Harada, and R. Thawonmas, "Combining Pathfinding Algorithm with Knowledge-based Monte-Carlo Tree Search in General Video Game Playing," in *Proc. of the IEEE Conf. on Comput. Intell. and Games*. IEEE, 2015, pp. 523–529.
- [26] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 100–107, 1968.
- [27] E. J. Jacobsen, R. Greve, and J. Togelius, "Monte Mario: Platforming with MCTS," in *Proc. of the 2014 Conf. on Genetic and Evolutionary Computation*. ACM, 2014, pp. 293–300.
- [28] F. Frydenberg, K. R. Andersen, S. Risi, and J. Togelius, "Investigating MCTS Modifications in General Video Game Playing," in *Proc. of the IEEE Conf. on Comput. Intell. and Games*. IEEE, 2015, pp. 107–113.
- [29] T. Schuster, "MCTS Based Agent for General Video Games," Master's thesis, Department of Knowledge Engineering, Maastricht University, Maastricht, the Netherlands, 2015.
- [30] I. Millington and J. Funge, *Artificial Intelligence for Games*, 2nd ed. Morgan Kaufmann, 2009.

# Position-based Reinforcement Learning Biased MCTS for General Video Game Playing

Chun-Yin Chu

Graduate School of Information Science and Engineering  
Ritsumeikan University  
Kusatsu, Shiga, Japan

Suguru Ito, Tomohiro Harada, Ruck Thawonmas

College of Information Science and Engineering  
Ritsumeikan University  
Kusatsu, Shiga, Japan

**Abstract**—This paper proposes an application of reinforcement learning and position-based features in rollout bias training of Monte-Carlo Tree Search (MCTS) for General Video Game Playing (GVGP). As an improvement on Knowledge-based Fast-Evo MCTS proposed by Perez et al., the proposed method is designated for both the GVG-AI Competition and improvement of the learning mechanism of the original method. The performance of the proposed method is evaluated empirically, using all games from six training sets available in the GVG-AI Framework, and the proposed method achieves better scores than five other existing MCTS-based methods overall.

**Keywords**—General Video Game Playing; Reinforcement Learning; Monte-Carlo Tree Search;

## I. INTRODUCTION

Despite being a new research topic, General Video Game Playing (GVGP) has gathered the interests of many AI researchers recently. Monte-Carlo Tree Search (MCTS) is widely applied in the field of GVGP and has proven to be effective in a wide range of games. In the past few years, many researchers have proposed modifications to MCTS so as to further improve its performance in GVGP, and Perez et al.'s Knowledge-based Fast-Evolutionary MCTS (KB Fast-Evo MCTS) [1] is one of such proposals. Using  $(1 + 1)$  Evolutionary Strategy (ES) to train the rollout bias and game-specific knowledge acquired via a rollout as a rollout reward, KB Fast-Evo MCTS was successful in outperforming standard UCT-based MCTS [2] in the domain of GVGP. While innovative and effective, KB Fast-Evo MCTS bears a number of shortcomings. The training mechanism  $(1 + 1)$  ES might be efficient, but too randomized as a learning mechanism; and using Euclidean distances between the avatar and other objects as the features oversimplifies the game state at hand. To overcome such shortcomings, this paper proposes an improvement on KB Fast-Evo MCTS, by replacing  $(1 + 1)$  ES and distance-based features with reinforcement learning and position-based features.

Reinforcement learning is no stranger to application in GVGP. In past researches, reinforcement learning has been applied in training a game-playing agent [3] and recognizing contingency awareness in Atari 2600 games [4]. However, in our approach, reinforcement learning is used for training rollout bias of MCTS, instead of feature extraction or a stand-alone decision-making mechanism. Being able to update the weight matrix at every turn of the rollout, reinforcement

learning is an efficient and flexible mechanism for training rollout bias.

In order to apply reinforcement learning in training rollout bias, an effective feature extraction method for sampling game states is instrumental. Game states in GVGP are often complicated, involving dozens of objects and NPCs. Thus, a feature extraction method that can simplify the game state while preserving its meaningful features is necessary. Previous studies have attempted the use of various features, for instance, the Euclidean distances between avatar and other objects [1], or game statistics such as time steps and scores [4]. In this paper, we propose the use of position, i.e., relative coordinates, of nearby objects as the extracted features for representing the game state and biasing rollout.

In this paper, our proposal of using reinforcement learning and position-based features for training rollout bias is discussed. Open Loop search structure is also incorporated to further enhance the performance of our method. The method described in this paper is part of our entry in the GVG-AI Competition 2016. While our agent applies other techniques such as Breath-First-Search and pathfinding, this paper focuses mainly on the use of reinforcement learning and position-based features for training rollout bias.

## II. BACKGROUND AND RELATED WORK

### A. General Video Game Playing

The goal of GVGP research is to develop a game-playing agent that can play various types of games. While traditional general game playing focuses on board games, GVGP mainly concerns real-time 2D video games. There are two active GVGP platforms - the Arcade Learning Environment (ALE) [5] and the GVG-AI Framework. Using the latter as its platform, the GVG-AI Competition has picked up steam and received more than 70 submissions in last year [6].

The major difference between ALE and GVG-AI Framework is that the former is emulator-based while the latter is model-based. On the one hand, ALE provides only the visual outputs and memory data, and the AI program has to deduce the current game state from these inputs. On the other hand, the GVG-AI Framework provides information on the current game state directly from the game engine to the game-playing agent. The availability of game state information does not only clear the hurdle of deducing game states from visual outputs, but also allows simulation and exploration of future game states, thus allowing for simulation-based approaches such as MCTS.

### B. MCTS variants for GVG-AI Competition

Relying on not domain-specific knowledge but random sampling of the search space for decision making, MCTS is a viable and versatile technique in GVGP. In past competitions and studies, application of MCTS in GVG-AI framework is frequently attempted and researched. For instance, apart from the basic UCT (Upper Confidence Bound 1 applied to trees), researchers have attempted a combination of UCT and influence map [7]. Other researchers investigated various modifications of MCTS, such as reverse penalty and MixMax backups, in the GVG-AI framework [8].

One of the more remarkable MCTS variants is KB Fast-Evo MCTS, proposed by Perez et al. in 2014 [1]. KB Fast-Evo MCTS improves UCT in two ways: the introduction of a knowledge-based reward formula and the use of rollout bias trained by  $(1 + 1)$  ES. In the GVG-AI framework, the game-playing agent has no knowledge of the game rule prior to playing the game, meaning that there is no telling of which objects are beneficial to the player (i.e., the score increases upon collision) and which objects are hostile (i.e. score is reduces or the player is killed upon collision). KB Fast-Evo MCTS overcomes this by acquiring knowledge during rollouts. The effect of colliding with different types of objects is examined in simulations and beneficial objects are identified in the process. The knowledge acquired is then applied to reward calculation at the rollout end. Similar to UCT in the GVG-AI framework, KB Fast-Evo MCTS uses score change as a main reward. However, if the score has not changed after the rollout, i.e.,  $\Delta R = 0$ , knowledge change  $\Delta Z = \sum_{i=1}^N \Delta K_i$  and distance change  $\Delta D = \sum_{i=1}^N \Delta D_i$  are used instead, where  $N$  is the number of object types; a rollout that increases knowledge and reduces the distance between the avatar and beneficial or unknown object will lead to a higher reward, according to the following formulae:

$$\Delta K_i = \begin{cases} Z_{iF} & : Z_{i0} = 0 \\ \frac{Z_{iF}}{Z_{i0}} - 1 & : \text{Otherwise} \end{cases} \quad (2.1)$$

$$\Delta D_i = \begin{cases} 1 - \frac{D_{iF}}{D_{i0}} & : Z_{i0} = 0 \text{ OR } D_{i0} > 0 \text{ AND } \bar{x}_i > 0 \\ 0 & : \text{Otherwise} \end{cases} \quad (2.2)$$

$$\text{Reward} = \begin{cases} \Delta R & : \Delta R \neq 0 \\ 0.66 \times \Delta Z + 0.33 \times \Delta D & : \text{Otherwise} \end{cases} \quad (2.3)$$

In the above formulae,  $Z_{i0}$  and  $Z_{iF}$  represent the number of occurrences of event  $i$  before and after the rollout, respectively.  $D_{i0}$  and  $D_{iF}$  are the Euclidean distance to the closest sprite type  $i$  at the beginning of rollout and after the rollout, respectively.  $\bar{x}_i$  is the average score change triggered by event  $i$ .

Another way by which KB Fast-Evo MCTS enhances the performance of UCT is the use of rollout bias, which is trained by  $(1 + 1)$  ES. Normally, UCT performs rollouts randomly, meaning that each action during a rollout is selected randomly. Selecting actions randomly may be simple and efficient, but not effective and intelligent. KB Fast-Evo MCTS attempts to

improve the quality of rollouts by tuning the rollouts using a weight vector and features extracted from the game state. In KB Fast-Evo MCTS, for each type of object in the game state, the Euclidean distance between the avatar and the closest object of that type is extracted as a feature of the game state. The feature value  $f_j$  for object type  $j$  is then used in calculating the weight of action  $i$ ,  $a_i$ , using the formula below:

$$a_i = \sum_{j=1}^N w_{ij} \times f_j \quad (2.4)$$

In the above formula,  $N$  is the total number of features, and  $w_{ij}$  is the  $ij$ -th element in the weight matrix  $W$  being trained by  $(1 + 1)$  ES, using the reward of the rollout as the fitness value. The probability of selecting each action among  $A$  available actions is then calculated using the following softmax formula:

$$P(i) = \frac{e^{-a_i}}{\sum_{j=1}^A e^{-a_j}} \quad (2.5)$$

Although KB Fast-Evo MCTS introduced many novel modifications to UCT, the algorithm is not without flaws. For example, Jim van Eeden [9] criticized that the use of Euclidean distances does not take obstacles into account, and the ES-trained weight matrix converges too quickly, therefore not being able to evolve in the later part of the game. In light of its shortcomings, he further improved KB Fast-Evo MCTS by introducing the A\* pathfinding algorithm, changing the evolutionary approach, and fine-tuning the parameters and formulae of the original method.

Another noteworthy MCTS variant is Open Loop Search for GVGP [10], proposed by Perez et al. 2015. In ordinary UCT, game states visited during rollouts will not be recorded and remembered; but in Open Loop Search, statistics of each state, i.e., the average score and highest score, are stored in the game tree and carry over to inform the next search. Moreover, Perez et al.'s method introduced the use of pheromone trail, adding negative and degrading potential field to recently visited positions so as to discourage the avatar from staying in the same position during the simulation. The search structure is compatible to various search algorithms, including Rolling Horizon Evolutionary Algorithm, MCTS and Directed Breath-First-Search.

### C. Reinforcement Learning in GVGP

Reinforcement Learning is a common technique in the field of GVGP. Google DeepMind team developed deep reinforcement learning, a variant of Q-Learning that can learn playing Atari 2600 games and outperform human experts in several games [11]. Hausknecht et al. [12] experimented with four different types of evolutionary neural networks and showed that evolved policies are capable of beating human experts in three Atari 2600 games. SARSA( $\lambda$ ), a traditional model-free reinforcement learning method, was applied as a benchmark for the ALE platform [5].

While many applications of reinforcement learning in GVGP have been proposed, these methods can hardly be directly applied in the GVG-AI framework for several reasons. One is that existing reinforcement learning methods for GVGP



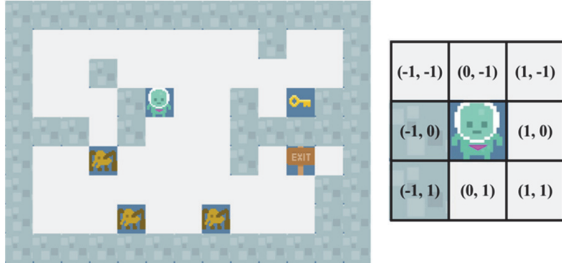


Fig. 1. A game state in the zelda game (left) and extracted features (right)

are catered the ALE platform, not the GVG-AI framework. Also, the existing methods require a long training time to learn the game rules beforehand, but the Planning track of GVG-AI Competition allows little time for training before the game starts, rendering existing reinforcement learning or neural network techniques inapplicable. Nonetheless, these existing methods are inspirations for our proposed method.

Apart from its application in GVGP, reinforcement learning has been widely adopted to enhance MCTS in game AI. Using temporal difference values to evaluate game states, Vodopivec and Šter proposed an enhancement of the UCT algorithm with the temporal difference algorithm [13]. While their work precedes our study, there are many notable differences between the two. First, their method used reinforcement learning for estimating the value of game state, whereas our study investigates the use of reinforcement learning for training rollout bias. And more importantly, their method was applied to two-player turn-based games such as Gomoku and Tic-Tac-Toe, and the parameters were tuned offline on 10,000 to 100,000 games, while our method is intended for single-player real-time videogames with little training time.

Recently, the use of reinforcement learning and MCTS in Computer Go has seen major. Using reinforcement learning to train the policy and value networks in an MCTS algorithm, Silver et al. successfully built a Computer Go AI called AlphaGo that managed to defeat human professional players in full-size Go games [14]. Prior to AlphaGo, Silver has also introduced temporal-difference search, which is a combination of reinforcement learning and simulation-based search methods [15]. Due to their extensive uses of reinforcement learning techniques, AlphaGo and temporal-difference search are relevant to our work, but their work concentrates at Computer Go, allowing considerable amount of time and computational resource for pre-game training and simulation between steps. Since agents in the Planning track of GVG-AI Frameworks are subject to real-time gameplay and short simulation time, techniques in AlphaGo and temporal-difference search are not directly applicable in our case.

### III. PROPOSED METHOD

Our proposed method is based on KB Fast-Evo MCTS and improves the original method by four major modifications: the use of position-based features, the use of Q-Learning for rollout bias training, a new knowledge-based reward formula, and incorporation of Open Loop structure. Each of our

proposed improvements is discussed in details in the following. Furthermore, the effectiveness of each improvement is evaluated in Section IV.

#### A. Position-based Feature

In KB Fast-Evo MCTS, the Euclidean distances between the avatar and other objects are extracted as features to represent a game state. Even though the Euclidean distance is easy to calculate, using it as a feature oversimplifies the game state, because the distance ignores obstacles in the game map, and neglects the relative position of the object. An object at the left of the avatar and another object at the right of the avatar may have the same Euclidean distance, but their positions are different. Nonetheless, using the Euclidean distance feature omits the relative positions and directions of objects, rendering the feature values not representative of the actual game state.

In order to overcome such weakness, we propose the use of relative coordinates feature. In each game state, the 8 grid cells surrounding the avatar are scanned and for each object existing in these cells, the object type  $t$  and its coordinates relative to the avatar  $(x, y)$  are extracted as features. The possible values of  $x$  and  $y$  are 1, 0 and -1. Using Fig. 1. as an example, assuming the wall sprite is type 0, the extracted feature list will be  $[(\text{type}:0, x:-1, y:0), (\text{type}:0, x:-1, y:1)]$ . The weight matrix is adjusted to fit the new feature set. In the original KB Fast-Evo MCTS, the weight matrix is a 2D matrix with size  $A \times N$ , where  $A$  and  $N$  are the number of available actions and the number of object types, respectively. Using positions as features, the weight matrix is a 3D matrix with size  $A \times N \times P$ , where  $P$  is the total number of possible positions (i.e., 8).

Since the representation of features has changed, so should the formula for calculating the weight of each action. As shown in formula (2.4) in the previous section, KB Fast-Evo MCTS computes the weight of each action by multiplying the corresponding weight value by the feature value (i.e., the Euclidean distance of the object). Using the proposed position-based features, for each available action, the sum of weights of all features in the extracted feature set  $F$  is used as the weight of action  $i$ ,  $a_i$ , as shown in the following formula:

$$a_i = \sum_{j=1}^F w_{ij} \quad (3.1)$$

In contrast to formula (2.4), the above formula does not include the feature value. This is because features are defined as item type and relative position. For every set of action  $i$  and feature  $j$ , a weight value is defined and evolved separately. Since feature (i.e., the position of a sprite) is already reflected in the specification of  $j$  in  $w_{ij}$ , there is no need to include another feature value in the formula.

In our implementation, the size of extracted grid is set to  $3 \times 3$ , but it is possible to increase the size (e.g.,  $4 \times 4$ ,  $5 \times 5$ ) to extract more information from the state. However, in our experiment, increasing the grid size did not improve the performance significantly, and may run the risk of time overspent and running out of memory (since more information has to be extracted and stored in each step). Therefore, the grid size is set to  $3 \times 3$  in our entry.

### B. Q-Learning for Rollout Bias Training

#### Algorithm 1: Q-Learning Biased Rollout

```

1. while rollout not finished do
2.    $F \leftarrow \text{extractFeatures}(s)$ 
3.    $a \leftarrow \text{selectAction}(F, Q)$ 
4.    $s' \leftarrow \text{advanceStep}(s, a)$ 
5.    $r \leftarrow \text{getReward}(s, s')$ 
6.    $F' \leftarrow \text{extractFeatures}(s')$ 
7.   store transition  $(F, a, r, F')$  in  $D$ 
8.   sample a mini-batch of transitions,  $T$ , from  $D$  randomly
9.   foreach  $t_i$  in  $T$ 
10.     $a' \leftarrow \text{greedy}(t_i.F', Q)$ 
11.    foreach  $f_j$  in  $t_i.F$ 
12.       $\text{maxReward} \leftarrow 0$ 
13.      foreach  $f_j'$  in  $t_i.F'$ 
14.         $\text{maxReward} \leftarrow \hat{Q}(f_j', a') + \text{maxReward}$ 
15.       $\Delta Q \leftarrow \alpha(t_i.r + \gamma \text{maxReward} - Q(f_j, t_i.a))$ 
16.       $Q(f_j, t_i.a) \leftarrow Q(f_j, t_i.a) + \Delta Q$ 
17.   Every  $C$  steps reset  $\hat{Q} = Q$ 
18. end while

```

KB Fast-Evo MCTS uses  $(1 + 1)$  ES for training the weight matrix, but the learning mechanism has several drawbacks. As criticized by Eeden [9], if the roller achieved a high score in a rollout at the early stage of the game, the learning mechanism will suffer from early convergence. This is because  $(1 + 1)$  ES algorithm decreases its noise factor whenever the evolution is not successful (i.e., the fitness value of the current individual did not surpass the fitness value of the best individual), if the weight matrix fails to evolve consecutively, the noise factor may fall too low, rendering mutation meaningless.

In our proposed method, we attempted to apply Q-Learning in rollout bias training. The pseudo-code of the proposed learning mechanism is depicted by Algorithm 1. The  $Q$  in the algorithm corresponds to the weight matrix  $W$  in the previous section. In the  $\text{selectAction}(F, Q)$  function, after the weight of each action is computed, the bias of each action  $i$  is determined by a logistic function:

$$B(i) = \frac{1}{1 + e^{a_i}} \quad (3.2)$$

The probability of selecting each action  $i$  is then computed by

$$P(i) = \frac{B(i)}{\sum_{j=1}^A B(j)} \quad (3.3)$$

The learning mechanism is based on the growing-batch approach [16], which has also been applied by Mnih et al. [17]. We used the following parameters: learning rate  $\alpha = 0.1$ , discounting factor  $\gamma = 0.1$ , size of  $D = 500$ , mini-batch size = 10 and  $C = 500$ . The last one,  $C$ , appears on line 17 of Algorithm 1 and affects how frequent the values of  $\hat{Q}$  are updated. The values of the parameters were decided empirically. Unlike  $(1 + 1)$  ES, our proposed method updates weight at every step in the rollout, and the score change of each action is applied as a reward in the learning. Only weight values that correspond to the extracted features are updated in

the weight matrix.

### C. New Knowledge-based Reward Formula

In GVG-AI Competition, the game-playing agent must finish its search and decide the next action within 40 ms, thus many UCT-based methods set a maximum rollout depth to limit the time spent in the tree search. Hence, most rollouts cannot reach the end of the game or even receive any score change, and such rollouts contribute little to the evaluation of tree nodes. In KB Fast-Evo MCTS, when the simulated game has not reached an end state and has no score change at the end of the rollout, the increase of knowledge and change in distance are used as the reward instead.

KB Fast-Evo MCTS's reward function (2.3) poses a number of problems. As Eeden [9] pointed out, the knowledge change adds little information, and since in our proposed Q-Learning mechanism, the reward is calculated in each rollout step, the change in distance and knowledge is insignificant. When we tested the Q-Learning mechanism with the reward function, we found that the values of  $\Delta Z$  and  $\Delta D$  were negligible in most cases. Furthermore,  $\Delta D$  is calculated based on the Euclidean distance, which neglects obstacles, in KB Fast-Evo MCTS. The following reward function is, therefore, proposed to replace the original formula:

$$r(s, a) = \begin{cases} 1000 : \text{if avatar wins in state } s \\ -1000 : \text{if avatar loses in state } s \\ 10 \times \Delta R + \text{Pheromones}(s) + 0.1 \times \sum_i \Delta D_i \end{cases} \quad (3.4)$$

In the above formula,  $\Delta R$  represents the score difference between the current state and the previous;  $\text{Pheromones}(s)$  is adopted directly from [10];  $\Delta D_i$  is the change of distance between the avatar and the beneficial or unknown sprite type  $i$ . For each beneficial or unknown sprite type  $i$ , only the closest object to the avatar is considered, and the change of distance is calculated using the A\* algorithm in terms of the number of tiles. Beneficial sprites are identified by knowledge acquired in rollouts, similar to KB Fast-Evo MCTS.

The A\* algorithm being used is similar to the one introduced in [18]. The nature of each type of sprite is learned through simulations during Monte-Carlo rollouts. While information on the game rules is not available, the agent can simulate a collision with different sprites using the forward model given by the game engine, and examine the effect of the collision. By doing so, the agent can identify which sprite affects the score and which sprite can be traversed. Such information is applied in the A\* algorithm to select beneficial sprites as targets and identify the shortest path between the avatar and any given object. After every simulated step, A\* algorithm is applied to the new game state in order to compute the distance change. While there may be concerns that using A\* algorithm so frequently may cause severe overhead, our experiment, which is described in Section IV, has shown that A\* algorithm combined with the open-loop structure vastly improves the performance of our method.

### D. Open Loop Structure

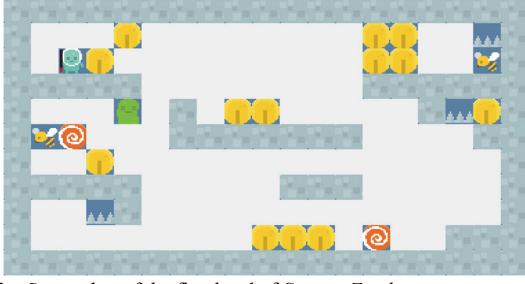


Fig. 2 Screenshot of the first level of *Survive Zombies*

Apart from the aforementioned modification, the Open Loop Search technique proposed in [10] is integrated in our entry as well. Each tree node in our entry stores an action and its statistics, but not the game state; and statistics data, namely the average score and the highest score, of every tree node is retained after each game step for the search at the next game step. At the end of a search, for each child node of the root, the weighted sum of the average score and the highest score is computed, and the action with the highest sum is performed in the game space.

#### IV. EVALUATION

In order to analyze and evaluate of our proposed method, several tests have been performed. First, using the first level of *Survive Zombies*, a comparison of (1 + 1) ES and Q-Learning was undertaken. Then, a series of tournaments were performed to evaluate the performance of our proposed method.

##### A. Analysis of Learning Process in (1+1) ES and Q-Learning

Jim van Eeden has criticized that (1 + 1) ES in KB Fast-Evo MCTS converged too quickly [9]. This criticism is illustrated by Fig. 3, which shows the normalized weight values for the “honey” object trained by (1 + 1) ES in the first level of *Survive Zombies*. Depicted in Fig. 2, *Survive Zombies* is a relatively simple game where the avatar has to collect as many “honey” object as possible while evading the zombies. The latest version of the *Survive Zombies* game was used here. The avatar gains a score and a health point for every honey

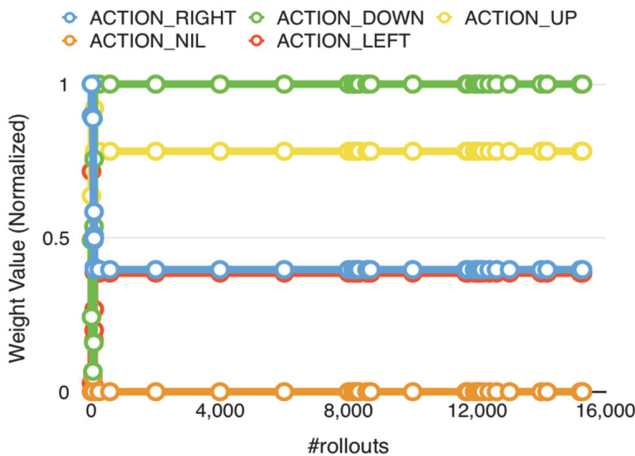


Fig. 3. Normalized weight values for the honey object in the first level of *Survive Zombies* trained by (1 + 1) ES

collected. The weight value of each action  $i$  for the honey object  $w_{i,honey}$  was trained using (1 + 1) ES algorithm of KB Fast-Evo MCTS over the whole game, and the values were then normalized and plotted. As shown in Fig. 3, the weight values converged at around 90<sup>th</sup> rollout, and they remained largely unchanged for the rest of the game. The convergence occurred very early, considering that more than 15,000 rollouts were performed throughout the game. The early convergence happened because the noise factor had fallen to an extremely low value, after which the weight matrix failed to evolve in several iterations.

The early convergence alludes to some concerns. The early convergence means that learning after the first hundred rollouts was ineffective and could not contribute any meaningful change to the weight values. This is particularly problematic, considering that *Survive Zombies*, like most games in GVG-AI Framework, is dynamic and fast-paced. In this game, the avatar is supposed to move quickly in order to collect the yellow honey produced by bees, which move freely around the stage. The relative position between the avatar and the nearest honey changes vigorously throughout the game, but the change in game state could not be reflected in the weight matrix, due to early convergence. While convergence may be desirable in static problem, in the dynamic environment of real-time games, early convergence might cause inflexibility and irresponsiveness.

Fig. 4 shows how Q values evolved during a *Survive Zombie* game. The weight value of each action  $i$  for the feature [type: honey, x:0, y:1] was evolved using Algorithm 1. In this feature, a honey is positioned right under the avatar; thus it is logical that ACTION\_DOWN should attain the highest weight value, since moving downwards will immediately yield a reward. As such, the weight of ACTION\_DOWN remained the highest in most of the game, but the weight values for other actions also show a lot of adjustments throughout the game. The weight values changed drastically near the end, because as the game approached the end state, game over occurred much more frequently in the rollout, therefore introducing a lot of reward with extreme value to the weight matrix.

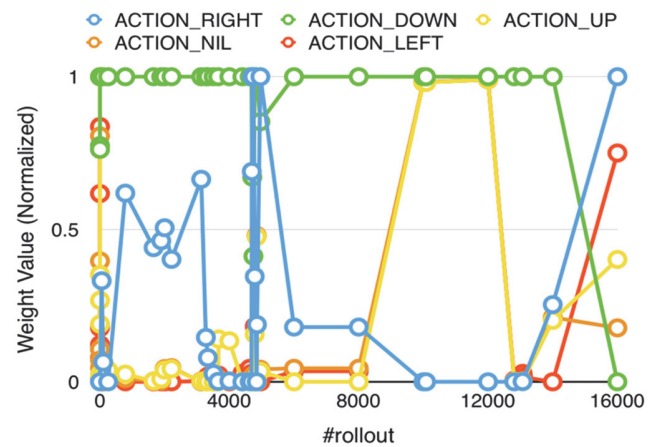


Fig. 4. Normalized weight values for the feature (type: honey, x: 0, y: 1) in the first level of *Survive Zombies* trained by Q-Learning

TABLE I. F1 SCORES FOR EACH LEARNING MECHANISM AND FEATURE PAIR

	ES-D	ES-P	QL-D	QL-P
Training Set 1	154	170	184	192
Training Set 2	189	192	129	193
Training Set 3	173	190	171	192
Training Set 4	179	189	142	199
Training Set 5	180	202	145	203
Training Set 6	213	200	204	215
Total	1088	1143	975	1194

### B. Evaluation of Q-Learning and Position-based Features

The performance of Q-Learning and position-based features was tested. At the time of writing, there are 6 training game sets available in the GVG-AI Framework, and all of them were used in our experiments. The latest version of the platform and game level files as of 9<sup>th</sup> April 2016 were downloaded from the official GitHub<sup>1</sup>. Each game set contains 10 games, each game has 5 levels, and each level is played 5 times by each of the controllers being tested. The evaluation method follows the GVG-AI Competition rules<sup>2</sup>. For each game, each controller is ranked according to its win rate (if two or more controllers achieve the same win rate for the same game, the average scores are compared instead). Based on the ranking, each controller is awarded points according to the F1 Score System. The overall ranking is determined by the total points earned by each controller. As the competition server is under maintenance at the time of writing, the tests were performed on our personal computer (Mac OS X, 3.5 GHz Intel Core i5, 8GB Memory). There are four combinations being tested:

- ES-D: Using (1 + 1) ES as the learning mechanism, Euclidean distance as the feature;
- ES-P: Using (1 + 1) ES as the learning mechanism, position as the feature;
- QL-D: Using Q-Learning as the learning mechanism, Euclidean distance as the feature; and
- QL-P: Using Q-Learning as learning mechanism, position as the feature.

The implementation of each controller was based on the Fast-Evo MCTS suggested in [1], and all controllers used score change as a rollout reward (without the knowledge-based scoring function used in [1]). The test results are shown in Table I. Table I shows only the final F1 score of each controller. The reader is referred to our homepage<sup>3</sup> for the detailed results and replay log of every controller for each game. As shown in the results, QL-P gained the highest F1 score in the overall ranking. Thus, we conclude that Q-Learning and position-based feature is the best learning mechanism and feature combination among the methods being tested.

### C. Evaluation of New Reward Formula and Open Loop

Next, the effect of the new reward formula and open loop structure was evaluated. Four combinations were tested:

TABLE II. F1 SCORES FOR EACH LOOP STRUCTURE AND REWARD FORMULA PAIR

	CL-Raw	OL-Raw	CL-New	PB-RL MCTS
Training Set 1	168	142	141	237
Training Set 2	160	150	147	243
Training Set 3	167	142	189	226
Training Set 4	162	135	199	213
Training Set 5	192	148	173	217
Training Set 6	190	183	180	236
Total	1039	900	1029	1372

- CL-Raw: Using close loop with the score change reward, which is the same as QL-P in Section IV B;
- OL-Raw: Using open loop with the score change reward;
- CL-New: Using close loop with the new formula reward; and
- PB-RL MCTS: Using open loop with the new formula reward. (our final proposed method)

All controllers used Q-Learning for rollout bias training and position as the feature, as it is proven in the previous test that this combination begets the best performance. The results of QL-P in the last section were reused as CL-Raw here. The test results are depicted in Table II. Detail breakdown of the test results and replay log is available on our homepage<sup>3</sup>. An interesting behavior can be spotted in Table II: while open loop and the new reward formula did not improve performance when applied separately, the two modifications offer a significant boost to the performance when combined together. Clearly, the open loop structure and the new reward formula have a strong synergy effect, and achieved the best performance in all game sets among the four combinations.

### D. Evaluation of the Final Proposed Method

After confirming that the combination of our proposed Q-Learning algorithm, position feature, new reward formula and open loop structure indeed provides the best performance among all other combinations, a set of final tests was undertaken to compare our proposed method with other existing MCTS variants, including:

- sampleOLMCTS: a benchmark controller provided by the GVG-AI Framework, which is an open loop version of the UCT algorithm;
- KB Fast-Evo MCTS: the MCTS variant proposed in [1], which applied rollout bias and a knowledge-based reward formula to improve the performance of MCTS, the detail of the algorithm is explained in Section II;
- AIJim: an upgraded version of KB Fast-Evo MCTS, proposed in [8];
- Frydenberg: a modified UCT program proposed in [7]. UCT + MixMax + Reverse Penalty was used for Training Set 1 as it is reported that this combination performs the best for the set. For other game sets, UCT + Reverse Penalty was used in our test; and

1. <https://github.com/EssexUniversityMCTS/gvgai>

2. <http://gvgai.net/evaluation.php>

3. <http://www.ice.ci.ritsumeai.ac.jp/~ruck/downloads.html>

- TeamTopBug (MCTS): the open loop search technique proposed in [9]. Since the purpose of our experiment is to compare multiple MCTS variants, the MCTS version of the method was used.

The test results of the PB-RL MCTS controller in Section IV C were reused in this section. Since there are 60 games in total, each controller played each level 5 times, and there were 6 controllers (5 MCTS variants and our proposed method) being tested; a total of  $60 \times 5 \times 5 \times 6 = 9000$  game levels results were considered for this test, including the reused results for our proposed method. The test results are presented in Appendix. Unfortunately, Frydenberg did not perform well in our test environment, while AIJim achieved good results, especially in deterministic games such as *sokoban*, *catapults* and *escape*, and outperformed KB Fast-Evo MCTS overall. TeamTopBug performed well and stably, and came out on top in Training Set 5 and 6. Nonetheless, our proposed method won in all of the rest training sets, and is the overall champion in our test.

Compared with other MCTS variants, the proposed method achieved the highest average win rate in all game sets but Training Set 5; and among all game sets, the proposed method earned the highest win rate in Training Set 1, followed by Training Set 2. Games in Training Set 1 and 2 are relatively simple, and can be solved easily using MCTS and pathfinding algorithms, i.e., *Aliens*, *Butterflies*, *Portals*, *Zelda*, *Camel Race*, *Firestorms*, etc. Since the proposed method is based on MCTS and utilizes pathfinding in its reward formula, it can achieve good performance in those games. However, games in other game sets are far more complicated, and involve a lot of puzzle game elements. In particular, games in Training Set 3 and 6 are mainly puzzle games which are difficult even for human players, and most of the controllers being tested struggled at those games. Nonetheless, the proposed method achieved a higher average win rate than other competitors in those game sets as well. Overall, the proposed method performed well across different types of games, but was relatively weak in puzzle games like *WaitForBreakfast*, *Race Bet 2* and *Cook me Pasta*, resulting in less satisfying performance in Training Set 5 and 6.

## V. CONCLUSIONS AND FUTURE WORK

This paper proposed the use of Q-Learning as learning mechanism for training rollout bias in GVGP, the use of relative position as feature, and a new knowledge-based reward formula for evaluating rollout actions. The proposed method was tested against 5 existing MCTS variants in 60 games, and the proposed method achieved the best overall performance. As our future work, applying the Q-Learning algorithm and position feature with Genetic Algorithm or other search algorithm can be interesting future work.

## ACKNOWLEDGEMENT

We would like to express our gratitude to Diego Perez-

Liebana, Jim van Eeden and Prof Julian Togelius for kindly providing the source code of their controllers, which were used in our experiments.

## REFERENCES

- [1] D. Perez, S. Samothrakis and S. Lucas, "Knowledge-based fast evolutionary MCTS for general video game playing," in *Proceedings of 2014 IEEE Conference on Computational Intelligence and Games*, Dortmund, pp. 1-8, 2014.
- [2] L. Kocsis and C. Szepesvari. "Bandit based monte-carlo planning," in *Proceedings of Machine Learning: ECML 2006*, Springer-Verlag Berlin Heidelberg, pp. 282-293, 2006.
- [3] Y. Naddaf. "Game-independent AI agents for playing Atari 2600 console games," M.S. thesis, University of Alberta, Canada, 2010.
- [4] M. G. Bellemare, J. Veness, and M. Bowling. "Investigating contingency awareness using Atari 2600 games," in *Proceedings of the 26th Conference on Artificial Intelligence*, pp. 864-871, 2012.
- [5] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. "The Arcade Learning Environment: an evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253-279, Jul. 2012.
- [6] D. Perez, S. Samothrakis, J. Togelius, T. Schaul, and S. Lucas. "General Video Game AI: competition, challenges and opportunities," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pp. 4335-4337, 2016.
- [7] P. Hyunsoo, and K. J. Kim. "MCTS with influence map for general video game playing," in *Proceedings of 2015 IEEE Conference on Computational Intelligence and Games*, pp. 534-535, 2015.
- [8] F. Frydenberg, K. R. Andersen, S. Risi, and J. Togelius. "Investigating MCTS modifications in general video game playing," in *Proceedings of 2015 IEEE Computational Intelligence and Games*, pp. 107-113, 2015.
- [9] J. G. van Eeden. "Analysing and improving The Knowledge-based Fast Evolutionary MCTS algorithm," Master thesis, Utrecht University, Netherlands, 2015.
- [10] D. Perez, J. Dieskau, M. Hunermund, S. Mostaghim, and S. Lucas. "Open loop search for general video game playing," in *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*, pp. 337-344, 2015.
- [11] V. Mnih, K. Kavukcuoglu, D. Silver et al. "Playing Atari with deep reinforcement learning," in *Proceedings of the NIPS Deep Learning Workshop*, 2013.
- [12] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone. "A neuroevolution approach to general Atari game playing," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 4, pp. 355-366, Mar. 2014.
- [13] T. Vodopivec and B. Šter. "Enhancing upper confidence bounds for trees with temporal difference values," in *Proceedings of 2014 IEEE Conference on Computational Intelligence and Games*, pp. 1-8, 2014.
- [14] D. Silver, A. Huang, et al. "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484-489, Jan. 2016.
- [15] D. Silver. "Reinforcement learning and simulation-based search," Doctor of philosophy thesis, University of Alberta, Canada, 2009.
- [16] L. Sascha, T. Gabel, and M. Riedmiller. "Batch reinforcement learning," in *Reinforcement Learning*, vol. 12. M. Wiering and M. van Otterlo, Ed. Springer Berlin Heidelberg, pp. 45-73, 2012.
- [17] V. Mnih, K. Kavukcuoglu, D. Silver, et al. "Human Level Control Through Deep Reinforcement Learning," *Nature*, vol. 518, no. 7540, pp. 529-533, Feb. 2015.
- [18] C. Y. Chu, H. Hashizume, Z. Guo, T. Harada and R. Thawonmas, "Combining pathfinding algorithm with Knowledge-based Monte-Carlo tree search in general video game playing," in *Proceedings of 2015 IEEE Conference on Computational Intelligence and Games*, pp. 523-529, 2015.



APPENDIX  
TESTS RESULTS OF PROPOSED METHOD AND  
EXISTING METHODS

TABLE A I. TRAINING SET 1 (CIG2014TRAININGSETGAMES)

	sampleOL MCTS		KB Fast-Evo MCTS		AIJim		Frydenberg		TeamTopbug (MCTS)		PB-RL MCTS	
	Avg	Win%	Avg	Win%	Avg	Win%	Avg	Win%	Avg	Win%	Avg	Win%
aliens	64.36	1	67	1	66.52	0.92	52.84	0.48	69	1	66.8	1
boulder dash	10.84	0.12	155	0.04	146.52	0.08	9.04	0.12	0	0	452.24	0.04
butter- flies	28.4	0.92	31.84	1	30.32	0.92	31.28	0.92	26.32	1	27.04	1
chase	1.28	0	3.32	0.04	3.56	0.04	1.48	0	4.52	0.36	6.88	0.68
frogs	0.16	0.16	0.44	0.44	0.52	0.52	0	0	0.8	0.8	0.84	0.84
missile command	2.52	0.52	1.56	0.28	6.04	0.8	-1.16	0.2	-1.6	0.2	6.68	0.88
portals	0.08	0.08	0.12	0.12	0.32	0.32	0.08	0.08	0.88	0.88	0.8	0.8
sokoban	0.96	0.12	1.08	0.2	1.68	0.36	0.32	0	1.4	0.2	1	0.2
survive zombies	-5.92	0.12	-3.4	0.36	-6.32	0.04	-9.24	0	-9.04	0.16	-11.4	0.12
zelda	4.12	0.2	3.64	0.08	4.2	0.4	1	0.16	2.08	0.04	7	1
Avg. Win%	0.324		0.356		0.44		0.196		0.464		0.656	

TABLE A II. TRAINING SET 2 (CIG2014VALIDATIONSETGAMES)

	sampleOL MCTS		KB Fast-Evo MCTS		AIJim		Frydenberg		TeamTopbug (MCTS)		PB-RL MCTS	
	Avg	Win%	Avg	Win%	Avg	Win%	Avg	Win%	Avg	Win%	Avg	Win%
camel- race	-0.52	0.12	-0.6	0.08	0.2	0.6	-0.04	0.48	1	1	1	1
digdug	9.76	0	9	0	9.28	0	2	0	2.44	0	33.6	0
fire- storms	-0.52	0.2	-0.2	0.12	-1.2	0.24	-1.2	0.16	1	1	1	1
infection	9.96	0.96	9.6	0.96	17.8	1	-1	0.96	2.84	0.88	22.88	1
fire- caster	7.88	0	372.88	0	12.32	0	9.36	0	2.2	0	406.28	0
overload	290	0.12	419.32	0.04	840	0.28	32.84	0.2	2.2	0	1588.24	0.28
pacman	62.44	0	69.88	0	339.76	0	225.64	0	245.6	0.24	631.32	0.16
seaquest	1342	0.52	2490.52	0.96	2002.08	0.56	683.04	0	1120.96	0.64	4604.32	0.68
whack- amole	24.08	1	24.32	0.76	20.84	0.4	12.4	0.44	46.56	1	39.24	0.8
ego- mania	5.56	0	3.88	0.08	1.04	0	5.76	0.08	0.44	0	71.84	0.68
Avg. Win%	0.292		0.3		0.308		0.232		0.476		0.56	

TABLE A III. TRAINING SET 3 (CIG2015TRAININGSETGAMES)

	sampleOL MCTS		KB Fast-Evo MCTS		AIJim		Frydenberg		TeamTopbug (MCTS)		PB-RL MCTS	
	Avg	Win%	Avg	Win%	Avg	Win%	Avg	Win%	Avg	Win%	Avg	Win%
bait	0.96	0.08	2.48	0	4.16	0.08	1.16	0	2.8	0	2.28	0.04
bolo adventures	0.84	0	0.12	0	0	0	0.52	0	0	0	1.2	0.2
brainman	23.96	0.08	25.12	0.04	23.84	0.04	23.44	0	21.2	0	24.8	0
chips challenge	2.84	0	2.56	0	3.68	0	1.56	0	1.96	0	4.88	0
modality	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
painter	337.68	0.76	109.76	1	784	0.2	531.68	0.2	27.56	1	34.56	1
real portals	1.28	0	0.12	0	5.44	0	1.28	0	0	0	1.84	0
real sokoban	0.28	0	1.2	0	1.32	0	0.16	0	1.2	0	1.2	0
the citadel	1.12	0.04	1.84	0.12	1.04	0	0.72	0.04	3.6	0.4	1.72	0.08
zen puzzle	23.92	0.12	27.6	0.44	20.68	0.2	21.8	0	24.44	0	23.28	0.2
Avg. Win%	0.128		0.18		0.072		0.044		0.16		0.172	

TABLE A IV. TRAINING SET 4 (CIG2014TESTSETGAMES)

	sampleOL MCTS		KB Fast-Evo MCTS		AIJim		Frydenberg		TeamTopbug (MCTS)		PB-RL MCTS	
	Avg	Win%	Avg	Win%	Avg	Win%	Avg	Win%	Avg	Win%	Avg	Win%
rogue like	3.92	0	0.52	0	6.56	0.12	2.68	0	0.48	0	14.8	0.08
surround	1	1	1	1	1	1	1.96	1	1	1	1	1
catapults	2.56	0	2.92	0.12	2.96	0.16	1.72	0	0.68	0	2.36	0
plants	17.32	0.08	5.36	0	1.6	0	4.08	0.04	1	0	-0.6	0.04
plaque attack	34.88	0.68	8.56	0.24	46.36	0.96	5.24	0.36	-19.2	0	45.84	1
jaws	112.48	0.68	1149.28	0.76	201.72	0	4.44	0	0.88	0.16	2144.04	0.4
labyrinth	0.04	0.04	0.24	0.24	0.52	0.52	0.04	0.16	0.8	0.8	1	1
boulder chase	248.04	0.12	9	0	15.8	0	9	0	8.88	0	663.44	0.12
escape	0	0	0	0	0.44	0.48	-0.04	0.16	0.4	0.4	0.08	0.08
lemmings	-22.4	0	-1.68	0	-1.92	0	-31.68	0	-0.04	0	-0.08	0
Avg. Win%	0.26		0.236		0.324		0.172		0.236		0.372	

TABLE A V. TRAINING SET 5 (GEC2015TESTSETGAMES)

	sampleOL MCTS		KB Fast-Evo MCTS		AIJim		Frydenberg		TeamTopbug (MCTS)		PB-RL MCTS	
	Avg	Win%	Avg	Win%	Avg	Win%	Avg	Win%	Avg	Win%	Avg	Win%
solarfox	9.28	0	5.24	0	6.96	0	8.08	0	7.28	0	5.8	0
defender	-24.28	0.64	-16.08	0.8	-45.08	0.44	-30.92	0.6	82.44	0.88	37.52	0.96
enemy citadel	1.04	0.08	1.12	0.04	0.6	0	0.48	0	1.48	0	1.6	0.04
crossfire	0.16	0.04	0.96	0.2	-1	0	-0.52	0.08	5	1	4.28	0.88
lasers	0	0	0	0	0	0	0	0	0	0	0	0
sheriff	6.48	1	8.44	0.84	7.28	0.72	0.44	0.08	9.16	1	7.76	0.8
chopper	-4.84	0.2	8.16	0.72	4.44	0.44	-5.2	0	-1.28	0.08	-7.6	0
superman	5.8	0	9.32	0	118.76	0	188.32	0.04	1153.64	0	592.8	0.12
waitfor breakfast	0.04	0.04	0.32	0.32	0.4	0.4	0.56	0.56	0.52	0.52	0.28	0.28
cakybaky	178.2	0	593.48	0	539.92	0	9.12	0	1366.36	0.04	1088.04	0
Avg. Win%	0.2		0.292		0.2		0.136		0.352		0.308	

TABLE A VI. TRAINING SET 6 (CEEC2015VALIDATIONSETGAMES)

	sampleOL MCTS		KB Fast-Evo MCTS		AIJim		Frydenberg		TeamTopbug (MCTS)		PB-RL MCTS	
	Avg	Win%	Avg	Win%	Avg	Win%	Avg	Win%	Avg	Win%	Avg	Win%
lasers2	0	0	0	0	0	0	0	0	0	0	0	0
hungry birds	0	0	0	0	0	0	0	0	40	0.4	80	0.8
cookme pasta	1.6	0	0.68	0	2.32	0	2.2	0	2.4	0	0.96	0
factory manager	-1.24	0.92	-0.2	0.96	-1.72	0.92	-0.56	0.92	1	1	1	1
racebet2	0.64	0.64	0.24	0.24	0.64	0.64	0.8	0.8	0.4	0.4	0.28	0.28
inter- section	1.32	1	1	1	-0.04	0.96	-1.72	0.88	123	1	108.2	1
black smoke	1.4	0	0.64	0	16.12	0.12	0.88	0	2.76	0	5.28	0
iceandfire	1.56	0	1.96	0	8.44	0.16	2.4	0	15.8	0.8	14.36	0.76
gymkhana	1.8	0	1.48	0	1.16	0	0.96	0	0.2	0	2.72	0.08
tercio	0	0	0	0	0	0	0	0	0	0	0	0
Avg. Win%	0.256		0.22		0.28		0.26		0.36		0.392	

TABLE A VII. OVERALL PERFORMANCE

	sampleOL MCTS	KB Fast- Evo MCTS	AIJim	Frydenberg	TeamTopbug (MCTS)	PB-RL MCTS
Training Set 1	128	151	151	104	161	187
Training Set 2	125	130	141	115	157	226
Training Set 3	162	173	176	115	141	189
Training Set 4	152	142	173	130	131	183
Training Set 5	153	162	130	142	189	166
Training Set 6	158	141	171	157	203	199
Total	878	899	942	763	982	1150



# Intrinsically Motivated Reinforcement Learning: A Promising Framework for Procedural Content Generation

Noor Shaker, *Member, IEEE*  
Aalborg University  
Copenhagen, Denmark  
Email: noor.shaker@gmail.com

**Abstract**—So far, Evolutionary Algorithms (EA) have been the dominant paradigm for Procedural Content Generation (PCG). While we believe the field has achieved a remarkable success, we claim that there is a wide window for improvement. The field of machine learning has an abundance of methods that promise solutions to some aspects of PCG that are still under-researched. In this paper, we advocate the use of Intrinsically motivated reinforcement learning for content generation. A class of methods that thrive for knowledge for its own sake rather than as a step towards finding a solution. We argue that this approach promises solutions to some of the well-known problems in PCG: (1) searching for novelty and diversity can be easily incorporated as an intrinsic reward, (2) improving models of player experience and generation of adapted content can be done simultaneously through combining extrinsic and intrinsic rewards, and (3) mix-initiative design tools can incorporate more knowledge about the designer and her preferences and ultimately provide better assistance. We demonstrate our arguments and discuss the challenges that face the proposed approach.

## I. INTRODUCTION

Psychologists distinguish between extrinsic and intrinsic motivations. While the former user a form of explicit reward to motivate the learner to move towards a predefined goal, the agent in the latter is driven by its own curiosity to discover its world while enjoying the process of information gathering. Intrinsic motivation, according to a number of studies, “leads organisms to engage in exploration, play, and other behavior driven by curiosity in the absence of explicit reward. These activities favour the development of broad competence rather than being directed to more externally-directed goals” [1].

Researchers have proposed the framework of *Intrinsically Motivated Reinforcement Learning* (IMRL) to explore behaviours guided by intrinsic motivations as a form of reward [1]. IMRL has a strong motivation in robotics where robots might be placed in an unknown environment and they are expected to learn their own world model through exploration and adaptation to new observations.

Computer games provide a unique testbed for several AI methods due to their rich environment and multifaceted nature that makes them well suited for wide range of studies in machine learning [2], computational creativity [3] and general AI [4]. We argue in this paper that intrinsically motivated reinforcement learning is a promising framework for procedural content generation and is a viable addition to the spectrum of methods investigated in the literature [5]. We believe the framework is well suited, yet not explored, in the Procedural Content Generation (PCG) paradigm. The IMRL framework promises solutions to the three main categories under which most of the work in the field of PCG revolves: (1) creating new, yet interesting and novel content, (2) continuous adaptation to players’ need and (3) supporting and inspiring game designers during the game design process.

In this work we investigate the applicability, practicality and advantages of the IMRL framework within the computer games domain. In particular, in this paper we formalise the framework of IMRL in the PCG domain and demonstrate its advantages in (1) exploring meaningful space of content through searching for diversity, prioritising quality, improvising content and encouraging rapid feedback; (2) establishing an efficient incorporation of players in the content generation process through simultaneous update of models of player experience and search for adapted content; and (3) providing effective collaborators for game designers through modelling her preferences and learning from the feedback. We finally discuss a number of key challenges that one should expect when implementing the IMRL framework.

After all, EA has so far been the dominant approach to PCG. What we propose is the use of a method that has been widely implemented in other fields yet minimally explored for PCG. The reader is advised not to view this paper as an EA vs. IMRL discussion. Rather, we are interested in highlighting the potential of a well established method in machine learning field and presenting an argument that encourages researchers to implement, test and compare the two paradigms. We expect that in some cases, the two methods would provide complementary strengths, so hybrid approaches are advisable.

## II. INTRINSIC MOTIVATION

The idea of intrinsic motivation was pioneered by Barto, Singh, & Chentanez [1] and has originally departed from studies on animal behaviour. The studies showed that dopamine plays a critical role both in the extrinsic motivational control guiding the behaviour towards a goal, and in the intrinsic motivational control associated with novelty and

exploration. The analysis showed activations of dopamine associated with salient stimuli which extinguishes as the stimuli become familiar [6], [7]. These findings are the key building blocks of the framework of *Intrinsically Motivated Reinforcement Learning* (IMRL) [1].

Due to their curious nature, intrinsically motivated learners enjoy more flexibility when faced with new problems or when the environment is constantly changing [1]. Such properties led to novel solutions in the field of robotics and machines that outperform traditional active learning methods [8].

To better understand intrinsic motivation, one could observe the behaviour of children and young infants and how they learn skills. Children continuously pursue exploration of the affordance of their world through performing primitive actions such as throwing, biting squashing or even shouting at new objects they encounter. As Wardle put it [9]: “Children do not play for a reward-praise, money, or food. They play because they like it”. An intrinsically motivated system aims to imitate constructive play: examining objects in the environment, discovering what/how they work and building knowledge and skills [10].

Formally, intrinsic motivation was defined according to Ryan and Deci [11] (pp. 56) as:

“The doing of an activity for its inherent satisfaction rather than for some separable consequence. When intrinsically motivated, a person is moved to act for the fun or challenge entailed rather than because of external products, pressures, or rewards.”

### III. INTRINSICALLY MOTIVATED REINFORCEMENT LEARNING

Reinforcement learning (RL) addresses the general problem of an agent learning to approximate an optimal behavioural policy through interactions with its environment in a trial-and-error process. The agent must learn a mapping that assign *rewards* to actions. Through its lifetime, the agent learns how to maximise the rewards it receives for performing actions. This reward signal is the only learning feedback obtained from the environment. The mapping, *policy*, constructed determines how the agent behave in each possible state.

The most common representation of a RL agent is a Markov decision processes (MDPs). At each time step,  $t$ , the agent observes the environment state,  $s_t$ , and chooses to perform an action accordingly. The action,  $a$ , is chosen from a finite set of possible actions,  $A$ , and by performing the action, the environment state becomes  $s_{t+1}$  with probability  $P(s_{t+1}|s_t, a_t)$ . Accordingly, the agent receives a reward  $r(s_t, a_t)$  and the process repeats. The goal of the agent is to chose a sequence of actions so that it gathers as much reward as possible. Formally, the agent needs to learn a state-action policy  $\pi : S \rightarrow A$ . The *optimal policy*  $\pi^*$  is typically defined as the policy that maximises the cumulative reward over all states

$$\pi^* = \underset{\pi}{argmax} V^\pi(s), (\forall s) \quad (1)$$

where  $V^\pi(s)$  is the cumulative reward received from state  $s$  using policy  $\pi$ . There are different approaches to find the optimal policy,  $\pi^*$ , most of which involve search in the *policy space* or *value function space*. Search in the policy space attempts to optimise the parameter for a given policy parametrisation. Value function methods attempts to learn a value function,  $V^{\pi^*}$ , which returns the expected cumulative reward for the optimal policy from any given state [12]. There are different approaches to learning the policy and the value function and the most commonly used ones are Temporal Difference (to learn the value function) [13], Gradient-based methods [14], model-based methods [15] and dynamic programming and evolutionary algorithms (for learning the optimal policy) [12].

Recently, a novel framework for intrinsically motivated reinforcement learning, where search for information is a goal in itself, was proposed [1]. Studies of intrinsic motivations started in the field of psychology where researchers were interested in investigating the cognitive processes underlying intrinsic motivation [16], [11]. As this direction progresses and matures, it gathered increasing interest from researchers in developmental robotics where several computational models have been developed [17].

The goal of intrinsically motivated systems is to maximise their knowledge about the world through executing sequences of actions that leads to observations yielding maximal accurate world model [18]. To achieve this, Intrinsic Reward (IR) is given which is proportional, for instance, to the predictor’s surprise/ information gain given the history of observed world states. Intrinsic rewards can come from a number of sources including surprise [18], novelty [19], learning progress [20] or emotions [21].

### IV. CURIOSITY, NOVELTY AND SURPRISE

The literature identified a number of properties that contribute to intrinsic motivation. Such properties include *fun*, *challenge*, *knowledge acquisition*, *novelty*, or *curiosity*. Studies have shown that such abstract qualities are usually hard to define let alone to be computationally represented [18], [17]. In what follows we review previous attempts on quantifying such attributes so that they can be implemented within the IMRL framework.

*Artificial curiosity* (AC) [22] refers to the process of developing skills through play and is closely tied to intrinsic motivation [23]. Curiosity is “the drive to actively explore the interesting regions in search space that most improve the models predictions or explanations of what is going on in the world” [24]. AC was originally introduced for RL [22] where it has been used mainly for active learning [25] to explain patterns of human visual attention and to help understand abstract concepts such as beauty and creativity [18].

According to this theory, a curious agent needs two learning components: a general reinforcement learner and an internal world model (i.e., predictor of the agent’s world

given the growing history of perceptions and actions). The learning progress or expected improvement of the model becomes an intrinsic reward for the reinforcement learner. Hence, to achieve high intrinsic reward, the reinforcement learner is motivated to create new experiences such that the model progresses faster. In other words, the learner is regarded as being curious about its world [10].

Artificial curiosity couples RL with an intrinsic reward motivation. Many forms of such intrinsic reward for AC systems can be found in the literature. A function of the difference between the posterior and the prior knowledge about the world forms one example of IR of curiosity that has been widely used [26].

*Novelty* has been used frequently in the literature as an intrinsic motivation [1], [27]. Traditionally, a novel reward is given to an event,  $e$ , that is proportional to the probability of observing the event [28]:

$$r(e, t) = C \cdot (1 - P(e, t)) \quad (2)$$

where  $C$  is a constant,  $P(e, t)$  is the probability of observing certain events  $e$  at time  $t$ .

*Surprise* is typically understood as “the observation of an event that violates strongly expectations, i.e., an event that occurs and was strongly expected not to occur” [28]. Mathematically, it can be defined as:

$$r(e, t) = C \cdot \frac{1}{P(e, t)} \quad (3)$$

Note that according to this definition, for an event to be surprising, it should occur in cases where there are not so many other options that could have happened (an event drawn from a uniform distribution where it is different from all other events, is not necessarily surprising) [28]. Surprise is also known as *contextual novelty* as it is related to a particular event happening within a short time span.

Curiosity, novelty or surprise are valuable sources for driving agent’s actions towards exploration, however they do not guarantee learning. According to [29], “the mere fact that an event is novel or surprising does not guarantee that it contains regularities that are detectable, generalizable or useful. Therefore, heuristics based on novelty can guide efficient learning, but are very inefficient in large open ended spaces, where they only allow the agent to collect very sparse data and risk trapping him in unlearnable tasks”. Such heuristics-based approaches have been tested in small and closed spaces [27]. Their applicability to large spaces is impractical and more powerful mechanisms for maximising learning are usually needed [8], [29].

It is worth mentioning that there are other factors to intrinsic motivation than those mentioned above. A curious agent is usually driven by exploration [19], [30]. In another family of intrinsically motivated systems, the actions are chosen so that the error in predicting the environment or certain situation is minimised. In such scenario, the goal is to master a specific task rather than explore the space [31]. Some intrinsically motivated systems rely on *Learning progress* to decrease the error in prediction. The error in prediction

decreases maximally fast [22], [20], [31]. It has been argued that this form of intrinsic motivation is largely correlated to aspects of children’s development [31]. For this approach to effectively work, one needs to identify what is usually referred to as *progress niches*: situations that are neither too hard nor too easy to predict. Once identified, the niches are used to guide the behaviour towards progress in development. When a niche is learned, it is automatically discarded as it becomes predictable and learning continues.

## V. PROCEDURAL CONTENT GENERATION

Procedural Content Generation (PCG) refers to the process of automatic design of game content with little or no human interference [5]. The field has enjoyed increasing attention in recent years and rapidly expanded both in academia and the industry. PCG methods are widely used, and very beneficial, in cases that require generation of massive amount of content that is similar yet variant. Popular examples include the use of variations of fractal methods to generate trees and vegetation [5] as demonstrated in the movie *Avatar* and in the game *No Man’s Sky*.

Research in PCG is motivated by three main streams; generating infinite variations of content, adapting content generation to players and providing aid during the game design process. In what follows, we elaborate on each of these categories and demonstrate that RL methods powered by intrinsic rewards promise solutions in all of these areas.

## VI. INTRINSICALLY MOTIVATED REINFORCEMENT LEARNING FOR CONTENT GENERATION

One of the main motivations behind the use of PCG in games is that it permits automatic exploration of infinite variations of content. This is attributed to a great extent to the efficiency of well established stochastic search methods such as Evolutionary Algorithms (EA) and the applicability of exhaustive search to constraint satisfaction problems [32]. Most of the research in the field of PCG navigates the search space of possible content through some form of evolution combined with predefined quality criteria that guides the search towards better fit individuals (there are of course some exceptions where other approaches such as constraint satisfaction are used [32] but EA is still considered the dominant paradigm). The hope is that, with efficient representation of the search space, a proper definition of a fitness function and enough time, one could generate infinite amount of content variations that satisfy certain quality measure(s). This type of methods falls under the umbrella of Search-based Procedural Content Generation (SB-PCG) [33], a paradigm that has been widely adopted to generate different types of content that ranges from necessary game components such as tracks in racing games [34], levels in 2D platformers and physics-based games [35] and maps in strategy games [36], to the creation of decorations and auxiliary items such as weapons in shooter games [37] and towers in defence games [38], to even the generation of complete games [39].

To generate content with SB-PCG, one needs to identify *content representation* and define an accurate

*fitness* measure of content quality. Different measures of quality have been proposed in the literature that mostly relate to a predefined scale of playability [35], fun [40] or difficulty [41]. While these measures find good solutions for traditional problems such as learning, planning, or creating playable content, there is no clear guidelines for how such methods can be adopted to creating solutions of more abstract qualities such as beauty, novelty, curiosity or surprise. Such aspects are generally hard to model or capture computationally. Previous attempts to model such aspects within the IMRL framework, such as the ones we described in Section IV, have proven effective and shown promising results when dealing with such attributes [42], [18], [24].

One could claim that measures of curiosity and novelty, for instance, can be encoded in the fitness function of EA. Although possible, and indeed tested [43], [44], evolution through novelty is still in its infancy and investigation on its applicability to the field of PCG is still in its early phases. Even if such approach is viable, this particular setting of EA works well for episodic content generation (where interaction between the user and the system is suspended until the system produces the final results, e.g. one level). RL is particularly interesting for online/interactive PCG systems where intermediate interactions and progressive feedback from the user are necessary to guarantee the quality of the solution (as we will discuss in Section VII and VIII). As the direction of implementing novelty in traditional PCG systems and the direction we propose in this paper are still in their infancy, we believe that comparison between these lines constitutes a future direction as both become more mature.

Traditional PCG methods are usually evaluated based on how well the solutions found satisfy the objectives defined. This measure of quality however discards one of the very important aspects of a good PCG system: diversity or expressiveness of the solution space [45]. A PCG system that generates hundreds of good solutions that differ in a few small details should not be regarded as a better system than the one that generates only a few solutions but with more interesting variations. Defining measures of qualities for PCG systems, where diversity, novelty and surprise (as computationally defined in Section IV) play key roles, is recently gaining more attention [46]. So far, these measures are employed to evaluate the content generated and they are not regarded as goals by themselves (with very few exceptions [47]). Steering content generation through intrinsic rewards towards solutions that encompasses such notions constitutes an important motivation behind the direction we advocate in this work.

We believe IMRL is promising for generating content. We summarise the reasons loosely following the traditional terminology to identify PCG systems:

- Intrinsic vs. extrinsic PCG: Most current PCG systems create content according to predefined explicit *goodness* measures that relate to design constraints, playability or factors of player experience. Alternatively, an

progresses towards solutions that satisfy the aforementioned objectives while being internally motivated to creating content of wider diversity and better aesthetic quality through its curious nature. In a PCG system that create content for a 2D platform game such as *Spelunky* for instance, an external reward could be given for arranging blocks so that the final design is playable. The system could also employ an intrinsic reward for novel placement of components measured according to Equation 2.

- Partially vs. fully observable domain knowledge: Most PCG systems attempt to create content for predefined games or game domains. The system is usually equipped with knowledge about the representation, mechanics and rules. Very few studies approached the problem of generating complete games with no (or minimal) prior knowledge [48], [39]. This is largely attributed to the difficulty in searching a wide space of possibilities when the system is bounded by only a few constraints. IMRL can be applied to problems for which domain knowledge is only partially observable or expensive to obtain. RL is particularly interesting when the goal is to improvise new types of games from scratch with no or minimal domain knowledge. The system starts from an initial, minimal definition of the game and gradually increase the game complexity through, for example, interactions with the player. For instance, in a simple physics-based game that teaches a child the basic physics laws, tapping a floating bubble will cause the candy inside to fall down. Once the player learns this principle and the game observe this (by monitoring the score), the game detects a gradual increase in the state of boredom (according to the PEM). Driven by its inherent pursue to please the player and to learn something new, the game initiates a new concept such as blowing a bubble or introduces a more complex mechanic (by picking from an annotated repository or combining previously learnt concepts) such as attaching the candy inside the bubble to another item to prevent it from falling when tapping the bubble. Using layered learning to acquire hierarchical skills that permit complex adaptive behaviour can be applied for this purpose [49], [30].
- Standalone vs interactive PCG: Most current PCG systems work under the standalone or the mixed-initiative paradigms. Unlike standalone systems, interaction with the system is permitted in the mixed-initiative approach. the interaction however is used as a form of reward for functional solutions and not through the intermediate steps towards building the solution. For instance, in *Galactic Arms Race* [37], a game pioneered in introducing the mixed-initiative paradigm for content evolution, each customised weapon produced is based on the history of the user's preferences demonstrated by the usage patterns of previous weapons. Although the system is considered highly interactive, adjusting the behaviour of the system through interaction with the user only

happens when the final solution (a functional weapon) is ready. Potentially, Interaction could also be introduced earlier when producing micro content. A PCG system could as well be rewarded for the incremental progress rather than the quality of the final solution. RL is well suited for such scenarios.

## VII. INTRINSICALLY MOTIVATED REINFORCEMENT LEARNING FOR EXPERIENCE-DRIVEN PROCEDURAL CONTENT GENERATION

Player experience plays a key role within the Experience-Driven Procedural Content Generation (ED-PCG) paradigm [50]. ED-PCG methods are centred around the player as they rely on the player's implicit or explicit feedback to evaluate content quality [51]. A number of sources have been investigated as measures of player experience: subjective feedback can be collected from questionnaires [52], [53], objective measures such as heart rate, skin conductance or facial expressions [54], [53] and data collected from players' behaviour during the in-game interactions [37]. Any of these signals, or a combination of them, can be implemented in a PCG system.

Few studies have successfully adopted players' feedback in the content generation loop [52], [53]. Those that do usually utilise classification methods to profile players. Identified personas are then used by a search method (Gradient Ascent Optimisation, EA or stochastic search) to select the next content [53], [52], [55]. These methods work well when a large enough dataset is available for classification. In contrast, RL is particularly interesting in situations in which labels are sparse and/or when we want to reward the subsequent progress of learning about the player and not only the final outcome of the system.

Recently it has been suggested that interactive learning [56], that combines ideas from learning by demonstration, learning by exploration and tutor feedback, might be a new perspective on learning in applications such as robot learning [57]. Extra reinforcement signals traditionally used include: requesting actions [58], preferences between states [59], iterations between practice and user feedback [60] or merely negative and positive reinforcement signals [61]. Studies have shown that systems that augment human signals in their learning progress outperform other autonomous learning alternatives [61].

The IMRL is an interesting framework for ED-PCG. In the IMRL framework, the intrinsic reward signal can incorporate several reward components coming from different sources. While curiosity, surprise and novelty constitute some of the frequently used rewards as intrinsic motivation, emotions have recently been suggested as a viable supplementary signal. Joy, fear and distress [42], as well as basic emotional dimensions defined in the appraisal theory such as valence and arousal [62] are translated into numerical features that provide intrinsic reward in a reinforcement learning context.

IMRL can be extended, under similar settings, to incorporate players' feedback to guide content generation.

During an interaction session, the system is driven to increase its knowledge about the player –i.e. learning a better Player Experience Model (PEM)– and consequently to provide better content. The steps the system follows can be described as follows:

- 1) Observe the current game state,  $s_t$ .
- 2) Explore the action space,  $A$ , for a set of content modifications that has the desired effect on player experience.
- 3) Perform the set of actions,  $a_t$ , to modify the game content and observe the new game state,  $s_{t+1}$ .
- 4) Predict how the current change of content affects player's experience,  $\hat{p}_{t+1}$ .
- 5) Present the best predicted state,  $s_{t+1}^*$ , to the player and inquire the player about the actual change in experience,  $p_{t+1}$ .
- 6) Use the error in predicting player experience,  $e = \hat{p}_{t+1} - p_{t+1}$ , to update the predictive model (PEM).
- 7) Use the learning progress,  $e_{t+1} - e_t$ , to update policy.
- 8) Return to step 1.

Models of player experience –which map content to affective states (see for example [52], [63])– are essential in this process as they serve two main purposes: they are used to predict the appeal of content to a specific player (step 2 and 4), and to guide the search of *good* content (step 2). The hope is that, with enough time, the game will eventually learn an accurate estimator of player preferences and be able to effectively create personalised content. For the method to work, the feedback coming from an actual player is used to adjust the behaviour of the system. Two adjustment functions are derived from human response, the first function calculates the difference between the system's belief and the actual knowledge of player experience (step 6). Techniques such as back propagation can then be implemented to adjust the PE models to become better predictors. The second function calculates the accumulated differences of the system predictions of player experience through time (i.e. it measures how the system progresses in learning to predict player experience) (step 7).

Through the aforementioned procedure, the quality of the presented content and the accuracy in predicting player experience will continuously increase. Once PEMs with sufficient accuracy are built, these models can be used in absence of the player. The possibility that the player can interfere when necessary to adjust the behaviour can still be present.

We identified a number of key advantages for the use of IMRL within the ED-PCG framework:

- Active learning of personalised models of PE: The IMRL can start from average models of PE which can be actively adjusted during the in-game sessions to better fit a specific player preference. There will be minimal need to collect data about player-specific behaviour and build personalised models offline.

PEMs and presenting personalised content can be done simultaneously. As PEMs become more accurate, the content presented becomes better as well as interesting (since the system explores the content space in its pursuit of maximising its knowledge about the player).

Note that emotion is of complex nature, and modelling it has proven to be a hard problem. However, recent trends have shown that there are some aspects of players' emotion that we could quantify with powerful machine learning methods such as neural networks [64]. These models could be used within the proposed framework and incremental learning approaches could be used to improve the model performance at runtime. Needless to say, the performance of the proposed approach is greatly bounded by the accuracy of the PEMs and the efficiency of the update methods.

### VIII. INTRINSICALLY MOTIVATED REINFORCEMENT LEARNING FOR MIXED-INITIATIVE DESIGN TOOLS

This case could be regarded as a special setting of the previous section where the feedback in this scenario comes from the designer rather than the player.

So far, PCG systems designed to provide assistant for game designers can be categorised into two main directions: (1) systems that are based on constraint satisfaction and work by observing constraints or partially designed content and propose solutions with minimum conflicts [65]; and (2) systems that generate large amount of content in an aim to provide diversity [66]. Since it is impractical for the designer to iterate through the full space, such methods are usually supported with interactive visualisation techniques that project the space of generated content on predefined, problem specific, expressive axis [46]. In both approaches, the feedback coming from the designer is treated as either constraints that should be satisfied or an indicator of preference that is translated to a measure of similarity.

IMRL can be a viable solutions for designing efficient mixed-initiative design tools that learns from designers. Such approaches are also more appealing to human designers as they are inquired about feedback frequently without delaying it to the end of the process. An IMRL mixed-Initiative tool makes use of established supervised learning techniques to model a humans reward function and uses the learned model to create content so that the reward is maximised (although in some studies, human reward can be directly applied as an enforcement signal without the need to construct a model).

There is a well-established body of work in the field of robot learning on implementing RL framework that incorporates a human in the learning process. Those can be a potential inspiration of implementing a similar method for designing assistive tool for games. The feedback coming from the user can take many forms: some learning systems relies on humans providing advice encoded in a domain-specific language [67]. Another approach is to learn through demonstration provided by human trainers which can take the form of a sequence of actions [68].

task while the system monitor the process [69]. The other possibility is to gather human feedback as a scalar reward that indicates a positive, negative or neutral signal [61]. Some of these methods are more appealing to humans while others are easier to implement in a computational models. We hypothesise that there will be interesting and useful lessons to be learned and challenges to be overcome by investigating these approaches.

### IX. CHALLENGES

We argue that IMRL framework is a feasible solution to the main problems in PCG. Naturally, the application of IMRL to PCG presents a number of interesting domain-specific challenges. Some of these challenges are also common in traditional methods for content generation:

#### A. Choice of an appropriate content representation

Similar to EA, Content Representation (CR) plays a very important role in the efficiency of the search algorithm. In the IMRL framework, CR is particularly important for detecting salient features that support the discovery of novel solutions. For this purpose, it is important that CR should not be limited to the visual appearance of the game but should also include other facets that contribute to what the game is actually about and that could be of potential interest for curious behaviour. For instance, novel content could be generated as a result of a new combination or mutation of existing game rules and mechanic [48], [70].

#### B. Choice of an appropriate action representation

The possible actions available to the RL system are usually those defined by the game mechanics and rules. However, since we are interested in creative behaviour and improvisation, the system should be allowed to apply *meaningful* modifications even to the core elements of the game. While this will significantly increase the size of the search space, the hope is that with efficient representation and modelling methods, this will also boost spotting novel instances.

#### C. Choice of reward function

Judging the quality of content is what actually steers the behaviour of a generator. While the most significant reward for a curious agent comes from itself as a pursue of knowledge and discovery, additional augmenting signals can be incorporated to maintain a desirable behaviour. As discussed, extrinsic rewards can come from game-specific constraints, but they can also come from a human interacting with the system either as a player providing feedback about experience or as a designer looking for inspiration or assistance. The actual formulation for integrating different sources of rewards is a nontrivial empirical question but one can draw from previous research on multi-objective optimisation [71] and cooperative coevolution [72]. Special care should also be taken when humans are present in the loop as they have a general tendency of being inconstant in the rewards they provide and their preference might change over time [73]. Moreover, studies have shown that people



have different patterns in assigning rewards (some have clear, strong opinion while others do not) and some are more dedicated than others [73]. All of these factors have a great influence on the system's behaviour and learning progress.

## X. CONCLUSIONS AND FUTURE DIRECTIONS

In this paper, we advocate the use of intrinsically motivated reinforcement learning approaches for automatic generation of game content. We discussed our motivations and highlighted the factors that inspired this work. We extensively surveyed previous work on exploring the applicability of IMRL to different problems that PCG can benefit from. We further formulated the main features that promote the use of intrinsically motivated systems for game content generation. We investigated a number of directions for implementing an IMRL system for PCG and highlighted a number of interesting use cases: (1) efficient exploration of the content space and higher chances of creating diverse and interesting artefacts, (2) active construction of models of player experience while generating personalised content and (3) building efficient mixed-initiative design tools. We explored a number of directions where IMRL is promising and we present the challenges expected.

This work is primarily motivated by our belief that metrics of novelty, surprise and diversity are what truly identify a good content generator. Thereafter, most of the discussion presented in the paper revolves around these aspects. Through our survey of previous work and investigations of current PCG systems, we concluded that the proposed framework is indeed promising and worth further research in this specific domain.

Most of the points highlighted in this paper are milestones for future work. We are currently working on investigating the applicability of the framework for generating diverse, yet playable content for a physics-based game and our ultimate goal is to implement a system that improvises complete games learning from the interaction with the player. The purpose of this paper is to invite researchers in the field to explore the potential of the framework for game content generation as we believe it has many desirable characteristics that motivate further investigations.

## ACKNOWLEDGMENTS

The research was supported in part by the Danish Research Agency, Ministry of Science, Technology and Innovation; project "PlayGALe" (1337-00172).

## REFERENCES

- [1] N. Chentanez, A. G. Barto, and S. P. Singh, "Intrinsically motivated reinforcement learning," in *Advances in neural information processing systems*, 2004, pp. 1281–1288.
- [2] M. Bowling, J. Fürnkranz, T. Graepel, and R. Musick, "Machine learning and games," *Machine learning*, vol. 63, no. 3, pp. 211–215, 2006.
- [3] A. Liapis, G. N. Yannakakis, and J. Togelius, "Computational game creativity," in *Proceedings of the Fifth International Conference on Computational Intelligence and Games (CGI)*, 2016.
- [4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
- [5] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2015.
- [6] P. Dayan and B. W. Balleine, "Reward, motivation, and reinforcement learning," *Neuron*, vol. 36, no. 2, pp. 285–298, 2002.
- [7] P. Reed, C. Mitchell, and T. Nokes, "Intrinsic reinforcing properties of putatively neutral stimuli in an instrumental two-lever discrimination task," *Animal Learning & Behavior*, vol. 24, no. 1, pp. 38–45, 1996.
- [8] A. Baranes and P.-Y. Oudeyer, "Active learning of inverse models with intrinsically motivated goal exploration in robots," *Robotics and Autonomous Systems*, vol. 61, no. 1, pp. 49–73, 2013.
- [9] F. Wardle, "Getting back to the basics of childrens play," *Child Care Information Exchange*, vol. 57, pp. 27–30, 1987.
- [10] H. Q. Ngo, M. D. Luci, A. Foerster, and J. Schmidhuber, "Learning skills from play: Artificial curiosity on a katana robot arm," in *IJCNN*, 2012, pp. 1–8.
- [11] R. M. Ryan and E. L. Deci, "Intrinsic and extrinsic motivations: Classic definitions and new directions," *Contemporary educational psychology*, vol. 25, no. 1, pp. 54–67, 2000.
- [12] D. E. Moriarty, A. C. Schultz, and J. J. Grefenstette, "Evolutionary algorithms for reinforcement learning," *J. Artif. Intell. Res. (JAIR)*, vol. 11, pp. 241–276, 1999.
- [13] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 1998.
- [14] J. Peters, S. Vijayakumar, and S. Schaal, "Reinforcement learning for humanoid robotics," in *Proceedings of the third IEEE-RAS international conference on humanoid robots*, 2003, pp. 1–20.
- [15] K. Doya, K. Samejima, K.-i. Katagiri, and M. Kawato, "Multiple model-based reinforcement learning," *Neural computation*, vol. 14, no. 6, pp. 1347–1369, 2002.
- [16] M. Csikszentmihalyi, "Flow and the psychology of discovery and invention," *New Yprk: Harper Collins*, 1996.
- [17] P.-Y. Oudeyer, F. Kaplan, and V. V. Hafner, "Intrinsic motivation systems for autonomous mental development," *Evolutionary Computation, IEEE Transactions on*, vol. 11, no. 2, pp. 265–286, 2007.
- [18] J. Schmidhuber, "Driven by compression progress: A simple principle explains essential aspects of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes," in *Anticipatory Behavior in Adaptive Learning Systems*. Springer, 2009, pp. 48–76.
- [19] X. Huang and J. Weng, "Novelty and reinforcement learning in the value system of developmental robots," 2002.
- [20] F. Kaplan and P.-Y. Oudeyer, "Maximizing learning progress: an internal reward system for development," in *Embodied artificial intelligence*. Springer, 2004, pp. 259–270.
- [21] P. Sequeira, F. S. Melo, and A. Paiva, "Learning by appraising: an emotion-based approach to intrinsic reward design," *Adaptive Behavior*, p. 1059712314543837, 2014.
- [22] J. Schmidhuber, "Curious model-building control systems," in *Neural Networks, 1991. 1991 IEEE International Joint Conference on*. IEEE, 1991, pp. 1458–1463.
- [23] —, "Formal theory of creativity, fun, and intrinsic motivation (1990–2010)," *Autonomous Mental Development, IEEE Transactions on*, vol. 2, no. 3, pp. 230–247, 2010.
- [24] T. Schaul, Y. Sun, D. Wierstra, F. Gomez, and J. Schmidhuber, "Curiosity-driven optimization," in *Evolutionary Computation (CEC), 2011 IEEE Congress on*, 2011, pp. 1343–1349.
- [25] T. Pfingsten, "Bayesian active learning for sensitivity analysis," in *Machine Learning: ECML 2006*. Springer, 2006, pp. 353–364.
- [26] J. Storck, S. Hochreiter, and J. Schmidhuber, "Reinforcement driven information acquisition in non-deterministic environments," in *Proceedings of the international conference on artificial neural networks, Paris*, vol. 2. Citeseer, 1995, pp. 159–164.
- [27] S. Thrun, "Exploration in active learning," *Handbook of Brain Science and Neural Networks*, pp. 381–384, 1995.
- [28] P.-Y. Oudeyer and F. Kaplan, "What is intrinsic motivation? a typology of computational approaches," *Frontiers in neurorobotics*, vol. 1, p. 6, 2006.

- [29] J. Gottlieb, P.-Y. Oudeyer, M. Lopes, and A. Baranes, "Information-seeking, curiosity, and attention: computational and neural mechanisms," *Trends in cognitive sciences*, vol. 17, no. 11, pp. 585–593, 2013.
- [30] A. Stout, G. D. Konidaris, and A. G. Barto, "Intrinsically motivated reinforcement learning: A promising framework for developmental robot learning," DTIC Document, Tech. Rep., 2005.
- [31] F. Kaplan and P.-Y. Oudeyer, "The progress-drive hypothesis: an interpretation of early imitation," Cambridge University Press., Tech. Rep., 2007.
- [32] A. M. Smith and M. Mateas, "Answer set programming for procedural content generation: A design space approach," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 3, no. 3, pp. 187–200, 2011.
- [33] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation," in *Proceedings of EvoApplications*, vol. 6024. Springer, 2010.
- [34] L. Cardamone, D. Loiacono, and P. Lanzi, "On-line neuroevolution applied to the open racing car simulator," in *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, 2009, pp. 2622–2629.
- [35] N. Shaker, M. Shaker, and J. Togelius, "Evolving playable content for cut the rope through a simulation-based approach," in *AIIDE*, 2013.
- [36] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelbäck, and G. Yannakakis, "Multiobjective exploration of the starcraft map space," in *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*. Citeseer, 2010, pp. 265–272.
- [37] E. J. Hastings, R. K. Guha, and K. O. Stanley, "Evolving content in the galactic arms race video game," in *Proceedings of the 5th international conference on Computational Intelligence and Games*, ser. CIG'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 241–248.
- [38] P. Avery, J. Togelius, E. Alistar, and R. van Leeuwen, "Computational intelligence and tower defence games," in *IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2011, pp. 1084–1091.
- [39] M. Cook and S. Colton, "Multi-faceted evolution of simple arcade games," in *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*. IEEE, 2011, pp. 289–296.
- [40] J. Togelius, R. De Nardi, and S. M. Lucas, "Making racing fun through player modeling and track evolution," 2006.
- [41] N. Sorenson and P. Pasquier, "The evolution of fun: Automatic level design through challenge modeling," in *Proceedings of the First International Conference on Computational Creativity. Lisbon, Portugal: ACM*, 2010, pp. 258–267.
- [42] J. Broekens, E. Jacobs, and C. M. Jonker, "A reinforcement learning model of joy, distress, hope and fear," *Connection Science*, vol. 27, no. 3, pp. 215–233, 2015.
- [43] J. Lehman and K. O. Stanley, "Abandoning objectives: Evolution through the search for novelty alone," *Evolutionary computation*, vol. 19, no. 2, pp. 189–223, 2011.
- [44] —, "Evolving a diversity of virtual creatures through novelty search and local competition," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. ACM, 2011, pp. 211–218.
- [45] E. Butler, E. Andersen, A. M. Smith, S. Gulwani, and Z. Popović, "Automatic game progression design through analysis of solution features," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 2015, pp. 2407–2416.
- [46] B. Horn, S. Dahlskog, N. Shaker, G. Smith, and J. Togelius, "A comparative evaluation of procedural level generators in the mario ai framework," *Proceedings of Foundations of Digital Games*, 2014.
- [47] A. Liapis, G. N. Yannakakis, and J. Togelius, "Enhancements to constrained novelty search: Two-population novelty search for generating game content," in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM, 2013, pp. 343–350.
- [48] J. Togelius and J. Schmidhuber, "An experiment in automatic game design," in *IEEE Symposium On Computational Intelligence and Games. CIG'08*. IEEE, 2008, pp. 111–118.
- [49] V. Soni and S. Singh, "Reinforcement learning of hierarchical skills on the sony aibo robot," in *Proceedings of the Fifth International Conference on Development and Learning*, 2006.
- [50] G. N. Yannakakis and J. Togelius, "Experience-driven procedural content generation," *Affective Computing, IEEE Transactions on*, vol. 2, no. 3, pp. 147–161, 2011.
- [51] O. Polozov, E. O'Rourke, A. M. Smith, L. Zettlemoyer, S. Gulwani, and Z. Popovic, "Personalized mathematical word problem generation," in *Proceedings of the 24th International Joint Conference on Artificial Intelligence*. AAAI Press, 2013, pp. 167–175.
- [52] N. Shaker, J. Togelius, and G. N. Yannakakis, "Towards automatic personalized content generation for platform games," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*. AAAI Press, 2010.
- [53] P. M. Blom, S. Bakkes, C. T. Tan, S. Whiteson, D. Roijers, R. Valenti, T. Gevers *et al.*, "Towards personalised gaming via facial expression recognition," 2014.
- [54] H. P. Martínez, M. Garbarino, and G. N. Yannakakis, "Generic physiological features as predictors of player experience," in *Affective computing and intelligent interaction*. Springer, 2011, pp. 267–276.
- [55] N. Shaker, G. Yannakakis, J. Togelius, M. Nicolau, and M. O'Neill, "Evolving personalized content for super mario bros using grammatical evolution," 2012.
- [56] M. N. Niolescu and M. J. Mataric, "Natural methods for robot task learning: Instructive demonstrations, generalization and practice," in *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*. ACM, 2003, pp. 241–248.
- [57] J. Grizou, M. Lopes, and P.-Y. Oudeyer, "Robot learning simultaneously a task and how to interpret human instructions," in *Development and Learning and Epigenetic Robotics (ICDL), 2013 IEEE Third Joint International Conference on*. IEEE, 2013, pp. 1–8.
- [58] M. Lopes, F. Melo, and L. Montesano, "Active learning for reward estimation in inverse reinforcement learning," in *Machine Learning and Knowledge Discovery in Databases*. Springer, 2009, pp. 31–46.
- [59] M. Mason and M. Lopes, "Robot self-initiative and personalization by learning through repeated interactions," in *Human-Robot Interaction (HRI), 2011 6th ACM/IEEE International Conference on*. IEEE, 2011, pp. 433–440.
- [60] K. Judah, S. Roy, A. Fern, and T. G. Dietterich, "Reinforcement learning via practice and critique advice," in *AAAI*, 2010.
- [61] W. B. Knox and P. Stone, "Tamer: Training an agent manually via evaluative reinforcement," in *Development and Learning, 2008. ICDL 2008. 7th IEEE International Conference on*. IEEE, 2008, pp. 292–297.
- [62] P. Sequeira, F. S. Melo, and A. Paiva, "Emotion-based intrinsic motivation for reinforcement learning agents," in *Affective computing and intelligent interaction*. Springer, 2011, pp. 326–336.
- [63] H. Martinez, A. Jhala, and G. Yannakakis, "Analyzing the impact of camera viewpoint on player psychophysiology," in *International Conference on Affective Computing and Intelligent Interaction and Workshops*. IEEE, 2009, pp. 1–6.
- [64] N. Shaker, G. Yannakakis, J. Togelius, M. Nicolau, and M. O'Neill, "Fusing visual and behavioral cues for modeling user experience in games," *IEEE Transactions on System Man and Cybernetics; Part B: Special Issue on Modern Control for Computer Games*, pp. 1519–1531, 2012.
- [65] A. M. Smith and M. Mateas, "Variations Forever: Flexibly Generating Rulesets from a Sculptable Design Space of Mini-Games," *IEEE Transactions on Computational Intelligence and AI in Games*, 2010.
- [66] A. Liapis, G. N. Yannakakis, and J. Togelius, "Designer modeling for sentient sketchbook," in *Computational Intelligence and Games, IEEE Conference on*. IEEE, 2014, pp. 1–8.
- [67] G. Kuhlmann, P. Stone, R. Mooney, and J. Shavlik, "Guiding a reinforcement learner with natural language advice: Initial results in robocup soccer," in *The AAAI workshop on supervisory control of learning and adaptive systems*, 2004.
- [68] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 1.
- [69] S. Schaal, "Is imitation learning the route to humanoid robots?" *Trends in cognitive sciences*, vol. 3, no. 6, pp. 233–242, 1999.
- [70] M. Cook and S. Colton, "A rogue dream: Automatically generating meaningful content for games," in *Proceedings of the AIIDE Workshop on Experimental AI and Games*, 2014.
- [71] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii," in *Parallel problem solving from nature PPSN VI*. Springer, 2000, pp. 849–858.
- [72] M. A. Potter and K. A. De Jong, "Cooperative coevolution: An architecture for evolving coadapted subcomponents," *Evolutionary computation*, vol. 8, no. 1, pp. 1–29, 2000.
- [73] C. L. Isbell Jr, M. Kearns, S. Singh, C. R. Shelton, P. Stone, and D. Kormann, "Cobot in lambdamoo: An adaptive social statistics agent," *Autonomous Agents and Multi-Agent Systems*, vol. 13, no. 3, pp. 337–354, 2006.

# MCTS/EA Hybrid GVGAI Players and Game Difficulty Estimation

Hendrik Horn\*, Vanessa Volz\*, Diego Pérez-Liébana†, Mike Preuss‡

\*Computational Intelligence Group  
TU Dortmund University, Germany  
Email: firstname.lastname@tu-dortmund.de

† School of Computer Science and Electronic Engineering  
University of Essex, Colchester, UK  
dperez@essex.ac.uk

‡Department of Information Systems  
Westfälische Wilhelms-Universität Münster, Germany  
Email: mike.preuss@uni-muenster.de

**Abstract**—In the General Video Game Playing competitions of the last years, Monte-Carlo tree search as well as Evolutionary Algorithm based controllers have been successful. However, both approaches have certain weaknesses, suggesting that certain hybrids could outperform both. We envision and experimentally compare several types of hybrids of two basic approaches, as well as some possible extensions. In order to achieve a better understanding of the games in the competition and the strength and weaknesses of different controllers, we also propose and apply a novel game difficulty estimation scheme based on several observable game characteristics.

## I. INTRODUCTION

The General Video Game AI (GVGAI) competition is an attempt to create artificial intelligence that is not tailored towards a specific game (akin to General Game Playing, GGP). In contrast to GGP games, GVGAI games are modeled after real, well known (albeit simple) video games and incorporate non-deterministic behavior of NPCs. Thus, learning to play these games is not trivial, despite the forward model offered by the GVGAI setup that can be used to explore possible futures of the current game state. The inherent non-determinism discourages plain game-tree search methods and renders this environment suitable to non-deterministic learning algorithms such as Monte-Carlo Tree Search (MCTS) and Evolutionary Algorithms (EA). Both approaches have been shown to work well, but MCTS based controllers tend to exhibit the best overall performance [14]. Still, as of yet, no submitted controller has been able to consistently be successful on all games, showing that all controllers have their weaknesses and strengths. A hybrid controller that combines the strengths of both methods therefore seems promising. But, to our knowledge, few combinations of the aforementioned algorithms into a single GVGAI controller have been suggested (cf. section III).

In this work, we therefore explore different hybridizations, namely (1) integrating parts of the MCTS method into a rolling horizon EA [7, 12], and (2) splitting the computation budget between both methods. Both combinations are experimentally

shown to perform well, with the first hybrid possessing a small advantage. Next to these two naïve hybrids, we also try out further modifications addressing weaknesses discovered during the experiments. However, while improvements are visible in one area, the variations introduced new weaknesses to the controllers. For a more robust controller, further research is needed on how to balance the different components.

As a first step in this direction, we analyze the characteristics of different games and their correlation with the overall winrates of the different controllers in an attempt to uncover strengths and weaknesses. The analysis is based on a difficulty estimation scheme that uses different models to predict controller winrates as a proxy for difficulty from several observable game characteristics.

In the following, we first introduce the GVGAI framework (sect. II) and discuss related work in sect. III. The proposed hybrid controllers are explained in sect. IV and experimentally analyzed in sect. V. Section VI contains the analysis of the difficulty of the games. The paper concludes with a brief summary and outlook in sect. VII.

## II. THE GVGAI FRAMEWORK AND COMPETITION

The General Video Game AI framework is an extension of *py-vgdl*, a benchmark for planning and learning problems implemented by Tom Schaul [15]. This environment proposes a Video Game Description Language (VGDL) to describe two-dimensional real-time games in a very concise and object oriented manner. GVGAI utilizes this implementation, providing a responsive forward model to simulate actions within the game and an interface for controllers to participate in an open competition. The results and rules of this contest, which was initiated in 2014, can be found in [14].

The GVGAI framework communicates information about the game state to the controller via Java objects, although information about the nature of the game, its rules, the type of sprites present and the victory conditions are not provided. Information received contains the game status (score, winner -

if any -, and current time step), the available set of actions on the game, the player's state (position, resources collected) and the position of the different sprites, identified by an integer id, in the level.

Controllers can use 1s of CPU time for initialization, and 40ms at every game tick to return a valid action to play the game. If these limits are not respected, the controller loses the game automatically, with the exception of actions returned between 40 and 50ms, in which case the action executed in the game is *NIL* (no movement applied). During the allocated time, the controller can employ a forward model to explore the effects of actions, by rolling the current game state forward and reaching potential future states. It is important to highlight that most games have non-deterministic elements, so it is a responsibility of the controller to deal with the distribution of next states that the forward model provides from the same pair of state and action.

At the time of writing, the framework contains 80 single-player (some of them used in this research) and 10 two-player games. The single-player planning track was run as a competition in 2014 [14] and 2015, attracting more than 70 entries in total.

### III. BACKGROUND

#### A. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) [2] is a very popular technique that iteratively builds an asymmetric tree by sampling the search space. It builds estimates of the action-values for the different states found during the search, by repeating a sequence of 4 consecutive steps: Tree Selection, Expansion, Monte Carlo Simulation and Back-propagation.

During the *Tree Selection* phase, the tree is navigated from the root according to a *Tree Policy*, until a node with actions not yet expanded is reached. A very common policy is UCB1, described in equation 1 [8],

$$a^* = \arg \max_{a \in A(s)} \left\{ Q(s, a) + C \sqrt{\frac{\ln N(s)}{N(s, a)}} \right\} \quad (1)$$

where  $N(s)$  indicates the number of visits to state  $s$ ,  $N(s, a)$  the number of times an action  $a$  is taken from  $s$ , and  $Q(s, a)$  the empirical average of the rewards obtained from  $s$  through  $a$ . This policy balances between exploitation (first term of equation 1) and exploration (second term), tempered with the value of  $C$ .

During the *Expansion* phase, a new node is added to the tree as a new child. Then, a *Monte Carlo Simulation* starts from that point until reaching the end of the game or a predetermined depth, selecting actions according to a *Default Policy*, which typically selects moves uniformly at random. Finally, the *Back-propagation* step updates the  $Q(s, a)$  values of all nodes visited during the *Tree Selection* step using the reward observed in the state reached at the end of the *Monte Carlo Simulation*.

MCTS has been successfully used in General Game Playing (GGP) [1], winning the 2007 and 2008 AAAI GGP competitions, and it has been extensively used in the GVGAI competitions up to date. The winner of the 2014 GVGAI competition, Adrien Couëtoux, implemented *OLETS* (Open Loop Expectimax Tree Search) [14], a variant of MCTS without Monte Carlo simulations. The tree is navigated from the root, using a variant of UCB1, until a new node is added, which state is evaluated and the result Back-propagated to the nodes of the tree. Also in the 2014 competition, the third ranked entry was the provided sample MCTS controller. It's worth highlighting that this controller employed a very simple state evaluation function, which used the game score plus a high positive (negative) value if the game was won (resp. lost), but it still performed well across several games.

#### B. Rolling Horizon Evolutionary Algorithms

Traditionally, in planning problems, evolutionary algorithms (EA) are used offline to train a controller or solver that then tackles the real problem [5]. A rolling (or receding) horizon EA (RHEA) operates by evolving a sequence of actions online, out of which only the first one of the best sequence or individual is performed in the real game. A fast EA, executed at every time step, tries to determine the best action sequence from the current state, evaluating each individual with an evaluation of the state reached at the end of such sequence.

This algorithm was first implemented for the Physical Travelling Salesman Problem (PTSP) by Perez et al. [12], showing a better performance than MCTS in the PTSP. More recently, the work by Justesen et al. [7] shows that RHEA can achieve high performance in Hero Academy, a game with a very large branching factor.

The GVGAI framework includes a sample controller that implements a steady state RHEA, known as microbial GA [6]. In this controller, individuals are compared in pairs, chosen at random, and the one with the worst fitness is mutated randomly, also taking parts from the other's genome with a small probability. Although this controller ranks worse than the MCTS Sample controller in all game sets, many participants have worked with different versions of this algorithm, and about 50% of the top 10 entries in the final rankings of the 2015 competitions were RHEA based algorithms<sup>1</sup>.

#### C. Hybrids and Hyper-heuristics

The GVGAI Competition took place in three different legs during 2015, with a different winner for each one of them. Although the algorithms were different, all three had something in common: they were a combination of techniques that were selected depending on certain characteristics observed in the games. The winner of the first leg (*YOLOBOT*; and overall winner of the championship), employs Best First Search or MCTS to reach a targeted sprite in the game, depending on the game being identified as deterministic or not, respectively. The winner of the second leg, *Return42*, differentiates the game

<sup>1</sup>Results available at [www.gvgai.net](http://www.gvgai.net)

according to the same concept, using A-Star for pathfinding in deterministic games, and random walks for stochastic scenarios. Finally, the winner of the last leg, *YBCRIBER*, combines a danger prevention mechanism with iterative width (IW [9]), using pruning and a dynamic look-ahead scheme to perform statistical learning on the different sprites of the game [4].

A different approach, which the research expands on, is the combination of several techniques into a hybrid algorithm. Specifically, combinations of MCTS and EA in GVGAI have been tried in previous works. For instance, Perez et al. [11] combined Fast Evolution with MCTS for General Video Game Playing. The objective was to evolve a set of weights  $W = \{w_0, w_1, \dots, w_n\}$  to bias action selection during the Monte Carlo simulations of the algorithm. In this work, every MC simulation evaluates a single individual, providing as fitness the reward calculated at the state reached at the end. For each state, a set of features  $F = \{f_0, f_1, \dots, f_n\}$  is extracted, and the relative strength of each action ( $a_i$ ) is calculated as a linear combination of features and weights. On each move during the simulation, actions are picked at random with probabilities derived from applying a Softmax function to the relative strengths of each action. For more information about this algorithm, the reader is referred to [10].

In a different approach [13], a RHEA builds a tree search while evaluating the different individuals. Every time a sequence of actions is executed from the current state, new nodes will be added to this tree, one per action performed. As some initial sequences are repeated during the evaluation of the different individuals, most children of the root node will be visited repeatedly, allowing it to calculate an average of the rewards or fitness obtained by the different individuals. Finally, the recommendation policy (that chooses which action to take in the real game), can select the move based on this value, rather than depending only on the best individual. This helps reduce the effect of noise in the state evaluation of stochastic games without the need of evaluating the best individual multiple times.

#### IV. MCTS/EA HYBRIDIZATIONS FOR GVGAI

Of course, there are many ways to combine MCTS and EAs within one controller, and some of these possibilities have already been explored, as described in sect. III-C. When envisioning more direct hybrids, it appears to be simpler to augment a RHEA (see sect. III-B) with components taken from MCTS than the other way around. Something we have to keep in mind is that we are situated in a realtime environment. As there is a constant limit (40 ms) of time that can be employed for computing the next move, inserting a new mechanism into an existing controller at the same time means to reduce the available resources for the original algorithm.

Note that the parameter values employed for the different controllers stem from manual testing limited by the runtime of the experiments and could still be improved. Running a controller on 10 games with 5 levels each and several repeats can take several hours.

##### A. RHEA with rollouts: *EARoll*

This simple scheme takes over the RHEA algorithm and extends it with rollouts: when the simulation of the moves contained in the currently simulated individual/genome is finished, a predefined number of rollouts of parametrized length is performed from that point on, and the fitness of a move combination is computed as the average of the EA part and the MCTS part.

In order to characterize the type of hybridization, one could say that a minimal MCTS is performed during the assessment of move combinations within an EA. We therefore call this an integrated hybrid. The expected benefit of adding rollouts is that the algorithm can look into the future a bit further and thus the chance of detecting a critical path (that would with high probability lead to a loss) increases. It can therefore avoid such paths much earlier. Just extending the length of the genome (the number of consecutive moves evolved) would not have the same effect as that would mean that only one specific move combination (albeit a longer one than before) is tried. Taking into account that GVGAI games are usually non-deterministic, the chances of finding exactly that move combination that leads into a critical path can be expected to be much higher if we sparsely sample a game tree from a specific starting point than if we try only one move combination.

However, this advantage comes at a cost: adding rollouts of course uses up precious computation time, so that the number of moves that can be evaluated within the time constraints decreases. The results reported in this work have been obtained with the following parameters: genome length: 8 steps, simulation depth 9 steps, population size 7, and number of rollouts 300.

##### B. RHEA, then MCTS for alternative actions: *EAaltActions*

This alternative hybrid approach resembles an ensemble approach: after running the RHEA for a predefined time, we use MCTS for checking alternative solutions. That is, the MCTS is allowed to compute a suitable move first, excluding the one chosen by the RHEA. After finishing the MCTS run, we compare the results of the best moves detected by both algorithm and use the better one.

It is an open question how the available time (40 ms) should be distributed between the two approaches. For reasons of simplicity, we spend approximately the same amount of time on both. Our results have been obtained with the following parameters: genome length = simulation depth = 9 steps, population size 5, and number of rollouts 20.

##### C. *EARoll* plus sequence planning: *EARoll-seqPlan*

The basic idea of this controller is to reuse a computed sequence of moves (a plan). In keeping an existing plan, one could just start the computation from the next planned move. After every move, the first move is removed from the plan and the whole plan is shifted upwards. In a deterministic environment, this would make a lot of sense because in GVGAI one-player games, we have no opponent, and the NPCs are usually not very aggressive. Thus, continuing the

plan at least a few steps could save computation time for further exploration of future states. However, it is known that the GVGAI games are often heavily non-deterministic, so that it is not easy to predict how well our approach works. In contrast all other controllers suggested here, we cannot react quickly if something unforeseen happens.

We run this controller with the following parameters: genome (plan) length 8 steps, simulation depth 9 steps, population size 5, and number of rollouts 20.

#### D. *EAroll + occlusion detection: EAroll-occ*

Looking at the behavior of the EAroll controller, from time to time we observe that it stands still for some iterations and then performs a sequence of moves it could have started earlier. The reason for this is that an action sequence which leads to a reward (as, e.g., moving onto a diamond in the game boulder dash) has the same fitness if some inconsequential actions are added to the beginning of the list. In the literature, this problem is sometimes addressed with decaying rewards, but this approach needs to be parameterized. Instead, we want to completely remove any unnecessary actions from the sequence to improve the overall performance of the controller. In order to be able to detect the right position within the action sequence to start the execution, we need to reserve some time (5 ms) so that we can do a binary search for the starting position from the middle of the sequence to the beginning, watching out for the first occasion for which the reward stays the same. We then remove all earlier moves from the sequence.

This controller is run with the following parameter values: simulation depth 9 steps, population size 6, and number of rollouts 100.

#### E. *EAroll + NPC attitude check: EAroll-att*

The last controller variant also employs EAroll as base controller and tries to determine the attitude of the different NPC characters in order to allow or forbid moves into their neighborhood. As there is no information available at the start of the game that allows us to infer the consequences of collisions between the avatar and specific NPCs, this has to be learned during runtime. Fortunately, while setting up the next move, we simulate a lot of steps we do not actually perform, but we obtain information about when the controller would be rewarded or would lose a game. We assume that the behavior of a specific NPC type does not change completely over time (which is not always true, cf. Pac Man) and memorize if the collision had a good, negative or no effect. During the following game phases, we utilize this knowledge. Whenever the avatar moves into the vicinity of an NPC (this is parametrizable, we use a distance of 2 here), the corresponding move gets an extra reward or penalty.

The parameter values employed for this controller are: genome length 7 steps, simulation depth 8 steps, population size 10, and number of rollouts 300.

### V. EXPERIMENTAL ANALYSIS

The most promising hybridizations and corresponding parametrizations as described in IV, as well as the original

MCTS and RHEA examples were tested and compared extensively using the GVGAI framework (see II). Each of the 7 controllers was run 20 times on each of the 5 levels of every game in game sets 1 and 2. This results in 100 playthroughs per controller per game and thus in 14 000 runs total.

In this section, the generality of the controllers will be examined based on their performances on all 20 games. In contrast, in section VI, the results are inspected more closely in terms of how certain aspects of a game affect the performance of different controllers.

In order to obtain interpretable results that are comparable across games, we use winrates as a measure of performance since the scoring systems differ between games. However, it has to be noted that the scores potentially contain more information that could help distinguish different controllers in a statistically significant manner. In terms of measuring performance, we first compute the confidence intervals for the true winrates  $\pi$  of a controller on a game based on its measured winrate  $\hat{\pi}$  in the experiment with  $n = 100$  samples and  $\alpha = 0.05$  (see Figure 1), assuming a binomial distribution and using the Pearson-Klopper method<sup>2</sup>. The experimental winrates are each marked with a circle within the respective interval.

From Figure 1 it is apparent that there are some games where all controllers either perform similarly badly (Camel-race, Dig Dug, Firecaster, Eggomania, Boulderdash) or well (Aliens, Butterflies). In most other games, the controllers EAaltActions, EAroll-seqPlan, EAroll-occ perform a little worse than the rest. However, there does not seem to be a clear pattern which of the other controllers is ahead of the rest in all games. Even if just taking the experimental winrates into account, there is no clear winner across all or most of the games.

However, there are some obvious differences when analyzing the controllers' overall performance, for example using the GVGAI rating system (see [14]). The GVGAI framework determines the ranking of all competing controllers on each game and rewards them according to the Formula 1 scoring system with between 25 and 0 points. The rankings on each game are determined using the winrate as a first criterion and the obtained scores as a tiebreaker. Usually, the completion time is used as a secondary tie breaker, which was dropped for this paper as we were not looking at computational speed in this context. The resulting ratings for all controllers on game sets 1 and 2 are listed in table V.

According to the ratings, EAroll performs best on both game sets, although its lead is bigger on game set 1. Overall, the ratings are much more consistent in game set 1 with EAroll-seqPlan and EAroll-occ constantly on the last two ranks while sometimes placing 2nd and 3rd on game set 2. In contrast, the rating of the MCTS controller is very robust and steady across all games. This is reflected in the total rating: EAroll-seqPlan and EAroll-occ are on the last two ranks regarding overall performance and MCTS is on 2nd place, benefiting from its

<sup>2</sup>R package binom



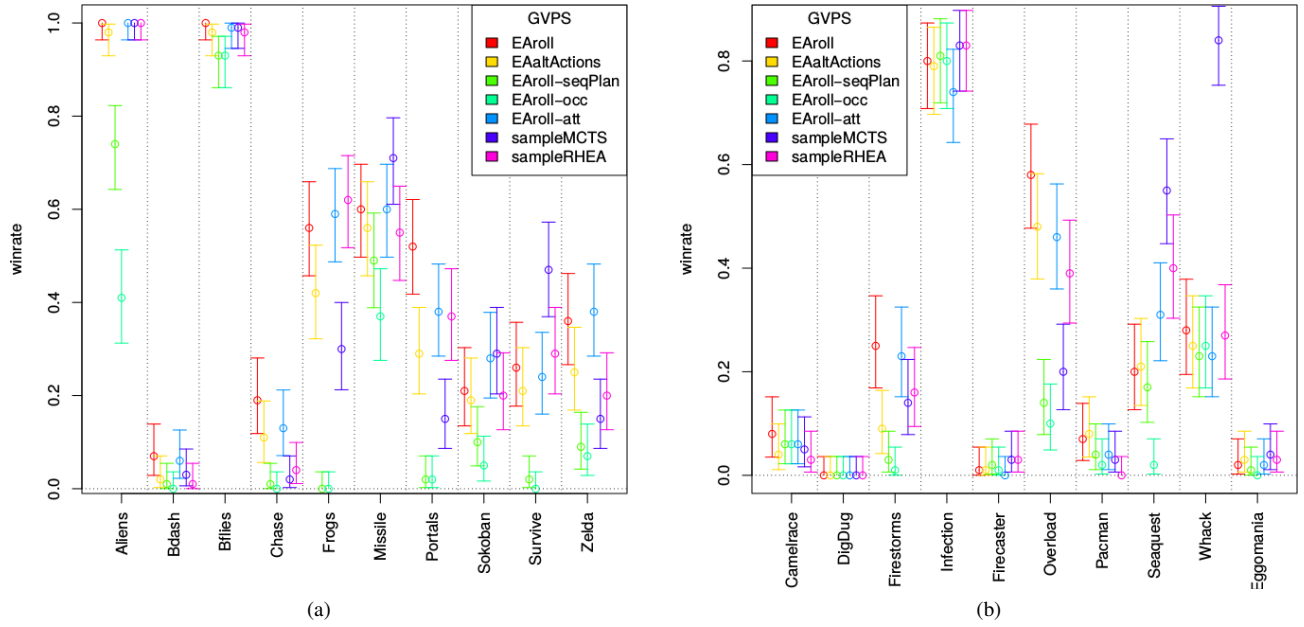


Fig. 1. Confidence Intervals ( $\alpha = 0.05$ ) for the winrates of all tested controllers on game set 1 (fig. 1a) and 2 (fig. 1b)

consistent performance. The RHEA controller and EAroll-att score similarly, with EAaltActions following behind.

However, as is apparent in figure 1, many confidence intervals for the winrates overlap and are therefore not as clear an indicator of controller performance as the resulting difference in GVGAI scores might suggest. Therefore, in order to obtain a better idea of significant differences between the overall performances of the controllers, we compute the Elo-ratings based on a pairwise comparison. The idea of the Elo-rating is to estimate the skill of a player based on the outcomes of past matches, taking into consideration the skill-level of the opponents. For more details, refer to [3].

The pairwise-comparison is conducted based on the confidence intervals for the winrates depicted in figure 1. The performances of two controllers are incomparable if the corresponding confidence intervals overlap. If they do not, one controller plays significantly better ( $\alpha = 0.05$ ) than the other. In order to translate the comparisons to a format suitable for the Elo-rating, a controller performing significantly better than another *wins* a comparison, the other one *loses*. If the controllers are incomparable, the result is a *draw*.

It is important to note that in the GVGAI context, there is no performance development to be expected across games,

since no data is transferred between games (or even runs). Therefore, for the purpose of computing the Elo-rating, all comparisons are considered to have occurred within the same time period, to avoid a bias towards the last games played. The resulting ratings and ranks are listed in table V. Both the Elo as employed by FIDE and the Glicko rating systems result in the same ranks for the controllers.

The GVGAI- and Elo-Ratings agree on placing EAroll-seqPlan and EAroll-occ on the last two ranks, which is unsurprising since they frequently seem to be performing worse than the other controllers. The Elo-Rating indicates that the first place for EAroll is due to statistically significant performance differences as well. While EAaltActions is on rank 5 in both rankings, the rest of the controllers EAroll-att, MCTS and RHEA have different rankings which seem to indicate that between those controllers, there is no clear difference in terms of overall performance.

## VI. GAME DIFFICULTY ESTIMATION

Even though some of the developed controllers are clearly not performing as well as others across all games, it is apparent from figure 1 that some games seem to be easier for all controllers than others. Additionally, despite performing

TABLE I  
GVGAI-RATINGS FOR ALL TESTED CONTROLLERS FOR GAME SETS 1 & 2

Rank	Player	Rating Set 1	Rating Set 2	Total
1	EAroll	206	178	384
2	sampleMCTS	163	163	326
3	sampleRHEA	138	152	290
4	EAroll-att	175	110	285
5	EAaltActions	118	141	259
6	EAroll-seqPlan	80	113	193
7	EAroll-occ	62	86	148

TABLE II  
ELO-RATINGS FOR ALL TESTED CONTROLLERS BASED ON PAIRWISE PERFORMANCE COMPARISONS ON ALL GAMES IN GAME SETS 1 & 2

Rank	Player	Rating	Win	Draw	Loss
1	EAroll	2510	27	89	4
2	EAroll-att	2497	25	92	3
3	sampleRHEA	2443	20	98	2
4	sampleMCTS	2376	27	79	14
5	EAaltActions	2348	17	97	6
6	EAroll-seqPlan	1700	2	79	39
7	EAroll-occ	1525	0	70	50

at a similar level for most games, in some games, certain controllers perform significantly better or worse than the others. The best example for this is Whack-A-Mole where the standard MCTS performs significantly better than all other controllers. In this section, we take a closer look at the games in question to explain the discovered patterns.

As a preparation for a more detailed analysis we identified 10 characteristics of games that might impact the performance of controllers, extending the characterization in [14]. In most cases, the values assigned to the games per characteristic correspond to the fragment of time that the game exhibits the specific characteristic:

- enemies: Do opposing NPCs exist?
- puzzle: Does the game contain a puzzle element?
- indestructible: Are the NPCs indestructible?
- random: Is the NPCs' behavior stochastic?
- stages: Are there multiple stages to a winning condition?
- pathfinding: Is finding a path through the level necessary?
- traps: Does the game contain traps or missiles?
- chase: Do NPCs chase the player with negative effects?

There are a few exceptions, however:

- actions: Number of actions allowed in the game divided by number of actions a controller within the GVGAI framework can access (5)
- NPCs: Normalized average number of NPCs

The evaluation of the respective characteristics is done manually and may therefore contain a bias, but the characteristics were chosen so that a minimal amount of personal judgment is needed. The resulting difficulty estimation for all games in game sets 1 and 2 is shown in figure 2a with table III as legend. Considering the plot, the games seem to vary considerably in terms of difficulty and the type and combination of challenges a controller faces are diverse as well. Since the purpose of the GVGAI competition is to determine *general* video game players, this diversity between the games is expected and advantageous.

However, only in some cases does the sum of the various difficulty characteristic seem to correspond to the actual performance of the controllers, even if the controllers all perform on a similar level. For example, while Boulderdash is very difficult according to figure 2a and seems to be problematic for all controllers (cf. figure 1a), Camelrace and Firecaster result in similarly low winrates (cf. 1b) despite being considered to be much easier (cf. figure 2a). It is thus obvious, that even if the identified characteristics can describe the difficulty of a game appropriately, some factors are more important than others, some even have a positive effect on controller winrates.

To analyze the importance of the characteristics, we estimate the variable importance based on the  $R^2$  statistic of a non-parametric regression model using only one predictor against the intercept only null model as described in the minerva R package documentation<sup>3</sup>. According to the Maximal Information Coefficient (MIC) that estimates the relationship

strength between a difficulty characteristic and the winrate of each controller, none of the identified characteristics seem to be irrelevant. However, with average MIC values across controllers of between 0.18 and 0.41, it is clear that the relationship is more complex and can not be expressed with only one predictor. Nevertheless, the characteristics pathfinding and NPCs seem to have the highest linear relationship strength, followed by indestructible and traps. The Total Information Coefficient (TIC) reports high statistical dependence between the aforementioned characteristics and the controller winrates as well. With an average Maximum Asymmetry Score (MAS) of 0.08, all relationships appear to be relatively monotonous. Additionally, the Minimum Cell Number (MCN) is 2 for almost all relationships, indicating simple functions that can be covered with very few cells.

The various metrics mentioned indicate that it should be possible to create relatively simple models to predict the winrates of the controllers based on the difficulty characteristics. We will first learn a model that predicts controller performance based on the performance data of all controllers on both game sets. Naturally, the model in this case will only be able to pick up on the general trend, not on individual strength and weaknesses of single controllers. We used

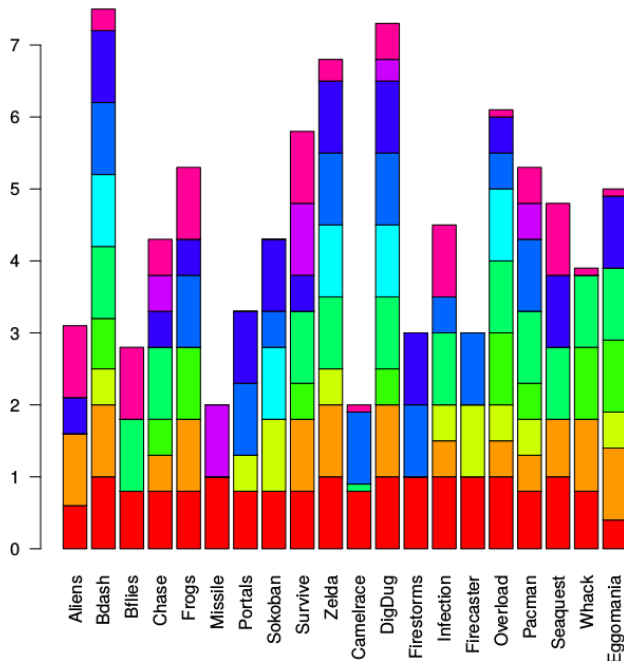
- a regression (linear, logit and logistic),
- an artificial neural network (1 hidden layer, 10 nodes),
- a random tree and forest

model with 10-fold cross-validation and a randomly drawn 90%/10% split to predict the winrates.

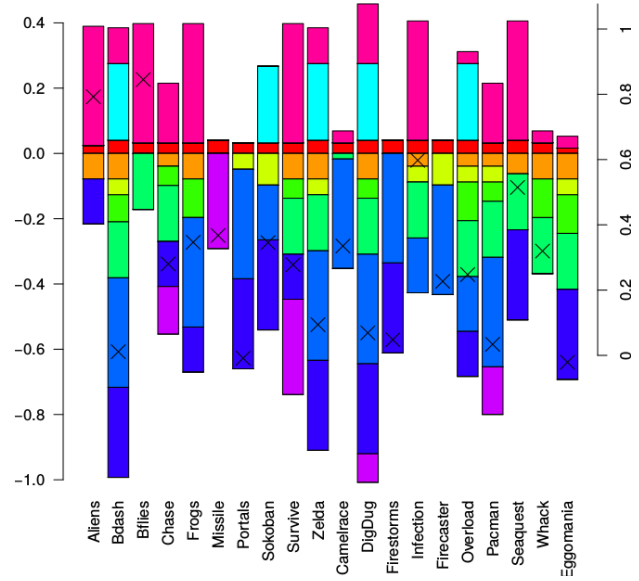
The neural network had the lowest mean squared error consistently (average of  $\approx 0.02$ ), but linear regression and both the random tree and random forest have very acceptable error rates as well (average MSE of 0.05, 0.03, 0.03, respectively). Therefore, it can be seen that the identified difficulty characteristics have an effect on the winrates throughout the controllers and explain them decently. In order to analyze the influence of the different characteristic in greater detail, regression models are used for further analysis since they are easily interpretable and comparable, while still making accurate predictions for this problem. The result of a linear regression model trained on all available data is shown in figure 2b along with the predicted winrates.

The plot shows that, while most of the characteristics have an adverse effect on the predicted winrates, higher NPC, stages and action values actually seem to benefit the controllers. For NPC and action, this can be explained by the fact that all controllers are based on using the forward model in the GVGAI framework to try out different actions. This strategy works better, the earlier a sequence of actions can be evaluated in terms of the expected outcome. Having more NPCs (i.e. a high NPC value) and actions bound to every possible option available to the controller (i.e. a high actions value) results in more frequent events in the games and thus facilitates the evaluation of an action sequence. It is not clear why more complex winning conditions (as expressed by stages) improve the winrates of a controller or if this behavior is the result of having only 5 of the game with stacked winning conditions.

<sup>3</sup><https://cran.r-project.org/web/packages/minerva/minerva.pdf>



(a)



(b)

Fig. 2. Difficulty characteristics (see table III) of all games in game sets 1 and 2. Fig. 2a as estimated and fig. 2b as weighted by linear regression model. Crosses in fig. 2b represent predicted winrates to be read with the right y-axis.

TABLE III

COLORS ASSIGNED TO DIFFERENT DIFFICULTY CHARACTERISTICS

actions	enemies	puzzle	indestructible
random	stages	pathfinding	traps
chase	NPCs		

The most important characteristics in terms of the collective model are random, NPCs, chase, traps, pathfinding and stages. This can also be explained by common traits of the controllers. For example, non-deterministic games (with a high random value) decrease the reliability of the forward model. If the game involves the need to find paths and avoid traps, a general video game player that is forced to rely on exploration is at a disadvantage to strategically searching players.

However, while the collective model presented in figure 2b explains the general difficulty of games well, it is not possible to ascertain the strengths and weaknesses of individual controllers or explain the differences between controller performances on a single game. For this reason, we also learned linear regression models on all available data separately for each controller, but across all games. The resulting model coefficients are visualized in figure 3.

There are several clear differences between the controllers. For example, for the MCTS controller, pathfinding seems to be a much bigger problem than for the others, while a high number of enemies has a positive influence on its winrate. The number of actions also appears to be a much larger positive influence when compared to the other controllers, whereas the number of NPCs is less important. All these factors influence the branching factor of the game-tree and/or the number of viable options for the next action, thus indicating games where

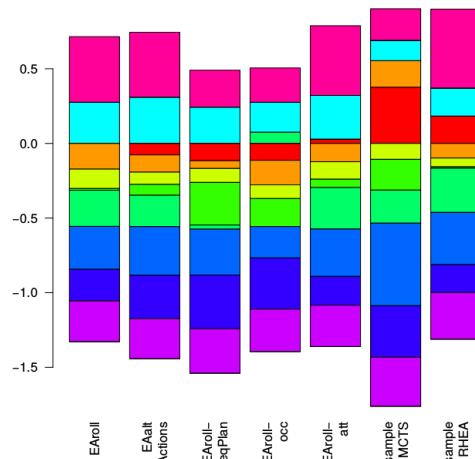


Fig. 3. Linear regression model coefficients for difficulty characteristics (see table III) visualized per controller

a Monte-Carlo approach is less likely to succeed. The MCTS controller deals well with games where the NPCs exhibit randomized behavior, probably for as long as it can execute enough rollouts. The observations also explain why MCTS is doing so well on the game Whack-a-Mole as it exhibits none of the problematic characteristics.

For the RHEA controller, being able to distinguish action sequences quickly is very important, as is reflected by the stress on the number of NPCs and events in the game. Interestingly, EAroll seems to not be affected by this difficulty as much. It seems to deal with almost all of the difficulty characteristics equally well, which explains its robust performance across all games. Its winrate seems to be almost independent from the existence of indestructible NPCs, while the modifications of

this controller have more trouble dealing with this. This is also true for the MCTS controller, while the RHEA controller is also not affected by this difficulty.

Generally, it does seem that the hybrid controllers have inherited characteristics of both controllers, resulting in more robust controllers (especially with controllers EAroll and EAroll-att), thus leading to a better overall performance. The modifications of the EAroll controller seem to fix some of its weaknesses as intended, but at the same time opening up new problems. EAroll-seqPlan, for example, is much less affected by the unpredictability of some games, possibly because it is able to save and propagate a lot more information. On the flipside, the controller is much more susceptible to indestructible enemies. However, it could very well be that these modifications and the hybridization in general could achieve better spread of strengths and weaknesses, even eliminating some, if tuned more thoroughly.

## VII. CONCLUSION AND OUTLOOK

Although not all of the presented hybrid RHEA/MCTS controller variants play better than the original sample controllers, we can state that there is obviously some potential in putting these two base algorithms together in order to obtain better GVGAI controllers. Judging from the difficulty analysis, the hybridization made the resulting controllers more robust. We intend to continue this line of research in (at least) two directions: a) the parametrization of our controllers has not been analyzed systematically, performance may deviate largely from our results with different parameter values, and b) it may be good to dynamically switch on/off the single modules we suggested (sequence planning, occlusion detection and NPC attitude check) as they fit for a given game. The same could also be envisioned on a larger scale for the base algorithms.

However, this requires a clear understanding of and the reasons for the effects of different modifications as well as a way to detect the difficulty characteristics of a game in real-time. A feature based difficulty rating as utilized here can be a step into that direction. Feature-based surrogate models could be employed for predicting which controller should be used for an unknown game after testing a number of actions and events. An interesting further use of the difficulty rating could be to support the selection of a set of games with balanced and distinct challenges for the GVGAI competition.

## REFERENCES

- [1] Y. Björnsson and H. Finnsson. “Cadiaplayer: A Simulation-Based General Game Player”. In: *IEEE Trans. on Computational Intelligence and AI in Games* 1.1 (2009), pp. 4–15.
- [2] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. “A Survey of Monte Carlo Tree Search Methods”. In: *IEEE Trans. on Computational Intelligence and AI in Games* 4:1 (2012), pp. 1–43.
- [3] A. E. Elo. *The Rating of Chessplayers, Past and Present*. B.T. Batsford, London, UK, 1978.
- [4] T. Geffner and H. Geffner. “Width-based Planning for General Video-Game Playing”. In: *Proc. of the IJCAI Workshop on General Intelligence in Game Playing Agents (GIGA)*. 2015.
- [5] F. J. Gomez and R. Miikkulainen. “Solving Non-Markovian Control Tasks with Neuroevolution”. In: *Proc. of the International Joint Conference on Artificial Intelligence*. Kaufmann, San Francisco, CA, 1999, pp. 1356–1361.
- [6] I. Harvey. “The Microbial Genetic Algorithm”. In: *Advances in artificial life. Darwin Meets von Neumann*. Springer, Berlin, Germany, 2011, pp. 126–133.
- [7] N. Justesen, T. Mahlmann, and J. Togelius. “Online Evolution for Multi-Action Adversarial Games”. In: *Applications of Evolutionary Computation*. Springer, Berlin, Germany, 2016, pp. 590–603.
- [8] L. Kocsis and C. Szepesvári. “Bandit based Monte-Carlo Planning”. In: *Proc. of the European Conference on Machine Learning*. Springer, 2006, pp. 282–293.
- [9] N. Lipovetzky and H. Geffner. “Width and Serialization of Classical Planning Problems”. In: *Proc. of the European Conference on Artificial Intelligence*. 2012, pp. 281–285.
- [10] S. Lucas, S. Samothrakis, and D. Perez. “Fast Evolutionary Adaptation for Monte Carlo Tree Search”. In: *Applications of Evolutionary Computation*. Springer, Berlin, Germany, 2014, pp. 349–360.
- [11] D. Perez, S. Samothrakis, and S. Lucas. “Knowledge-Based Fast Evolutionary MCTS for General Video Game Playing”. In: *Proc. of the IEEE Conference on Computational Intelligence and Games*. IEEE Press, Piscataway, NJ, 2014, pp. 1–8.
- [12] D. Perez, S. Samothrakis, S. Lucas, and P. Rohlfshagen. “Rolling Horizon Evolution versus Tree Search for Navigation in Single-Player Real-Time Games”. In: *Proc. of the Conference on Genetic and Evolutionary Computation*. ACM Press, New York, NY, 2013, pp. 351–358.
- [13] D. Perez-Liebana, J. Dieskau, M. Hunermund, S. Mostaghim, and S. Lucas. “Open Loop Search for General Video Game Playing”. In: *Proc. of the Conference on Genetic and Evolutionary Computation*. ACM Press, New York, NY, 2015, pp. 337–344.
- [14] D. Perez-Liebana, J. Togelius, S. Samothrakis, T. Schaul, S. M. Lucas, A. Couetoux, J. Lee, C.-U. Lim, and T. Thompson. “The 2014 General Video Game Playing Competition”. In: *IEEE Trans. on Computational Intelligence and AI in Games* (2015).
- [15] T. Schaul. “A Video Game Description Language for Model-based or Interactive Learning”. In: *Proc. of the IEEE Conference on Computational Intelligence in Games*. IEEE Press, Piscataway, NJ, 2013, pp. 193–200.

# General General Game AI

Julian Togelius  
Tandon School of Engineering  
New York University  
New York, New York  
julian@togelius.com

Georgios N. Yannakakis  
Institute of Digital Games  
University of Malta  
Msida, Malta  
georgios.yannakakis@um.edu.mt

**Abstract**—Arguably the grand goal of artificial intelligence research is to produce machines with general intelligence: the capacity to solve multiple problems, not just one. Artificial intelligence (AI) has investigated the general intelligence capacity of machines within the domain of games more than any other domain given the ideal properties of games for that purpose: controlled yet interesting and computationally hard problems. This line of research, however, has so far focused solely on one specific way of which intelligence can be applied to games: playing them. In this paper, we build on the general game-playing paradigm and expand it to cater for all core AI tasks within a game design process. That includes general player experience and behavior modeling, general non-player character behavior, general AI-assisted tools, general level generation and complete game generation. The new scope for *general general game AI* beyond game-playing broadens the applicability and capacity of AI algorithms and our understanding of intelligence as tested in a creative domain that interweaves problem solving, art, and engineering.

## I. INTRODUCTION

By now, an active and healthy research community around computational and artificial intelligence (AI)<sup>1</sup> in games has existed for more than a decade — at least since the start of the IEEE Conference on Computational Intelligence and Games (CIG) and the Artificial Intelligence and Interactive Digital Entertainment (AIIDE) conference series in 2005. Before then, research has been ongoing about AI in board games since the dawn of automatic computing. Initially, most of the work published at IEEE CIG or AIIDE was concerned with learning to play a particular game as well as possible, or using search/planning algorithms to play a game as well as possible without learning. Gradually, a number of new applications for AI in games and for games in AI have come to complement the original focus on AI for playing games [1]. Papers on procedural content generation, player modeling, game data mining, human-like playing behavior, automatic game testing and so on have become commonplace within the community. There is also a recognition that all these research endeavors depend on each other [2]. Games appear to be an ideal domain for realizing several long-standing goals of AI

including affective computing [3], computational creativity [4] and ultimately *general* intelligence [5], [6].

However, almost all research projects in the game AI field are very *specific*. Most published papers describe a particular method — or a comparison of two or more methods — for performing a single task (playing, modeling, generating etc.) in a single game. This is problematic in several ways, both for the scientific value and for the practical applicability of the methods developed and studies made in the field. If an AI approach is only tested on a single task for a single game, how can we argue that is an advance in the scientific study of artificial intelligence? And how can we argue that it is a useful method for a game designer or developer, who is likely working on a completely different game than the method was tested on?

Within AI focused on playing games, we have seen the beginnings of a trend towards generality. The study of general artificial intelligence through games — general game playing — has seen a number of advancements in the last few years. Starting with the General Game Playing Competition, focusing on board games and similar discrete perfect information games, we now also have the Arcade Learning Environment and General Video Game AI Competition, which offer radically different takes on arcade video games. Advancements vary from the efforts to create game description languages suitable for describing games used for general game playing [7], [8], [9], [10] to the establishment of a set of general video game AI benchmarks [7], [11], [12] to the recent success of deep Q-learning in playing arcade games with human-level performance just by processing the screen's pixels [13].

While the general game playing is studied extensively and constitutes one of the key areas of game AI [2] we argue that the focus of generality solely with regards to the performance of game-playing agents is *very narrow* with respect to the spectrum of roles for general intelligence in games. The types of general intelligence required within game development include game and level design as well as player and experience modeling. Such skills touch upon a diverse set of cognitive and affective processes which have until now been ignored by general AI in games. For general game AI to be truly general, it needs to go beyond game playing while retaining the focus on addressing more than a single game or player.

In other words, we are arguing that we need to extend the generality of general game playing to all other ways in which

Authors contributed equally to this paper and are listed in alphabetical order.

<sup>1</sup>In the article, we will mostly use the terms “AI in games” or “game AI” to refer to the whole research field, including the various techniques commonly thought of as computational intelligence, machine learning, deep learning etc. “AI” just rolls off the tongue more easily.

AI is (or can be) applied to games. More specifically we are arguing that the field should move towards methods, systems and studies that incorporate three different types of generality:

- 1) *Game* generality. We should develop AI methods that work with not just one game, but with any game (within a given range) that the method is applied to.
- 2) *Task* generality. We should develop methods that can do not only one task (playing, modeling, testing etc) but a number of different, related tasks.
- 3) *User/designer/player* generality. We should develop methods that can model, respond to and/or reproduce the very large variability among humans in design style, playing style, preferences and abilities.

We further argue that all of this generality can be embodied into the concept of *general game design*, which can be thought of as a final frontier of AI research within games. We assume that the challenge of bringing together different types of skillsets and forms of intelligence within autonomous designers of games not only can advance our knowledge about human intelligence but also advance the capacity of general artificial intelligence. The paper briefly reviews the state of the art within each of the major roles an AI can take within game development, broadly following the classification of [2]. In particular AI can (1) take a non-human-player (*non-player*) role and either play games (in lieu of a human) or control the behavior of non-player characters (see Section II), (2) *model player* behavior and experience (see Section III), (3) *generate* content such as levels or complete games (see Section IV), or 4) *assist* in the design process through both the modeling of users (designers) and the generation of appropriate content (see Section V). For each of these roles we argue for the need of generality and we propose ways that this can be achieved. We conclude with a discussion on how to nudge the research field towards addressing general problems and methods.

It is important to note that we are not arguing that more focused investigations into methods for single tasks in single games are useless; these are often important as proofs-of-concept or industrial applications and they will continue to be important in the future, but there will be increasing need to validate such case studies in a more general context. We are also not envisioning that everyone will suddenly start working on general methods. Rather, we are positing generalizations as a long-term goal for our entire research community.

## II. GENERAL NON-PLAYERS

A large part of the research on AI for games is concerned with building AI (i.e a non-(human)player) for playing games, with or without a learning component. Historically, this has been the first and for a long time only approach to using AI in games. Even before the beginning of AI as a research field, algorithms were devised to play games effectively. For instance, Turing himself (re)invented the Minimax algorithm to play Chess even before he had a working digital computer [14].

For a long time, research on game-playing AI was focused on classic board games, and Chess was even seen as “the drosophila of AI” [15] — at least until we developed software

capable of playing Chess better than humans, at which point Chess-playing AI somehow seemed a less urgent problem. The fact that Chess became a less relevant problem once humans had been beaten itself points to the need for focusing on more general problems. The software that first exhibited superhuman Chess capability, Deep Blue, consisted of a Minimax algorithm with numerous Chess-specific modifications and a very highly tuned board evaluation function; the software was useless for anything else than playing Chess [16]. This led commentators at the time to argue that Deep Blue was “not really AI” after all [17]. The same argument could be made about AlphaGo, the AI that finally conquered the classic board game *Go* [18].

Even nowadays a large part of game AI research focuses on developing AI for playing games — either as effectively as possible, or in the style of humans (or a particular human), or with some other property [2]. Much of the research on playing videogames is organized around a number of competitions or common benchmarks. In particular, the IEEE CIG conference series hosts a respectable number of competitions, the majority of which focus on playing a particular game; popular competitions are built around games such as TORCS (a car racing game), *Super Mario Bros* (Nintendo, 1985), *Ms. Pac-Man* (Namco, 1982), *Unreal Tournament 2004* (Epic Games, 2004) and *StarCraft* (Blizzard Entertainment, 1998). These competitions are typically won by the submitted AI agent that plays the game best. What can be observed in several of these competitions, is that when the same competition is run multiple years there is indeed an improvement in performance, but not necessarily an improvement in the sophistication of the AI algorithms or their centrality to the submitted agent. In fact, the opposite trend can sometimes be discerned. For example, the simulated car racing competition started out with several competitors submitting car-driving agents which were to a large extent based on machine learning (e.g. neuroevolution). In subsequent years, however, these were outperformed by agents consisting of a large amount of hand-coded domain-specific rules, with any learning relegated to a supporting role [19], [20]. Similarly, the *StarCraft* competition has seen a number of AI-based agents performing moderately well, but the winner in several rounds of the competition consists almost entirely of hand-crafted strategies with almost no presence of what would normally be considered AI algorithms, and certainly no applicability outside *StarCraft* [21].

### A. Gameplaying

Interestingly, the problem of playing games is the one that has been most generalized so far. There already exist at least three serious benchmarks or competitions attempting to pose the problem of playing games *in general*, each in its own imperfect way. The first of these is the General Game Playing Competition, often abbreviated GGP [7]. This competition has been running for more than ten years, and is based on a special-purpose game description language useful for encoding board game-like games, with discrete world state and in most cases perfect information. The submitted agents get access to the complete source code of the games they are tested on.



Every time the competition is run a few games are hand-crafted for testing new games. The Arcade Learning Environment (ALE) is instead built on an emulator of the Atari 2600 game console and includes a library of classic games [12]. Agents are only given a feed of the raw screen output. Compared to GGP, ALE has the advantage of using real video games, but the disadvantage that all games are known and creating new games requires very considerable effort. The General Video Game AI Competition (GVGAI) combines the focus on video games (from ALE) with the approach to build the competition games in a description language which allows new games to be created for each competition [9], [8], [11]. Currently, around 80 games are implemented, and for every competition 10 new games are implemented. Most games are adaptations of classic 80's arcade games, but the long-term goal is for new games to be generated automatically [22]. Agents are given access to a partial game state observation and a complete forward model.

The results from these competitions so far indicate that general purpose search and learning algorithms by far outperform more domain-specific solutions and “clever hacks”. Somewhat simplified, we can say that variations of Monte Carlo Tree Search perform best on GVGAI and GGP, and for ALE (where no forward model is available so learning a policy for each game is necessary) reinforcement learning with deep networks performs best [13]. This is a very marked difference to the results of the game-specific competitions, which as discussed above tend to favor domain-specific solutions.

While these are each laudable initiatives and currently the focus of much research, in the future we will need to expand the scope of these competitions and benchmarks considerably, including expanding the range of games available to play and the conditions under which gameplay happens. We need game playing benchmarks and competitions capable of expressing any kind of game, including puzzle games, 2D arcade games, text adventures, 3D action-adventures and so on; this is the best way to test general AI capacities and reasoning skills. We also need a number of different ways of interfacing with these games — there is room both for benchmarks that give agents no information beyond the raw screen data but give them hours to learn how to play the game, and those that give agents access to a forward model and perhaps the game code itself, but expects them to play any game presented to them with no time to learn. These different modes test different AI capabilities and tend to privilege different types of algorithms. It is worth noting that the GVGAI competition is currently expanding to different types of playing modes, and has a long-term goal to include many more types of games [22].

We also need to differentiate away from just measuring how to play games optimally. In the past, several competitions have focused on agents that play games in a human-like manner; these competitions have been organized similarly to the classic Turing test [23], [24]. Playing games in a human-like manner is important for a number of reasons, such as being able to test levels and other game content as part of search-based generation, and to demonstrate new content to players. So far, the question of how to play games in a

human-like manner *in general* is mostly unexplored; some preliminary work is reported in [25]. Making progress here will likely involve modeling how humans play games in general, including characteristics such as short-term memory, reaction time and perceptual capabilities, and then translating these characteristics to playing style in individual games.

### B. Non-player Behavior

Many games have non-player characters (NPCs), and AI can help in making NPCs believable, human-like, social and expressive. Years of active research have been dedicated on this task within the fields of affective computing and virtual agents. The usual approach followed is the construction of top-down agent architectures that represent various cognitive, social, emotive and behavioral abilities. The focus has traditionally been on both the modeling of the agents behavior but also on its appropriate expression under particular contexts. A popular way for constructing a computational model of agent behavior is to base it on a theoretical cognitive model such as the Belief-Desire-Intention agent model [26] and the OCC model [27], [28], [29] which attempts to effect human-like decision making, appraisal and coping mechanisms dependent on a set of perceived stimuli. The use of such character models has been dominant in the domains of intelligent tutoring systems [30], embodied conversational agents [29], and affective agents [31] for educational and health purposes. Similar types of architectures for believable and social agents exist in games such as *Facade* [32], *Prom Week* [33], *World of Minds* [34] and *Crystal Island* [35]. Research in non-player character behavior in games is naturally interwoven with research in computational and interactive narrative [36], [32], [37], [38] and virtual cinematography [39], [40], [41].

One would expect that characters in games would be able to perform well under any context and game (seen or unseen) in similar ways humans do. Not only would that be a far more effective approach for agent modeling but it would also advance our understanding about general emotive, social and behavioral patterns. However, as with the other uses of AI in games, the construction of agent architectures for behavior modeling and expression is heavily dependent on particular game contexts and *specific* to (and optimized for) a particular game. While a number of studies within affective agents focus on domain-independent (general) emotive models [31] we are far from obtaining general, context-free, “plug-n-play” computational models for agents that are applicable across games, game genres and players. The vision here is that we could create general NPCs, that could easily be dropped into any given game and adapt (autonomously and/or with designer guidance) to the requirements of a particular game, so that they can behave believably and effectively in their new context.

Clearly there are general patterns of rational and (socially) believable behavior that can be detected across games and players. An NPC in *Prom Week*, for instance, should be able to transfer aspects of its social intelligence to e.g. *Facade*; but how much of such patterns are relevant for a platformer of a first-person shooter NPC? This is an open research

question. To make a paradigm shift towards generality we argue that we need to focus on aspects of a top-down agent architecture that, by nature, are more general than others. Personality, for instance, can be modeled in an abstract, domain-independent way [42] while moods — compared to emotions — define longer lasting and less specific notions [43] that could be modeled in a context-independent way [28]. Emotions are more specific and domain-dependent aspects of a computational agent architecture, as they are heavily context-dependent. It has been suggested, however, that personality and emotion are heavily interlinked and only differentiated by time and duration since personality can be viewed as the seamless expression of emotion [44]. In that regard general emotive patterns across games and players can be identified if general context-independent features that characterize general behavior can be extracted and used for modeling NPC behavior; these include generic features such as winning, losing, achieving rewards as well as progression or tension curves across games. These features can be either manually designed or machine learned from annotated player data.

### III. GENERAL PLAYER EXPERIENCE MODELING

It stands to reason that general intelligence implies (and is tightly coupled with) general emotional intelligence [45]. The ability to recognize human behavior and emotion is a complex yet critical task for human communication. Throughout evolution, we have developed particular forms of advanced cognitive, emotive and social skills to address this challenge. Beyond these skills, we also have the capacity to detect affective patterns across people with different moods, cultural backgrounds and personalities. This generalization ability also extends, to a degree, across contexts and social settings.

Despite their importance, the characteristics of social intelligence have not yet been transferred to AI in the form of general emotive, cognitive or behavioral models. While research in affective computing [3] has reached important milestones such as the capacity for real-time emotion recognition [46] — which can be faster than humans under particular conditions — all key findings suggest that any success of affective computing is heavily dependent on the domain, the task at hand and the context in general. This *specificity* limitation is particularly evident in the domain of games [47] as most of work in modeling player experience focuses on particular games, under particular and controlled conditions within particular small sets of players (see [48], [49], [50], [51] among many).

For AI in games to be general beyond game-playing it needs to be able to recognize general emotional and cognitive-behavioral patterns. This is essentially AI that can detect context-free emotive and cognitive reactions and expressions across context and builds general computational models of human behavior and experience which are grounded in a general golden standard of human behavior. So far we have only seen a few proof-of-concept studies in this direction. Early work in the game AI field focused on the ad-hoc design of general metrics of player interest that were tested across different predator-prey games [52], [53]. A more recent example is the work

of Martinez et al. [54] in which physiological predictors of player experience were tested for their ability to capture player experience across two dissimilar games: a predator-prey game and a racing game. The findings of that study suggest that such features do exist. Shaker et al. [55] later used the same approach with different games.

Another study by Martinez et al. on deep multimodal fusion can be seen as an embryo for further research in this direction [51]. Various modalities of player input such as player metrics, skin conductance and heart activity, have been fused using deep architectures which were pretrained using autoencoders. Even though that study was rather specific to a particular game, its deep fusion methodology can be expanded across variant data corpora — such as the DEAP [56] and the platformer experience [57] datasets — and player metrics datasets that are openly available such as the game trace archive [58]. Discovering entirely new representations of player behavior and emotive manifestations across games, modalities of data, and player types is a first step towards achieving general player modeling. Such representations can, in turn, be used as the basis for deriving the *ground truth* of user experience in games.

### IV. GENERAL CONTENT GENERATION

The study of procedural content generation (PCG) [59] for the design of game levels has reached a certain extent of maturity and is, by far, the most popular domain for the application of PCG algorithms and approaches (e.g. see [60], [2], [48] among many). In this section we first discuss this popular facet of computational game creativity and then connect it to the overall aim of general complete game generation.

#### A. Level Generation

Levels have been generated for various game genres such as dungeon-crawlers [61], [62], horror games [63], space-shooters [64], first-person shooters [65], [66], and platformers [49]. Arguably the platformer genre — through the Mario AI Framework, which builds on a clone of Super Mario Bros [67], [68] — can be characterized as the “*drosophila* of PCG research”. A number of approaches such as constructive methods, search-based PCG [60], experience-driven PCG [48], solver-based PCG [69], data-driven PCG [70], [71], [72] or mixed-initiative PCG [73], [74] have been used for the creation of platformer levels in the Mario AI Framework with algorithms varying from simple multi-pass processes [75] to evolving grammars [76], exhaustive search on crowdsourced models of experience [77], and constraint solvers such as answer set programming [73]. What is common in all of the above studies is their *specificity* and strong dependency of the representation chosen onto the game genre examined. In particular for the Mario AI Framework, the focus on a single level generation problem has been very much a mixed blessing: it has allowed for the proliferation and simple comparison of multiple approaches to solving the same problem, but has also led to a clear overfitting of methods. Even though some limited generalization is expected within game levels of the

same genre the level generators that have been explored so far clearly do not have the capacity of *general* level design.

As with the other sub-tasks of game design discussed in this paper we argue that there needs to be a shift in how level generation is viewed. The obvious change of perspective is to create *general level generators* — level generators with general intelligence that can generate levels for any game (within a specified range). That would mean that levels are generated successfully across game genres and players and that the output of the generation process is a that is meaningful and playable well as entertaining for the player. Further, a general level generator should be able to coordinate the generative process with the other computational game designers who are responsible for the other parts of the game design.

To achieve general level design intelligence algorithms are required to capture as much of the level design space as possible at different representation resolutions. We can think of representation learning approaches such as deep autoencoders [78] capturing core elements of the level design space and fusing variant game genres within a sole representation — as already showcased by a few studies in the PCG area e.g.in [79]. A related effort is the Video Game Level Corpus [80] which aims to provide a set of game levels across multiple games and genres which can be used for training level generators for data-driven procedural content generation.

The first attempt to create a benchmark for general level generation has recently been launched in the form of the Level Generation Track of the GVGAI competition. In this competition track, competitors submit level generators capable of generating levels for unseen games. The generators are then supplied with the description of several games, and produce levels which are judged by human judges [81]. Initial results suggest that constructing competent level generators that can produce levels for any game is much more challenging than constructing competent level generators for a single game.

### B. Game Generation

While level generation, as discussed above, is one of the main examples of procedural content generation, there are many other aspects (or “facets”) of games that can be generated. These include visuals, such as textures and images; narrative, such as quests and backstories; audio, such as sound effects and music; and of course the generation of all kinds of things that go into game levels, such as items, weapons, enemies and personalities [82], [59]. However, an even greater challenge is the generation of complete games, including some or all of these facets together with the rules of the game.

There have been several attempts to generate games, including their rules. These include approaches based on artificial evolution [83], [84], [85], [86], [87], [88], attempts based on constraint satisfaction [89], [69], [90] and attempts based on matching pattern databases [91], [92]. Some of these attempts have “only” generated the game rules, whereas others have included some aspects of graphics or theming of the game. We are not aware of any approach to generating games that tries to generate more than two of the facets of games listed

above. We are also not aware of any game generation system that even tries to generate games of more than one genre. Multi-faceted generation systems like *Sonancia* [63], [93] co-generate horror game levels with corresponding soundscapes but do not cater for the generation of rules.

It is clear that the very domain-limited and facet-limited aspects of current game generation systems result from intentionally limiting design choices in order to make the very difficult problem of generating complete games tractable. Yet, in order to move beyond what could be argued to be toy domains and start to fulfill the promise of game generation, we need systems that can generate multiple facets of games at the same time, and that can generate games of different kinds.

A few years ago, something very much like general game generation was outlined as the challenges of “multi-content, multi-domain PCG” and “generating complete games” in another vision paper co-authored by a number of researchers active in the game AI and PCG communities [94]. It is interesting to note that there has not seemingly been any attempt to create more general game generators since then, perhaps due to the complexity of the task. Currently the only genre for which generators have been built that can generate high-quality games is abstract board games; once more genres have been “conquered”, we hope that the task of building more general level generators can begin.

## V. GENERAL AI-ASSISTED GAME DESIGN TOOLS

The area of AI-assisted game design tools has seen significant research interest in recent years [2] with contributions mainly on the level design task [74], [73], [95], [96], [97], [98], [99]. All tools however remain *specific* to the task they were designed for and their underlying AI focuses on understanding the design process [74], on the generation of a specific facet (or domain) within a game [73] or on both [95].

To illustrate the problems with game-specificity: The authors have demonstrated AI-assisted game design tools to game developers outside academia numerous times, and the feedback have often been that the developers would want something like the demonstrated tools—for their own games. For example, *Ropossum* [96], [100] can greatly assist in designing levels for *Cut the Rope*, but that game is already released and has plenty of levels. Meanwhile, a game developer working on another game, even another physics puzzler, is not helped by *Ropossum* and might not have the time or knowledge to implement the ideas behind it for their own game.

General intelligence is required for tools as much it is required for the other areas of game artificial intelligence. Tools equipped with general capacities to assist humans across game tasks (such as level and audio design) and games genres but also learn to be general across design styles and preferences can only empower the creative process of game development at large. To be general a tool needs to be able to recognize general tasks and procedures during the game design process. An obvious direction towards designer-general tools is through the computational modeling of designers [95] across different tasks for identifying general design patterns as well

as personal aesthetic values, styles and procedures. That can be achieved to a degree through imitation learning and sequence mining [101] techniques resulting in designer models that are general across users of the tool. Tools can be general across games too. A level design assistive tool, for instance, can be trained to identify common successful design patterns across game levels of variant genres; in this case levels can be represented as 2D or 3D maps enriched with item placement and playtrace information that can be fused using e.g. deep autoencoders which has been proved a successful method to combine modalities of different resolution and type (such as time series vs. discrete events) [102], [51]. Finally tools can be general across game design tasks. For instance, a level tool would be able to recommend good sound effects for the level it just co-designed with a human designer if it is equipped with a transmedia (level to audio) representation as e.g. in the work of Horn et al. [103]. Such representations could potentially be machine-learned from existing level-audio patterns.

One path towards building general AI-assisted game design tools is to build a tool that works with a general framework, being able to work on any game expressed in that framework. Such an effort is currently underway for the General Video Game AI framework, for games expressed in VGDL; early work on this project explores how design patterns can be recommended between games [104].

## VI. THE ROAD AHEAD AND HOW TO STAY ON PATH

In this paper we have argued that the general intelligence capacity of machines needs to be both explored and exploited in its full potential (1) across the different *tasks* that exist within the game design and development process, including but absolutely no longer limited game playing; (2) across different *games* within the game design space and; (3) across different *users* (players or designers) of AI. We claim that, thus far, we have underestimated the potential for general AI within games. We also claim that the currently dominant practice of only designing AI for a specific task within a specific domain will eventually be detrimental to game AI research as algorithms, methods and epistemological procedures will remain specific to the task at hand. As a result, we will not be manage to push the boundaries of AI and exploit its full capacity for game design. We are inspired by the general game-playing paradigm and the recent successes of AI algorithms in that domain and suggest that we become less specific about all subareas of the game AI field including player modeling, emotive expression, game generation and AI-assisted design tools. Doing so would allow us to detect and mimic different general cognitive and emotive skills of humans when designing games — a creative task that fuses problem solving, artwork and engineering skills.

It might be worth noting that we are not alone in seeing this need. For example, Zook argues for the use of various game-related tasks (not just game playing) to be used in artificial general intelligence research [105]. It also worth noting, again, that we are not advocating that all research within the CI/AI in games field focuses on generality right now; studies on

particular games and particular tasks are still valuable, given how little we still understand and can do. But over time, we predict that more and more research will focus on generality across tasks, games and users, because it is in the general problems the interesting research questions of the future lay.

The path towards achieving general game artificial intelligence is still largely unexplored. For AI to become less specific — yet remain relevant and useful for game design — we envision a number of immediate steps that could be taken: first, and foremost the game AI community needs to adopt an *open-source* accessible strategy so that methods and algorithms developed across the different tasks are shared among researchers for the advancement of this research area. Venues such as the current game AI research portal<sup>2</sup> could be expanded and used to host successful methods and algorithms. For the algorithms and methods to be of direct use particular technical specifications need to be established — e.g. such as those established within game-based AI benchmarks — which will maximize the interoperability among the various tools and elements submitted. Examples of benchmarked specifications for the purpose of general game AI research include the general video game description language (VGDL) and the puzzle game engine PuzzleScript<sup>3</sup>. Finally, following the GVGAI competition paradigm, we envision a new set of competitions rewarding general player models, NPC models, AI-assisted tools and game generation techniques. These competitions would further motivate researchers to work in this exciting research area and enrich the database of open-access interoperable methods and algorithms directly contributing to the state of the art in computational general game design.

## REFERENCES

- [1] G. N. Yannakakis, “Game AI revisited,” in *Proceedings of the 9th conference on Computing Frontiers*. ACM, 2012, pp. 285–292.
- [2] G. N. Yannakakis and J. Togelius, “A Panorama of Artificial and Computational Intelligence in Games,” 2014.
- [3] R. W. Picard and R. Picard, *Affective computing*. MIT press Cambridge, 1997, vol. 252.
- [4] A. Liapis, G. N. Yannakakis, and J. Togelius, “Computational game creativity,” in *Proceedings of the 5th International Conference on Computational Creativity*, 2014.
- [5] J. Laird and M. VanLent, “Human-level ai’s killer application: Interactive computer games,” *AI magazine*, vol. 22, no. 2, p. 15, 2001.
- [6] T. Schaul, J. Togelius, and J. Schmidhuber, “Measuring intelligence through games,” *arXiv preprint arXiv:1109.1314*, 2011.
- [7] M. Genesereth, N. Love, and B. Pell, “General game playing: Overview of the AAAI competition,” *AI Magazine*, vol. 26, no. 2, pp. 62–72, 2005.
- [8] T. Schaul, “A video game description language for model-based or interactive learning,” in *Proceedings of the 2013 IEEE Conference on Computational Intelligence in Games*, 2013, pp. 1–8.
- [9] M. Ebner, J. Levine, S. M. Lucas, T. Schaul, T. Thompson, and J. Togelius, “Towards a video game description language,” *Dagstuhl Follow-Ups*, vol. 6, 2013.
- [10] C. Martens, “Ceptre: A language for modeling generative interactive systems,” in *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.
- [11] D. Perez, S. Samothrakis, J. Togelius, T. Schaul, S. Lucas, A. Couëtoux, J. Lee, C.-U. Lim, and T. Thompson, “The 2014 general video game playing competition,” 2015.

<sup>2</sup><http://www.aigameresearch.org/>

<sup>3</sup><http://www.puzzlescript.net/>

- [12] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *arXiv preprint arXiv:1207.4708*, 2012.
- [13] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [14] A. M. Turing, M. Bates, B. Bowden, and C. Strachey, "Digital computers applied to games," *Faster than thought*, vol. 101, 1953.
- [15] N. Ensmenger, "Is chess the drosophila of ai? a social history of an algorithm," *Social studies of science*, p. 0306312711424596, 2011.
- [16] M. Campbell, A. J. Hoane, and F.-h. Hsu, "Deep blue," *Artificial intelligence*, vol. 134, no. 1, pp. 57–83, 2002.
- [17] M. Newborn, *Kasparov versus Deep Blue: Computer chess comes of age*. Springer Science & Business Media, 2012.
- [18] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
- [19] J. Togelius, S. Lucas, H. D. Thang, J. M. Garibaldi, T. Nakashima, C. H. Tan, I. Elhanany, S. Berant, P. Hingston, R. M. MacCallum *et al.*, "The 2007 ieeec simulated car racing competition," *Genetic Programming and Evolvable Machines*, vol. 9, no. 4, pp. 295–329, 2008.
- [20] D. Loiacono, P. L. Lanzi, J. Togelius, E. Onieva, D. A. Pelta, M. V. Butz, T. D. Lönneker, L. Cardamone, D. Perez, Y. Sáez *et al.*, "The 2009 simulated car racing championship," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 2, no. 2, pp. 131–147, 2010.
- [21] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game ai research and competition in starcraft," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 5, no. 4, pp. 293–311, 2013.
- [22] D. Perez-Liebana, S. Samothrakos, J. Togelius, T. Schaul, and S. M. Lucas, "General video game ai: Competition, challenges and opportunities," in *Proceedings of AAAI*, 2016.
- [23] P. Hingston, "A new design for a turing test for bots," in *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*. IEEE, 2010, pp. 345–350.
- [24] N. Shaker, J. Togelius, G. N. Yannakakis, L. Poovanna, V. S. Ethiraj, S. J. Johansson, R. G. Reynolds, L. K. Heether, T. Schumann, and M. Gallagher, "The turing test track of the 2012 mario ai championship: entries and evaluation," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013, pp. 1–8.
- [25] A. Khalifa, A. Isaksen, J. Togelius, and A. Nealen, "Modifying mcts for human-like general video game playing," in *Proceedings of IJCAI*, 2016.
- [26] A. S. Rao and M. P. Georgeff, "Modeling rational agents within a bdi-architecture," *KR*, vol. 91, pp. 473–484, 1991.
- [27] A. Ortony, G. L. Clore, and A. Collins, *The cognitive structure of emotions*. Cambridge university press, 1990.
- [28] A. Egges, S. Kshirsagar, and N. Magnenat-Thalmann, "Generic personality and emotion simulation for conversational agents," *Computer animation and virtual worlds*, vol. 15, no. 1, pp. 1–13, 2004.
- [29] E. André, M. Klesen, P. Gebhard, S. Allen, and T. Rist, "Integrating models of personality and emotions into lifelike characters," in *Affective interactions*. Springer, 2000, pp. 150–165.
- [30] C. Conati, "Intelligent tutoring systems: New challenges and directions," in *IJCAI*, vol. 9, 2009, pp. 2–7.
- [31] J. Gratch and S. Marsella, "A domain-independent framework for modeling emotion," *Cognitive Systems Research*, vol. 5, no. 4, pp. 269–306, 2004.
- [32] M. Mateas and A. Stern, "Façade: An experiment in building a fully-realized interactive drama," in *Game Developers Conference*, vol. 2, 2003.
- [33] J. McCoy, M. Treanor, B. Samuel, M. Mateas, and N. Wardrip-Fruin, "Prom week: social physics as gameplay," in *Proceedings of the 6th International Conference on Foundations of Digital Games*. ACM, 2011, pp. 319–321.
- [34] M. P. Eladhari and M. Mateas, "Semi-autonomous avatars in world of minds: A case study of ai-based game design," in *Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology*. ACM, 2008, pp. 201–208.
- [35] J. Rowe, B. Mott, S. McQuiggan, J. Robison, S. Lee, and J. Lester, "Crystal island: A narrative-centered learning environment for eighth grade microbiology," in *workshop on intelligent educational games at the 14th international conference on artificial intelligence in education, Brighton, UK, 2009*, pp. 11–20.
- [36] D. Thue, V. Bulitko, M. Spetch, and E. Wasylishen, "Interactive storytelling: A player modelling approach," in *AIIDE*, 2007, pp. 43–48.
- [37] M. O. Riedl and V. Bulitko, "Interactive narrative: An intelligent systems approach," *AI Magazine*, vol. 34, no. 1, p. 67, 2012.
- [38] R. M. Young, M. O. Riedl, M. Branly, A. Jhala, R. Martin, and C. Saretto, "An architecture for integrating plan-based behavior generation with interactive game environments," *Journal of Game Development*, vol. 1, no. 1, pp. 51–70, 2004.
- [39] D. K. Elson and M. O. Riedl, "A lightweight intelligent virtual cinematography system for machinima production," in *AIIDE*, 2007, pp. 8–13.
- [40] A. Jhala and R. M. Young, "Cinematic visual discourse: Representation, generation, and evaluation," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 2, no. 2, pp. 69–81, 2010.
- [41] P. Burelli, "Virtual cinematography in games: investigating the impact on player experience," *Foundations of Digital Games*, 2013.
- [42] H. J. Eysenck, *Biological dimensions of personality*. Guilford Press, 1990.
- [43] S. Kshirsagar, "A multilayer personality model," in *Proceedings of the 2nd international symposium on Smart graphics*. ACM, 2002, pp. 107–115.
- [44] D. Moffat, "Personality parameters and programs," in *Creating personalities for synthetic actors*. Springer, 1997, pp. 120–165.
- [45] J. D. Mayer and P. Salovey, "The intelligence of emotional intelligence," *Intelligence*, vol. 17, no. 4, pp. 433–442, 1993.
- [46] Z. Zeng, M. Pantic, G. I. Roisman, and T. S. Huang, "A survey of affect recognition methods: Audio, visual, and spontaneous expressions," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, no. 1, pp. 39–58, 2009.
- [47] G. N. Yannakakis and A. Paiva, "Emotion in games," *Handbook on Affective Computing*, pp. 459–471, 2014.
- [48] G. N. Yannakakis and J. Togelius, "Experience-driven procedural content generation," *Affective Computing, IEEE Transactions on*, vol. 2, no. 3, pp. 147–161, 2011.
- [49] N. Shaker, S. Asteriadis, G. N. Yannakakis, and K. Karpouzis, "A game-based corpus for analysing the interplay between game context and player experience," in *Affective Computing and Intelligent Interaction*. Springer Berlin Heidelberg, 2011, pp. 547–556.
- [50] —, "Fusing visual and behavioral cues for modeling user experience in games," *Cybernetics, IEEE Transactions on*, vol. 43, no. 6, pp. 1519–1531, 2013.
- [51] H. P. Martínez and G. N. Yannakakis, "Deep multimodal fusion: Combining discrete events and continuous signals," in *Proceedings of the 16th International Conference on Multimodal Interaction*. ACM, 2014, pp. 34–41.
- [52] G. Yannakakis and J. Hallam, "A generic approach for obtaining higher entertainment in predator/prey computer games," *Journal of Game Development*, 2005.
- [53] G. N. Yannakakis and J. Hallam, "A generic approach for generating interesting interactive pac-man opponents," in *IEEE CIG, year=2005*.
- [54] H. Perez Martínez, M. Garbarino, and G. Yannakakis, "Generic physiological features as predictors of player experience," *Affective Computing and Intelligent Interaction*, pp. 267–276, 2011.
- [55] N. Shaker, M. Shaker, and M. Abou-Zleikha, "Towards generic models of player experience," in *Proceedings, the Eleventh Aaai Conference on Artificial Intelligence and Interactive Digital Entertainment (aiide-15)*. AAAI Press, 2015.
- [56] S. Koelstra, C. Mühl, M. Soleymani, J.-S. Lee, A. Yazdani, T. Ebrahimi, T. Pun, A. Nijholt, and I. Patras, "Deap: A database for emotion analysis; using physiological signals," *Affective Computing, IEEE Transactions on*, vol. 3, no. 1, pp. 18–31, 2012.
- [57] K. Karpouzis, G. N. Yannakakis, N. Shaker, and S. Asteriadis, "The platformer experience dataset," in *Affective Computing and Intelligent Interaction (ACII), 2015 International Conference on*. IEEE, 2015, pp. 712–718.
- [58] Y. Guo and A. Iosup, "The game trace archive," in *Proceedings of the 11th Annual Workshop on Network and Systems Support for Games*. IEEE Press, 2012, p. 4.
- [59] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2015.

- [60] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," 2011.
- [61] R. van der Linden, R. Lopes, and R. Bidarra, "Procedural generation of dungeons," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 6, no. 1, pp. 78–89, 2014.
- [62] L. Johnson, G. N. Yannakakis, and J. Togelius, "Cellular automata for real-time generation of infinite cave levels," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM, 2010, p. 10.
- [63] P. Lopes, A. Liapis, and G. N. Yannakakis, "Sonancia: Sonification of procedurally generated game levels," in *Proceedings of the ICCG workshop on Computational Creativity & Games*, 2015.
- [64] A. K. Hoover, W. Cachia, A. Liapis, and G. N. Yannakakis, "Audioinspace: Exploring the creative fusion of generative audio, visuals and gameplay," in *Evolutionary and Biologically Inspired Music, Sound, Art and Design*. Springer International Publishing, 2015, pp. 101–112.
- [65] L. Cardamone, G. N. Yannakakis, J. Togelius, and P. L. Lanzi, "Evolving interesting maps for a first person shooter," in *Applications of Evolutionary Computation*. Springer, 2011, pp. 63–72.
- [66] W. Cachia, A. Liapis, and G. N. Yannakakis, "Multi-level evolution of shooter levels," in *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.
- [67] J. Togelius, N. Shaker, S. Karakovskiy, and G. N. Yannakakis, "The mario ai championship 2009-2012," *AI Magazine*, vol. 34, no. 3, pp. 89–92, 2013.
- [68] B. Horn, S. Dahlskog, N. Shaker, G. Smith, and J. Togelius, "A comparative evaluation of procedural level generators in the mario ai framework," 2014.
- [69] A. M. Smith and M. Mateas, "Answer set programming for procedural content generation: A design space approach," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 3, no. 3, pp. 187–200, 2011.
- [70] A. Summerville and M. Mateas, "Super mario as a string: Platformer level generation via lstm," *arXiv preprint arXiv:1603.00930*, 2016.
- [71] M. Guzdial and M. Riedl, "Toward game level generation from gameplay videos," *arXiv preprint arXiv:1602.07721*, 2016.
- [72] S. Snodgrass and S. Ontanon, "A hierarchical mdmc approach to 2d video game map generation," in *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.
- [73] G. Smith, J. Whitehead, and M. Mateas, "Tanagra: A mixed-initiative level design tool," in *Proceedings of the Fifth International Conference on the Foundations of Digital Games*. ACM, 2010, pp. 209–216.
- [74] G. N. Yannakakis, A. Liapis, and C. Alexopoulos, "Mixed-initiative co-creativity," in *Proceedings of the 9th Conference on the Foundations of Digital Games*, 2014.
- [75] N. Shaker, J. Togelius, G. N. Yannakakis, B. Weber, T. Shimizu, T. Hashiyama, N. Sorenson, P. Pasquier, P. Mawhorter, G. Takahashi *et al.*, "The 2010 mario ai championship: Level generation track," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 3, no. 4, pp. 332–347, 2011.
- [76] N. Shaker, M. Nicolau, G. N. Yannakakis, J. Togelius, and M. O. Neill, "Evolving levels for super mario bros using grammatical evolution," in *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*. IEEE, 2012, pp. 304–311.
- [77] N. Shaker, G. N. Yannakakis, and J. Togelius, "Crowdsourcing the aesthetics of platform games," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 5, no. 3, pp. 276–290, 2013.
- [78] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 1096–1103.
- [79] A. Liapis, H. P. Martinez, J. Togelius, and G. N. Yannakakis, "Transforming exploratory creativity with delenox," in *Proceedings of the Fourth International Conference on Computational Creativity*. AAAI Press, 2013, pp. 56–63.
- [80] A. J. Summerville, S. Snodgrass, M. Mateas, and S. O. Villar, "The vglc: The video game level corpus," *arXiv preprint arXiv:1606.07487*, 2016.
- [81] A. Khalifa, D. Perez-Liebana, S. M. Lucas, and J. Togelius, "General video game level generation," in *Proceedings of IJCAI*, 2016.
- [82] A. Liapis, G. N. Yannakakis, and J. Togelius, "Computational game creativity," in *Proceedings of the Fifth International Conference on Computational Creativity*, vol. 4, no. 1, 2014.
- [83] C. Browne, "Automatic generation and evaluation of recombination games," Ph.D. dissertation, Queensland University of Technology, 2008.
- [84] J. Togelius and J. Schmidhuber, "An experiment in automatic game design," in *Proceedings of the 2008 IEEE Symposium on Computational Intelligence and Games*, 2008, pp. 111–118.
- [85] T. S. Nielsen, G. A. Barros, J. Togelius, and M. J. Nelson, "Towards generating arcade game rules with vgd1," in *Computational Intelligence and Games (CIG), 2015 IEEE Conference on*. IEEE, 2015, pp. 185–192.
- [86] M. Cook and S. Colton, "Multi-faceted evolution of simple arcade games," in *CIG*, 2011, pp. 289–296.
- [87] J. M. Font, T. Mähmann, D. Manrique, and J. Togelius, "Towards the automatic generation of card games through grammar-guided genetic programming," in *FDG*, 2013, pp. 360–363.
- [88] J. Kowalski and M. Szykuła, "Evolving chess-like games using relative algorithm performance profiles," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2016, pp. 574–589.
- [89] M. J. Nelson and M. Mateas, "Towards automated game design," in *AI\* IA 2007: Artificial Intelligence and Human-Oriented Computing*. Springer, 2007, pp. 626–637.
- [90] A. Zook and M. O. Riedl, "Automatic game design via mechanic generation," in *AAAI*, 2014, pp. 530–537.
- [91] M. Treanor, B. Blackford, M. Mateas, and I. Bogost, "Game-o-matic: Generating videogames that represent ideas," in *Procedural Content Generation Workshop at the Foundations of Digital Games Conference*, 2012.
- [92] M. Cook and S. Colton, "Ludus ex machina: Building a 3d game designer that competes alongside humans," in *Proceedings of the 5th International Conference on Computational Creativity*, 2014.
- [93] P. Lopes, A. Liapis, and G. N. Yannakakis, "Framing tension for game generation," in *Proceedings of the Seventh International Conference on Computational Creativity*, 2016.
- [94] J. Togelius, A. J. Champandard, P. L. Lanzi, M. Mateas, A. Paiva, M. Preuss, and K. O. Stanley, "Procedural content generation in games: Goals, challenges and actionable steps," *Dagstuhl Follow-Ups*, vol. 6, 2013.
- [95] A. Liapis, G. N. Yannakakis, and J. Togelius, "Designer modeling for personalized game content creation tools," in *Proceedings of the AIIDE Workshop on Artificial Intelligence & Game Aesthetics*, 2013.
- [96] N. Shaker, M. Shaker, and J. Togelius, "Ropossum: An authoring tool for designing, optimizing and solving cut the rope levels," in *AIIDE*, 2013.
- [97] E. Butler, A. M. Smith, Y.-E. Liu, and Z. Popovic, "A mixed-initiative tool for designing level progressions in games," in *Proceedings of the 26th annual ACM symposium on User interface software and technology*. ACM, 2013, pp. 377–386.
- [98] D. Karavolos, A. Bouwer, and R. Bidarra, "Mixed-initiative design of game levels: Integrating mission and space into level generation," in *Proceedings of the 10th International Conference on the Foundations of Digital Games*, 2015.
- [99] B. Kybartas and R. Bidarra, "A semantic foundation for mixed-initiative computational storytelling," in *Interactive Storytelling*. Springer, 2015, pp. 162–169.
- [100] N. Shaker, M. Shaker, and J. Togelius, "Evolving playable content for cut the rope through a simulation-based approach," in *AIIDE*, 2013.
- [101] H. P. Martínez and G. N. Yannakakis, "Mining multimodal sequential patterns: a case study on affect detection," in *Proceedings of the 13th international conference on multimodal interfaces*. ACM, 2011, pp. 3–10.
- [102] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 689–696.
- [103] B. Horn, G. Smith, R. Masri, and J. Stone, "Visual information vases: Towards a framework for transmedia creative inspiration," in *Proceedings of the Sixth International Conference on Computational Creativity June*, 2015, p. 182.
- [104] T. Machado, I. Bravi, Z. Wang, J. Togelius, and A. Nealen, "Recommending game mechanics," in *Proceedings of the FDG Workshop on Procedural Content Generation*, 2016.
- [105] A. Zook, "Game AGI beyond Characters," in *Integrating Cognitive Architectures into Virtual Character Design*. IGI Global, 2016, pp. 266–293.



# A Holistic Approach for Semantic-Based Game Generation

Owen Sacco  
Institute of Digital Games  
University of Malta  
Msida, Malta  
owensacco@gmail.com

Antonios Liapis  
Institute of Digital Games  
University of Malta  
Msida, Malta  
antonios.liapis@um.edu.mt

Georgios N. Yannakakis  
Institute of Digital Games  
University of Malta  
Msida, Malta  
georgios.yannakakis@um.edu.mt

**Abstract**—The Web contains vast sources of content that could be reused to reduce the development time and effort to create games. However, most Web content is unstructured and lacks meaning for machines to be able to process and infer new knowledge. The *Web of Data* is a term used to describe a trend for publishing and interlinking previously disconnected datasets on the Web in order to make them more valuable and useful as a whole. In this paper, we describe an innovative approach that exploits Semantic Web technologies to automatically generate games by reusing Web content. Existing work on automatic game content generation through algorithmic means focuses primarily on a set of parameters within constrained game design spaces such as terrains or game levels, but does not harness the potential of already existing content on the Web for game generation. We instead propose a holistic and more generally-applicable game generation solution that would identify suitable Web information sources and enrich game content with semantic meta-structures.

## I. INTRODUCTION

Creating deep and largely non-linear games that are market-competent costs in time, effort and resources which often cannot be afforded by small-medium enterprises, especially by independent game development studios. As most of the tasks involved in developing games are labour- and creativity-intensive, our vision is to reduce software development effort and enhance design creativity by automatically generating novel and semantically-enriched content for games from Web sources. In this paper we envision a game generator that extracts information directly from the Web such as from wiki articles or images that are freely available. Following a semantic-based game generation approach not only can reduce the time and cost of game content creation but also directly contribute to web-informed yet unconventional game design.

Suppose a system where a user enters a description of a video game that s/he desires. The description could contain some keywords such as “*action adventure video game*”, or could contain a complete narrative that describes the whole plot of the game. The system would then generate a video game based on the user’s input by reusing Web content. Advanced features would be provided such as properties that the user can select prior to the system generating the game. An editor would also be provided for modifying the video game prototype after it is generated. All content generated from this system would be semantically enriched described

using standard structured meta-formats that would enable the content to be re-publishable on the Web for easy interlinking and consumption. This system would be beneficial for non-technical users without any background in developing digital games. For example by using this system, educators can generate educational games for their students about a particular subject without having to develop the game. Another example, health practitioners can generate a game by using this system for their patients to treat a particular disease. Moreover, this system would also be beneficial for experienced developers since it would reduce the time and effort for them to develop games by generating an unconventional set of playable game prototypes. Furthermore, our approach would publish semantically enriched game information that can be reused in other games.

Games are composed of different domains (or *facets*) that contribute to the game’s look, feel and experience [24]. These facets include visuals, audio, narrative, gameplay, game design and level design. Each facet can be regarded as an independent model containing specific content, and a game is created when each of these models are interlinked together based on the game’s requirements. Current work on automatic generation of content comprise of algorithms that generate limited in-game entities, such as *SpeedTree* [39] that generates trees and vegetation as part of the visuals facet, or the Ludi system [7] which generates game rules for two-player board games as part of the game design facet. Although such algorithms are beneficial for automatic generation of content, it is still rare that more than one domain is considered — e.g. in the work of Cook [10], Lopes et al. [26] and Riedl et al. [18] — requiring a substantial amount of manual effort by game developers to create content.

Inclusion of digital real-world data (e.g. recent news, real landscapes or historical events) in game environment is a practice used to increase the reality of scenarios and game play. Many flight simulators (e.g. *Flight Simulator X* (Microsoft Game Studios, 2006) or *Flight Pro Sim*<sup>1</sup>) include digital models of objects such as actual landscapes or airports and combine them with multiplayer mode to provide more realistic experiences. This trend is also visible in some of the sports

<sup>1</sup>Flight Pro Sim — <http://flightprosim.com>

games, such as the *SSX* series of games (EA Sports, 2000) where the NASA topography was used for the creation of snowboarding trails [30]. The inclusion of historical events is especially visible in strategic games where players can participate in scenarios resembling actual campaigns and immerse in history such as the *Call of Duty* (Activision, 2003) or the *Medal of Honor* (Electronic Arts, 1999) series. Moreover, games often include historic information within the plot that gamers can interact with, such as in the *Assassin's Creed* (Ubisoft, 2007) series, which contributes to a more immersive gaming experience. However, most of this content is currently created manually even though such information is widely available on the Web.

Web content is dispersed over the Internet in the form of blogs, microblogs, forums, wikis, social networks, review sites, and other Web applications which are currently disconnected from one another. The datasets created by these communities all contain information which can be used to generate or reuse content in games, but are not easily discoverable. The emerging Web of Data trend [6], where datasets are published in a standard form for easy interlinking, enables to essentially view the whole Web as one massive integrated database. Nevertheless, game information is still not enriched with meta-structures that could be used both on the Web and also in games. With such rich meta-structures that add more meaning to content, this would enable Web content to be reused in games. Moreover, the representation of *semantically-enriched* and *semantically-interlinked* content would enable game generators to infer how content can be interacted within the game world without having to rely on software development procedures that require laborious annotation of how each entity can be interacted within the game.

In this paper, we introduce an approach that automatically generates games from Web content through the use of rich-meta structures to describe game content. This research direction would provide a standard format for structuring and describing content for each game facet that would in turn be interlinked to automatically generate games. Our approach uses content from review gaming sites, game ranking sites and walkthroughs to generate games. This enriched content would be represented as interlinked game facet graphs containing information including visuals, audio, gameplay, rules, levels, user profiles, character information, and other relevant game information that could be extracted from diverse Web sources. The content is extracted and semantically enriched with meta-structures using new and/or already existing Semantic Web ontologies. Among other applications, our model could be used: (1) to define meta-structures for characterising and representing game data abstractly that could then be re-used on the Web from games; and (2) to integrate game content from the Web within games. This innovative way of game creation could lead to a new kind of real-time game experience which also implies the design of new game play and rules based on semantic information. The game generation approach envisaged is ambitious but nevertheless feasible with current technology as discussed in the remaining sections of this paper.

The remainder of this paper is as follows. Section II reviews current work on procedural game generation and semantics in games and Section III offers core background information about the Web of data. In Section IV we detail our approach of generating games using semantic data and Section V concludes the paper by providing an overall discussion about the future steps of this work.

## II. PROCEDURAL CONTENT GENERATION AND SEMANTICS

Development and research on automatic content generation mainly focuses on procedural content generation (PCG) that refers to the creation of game content automatically via algorithmic means [35]. Game content, primarily levels and visual assets, have been generated algorithmically in the game industry for decades. From the dungeons of *Rogue* (Toy and Wichman, 1980) to the universe of *Elite* (Acornsoft, 1984), procedural content generator has largely focused on the spatial structures of a specific game (with a specific, human authored ruleset and narrative). However, recent advances in PCG in academia has targeted a broader range of game facets: from the game rules [7] to soundscapes [27], and from the story [32] to the game's shaders [20], there is significant potential in computational creativity [24] in game design that has not been considered previously. Even though some generators have used real-world information as a starting point (seed) or a post-processing step (decoration) of their generative process most types of game content are not generated based on real-world information — or more appropriately, the human engineers insert their real-world assumptions (e.g. on what constitutes a “valid” castle) into the generator.

*Angelina* [11] uses *Guardian* articles as a seed for a platformer level: the textual information is parsed (extracting nouns) and using sentiment word analysis the “tone” of the article is computed (positive or negative). The platformer level is then decorated (i.e. colors, backgrounds, sound effects) based on the topics and tone, via Google search. *Game-O-Matic* [36] on the other hand relies on human-provided entities and their interactions (ways in which an entity affects another, such as “Hunger harms man”) to generate simple games. On the other hand, the games' mechanics are based on combined game behaviours which are provided by the generator's designers to simulate the interactions. The in-game entities' appearance are found via Google image search, and rendered as such in-game. It is obvious that in these examples, the real-world information provided by the Web mostly act as decoration to generated content (platformer levels or arcade games) which are created without influence from real-world data. Instead, the proposed approach goes beyond merely decorating known good generated results but instead integrates the real-world data more in all facets of generation. Another approach is *A Rogue Dream* [9], which uses a single word from the user (acting as the identity of the player's avatar, such as cat) and uses the auto-complete function of Google queries (e.g. “why do cats hate...”) to find the semantic identity of enemies, goals and special abilities of the player avatar.

The semantic identity (e.g. “dogs” in the above example) is depicted visually via Google image search. A Rogue Dream uses the associations existent in the communal knowledge pool (as popular search topics) but does not rely on structured data, as the current paper proposes; instead, it directly parses the text of autocompleted Google queries without, for instance, checking whether the potential enemy is indeed of type “living creature” or “person”.

Perhaps the closest attempt at using structured real-world data is the Data Adventures project [2], [3] which uses SPARQL queries (see Section III for a detailed description of SPARQL) on DBpedia to discover links between two or more individuals: the discovered links are transformed into adventure games, with entities of type “Person” becoming Non-Player Characters (NPCs) that the player can converse with, entities of type “City” becoming cities that the player can visit, and entities of type “Category” becoming books that the player can read. The proposed approach enhances the concepts explored in Data Adventures by using game information (from real-world games such as those found in Metacritic) in order to discover possible new mechanics: Data Adventures instead relies on hand-authored mechanics existent in traditional adventure games of the 1980s.

Most of these methods do not reuse already generated content from the Web and do not add semantics to the generated content which could be used by the game engine to infer (new) knowledge. Games could take any form in real-time based on semantic information, and semantic structures for game information create standard ways for publishing and reusing content in many games.

The advantages of using rich semantic information to automatically generate games are numerous [37] as more complex, open-world, non-linear, games incorporating very rich forms of interaction are possible (i.e. authentic sandbox games). Current work in using semantics in games focuses on the use of semantic information to generate game worlds or to describe interactions with game worlds such as the work in [21], [28], [38]. Although these provide useful insights in generic semantic models that describe interactions with game worlds, they do not offer vocabularies for describing game content and they neither provide a generic approach for reusing Web content to generate games.

Attempts in game ontology creation are relevant to our approach, hence, we outline the four key game-based ontologies existent currently. The Game Ontology Project [43] is a wiki-based knowledge-base that aims to provide elements of gameplay. However, this project does not take into consideration game content and does not provide fine-grained concepts that cover different aspects of information within the different game facets. Moreover, it does not provide a vocabulary to be consumed by data described in RDF which could make it potentially useful for game generation. The Digital Game Ontology [8] provides an ontology by aligning with the Music Ontology, and the Event and Timeline ontology, to provide concepts that describe digital games. However, the vocabulary is not available and in this regard, it is unclear what game

concepts this vocabulary provides. Finally, the Video Game Ontology [31] provides concepts for defining interoperability amongst video games and the Game2Web ontology [33] focuses on linking game events and entities to social data. Although these vocabularies are useful for describing several aspects of game information the ontologies are still limited to specific features of particular facets of game generation.

### III. BACKGROUND: THE WEB OF DATA

The *Web of Data* is evolving the Web to be consumed both by machines and humans whereas the traditional Web resulted to be for human consumption only. Indeed, machines cannot process additional meaning from the content found in Web pages since they are simply text and similarly from the non-typed links which do not contain any additional meaning about the relationships amongst the linked pages. Therefore, the *Web of Data* provides various open data formats which have emerged from the Semantic Web.

#### A. The Semantic Web

The Semantic Web [5] provides approaches for structuring information on the Web by using metadata to describe Web data. The advantage of using metadata is that information is added with meaning whereby Web agents or Web enabled devices can process such meaning to carryout complex tasks automatically on behalf of users. Another advantage is that the semantics in metadata improved the way information is presented, for instance merging information from heterogeneous sources on the basis of the relationships amongst data, even if the underlying data schemata differ. Therefore, the Semantic Web encouraged the creation of meta-formats to describe metadata that can be processed by machines to infer additional information, to allow for data sharing and to allow for interoperability amongst Web pages. The common format and recommended by W3C for Semantic data representation [4] is the Resource Description Framework (RDF)<sup>2</sup>.

#### B. Resource Description Framework (RDF)

RDF is a framework that describes resources on the World Wide Web. Resources can be anything that can be described on the Web; being real-world entities such as a person, real-world objects such as a car and abstract concepts such as defining the concept of game review scores. RDF provides a framework for representing data that can be exchanged without loss of meaning. RDF uniquely identifies resources on the Web by means of Uniform Resource Identifiers (URIs). Resources are described in RDF in the form of triple statements. A triple statement consists of a *subject*, a *predicate* and an *object*. A subject consists of the unique identifier that identifies the resource. A predicate represents the property characteristics of the subject that the resource specifies. An object consists of the property value of that statement. Values can be either literals or other resources. Therefore, the predicate of the RDF statement describes relationships between the subject and the object. If a triple had to be depicted as a graph, the subject

<sup>2</sup>RDF — <http://www.w3.org/TR/REC-rdf-syntax/>

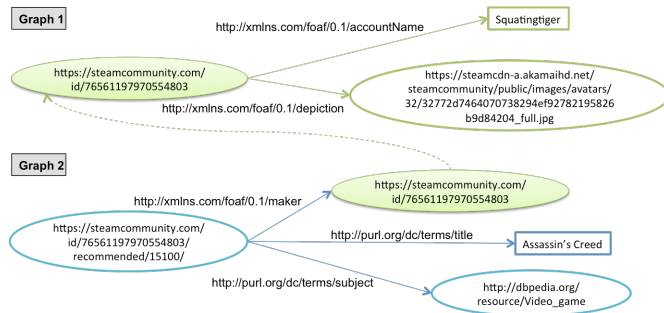


Fig. 1. Examples of graphs that interlink variant resources

and object are the nodes and the predicate connects the subject to the object node. The set of triples describing a particular resource form an RDF graph (Fig. 1).

RDF data can be queried by using an RDF query language called SPARQL<sup>3</sup>. SPARQL queries take the form of a set of triple patterns called a basic graph pattern. SPARQL triple patterns are similar to RDF triples with the difference that in a SPARQL triple, each subject, predicate and object can be bound to a variable; the variable's value to be found in the original data graph. When executing a SPARQL query, the resulting RDF data matches to the SPARQL graph pattern.

Moreover, the RDF data may require more meaning to describe its structure and therefore, an RDF vocabulary modelled using the RDF Schema (RDFS)<sup>4</sup> can be used to describe the RDF data's structure. Apart from vocabularies, RDF data may pertain to a specific domain which its structure needs to be explicitly defined using ontologies modelled by RDFS and/or OWL<sup>5</sup>. For example, ontologies may describe people such as the Friend of a Friend (FOAF)<sup>6</sup> ontology or may describe information from gaming communities to interlink different online communities such as the Semantically-Interlinked Online Communities (SIOC)<sup>7</sup> ontology.

### C. Linked Data

As mentioned previously, when describing a particular resource within a graph, a URI is assigned to that resource which can be referred to in other graphs using that particular URI. For instance, if a particular resource represents a person within another graph that describes information about that person, the person's (resource) URI can be used for example when describing that s/he is the creator of a game review which is described in another graph; as illustrated in Fig. 1. Hence this makes it easy to link data together from different datasets and thus creating *Linked Data*<sup>8</sup>. Datasets which are easily accessible are linked forming the Linking Open Data (LOD) cloud<sup>9</sup> which forms part of the *Web of Data*. In order to publish

data in the LOD cloud, it must be structured adhering to the Linked Data principles as stated in [19] and the Data on the Web best practices as stated in [14].

The benefit of linking data is that links amongst data are explicit and try to minimise redundant data as much as possible. Therefore, similar to hyperlinks in the conventional Web that connect documents in a single global information space, Linked Data enables data to be linked from different datasources to form a single global data space [19].

## IV. APPROACH

Given the core aims of this work our approach for semantic-based game generation is operationalised via the following sequence of key processes (as illustrated in Fig. 2): (1) the user inputs keywords or a narrative of the game s/he desires; (2) a list of game genres are extracted from the user's input; (3) rankings for each game are extracted from different game review and scoring sites, and the games are ranked according to the aggregate score; (4) game content, such as game plot and game walkthroughs are extracted from diverse Web sources for each of the high ranked games; (5) game information related to the different game facets (for example entities, actions etc.) is extracted from the game content and game facets RDF graphs are created; (6) the new game is generated by merging the game facets RDF graphs of the different games.

In the following subsections we detail the processes listed above and suggest ways to realise them.

### A. Extracting Game Genres and Game Lists

Video game genres and lists of games for particular genres can easily be extracted from Wikidata<sup>10</sup> and DBpedia<sup>11</sup> through their SPARQL endpoints. Wikidata consists of a collaborative editing knowledge base that provides common source of data for Wikipedia<sup>12</sup> and it collects data in a structured form allowing data to be easily reused. DBpedia also extracts structured information from Wikipedia and publishes this structured information on the Web. Hence, both Wikidata and DBpedia are good sources of structured knowledge to extract game information already enriched in semantic meta-formats.

In our approach, a user enters some keywords that describe the game to be generated. Keyphrase extraction techniques, such as those described in [40], can be used to identify game genres from the user's input. A graph of game genres can be constructed from e.g. Wikidata, DBpedia and WordNet [15] that will be used as a reference video game genre vocabulary whilst extracting keyphrases. Once the game genre is identified, a list of games for that particular genre can be extracted from Wikidata and DBpedia. For example, the query in Fig. 3 extracts the list of *action-adventure video games* from Wikidata as illustrated in Fig. 2, where the property P136 refers to the property genre and the item Q343568 represents the action-adventure game genre.

<sup>3</sup>SPARQL — <http://www.w3.org/TR/rdf-sparql-query/>

<sup>4</sup>RDFS — <http://www.w3.org/TR/rdf-schema/>

<sup>5</sup>OWL 2 — <http://www.w3.org/TR/owl2-overview/>

<sup>6</sup>FOAF — <http://www.foaf-project.org/>

<sup>7</sup>SIOC — <http://sioc-project.org/>

<sup>8</sup>Linked Data — <http://linkeddata.org/>

<sup>9</sup>Linking Open Data (LOD) cloud — <http://lod-cloud.net>

<sup>10</sup>Wikidata — <https://www.wikidata.org>

<sup>11</sup>DBpedia — <http://wiki.dbpedia.org>

<sup>12</sup>Wikipedia — <https://www.wikipedia.org>

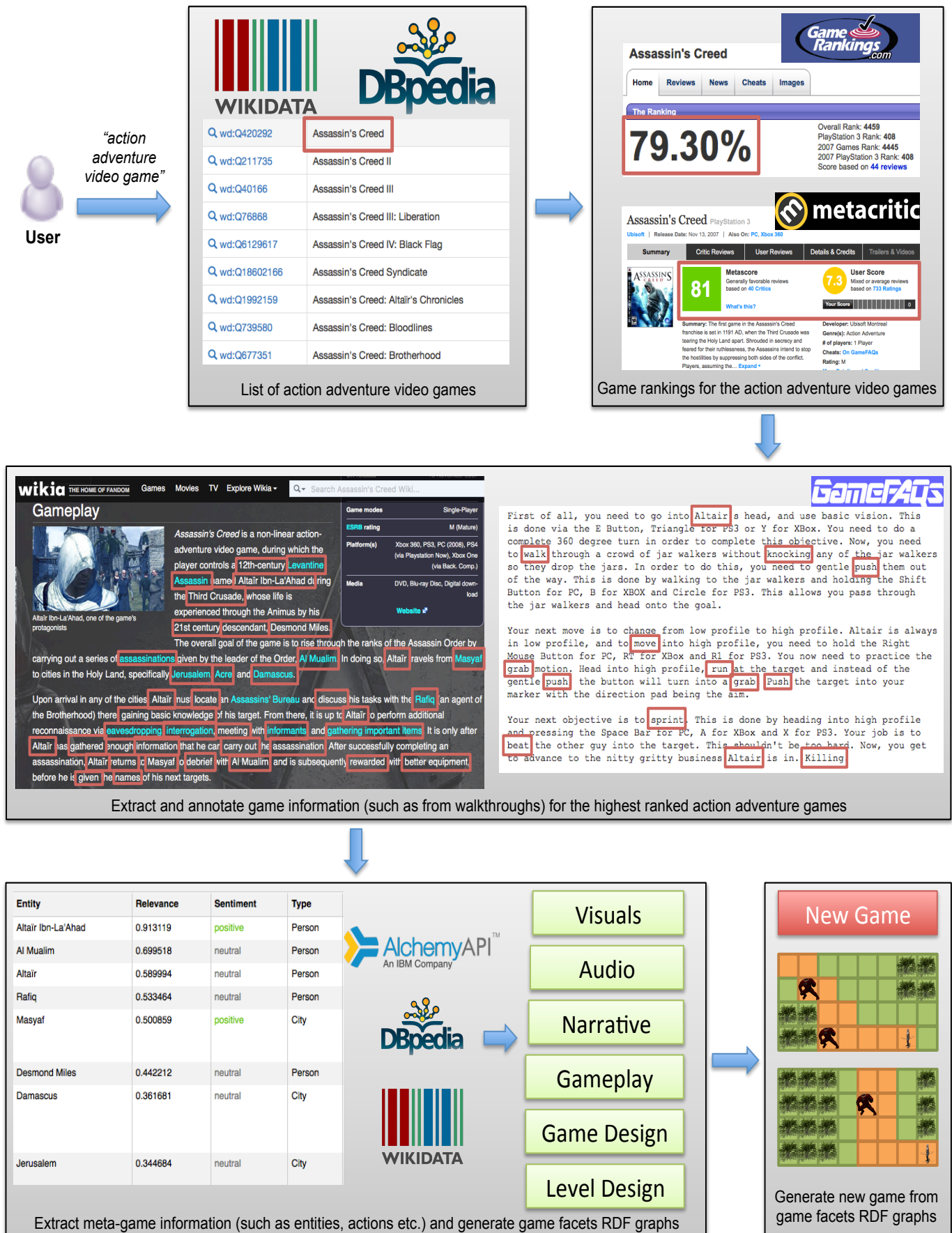


Fig. 2. An semantic approach for game generation

```

SELECT ?game ?gameLabel
WHERE
{
    ?game ?p ?o .
    ?o ps:P136 wd:Q343568.
    SERVICE wikibase:label {
        bd:serviceParam wikibase:language "en".
    }
}
ORDER BY ?gameLabel

```

Fig. 3. A SPARQL query that extracts a list of action adventure video games from Wikidata

We envisage to extend our approach by allowing the user to input a full narrative or a detailed description of the new game. The text is then parsed to understand the narrative (relating words to their semantic meaning) by using knowledge-bases such as *GluNet* [23], that provide a semantically rich lexical commonsense vocabulary intended to assist computational storytelling in computer games. *GluNet* maps existing resources together from *WordNet* — a lexical database of words [15] — *VerbNet* — a lexical database of verb semantics and syntax [22] — *FrameNet* — a lexical database of frame semantics [1] — and *ConceptNet* — a commonsense knowledge-base for relating narrative concepts to games [25]. Moreover, an RDF graph can be constructed out of the semantic relationships amongst the words in the narrative, and DBpedia Spotlight [12] will be used to automatically annotate the words to their semantic representations so that they are included in the RDF graph. This RDF graph can be used when generating the new game as a reference to what the user desires.

### B. Extracting Game Rankings

Game rankings can be extracted from various review sites in order to rank each game in the list extracted during the previous stage. Metacritic<sup>13</sup> and GameRankings<sup>14</sup> are two sites that provide weighted scores for games which can be used for this step of the game generation process.

Metacritic's metascores, for instance, are weighted average scores aggregated from the reviews created within the site. Metacritic game pages can easily be accessed by following the link provided in the page about the same game in Wikidata. Metacritic uses microdata to markup its pages. Microdata is a standard specification to add metadata within existing content on web pages by annotating HTML elements with machine-readable tags. The most popular vocabulary used to markup microdata content currently is Schema.org<sup>15</sup> that provides a collection of commonly used markup vocabularies. Therefore, the scores can easily be extracted from Metacritic by using a microdata parser to retrieve values annotated with the property `ratingValue`. Apart from scores, more semantically annotated game content can be parsed for later consumption. DBpedia also provides semantically annotated weighted scores

from various review sites such as from GameSpot<sup>16</sup>, IGN<sup>17</sup>, GamesRadar<sup>18</sup>, amongst others. While the GameRankings scores are weighted average ratings from various offline and online sources currently no direct link exists to the specific GameRankings page for a particular game and a crawler would be required to find the specific page. Moreover, GameRankings does not markup its content and a scrapper would be required to extract rankings and game information.

All aforementioned scores extracted from the various sites can be defined in RDF using vocabularies such as Schema.org, and the RDF graphs would be stored in a public-accessible RDF store. These scores can, in turn, be used to rank the games in the list extracted during the previous phase.

### C. Extracting Game Information

Game content, such as game plot, gameplay, game character information etc. can be found in various sources scattered around the Web, and when aggregated together, can provide in-depth details about the game mechanics. For instance Wikia<sup>19</sup> consists of encyclopaedias, each one specialised in a particular topic that covers much greater information and more comprehensible detail than Wikipedia. Wikia provides a detailed API that allows easy access to searching and extracting most of the content. GameFaq<sup>20</sup> is another source that provides game content such as walkthroughs from which in-depth detail about level design, game design and gameplay for a particular game could be extracted. Unlike Wikia, GameFaq does not provide an API and requires Web scrapping in order to extract the content. Other sources of game information from which game mechanics could be derived from include user review sites. Most of these sources contain unstructured information and require natural language processing techniques in order to parse and process the text meaningfully.

The challenges that game information extraction brings about include: (1) how to parse and understand which text is suitable to model visual information, audio information, narrative, level design, game design or gameplay; and (2) how to semantically represent each different game facet as RDF graphs. Entity recognition techniques could be used to identify and classify entities such as persons or locations in the text. The Stanford Named Entity Recogniser (NER) [16] tool could be used to extract entities and DBpedia spotlight would be used to match these entities to DBpedia resources described in RDF. Extracting entities and relevant information about these entities could be used to describe the visual aspects of the game, game character information, or other game entities. Part-of-speech tagging could then be used to identify which words are nouns or verbs where verbs could identify what actions can be performed in a game or the rules the game would have. As explained previously, *GluNet*, that contains *VerbNet* the lexical database of verb semantics, could be used to identify

<sup>13</sup>Metacritic — <http://www.metacritic.com>

<sup>14</sup>Game Rankings — <http://www.gamerankings.com/>

<sup>15</sup>Schema.org — <http://schema.org/>

<sup>16</sup>GameSpot — <http://www.gamespot.com>

<sup>17</sup>IGN — <http://www.ign.com>

<sup>18</sup>GamesRadar — <http://www.gamesradar.com/>

<sup>19</sup>Wikia — <http://www.wikia.com>

<sup>20</sup><http://www.gamefaqs.com/>



the verbs in the text. Moreover, ConceptNet could be used to identify gameplay rules; for instance in [29], the authors used both ConceptNet and WordNet to generate and define game rules. However, in [29] game rules are generated from a set of predefined verbs for a particular type of game, whereas in our work we propose that rules are generated from verbs extracted from Web content of different games (and based on common-sense reasoning using ConceptNet), creating new unconventional and unexpected game rules.

Once all game content has been parsed and classified, gaming information can be structured in RDF using common vocabularies such as SKOS<sup>21</sup> for describing knowledge organization systems (such as thesauri) concepts, Dublin Core<sup>22</sup> for describing provenance information, FOAF<sup>23</sup> for describing information about people, SIOC<sup>24</sup> for interlinking different online communities together, and Review vocabulary<sup>25</sup> for describing review information, amongst other vocabularies. However, new ontologies, that describe game levels or game design would be required to define new concepts which are not found in current semantic vocabularies. Game rules can be defined using the Semantic Web Rule Language (SWRL)<sup>26</sup> that enables Horn-like rules to be combined with an OWL knowledge-base. Ultimately, a number of RDF graphs representing various game content types can be created for each game creating an RDF store which yields a semantic knowledge-base of game information.

#### D. Generating Games from Semantic Information

In the final phase of the envisaged process the new game is generated by combining the several RDF graphs together, or even with other RDF graphs that are already stored in the knowledge base and that relate to the user's requirements. Ontology alignment and semantic matching techniques are used to find semantic relationships amongst ontologies [13] and to identify whether graph structures are semantically related [17] — to find the correspondences and mappings amongst the RDF graphs describing the game information. Through these correspondences, the RDF graphs are merged to form the new game graph. This new game could result to an entirely new and unconventional game genre. A semantic game engine, based on RDF and SWRL reasoners, would then be required to parse the merged RDF graphs in order to transform the semantic information into a playable game.

In its simplest instance, the ontology can directly guide the generative process — for instance an entity of type *Person* can instantiate an NPC (with details or visual appearance of the NPC provided within the ontology) — similar to the work in [2]. Moreover, the ontology can indirectly specify the parameter vector of the generator: for instance a level with the

type of “dungeon” can adjust the parameters of the generator to create levels with low linearity [34], due to the relationship between “dungeon” and “maze”. A more ambitious target for semantic game generation is to use machine learning to identify patterns (e.g. visual patterns of textures, playtrace patterns in levels) of content in existing game within the ontology; the learned model of content quality and semantic information can then be used as an objective for a search-based content generator, targeting content with similar patterns as to those in existing games.

## V. CONCLUSION

In this paper we presented our envisaged approach for generating games via semantic information extracted from diverse Web content. We have provided some first insights on what game content could be extracted to generate unconventional games, how to semantically enrich such content, and how games can be generated from these semantic representations. Apart from generating games, the benefit of adding semantic information to game content is many-fold: enriched game information can be published on the Web for interlinking and consumption, enriched game content can easily be reused in games without requiring any effort to modify game artefacts, and real-time interactions amongst objects in games could easily be achieved.

In contrast to current automated game generation processes such as traditional procedural content generation practices, our approach enables the use of massive amounts and dissimilar types of content from online sources. This allows content to be automatically generated whilst taking into consideration player models derived from user information stored across various online datasets [41] thereby realising a semantically-enriched version of the experience-driven PCG framework [42]. For example, Metacritic contains user reviews which can provide a quantifiable (based on the user scores) or qualitative (based on sentiment-word analysis of the textual review) model of the contributing user base. This model can be used to create game content or complete games, which are expected to appeal to the entire community or to specific parts of the community, based for instance on demographics or skill or interests collected from user's steam achievements or favoured games, respectively. On the other hand, an indirect model of player engagement with specific types of content can be gleaned from the mostly user-generated wikia pages. Pages with popular characters and locations or challenging game levels are expected to have more textual contributions (due to being updated more often by more people). This can be used to create content similar to existing game content popular in one or more wikia user communities.

With the novel approach proposed in this paper we envisage not only the generation of personalised digital games autonomously but also the creation of games that are perceived as being unconventional and unexpected, yet engaging and playable.

<sup>21</sup>Simple Knowledge Organization System (SKOS) — <http://www.w3.org/2004/02/skos/core#>

<sup>22</sup>Dublin Core — <http://dublincore.org/documents/dcmi-terms/>

<sup>23</sup>FOAF — <http://xmlns.com/foaf/0.1/>

<sup>24</sup>SIOC — <http://rdfs.org/sioc/ns#>

<sup>25</sup>Review vocabulary — <http://purl.org/stuff/rev#>

<sup>26</sup>SWRL — <https://www.w3.org/Submission/SWRL/>

## ACKNOWLEDGMENT

The research work disclosed in this publication is partially funded by the REACH HIGH Scholars Programme — Post-Doctoral Grants. The grant is part-financed by the European Union, Operational Programme II — Cohesion Policy 2014-2020 Investing in human capital to create more opportunities and promote the wellbeing of society — European Social Fund.

## REFERENCES

- [1] C. F. Baker, C. J. Fillmore, and J. B. Lowe. The Berkeley FrameNet Project. In *17th International Conference on Computational Linguistics, COLING'98*, 1998.
- [2] G. A. Barros, A. Liapis, and J. Togelius. Playing with Data: Procedural Generation of Adventures from Open Data. In *1st International Joint Conference of DiGRA and FDG, DiGRA-FDG'16*, 2016.
- [3] G. A. Barros, A. Liapis, and J. Togelius. Who Killed Justin Bieber? Murder Mystery Generation from Open Data. In *Seventh International Conference on Computational Creativity, ICCCC'16*, 2016.
- [4] T. Berners-Lee. *Semantic Web Road Map*, September 1998.
- [5] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284:34–43, 2001.
- [6] C. Bizer, T. Heath, K. Idehen, and T. Berners-Lee. Linked Data on the Web (LDOW2008). In *17th International Conference on World Wide Web, WWW '08*, 2008.
- [7] C. Browne and F. Maire. Evolutionary Game Design. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(1):1–16, 2010.
- [8] J. T. C. Chan and W. Y. F. Yuen. Digital Game Ontology: Semantic Web Approach on Enhancing Game Studies. In *9th International Conference on Computer-Aided Industrial Design and Conceptual Design, CAID/CD 2008*, 2008.
- [9] M. Cook and S. Colton. A Rogue Dream: Automatically Generating Meaningful Content For Games. In *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2014.
- [10] M. Cook, S. Colton, and J. Gow. Automating Game Design in Three Dimensions. *AISB Symposium on AI and Games*, 2014.
- [11] M. Cook, S. Colton, and A. Pease. Aesthetic Considerations for Automated Platformer Design. In *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE'12*, 2012.
- [12] J. Daiber, M. Jakob, C. Hokamp, and P. N. Mendes. Improving Efficiency and Accuracy in Multilingual Entity Extraction. In *9th International Conference on Semantic Systems (I-Semantics)*, 2013.
- [13] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. *Ontology Matching: A Machine Learning Approach. Handbook on Ontologies*. Springer Berlin Heidelberg, 2004.
- [14] B. Farias Lscio, C. Burle, and N. Calegari. W3C. Data on the Web Best Practices. 19 May 2016. W3C Working Draft. <http://www.w3.org/TR/dwbp/>.
- [15] C. Fellbaum, editor. *WordNet: an electronic lexical database*. MIT Press, 1998.
- [16] J. R. Finkel, T. Grenager, and C. Manning. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. In *43rd Annual Meeting on Association for Computational Linguistics, ACL '05*, 2005.
- [17] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. S-Match: an algorithm and an implementation of semantic matching. In *First European Semantic Web Symposium, ESWS'04*, 2004.
- [18] K. Hartsook, A. Zook, S. Das, and M. O. Riedl. Toward supporting stories with procedurally generated game worlds. In *IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2011.
- [19] T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Morgan and Claypool, 2011.
- [20] A. Howlett, S. Colton, and C. Browne. Evolving pixel shaders for the prototype video game subversion. In *The Thirty Sixth Annual Convention of the Society for the Study of Artificial Intelligence and Simulation of Behaviour (AISB'10)*, 2010.
- [21] J. Kessing, T. Tutenel, and R. Bidarra. Designing semantic game worlds. In *The Third Workshop on Procedural Content Generation in Games, PCG'12*. ACM, 2012.
- [22] K. Kipper, A. Korhonen, N. Ryant, and M. Palmer. A large-scale classification of english verbs. *Language Resources and Evaluation*, 42(1):21–40, 2008.
- [23] B. Kybartas and R. Bidarra. A Semantic Foundation for Mixed-Initiative Computational Storytelling. In *Interactive Storytelling: 8th International Conference on Interactive Digital Storytelling, ICIDS'15*. Springer, 2015.
- [24] A. Liapis, G. N. Yannakakis, and J. Togelius. Computational game creativity. In *Fifth International Conference on Computational Creativity, ICCCC'14*, 2014.
- [25] H. Liu and P. Singh. ConceptNet – A Practical Commonsense Reasoning Tool-Kit. *BT Technology Journal*, 22(4):211–226, Oct. 2004.
- [26] P. Lopes, A. Liapis, and G. N. Yannakakis. Sonancia: Sonification of procedurally generated game levels. In *1st Computational Creativity and Games Workshop*, 2015.
- [27] P. Lopes, A. Liapis, and G. N. Yannakakis. Targeting Horror via Level and Soundscape Generation. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE'15*, 2015.
- [28] R. Lopes and R. Bidarra. A semantic generation framework for enabling adaptive game worlds. In *8th International Conference on Advances in Computer Entertainment Technology*. ACM, 2011.
- [29] M. J. Nelson and M. Mateas. Towards automated game design. In *AI\*IA 2007: Artificial Intelligence and Human-Oriented Computing*, pages 626–637. Springer, 2007.
- [30] K. Orland. Ars Technica: How NASA topography data brought dose of reality to SSX snowboarding courses.
- [31] J. Parkkila, F. Radulovic, D. Garijo, M. Poveda-Villalón, J. Ikonen, J. Porras, and A. Gómez-Pérez. An ontology for videogame interoperability. *Multimedia Tools and Applications*, pages 1–20, 2016.
- [32] J. Robertson and R. M. Young. Automated gameplay generation from declarative world representations. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE'15*, 2015.
- [33] O. Sacco, M. Dabrowski, and J. G. Breslin. Linking in-game events and entities to social data on the web. In *Games Innovation Conference (IGIC), 2012 IEEE International*, pages 1–4, Sept 2012.
- [34] G. Smith and J. Whitehead. Analyzing the Expressive Range of a Level Generator. In *Workshop on Procedural Content Generation in Games, PCGames'10*. ACM, 2010.
- [35] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne. Search-Based Procedural Content Generation: A Taxonomy and Survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):172–186, Sept 2011.
- [36] M. Treanor, B. Blackford, M. Mateas, and I. Bogost. Game-O-Matic: Generating Videogames That Represent Ideas. In *The Third Workshop on Procedural Content Generation in Games, PCG'12*. ACM, 2012.
- [37] T. Tutenel, R. Bidarra, R. M. Smelik, and K. J. D. Kraker. The Role of Semantics in Games and Simulations. *Computers in Entertainment*, 6(4):57:1–57:35, Dec. 2008.
- [38] T. Tutenel, R. M. Smelik, R. Bidarra, and K. J. de Kraker. Using Semantics to Improve the Design of Game Worlds. In *5th Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE'09*, 2009.
- [39] I. D. Visualization. Speedtree., 2010.
- [40] I. H. Witten, G. W. Paynter, E. Frank, C. Gutwin, and C. G. Nevill-Manning. Kea: Practical automatic keyphrase extraction. In *The fourth ACM conference on Digital libraries*. ACM, 1999.
- [41] G. N. Yannakakis, P. Spronck, D. Loiacono, and E. Andre. Player Modeling. In *S. M. Lucas, et al. eds. Artificial and Computational Intelligence in Games*. s.l.: Dagstuhl Seminar, 2013.
- [42] G. N. Yannakakis and J. Togelius. Experience-driven procedural content generation (extended abstract). In *International Conference on Affective Computing and Intelligent Interaction, ACII'15*, 2015.
- [43] J. P. Zagal and A. Bruckman. The Game Ontology Project: Supporting Learning While Contributing Authentically to Game Studies. In *8th International Conference on International Conference for the Learning Sciences, ICLS'08*, 2008.

# Deep Q-Learning using Redundant Outputs in Visual Doom

Hyunsoo Park, and Kyung-Joong Kim\*

Department of Computer Science and Engineering  
Sejong University  
Seoul, South Korea  
rex8312@gmail.com, kimkj@sejong.ac.kr

**Abstract**— Recently, there is a growing interest in applying deep learning in game AI domain. Among them, deep reinforcement learning is the most famous in game AI communities. In this paper, we propose to use redundant outputs in order to adapt training progress in deep reinforcement learning. We compare our method with general  $\epsilon$ -greedy in ViZDoom platform. Since AI player should select an action only based on visual input in the platform, it is suitable for deep reinforcement learning research. Experimental results show that our proposed method archives competitive performance to  $\epsilon$ -greedy without parameter tuning.

**Keywords**—deep reinforcement learning; reinforcement learning; vizdoom; first-person perspective game;

## I. INTRODUCTION

In recent years, the deep learning has become famous in various domains. Especially, it shows better performance than conventional methods in handling high dimensional data such as visual inputs. Deep reinforcement learning (Deep Q-Learning; DQL) is the one of representative works in the game AI domain. Basically, it is a combination of deep learning with Q-learning and can learn an AI game player handles raw pixel or text inputs.

Fig. 1. ViZDoom platform (basic example)



Visual Doom (ViZDoom) is the one of AI competition platforms opened recently [1], also OpenAI Gym [2] includes it. It is based on the Doom, the famous classic first person shooter game (Fig. 1). AI players on this platform can only obtain visual input and some variables (eg. Health and armor). The platform does not provide more detailed structured data like map data for navigation or forward models for simulations. It opens new challenge for traditional game AI methods.

The DQL is one of promising solutions to make an AI for ViZDoom. The DQL is one of representative deep learning works in game AI domain. Mnih *et al.* introduce DQL in 2013 [3]. They show DQN can learn how to play various Atari 2600 games. After success of DQL, there have been a lot of works about deep learning in game AI. However, many DQL studies focused on 2-D games unlike ViZDoom. Since ViZDoom provides only first-person view, the player cannot see whole environment (obtains limited information). Furthermore, view angle change makes different visual inputs for the same object.

Exploration and exploitation dilemma is one of important problems in reinforcement learning. It's dilemma between trying new situation in order to get information about environment (exploration) or pursues rewards based on the current knowledge (exploitation). If the player tries the exploration too much or pursues rewards too much, it is possible to reduce total rewards. Because of this reason, the mechanism of how to deal with this dilemma is important.

In many DQL studies, they use  $\epsilon$ -greedy algorithm to handle this dilemma. Simply, the  $\epsilon$ -greedy selects a random action with  $\epsilon$  probability ( $\epsilon$  belongs to  $[0, 1]$ ). Otherwise, it selects the action with the highest rewards based on the current knowledge. Generally,  $\epsilon$  value is initially set as high value and gradually reduced at each learning iteration. The  $\epsilon$ -greedy is easy to use and shows good performance. But it is not a kind of adaptive learning process and there are some parameters should be determined properly.

In this paper, we propose an algorithm to balance exploration and exploitation. Generally, the number of output nodes in neural network of DQL is the same to the actions that a player can perform. The output of each node is interpreted as an expected Q-value of each action. In our proposed method, we add multiple pair of nodes into the output layer. For instance, if there are two possible actions, then total number of output nodes is  $2 \times 10 = 20$ . We use these redundant outputs to measure uncertainty of each action's Q-value in the current state. Using this information, proposed method could estimate the progress of learning and use it to balance exploration and exploitation. Osband *et al.* also use redundant output to boost training efficiency [4]. However, our proposed methods are simpler than the previous work.

This work was supported by the National Research Foundation of Korea (NRF) grant (2013 R1A2A2A01016589), Ministry of Culture, Sports and Tourism (MCST) and Korea Creative Content Agency (KOCCA) in the Culture Technology (CT) Research & Development Program 2016.

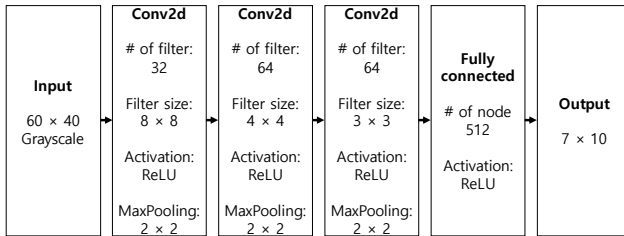
\*: corresponding author

In order to compare performance of our proposed method and  $\epsilon$ -greedy, we use ViZDoom's basic example environments. In the experimental result, our proposed method gets similar total rewards eventually without parameter tuning.

## II. PROPOSED METHOD

Fig. 2. shows an architecture of neural networks in our experiments. There are six layers including input and output layers. In the input layer, we provide input data as gray scale  $60 \times 40$  image at each game tick. Next, there are three convolution layer with ReLU activation function. First convolution layer has 32 filters (size:  $8 \times 8$ ), second convolution layer has 64 filters (size:  $4 \times 4$ ), third convolution layer has 64 filters ( $3 \times 3$ ). Also, each convolution layer is with max pooling layer ( $2 \times 2$ ). The next layer is a fully connected layer with ReLU activation function (512 nodes). The final layer is an output layer. In  $\epsilon$ -greedy, the number of nodes is equal to the number of actions. But, our proposed method needs 10 times more output nodes than normal. We choose 10 empirically considering diversity of outputs and computation time. Our proposed method needs more computation time than  $\epsilon$ -greedy, but it's not significant.

Fig. 2. Neural network architecture



The ViZDoom's basic example environment allows three buttons (Left (L), Right (R) and Shoot (S)). We define 7 possible actions (press L, R, S, L+S, R+S, L+R and nothing). If press two keys in same time, like move (L and R) and S, AI perform two behavior at same time (fire pistol while moving), except press L+R. In this case, AI do nothing. Therefore, the number of output nodes is set as  $7 \times 10 = 70$ . It has 10 redundant sets, and each set has 7 outputs.

For the balance of exploration and exploitation, this method chooses one set randomly and selects an action with the highest Q-value from the set. As a result, the AI chooses from random actions (exploration) when there are disagreement on the highest Q-value action among the redundant sets. If Q-value across the sets are similar each other, it chooses the best action (exploitation). We update neural net's weights of the action over sets at once. In a testing mode, our method selects an action from the voting of all sets.

## III. EXPERIMENTS

We use ViZDoom's basic example for our experiments (Fig. 1). When starts a new game, there is a small room and a target at random position of opposite side of the room. The goal of this environment is to hit the target as soon as possible.

AI player gets 100 points when hits the target, -6 points when misses the target, and -1 point each time nothing happened.

Our proposed method and  $\epsilon$ -greedy use the same neural network architecture except the number of output nodes. We use mean squared error as an objective function and RMSProp as an optimizer (learning rate:  $10^{-5}$ ). We use replay memory size 10,000 and batch size 32, The  $\epsilon$  value of  $\epsilon$ -greedy starts with 1.0 and gradually reduces during 20,000 training iterations to 0.1 after then it holds 0.1. We test model at each 5,000 iterations. In this time, we run 100 episodes with test mode action selection method.

Fig. 3. Total rewards in training and testing

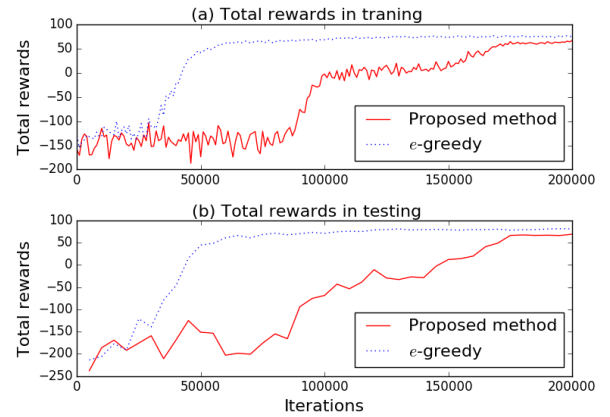


Fig. 3. shows experimental results. It shows total rewards change in training and testing (average of five experiments). According to experimental results, the proposed method gets total rewards similar to  $\epsilon$ -greedy's one after enough training. Usually, final trained models of both models perform optimal behavior (move to proper position and shoot accurately). Although it takes more iterations, there are only one parameter (the size of redundancy in output nodes), but  $\epsilon$ -greedy needs more (eg.  $\epsilon$ 's upper/lower bound, and update) parameters.

## IV. CONCLUSION AND FUTURE WORKS

In this paper, we propose using redundant output to explore game environments in DQL. The most general method that can handle exploration and exploitation dilemma in reinforcement learning is the  $\epsilon$ -greedy. Our proposed method can archive similar results with enough training iterations. It can adapt training progress with redundant output nodes.

## REFERENCES

- [1] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, "ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning," arXiv:1605.02097 [cs], May 2016.
- [2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," arXiv:1606.01540 [cs], Jun. 2016.
- [3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," arXiv:1312.5602 [cs], Dec. 2013.
- [4] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep Exploration via Bootstrapped DQN," arXiv:1602.04621 [cs, stat], Feb. 2016.

# Computational Intelligence and Cognitive Performance Assessment Games

Christoffer Holmgård  
Game Innovation Lab  
NYU Tandon School of Engineering  
Brooklyn, New York 11021  
christoffer@holmgard.org

Julian Togelius  
Game Innovation Lab  
NYU Tandon School of Engineering  
Brooklyn, New York 11021  
julian@togelius.com

Lars Henriksen  
Apex Group ApS  
Pilestræde 43  
1112 Copenhagen K, Denmark  
lars@apex.dk

**Abstract**—In this paper, we present the idea that game design, player modeling, and procedural content generation may offer new methods for modern psychological assessment, allowing for daily cognitive assessment in ways previously unseen. We suggest that games often share properties with psychological tests and that the overlap between the two domains might allow for creating games that contain assessment elements and provide examples from the literature that already show this. While approaches like these are typically seen as adding noise to a particular instrument in a psychometric context, research in player modeling demonstrates that it is possible to extract reliable measures corresponding to psychological constructs from in-game behavior and performance. Given these observations, we suggest that the combination of game design, player modeling, and procedural content generation offers new opportunities for conducting psychometric testing with a higher frequency and a higher degree of personalization than has previously been possible. Finally, we describe how we are currently implementing the first version of this vision in the form of an application for mobile devices that will soon be used in upcoming user studies.

## I. INTRODUCTION

Psychological testing and assessment has been practiced since at least 2200 B.C. in China [1], while its modern Western form can trace its roots to the end of the 19th century [2]. Since then, the basic practice of testing has followed roughly the same template: individuals are tested before some liminal decision point, e.g. when a person is eligible for moving into a different grade in school, is applying for a new position, or when the effects of a training program need to be assessed. However, we believe that psychological assessment could be used to greater effect if it were possibly to apply it not only at these liminal points, but more often, for instance on a daily basis. Based on the existing literature which shows that games, digital or otherwise, share many properties with psychological tests and may be capable of capturing much of the same information about their users, we intuit that games could be useful enablers for this vision.

If games can be used for psychological assessment, this may open up for new application areas where games or game-like systems can contribute with novel properties - for instance their ability to sustain user engagement and motivate recurrent use through intrinsically and extrinsically motivating features [3], [4].

One example of a novel use case could be for workers who currently undergo uncomfortable drug screening tests based on e.g. biological samples. Workers could avoid these, instead documenting their readiness for work simply by performing a short, daily cognitive test. In effect, a daily cognitive test could be a “breathalyzer for the brain”, testing a person’s fitness for a particular job or function immediately before starting it. Here, we suggest a number of principles to enable this, centered around using digital games to enable longitudinal, frequent, within-subject measurement of cognitive performance characteristics.

The field of psychometrics has, over the course of more than a century, developed practices for dealing with the specific challenges of testing human skills and capabilities at specific points in time. When developing new instruments and protocols for applying them, three challenges tend to recur in psychometrics: ensuring that the subjects taking tests are motivated and performing at a level representative of their best or typical behavior or abilities, ensuring that tests are valid, and ensuring that they are reliable [1].

Psychometrics already has a number of established and documented approaches to address these challenges, including different conceptions of validity, different ways of measuring reliability, and approaches that attempt to attune themselves to extract the maximal amount of information about the subject from a single test, such as Item Response Theory [5] based tests and Computerized Adaptive Testing [6].

In this paper, we argue that digital game design coupled with player modeling and procedural content generation can offer psychometrics a new set of approaches that may lead to new testing paradigms. Some of these ideas have already been explored in the games literature, extensively even, while others, to the best of our knowledge, are novel and so far unexplored.

We suggest three main design principles drawn from game design and computational intelligence for games and proceed to describe a currently ongoing research project that attempts to realize these ideas, including a description of the current prototype and planned user study. The three approaches from games and computational intelligence we believe could enable longitudinal, frequent, within-subject measurement of psychological performance characteristics are: 1) Using (digital)

game design to motivate subjects to participate in frequent testing. 2) Using player modeling to ensure that the maximal amount of information is learned about the subject from each play through. 3) Using personalized procedural content generation to mitigate and control learning effects, also known as test-retest effects [1], ensuring that tests remain valid and reliable in spite of frequent test administration.

The rest of this paper is structured into four main parts. First, we visit the state-of-the-art in cognitive assessment identifying where games or game-like applications may provide value and novelty. Second, we describe related work in using games to measure psychological characteristics. Third, we describe in detail how the three approaches listed above could be used to realize a specific, novel form of psychological assessment. Fourth, we describe how these ideas drive the design, implementation, and refinement of a prototype application called *SkillShow*. The purpose of *SkillShow* is to provide a platform for daily testing of performance indicators for simultaneous capacity [7], [8] and inductive reasoning intelligence [9].

## II. RELATED WORK

In this section, we start by describing the use of computational intelligence in *Work and Organizational Psychology* (WOP), with a focus on identifying challenges in assessment and selection. We then move on to describing related work in the use of games for identifying psychological characteristics and individual differences in players.

### A. Work and Organizational Psychology

The field of WOP has been engaged in addressing the problem of personnel selection and training for more than a century. A typical approach has been to combine fundamental psychometric measurements such as cognitive tests and personality tests with samples of work and interviews for assessment [10]. The use of computerized adaptive testing is also well known in psychology and psychometrics [6], but WOP has only recently started to apply methods from the field of computational intelligence. A nascent movement in WOP is poised to embrace the applications of methods from the artificial and computational intelligence communities [11]. This may bring new levels of specificity and falsifiability to sub-disciplines such as job and task analysis, work behavior measurement, motivation modeling, performance management, personnel assessment and selection, and the modeling of individual differences [11]. The purpose of the research described here is to leverage computational game intelligence to embrace this opportunity.

### B. Games for Assessment and Measuring Psychological Characteristics

Often, training simulations/games (such as those used in military or corporate settings) and educational games (such as those used in schools) have been used as measures of the learners'/players' abilities in the subject matter, providing a form of assessment. Sometimes, these assessments have been focused on documenting performance in a narrow field or

curriculum or they have been focused on assessing not the students, but the transfer capability of the simulation or game [12].

However, games and assessment at the more general level have been related within the field of simulation and gaming for decades. As early as 1978, Spitz argued that performance in games such as Mancala and Three-in-a-row are related to intelligence [13]. Jones et al. [14] and Jones [15] described how video games might be used for performance assessment. Work in this vein continued throughout the 1980's and 1990's with a focus on assessing intelligence through video games [16], [17].

Using commercial games to quantitatively assess other psychological characteristics and individual differences is a newer idea. Recently, significant work has been done in measuring personality, such as the work by Van Lankveld et al. [18], [19] who specifically suggested games as personality profiling tools [20]. Yee et al. [21] have conducted work in the same vein while Canossa et al. [22] focused on assessing differences in life motivations and later also personality [23]. Tekofsky et al. have shown how play style varies with age [24] in *Battlefield 3* [25] and how performance and speed in the game decrease with age [26]. Finally, Boot et al. recently released an overview of the use of video games as tool for investigating cognitive processes [27].

Meanwhile, the notion of modeling the player for reasons directed at the game experience itself or for impacting the player, such as adaptively changing game difficulty or other parameters to control e.g. player engagement, has been explored for a number of years [28]–[31]. Particularly within applied games, such as educational games or games for health treatment, this has been a focus of significant research into both affective responses and in-game behavior [32]–[34]. Work by Schute et al. [35], [36] has shown how assessment can be built into (educational) games in a way that allows for automatic assessment of student mastery of curricular topics.

Neuroscientifically derived tasks have started appearing in gamification-based frameworks targeting talent identification for recruitment and organizational placement purposes [37]. These approaches to characterizing individuals through games and gamified activities make their way into the general field of WOP, as exemplified by companies such as *Knelf*, *Knack*, *pymetrics*, and *owiwi*<sup>1</sup>, contributing to a general change toward a higher reliance on objective data, in the sense of Yannakakis et al. [31]<sup>2</sup>, and computational intelligence, supporting or supplanting traditional expert-based assessment practices [11].

Altogether, this growing body of research and products shows that activities that take place in games share characteristics with activities included in psychological assessment instruments, from small arcade games to expansive role playing games: The interactions with NPCs and the environment in a game like *Fallout 3* [38] may provide information about

<sup>1</sup>qa.knelf.com; knack.it; pymetrics.com, owiwi.gr

<sup>2</sup>We still consider models built on objective data subjective in the sense that any model's methods and data set are designed, selected, and deployed by individuals or groups thereof.



the same characteristics as the verbal indications given in response to a personality test [23]. Or the performance in a real time strategy game such as *StarCraft 2* [39] may be indicative of cognitive motor performance, a cognitive characteristic normally measured with a specialized test [40]. At the other end of this spectrum, simple tasks drawn from neuroscience, such as the Go/No-Go task, a task developed for assessing attention and inhibitory control/disinhibition, may be put in a gamified context [37] and can be perceived as a game activity by the test subject [41]. This shows that the lines between psychological assessment instruments and games in some instances can be blurry and that the two might be combined for some purposes. In the next section, we show why we believe daily cognitive assessment would be a suitable use case for such a combination.

### III. CHALLENGES FOR DAILY COGNITIVE ASSESSMENT

In this section, we identify some of the specific challenges that prevent the use of daily cognitive assessment in WOP today and suggest how game design, computational intelligence, and procedural content generation together might address these. We start by reviewing how game design may alleviate typical issues in test fatigue and retain motivation for daily testing. Afterwards, we suggest that player modeling may drive personalized procedural content generation which in turn can support test-retest reliability while improving the subject/player experience. Building on this idea, we propose that existing research in active player modeling might be used to configure assessment tasks to gather more relevant information about individual players. Then, we propose that assessment games may also provide logistical benefits to frequent cognitive assessment by leveraging existing technologies and principles from game telemetry and game analytics. Finally, we expand upon what we believe is the potential of procedural content generation for psychological assessment.

#### A. Motivation for Tests

When candidates take psychological assessment tests today, we typically assume that they will be performing at the best possible level they are capable of at that specific moment in time. One reason for this could be that the test is acting as a gatekeeper between the individual and some desired outcome, so we assume the candidate is striving to perform well. Another reason could be that the test is assessing a person's performance levels as part of monitoring during e.g. rehabilitation, where we assume that the patient's goals are aligned with yielding an indicative assessment. However, we could risk that subjects were nervous test takers or under the influence of stereotype threat [42] and therefore performing below their actual optimum. In some instances we may suspect that individuals' goals are not aligned with identifying their maximal performance, such as when insurance claims are involved or e.g. during evaluation for conscription into armed forces. Still, even if subjects might be stressed or malinger we would assume that they took an active and engaged stance toward the test and we are less concerned about the

subject's engagement with the test. Extrinsic motivation, other contextual factors, or simply the novelty of the test typically motivate the subject to engage [1].

In the case of frequent testing with the same test(s), however, the circumstances may be radically different. Exposure to the same test or similar tests over and over again is known to cause test or survey fatigue in humans. Simply put, it becomes boring. Subjects who are bored cannot be expected to produce data that is as indicative and valid as data from subjects who are highly motivated for and/or engaged with the test [43]. The amount of noise in the collected data should be expected to increase.

For the use case envisioned here, where we want to assess the same individual on a daily basis, principles drawn from game design seem like suitable approaches to enriching an assessment test with motivating elements reducing this undesirable noise. Identifying gratifying core loops for games that simultaneously work as cognitive performance indicators would be one approach to ensure that the subject remains attentive to and engaged with the test.

Appealing and motivating game designs often incorporate strong elements of feedback to the player, in order to communicate the game's evaluation of her performance and to guide her to play in certain ways or take certain actions [44], [45]. Typically, psychometric test construction avoids these kinds of feedback loops in order to minimize the amount of noise introduced into the test situation and to keep tests comparable. By keeping the context static it becomes easier to assess the individual and compare individuals [1]. In contrast, feedback and performance communication is considered integral to game design [44].

As noted above, the literature on game based testing shows that it is possible to create motivating games rich in feedback that still have acceptable validity as assessment tools. This indicates that it should also be possible to motivate the player through interesting, varying gameplay while still accurately evaluating player performance through player models that take the game as context into consideration [46]–[48] and by extension we should most likely also be able to evaluate their cognitive characteristics.

A problem related to dealing with the context of the game is dealing with the developing expertise of the player, as she becomes more proficient through experience. This, in turn, is related to the problem of test-retest reliability in psychological testing, which we approach in the following section.

#### B. Test-retest Reliability

Game design and player modeling excel at evaluating individuals in manners perceived as fair across contexts that are comparable in general, but vary in their specific configuration. Scores obtained in individual play sessions of e.g. *Tetris* [49] are generally considered comparable to one another even though the specific sequence of tetrominoes encountered may have been different. The balancing of the game is assumed to provide guarantees that even though specific game instances are unique, their difficulties are roughly equal over time and

part of the gameplay is managing this uncertainty [50]. If games feature progression, a player generally expects a well-planned challenge curve that matches her development of skill over time, or even adapts to it [28].

In psychological assessment, the development of expertise with regard to the test itself is generally viewed as undesirable, as this is assumed to obscure actual underlying performance characteristics of the subject which are applied to the test as a task, but not trained by or developed from it [1]. In games, and in particular games that leverage computational intelligence for difficulty adjustment, this development is generally leveraged as an asset. Knowing the player's development of expertise allows the game designer or the game artificial intelligence to configure the game to an appropriate difficulty level. Importantly, a game may keep a history of the player's skill development and may store the context for the exhibited performances too [51], [52]. This shows that player modeling already contains the necessary frameworks to track and calibrate for the player's development of expertise over time by adjusting the challenge/difficulty [31]. For the case of daily cognitive assessment using games, this becomes particularly important, as each subject/player may follow an individual learning curve. We engage with this topic in the following section.

### C. Individualized Test Sensitivity

Classical psychological assessment would typically provide all subjects with the same instrument and use principles such as e.g. Item Response Theory, time limits, or progress measures through instrument items to gain sufficient information about each subject [1]. More recently, Computerized Adaptive Testing has started providing methods for making individualized test configurations that adapt during testing by selecting appropriate items from item pools [6]. This approach has a natural counterpart within games where personalized [53], experience driven content generation [54] is capable of generating content that is appropriate to e.g. a player's skill level or emotional state. Additionally, research has shown how player models can be used to drive not only the generation of content that matches the player's desired experience, but also content that will reveal the maximal amount of information about the player. As such, the combination of player modeling and procedural content generation provides methods for games that may extend current practices within psychological testing.

### D. Testing Costs

A typical concern that may limit the application of psychological assessment today is the cost of deploying tests. Even though many psychological tests are now available in digital versions, ported from their original paper version and scored automatically by local or remote software, they still typically assume controlled environments and administration by professionals. Deploying psychological assessment solutions in the form of games on mobile devices will be able to tap into existing, standardized distribution platforms and may leverage existing data collection, aggregation, and analysis frameworks

[51]. Taking this into account, games applied as assessment tools might be able to extend current psychological testing practice through the infrastructure that games (and mobile games in particular) bring.

In the following section we go into deeper detail about how player modeling and procedural content generation may offer new methods for psychological testing.

## IV. PROCEDURAL CONTENT GENERATION IN ASSESSMENT GAMES

The promise of procedural content generation is to automatically create new games, new game variants or simply new game content each time a test is taken. By now, the general problem of generating content for games is fairly well understood, and a number of effective methods exist for e.g. generating levels, textures, puzzles, characters, and vegetation for games ranging from platformers to puzzle games to open-world adventures [55]. In the context of cognitive performance assessment games the role of procedural content generation would be to introduce variety, which serves both to avoid learning effects and to ensure continued player engagement.

Procedural generation can be applied to different levels in a game—in more constrained settings it might be a question of changing a few parameters, in less constrained settings a matter of generating new structural content such as a level, and in the least constrained setting procedural generation can be applied to the very rules of a game, creating new variants, but with similar underlying challenges.

We hypothesize that the less constrained the procedural generation is, the better learning effects can be avoided. If the test consists of the same game at each occasion with only minor variations in the parameters or the level, learning effects will likely only be partially mitigated—there will still be significant training benefits from having played another (similar) version of the same game. If instead an entirely new game is generated for each test, learning effects are likely to be negligible, as the time and effort spent taking previous tests is not likely to improve the individual's performance on the new test. On the other hand, the more the game changes between tests, the less comparable the results will be and reliability may suffer as a result and any prior established validity may be invalidated. This is a trade-off which will need to be explored in more depth in future research. Particularly investigating how much change along one or more dimensions impacts validity, as measured using e.g. well-known psychological tests as criteria, will be a significant challenge.

An initial strategy toward addressing this could be simulation-based testing of the generated games. This approach should be able to ascertain the level at which they challenge a particular cognitive skill, or in other words the performance/behavior in a game relative to a given latent construct. To do this, we envision developing computational agents that can learn to play any of these games with the same skill and playing style as a particular human player. We refer to these player-imitating agents as *procedural personas* [56], adaptive agents that learn to reproduce human play

skill, preferences, and implicitly playing styles in a single game [56]. This is done by identifying common patterns of skills and preferences in games, and biasing existing game-playing algorithms with these patterns. Such agents would be trained on a player's testing/playing history and become increasingly representative over time. Creating computational agents that can learn to play the testing games in a similar way to the human player/test-taker, and with the same performance, is not a trivial task, both because the performance needs to carry over from the game variant the agent was trained on to the new game variant, and because most computational game-playing agents tend to play in distinctly non-human-like ways. When a new game, or a new variation of a game, has been generated, it can first be played by the procedural persona to set a baseline for the player's expected performance on that game. This moves the problem of reliability from the particular instance of the test/game and instead places it on the persona as a user/player model. As long as this is representative of the player it should enable a procedural content generation system to choose configurations with an appropriate discriminatory ability. Prior work has demonstrated that it is possible to represent skill in game playing agents [57] and to bias game playing agents towards exhibiting more human-like play styles [58], [59].

Modeling players' performance and playing style can be taken even further in order to more accurately assess the player. The paradigm of Active Player Modeling uses active learning in player modeling [60]: the space of content is searched for the content areas where the model is least certain about the player's performance. In other words, the game content generator probes the content space to maximally improve its knowledge about the player. This approach could be very effective for exploratory cognitive assessment.

Active player modeling can be usefully compared to Computerized Adaptive Testing, where test items are chosen from a pool in order to provide the most appropriate test questions for a given test-taker in order to maximize the information gained from each item [6]. The difference to our envisioned system is that each configuration of the test/game, comparable to an item or an item set, is selected or generated and then configured based on simulations in response to the player model, which is continuously updated. This could allow for accurate generation of tests for individual test-takers, taking into account variation among multiple dimensions of cognitive performance.

Given our assumption that games applied as assessment instruments might bring new methods to the assessment of cognitive characteristics within motivation, reliability, sensitivity, and cost, we have developed a prototype test platform, which we describe in the following section.

## V. THE SKILLSHOW PROTOTYPE

In this section we describe the work-in-progress SkillShow prototype. The prototype is built in accordance with the three principles outlined above in Section I: game design should be applied to build motivation, player modeling in context should be applicable through game playing agents, and

TABLE I  
OVERVIEW OF THE DESIGN CHARACTERISTICS OF THE *Interrupted SET* TEST/GAME.

<b>Game genre/template</b>	Card game: SET
<b>Game mechanics</b>	Set identification, sorting, search
<b>Test inspiration</b>	SIMKAP
<b>Latent construct</b>	Simultaneous capacity

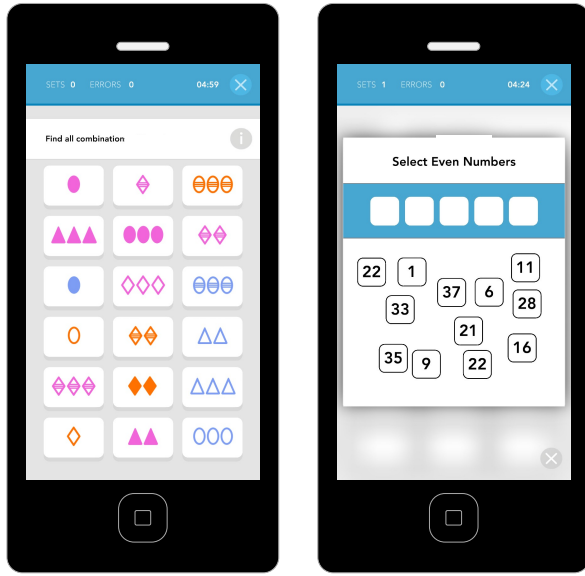
procedural content generation should be supported to ensure novelty and maximal individual information gain from each test session. In this application the three principles are applied in moderation and the application design still leans heavily on existing psychological tests. SkillShow is designed to measure two different latent constructs: *simultaneous capacity* [8] and *inductive reasoning intelligence* [9]. Each construct will be measured through a test/game that combines and adapts existing game designs and existing psychological tests. Given that we can successfully demonstrate that the suggested approaches work for these tests/games, we envision moving on to more complex tests/games later. Below, we describe each of the two tasks that we currently plan to include in our user studies.

### A. Interrupted SET

The first mode of SkillShow is titled *Interrupted SET*. The fundamental psychological task is drawn from a well-known assessment instrument called *SIMKAP* [7], [8]. It is typically used in the assessment and selection of personnel for critical functions such as ship captains, fighter pilots, or air traffic controllers. *SIMKAP* measures a construct called simultaneous capacity: an individual's ability to perform several mental operations simultaneously and switching between these tasks based on outside demands. In order to fuse the properties of the *SIMKAP* test with a motivating game design that supports player modeling and adaptive procedural content generation, we borrow and adapt the rules of the game *SET* [61]. The game requires players to analyze a selection of 18 cards in order to create sets of three cards that are either all identical or all unique on four dimensions: count, color, shape, and fill type. The player must identify as many sets as possible within a time limit. While the player is solving this task we intermittently interrupt the player by presenting distracting, overlaid tasks that require the player to either sort integers or conduct visual search for characters. The resulting test/game is characterized in Table I and the two game modes are displayed in Figure 1. We expect the challenge to be configurable through the number of sets present in the each initial card spread and the frequency and complexity of the distracting tasks.

### B. Simple MULTIFLUX

The second mode of SkillShow is titled *Simple MULTIFLUX* and is adapted from the work of Kröner et al. who show how interactive computer simulations can be used to assess *inductive reasoning intelligence* [9]. They develop a simulation-based task where the subject must go through three stages of understanding a dynamic system: identifying rules,



(a) Identifying sets. (b) One kind of interruption.

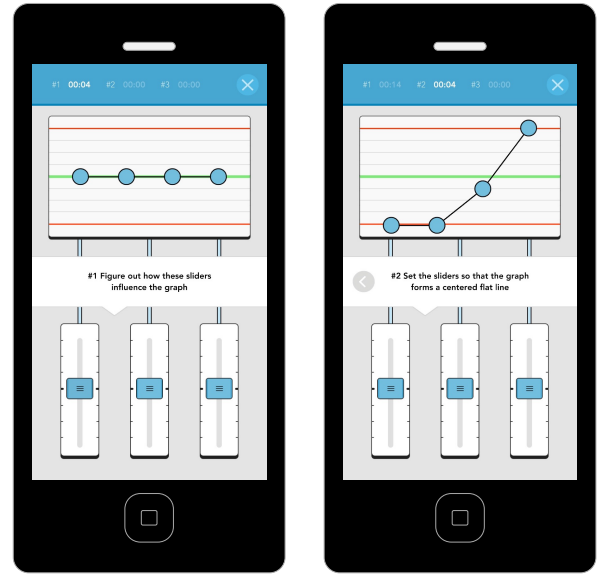
Fig. 1. Two modes of Interrupted SET.

TABLE II  
OVERVIEW OF THE DESIGN CHARACTERISTICS OF THE *Simple MULTIFLUX* TEST/GAME.

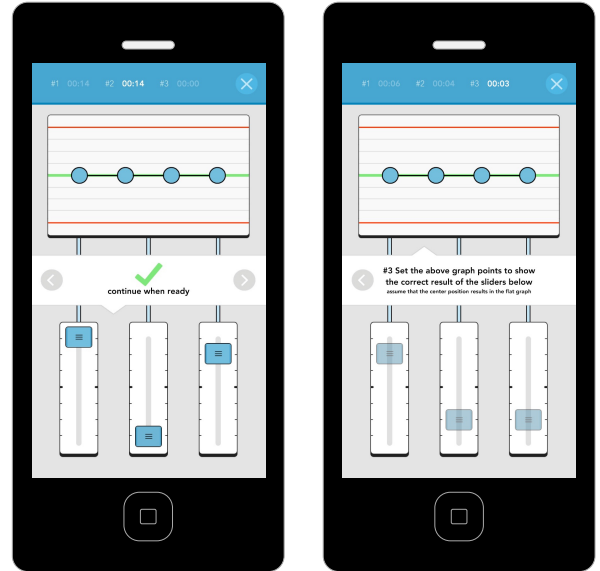
<b>Game genre/template</b>	Puzzle games
<b>Game mechanics</b>	Dynamic systems learning/manipulation
<b>Test inspiration</b>	MULTIFLUX
<b>Latent construct</b>	Inductive reasoning intelligence

applying rules, and demonstrating understanding of rules. First, the subject is given time to analyze the relations between a number of control inputs and the outputs of an abstract, simulated machine. Secondly, the subject is given an instance of this abstract machine and asked to bring it into a particular goal state using the acquired information. Thirdly, the subject is given a pre-configured machine and asked to determine how the controls must be arranged to cause this output. This test is reminiscent of many abstract puzzle games and well suited to the mobile format. A design overview of this test/game is shown in Table II. In order to increase the appeal of the task, we designed a compelling user interface, simplifying the display and adding visual, auditive, and animation feedback. We did not change the core rules of the test as this was deemed appropriate and engaging in its original form. The resulting design is displayed in Figure 2.

The original MULTIFLUX test has four inputs and four outputs and gave subjects 7 minutes and 30 seconds to solve the tasks. Through informal experimentation, we deemed this to be too complex, and reduced the problem to our *Simple MULTIFLUX* variant with only three inputs and three outputs and a 5 minute time limit. We expect the difficulty of this test/game to be configurable via the complexity of the function that maps inputs to outputs in the simulated machine.



(a) Learning condition. (b) Application condition.



(c) Solved application condition. (d) Demonstration condition.

Fig. 2. Four states of Simple MULTIFLUX.

### C. Feedback and Evaluation Screens

In order to support player motivation and adherence both test/games in SkillShow are built with rich visual and auditive feedback. The tests/games are built to a production value that would seem familiar to a player used to playing premium puzzle games on their mobile phone. Additionally, the application includes immediate feedback after each play session. The application rates the player's performance in the latest play in relation to previous performances and displays a graph with recent sessions, shown in Figure 3.

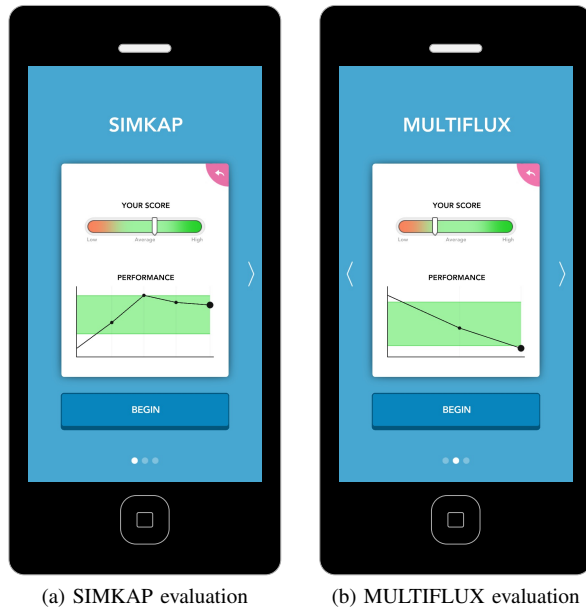


Fig. 3. Two evaluation screens from SkillShow providing individualized feedback.

## VI. EVALUATING THE SKILLSHOW APPLICATION

The next step for evaluating the SkillShow application's usefulness for daily cognitive assessment will be to conduct a pilot user study with the two games. At the present time, two user study configurations are planned.

First, since the tests/games are based on existing psychological assessment instruments, a longitudinal study where subjects/players play the two games once a day will be run. Intermittently, during this period of time, the players will be asked to complete the original tasks under lab conditions. In addition to conducting the original, validated tasks that the tests/games are based upon, the subjects/players will also be asked to complete a battery of other, validated tests measuring well-known psychological constructs, such as attention and intelligence. Additionally, once a baseline is established for the players, a number of interventions will be staged, where participants will be split into a treatment group and a control group. The treatment group will undergo a procedure known to reduce cognitive performance; currently we are contemplating a moderate amount of sleep deprivation. We expect the treatment group to exhibit significantly worse performance in both games on the occasion when they are sleep deprived, relative to their baseline, and we expect them to return to this baseline when they are no longer sleep deprived. We expect this effect to be present for both the reference psychological tests, and the SkillShow versions. For the control group, we expect no effect other than performance increases due to learning effects.

Once this pilot study is complete we intend to build individual player models, modeling skill and style over time. This, in turn, will allow us to construct procedural difficulty adjustment systems for each task, that may be used in a subsequent second pilot study. If learning and development of skill is observed

in the first study, the purpose of this second study will be to enable the SkillShow application to keep challenge steady and maximize information gain by adjusting the difficulty of the tasks to match the players' skill development.

## VII. CONCLUSION

In this paper, we outlined a number of reasons, based on the literature, to assume that games applied as cognitive tests may be able to facilitate daily cognitive assessment through motivating, reliable, personalized, and cost effective tests. We described four key ways in which combining this approach with player modeling and procedural content generation might bring novel methods to cognitive testing in general. Additionally, we described a prototype for a first pilot user study testing the efficacy of games as daily assessment tools. The tasks included in the prototype are relatively conservative interpretations of existing psychological tests, but fused with game mechanics drawn from existing games, and game design principles such as such rich interfaces and feedback. If the pilot-studies show the tests/games can provide valid and reliable assessment we will expand the prototype application with more elaborate games, player modeling, and procedural content generation.

The overarching vision described in this paper is to explore the possibilities in combining key elements from game design and computational intelligence in games, specifically player modeling and procedural content generation, with psychological testing. The SkillShow prototype represents our first step in exploring these possibilities.

## ACKNOWLEDGMENT

The SkillShow application is developed and owned by Apex Group ApS and Informatics ApS.

## REFERENCES

- [1] R. Gregory, *Psychological Testing: History, Principles, and Applications*. Pearson, 2011.
- [2] J. M. Cattell, "Mental tests and measurements with remarks by F. Galton," *Mind*, vol. 15, no. 59, pp. 373–381, 1890.
- [3] M. R. Lepper and T. W. Malone, "Intrinsic motivation and instructional effectiveness in computer-based education," *Aptitude, learning, and instruction*, vol. 3, pp. 255–286, 1987.
- [4] C. Klimmt and T. Hartmann, "Effectance, self-efficacy, and the motivation to play video games," 2006.
- [5] F. B. Baker, *The basics of item response theory*. ERIC, 2001.
- [6] H. Wainer, N. J. Dorans, R. Flaugher, B. F. Green, and R. J. Mislevy, *Computerized adaptive testing: A primer*. Routledge, 2000.
- [7] B. Rosmark, "Validering av ett simultankapacitetstests prediktionsförmåga av framgång i utbildningen av båtschefer (stridsbåtsförare)," *Stockholm, Sweden: Pliktverket Regionkontor Stockholm*, 2001.
- [8] O. Bratfisch and E. Hagman, "Simultankapazität/multi-tasking (simkap) version 24.00: Handanweisung (simultaneous capacity/multi-tasking (simkap) release 24.00: Manual)," *Mödling, Austria: Schuhfried*, 2003.
- [9] S. Kröner, J. Plass, and D. Leutner, "Intelligence assessment with computer simulations," *Intelligence*, vol. 33, no. 4, pp. 347–368, 2005.
- [10] N. Chmiel, *An introduction to work and organizational psychology: a European perspective*. John Wiley & Sons, 2008.
- [11] J. M. Weinhardt and J. B. Vancouver, "Computational models and organizational psychology: Opportunities abound," *Organizational Psychology Review*, vol. 2, no. 4, pp. 267–292, 2012.
- [12] J. Chin, R. Dukes, and W. Gamson, "Assessment in simulation and gaming: a review of the last 40 years," *Simulation & Gaming*, vol. 40, no. 4, pp. 553–568, 2009.

- [13] H. H. Spitz, "The universal nature of human intelligence: Evidence from games," *Intelligence*, vol. 2, no. 4, pp. 371–379, 1978.
- [14] M. B. Jones, R. S. Kennedy, and A. C. Bittner Jr, "A video game for performance testing," *The American Journal of Psychology*, pp. 143–152, 1981.
- [15] M. B. Jones, "Video games as psychological tests," *Simulation & Games*, 1984.
- [16] P. Rabbitt, N. Banerji, and A. Szymanski, "Space fortress as an iq test? predictions of learning and of practised performance in a complex interactive video-game," *Acta Psychologica*, vol. 71, no. 1, pp. 243–257, 1989.
- [17] E. Donchin, "Video games as research tools: The space fortress game," *Behavior Research Methods, Instruments, & Computers*, vol. 27, no. 2, pp. 217–223, 1995.
- [18] G. Van Lankveld, S. Schreurs, and P. Spronck, "Psychologically verified player modelling," in *GAMEON*, 2009, pp. 12–19.
- [19] G. Van Lankveld, S. Schreurs, P. Spronck, and J. Van Den Herik, "Extraversion in games," in *International Conference on Computers and Games*. Springer, 2010, pp. 263–275.
- [20] G. van Lankveld, P. Spronck, J. Van den Herik, and A. Arntz, "Games as personality profiling tools," in *Conference on Computational Intelligence and Games*. IEEE, 2011, pp. 197–202.
- [21] N. Yee, N. Ducheneaut, L. Nelson, and P. Likarish, "Introverted elves & conscientious gnomes: The expression of personality in world of warcraft," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2011, pp. 753–762.
- [22] A. Canossa, J. B. Martinez, and J. Togelius, "Give me a reason to dig: Minecraft and psychology of motivation," in *Conference on Computational Intelligence and Games*. IEEE, 2013, pp. 1–8.
- [23] A. Canossa, J. B. Badler, M. S. El-Nasr, S. Tignor, and C. R. Colvin, "In your face(t). impact of personality and context on gameplay behavior," in *Foundations of Digital Games*, 2015.
- [24] S. Tekofsky, P. Spronck, A. Plaat, J. Van den Herik, and J. Broersen, "Psyops: Personality assessment through gaming behavior," in *BNAIC 2013: Proceedings of the 25th Benelux Conference on Artificial Intelligence*, 2013.
- [25] EA DICE, *Battlefield 3*. Electronic Arts, 2011.
- [26] S. Tekofsky, P. Spronck, M. Goudbeek, A. Plaat, and J. van den Herik, "Past our prime: A study of age and play style development in battlefield 3," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 7, no. 3, pp. 292–303, 2015.
- [27] W. R. Boot, *Video games as tools to achieve insight into cognitive processes*. Frontiers Media SA, 2015.
- [28] R. Hunicke, "The case for dynamic difficulty adjustment in games," in *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*. ACM, 2005, pp. 429–433.
- [29] R. Dias and C. Martinho, "Adapting content presentation and control to player personality in videogames," in *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology*. ACM, 2011, p. 18.
- [30] S. C. Bakkes, P. H. Spronck, and G. van Lankveld, "Player behavioural modelling for video games," *Entertainment Computing*, vol. 3, no. 3, pp. 71–79, 2012.
- [31] G. N. Yannakakis, P. Spronck, D. Loiacono, and E. André, "Player Modeling," in *Artificial and Computational Intelligence in Games*. Saarbrücken/Wadern: Dagstuhl Publishing, 2013, pp. 45–55.
- [32] J. Robison, S. McQuiggan, and J. Lester, "Evaluating the consequences of affective feedback in intelligent tutoring systems," in *3rd International Conference on Affective Computing and Intelligent Interaction and Workshops*. IEEE, 2009, pp. 1–6.
- [33] M. D. Kickmeier-Rust and D. Albert, "Educationally adaptive: Balancing serious games," *International Journal of Computer Science in Sport*, vol. 11, no. 1, 2012.
- [34] H. Wang, H.-T. Yang, and C.-T. Sun, "Thinking style and team competition game performance and enjoyment," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 7, no. 3, pp. 243–254, 2015.
- [35] V. J. Shute and R. Glaser, "A large-scale evaluation of an intelligent discovery world: Smithtown," *Interactive Learning Environments*, vol. 1, no. 1, pp. 51–77, 1990.
- [36] V. J. Shute, "Stealth assessment in computer-based games to support learning," *Computer games and Instruction*, vol. 55, no. 2, pp. 503–524, 2011.
- [37] A. B. Collmus, M. B. Armstrong, and R. N. Landers, "Game-thinking within social media to recruit and select job candidates," in *Social Media in Employee Selection and Recruitment*. Springer, 2016, pp. 103–124.
- [38] Bethesda Game Studios, *Fallout 3*. Bethesda Softworks, 2008.
- [39] Blizzard Entertainment, *StarCraft II: Wings of Liberty*. Blizzard Entertainment, 2010.
- [40] J. J. Thompson, M. R. Blair, and A. J. Henrey, "Over the hill at 24: persistent age-related cognitive-motor decline in reaction times in an ecologically valid video game task begins in early adulthood," *PloS one*, vol. 9, no. 4, p. e94215, 2014.
- [41] A. Lieberoth, "Shallow gamification. testing psychological effects of framing an activity as a game," *Games and Culture*, vol. 10, no. 3, pp. 229–248, 2015.
- [42] C. M. Steele and J. Aronson, "Stereotype Threat and the Intellectual Test Performance of African Americans," *Journal of Personality and Social Psychology*, vol. 69, no. 5, p. 797, 1995.
- [43] J. E. Beck, "Engagement tracing: using response times to model student disengagement," in *Artificial Intelligence in Education*. IOS Press, 2005, pp. 88–95.
- [44] T. Fullerton, C. Swain, and S. Hoffman, *Game Design Workshop: Designing, Prototyping, and Playtesting Games*. Focal Press, 2004.
- [45] K. Salen and E. Zimmerman, *Rules of Play: Game design fundamentals*. MIT press, 2004.
- [46] P. H. M. Spronck *et al.*, *Adaptive Game AI*. UPM, Universitaire Pers Maastricht, 2005.
- [47] A. Zook, S. Lee-Urban, M. R. Drinkwater, and M. O. Riedl, "Skill-based mission generation: A data-driven temporal player modeling approach," in *Proceedings of the Third Workshop on Procedural Content Generation in Games*. ACM, 2012, p. 6.
- [48] G. N. Yannakakis and J. Togelius, "A Panorama of Artificial and Computational Intelligence in Games," *IEEE Transactions on Computational Intelligence and AI in Games*, no. 99, p. 1, 2014.
- [49] A. Pajitnov and V. Pokhilko, *Tetris*. Alexey Pajitnov, 1984.
- [50] G. S. Elias, R. Garfield, and K. R. Gutschera, *Characteristics of Games*. MIT Press, 2012.
- [51] M. S. El-Nasr, A. Drachen, and A. Canossa, *Game Analytics: Maximizing the Value of Player Data*. Springer Science & Business Media, 2013.
- [52] J. J. Thompson, M. R. Blair, L. Chen, and A. J. Henrey, "Video game telemetry as a critical tool in the study of complex skill learning," *PloS one*, vol. 8, no. 9, p. e75129, 2013.
- [53] N. Shaker, G. Yannakakis, J. Togelius, M. Nicolau, and M. O'Neill, "Evolving personalized content for super mario bros using grammatical evolution," in *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2012.
- [54] G. N. Yannakakis and J. Togelius, "Experience-driven procedural content generation," *IEEE Transactions on Affective Computing*, vol. 2, no. 3, pp. 147–161, 2011.
- [55] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2016.
- [56] C. Holmgård, A. Liapis, J. Togelius, and G. N. Yannakakis, "Evolving personas for player decision modeling," in *Conference on Computational Intelligence and Games*. IEEE, 2014.
- [57] A. Zook, B. Harrison, and M. O. Riedl, "Monte-carlo tree search for simulation-based strategy analysis," in *Foundations of Digital Games*, 2015.
- [58] D. Whitehouse, P. I. Cowling, E. J. Powley, and J. Rollason, "Integrating monte carlo tree search with knowledge-based methods to create engaging play in a commercial mobile game," in *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2013.
- [59] A. Khalifa, A. Isaksen, J. Togelius, and A. Nealen, "Modifying mcts for human-like general video game playing," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)*, 2016.
- [60] J. Togelius, N. Shaker, and G. N. Yannakakis, "Active player modelling," in *Foundations of Digital Games*, 2014.
- [61] M. J. Falco, J. Langdon, and F. Vohwinkel, *SET*. 999 Games, 1988.



# Changing Video Game Graphic Styles Using Neural Algorithms

Byungho Yoo

Department of Computer Science and Engineering  
Sejong University, Seoul, South Korea  
bingo0080@gmail.com

Kyung-Joong Kim

Department of Computer Science and Engineering  
Sejong University, Seoul, South Korea  
kimkj@sejong.ac.kr

**Abstract**—Recently, procedural content generation (PCG) has attracted positive attentions from gamers and applied for various content types such as maps, items and so on. Deep neural networks have been reported that they have potential to learn styles of artistic images. In this study, we propose to apply convolutional neural networks to change artistic styles of video game graphics. It's expected to change original games into different styles (modern, old-fashioned, scientific, and so on) given the input images. We applied the neural styling algorithm to the game images from Hedgewars, an open-source turn-based strategy game. Our results show that styles of video games can be changed from an input styling image.

**Keywords**—Procedural content generation, Convolutional neural network, Image conversion, Deep learning application

## I. INTRODUCTION

Deep learning has been successfully applied to some game AI applications. Most of them focused on building AI players from reinforcement or supervised learning. In recent studies, they used the game screen as inputs to the AI player and the deep neural networks process the raw pixel information to select proper actions. Google's recent success on Go demonstrated the potential of deep reinforcement learning to solve very complex games.

It has been known that the convolutional neural network (CNN) is not just powerful on image recognition but also useful for image styling and super resolution. Recently, Champandard applied deep learning to scale up and style pixel art for Minecraft textures [1].







In this study, we applied a neural styling algorithm to all game images from a single open-source game. Our research question is to see what makes difficult to use the neural styling for all aspects of games. Because game images have unique property separated from other graphics, artistic, or photo images, it's important to design a system suitable for the game graphics styling.

## II. A NEURAL ALGORITHM OF ARTISTIC STYLE

Gatys *et al.*, used convolutional neural network to separate contents and styles of artistic images [2]. They demonstrated that the neural network can combine contents of arbitrary images with styles of well-known artists' works. They used Vincent van Gogh's The Starry Night for style representation and created photographs with the style. Because the source code

was open to the public, there are web services to support the styling of small number of images [3].

Fig. 1 Applying neural styling algorithms to commercial game screen shots (it just converted a single screen image using <https://dreamscopeapp.com/>)

Screen shot	Outputs from Neural Styling Algorithm (with names of style input images)	
		
Angry Birds	Clayton Kashuba	Picasso Blue
		
StarCraft	Winter Solstice	Blue Mosaic

## III. PROPOSED SYSTEMS AND EXPERIMENTS

It needs high-end graphic cards to get high quality outcomes because it uses GPGPU. The higher outcome image resolution is necessary, the more memory space is required. For example, 7 GB of graphic card memory size is recommended to apply neural styling for an 1024 by 1024 pixels image. Even 800 x 800 images, it needs 5GB memory for the neural styling. Most of graphic card with GPUs support 2GB ~ 4GB memory. When the resolution exceeds the memory requirement, it's rescaled to affordable image size and processed by the neural styling algorithm. The outcome images are rescaled to the original input size. Because recent state-of-the art graphic cards provide up to 12GB memory size, it's possible to convert them without resizing with special hardware support. Table 1 summarizes the specification of our hardware systems. In our system, images larger than 800 by 800 or smaller than 64 by 64 are rescaled.

TABLE I. HARDWARE SEPCIFICATION USED IN NEURAL STYLING

CPU	intel(R) Core(TM) i7-4790 3.60 Hz
RAM	16.0GB
GPU	Geforce GTX 980 6GB GDDR5

There are some parameters to be tuned for the styling. For example, content weight can control the loss of original content

and important to adjust the level of change. It can balance between the style and content representation to produce combined outcomes.

Fig. 2 Styling of game play screens using the input image and convolutional neural networks (two videos of the game playing is provided in the supplementary part.)

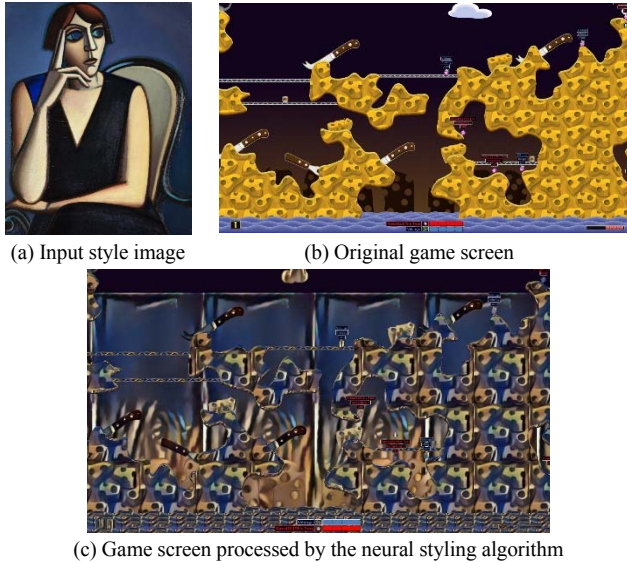
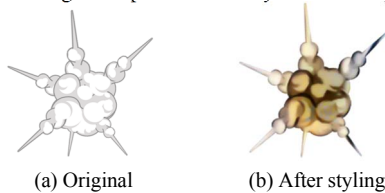


Fig. 3 Change of explosion effect by the neural styling



Hedgewars is an open-source turn-based strategy game (<http://www.hedgewars.org/>). When we select the target game for this experiment, we search for an open-source game with all the game images accessible in a widely acceptable file format such as JPEG or PNG. Some open-source games are excluded because parts of images are stored as resources not supported in neural styling.

In the game, the total size of all images files (formatted in JPEG and PNG) is 70MB. We applied the neural styling algorithm [4] to the images one by one and took about 12 hours. For the big size images, it included rescaling process and PNG files with alpha channel, it did reconstruction of the transparency channel after neural styling. PIL (Python Imaging Library) was used. Although the neural styling is promising, it's not yet realistic to do the conversion of all game images in real-time. Instead, the game designers prepare several sets of game images styled with different input images (modern, old-fashioned, scientific and so on) and allow users to select their preference when they play the game. If the styling is targeting only the small portion of all image files, it's realistic to apply them in real-time. For example, the conversion is applied only to the main character or small-sized items.

In this study, we just used a single input style image to convert all the game images. However, it's more desirable to use several styles to convert game images. For example, different styles can be used for background and foreground objects. Also, it's necessary to apply the styling selectively and some images are not suitable for the change. Images with letters and symbols can be damaged after the styling.

#### IV. CONCLUSIONS AND FUTURE WORKS

The outcomes show the potential of neural styling for video games. In this study, we just applied to a single game with one input style. There is still enough room for improvement for better use of the automatic styling in video games. We found that the neural styling has not yet fully supported all kinds of graphic images from games. We addressed issues to handle large size image files, real-time processing, alpha channel in PNG file format, and letter/symbols distortion. There are additional potential problems to be considered for wide use of this system. For example, they're as follows.

- **Content descriptions:** Assuming that certain game item description says it is a sword, but this algorithm transmutes the item design into something else which is not seen as a sword.
- **Paired contents:** Some contents are paired each other such as a key and locked door. Because they're designed to be connected, it's important to keep the contents in their original position.
- **Continuous motions:** Animation of characters are coming from lots of frame images. It's not fully tested that the neural styling can produce smooth transition of similar frames.

#### ACKNOWLEDGEMENTS

This work was supported by the National Research Foundation of Korea (NRF) grant (2013 R1A2A2A01016589), National program for Excellence in Software program (R7718-16-1005), Ministry of Culture, Sports and Tourism (MCST) and Korea Creative Content Agency (KOCCA) in the Culture Technology (CT) Research & Development Program 2016.

#### SUPPLEMENTARY VIDEOS

(Before) <https://www.youtube.com/watch?v=odbl6lIQFjs>  
(After) <https://www.youtube.com/watch?v=XW1uMpm1cRc>

#### REFERENCES

- [1] nucl.ai. (2016). *Minecraft, ENHANCE! Neural Networks to Upscale & Stylize Pixel Art*. [online] Available at: <https://nucl.ai/blog/enhance-pixel-art/> [Accessed 7 Jul. 2016].
- [2] Gatys, L., Ecker, A. and Bethge, M. (2015). *A Neural Algorithm of Artistic Style*. [online] Arxiv.org. Available at: <http://arxiv.org/abs/1508.06576> [Accessed 4 May 2016].
- [3] Deepart.io. (2016). [online] Available at: <https://deepart.io/hire/> [Accessed 7 Jul. 2016].
- [4] GitHub. (2016). dmlc/mxnet. [online] Available at: <https://github.com/dmlc/mxnet/tree/master/example/neural-style> [Accessed 31 May 2016].

# Evolving Missions for *Dwarf Quest* Dungeons

Daniel Karavolos  
Institute of Digital Games  
University of Malta

e-mail: daniel.karavolos@um.edu.mt

Antonios Liapis  
Institute of Digital Games  
University of Malta

e-mail: antonios.liapis@um.edu.mt

Georgios N. Yannakakis  
Institute of Digital Games  
University of Malta

e-mail: georgios.yannakakis@um.edu.mt

**Abstract**—This paper describes a search-based level generation approach that uses the search space of action sequences, represented as graphs, rather than spatial layouts. The search is guided by mutation operators that manipulate the graph topology, and the paper explores various objective functions that are based on generic level evaluation metrics. The evolved action sequences are passed to a grammar-based system and a layout solver transforms them into dungeon levels for the *Dwarf Quest* game.

## I. INTRODUCTION

Procedural content generation (PCG) in games has received considerable attention; this paper uses search-based PCG approaches [1] to evolve action sequences (*missions*) of a hero traversing a dungeon, which can then be transformed into a dungeon level. The dual representation for game levels (as a mission and as a space) was introduced in [2] and expanded in [3], where the mission graph was created via a graph grammar while the architecture was built from shape grammars which rewrite mission nodes into rooms of various sizes. Using an indirect representation of levels as mission graphs, the generator can evolve the player's sequence of key actions rather than the explicit sequence of rooms they have to visit. While the level geometry and the action sequence are linked (i.e. the latter constrains the former), the action sequence is a more concise representation as it omits trivial information (e.g. empty rooms or walls). Moreover, the action sequences are represented as a graph of nodes which can evolve via simpler genetic operators with a better locality. Finally, parsing the graph directly allows for fast and simple evaluations of the decision density of a player traversing a level from start to finish. This demo paper focuses on the generation of missions for the dungeon crawl game *Dwarf Quest* (Wild Card Games 2013). The final levels, showcased in Fig. 3, can be played in the *Dwarf Quest* engine (see Fig. 1).



Fig. 1. Screenshot of a *fight* in a generated level in *Dwarf Quest*.

## II. MISSION EVOLUTION

The algorithm evolves mission graphs represented as a list of nodes and edges. Nodes represent abstract player actions, such as solving a puzzle. This abstract action will later be transformed into a specific action by a grammar, which is then transformed into one or more rooms in which the action will take place by a layout solver. There are 14 types of nodes, split into four categories: *fight* (i.e. monsters), *puzzle* (i.e. environmental hazards), *reward* (i.e. powerups), and *neutral* (i.e. start or end of the dungeon). The hero's goal is to traverse the mission graph starting from the start node and reaching the end node. A more detailed description of node types and their transformation into level spaces is provided in [4]. For evolution, each node is stored as an integer (identifying its node type). Edges connect two nodes, and are represented by three parameters: the index of the starting node, the index of the ending node, and a flag on whether the edge is directed.

Following the search-based PCG paradigm, an initial population of mission graphs (with only the start and end node) is evolved to maximize a fitness function of one or more objectives. Evolution is carried out via mutation alone, and increases the topology of these initial individuals. Mutation operators can insert a new node between two existing nodes (linking them appropriately), deleting a random node or changing its type, or adding and deleting edges between two random nodes. To ensure that mutation is not destructive, mutation operators do not affect the start or end node and do not place more than one boss node or one altar node per level.

Inspired in part by generic level evaluations of [5], five fitness dimensions are designed to drive evolution (alone or combined into a weighted sum). Based on preliminary tests and designer intuition, all metrics were normalized to  $[0, 1]$ , with high scores assigned to (qualitatively) desirable content.

The five fitnesses include: (a) shortest path ( $f_p$ ) between start and end nodes in terms of nodes, normalized to reward paths of 5 to 10 nodes; (b) exploration ( $f_e$ ) which uses flood fill from the start node to evaluate how many nodes the hero visits before reaching the end node, normalized to reward explored nodes equal to three times the nodes on the shortest path; (c) variation ( $f_v$ ) as the ratio of edges connecting nodes of different categories (except neutral nodes); (d) dispersed rewards ( $f_s$ ) as the number of nodes considered *safe* [5] to rewards<sup>1</sup>; (e) balanced rewards ( $f_b$ ) evaluating if each reward

<sup>1</sup>Safe nodes are much closer to one reward versus all other rewards.



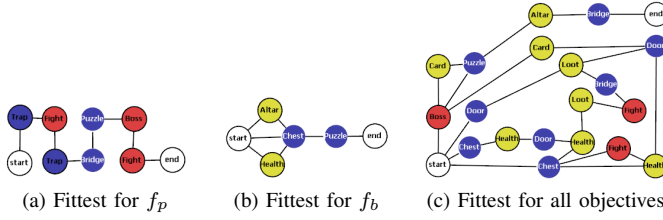


Fig. 2. Mission graphs of the fittest individuals for certain fitnesses. Graphs have white (neutral), red (fight), yellow (reward), and blue (puzzle) nodes.

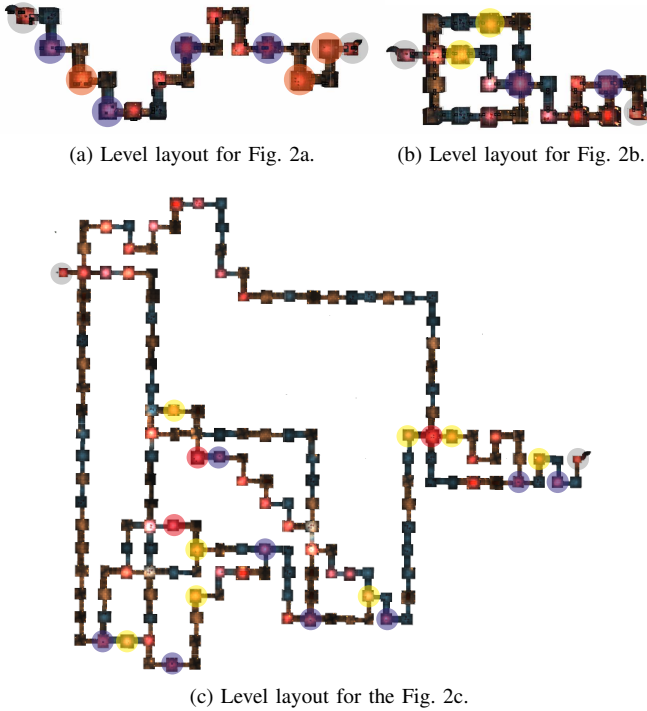


Fig. 3. Level layouts created from the fittest mission graphs of Fig. 2. Rooms included in the mission graph are highlighted as circles of the same colors as Fig. 2. Gray circles are the start node (left-most) and end node (right-most).

has an equal number of safe nodes around it.

The game’s final levels are created via the mixed-initiative grammar-based system of [4], which transforms the evolved mission graphs into a larger and more detailed graph. This in turn is converted into *Dwarf Quest* levels by the layout solver described in [6]. Post-processing edits are applied to the evolved mission graphs, to abide to the layout solver’s constraints (e.g. as rooms in *Dwarf Quest* must have two or more doors, non-neutral nodes with one edge are omitted).

### III. RESULTS

Figure 2 shows the fittest mission graphs for some sample objectives, and a combination of all objectives. The graph for  $f_p$  is a linear path to the end node, since this objective does not reward branching. The graph for  $f_b$  has two rewards placed symmetrically to all other nodes: due to the reward nodes’ edges, all nodes are actually unsafe (i.e. equally close) to both rewards and thus the mission graph is “balanced” in terms

of safe areas around rewards. This example highlights some artifacts which can be caused when evolving towards a single objective without e.g. rewarding more safe nodes (via  $f_s$ ).

When all objectives are optimized, the resulting graph is much larger than the other graphs, which is caused by the interaction between  $f_e$  and  $f_p$ ; due to  $f_v$ , there are no two nodes of the same type adjacent to each other. Furthermore, due to the high branching factor at the start, two ‘areas’ seem to emerge; a straight path to the end node, and a maze-like structure that either leads back to the start or to a boss fight.

Figure 3 illustrates level architectures for *Dwarf Quest* based on the evolved mission graphs of Figure 2. The actual rooms which contain nodes in the mission graph are shown in circles of different colors. The level in Fig. 3c is created from the mission graph of Fig. 2c, and it is immediately obvious that most rooms in the final *Dwarf Quest* level are empty and in many cases form long corridors to connect the nodes. This is due to the graph’s high branching factor, which forces the layout solver to connect nodes that are spatially far away. In contrast, the central part of the dungeon has fewer empty rooms, with only a couple of rooms between each pair of mission graph nodes. The simpler graphs of Fig. 2a and 2b, however, result in levels with few empty rooms, and thus the layout solver seems less suited for creating levels with high branching factors or complex topologies.

### IV. CONCLUSION

This demo paper described an approach for generating game levels by evolving their indirect representation rather than their layout. Mission graphs representing the possible action sequences of the player for reaching the end of the level were evolved towards different objectives inspired by general game design patterns such as exploration, balance and safety.

### V. ACKNOWLEDGMENT

We would like to thank Dylan Nagel for giving us full access to *Dwarf Quest*’s source code, as well as Rafael Bidarra and Roland van der Linden for the layout solver of *Dwarf Quest*. This work was supported, in part, by the FP7 Marie Curie CIG project AutoGameDesign (project no: 630665).

### REFERENCES

- [1] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, “Search-based procedural content generation: A taxonomy and survey,” *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 3, no. 3, pp. 172–186, 2011.
- [2] J. Dormans, “Adventures in level design: generating missions and spaces for action adventure games,” in *Proceedings of the FDG Workshop on Procedural Content Generation in Games*, 2010.
- [3] J. Dormans and S. C. J. Bakkes, “Generating missions and spaces for adaptable play experiences,” *IEEE Transactions on Computational Intelligence and AI in Games. Special Issue on Procedural Content Generation*, vol. 3, no. 3, pp. 216–228, 2011.
- [4] D. Karavolos, A. Bouwer, and R. Bidarra, “Mixed-initiative design of game levels: Integrating mission and space into level generation,” in *Proceedings of the Foundations of Digital Games Conference*, 2015.
- [5] A. Liapis, G. N. Yannakakis, and J. Togelius, “Towards a generic method of evaluating game levels,” *Proceedings of the AAAI Artificial Intelligence for Interactive Digital Entertainment Conference*, 2013.
- [6] R. van der Linden, “Designing procedurally generated levels,” Master’s thesis, TU Delft, 2013.

# Stylized Facts for Mobile Game Analytics

Anders Drachen<sup>\*</sup>, Nicholas Ross<sup>†</sup>, Julian Runge<sup>‡</sup> and Rafet Sifa<sup>§</sup>

<sup>\*</sup>Aalborg University

Email: andersdrachen@gmail.com

<sup>†</sup>University of San Francisco, San Francisco, California 94105

Email: ncross@usfca.edu

<sup>‡</sup>Wooga, Germany

Email: julian.runge@gmail.com

<sup>§</sup> Fraunhofer IAIS

Email: rafet.sifa@iais.fraunhofer.de

**Abstract**—There are numerous widely disseminated beliefs in the rapidly growing domain of Mobile Game Analytics, notably within the context of the Free-to-Play model. However, the field remains in its infancy, as there is limited conclusive empirical knowledge available across industry and academia, to provide evidence for these beliefs. Additionally, the current knowledge base is highly fragmented. For Mobile Game Analytics to mature, empirical frameworks are needed. In this paper the concept of stylized facts is presented as a means to develop an initial framework for a common understanding of key hypotheses and concepts in the field, as well as organizing the available empirical knowledge. A focus on stylized facts research will not only facilitate communication but also, more importantly, improve the quality and actionability of insights. Unified terminology and a comprehensive collection of stylized facts can be the building blocks for a conceptually well-founded understanding of mobile gaming.

## I. INTRODUCTION

Game Analytics as a domain of research and inquiry has rapidly emerged within the past ten years, going from being virtually unknown in industry and academia to forming a core part of game development and research [1]–[8]. Within the larger umbrella of Game Analytics, mobile games have grown to form a substantial part of the industry in terms of both revenue and number of games developed. For example, in 2015 the Apple App Store carried almost 400,000 games on its mobile platform, a platform that did not exist ten years prior [9]. Academic and industry games research has expanded in parallel [10]–[13]. The vast majority of mobile games follow the freemium business model, i.e. they are Free-to-Play (F2P,FtP) and generate revenue via In-App Purchases (IAPs) [14]. F2P games are thus dependent on the willingness of users to make purchases, and therefore require analytical support. This has prompted a surge in research on player behavior, with a focus on profiling, monetization, funnel analysis, onboarding, prediction and the impact of design [1], [5], [8], [15].

A byproduct of this rapid development is a general immaturity in the field, the common symptom of which is an uneven level of analytical capacity across both industry and academics. This is especially true for Small-Medium sized Enterprises (“SMEs”) which have trouble keeping up due to the specialized knowledge and investment required

to take advantage of behavioral data. On the academic side, researchers struggle to keep up with the work being done in a field where industry has access to both more resources and more data. In essence, the field is in its infancy and the available knowledge is heavily fragmented, not the least due to Game Analytics interdisciplinary nature and the lack of knowledge sharing between academia and industry. One result of this immaturity is a lack of standardization and effective communication between industry and academics [8].

The root cause of these problems is the lack of a framework for organizing current knowledge and prioritizing interesting problems. This is unlike older and more mature fields such as Economics, where the research agenda is better defined and open questions more broadly known. For example, in the field of economic growth, specific questions regarding why poor countries are poor and rich countries are rich are well defined and deemed important, allowing researchers to prioritize [16]. Similar, in Game AI there are specific, well-known flagship areas based on previous research [17]. Researchers in those fields, given a hypothetically perfect data set, would use that data to solve these questions first. In the field of Mobile Game Analytics, this prioritization is not explicit.

The current situation is common for new empirical research domains, but there are clear benefits to establishing research frameworks for guiding the development of new research and organizing existing knowledge. In the case of Mobile Game Analytics, where no such framework exists, building one involves testing prevalent ideas about player behavior which currently have limited empirical backing.

In this paper the current state-of-the-art of Mobile Game Analytics is described in terms of the distribution of knowledge in the domain, the general forces driving current research and interest in mobile games. The concept of *stylized facts* is introduced in the context of Mobile Game Analytics for the purpose of providing a vehicle for developing a framework, across both industry and academics, for research in the area. To be specific, such a framework should be an aggregation of current knowledge into recognizable areas, a synchronization of terminology and definitions, and a system for organizing and defining the open problems in the domain.

Unfortunately, given the fragmented state of empirical knowledge in Mobile Game Analytics, it is not currently possible to define stylized facts. To facilitate their creation, this study introduces two proto-stylized fact concepts which describe situations where less empirical validation is available: *beliefs* and *hypothetical stylized facts*. These, respectively, represent situations where none/highly limited vs. some empirical support is available. Namely, they represent lower levels of empirical validation, which are needed given the current state of available empirical evidence in Mobile Game Analytics. Introducing these concepts should provide a roadmap for structuring current knowledge and building towards a situation where stylized facts can be generated and validated.

Stylized facts originate in the Social Sciences, notably Economics [18], [19]. [18] defined stylized facts as: “*such properties, common across a wide range of instruments, markets and time periods are called stylized empirical facts.*” They are observations that have been made in so many contexts that they are widely understood to be empirical truths, and used as the basis for other theories. Stylized facts are essentially a simplified presentation of a broad, empirical finding, similar to basic theories describing the relationship between variables [20]. They are sometimes referred to as statistical regularities and can provide a basis for aggregating and communicating knowledge in a field, being a particularly popular tool in Economics [18], [19]. Collections of stylized facts form frameworks of organized knowledge in Economics and help define areas of open research. They thus bear similarity to the use of standards in the domain of Human-Computer Interaction and Computer Science, although less specifically defined [21].

The Game Analytics field, not only in mobile/F2P games but perhaps especially therein, is rife with empirical ideas and statements that are presented like stylized facts, but have yet to be rigorously validated. For example, in F2P games, it is a widely held belief that player retention correlates with monetization, i.e. as the frequency users play a game increases so does the revenue generated by that game. If this belief was based on a substantial amount of empirical research, even if the details varied slightly over different contexts (e.g. match-3 games vs. collectible card games, PC versus mobile platforms), it could be labeled as a stylized fact. However, there is currently little available empirical evidence in mobile games to convincingly define this as a stylized fact.

A framework, such as that provided by stylized facts, is needed to mature research in this domain. Towards this end we propose, based on current evidence and industry interest, a set of hypothetical stylized facts that should help focus research on open problems. Studying these problems will push research towards turning these hypothetical stylized facts into actual stylized facts and thus establish a basic organizational framework. Luckily, there is already a base amount of empirical research done in the areas that we consider ripe for the establishment of stylized facts; our study builds upon that

effort.

The organization of this paper is as follows. Section 2 provides background and a literature review of Mobile Games Analytics, Section 3 defines stylized facts, with a focus on showing how they help mature a field, while Section 4 presents a set of hypothetical stylized facts of interest to both academic and industry researchers. Finally, Sections 5 and 6 provides a vision for future research and a conclusion, respectively.

## II. BACKGROUND: THE STATE OF MOBILE GAME ANALYTICS

There are numerous reasons for the interest in mobile games research, most of which stem from the unique opportunity they present for studying behavior, including:

- The technological underpinnings of the mobile platform and its dissemination into society, coupled with the introduction and spread of games on these platforms, provides the opportunity for behavioral data collection at an unprecedented scale.
- Mobile games form a unique opportunity to inform game design and game research: devices are ubiquitous, both across geography and culture, and are evolving very rapidly with new devices being introduced in the marketplace continually.
- Mobile devices are also often continually network connected, which enables constant behavioral measurement.
- Mobile games are generally cheaper to develop when compared to consoles and, thanks to pre-installed distribution services such as the Apple App Store, mobile games are easy to find (if the player knows the name of the game - there are hundreds of thousands of mobile games available making exploratory searches difficult), install and uninstall. The games are also, in the majority of cases, F2P, implying that the barrier to entry for a user is only the time it takes to download [14]. This has yielded a situation where a user’s investment in a title is low, impacting the design of mobile games and the approach to revenue generation on these platforms (commonly referred to as monetization) [5], [6], [8].

Jointly, these aspects of mobile games provide a unique opportunity for large-scale research and, thanks to the technology powering mobile devices, detailed tracking of user behavior. In other words, games researchers have access to broader and deeper datasets concerning user behavior than ever before; and this across audiences that are, while already large and diverse, continually growing.

However, due to the lack of standardized knowledge and shared frameworks in Mobile Game Analytics, a primary challenge facing all games researchers is deciding where their energy should be spent, i.e. how to utilize this available data, or discover what data is needed to address a given question, or even what the most important open questions are [5], [6], [8], [22], [23]. Even basic best practices for using the currently available, overwhelming amount of behavioral telemetry data for games is lacking [1], [8], [13]. This is notably critical given the diversity of games and contexts in which they are played.



On the industry side, the rise of analytics has led to the hiring of data analysts, machine learning specialists and data mining experts at prodigious rates to take advantage of these new data sources. To be successful in the F2P space requires actively leveraging data to inform decision-making [5]. Today large publishers have dozens of data scientists and user researchers working for them, far outstripping the academic ranks. However, SMEs are struggling to catch up, especially small developers operating on constrained budgets [24].

Despite the surge of interest in Mobile Game Analytics, there remains a dearth of available information from the industry. This can, to a large degree, be explained by the business value of behavioral data from games, the sensitivity of personal data, and by the relative recent introduction of analytics in games. As yet, there are no associations of game analytics or other formal bodies that can help promote knowledge sharing or standardization. Confidentiality around managerially sensitive metrics such as revenue and churn/retention makes knowledge sharing difficult apart from high-level discussions, as evident in presentations at industry events, white papers, reports and blog posts released by analysts, e.g. [25], [26]. Similarly, aggregate information on player behavior across many games is only available for publishers, or via analytics companies and generally locked behind paywalls.

In summary, industry knowledge has a tendency to stay within the confines of developers or publishers rather than be disseminated broadly. However, even when efforts are made to communicate information, the lack of shared definitions, terminology and understanding are obvious. The lack of consistent terminology signifies that we are not operating from the same baseline because of the lack of underpinning stylized facts. This leads to a continual loop of companies and researchers re-inventing the same concepts, terms and solutions.

Academic Mobile Game Analytics research finds itself in a generally similar situation. This includes the level of secrecy, as the majority of academic papers contributing to the field of Game Analytics (including F2P games), do not release the datasets used. This is commonly because the research is carried out in collaboration with a company which needs to keep the raw data confidential, or because additional studies can be performed and published on the dataset. Furthermore, similar to industry, the interdisciplinary nature of Mobile Game Analytics means that publications are distributed across numerous databases, journals, publishers and indexes and thus challenging to discover. Academic research is also often locked behind publisher paywalls, which means that the knowledge generated is not readily available to the industry, especially SMEs. It is also the case that academic research in Mobile Game Analytics tends to redefine key concepts such as churn, retention, life-time value etc. in each new paper, rather than coalescing towards shared definitions.

Academic research work in Mobile Game Analytics, F2P games included, is currently fragmented, covering a wide variety of business intelligence problems, rather than aligned

around a set of open problems. This is to be expected in the explorative phase of a new domain being established, but means that even when studies overlap they tend to define key terms differently. Examples from interdisciplinary studies include behavioral profiling [1], [27], [28], player activity analysis [29], [30], social network analysis [31] churn and retention analysis [10], [12], [32]–[34], premium user identification [13], [35], automatic game content generation [36], [37], abusive content analysis [38], opponent difficulty adjustment [39] and recommender systems [40]. It is important to note that, with the recent popularity of freemium models, more and more studies are devoted to study player behavior in F2P games [10], [12], [13], [33], [35], [38], [39]. A general observation across these studies is that they introduce individual problem definitions, and report data and methods differently, indicating a lack of standards for reporting work in Mobile Game Analytics.

### III. INTRODUCING STYLIZED FACTS

As mentioned in the introduction, this study's purpose is to introduce the concept of stylized facts, as they are used in Social Science and Economics, into the context of Game Analytics. This is in order to provide a framework for structuring knowledge in the field and define open problems.

In Economics, stylized facts were introduced based on the need to create a set of common beliefs for organizing knowledge. [41] defined stylized facts as succinct encapsulations of statistical information regarding a topic: "*Stylized facts can be a useful way of organizing one's thinking about phenomena of interest, giving a broad direction to theorizing and mapping out an agenda for empirical work.*" Among the first clear uses of stylized facts was in the Economics subfield of economic growth. Explicitly writing down a list of stylized facts helped define where the field stood as well as areas for future research [42]. In particular, Kaldor [43] defined six facts regarding economic growth that any model of growth should explain. While the facts were empirical in nature, they were not perfect since, as Kaldor [43] admitted, the goal was to: "*concentrate on broad tendencies, ignoring individual detail.*" Choosing to focus on broad features allowed for a level of abstraction that pushed the field forward. Similarly, creating a set of stylized facts for the mobile video game industry has the potential to push forward video game research in a number of different ways:

- 1) **Research evaluation:** Having a framework to evaluate research creates a well-defined environment for valuing incremental contributions. In particular, research is often evaluated using a Bayesian framework where value is ascribed to how the result changes our beliefs [44]. Research that either strongly confirms a *weak prior belief* or cast doubts upon *strong prior beliefs* is valued above research that either confirms an already strong prior or weakly counteracts a weak prior. When researchers have a set of stylized facts to base their research on they can be leveraged as priors, allowing us to better understand the contribution of a piece of research. Specifically, not having a set of stylized facts creates

an environment where research tends toward exploration and technique, rather than interpretation.

To take an example, [12] provided the first formal definition of the churn problem in games and provided prediction models across five mobile game titles. As part of the contribution, the authors define a set of behavioral features and describe their influence on the process of classifying churning players, i.e. for predicting future player departure. Despite the importance of understanding player churn, particularly for practitioners, and the existence of other recent papers in this area [10], [13], [33], it is difficult for non-experts to evaluate the importance of the work because there is no baseline to evaluate it against. Furthering this example, consider the case of applying Hidden Markov Models to churn prediction by [10]. Does this research provide a new understanding of player behavior in F2P mobile games, or is the contribution a confirmation of something already known but in a new context? Trying to appraise research without defined prior stylized facts (or another system for organizing and evaluating empirical knowledge), forces us to focus on the technique used by the authors rather than the interpretation of the results of that technique. In other words, an author can present a novel method for solving a problem of interest, but without a set of stylized facts, evaluating the method's contribution can be challenging.

2) **Research focus:** Having a framework of stylized facts allows researchers - whether in academia or industry - to focus their research energy. For example, if we accept the idea that "retention in core games is lower than in more casual titles," then studies confirming, denying or defining when this fact is true become priorities for researchers (when given a dataset that permits investigating this broad fact). In much the same way that the equity premium puzzle [45] provides a prioritization mechanism for research in the Finance field, so would the establishment of stylized facts in Game Analytics.

3) **Standards and reporting:** Having stylized facts provides a consistent framework for information to provide regarding a game title when reporting research. In many other applied fields, studies attempt to present consistent, basic information about the subjects in their study. For example, empirical studies focusing on mergers and acquisition activities report the Herfindahl-Hirschman index for that industry, as it provides a numerical data point addressing the current industry concentration. While researchers in this field know that this number is not a perfect encapsulation of industry concentration, it does provide baseline information that is useful for other researchers.

In games research, the lack of stylized empirical facts has hindered the presentation of this type of foundational information about the subject of research.

The natural question arising from the above considerations is how to identify and define stylized facts in mobile games. Fortunately, Game Analytics is an applied field in an area where analytics are ubiquitous: simply putting a game up on the Apple App store or the Google Play store generates basic KPIs about a title such as downloads and revenue. Many applications also contain more advanced tracking systems [46]

and thus game developers are generally familiar with the concept of analytics and understand the importance of it, even if analytics is not deeply integrated within every company [5], [6], [8].

Stylized facts rest on a substantial body of empirical knowledge which amalgamates to high-level, crisp, facts. The current state-of-the-art in Game Analytics for mobile/F2P games is not in a place where enough empirical work has been done to support stylized facts as they occur in Economics. If we were to represent stylized facts in games currently, they would sound like facts, but would have a much weaker empirical basis than in other fields. In essence, while we cannot currently present a series of stylized facts for Game Analytics, we can provide examples of what they would look like and what they could be used for and include any limited empirical evidence that is currently available. Doing so provides a starting point or guide for researchers to either validate (or invalidate) them.

We will therefore refer to these as *hypothetical stylized facts*. Where stylized facts are normally defined bottom-up, these are defined top-down, i.e. based on current high-level ideas and perceptions in the F2P Game Analytics domain. While there maybe some available empirical evidence supporting these hypothetical stylized facts, it is clearly not enough to rigorously, generally, support them. These hypothetical stylized facts will need to be examined in the future, and refuted or confirmed for different contexts. They will change and some may gradually mature into actual stylized facts.

An example of a hypothetical stylized fact is represented in [1], who examined the belief that playtime distributions follow a power law. The authors indicated prior work across academia and industry that described or analyzed playtime distributions. They then demonstrate across over 3,000 games that playtime distribution could be modeled using a Weibull distribution. This led the authors to propose a "playtime principle" suggesting that playtime in games follows a Weibull distribution, and furthermore described the potential cause in terms of rising and falling components of human interest. Importantly, the authors acknowledge that this principle needed further validation. This is an example of researchers taking a prevalent but minimally supported belief, compiling what empirical evidence exists and adding to it, and using this as a the basis for proposing a hypothetical stylized fact. Other potential stylized facts in Game Analytics exist in behavioral profiling [1], [47] and churn prediction [10], [12], [13] as well as in network balancing in Massively Multi-Player Online Games (MMOGs) [48].

We contrast hypothetical stylized facts against *beliefs*. Beliefs are, in this context, perceptions with little or no documented empirical evidence. There are many beliefs in Mobile Game Analytics that often get presented as stylized facts without the required empirical basis. These include statements such as:

- "Hard core games have higher monetization" [49]
- "Monetization and retention move in different directions" [50]
- "Casual Users are not as engaged as other users" [51]

- “Tablet Users Monetize better than SmartPhone Users” [52]

Beliefs can give rise to hypothetical stylized facts through amalgamation and analysis of empirical knowledge, which can then, through further research, become stylized facts.

#### A. The importance of context

Whereas stylized facts in Economics are often presented as being context-independent, this is only partially correct and commonly pointed out in the literature. The idea is not that they apply everywhere but are a general trend, and that exceptions to the trend are of interest [42]. In Mobile Game Analytics, beliefs are often presented as stylized facts that are context-independent, e.g. the higher the difficulty of a game, the quicker the drop in retention. However, games are highly varied in their design and how they approach building user experience. Games can also be played alone or with others, physically or online. While research on the effect of context on gameplay and user experience remains somewhat limited [53], context would appear to play a significant role in player behavior. The expectation is that stylized facts in Mobile Game Analytics need to be accompanied by definitions for these different contexts or conditions, or even for some stylized facts to be defined directly in relation to a specific set of conditions.

### IV. HYPOTHETICAL STYLIZED FACTS: A FIRST ATTEMPT

In this section, we create a set of hypothetical stylized facts that we believe, if studied and researched, mature the field of Mobile Game Analytics. In particular, our stylized facts relate monetization, retention and engagement, which are frequently studied academically while also being of prime interest in industry. Despite their importance, however, there are opposing views on how they relate to each other [50], [54] and it is apparent that a comprehensive answer relating all three is not straightforward [55].

There is already some literature present around this topic. Most analytical models of free-to-play build on the assumption that retention comes first and determines monetization [56]. However, a look into the marketing literature reveals that the causality between retention and monetization is unlikely to be unidirectional [57]–[59]. A number of papers in this field also relate to how pricing and sunk costs (such as an initial purchase or previous engagement) affect the level and pattern of consumption [57], [60]. Paywalls, which block content from customers in much the same way that some F2P games monetize, have also been studied in the literature [61], [62] while industry sources have more directly considered the effect of blocking core gameplay through monetization practices [63]. In other words, while there is some literature discussing the relationship between retention and monetization, it has yet to be clearly defined.

Further, engagement (as often measured by sessions, rounds or time spent in game per day) and retention are generally assumed to go hand in hand. [50] While this may largely hold, there are clear instances where a game with lower retention is characterized by more engagement. As an example, consider

the iOS versions of Wooga’s Pearl’s Peril and Jelly Splash. While Jelly Splash has a more than ten percentage points higher day one retention than Pearl’s Peril, players of Pearl’s Peril play more than twice as many sessions per day as Jelly Splash players.

Given the incomplete and fragmented empirical information regarding these important topics, we propose the following analytical notation to facilitate the creation of stylized facts. Let monetization be defined as  $\mu$ , retention as  $\rho$  and engagement as  $\gamma$ . Revenue (and success) of a free-to-play game can then be written as  $\pi = f(\rho, \mu, \gamma)$ . Using this notation, we propose the following stylized facts, writing them as simple derivatives:

- $\frac{d\pi}{d\rho} > 0$ ,  $\frac{d\pi}{d\gamma} > 0$ ,  $\frac{d\pi}{d\mu} > 0$  : Revenue  $\pi$  derived from a free-to-play mobile game is increasing in retention  $\rho$ , in engagement  $\gamma$  and in monetization  $\mu$  [56], [64].
- $\frac{d\mu}{d\rho} > 0$ ,  $\frac{d\mu}{d\gamma} > 0$  : Monetization (potential)  $\mu$  is increasing in both retention  $\rho$  and engagement  $\gamma$  [56], [64].
- $\frac{d\rho}{d\mu} \geq 0$ ,  $\frac{d\gamma}{d\mu} \geq 0$  : Retention  $\rho$  and engagement  $\gamma$  are non-decreasing in changes to monetization  $\mu$  (abstracting from dubious monetization mechanisms that may adversely affect usage).
- $\frac{d\rho}{d\gamma} \geq < 0$ ,  $\frac{d\gamma}{d\rho} \geq < 0$  : Retention  $\rho$  and engagement  $\gamma$  can increase or decrease in each other or even be stable contingent on different levels of each other.

Functions and derivatives provide a concise, analytically rigorous, notion to define the associations between these important levers that can be easily simplified when presenting to lay audiences. While this is a condensed example, it provides the basis for further discussion about what kind of format stylized facts in Mobile Game Analytics should take and how they could be published to the community.

In the above sections, we have provided citations to published information where possible, but a number of the hypothetical stylized facts discussed above are known in the industry, but, because of confidentiality concerns, are more difficult to find direct information regarding. Later we will talk more about these types of informational asymmetry issues.

### V. DISCUSSION: A VISION FOR THE FUTURE

In order to push our field forward, we need to move beliefs to hypothetical facts towards stylized facts. This will require collaboration between academia and industry across a number of venues of inquiry, from improving collaborations and fostering data sharing, jointly shaping definitions and building the framework for standardization.

The previous section laid out four hypothetical stylized facts, but in doing so, it obfuscated an important issue currently surrounding Mobile Game Analytics, the lack of consistent definitions. As in other new and immature fields, one of the major hurdles to achieving a set of stylized facts is the lack of consistent definitions. Consider the case of retention, or the likelihood that user returns to a title. Depending on who you ask, you get a different definition [65]: “*The thing is that, however paramount it is for app publishers, retention has a problem. Everyone talks about it, but there seems to be no clear consensus on a common definition of what retention*

*really means nor how it is actually calculated. The truth is that there are several ways to compute this metric, all of which lead to sensibly different results. In turn, people often end up comparing apples with oranges.”*

Other practitioners have also lamented the inconsistencies surrounding the definition of retention, monetization and engagement. Given that these terms are used throughout both industry and academics, the fact that there is not an easily understood definition is a reminder of the research opportunity in this field. As researchers we can help define these measures by studying how different possible definitions correlate with each other and other business objectives. Importantly, researchers can also pursue understanding where and when different definitions break down or when they should not be used.

This type of foundational work is common in other applied fields. For example, looking at the history of gross domestic product reveals a litany of discussions regarding how different types of production should be measured [66], let alone how to interpret and use it. For video games, similar discussions regarding monetization, engagement and retention would create a common knowledge base that would push forward both academic and industry research. However, achieving these definitions require collaboration across academia and industry.

The opportunities in mobile games across industry and academia are substantial, as are the challenges. These are not issues that can be dealt with rapidly but rather require a shift in operational and strategic mindset. In particular, the primary reason for the immaturity of Mobile Game Analytics is a weakness in industry-academic partnerships.

In fact, industry usually has earlier and better access to the most recent developments and information and thus tends to be ahead of academic research. The raw capacity for analytics work in the industry massively outstrips that in the associated academic areas. Industry research, on the other hand, usually lacks the analytical structure and rigor present in more academically focused research. To increase the usefulness of academic research, the goal should be to leverage industry's experience to find facts that are in the intersection of what researchers can do and what industry finds useful.

In our experience, many of the industry-academic partnerships are closer to a “hand-off” where academics are given data, with some minor restrictions and allowed to “do research on it,” with practitioners exerting little influence past this point, or expecting much of a return. While this type of relationship can yield some interesting techniques and insight, the lack of an ongoing cooperation between academic and industry researchers has a negative effect on both groups. On the academic side, the one-off, almost transactional, nature of these relationships leaves researchers in a bind: while they would like to engage in research that would push forward academic and industry agendas, because they are not involved on the industry side they are left to their own devices, attempting to “guess” what industry would like to see. On the other hand, industry practitioners are not finding academic research that useful, as long academic time lines and the directionless nature

of the relationship ends up influencing the type of research that is being done.

Industry and the academic community need to work together more efficiently to take the understanding of mobile games to the next level. To make our proposition actionable, it is of paramount importance to point at tools that enable the joint development of stylized facts. While stylized facts are everywhere - they appear in many professional conversations, often in the form of intuitive assumptions - generating and collecting them in a structured manner is a challenge. We put forward two suggestions for action in this regard:

1) **Improved data provisioning from companies:** While much work has been done in Game Analytics, many gaming companies do not yet understand the strategic relevance of making their data accessible to researchers and allowing them to write about the insights gleaned. Similarly, academics may not see the relevance of working with industry. It should be especially noted that relating findings to basic game characteristics is important to encourage scientific debate. For instance, studies that fail to include core information regarding a game, such as genre or basic statistics regarding retention and monetization lower their contribution potential. In particular, academic researchers should make, as a condition for doing research, the publication of basic statistical information about a title (any personal information should be anonymized); and industry should support this requirement in the interest of maturing the domain as a whole. On the research evaluation side, journals and conferences should encourage authors to include this information. Studies without this basic information is of less value to both practitioners and other academic researchers.

2) **Research cooperations:** Research cooperations, where academics work with a company, are becoming more widespread. However, like other industry-academic partnerships, they sometimes suffer from a lack of communication (and particularly communication depth) to ensure an aligned and deep understanding of data and insights. In order to yield a successful research cooperation, two key features are desirable. First, academic researchers have to be sympathetic to the realities of industry. This means focusing on research efforts that maybe more short-term and less rigorous, with the understanding that the academic researcher can revisit a topic on their own when deciding to pursue publication. Secondly, industry partners need to be sympathetic to the realities of academics. Primarily, this means assisting researchers in becoming informed about the processes and data of a company. This may mean spending time with the researcher on multiple occasions, and putting them into contact with other members of a game development team, such as data analysts, product manager and producers. If an academic researcher does not have context for the game or situation, the result is likely to be less useful. In this regard, we especially want to mention the concept of “researcher in residence where a researcher joins a company and works on-site. The physical presence of the researcher is of inconceivable value as it allows for a plethora of informal touch points within an organization. This exposes

the researcher to the challenges faced by a company in a way not possible as part of a more traditional research cooperation, and - potentially more importantly - facilitates the delivery of valuable impulses derived from thorough research.

These suggestions are complimentary to the current set of activities that assist in the creation of stylized facts. On the industry side, conferences, such as the Game Developers Conference and analytics-focused talks and panels increase connectivity between like-minded industry-based analytics researchers. However, due to the confidentiality of sensitive information such as companies' revenue and churn/retention rates, this type of data is neither consistent or plentiful.

Putting together fruitful industry-academic partnerships is one first step toward creating precise and useful definitions. If academics attempt to define these key terms in isolation, the resulting terms may not conform to how industry uses them, further pushing academics and practitioners apart. For Mobile Game Analytics to mature as a field, there must be an increase in industry-academics partnerships to, first, define key terms and secondly, turn beliefs into stylized facts. Without an increase in this type of partnership, Mobile Game Analytics researchers will fail to take advantage of the opportunity presented to them.

## VI. CONCLUSION

Mobile Game Analytics has emerged rapidly within the past decade, from virtually unknown to playing a foundational role in what has become a major part of the game sector in both industry and academia. Given the formative stage of the domain, it is not surprising that current knowledge is fragmented, often not publicly accessible, and that there is a lack of standardization. However, any domain of inquiry needs to mature, and this is also the case for Mobile Game Analytics.

In this paper, the current status of Mobile Game Analytics has been described and the challenges discussed. Furthermore, a high-level vision has been put forth for moving the domain of Mobile Game Analytics to a more comprehensive, firm base, via adopting the idea and concept of *stylized facts* and blending them into analytical models.

Adopting such a framework will enable academia, industry and especially their collaborations, to contribute to a better understanding of mobile/Free-to-Play games in a more effective manner. Without a firm definition of the concepts and terms used in Mobile Game Analytics research, industry and academic practitioners, as well as non-experts, will continue to be challenged to obtain clear information on the state of the art of knowledge in the domain or even best practices.

The proposed approach can provide both framework and direction for future research efforts. To get to that point, we, as researchers, need to spend our energy working on clearly defining terms such as monetization, retention and engagement. We also need to focus on assessing, evaluating and building empirical evidence towards constructing hypothetical and then actual stylized facts; testing and pushing to understand more within the unique opportunity that mobile games provide across academia and industry.

## REFERENCES

- [1] C. Bauckhage, A. Drachen, and R. Sifa, "Clustering game behavior data," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 7, no. 3, pp. 266–278, 2015.
- [2] J. Bohannon, "Game-miners Grapple with Massive Data," *Science*, vol. 330, no. 6000, pp. 30–31, 2010.
- [3] Swrve, "Swrve Monetization Report 2014," "<http://landingpage.swrve.com/rs/swrve/images/swrve-monetization-report-0114.pdf>", 2014.
- [4] G. Yannakakis, "Game AI Revisited," in *Proc. of ACM Computing Frontiers Conference*, 2012, pp. 285–292.
- [5] W. Luton, *Free-to-Play: Making Money From Games You Give Away*. New Riders, 2013.
- [6] T. Fields and B. Cotton, *Social Game Design: Monetization Methods and Mechanics*. Morgan Kaufmann, 2011.
- [7] A. Drachen, C. Thureau, J. Togelius, G. Yannakakis, and C. Bauckhage, "Game Data Mining," in *Game Analytics: Maximizing the Value of Player Data*, M. S. El-Nasr, A. Drachen, and A. Canossa, Eds. Springer, 2013.
- [8] M. S. El-Nasr, A. Drachen, and A. Canossa, *Game analytics: Maximizing the value of player data*. Springer Science & Business Media, 2013.
- [9] Statista, "Number of available apps in the itunes app store from 2008 to 2015 (cumulative)," "<http://www.statista.com/statistics/268251/number-of-apps-in-the-itunes-app-store-since-2008/>", July 2015.
- [10] J. Runge, P. Gao, F. Garcin, and B. Faltings, "Churn Prediction for High-value Players in Casual Social Games," in *Proc. of IEEE CIG*, 2014.
- [11] R. Sifa, C. Bauckhage, and A. Drachen, "The Playtime Principle: Large-scale Cross-games Interest Modeling," in *Proc. of IEEE CIG*, 2014, pp. 365–373.
- [12] F. Hadji, R. Sifa, A. Drachen, C. Thureau, K. Kersting, and C. Bauckhage, "Predicting Player Churn in the Wild," in *Proc. of IEEE CIG*, 2014.
- [13] H. Xie, S. Devlin, D. Kudenko, and P. Cowling, "Predicting Player Disengagement and First Purchase with Event-frequency Based Data Representation," in *Proc. of CIG*, 2015.
- [14] Y. LeJacq, "Something for nothing: How the videogame industry is adapting to a 'freemium' world," *International Business Times*, September 2012.
- [15] E. Seufert, *Freemium Economics: Leveraging Analytics and User Segmentation to Drive Revenue*. Morgan Kaufmann, 2014.
- [16] P. J. Klenow and A. Rodriguez-Clare, "Economic growth: A review essay," *Journal of monetary economics*, vol. 40, no. 3, pp. 597–617, 1997.
- [17] G. N. Yannakakis and J. Togelius, "A panorama of artificial and computational intelligence in games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 7, no. 4, pp. 317–335, 2015.
- [18] R. Cont, "Empirical properties of asset returns: stylized facts and statistical issues," *Quantitative Finance*, vol. 1, no. 2, pp. 223–236, 2001. [Online]. Available: <http://dx.doi.org/10.1080/713665670>
- [19] M. Sewell, "Characterization of financial time series," *UCL Department of Computer Science, Research Note*, no. RN/11/01, 2011.
- [20] K. S. Borden and B. B. Abbott, *Research design and methods: A process approach*. McGraw-Hill, 2002.
- [21] A. Dix, T. Roselli, and E. Sutinen, "E-learning and human-computer interaction: Exploring design synergies for more effective learning experiences," *Educational Technology & Society*, vol. 9, no. 4, pp. 1–2, 2006.
- [22] B. Chong, "Game analytics 101," "<https://www.raywenderlich.com/7559/game-analytics-101>", January 2012.
- [23] A. Bilas, "You don't need to be a data scientist," "<http://www.gamesindustry.biz/articles/2015-06-22-you-dont-need-to-be-a-data-scientist>", June 2015.
- [24] A. Rajpurohit, "Interview: Pallas horwitz, blue shell games on why data science is so critical for gaming studios," "<http://www.kdnuggets.com/2014/08/interview-pallas-horwitz-data-science-gaming-studios.html>", Aug 2014.
- [25] L. Mellon, "Applying metrics driven development to MMO costs and risks," <http://maggotranch.com/>, 2009.
- [26] G. Zoeller, "Game Development Telemetry," in *Game Developers Conference*, 2011.

- [27] C. Bauckhage and R. Sifa, "k-Maxoids Clustering," in *Proc. of KDML-LWA*, 2015.
- [28] A. Drachen, R. Sifa, C. Bauckhage, and C. Thureau, "Guns, Swords and Data: Clustering of Player Behavior in Computer Games in the Wild," in *Proc. of IEEE CIG*, 2012.
- [29] G. Wallner, "Sequential Analysis of Player Behavior," in *Proc. of ACM CHI Play*, 2015.
- [30] A. Drachen, G. N. Yannakakis, A. Canossa, and J. Togelius, "Player Modeling using Self-Organization in Tomb Raider: Underworld," in *Proc of IEEE CIG*, 2009.
- [31] N. Ducheneaut, N. Yee, E. Nickell, and R. J. Moore, "Alone together?: Exploring the Social Dynamics of Massively Multiplayer Online Games," in *Proc. of ACM SIGCHI HFICS*, 2006.
- [32] E. Harpstead, T. Zimmermann, N. Nagapan, J. J. Guajardo, R. Cooper, T. Solberg, and D. Greenawalt, "What Drives People: Creating Engagement Profiles of Players from Game Log Data," in *Proc. of ACM CHI Play*, 2015.
- [33] P. Rothenbuehler, J. Runge, F. Garcin, and B. Faltings, "Hidden Markov Models for Churn Prediction," in *Proc. of SAI IntelliSys*, 2015.
- [34] R. Sifa, C. Ojeda, and C. Bauckhage, "User Churn Migration Analysis with DEDICOM," in *Proc. of ACM RecSys*, 2015.
- [35] R. Sifa, F. Hadiji, J. Runge, A. Drachen, K. Kersting, and C. Bauckhage, "Predicting Purchase Decisions in Mobile Free-to-Play Games," in *Proc. of AAAI AIIDE*, 2015.
- [36] N. Shaker and M. Abou-Zleikha, "Alone We Can Do so Little, Together We Can Do so Much: A Combinatorial Approach for Generating Game Content," in *Proc. of AAAI AIIDE*, 2014.
- [37] G. N. Yannakakis and J. Togelius, "Experience-Driven Procedural Content Generation," *IEEE Transactions on Affective Computing*, 2011.
- [38] K. Balci and A. Salah, "Automatic Classification of Player Complaints in Social Games," *IEEE Transactions on Computational Intelligence and AI in Games*, 2015.
- [39] P. I. Cowling, S. Devlin, E. J. Powley, D. Whitehouse, and J. Rollason, "Player Preference and Style in a Leading Mobile Card Game," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 7, no. 3, pp. 233–242, 2015.
- [40] R. Sifa, C. Bauckhage, and A. Drachen, "Archetypal Game Recommender Systems," in *Proc. of KDML-LWA*, 2014.
- [41] P. Geroski, "What do we know about entry?" *International Journal of Industrial Organization*, vol. 13, no. 4, pp. 421 – 440, 1995, the Post-Entry Performance of Firms. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/016771879500498X>
- [42] C. I. Jones and P. M. Romer, "The New Kaldor Facts: Ideas, Institutions, Population, and Human Capital," in *American Economic Association*, 2009.
- [43] N. Kaldor, *The Theory of Capital: Proceedings of a Conference held by the International Economic Association*. London: Palgrave Macmillan UK, 1961, ch. Capital Accumulation and Economic Growth, pp. 177–222.
- [44] D. Burgstahler, "Inference from empirical research," *Accounting Review*, pp. 203–214, 1987.
- [45] R. Mehra and E. C. Prescott, "The equity premium in retrospect," *Handbook of the Economics of Finance*, vol. 1, pp. 889–938, 2003.
- [46] C. Klotzbach, "Enter the matrix: App retention and engagement," ["http://flurrymobile.tumblr.com/post/144245637325/appmatrix"](http://flurrymobile.tumblr.com/post/144245637325/appmatrix), May 2016.
- [47] R. Sifa, A. Drachen, C. Bauckhage, C. Thureau, and A. Canossa, "Behavior evolution in tomb raider underworld," in *Proc. of IEEE CIG*, 2013.
- [48] J. Feng and D. Saha, "A Long-term Study of a Popular MMORPG," in *Proc. ACM SIGCOMM WNSG*, 2007.
- [49] "How genre affects f2p game genres: A comparison," <https://deltadna.com/blog/the-best-f2p-game-genres-kpis/>, 2015.
- [50] "Forget monetisation - f2p success is all about retention and engagement," ["http://www.pocketgamer.biz/mobile-mavens/60951/forget-hahamonetisation-its-all-about-retention-and-engagement/"](http://www.pocketgamer.biz/mobile-mavens/60951/forget-hahamonetisation-its-all-about-retention-and-engagement/), March 2015.
- [51] "Casual versus core," <http://www.gamasutra.com/>, 2000.
- [52] "'exclusive: Nexus 7 monetizes better than other android tablets, says tinyco,'" ["http://www.adweek.com/socialtimes/exclusive-nexus-7-monetizes-better-than-other-android-tablets-says-tinyco/535738"](http://www.adweek.com/socialtimes/exclusive-nexus-7-monetizes-better-than-other-android-tablets-says-tinyco/535738), 2013.
- [53] M. Suznjevi, L. Skorin-Kapov, and M. Matijasevic, "The Impact of User, System, and Context factors on Gaming QoE: a Case Study Involving MMORPGs," in *Proc. Netgames Conference*, 2013, pp. 1–6.
- [54] S. E. Needleman, "Mobile-game makers try to catch more whales who pay for free games," *Wall Street Journal*, May 2015.
- [55] "Quora: How do you model the relationship between engagement/retention and monetization?" <https://www.quora.com/How-do-you-model-the-relationship-between-engagement-retention-and-monetization>, 2013.
- [56] E. Seufert, "A comprehensive free-to-play game model: revenue, dau, virality, and retention," <http://mobiledevmemo.com/free-to-play-spreadsheet-revenue-model/>, January 2013.
- [57] E. Shafir and R. H. Thaler, "Invest now, drink later, spend never: On the mental accounting of delayed consumption," *Journal of economic psychology*, vol. 27, no. 5, pp. 694–712, 2006.
- [58] D. Soman, "Effects of payment mechanism on spending behavior: The role of rehearsal and immediacy of payments," *Journal of Consumer Research*, vol. 27, no. 4, pp. 460–474, 2001.
- [59] D. Soman and J. T. Gourville, "Transaction decoupling: How price bundling affects the decision to consume," *Journal of Marketing Research*, vol. 38, no. 1, pp. 30–44, 2001.
- [60] J. T. Gourville and D. Soman, "The potential downside of bundling: How packaging services can hurt consumption," *Cornell Hospitality Quarterly*, vol. 42, no. 3, p. 29, 2001.
- [61] L. Chiou and C. Tucker, "Paywalls and the demand for news," *Information Economics and Policy*, vol. 25, no. 2, pp. 61–69, 2013.
- [62] H. Oh, A. Animesh, and A. Pinsonneault, "Free vs. for a fee: The impact of information pricing strategy on the pattern and effectiveness of word-of-mouth via social media," in *Proc. of ICIS*, 2013.
- [63] "6 free-to-play design lessons from clash of clans to engage players and increase profits," <http://virtualeconomists.com/blogs/news/10823889-6-free-to-play-design-lessons-from-clash-of-clans-to-engage-players-and-increase-profits>, 12 2013.
- [64] S. Reyburn, "Exclusive: Apsalar analyzes the correlation between mobile game engagement and monetization by genre," *AdWeek*, Feb. 2013.
- [65] T. Sommer, "User retention: Yes, but which one?" ["http://www.applift.com/blog/user-retention.html"](http://www.applift.com/blog/user-retention.html), February 2014.
- [66] D. Coyle, *GDP: A brief but affectionate history*. Princeton University Press, 2015.