Artificial and Computational Intelligence in Games

A Follow-up to Dagstuhl Seminar 12191

Edited by
Simon M. Lucas
Michael Mateas
Mike Preuss
Pieter Spronck
Julian Togelius

Dagstuhl Follow-Ups – Vol. 6 www.dagstuhl.de/dfu

Editors

Simon M. Lucas
School of Computer Science and
Electronic Engineering
University of Essex
sml@essex.ac.uk

Michael Mateas
Center for Games and Playable Media
University of California, Santa Cruz
michaelm@cs.ucsc.edu

Mike Preuss
European Research Center for
Information Systems
University of Münster
mike.preuss@uni-muenster.de

Pieter Spronck
Tilburg Center for Cognition and Communication
Tilburg University
p.spronck@uvt.nl

Julian Togelius
Center for Computer Games Research
IT University of Copenhagen
julian@togelius.com

ACM Classification 1998
I.2.1 Applications and Expert Systems: Games

ISBN 978-3-939897-62-0

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-939897-62-0.

Publication date
November, 2013

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/DFU.Vol6.12191.i

ISBN 978-3-939897-62-0 **ISSN 1868-8977** **http://www.dagstuhl.de/dfu**

DFU – Dagstuhl Follow-Ups

The series *Dagstuhl Follow-Ups* is a publication format which offers a frame for the publication of peer-reviewed papers based on Dagstuhl Seminars. DFU volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8977

www.dagstuhl.de/dfu

Contents

Artificial and Computational Intelligence in Games. *Dagstuhl Follow-Ups*, Volume 6, ISBN 978-3-939897-62-0.
Editors: Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius
Dagstuhl Publishing
Schloss Dagstuhl – Leibniz Center for Informatics, Germany

Preface

In May 2012, around 40 world-leading experts convened in Schloss Dagstuhl in Saarland, Southern Germany, to discuss future research directions and important research challenges for artificial and computational intelligence in games. The volume you are now reading is the follow-up volume to that seminar, which collects the distilled results of the discussions that went on during those May days. As organisers of the seminar and editors of the follow-up volume, it is our sincere hope that the chapters you are about to read will prove to be useful both as references for your existing research and as starting points for new research projects. In this introductory chapter, we give some background on the research field, describe how the seminar was held, and briefly present each of the following chapters.

The research field

Research into artificial intelligence (AI) and computational intelligence (CI)[1] started in the 1950s and has been growing ever since. The main focus of the research is to provide computers with the capacity to perform tasks that are believed to require human intelligence. The field is in constant need of good benchmark problems. All benchmark problems have their drawbacks – for instance, abstract mathematical problems might not be relevant to the real world, and complex robotics problems can be time- and resource-consuming. In the last decade, computer games have been considered a strong source of benchmark problems for human intelligence. Humans have played games for all of recorded history, and since 1980s video games have been a favourite pastime of people all over the world. Games provide a plethora of tough and interesting challenges for AI and CI, including developing artificial players, computationally creative systems that construct game content, agents that adapt to players, and systems that analyse and learn about players from their in-game behaviour. As games are developed to be challenging and interesting to humans, many game-related AI problems are relevant to understanding human cognition and creativity as well. Additionally, game design and development have a number of outstanding problems that could be solved by better and more appropriate AI, so there is an opportunity to make a contribution to real-world problems.

The study of AI and CI as applied to video games is rather new. A research field devoted to these topics has only started to coalesce within the last 8 years around the AAAI Artificial Intelligence and Interactive Digital Entertainment (AIIDE) and IEEE Computational Intelligence and Games (CIG) conferences[2]. The research field is not yet defined well, and the research community has been somewhat fractured along both the symbolic/non-symbolic axis and along the European/American axis. A set of common problems and a common terminology needed to be established, and ideas needed to be cross-fertilised. The Dagstuhl seminar which this volume is an outcome of was held in order to address these challenges.

[1] The terms AI and CI are here used more or less interchangeably, even though there is a historic divide in terms of both methods studied and membership of the research communities. In general, CI methods are more biologically inspired or statistical, whereas AI methods are more symbolical or logical, but there is a great deal of overlap.

[2] AI and CI research into classic board games has a longer history, and is concentrated around the ICGA Computer Games (CG) and Advances in Computer Games (ACG) conferences.

Artificial and Computational Intelligence in Games. *Dagstuhl Follow-Ups*, Volume 6, ISBN 978-3-939897-62-0.
Editors: Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius
Dagstuhl Publishing
Schloss Dagstuhl – Leibniz Center for Informatics, Germany

The seminar and the follow-up volume

In May 2012, we gathered around 40 researchers and practitioners of AI and CI in video games at Schloss Dagstuhl to discuss the challenges in the field and the approaches to make progress on important problems. The seminar was arranged so that participants formed work groups consisting of 3–10 people each, all experts on the topic of the group. The groups discussed their topic for one or two days, and then reported the consensus of their findings to the rest of the seminar. We explicitly instructed attendants to focus on the future of the field rather than the past, and to form groups around problems rather than particular methods. With so many world-class experts on these topics gathered in a single seminar to discuss the challenges of the future, it would have been a sin to not publish the outcome of the proceedings so it would be accessible to other researchers in the area.

Soon after the seminar, we published a Dagstuhl Report containing abstracts of the discussions in all groups. We then invited group members to write longer chapters on each topic, chapters which would include both surveys of the existing literature and discussions of future challenges, and thus serve as guides to the literature as well as starting points for new research projects. Unfortunately, not all groups wrote up a full chapter-length description of their conclusions for various reasons (mostly lack of time on part of the authors). The work groups that did not write full chapters were those on believable agents and social simulations, AI architectures for games, AI for modern board games, evaluating AI in games research, AIGameResearch.org and computational narrative. However, eight of the groups did write full chapters, and those are collected in this follow-up volume.

The chapters

This volume consists of eight chapters in addition to the Introduction you are currently reading. Each chapter is the outcome of a particular discussion group that met for two days, and wrote the chapter in the months after the symposium. To assure quality, single-blind peer review was carried out by other attendees of the seminar, and the final versions of the chapters have been edited to address the reviewers' concerns. In editing this volume, we have chosen to arrange the chapters so that they start with the more generic problems and methods and proceed to more specific applications. However, the web of interdependence between work on these topics is dense, with games for mobile platforms relying on pathfinding, general game playing on procedural content generation, procedural content generation on player modelling, etc.

Search in Real-Time Video Games

Almost every AI and CI technique can be seen as search in some way: search for solutions, paths, strategies, models, proofs, actions etc. Historically, the first applications of AI techniques to games – in particular, methods for playing board games – were all about searching the tree of possible actions and counter-actions, the "game tree". It therefore seems suitable to start this volume with a chapter about search. This chapter outlines the main types of search that are used in games today, and then proceeds to discuss the main challenges that search algorithms face when applied to real-time video games as opposed to board games. Arguably the greatest challenge is that real-time video games typically have near-continuous state and action space, so that the number of states and the number of actions that could be taken from each state (the branching factor) is enormous in comparison to the one for traditional board games. Also, the number of moves required to be taken

before the game reaches an end point with a natural reward function is much larger. However, concepts as game tree and branching factor cannot easily be applied to real-time games as moves are neither strictly sequential nor necessarily alternating between players. We find that even measuring the hardness of these problems is difficult. Approaches to overcoming the challenges include clustering or partitioning states, and statistical approaches such as Monte Carlo methods.

Pathfinding in Games

Pathfinding is a particular kind of search, where the objective is to find the shortest path (according to some metric) between two points. The workgroup (and thus the chapter) on pathfinding was motivated by the central importance of pathfinding for most video games. Algorithms for pathfinding consume a considerable amount of processing power in modern games, and whereas pathfinding might be considered a "solved problem" in some areas of AI, it most certainly is not in computer games. Inferior pathfinding is a substantial problem in published commercial computer games, with their complex dynamic environments and real-time processing requirements. Recent advances in pathfinding include path computation methods based on hierarchical abstractions, informed memory-based heuristic functions, symmetry reduction and triangulation-based map representations. The chapter also outlines future research challenges, which mainly relate to the following three subjects: (1) the dynamic nature of game maps, which can change at any time with, for instance, a destructible environment; (2) the sheer size of game maps coupled with memory limitations of game consoles; and (3) collaborative pathfinding for multiple agents.

Learning and Game AI

Machine learning is a very active research field in its own right, with a large number of applications in various domains. It would seem natural for an academic researcher to think that there were ample applications for learning algorithms in computer games. However, it is rather rare to see machine learning used in any published games, and commercial game developers tend to view such methods with utmost suspicion. The chapter on Learning and Game AI goes through some of the many potential applications for machine learning in games, such as balancing games, balancing players and finding design loopholes. The chapter also discusses some of the considerable challenges that are impeding the adoption of learning algorithms by the game industry, including explaining the models induced by learning algorithms to designers and players, the problem with finding good reward functions that reflect the quality of the game, and the high computational cost of many learning algorithms.

Player Modelling

Player modelling is a specific application of machine learning to games, where the goal is to model the behaviour, preferences or experience of the player. A central question tackled in the chapter on player modelling is to what extent an accurate model of a player can be constructed based on observations of the player's behaviour in a game, potentially enriched by information from demographics, questionnaires and psychophysiological measurements. With the growing amount of networking that game players engage in, the potential to acquire data for building player models is increasing all the time. Many modern computer games now "phone home" and report detailed information to their developers' servers about their players.

This information might be used to make games more entertaining and captivating for their players, which in turn translates to revenue for their developers. Researchers in CI and AI have much to contribute here, given the plethora of methods that have been developed in academia for similar problems. In the chapter, approaches based on unsupervised learning are contrasted with "theory-based" approaches based on ideas and models from psychology. One conclusion is that while it is relatively easy to create models for populations of players which predict how that population will respond, it is quite hard to create a model for an individual player, that explains and can anticipate that player's behaviour. A potential solution is to create dynamic models, i.e., a model for a player that is constantly evaluated and dynamically adapted to observations of the player, and player responses.

Procedural Content Generation: Goals, Challenges and Actionable Steps

The chapter on procedural content generation discusses methods for automatically generating game content such as maps, levels, items, quests and game rules. This is a set of important problems in game development, both to alleviate production costs for game content and to make new kinds of games that feature infinite and perhaps adaptive content generation. In recognition of this, procedural content generation has recently become one of the most active research topics within the CI/AI and games field. The chapter proposes a vision of a system that can make a complete game world, including characters, buildings, quests, and items for a given game-engine at the press of a button. To reach this vision, a number of technical and conceptual challenges need to be overcome; the list includes items such as better understanding the process of searching for game content artifacts, and representing artistic style in a content generator. Further, the chapter lists a number of specific projects which could be undertaken right away, and would contribute to addressing the main challenges for PCG and ultimately to realising the grand vision.

General Video Game Playing

Much game AI is developed to work specifically with one game, meaning that the solutions developed might be narrow in scope and contributing little to the greater problem of general artificial intelligence. In the small field of *general game playing*, approaches to creating agents that can proficiently play a wide range of games is studied. However, in actual practice general game playing tends to focus on variations of board games. This is to a large extent due to the field's focus on the General Game Playing Competition, where AI agents are tested on unseen games which in practice all are rather simple board games. The chapter on general video game playing argues for the importance of testing agents on multiple unseen games, but that these need to be more complex and multi-faceted than the games which have hitherto been used for general game playing. In particular, video games provide plenty of challenges related to coordination, timing, and navigation, that board games do not provide. The chapter proposes that one necessary component of a general video game playing system would be a language in which it is possible to specify complete video games, so that they can be generated by a special game engine.

Towards a Video Game Description Language

This chapter continues where the chapter on general video game playing left off and puts forward a concrete suggestion for a language that can specify simple video games. The language was developed by analysing three simple 2D arcade games from the early 1980's –

Frogger, Lunar Lander and Space Invaders – and finding a vocabulary for describing their common parts. It was found that a language could be structured around individual game objects, and defining their movement logics and the effects of collisions. The chapter includes sketches of how the three aforementioned games would be implemented in the proposed language.

Artificial and Computational Intelligence for Games on Mobile Platforms

The final chapter addresses the particular challenges and opportunities arising when using AI and CI in games for mobile platforms, such as smartphones and tablets. Compared to desktop and laptop computers, these devices typically have limitations in terms of battery life, processing power and screen size. However, they also have several features that are not usually part of conventional systems, such as location awareness, personal ownership and relatively low demands on graphics. The chapter on AI and CI for games on mobile platforms argues that these features make such platforms very well suited for experimentation with new AI-based game designs, especially those based on procedural content generation, personalisation and ubiquitous gaming.

Conclusions

The 2012 gathering at Schloss Dagstuhl was deemed a great success by all participants, and it drew a large part of this strength out of the agile and very adaptive style it was held in, with several unforeseen developments in themes and results. This follow-up volume exemplifies the high level of the scientific discussions and the strong focus on scientific progress of the seminar as a whole. We are pleased to announce that a follow-up seminar will be organized at Schloss Dagstuhl in 2014. Whereas the 2012 seminar treated the topics discussed as separate research areas, the 2014 seminar will focus on the integration of the various research fields. This is meant to achieve faster developments, improve visibility and acceptance of our algorithms and approaches in industry and open up new areas of research. We believe that integration is an exciting as well as necessary step in order to further shape and consolidate the research field of artificial and computational intelligence in games.

List of Authors

Elisabeth André
University of Augsburg
Germany
andre@informatik.uni-augsburg.de

Clare Bates Congdon
University of Southern Maine
USA
congdon@usm.maine.edu

Christian Bauckhage
Fraunhofer IAIS, Sankt Augustin
Germany
christian.bauckhage@iais.fraunhofer.de

Michael Buro
University of Alberta
Canada
mburo@cs.ualberta.ca

Michal Bida
Charles University of Prague
Czech Republic
Michal.Bida@mff.cuni.cz

Adi Botea
IBM Research, Dublin
Ireland
adibotea@ie.ibm.com

Bruno Bouzy
Université Paris Descartes
France
bruno.bouzy@parisdescartes.fr

Martin V. Butz
Eberhard Karls Universität Tübingen
Germany
butz@informatik.uni-tuebingen.de

Alex J. Champandard
AiGameDev.com, Vienna
Austria
alexjc@aigamedev.com

Peter I. Cowling
University of York
United Kingdom
peter.cowling@york.ac.uk

Marc Ebner
Ernst Moritz Arndt Universität, Greifswald
Germany
marc.ebner@uni-greifswald.de

Philip Hingston
Edith Cowan University
Australia
p.hingston@ecu.edu.au

Graham Kendall
University of Nottingham
United Kingdom and Malaysia
graham.kendall@nottingham.ac.uk

Pier Luca Lanzi
Politecnico di Milano
Italy
lanzi@elet.polimi.it

John Levine
University of Strathclyde
United Kingdom
john.levine@strath.ac.uk

Daniele Loiacono
Politecnico di Milano
Italy
daniele.loiacono@polimi.it

Michael Mateas
University of California, Santa Cruz
USA
michaelm@cs.ucsc.edu

Risto Miikkulainen
University of Texas, Austin
USA
risto@cs.utexas.edu

Héctor Muñoz-Avila
Lehigh University
USA
munoz@eecs.lehigh.edu

Dana Nau
University of Maryland
USA
nau@cs.umd.edu

Artificial and Computational Intelligence in Games. *Dagstuhl Follow-Ups*, Volume 6, ISBN 978-3-939897-62-0.
Editors: Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius
Dagstuhl Publishing
Schloss Dagstuhl – Leibniz Center for Informatics, Germany

Ana Paiva
INESC-ID, Lisboa
Portugal
ana.paive@inesc-id.pt

Mike Preuss
Technical University of Dortmund
Germany
mike.preuss@cs.tu-dortmund.de

Tom Schaul
New York University
USA
schaul@cims.nyu.edu

Moshe Sipper
Ben-Gurion University
Israel
sipper@cs.bgu.ac.il

Pieter Spronck
Tilburg University
The Netherlands
p.spronck@uvt.nl

Kenneth O. Stanley
University of Central Florida
USA
kstanley@eecs.ucf.edu

Tommy Thompson
University of Derby
United Kingdom
t.thompson@derby.ac.uk

Julian Togelius
IT University of Copenhagen
Denmark
julian@togelius.com

Georgios N. Yannakakis
University of Malta, Msida
Malta
georgios.yannakakis@um.edu.mt

Search in Real-Time Video Games

Peter I. Cowling[1], Michael Buro[2], Michal Bida[3], Adi Botea[4], Bruno Bouzy[5], Martin V. Butz[6], Philip Hingston[7], Héctor Muñoz-Avila[8], Dana Nau[9], and Moshe Sipper[10]

1 **University of York, UK**
peter.cowling@york.ac.uk
2 **University of Alberta, Canada**
mburo@cs.ualberta.ca
3 **Charles University in Prague, Czech Republic**
Michal.Bida@mff.cuni.cz
4 **IBM Research, Dublin, Ireland**
adibotea@ie.ibm.com
5 **Université Paris Descartes, France**
bruno.bouzy@parisdescartes.fr
6 **Eberhard Karls Universität Tübingen, Germany**
butz@informatik.uni-tuebingen.de
7 **Edith Cowan University, Australia**
p.hingston@ecu.edu.au
8 **Lehigh University, USA**
munoz@eecs.lehigh.edu
9 **University of Maryland, USA**
nau@cs.umd.edu
10 **Ben-Gurion University, Israel**
sipper@cs.bgu.ac.il

Abstract

This chapter arises from the discussions of an experienced international group of researchers interested in the potential for creative application of algorithms for searching finite discrete graphs, which have been highly successful in a wide range of application areas, to address a broad range of problems arising in video games. The chapter first summarises the state of the art in search algorithms for games. It then considers the challenges in implementing these algorithms in video games (particularly real time strategy and first-person games) and ways of creating searchable discrete representations of video game decisions (for example as state-action graphs). Finally the chapter looks forward to promising techniques which might bring some of the success achieved in games such as Go and Chess, to real-time video games. For simplicity, we will consider primarily the objective of maximising playing strength, and consider games where this is a challenging task, which results in interesting gameplay.

1998 ACM Subject Classification I.2.8 Problem Solving, Control Methods, and Search

Keywords and phrases search algorithms, real-time video games, Monte Carlo tree search, minimax search, game theory

Digital Object Identifier 10.4230/DFU.Vol6.12191.1

Artificial and Computational Intelligence in Games. *Dagstuhl Follow-Ups*, Volume 6, ISBN 978-3-939897-62-0.
Editors: Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius; pp. 1–19

Dagstuhl Publishing
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany

1 Introduction

Search algorithms have achieved massive success across a very wide range of domains, and particular success in board and card games such as Go, chess, checkers, bridge and poker. In each of these games there is a reasonably well-defined state-action graph (possibly with information sets in the games of imperfect information games such as bridge and poker). The success and generality of search for producing apparently strategic and human-competitive behaviours points to the possibility that search might be a powerful tool in finding strategies in video game AI. This remains true in multiplayer online strategy games where AI players need to consistently make effective decisions to provide player fun, for example in the role of supporting non-player character. Search is already very well embedded in the AI of most video games, with A* pathfinding present in most games, and ideas such as procedural content generation [78] gaining traction.

However, video games provide a new level of challenge when it comes to thinking about the sort of strategic behaviours where search has worked so well in board and card games. Principally this challenge is that the complexity of the naïvely defined state-action graph has both a branching factor and a depth that is orders of magnitude greater than that for even the most complex board games (e.g. Go), since we must make sometimes complex decisions (such as choice of animation and path) for a large number of agents at a rate of anything up to 60 times per second. Currently these problems are overcome using painstakingly hand-designed rule-based systems which may result in rich gameplay, but which scale rather poorly with game complexity and are inflexible when dealing with situations not foreseen by the designer.

In this article, we point towards the possibility that power and ubiquity of search algorithms for card and board games (and a massive number of other applications) might be used to search for strategic behaviours in video games, if only we can find sensible, general ways of abstracting the complexity of states and actions, for example by aggregation and hierarchical ideas. In some ways, we are drawing a parallel with the early work on chess, where it was felt that capturing expert knowledge via rules was likely to be the most effective high-level method. While capturing expert rules is currently the best way to build decision AI in games, we can see a bright future where search may become a powerful tool of choice for video game strategy.

In order to provide a coherent treatment of this wide area, we have focussed on those video games which have the most in common strategically with board and card games, particularly Real Time Strategy (RTS) and to a lesser extent First Person games. For these games a challenging, strong AI which assumes rational play from all players is a goal which is beyond the grasp of current research (although closer than for other video games where speech and emotion are needed), but which would likely be of interest to the games industry while providing a measurable outcome (playing strength) to facilitate research developments.

The paper is structured as follows: in section 2 we consider the current state of the art in relevant research areas, in section 3 we point to some of the research challenges posed by video games, and in section 4 we discuss promising approaches to tackling them. We conclude in section 5. The paper arose from the extensive, wide ranging and frank discussions of a group of thought leaders at the Artificial and Computational Intelligence in Games summit at Schloss Dagstuhl, Germany, in May 2012. We hope it may provide some inspiration and interesting directions for future research.

2 State of the Art

2.1 Search

Alpha Beta Minimax Search

Alpha-beta is a very famous tree search algorithm in computer games. Its history is strongly linked with the successes obtained in Computer Chess between 1950, with the Shannon's original paper [65], and 1997 when Deep Blue surpassed Gary Kasparov, the human world champion [1], [17], or indeed when Shaeffer and co-workers showed that with perfect play the game of Checkers is a draw [63], which is one of the largest computational problems ever solved. Alpha-beta [35] is an enhancement over Minimax which is a fixed-depth tree search algorithm [81]. At fixed depth d, Minimax evaluates nonterminal game positions with a domain-dependent evaluation function. It backs up the minimax values with either a min rule at odd depths or a max rule at even depths, and obtains a minimax value at the root. Minimax explores b^d nodes, where b is the search tree branching factor.

At each node, alpha-beta uses two values, alpha and beta. Without entering into the details, alpha is the current best value found until now, and beta is the maximal value that cannot be surpassed. During the exploration of the branches below a given node, when the value returned by a branch is superior to the beta value of the node, the algorithm safely cuts the other branches in the search tree, and stops the exploration below the node. Alpha-beta keeps minimax optimality [35]. Its efficiency depends mainly on move ordering. Alpha-beta explores at least approximately $2b^{d/2}$ nodes to find the minimax value, and consumes a memory space linear in d. Furthermore, in practice, the efficiency of alpha-beta depends on various enhancements. Transposition tables with Zobrist hashing [86] enables the tree search to reuse results when encountering a node already searched. Iterative deepening [68] iteratively searches at increasing depth enabling the program to approach an any time behaviour [39]. Minimal window search such as MTD(f) [56] uses the fact that many cuts are performed when the alpha-beta window is narrow. Principal variation search assumes that move ordering is right and that the moves of the principal variation can be searched with a minimal window [55]. The history heuristic gathers the results of moves obtained in previous searches and re-use them for dynamical move ordering [62].

Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search (MCTS) has revolutionised Computer Go since it was introduced by Coulom [20], Chaslot et al. [19] and Kocsis and Szepesvári [36]. It combines game tree search with Monte Carlo sampling. As for minimax tree search, MCTS builds and searches a partial game tree in order to select high quality moves. However, rather than relying on a domain-dependent evaluation function, Monte Carlo simulations (rollouts) are performed starting from each nonterminal game position to the end of the game, and statistics summarising the outcomes are used to estimate the strength of each position. Another key part of its success is the use of UCT (Upper Confidence Bounds for Trees) to manage the exploration/exploitation trade-off in the search, often resulting in highly asymmetric partial trees, and a large increase in move quality for a given computational cost. The resulting algorithm is an any-time method, where move quality increases with the available time, and which, in its pure form, does not require any a priori domain knowledge. Researchers have been exploring its use in many games (with notable success in general game playing [6]), as well as in other domains.

The basic algorithm can be enhanced in various ways, to different effect in different applications. One common enhancement is the use of transposition tables, and a related

idea: Rapid Action Value Estimation (RAVE) [24], which uses the all-moves-as-first (AMAF) heuristic [13]. Another common enhancement is to use domain knowledge to bias rollouts [9, 25]. This must be used with care, as stronger moves in rollouts do not always lead to better quality move selections. A recent survey [12] provides an extremely thorough summary of the current state of the art in MCTS.

A* Search

A* [29] is one of the most popular search algorithms not only in games (e.g., pathfinding), but also in other single-agent search problems, such as AI planning. It explores a search graph in a best-first manner, growing an exploration area from the root node until a solution is found. Every exploration step expands a node by generating its successors. A* is guided using a heuristic function $h(n)$, which estimates the cost to go from a current node n to a (closest) goal node. Nodes scheduled for expansion, that are kept in a so-called *open list*, are ordered according to a function $f(n) = g(n) + h(n)$, where $g(n)$ is the cost from the root node to the current node n. Nodes with a smaller f value are seen as more promising, being considered more likely to belong to a solution with an optimal cost. When h is *admissible* (i.e., $h(n)$ never overestimates the true cost between n and the nearest goal, $h^*(n)$), A* returns optimal solutions.

A* remembers all visited nodes, to be able to prune duplicate states and to reconstruct a solution path to the goal node. Both the memory and the running time can be one important bottlenecks for A*, especially in large search problems. Iterative Deepening A* (IDA*) [38] reduces the memory needs down to an amount linear in the depth of the exploration path, at the price of repeatedly expanding parts of the state space by iterative deepening. Weighted A* [58] can provide a significant speed-up at the price of a bounded solution sub-optimality. There are many other search methods that build on A*, including methods for hierarchical pathfinding, and methods for optimal search on multiple CPU cores.

Multi-Player Search

Less research has been done on game-tree search in multi-player games than in two-player zero-sum games, partly because results are harder to obtain. The original tree-search algorithm for multi-player games, Max-n [47], is based on the game-theoretic backward induction algorithm [54] for computing a *subgame-perfect equilibrium* (in which each player maximizes its return under the assumption of game-theoretic rationality). To this, Max-n adds game-tree pruning and bounded-depth search. Prob-max-n is an extension of Max-n with probabilities [75]. The results on pruning in Max-n are very weak: shallow pruning occurs at depth one, but deeper pruning does not occur except in the last branch, which is not effective when the branching factor is high [40]. Greater pruning can be achieved by replacing the assumption of game-theoretic rationality with a "paranoid" assumption that all of the other players are trying to minimize one's own return. This assumption reduces the multi-player game to a two-player game, on which one can use a two-player game-tree search algorithm [74] to compute a maximin strategy for the multi-player game. In terms of game-playing performance, neither the paranoid algorithm nor the Max-n assumption of game-theoretic rationality have been especially successful in producing strong AI players, and more sophisticated ways are needed to model human interactions in multiplayer games [83]. As in two-player games in which MCTS surpassed alpha-beta, MCTS also surpassed Max-n in multi-player games [73] with enhancements or not [53] and has become the reference algorithm.

2.2 Abstraction

Hierarchical Task Networks (HTNs) and Behaviour Trees

HTN planning is a technique for generating sequences of actions to perform *tasks* (complex activities). In a game environment, these might be activities to be performed by an computer-controlled agent. For each kind of task there are *methods* that provide alternative ways to decompose the task into *subtasks* (simpler tasks). Usually each method contains preconditions or constraints that restrict when it can be used. Planning proceeds by decomposing tasks recursively into simpler and simpler subtasks, until actions are reached that the agent can perform directly. Any time a task t is produced for which none of the available methods is applicable, the planner backtracks to look for other ways to decompose the tasks above t.

Some HTN planners are custom-built for particular application domains. Others (e.g., SIPE-2 [82], O-PLAN [22, 77], and SHOP2 [51]) are *domain-configurable*, i.e., the planning engine is independent of any particular application domain, but the HTN methods in the planner's input are specific to the planning domain at hand. Most HTN planners are built to do *offline* planning, i.e., to generate the entire plan before beginning to execute it. But in video game environments where there are strict time constraints and where the outcomes of the planned actions may depend on many unpredictable factors (e.g., other agents), planning is often done *online* (concurrently with plan execution) (e.g., [44]). To create believable sequences of actions for virtual agents in Killzone 2, an HTN planner generates single-agent plans as if the world were static, and replans continually as the world changes [80, 18].

In the context of game programming, a *behavior tree* is a tree structure that interleaves calls to code executing some gaming behavior (e.g., harvest gold in an RTS game) and the conditions under which each of these code calls should be made. In this sense, they are closer to hierarchical FSMs [32] than to HTN planning. At each node there are lists of pairs $(cond, child)$ where *cond* is a condition to be evaluated in the current state of the game and *child* is either a code call or a pointer to another node. In current practice, behavior trees generally are constructed by hand. In principle, HTN planning could be used to generate behavior trees; but most existing HTN planners incorporate an assumption of a single-agent environment with actions that have deterministic outcomes (in which case the tree is simply a single path). A few HTN planners can reason explicitly about environments that are multi-agent and/or nondeterministic [41, 43, 42], including a well-known application of HTN planning to the game of bridge [52, 70]. Furthermore, the online HTN planning used in some video games can be viewed an on-the-fly generation of a path through a behavior tree.

Temporal Abstraction

The abstraction of time in real-time video games provides unique challenges for the use of approaches which search a directed graph, where the nodes represent states and the arcs represent transitions between states. Note that here we interpret "state" loosely – the states themselves will usually be rather abstract representations of the state of the game, and the transitions will usually only represent a small subset of possible transitions. If we introduce continuous time, then a naive state transition graph has infinite depth, since we may make a transition at any time. There are two primary methods for overcoming this problem: time-slicing and event-based approaches. When using time-slicing we divide time into even segments, and use algorithms which are guaranteed to respond within the given time, or consider applications where a non-response is not problematic. There are many examples in pathfinding (e.g. Björnsson [5]). Another time-slicing approach is given in Powley et al. [60] where macro-actions are used to plan large-scale actions which are then executed frame by

frame, where each transition corresponds to a macro-action executed over several frames. Event-based approaches consider an abstraction of the tree where branching is only possible following some significant event. Zagal and Mateas [85] has further discussion of abstract time in video games.

Level of Detail AI

One of the main challenges of creating high-fidelity game AI is the many levels of behaviour that are needed: low level actions, agent operations, tactics, unit coordination, high-level strategy etc. For example, real-time strategy games require multiple levels of detail. The AI needs to control tasks at different levels including creating and maintaining an economy, planning a high-level strategy, executing the strategy, dealing with contingencies, controlling hundreds of units, among many others. To deal with this problem, so-called managers are often used to take care of different gaming tasks [64]. Managers are programs specialized in controlling one of the tasks in the real-time strategy game such as the economy. In games such as role-playing games and first-person shooters, a lot of CPU and memory resources are spent in the area where the player-controlled character is located whereas little or no CPU time is spent in other areas [10]. This is often referred to as level-of-detail AI. One possible approach is to use AI techniques such as AI planning to generate high-level strategies while using traditional programming techniques such as FSMs to deal with low-level control.

Hierarchical Pathfinding

Pathfinding remains one of the most important search applications in games. Computation speed is crucial in pathfinding, as paths have to be computed in real-time, sometimes with scarce CPU and memory resources. Abstraction is a successful approach to speeding up more traditional methods, such as running A* on a flat, low-level graph representation of a map.

Hierarchical Pathfinding A* (HPA*) [7] decomposes a map into rectangular blocks called clusters. In an abstract path, a move traverses an entire cluster at a time. Abstract moves are refined into low-level moves on demand, with a search restricted to one cluster. Hierarchical Annotated A* (HAA*) [28] extends the idea to units with variable sizes and terrain traversal capabilities. Partial Refinement A* (PRA*) [76] builds a multi-level hierarchical graph by abstracting a clique at level k into a single node at level $k + 1$. After computing an abstract path at a given level, PRA* refines it level by level. A refinement search is restricted to a narrow corridor at the level at hand. Block A* [84] uses a database with all possible obstacle configurations in blocks of a fixed size. Each entry caches optimal traversal distances between any two points on the block boundary. Block A* processes an entire block at a time, instead of exploring the map node by node. Besides abstractions based on gridmaps, triangulation is another successful approach to building a search graph on a map [23].

Dynamic Scripting

Scripts, programs that use game-specific commands to control the game AI, are frequently used because it gives the game flexibility by decoupling the game engine from the program that controls the AI [4]. This allows gamers to modify the NPC's behaviour [59]. Carefully crafted scripts provide the potential for a compelling gaming experience. The flip side of this potential is that scripts provide little flexibility which reduces replayability. To deal with this issue, researchers have proposed dynamic scripting, which uses machine learning techniques to generate new scripts or modify existing ones [71]. Using a feedback mechanism such as

the one used in reinforcement learning, scripts are modified towards improving some signal from the environment (e.g., score of the game).

Learning Action Hierarchies

Various research directions have acknowledged that an hierarchical organization of actions can be highly beneficial for optimizing search, for more effective planning, and even for goal-directed behavioral control in robotics [8]. Psychological research points out that the brain is highly modularly organized, partitioning tasks and representations hierarchically and selectively combining representation on different levels for realizing effective behavioral control [16, 30, 57]. While thus the importance of hierarchies is without question, how such hierarchical representations may be learned robustly and generally is still a hard challenge although some research exists studying this problem [31].

"Black Box" Strategy Selection

An intuitive approach to applying adversarial search to video games is to start off with a collection of action scripts that are capable of playing entire games and then – while playing – select the one executed next by a look-ahead procedure. In [61], for instance, this idea has been applied to base-defense scenarios in which two players could choose among scripts ranging from concentrating forces to spreading out and attacking bases simultaneously. To choose the script to execute in the next time frame, the RTSplan algorithm simulates games for each script pair faster than real-time, fill a payoff matrix with the results, and then solves the simultaneous-move games- in which actions now refer to selecting scripts – using linear programming. Actions are then sampled according to the resulting action probabilities. Equipped with an efficient fast-forwarding script simulator and an opponent modelling module that monitors opponent actions to maintain a set of likely scripts the opponent is currently executing, RTSplan was able to defeat any individual script in its script collection.

3 Challenges

3.1 Search

- **Massive branching factor / depth.** MCTS has been successfully applied to trees with large branching factor and depth in games such as Go or Amazons. However, video games branching factor and depth in generally several order of magnitudes greater. Furthermore, the action space and the time can be continuous leading to much more complexity. MCTS has been studied in continuous action space leading to Hierarchical Optimistic Optimization applied to Trees (HOOT) [48]. HOO [14] is an extension of UCB addressing the case of a continuous set of actions. However, HOOT is a very general approach which is unlikely to work in complex video games.
  Often, the massive branching factor is caused by the high number of characters acting in the game. One character may have a continuous set of actions. But even with a reduced set of actions, i.e. north, west, south, east, wait, the number of characters yields an action space complexity that is finite but that is huge and not tractable with general tools such as HOOT. The problem of the huge action set in video games cannot be dealt with MCTS approach alone. A partial solution could be grouping sibling moves, but it will probably not be sufficient. Some abstraction scheme must be found to divide the

number of actions drastically. There is some promise for macro-action techniques such as [60] but to date these have been considered only in rather simple video games.

The length of simulated video games is the second obstacle. With many characters acting randomly, a mechanism must lead the game to its end. Assuming the game ends, it should do it quickly enough, and the game tree developed should go deeply enough to bring about relevant decisions at the top level. A first solution to encompass the huge depth or the time continuity, is to deal with abstract sequences grouping consecutive moves. A sequence of moves would correspond to a certain level of task with a domain-dependent meaning, e.g. a fight between two armies, or an army moving from a starting area to another. Tree search should consider these tasks as actions in the tree.

- **Automatically finding abstractions.** Learning abstractions has been a recurrent topic in the literature. Works include learning abstraction plans and hierarchical task networks from a collection of plan traces. Abstracted plans represent high-level steps that encompass several concrete steps from the input trace. In a similar vein, learning HTNs enables the automated acquisition of task hierarchies. These hierarchies can capture strategic knowledge to solve problems in the target domain.

  Despite some successes, there are a number of challenges that remain on this topic that are of particular interest in the context of games: existing work usually assumes a symbolic representation based on first-order logic. But RTS games frequently require reasoning with resources, hence the capability to abstract numerical information is needed. In addition, algorithms for automated abstraction need to incorporate spatial analysis whereby suitable abstraction from spatial elements is elicited. For example, if the game-play map includes choke points it will be natural to group activities by regions as separated by those choke points. Also, abstraction algorithms must deal with time considerations because well-laid out game playing strategies frequently consider timing issues. Last but not least, such abstractions need to be made dependent upon the actual behavioural capabilities available in the game. Choke points may thus not necessarily be spatially determined in an Euclidean sense, but may rather be behavior-dependent. For example, a flying agent does not care about bridges, but land units do. Thus, a significant challenge seems to make abstraction dependent on the capabilities of the agent considered. Learning such abstractions automatically clearly remains an open challenge at this point.
- **Capturing manually-designed abstraction.** Related to the previous point, representation mechanisms are needed to capture manually-created abstractions. Such representation mechanisms should enable the representation of game-playing strategies at different levels of granularity and incorporate knowledge about numerical information, spatial information and time. A challenge here is to avoid creating a cumbersome representation that is either very difficult to understand or for which adequate reasoning mechanisms are difficult to create.
- **1-player vs 2-player vs multiplayer.** One of the biggest reasons why game-tree search has worked well in two-player zero-sum games is that the game-theoretic assumption of a "rational agent" is a relatively good model of how human experts play such games, hence algorithms such as minimax game-tree search can produce reasonably accurate predictions of future play. In multiplayer games, the "rational agent" model is arguably less accurate. Each player's notion of what states are preferable may depend not only on his/her game score but on a variety of social and psychological factors: cameraderie with friends, rivalries with old enemies, loyalties to a team, the impetus to "gang up on the winner," and so forth. Consequently, the players' true utility values are likely to be

nonzero-sum, and evaluation functions based solely on the game score will not produce correct approximations of those utilities. Consequently, an important challenge is how to build and maintain accurate models of players' social preferences. Some preliminary work has been done on this topic [83], but much more remains to be done.

- **Hidden information / uncertainty.** Asymmetric access to information among players gives rise to rich gameplay possibilities such as bluffing, hiding, feinting, surprise moves, information gathering / hiding, etc. In many (indeed most) card and board games asymmetric hidden information is central to interesting gameplay. Information asymmetry is equally important for many video games, including the real-time strategy and first-person games which are the primary focus of this article. When played between human players, much of the gameplay interest arises through gathering / hiding information, or exploiting gaps in opponents' knowledge. Creating AI which deals with hidden information is more challenging than for perfect information, with the result that many AI players effectively "cheat" by having access to information which should be hidden. In some cases this is rather blatant, with computer players making anticipatory moves which would be impossible for a human player without perfect information, in a way which feels unsatisfactory to human opponents. In other cases, there is a simulation of limited access to information. Ideally computer characters should have access to similar levels of sensory information as their human counterparts. The problems of hidden information in discrete domains is well-studied in game theory (see e.g. Myerson [50]), where the hidden information is captured very neatly by the information set idea. Here we have a state-action graph as usual, but each player is generally not aware of the precise state of the game, but the player's partial information allows the player to know which subset of possible states he is in (which is an information set from his point of view). Searching trees of information sets causes a new combinatorial explosion to be handled. This is a challenging new area for research, with some promise shown in complex discrete-time domains by the Information Set Monte Carlo Tree Search approach of Cowling et al. [21].
- **Simulating the video game world forward in time.** Rapid forward models could provide a glimpse of the world's future via simulation. This would allow researchers to use techniques based on state space search (e.g. Monte Carlo Tree Search, HTN, classical planning) more effectively either for offline or online AIs. However, modern games often feature complex 3D worlds with rich physics and hence limited capability to speed-up a simulation (there are hardware limitations of CPUs and GPUs). Moreover, games are real-time environments that can feature non-deterministic mechanisms, including actions of opponents, which might not be always reliably simulated. Opponent modelling [79] and level-of-detail AI [11] approach may provide a partial solution to these issues. However, these features are not always present in current games, and the design and development of forward models provides an interesting research challenge.
- **Explainability.** Explanation is an important component in computer gaming environments for two reasons. First, it can help developers understand the reason behind decisions made by the AI, which is very important particularly during debugging. Second, it can be useful to introduce game elements to new players. This is particularly crucial in the context of complex strategic games such as the civilization game series.

  Generating such explanations must deal with two main challenges. First, explanations must be meaningful. That is, the explanation must be understandable by the player and/or developer and the context of the explanation must also be understood. Possible snapshots from the game GUI could help to produce such explanations, highlighting the explanatory context. Second, explanations must be timely in the sense that they must be

rapidly generated and be shown at the right time. The latter point is crucial particularly if explanations occur during game play. They should not increase the cognitive load of the player, since this might result in the player losing interest in the game.

It seems important to distinguish between preconditions and consequences when automatically generating explanations. Preconditions are the subset of relevant aspects of the game that must be in a certain state for the offered explanation (of what the bot has done) to hold. Consequences are those the bot expected to occur when it executes a particular behavior in a the specified context. In order to generate meaningful explanations, preconditions and consequences must thus be automatically identified and presented in an accessible format to ensure the generation of meaningful explanations.

- **Robustness.** Video games must continue to work whatever the state of the game world and action of the players. hence any algorithm to which uses search to enhance strategy or other aspects of gameplay must always yield acceptable results (as well as usually yielding better results than current approaches). Explainability (discussed in the previous paragraph) allows for better debugging and greatly improves the chances for robustness. Robustness may be achieved by techniques such as formal proofs of correctness and algorithm complexity, or by having simple methods working alongside more complex search methods so that acceptable results are always available.
- **Non-smooth trees.** Many games (chess is a prime example) have trees which are non-smooth in the sense that sibling nodes at significant depth in the tree (or leaf nodes) often have different game theoretic values. This behaviour makes tree search difficult, and is arguably a reason why MCTS, while effective at chess AI, cannot compete with minimax/alphabeta. It seems likely and possible that game trees for some video games will have this pathology. While this makes heuristic search approaches that do not explore all siblings perform unreliably, one possible source of comfort here is that for complex game trees made up of highly aggregated states and actions, smoothness is quite close to the property that makes such a game playable (since otherwise a game becomes too erratic to have coherent strategies for a human player). While we raise this as a challenge and a consideration, successful resolution of some of the other challenges in this section may give rise to relatively smooth trees.
- **Red teaming, Real-World Problems.** Red Teaming is a concept that originated in military planning, in which a "Red Team" is charged with putting itself in the role of the enemy, in order to test the plans of the friendly force – the "Blue Team". The term Computational Red Teaming (CRT) has been coined to describe the use of computational tools and models to assist with this planning process. A technology review on CRT was recently carried out for the Australian Department of Defence [26]. One application domain for CRT is tactical battle planning, in which battles are simulated to test the effectiveness of candidate strategies for each side, and a search is overlaid to search for strong Red and/or Blue strategies. The task has a great deal in common with the search for tactics and strategies in Real Time Strategy games, and shares many of the same challenges: the strategy space is huge, involving coordinated movements and actions of many agents; the domain is continuous, both spatially and temporally; the "fog of war" ensures that much information is hidden or uncertain. It may be that approaches that have been developed to address these issues in CRT will also be helpful in RTS and similar games, and vice-versa.
- **Memory / CPU constraints.** CPU and Memory consumption and management represents a key issue in respect to computer game speed and scalability, even against a background of increasing parallel processing capability. State of the art game development

is primarily focused on the quality of the graphical presentation of the game environment. As a result, game engines tend to consume and constrain the computational power available for AI computations [15]. Therefore, an AI subsystem responsible for hundreds of active agents has to scale well, due to the resource demand in question. Even proven generic techniques like A* often need to be tweaked in respect to the game engine to be used effectively in real-time [27]. This makes the use of computationally intensive techniques, like classical planning or simulating decision outcomes, very problematic.

- **General Purpose Methods: Transfer learning.** Many of the above techniques have the aim of being generally applicable in whole sets or classes of game environments. Once a particular behavioral strategy of an agent has been established in one environment, the same strategy might also be useful in others. One approach to tackle such tasks is to abstract from the concrete scenario, producing a more general scheme. If such abstractions are available, then transfer learning will be possible. However, clearly the hard challenge, which does not seem to be answerable in the general sense, is how to abstract. Abstractions in time for transferring timing techniques may be as valuable as abstractions in space, such as exploration patterns, or abstraction on the representational format, for example, generalizing from one object to another. General definitions for useful abstractions and formalizations for the utility of general purpose methods are missing at the moment. Even more so, the challenge of producing a successful implementation of a general purpose AI, which, for example, may benefit from playing one RTS game when then being tested in another, related RTS game is still wide open.

4 Promising Research Directions and Techniques

4.1 Abstract Strategy Trees

The RTSplan "black box" strategy selection algorithm described earlier requires domain knowledge implemented in form of action scripts. Moreover, its look-ahead is limited in the sense that its simulations assume that players stick to scripts chosen in the root position. An adhoc solution to this problem is to add decision points in each simulation, e.g.. driven by game events such as combat encounters, and apply RTSplan recursively. This, however, can slow down the search considerably. Another serious problem is that switching game-playing scripts in the course of a game simulation episode may be inadequate to improve local behaviours. As an example, consider two pairs of squads fighting in different areas of an RTS game map, requiring different combat strategies. Switching the global game playing script may change both local strategies, and we therefore will not be able to optimize them independently. To address this problem, we may want to consider a search hierarchy in which game units optimize their actions *independent* from peers at the same level of the command hierarchy. As an example, consider a commander in charge of two squads. The local tasks given to them are decided by the commander who needs to ensure that the objectives can be accomplished without outside interference. If this is the case, the squads are independent of each other and their actions can be optimized locally. This hierarchical organization can be mapped to a recursive search procedure that on different command levels considers multiple action sequences for friendly and enemy units and groups of units, all subject to spatial and temporal constraints. Because the independence of sibling groups speeds up the search considerably, we speculate that this kind of search approach could be made computationally efficient.

As a starting point we propose to devise a 2-level tactical search for mid-sized RTS combat scenarios with dozens of units that are positioned semi-randomly on a small map

region. The objective is to destroy all enemy units. The low-level search is concerned with small-scale combat in which effective targeting order and moving into weapon range is the major concern when executing the top-level commands given to it. The high-level search needs to consider unit group partitions and group actions such as movement and attacking other groups. As there are quite a few high-level moves to consider, it may be worthwhile investigating unit clustering algorithms and promising target locations for groups. Now the challenge is to intertwine the search at both levels and to speed it up so that it can be executed in real-time to generate game actions for all units. Once two-level search works, it will be conceptually easy to extend it to more than two levels and – hopefully – provide interesting gameplay which challenges human supremacy in multi-level adversarial planning.

4.2 Monte Carlo Tree Search

Video games have a massive branching factor and a very high depth due to a large number of game agents, each with many actions, and the requirement to make a decision each frame. Since abstraction is a promising technique for video games that replaces the raw states by abstract states, the concrete actions by abstract actions or sequences of actions, the challenge is to use these abstractions in a tree search algorithm. This will give an anticipation skill to the artificial player. Instead of developing a tree whose nodes are raw states and arcs are actions, MCTS might be adapted such that nodes would correspond to abstract states and arcs to a group of actions or a sequence of actions, or even to a sequence of groups of actions. The strength of MCTS is to develop trees whose nodes contain simple statistics such as the number of visits and the number of successes. Therefore nothing forbids MCTS to build these statistics on abstract entities. To this extent MCTS combined with abstractions is a very promising technique for designing artificial agents playing video games.

4.3 Game Theoretic Approaches

Designing artificial players using abstraction and tree search for video games raises the question of backing up the information from child nodes to parent nodes when the nodes and the arcs are abstract. An abstract arc may correspond to a joint sequence of actions between the players such as: let army A fight against army B until the outcome is known. Given a parent node with many kinds of sequences completing with different outcomes, the parent node could gather the information in a matrix of outcomes. This matrix could be processed by using game theoretic tools to find out a Nash equilibrium and to back up the set of best moves. The following question would be then how to integrate such a game theoretic approach within a minimax tree search or a MCTS approach.

4.4 Learning from Replays

Learning in real-time games features a mixture of strong potential advantages and practical barriers. Benefits include the possibility to adapt to a new environment or a new opponent, increasing the gaming experience of users. Learning can reduce the game development time, adding more automation to the process and creating intelligent AI bots more quickly. On the other hand, a system that learns often involves the existence of a large space of parameter combinations that gets explored during learning. A large number of parameter combinations makes a thorough game testing very difficult.

An important challenge is making industry developers more receptive to learning, identifying cases where potential disadvantages are kept within acceptable limits. Competitions can

facilitate the development and the promotion of learning in real-time games, as a stepping stone between the academia and the industry.

Learning from replays is particularly appealing because game traces are a natural source of input data which exists in high volumes from network data of human vs human games. There are many parts of a game that can benefit from learning, and multiple learning techniques to consider. Reinforcement learning has been used to learn team policies in first-person shooter games [69]. In domain-independent planning, sample plans have been used to learn structures such as macro-operators and HTNs which can greatly speed up a search.

4.5 Partitioning States

Partitioning states into regions, subareas, choke points, and other relevant clusters is often related to spatial and temporal constraints that are associated with these partitionings. In the reinforcement learning domain, state partitionings have been automatically identified to improve hierarchical reinforcement learning [66]. The main idea behind this approach is to focus state partitionings on particular states which separate different subareas, thus focusing on choke points. However, the automatic detection depends on the behavioral capabilities of the reinforcement learning agent, thus offering a more general purpose approach to the problem. With respect to factored Markov Decision Processes, Variable Influence Structure Analysis (VISA) has been proposed to develop hierarchical state decomposition by means of Bayesian networks [33]. With respect to skills and skill learning, hierarchical state partitionings have been successfully developed by automatically selecting skill-respective abstractions [37].

While it might not be straight-forward, these techniques stemming from the reinforcement learning and AI planning literature, seem ready to be employed in RTS games. The exact technical transfer, however, still needs to be determined. In the future, intelligent partitioning of states brings may bring closer the prospect of developing autonomously learning, self-developing agents.

4.6 Evolutionary Approaches

It is interesting to speculate as to whether evolutionary approaches which have been used successfully by authors such as Sipper [67] might be used in video games. Evolutionary algorithms have also been used in RTS-like domains with some success. For example, Stanley et al. [72] use real-time neuroevolution to evolve agents for simulated battle games, demonstrating that human-guided evolution can be successful at evolving complex behaviours. In another example, Louis and Miles [45] used a combination of case-based reasoning and genetic algorithms to evolve strategies, and later [49] combined this to co-evolve players for a tactical game by evolving influence maps, and combining these with A^* search. Genetic programming was used in [34] to evolve behavior trees of characters playing FPS game capturing basic reactive concepts of the tasks. In a light survey of evolution in games, Lucas and Kendall [46] discuss many applications of evolutionary search in games, including video games.

More recently, a number of researchers have been using evolutionary or co-evolutionary search in the context of Red Teaming (as mentioned in Subsection 3.1). In this approach, the evolutionary algorithm (or similar, such as a Particle Swarm Optimisation algorithm) is used to search over a space of possible strategies for a given scenario. These strategies are represented at a high level, for example as a set of paths to be followed by agents representing the resources available to each side, organised into squads. Strategies are then evaluated

using low-fidelity agent-based simulations, called *distillations*, in which these orders provide the overall goals of the agents, and simple rules determine their movements and other actions. The outcomes of these simulations provide fitness values to drive the evolutionary search.

The idea of organising agents into a heirarchical command structure is similar to the *Level of detail AI* abstraction discussed in Subsection 2.2. Another possible hybrid approach is to use temporal abstraction to subdivide the planning period, a coevolutionary search to simultaneously identify sets of strong strategies for each side at each decision point, and MCTS to select from these the strongest lines of play. An initial investigation of an idea on these lines, on a small example scenario, can be found in [2].

4.7 Competitions and Software Platforms

In recent years numerous competitions have emerged presenting researchers with a good opportunity to compare their AI approaches in specific games and scenarios – BotPrize, Starcraft, Simulated Car Racing and Demolition Derby, and ORTS to name only a few.

The drawback of some of these competitions is the narrow universe of the respective games. As a result, AI players are created specifically for the game and the competition's aim – e.g. AI in StarCraft exploits game mechanics for micromanagement to win unit on unit combat, but are too specialised for other RTSs let alone other game genres. Thus the challenge of generating a general purpose AI is lost.

To overcome this drawback, it is necessary to introduce a conceptual and technical layer between the designed AI and the used game. This could provide the capability to compare AI designs across various games and scenarios. Deeper understanding of the common problems and technical details of games (e.g. RTS, FPS) and AIs is necessary to produce platforms, tools, techniques, and methodologies for AI creation for games without limiting them to specific virtual environment. An interesting development here is the Atari 2600 simulator (see [3]) which allows precise simulation of 1980s arcade games at thousands of times normal speed due to increases in hardware performance.

The first possible approach is to actually create a layer providing game abstractions for all games of a certain genre and develop AIs on top of it, like the state of the art software platforms xAitment, Havok AI, AI-Implant, and Recast. The second approach is to create a configurable proto-game capable of reflecting the most important mechanics of all games in a respective field (e.g. ORTS aims to provide a description of game mechanics for RTSs), which would be build using the state of the art design patterns of the present games. The third possible approach is to create a multi-game competition utilizing multiple games of the same type where AIs cannot exploit a certain game. Finally, since solving open AI problems (e.g. pathfinding) is one of the main issues within the gaming industry, it would be beneficial to create competitions aimed at solving problems posed by the video games industry. This could draw the game industry's attention and help to establish a bridge between industry and academia.

5 Conclusion

Search algorithms are already integral to video game AI, with A* pathfinding used in a huge number of games. This paper has reflected upon the wider applications of search in games, particularly the use of search in operational, tactical and strategic decision making in order to provide a more interesting gameplay experience. It seems that there are many rich research directions here, as well as many opportunities for academics to work together with

those from the games industry and to build more interesting and repayable games in the future.

In this paper we have considered how we may combine advances in search algorithms (such as Monte Carlo Tree Search, minimax search and heuristically guided search) with effective tools for abstraction (such as Hierarchical Task Networks, dynamic scripting, "black box" strategy selection, player modelling and learning approaches) to yield a new generation of search-based AI approaches for video games. Continuing advances in CPU speed and memory capacity make computationally intensive search approaches increasingly available to video games that need to run in real time. We have discussed the challenges that new techniques must face, as well as promising directions to tackle some of these issues.

We expect to see many advances in search-based video game AI in the next few years. It is difficult to predict where the most important advances will come, although we hope that this paper may provide some insight into the current state of the art, and fruitful future research directions as foreseen by a range of leading researchers in AI for search and abstraction in games.

References

1 T. Anantharaman, M. Campbell, and F. Hsu. Singular extensions: adding selectivity to brute force searching. *Artificial Intelligence*, 43(1):99–109, 1989.

2 D. Beard, P. Hingston, and M. Masek. Using monte carlo tree search for replanning in a multistage simultaneous game. In *Evolutionary Computation, 2012 IEEE Congress on*, 2012.

3 M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *ArXiv e-prints*, July 2012.

4 L. Berger. *AI Game Programming Wisdom*, chapter Scripting: Overview and Code Generation. Charles River Media, 2002.

5 Yngvi Bjornsson, Vadim Bulitko, and Nathan Sturtevant. TBA*: time-bounded A*. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-09)*, pages 431–436, 2009.

6 Yngvi Björnsson and Hilmar Finnsson. CadiaPlayer: A Simulation-Based General Game Player. *IEEE Trans. Comp. Intell. AI Games*, 1(1):4–15, 2009.

7 A. Botea, M. Müller, and J. Schaeffer. Near optimal hierarchical path-finding. *Journal of Game Dev.*, 1:7–28, 2004.

8 Matthew M. Botvinick, Yael Niv, and Andrew C. Barto. Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective. *Cognition*, 113(3):262–280, 2009.

9 Bruno Bouzy. Associating domain-dependent knowledge and Monte-Carlo approaches within a go playing program. *Information Sciences*, 175(4):247–257, 2005.

10 M. Brockington. *AI Game Programming Wisdom*, chapter Level-Of-Detail AI for a Large Role-Playing Game. Charles River Media, 2002.

11 Cyril Brom, T. Poch, and O. Sery. AI level of detail for really large worlds. In Adam Lake, editor, *Game Programming Gems 8*, pages 213–231. Cengage Learning, 2010.

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. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, March 2012.

13 B. Bruegmann. Monte Carlo Go. ftp://www.joy.ne.jp/welcome/igs/Go/computer/mcgo.tex.Z, 1993.

14 Sébastien Bubeck, Rémi Munos, Gilles Stoltz, and C. Szepesvari. Online optimization in x-armed bandits. In *Proceedings NIPS*, pages 201–208, 2008.

15 M. Buro and T. Furtak. RTS games and real-time AI research. In *Proceedings of the Behaviour Representation in Modeling and Simulation Conference*, pages 51–58, 2004.

16 Martin V. Butz, Olivier Sigaud, Giovanni Pezzulo, and Gianluca Baldassarre. Anticipations, brains, individual and social behavior: An introduction to anticipatory systems. In Martin V. Butz, Olivier Sigaud, Giovanni Pezzulo, and Gianluca Baldassarre, editors, *Anticipatory Behavior in Adaptive Learning Systems: From Brains to Individual and Social Behavior*, pages 1–18. 2007.

17 M. Campbell, A.J. Hoane, and F-H Hsu. Deep Blue. *Artificial Intelligence*, 134:57–83, 2002.

18 A Champandard, T Verweij, and R Straatman. The AI for Killzone 2's multiplayer bots. In *Proceedings of Game Developers Conference*, 2009.

19 Guillaume Maurice Jean-Bernard Chaslot, Jahn-Takeshi Saito, Bruno Bouzy, Jos W. H. M. Uiterwijk, and H. Jaap van den Herik. Monte-Carlo Strategies for Computer Go. In *Proc. BeNeLux Conf. Artif. Intell.*, pages 83–91, Namur, Belgium, 2006.

20 Remi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In H. van den Herik, Paolo Ciancarini, and H. Donkers, editors, *Computers and Games*, volume 4630 of *Lecture Notes in Computer Science*, pages 72–83. Springer Berlin / Heidelberg, 2007.

21 Peter I. Cowling, Edward J. Powley, and Daniel Whitehouse. Information Set Monte Carlo Tree Search. *IEEE Trans. Comp. Intell. AI Games*, 4(2), 2012.

22 K. Currie and A. Tate. O-Plan: The open planning architecture. *Artificial Intelligence*, 52(1):49–86, 1991.

23 Douglas Demyen and Michael Buro. Efficient triangulation-based pathfinding. In *Proceedings of the 21st national conference on Artificial intelligence – Volume 1*, AAAI'06, pages 942–947. AAAI Press, 2006.

24 Sylvain Gelly and David Silver. Monte-carlo tree search and rapid action value estimation in computer go. *Artificial Intelligence*, 175(11):1856–1875, 2011.

25 Sylvain Gelly, Yizao Wang, Rémi Munos, and Olivier Teytaud. Modification of UCT with Patterns in Monte-Carlo Go. Research Report RR-6062, INRIA, 2006.

26 Phillip Gowlett. Moving forward with computational red teaming. Technical Report DSTO-GD-0630, Joint Operations Division, Defence Science and Technology Organisation, Department of Defence, Fairbairn Business Park Department of Defence Canberra ACT 2600 Australia, March 2011.

27 Ross Graham, Hugh Mccabe, and Stephen Sheridan. Pathfinding in computer games. *ITB Journal Issue Number*, 8:57–81, 2003.

28 D. Harabor and A. Botea. Hierarchical Path Planning for Multi-Size Agents in Heterogeneous Environments. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games CIG-08*, pages 258–265, Perth, Australia, December 2008.

29 P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on System Sciences and Cybernetics*, SSC-4(2):100–107, 1968.

30 Masahiko Haruno, Daniel M. Wolpert, and Mitsuo Kawato. Hierarchical mosaic for movement generation. In T. Ono, G. Matsumoto, R.R. Llinas, A. Berthoz, R. Norgren, H. Nishijo, and R. Tamura, editors, *Excepta Medica International Coungress Series*, volume 1250, pages 575–590, Amsterdam, The Netherlands, 2003. Elsevier Science B.V.

31 Chad Hogg, Héctor Muñoz Avila, and Ugur Kuter. Htn-maker: learning htns with minimal additional knowledge engineering required. In *Proceedings of the 23rd National Conference on Artificial intelligence – Volume 2*, AAAI'08, pages 950–956. AAAI Press, 2008.

32 R. Houlette and D. Fu. *AI Game Programming Wisdom 2*, chapter The Ultimate Guide to FSMs in Games. Charles River Media, 2003.

33 Anders Jonsson and Andrew Barto. Causal graph based decomposition of factored mdps. *Journal of Machine Learning Research*, 7:2259–2301, December 2006.

34 Rudolf Kadlec. Evolution of intelligent agent behaviour in computer games. Master's thesis, Faculty of Mathematics and Physics, Charles University in Prague, 2008.

35 D. Knuth and R. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6:293–326, 1975.

36 Levente Kocsis and Csaba Szepesvari. Bandit based monte-carlo planning. In Johannes Furnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer Berlin / Heidelberg, 2006.

37 G. Konidaris and A. Barto. Efficient skill learning using abstraction selection. *Proceedings of the 21st International Joint Conference on Artifical Intelligence*, pages 1107–1112, 2009.

38 R. Korf. Depth-first Iterative Deepening: An Optimal Admissible Tree Search. *Artificial Intelligence*, 97:97–109, 1985.

39 R. Korf. Depth-first Iterative Deepening: an Optimal Admissible Tree Search. *Artificial Intelligence*, 27:97–109, 1985.

40 Richard Korf. Multi-player alpha-beta pruning. *Artificial Intelligence*, 49(1):99–111, 1991.

41 Ugur Kuter and Dana S. Nau. Forward-chaining planning in nondeterministic domains. In *National Conference on Artificial Intelligence (AAAI)*, pages 513–518, July 2004.

42 Ugur Kuter and Dana S. Nau. Using domain-configurable search control for probabilistic planning. In *National Conference on Artificial Intelligence (AAAI)*, pages 1169–1174, July 2005.

43 Ugur Kuter, Dana S. Nau, Marco Pistore, and Paolo Traverso. Task decomposition on abstract states, for planning under nondeterminism. *Artifical Intelligence*, 173:669–695, 2009.

44 H. Hoangand S. Lee-Urban and H. Munoz-Avila. Hierarchical plan representations for encoding strategic game ai. In *Proceedings of Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE-05)*, 2005.

45 S.J. Louis and C. Miles. Playing to learn: case-injected genetic algorithms for learning to play computer games. *Evolutionary Computation, IEEE Transactions on*, 9(6):669 – 681, dec. 2005.

46 S.M. Lucas and G. Kendall. Evolutionary computation and games. *Computational Intelligence Magazine, IEEE*, 1(1):10–18, Feb. 2006.

47 C. Luckhardt and K.B. Irani. An algorithmic solution for n-person games. In *5th national Conference on Artificial Intelligence (AAAI)*, volume 1, pages 158–162, 1986.

48 C. Mansley, A. Weinstein, and M. Littman. Sample-based planning for continuous action markov decision processes. In *Proceedings ICAPS*, 2011.

49 C. Miles and S.J. Louis. Towards the co-evolution of influence map tree based strategy game players. In *Computational Intelligence and Games, 2006 IEEE Symposium on*, pages 75–82, May 2006.

50 Roger B. Myerson. *Game Theory: Analysis of Conflict*. Harvard University Press, 1997.

51 Dana S. Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research (JAIR)*, 20:379–404, December 2003.

52 Dana S. Nau, S. Smith, and T. Throop. The Bridge Baron: A big win for AI planning. In *European Conference on Planning (ECP)*, September 1997.

53 Pim Nijssen and Mark Winands. Enhancements for multi-player monte-carlo tree search. In *ACG13 Conference*, Tilburg, 2011.

54 Martin J. Osborne. *An Introduction to Game Theory*. Oxford, 2004.

55 J. Pearl. Asymptotic properties of minimax trees and game-searching procedures. *Artificial Intelligence*, 14:113–138, 1980.

56 A. Plaat, J. Schaeffer, W. Pils, and A. de Bruin. Best-first fixed depth minimax algorithms. *Artificial Intelligence*, 87:255–293, November 1996.

57 Tomaso Poggio and Emilio Bizzi. Generalization in vision and motor control. *Nature*, 431:768–774, 2004.

58 I. Pohl. Heuristic Search Viewed as Path Finding in a Graph. *Artificial Intelligence*, 1(3):193–204, 1970.

59 F. Poiker. *AI Game Programming Wisdom*, chapter Creating Scripting Languages for Non-Programmers. Charles River Media, 2002.

60 Edward J Powley, Daniel Whitehouse, Graduate Student Member, and Peter I Cowling. Monte Carlo Tree Search with macro-actions and heuristic route planning for the Physical Travelling Salesman Problem. In *Proc. IEEE CIG, submitted*, 2012.

61 Franisek Sailer, Michael Buro, and Marc Lanctot. Adversarial planning through strategy simulation. In *Proceedings of the Symposium on Computational Intelligence and Games (CIG)*, pages 80–87, 2007.

62 J. Schaeffer. The history heuristic and Alpha-Beta Search Enhancements in Practice. *IEEE Transcations on Pattern Analysis and Machine Intelligence*, 11(11):1203–1212, November 1989.

63 Jonathan Schaeffer, Yngvi Björnsson Neil Burch, Akihiro Kishimoto, Martin Müller, Rob Lake, Paul Lu, and Steve Sutphen. Checkers is solved. *Science*, 317(5844):1518–1522, 2007. Work named by Science Magazine as one of the 10 most important scientific achievements of 2007.

64 B. Scott. *AI Game Programming Wisdom*, chapter Architecting an RTS AI. Charles River Media, 2002.

65 C.E. Shannon. Programming a computer to play Chess. *Philosoph. Magazine*, 41:256–275, 1950.

66 Özgür Simsek and Andrew G. Barto. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proc. ICML-2004*, pages 751–758, 2004.

67 Moshe Sipper. *Evolved to Win*. Lulu, 2011. Available at `http://www.lulu.com/`.

68 D.J. Slate and L.R. Atkin. Chess 4.5 – the northwestern university chess program. In P. Frey, editor, *Chess Skill in Man and Machine*, pages 82–118. Springer-Verlag, 1977.

69 Megan Smith, Stephen Lee-Urban, and Hector Muñoz-Avila. Retaliate: Learning winning policies in first-person shooter games. In *AAAI*, 2007.

70 Stephen J. J. Smith, Dana S. Nau, and Thomas Throop. Computer bridge: A big win for AI planning. *AI Magazine*, 19(2):93–105, 1998.

71 P. Spronck. *AI Game Programming Wisdom 3*, chapter Dynamic Scripting. Charles River Media, 2006.

72 K.O. Stanley, B.D. Bryant, and R. Miikkulainen. Real-time neuroevolution in the nero video game. *Evolutionary Computation, IEEE Transactions on*, 9(6):653–668, Dec. 2005.

73 Nathan Sturtevant. An analysis of uct in multi-player games. *International Computer Games Association*, 31(4):195–208, December 2011.

74 Nathan Sturtevant and Richard Korf. On pruning techniques for multi-player games. In *7th National Conference on Artificial Intelligence (AAAI)*, pages 201–207. MIT Press, 2000.

75 Nathan Sturtevant, Martin Zinkevich, and Michael Bowling. Prob-max-n: Playing n-player games with opponent models. In *National Conference on Artificial Intelligence (AAAI)*, 2006.

76 Nathan R. Sturtevant and Michael Buro. Partial pathfinding using map abstraction and refinement. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pages 1392–1397, 2005.

77 A. Tate, B. Drabble, and R. Kirby. *O-Plan2: An Architecture for Command, Planning and Control*. Morgan-Kaufmann, 1994.

78 Julian Togelius, G Yannakakis, K Stanley, and Cameron Browne. Search-based procedural content generation: A taxonomy and survey. *Computational Intelligence and AI in Games, IEEE Transactions on*, (99):1–1, 2011.

79 H. J. van den Herik, H. H. L. M. Donkers, and P. H. M. Spronck. Opponent Modelling and Commercial Games. In G. Kendall and S. Lucas, editors, *Proceedings of IEEE 2005 Symposium on Computational Intelligence and Games CIG'05*, pages 15–25, 2005.

80 Tim Verweij. A hierarchically-layered multiplayer bot system for a first-person shooter. Master's thesis, Vrije Universiteit of Amsterdam, 2007.

81 J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

82 David E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann, San Mateo, CA, 1988.

83 Brandon Wilson, Inon Zuckerman, and Dana S. Nau. Modeling social preferences in multi-player games. In *Tenth Internat. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, 2011.

84 Peter Yap, Neil Burch, Robert C. Holte, and Jonathan Schaeffer. Block A*: Database-driven search with applications in any-angle path-planning. In *AAAI*, 2011.

85 J. P. Zagal and M. Mateas. Time in Video Games: A Survey and Analysis. *Simulation & Gaming*, 41(6):844–868, August 2010.

86 A. Zobrist. A new hashing method with application for game playing. *ICCA Journal*, 13(2):69–73, 1990.

Pathfinding in Games

Adi Botea[1], Bruno Bouzy[2], Michael Buro[3], Christian Bauckhage[4], and Dana Nau[5]

1 **IBM Research, Dublin, Ireland**
adibotea@ie.ibm.com
2 **Université Paris Descartes, France**
bruno.bouzy@parisdescartes.fr
3 **University of Alberta, Canada**
mburo@cs.ualberta.ca
4 **Fraunhofer IAIS, 53754 Sankt Augustin, Germany**
christian.bauckhage@iais.fraunhofer.de
5 **University of Maryland, USA**
nau@cs.umd.edu

Abstract

Commercial games can be an excellent testbed to artificial intelligence (AI) research, being a middle ground between synthetic, highly abstracted academic benchmarks, and more intricate problems from real life. Among the many AI techniques and problems relevant to games, such as learning, planning, and natural language processing, pathfinding stands out as one of the most common applications of AI research to games. In this document we survey recent work in pathfinding in games. Then we identify some challenges and potential directions for future work. This chapter summarizes the discussions held in the pathfinding workgroup.

1998 ACM Subject Classification I.2.8 Problem Solving, Control Methods, and Search

Keywords and phrases path finding, search, games

Digital Object Identifier 10.4230/DFU.Vol6.12191.21

1 Introduction

Pathfinding is an important application of artificial intelligence to commercial games. Despite a significant progress seen in recent years, the problem continues to constantly attract researchers' attention, which is explained in part by the high performance demands that pathfinding implementations have to satisfy in a game. Pathfinding engines are allocated limited CPU and memory resources. Moves have to be computed in real time, and often there are many mobile units to compute paths for. Paths have to look sufficiently realistic to a user. Besides a more standard, single-agent pathfinding search on a fully known, static map, games feature more complicated variations of the problem, such as multi-agent pathfinding, adversarial pathfinding, dynamic changes in the environment, heterogeneous terrains and mobile units, incomplete information, and combinations of these.

This chapter looks at games pathfinding research, with an emphasis towards the future. It summarizes the discussions held in the Pathfinding Workgroup at the Dagstuhl Seminar on Computational and Artificial Intelligence in Games. A shorter summary of the group's findings is available as part of a report on the entire seminar [24]. The first half of this document overviews recent progress in the area. Then, current challenges and future work directions are discussed.

Artificial and Computational Intelligence in Games. *Dagstuhl Follow-Ups*, Volume 6, ISBN 978-3-939897-62-0.
Editors: Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius; pp. 21–31

Dagstuhl Publishing
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany

2 Survey of Pathfinding Methods

We start by indicating a popular, baseline approach to pathfinding. The three key elements of the baseline approach are a graph representation of a map, a search algorithm, and a heuristic function to guide the search. Grid maps are a popular way of discretizing a game map into a search graph. As part of the process, a map is partitioned into atomic square cells, sometimes called tiles. Depending on the map initial topology, a tile is marked as either traversable or blocked. A mobile unit can occupy one traversable tile at a time. Traversable tiles become nodes in the search graph. Graph edges connect adjacent traversable tiles. Depending on the grid type, adjacencies are defined in the 4 cardinal directions (4-connected grid maps), or in the 4 cardinal + 4 diagonal directions (8-connected grid maps).

According to Björnsson et al. [1], the A* [16] algorithm used to be the de facto standard in pathfinding search. A* is a well known best-first search method, used in many other search problems, including AI planning and puzzle solving, to name just a few. To speed up a search, A* uses a heuristic function $h(n)$, which estimates the distance from a given node n to a goal node. Heuristics that do not overestimate the true distance to a goal are said to be admissible. A* with an admissible heuristic returns optimal solutions. On 4-connected maps, the Manhattan distance is a well known admissible heuristic. Given two nodes with coordinates (x_1, y_1) and (x_2, y_2), the Manhattan distance is defined as $\Delta x + \Delta y$, where $\Delta x = |x_1 - x_2|$ and $\Delta y = |y_1 - y_2|$. On a map with no obstacles, the Manhattan distance coincides with the true distance. The Octile distance is a variant of the Manhattan distance adapted to 8-connected maps. It is defined as $\sqrt{2} \times m + (M - m)$, where $m = \min(\Delta x, \Delta y)$ and $M = \max(\Delta x, \Delta y)$.

Part of the work overviewed in the rest of this section presents extensions of the baseline approach, being based on online search. Extensions include using more informative heuristic methods, replacing a low-level grid map with a hierarchy of smaller search graphs, exploiting symmetries to prune the search space in A* search, and using triangulation map decomposition instead of grid maps.

Other contributions eliminate graph search at runtime. For example, compressed path databases rely on a database of pre-computed all-pairs shortest path (APSP) data. Moves are retrieved from a compressed APSP table. This is suitable to games where all potential paths are precomputed, to largely free the CPU from the burden of path computation during game play. The data compression helps reducing the memory overhead associated with caching the precomputed paths.

2.1 Beyond Manhattan: Modern Pathfinding Heuristics on Grid Maps

The Manhattan and the Octile heuristics have the advantage of being simple, fast to compute, and reasonably accurate on many map topologies. However, more informed heuristics can reduce the number of states expanded (or visited) in a search. In this section we overview other heuristics in pathfinding, which can achieve a better accurracy, depending on factors such as the map topology. They use pre-processing and additional memory to cache the results of the pre-processing.

Goldberg and Harrelson [9] describe an admissible heuristic as part of their ALT search method. Consider a given node l, called a landmark, on the graph. A pre-computation step provides a look-up table with true distances $d(n, l)$ from every node n to l. For simplicity, assume an undirected graph. Given two arbitrary nodes n_1 and n_2, $|d(n_1, l) - d(n_2, l)|$ is an admissible estimation of the true distance from n_1 to n_2. The ALT admissible heuristic takes the maximum $\max_l |d(n_1, l) - d(n_2, l)|$ over a subset of landmarks. Goldberg and Har-

relson applied their ideas to graphs such as roadmaps and grids with non-uniform costs. The heuristic has independently been developped by other researchers, including Sturtevant et al. [32], who call it the differential heuristic. Goldenbert et al. [11] show how the differential heuristic can be compressed. As an example of when the ALT heuristic is more informative than the Manhattan heuristic, we note that ALT can potentially identify unreachable locations (∞ heuristic value), whereas Manhattan might return a finite value.

ALTBest$_P$ [6] is a variant of the ALT heuristic that uses one single landmark, from a pre-defined subset of P landmarks. Specifically, at the beginning of a search, a landmark is selected that gives the highest heuristic distance for the initial state: $\arg\max_l |d(s,l) - d(g,l)|$, where s and g are the start and the goal. Furthermore, at each evaluated node, the heuristic given by the selected landmark is compared with the Manhattan heuristic, and their maximum is used as a heuristic evaluation.

Björnsson and Halldórsson [2] introduce the *dead-end heuristic* and the *gateway heuristic*, two methods that use an automated decomposition of the map into disjoint areas. The dead-end heuristic identifies areas that are irrelevant when solving a given instance, and assign a heuristic value of ∞ to all locations contained in such areas. The gateway heuristic identifies gateway points between adjacent areas in the partitioning, and pre-computes a table with optimal distances between pairs of entrances. Thus, additional memory, with a size quadratic to the number of gateways, is traded for an improved heuristic accuracy. A similar trade-off is taken by Sturtevant et al. [32], with their *canonical heuristic*, and Goldenberg et al. [10], with the *portal heuristic*. Sturtevant et al. [32] use the generic name of *true distance heuristics* for the family of heuristics that store exact distances between pairs of selected nodes.

2.2 Hierarchical Abstraction on Grid Maps

Hierarchical abstraction can potentially speed up a search by replacing it with a series of smaller searches. Abstract solutions computed in a higher-level state space representation are gradually refined, possibly through several hierarchical levels, down to the ground level of the original search space.

Pathfinding is an application where hierarchical abstraction has been successful. Hierarchical pathfinding methods, such as work mentioned in this section, can ensure that an initial abstract solution can be refined into a low-level solution. This eliminates the need to backtrack across hierarchical levels, thus eliminating a source of a potentially great slow down.

HPA* [3] decomposes a map into disjoint square sectors. Entrance points between adjacent sectors are identified and added as nodes into an abstracted search space. Abstract edges connect pairs of entrance points placed on the border of the same sector. In effect, in the abstracted space, a move traverses one sector in one step. Additional abstracted edges connect the start node (and the target) to the entrance points of its sector. An abstract solution contains macro moves such as sector-traversing moves. In a refinement step, a search restricted to the area of one sector converts a macro step into a sequence of actual moves. The method can have more than 2 hierarchical levels.

Partial Refinement A* (PRA* [34]) combines hierarchical abstraction with fast partial path refinement to effectively interleave path planning with path execution, which is relevant to robotics and video games. PRA* first builds a hierarchical map respresentation bottom-up by mapping small connected regions (e.g., traversable 2x2 tile squares) to tiles on the next level of abstraction while preserving connectivity. This greedy abstraction process creates pyramids of coarser and coarser map representations up to the point in which only single

tiles remain that represent the original connected components. For finding paths subject to given time and path quality constraints, we choose an abstraction layer and find nodes representing the start and endpoints on the original map. Starting at this level, PRA*(k) uses A* to compute the shortest path and then projects the first k steps down to the next map abstraction level, where the shortest path is determined by only considering a small corridor around the projected path, etc. This process quickly finds high-quality initial paths on the original map which allows objects to start moving in the right direction quickly. Later, Sturtevant [33] described a hybrid approach that combines ideas of HPA* and PRA* to decrease memory used for abstraction.

HAA* [12] extends HPA* in two directions. First off, in classical pathfinding, a map is partinioned into traversable terrain and blocked areas. HAA* makes a finer distinction, allowing to define several types of terrain, such as ground, water, and walls. Secondly, HAA* handles mobile units with variable sizes and variable terrain traversal capabilities.

The idea of splitting a map into disjoint rectangular blocks is also exploited in Block A* [39]. Given a pre-determined block size $m \times n$, Block A* pre-computes a database of all possible block topologies (i.e., all possible combinations of traversable and blocked grid cells). For each block, optimal distances between pairs of boundary cells are stored. At runtime, Block A* expands an entire block at a time, as opposed to one individual cell, as it is the case in regular A*. Block A* is adapted to perform any-angle pathfinding [8], which, unlike standard grid-based pathfinding, is not limited to the 8 major compass directions.

While the map decomposition used in HPA*, HAA* and Block A* explicitly assumes a gridmap encoding of the underlying topology, it seems that they could in principle be extended to other graphs with an (almost) planar structure. One possible approach would be to slice the bounding rectangle of the graph with square or rectangular blocks, obtaining the map decomposition. At the same time, PRA* appears to be more straightforward to apply to non-gridmap graphs, as its core abstraction method relies on identifying cliques.

2.3 Symmetry Elimination

The frequently used assumption that traversable areas on a map have a uniform traversal cost leads to a considerable degree of symmetry in a pathfinding problem. Consider a path encoding where each move is represented as a direction. For example, on a 4-connected gridmap, a path from A to B that goes three steps to the left and three steps upwards would be encoded as $lllluuu$. Assumming there are no obstacles in the grid-aligned rectangle having A and B as opposite corners, there are many equivalent paths from A to B, obtained by interleaving the moves: $llluluu$, $lluulu$ and so on. This can lead to an excessive, useless exploration of some parts of a map.

Symmetry reduction can be based on empty rectangles identified on a map [13, 14]. Given two nodes A and B on the border of an empty rectangle, all optimal paths connecting these are symmetrical to each other, and considering only one such a path is sufficient. Thus, search can be sped up by defining macro-operators to cross rectangles without visiting any interior nodes. Jump point search [15] takes a different approach to symmetry elimination. It allows pruning part of the successors of a node, ensuring that optimality and completeness are preserved.

2.4 Triangulation-Based Pathfinding

Grid-based map representations are common in pathfinding applications such as video games, because of their conceptual simplicity that leads to straight-forward and fast im-

plementations. However, there are several drawbacks that may render grid-based representations less useful in applications in which world objects don't have uniform sizes or shapes, they are not axis-aligned, or they can move in arbitrary directions. Another problem is that basic grid-based representations waste space in case large map regions are empty. To address these problems Demyen and Buro [7] present two pathfinding algorithms TA* (Triangulation A*) and TRA* (Triangulation Reduction A*). TA* uses dynamic constrained Delaunay triangulation (DCDT [18]) to represent maps in which obstacles are defined by polygons, and finds optimal any-angle paths for circular objects by running an A* like algorithm on graphs induced by the triangulation.

TRA* improves on TA* by applying a topological map abstraction that contracts each corridor in the triangulation graph to a single edge and tree components in which pathfinding is trivial are removed and handled separately. What is left is a graph only containing degree-3 nodes. Its size is proportional to the number of obstacles on the map, which can be considerably smaller than the triangulation graph. Because of this size reduction, TRA* runs faster than TA* and much faster than A* on common game maps while generating high-quality paths. With the geometric map represenation the mentioned restrictions of grid-based approaches are lifted. A TA* variant is being used in the StarCraft 2 game engine.

2.5 Real-Time Search

Real-time search [20] works under the assumption that an agent has a limited amount of time before having to make a move. Many researchers have addressed the problem since Korf's original work [20]. The general solving framework includes a planning step, a learning step and an acting step. The cycle repeats until the agent reaches the target. In the planning step, the agent explores the area around its current location. Learning uses the results of the exploration to update knowledge such as heuristic estimations of the current state or other states. Acting refers to making one or several moves, changing the current state of the agent. The many real-time algorithms available differ in the exploration strategy (e.g., A* search vs breadth first search), how to perform the heuristic update, and what state to select as the next current state.

For a long time, real-time search has not been considered as a viable approach to be implemented in a game. A main cause is the phenomenon of *scrubbing*: when real-time search reaches a plateau, an agent has to slowly learn more accurate heuristic values for the states in the plateau, before being able to escape. This results in an irrational-looking moving pattern of the agent, moving around within a local area for a long time. Solution paths can be very long, and so can be the total time to reach the target.

Recently, a line of research on real-time search has focused on eliminating, either completely or to a great extent, the learning step and thus the scrubbing behaviour [5, 23]. A key insight that contributes to the success of such methods is the idea of using *hill-climbing moves*. These are moves that look the most promising after searching forward only one step ahead, i.e., evaluating only the successors of a current state. Often, two nodes can be connected by performing solely hill-climbing moves. For example, consider two nodes within an area with no obstacles. When a path can be decomposed into subgoals such that the next sub-goal can be reached with hill-climbing moves, moves are computed fast, with no learning and scrubbing necessary.

2.6 Compressed Path Databases

Most pathfinding methods search at the path query time to compute a path on demand. An alternative approach is to perform path precomputation and lookup [36], returning paths orders of magnitude faster than online search. Straightforward ways of storing precomputed paths have prohibitive memory requirements, which has been a major impediment to adopting path lookup on a larger scale. Recently, a structure called a compressed path database [4] has been shown to achieve a massive lossless compression, bringing memory needs down to reasonable values for many grid maps. Compressed path databases exploit an idea previously observed and used in [27], with no application to games. In many cases, the first move on an optimal path, from a start location s to any target t in a contiguous remote area, is the same. The original authors call this property path coherence. For a start–target pair (s, t), let the first-move label of t identify an optimal move to take in s towards t. Compression involves finding, for a given s, a contiguous area A where all the targets $t \in A$ have the same first move label. This allows to replace many identical first-move labels, one for each (s, t) pair, with one single record for the pair (s, A).

Advantages of compressed path databases include the speed of path retrieval and the optimality of solutions. The first-move lag (i.e., the time to wait until the first move is performed) is very low, since each move is retrieved independently from the subsequent moves on the path. At the same time, compressed path databases pose a number of challenges to be addressed in the future, as we discuss later in this report.

2.7 Multi-Agent Pathfinding

Multi-agent pathfinding addresses the problem of finding paths for a set of agents going to their goals. Each agent may have its own goal or all the agents may have a global goal. At each timestep, all agents move according to their plans. The set of agents has to find the minimal cost for reaching the set of goals, or to maximize some quantity. The cost can be the elapsed time. Compared to mono-agent pathfinding, two obstacles appear clearly. The first one is the size of the set of (joint) actions which is exponential in the number of agents, which lead to difficulty for a global planning method. And when considering the elementary solutions to each mono-agent problem in order to find a global solution, the second one is managing collisions between agents.

Furthermore, multi-agent pathfinding can be cooperative, when all the agents help each other and try to optimize in the same direction. But it can be adversarial when an army tries to reach a point and another one prevents this from happening.

Starting points for cooperative pathfindind (CPF) are optimal methods like centralized A*. They work well on small problems. To avoid the exponential number of actions in the number of agents, Standley proposes the mechanism of operator decomposition (OD) [30]. Instead of considering that the set of agents decides its joint action (operator), the agents decide their elementary action in turn, one after each other. With some care to special cases, A* associated with OD and a perfect heuristic is admissible and complete [30]. Another idea for optimal solutions is independence detection (ID) [31]. Each agent plans its optimal path to the goal with A*. Then conflicts are detected, and resolved if possible. When a conflict between two agents is not solvable, the two agents are gathered into a group, and an optimal solution is found for this group. Conflict between groups are detected and so forth. When few conflicts occur this method avoids the exponential size of the set of actions, but in the worst case, this method is inferior to any centralized method.

The Increasing Cost Tree Search (ICTS) [28] is a two-level search: a global level and a

low level. At the global level, the search proceeds on an Incremental Cost Tree (ICT) in which one node corresponds to a vector of costs. The global cost is the sum of individual cost heuristic (SIC). At the root of the ICT the costs are the optimal costs of agents considered alone, and the global cost cannot be smaller. At depth δ, the sum of the costs of a node is the optimal cost plus δ. A node is a goal for ICT iff there is an effective combination of individual paths without conflict and with costs corresponding to the costs of the node. ICT search is a breadth-first search for a goal node that progressively increases the best possible optimal cost.

In video games, optimal complete methods are not always searched because they are not scalable. Instead, suboptimal, incomplete but tractable methods exist. Cooperative A* [29] decomposes the whole task into a series of single agent searches in a three-dimensional space including time. A wait move is added to the set of actions of the agents. The computed routes are stored in a reservation table that forbids to use its entries in future searches. Hierarchical cooperative A* (HCA*) uses the idea of Hierarchical A* [17] with the simplest possible hierarchy: the domain with all agents removed. An issue of HCA* is how to terminate: sometimes an agent sitting on its destination must move to give the way to another agent. Another issue is the ordering of agents, and the most important one is computing complete routes in the 3-dimensional space. Therefore, Windowed HCA* (WHCA*) simply limits the lookahead in each search it performs to a fixed-horizon time window [29].

Multi-agent pathfinding can be solved suboptimally in low-polynomial time [21, 26]. However, the practical performance of the available complete method is an open question. Recent research has lead to the creation of multiple suboptimal methods that are complete on well-specified subclasses of instances. Push-and-Swap [25] works on problems with at least two free nodes (i.e., problems where the number of nodes in the graph representing the map exceeds the number of mobile units by at least two). MAPP [38] works on problems of the Slidable class. TASS [19] works on problems with at least four free nodes. The TASS method is a tree-based agent swapping strategy. There are other sub-optimal methods that work well in practice, but whose completeness range has not been characterized [29, 35, 37].

3 Looking at the Future

In this section we discuss a number of current challenges and directions for the future.

3.1 Bridging the Industry–Academia Gap

As discussed earlier in the report, recently there has been a significant progress in academic research in multi-agent path finding. Computer games are often mentioned as an application domain in research papers, and game maps have been used as benchmark data. However, it appears that there is a need for a better alignment between problems addressed in academic research and problems that are truly relevant to game developers. Recent academic research has focused on objectives such as computing optimal solutions, or scaling sub-optimal methods to hundreds or even thousands of mobile units. Such objectives are important on their own but, in many real-time strategy games, solution optimality is not strictly needed, and there can be relatively few units interacting on a portion of a map. In such games, the ability to encircle and attack a moving enemy unit, which combines elements of multi-agent collaborative pathfinding, multi-agent adversarial pathfinding, and multi-agent moving-target search, could be quite important.

We believe that a better industry–academia integration in multi-agent pathfinding could benefit if game developers make available precise definitions of some of their problems and interest, and possibly data to experiment with. They could be made available on their own, or be part of a competition open to academic researchers. Recently, Sturtevant has organized a Grid-Based Path Planning Competition[1]. This first edition focused on single-agent pathfinding on fully known, static maps. This could pave the way towards more diverse types of competition problems, such as multi-agent path planning.

In multi-agent pathfinding, non-interfering paths that bring each agent from its current state to its goal state must be planned. The agents are considered as friendly, and one agent may give the way to another one. For example, workers mining and transporting ore is such a problem. The complexity of a problem depends on the connectivity of the grid, the rules governing agent movements, the size of the grid, the number of obstacles, the number of agents. The challenge is to find solutions that avoid collisions and deadlocks in the context of fast computations. Future work consist in filling the gap between optimal algorithms solving relatively simple problems and tractable algorithms solving very complex problems but not optimally.

In adversarial multi-agent pathfinding, encircling behaviors, pursuit behaviors and/or escaping behaviors must be planned. For example, encircling an enemy unit to attack it, guarding one or several chokepoints, catching up a prey. Some agents are inimical and their meetings lead to fight whose outcomes can be the partial weakening or the complete destruction of some of the agents. However, some other agents are friendly and must coordinate their movements to reach a goal needing cooperation, such as circling a prey. The terminal conditions of the plan can be fuzzy and hard to determine, and planning can be continual. Future work include the integration of the adversarial multi-agent pathfinding solutions with game tree solutions to video games considered as two-player games.

3.2 From Smart Agents to Smart Maps

While single-agent path planning can naturally be modelled with an agent-centered approach, other problems may require a decision-making entity that is different from individual mobile units. For example, in multi-agent path planning, both views could be considered. A *distributed* approach would be agent-centered, allowing each agent to make decisions based on its (possibly incomplete) knowledge about the rest of the problem. Many multi-agent pathfinding algorithms, however, including most methods discussed in this report, assume the existence of a centralized decision maker.

Taking the idea further, we argue that part of the intelligence built in a path-finding module could be associated with parts of the environment in a distributed manner. For example, a notion of a *smart chokepoint* could be used to schedule incoming units through a narrow area of a map. The idea is similar to a traffic light system controlling an individual intersection on a road map. More generally, local portions of a map, not restricted to certain types of topologies could be responsible for a smooth navigation inside their own areas.

A dynamic environment offers an additional scope for developing intelligent map functionality. As pointed out earlier in the report, advanced pathfinding algorithms can use pre-computed datastructures for a faster path computation. Dynamic changes can invalidate underlying datastructures. A smart map should be able to know when to trigger a repair, what area of the map should be affected, how to perform a repair, and how to handle ongoing navigation while a local datastructure is still under repair.

[1] `http://movingai.com/GPPC/index.html`

3.3 Time and Memory Constraints

While common formulations of pathfinding problems, such as single-agent pathfinding on a static graph, have a low polynomial complexity, the challenge is to satisfy the ambitious CPU and memory limits set in actual games. There tends to be a speed–memory trade-off, exhibited, for example, by methods that perform pre-processing and use the cached results of pre-processing for a faster online pathfinding. Path quality is a third performance measure that must be considered. The challenge is to improve the performance on one or more such criteria, without paying a price on the remaining criteria.

Compressed path databases (CPDs) illustrate the speed–memory tradeoff well, being a fast but memory-consuming method. Compressed path databases are more memory intensive than other pathfinding methods, such as techniques that involve A* search. Thus, we use CPDs as a case study, discussing how its memory footprint could be improved.

Future efforts on reducing their size could aim at preserving path optimality, or could relax this requirement, as a tradeoff to a better size reduction. In games, the optimality of solutions is not seen as a must-have feature. Suboptimal paths that look reasonable to a user are acceptable. A promising idea that involves suboptimal paths is combining compressed path databases with hierarchical abstraction. Recently, Kring et al. [22] have extended the HPA* algorithm to cache (uncompressed) all-pairs shortest paths for small rectangular subparts of a map. The idea could be taken further to store such local all-pairs shortest paths in a compressed form, in the style of CPDs. Furthermore, at the global level, a compressed path database could be computed for the abstract graph that HPA* uses. This approach would result in a hierarchical, two-level compressed path database.

Finally, we advocate a closer interaction with communities working on related problems, such as robot navigation, journey planning on a road map, and multi-modal journey planning in a city. Similarities between such domains and pathfinding in games have encouraged, to some extent, the transfer of ideas across domains. Clearly, future common events, such as workshops at big conferences, would help tighten the connections.

4 Conclusion

Summarizing discussions held in the Pathfinding Workgroup at the Dagstuhl Seminar 12191, this chapter focused on recent advances, current limitations and possible future directions in pathfinding research. A comprehensive literature survey was beyond the purpose of our work. Instead, our survey was meant to set the grounds for identifying challenges and future work ideas.

It seems that research in pathfinding is not in a danger of fading. The recent first edition of the pathfinding competition suggests that the interest remains high even for the standard, well studied single-agent pathfinding on a fully known grid map. Yet, some diversification to the focus of future research might be desirable. For example, in problems such as multi-agent pathfinding, there seems to be a gap between academic research and commercial game development. The two communities could mutually benefit from each other if game developers make available descriptions of problem variants that are particularly relevant to a game, together with performance specs (speed, memory) expected from a solving algorithm.

References

1 Y. Björnsson, M. Enzenberger, R. Holte, and J. Schaeffer. Fringe Search: Beating A* at Pathfinding on Game Maps. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games CIG-05*, pages 125–132, 2005.

2 Yngvi Björnsson and Kári Halldórsson. Improved heuristics for optimal path-finding on game maps. In *Proceedings of the Conference on AI and Interactive Digital Entertainment AIIDE-06*, pages 9–14, 2006.

3 A. Botea, M. Müller, and J. Schaeffer. Near optimal hierarchical path-finding. *Journal of Game Development*, 1:7–28, 2004.

4 Adi Botea. Ultra-fast Optimal Pathfinding without Runtime Search. In *Proceedings of the Conference on AI and Interactive Digital Entertainment (AIIDE-11)*, pages 122–127, 2011.

5 Vadim Bulitko, Yngvi Björnsson, and Ramon Lawrence. Case-based subgoaling in real-time heuristic search for video game pathfinding. *Journal of Artificial Intelligence Research (JAIR)*, 39:269–300, 2010.

6 Tristan Cazenave. Optimizations of data structures, heuristics and algorithms for path-finding on maps. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games CIG-06*, pages 27–33, 2006.

7 Douglas Demyen and Michael Buro. Efficient triangulation-based pathfinding. In *Proc. of the 21st National Conference on Artificial intelligence*, AAAI-06, pages 942–947, 2006.

8 David Ferguson and Anthony (Tony) Stentz. Using Interpolation to Improve Path Planning: The Field D* Algorithm. *Journal of Field Robotics*, 23(2):79–101, February 2006.

9 A. V. Goldberg and C. Harrelson. Computing the Shortest Path: A* Search Meets Graph Theory. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms SODA-05*, pages 156–165, 2005.

10 M. Goldenberg, Ar. Felner, N. Sturtevant, and J. Schaeffer. Portal-based true-distance heuristics for path finding. In *Proceedings of the Symposium on Combinatorial Search SoCS-10*, pages 39–45, 2010.

11 Meir Goldenberg, Nathan Sturtevant, Ariel Felner, and Jonathan Schaeffer. The compressed differential heuristic. In *Proceedings of the National Conference on Artificial Intelligence AAAI-11*, pages 24–29, 2011.

12 Daniel Harabor and Adi Botea. Hierarchical Path Planning for Multi-Size Agents in Heterogeneous Environments. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games CIG-08*, pages 258–265, Perth, Australia, December 2008.

13 Daniel Harabor and Adi Botea. Breaking path symmetries in 4-connected grid maps. In *Proceedings of the Conference on AI and Interactive Digital Entertainment AIIDE-10*, pages 33–38, 2010.

14 Daniel Harabor, Adi Botea, and Phil Kilby. Path Symmetries in Undirected Uniform-cost Grids. In *In Proceedings of the Ninth Symposium on Abstraction, Reformulation, and Approximation (SARA-11)*, 2011.

15 Daniel Harabor and Alban Grastien. Online graph pruning for pathfinding on grid maps. In *Proceedings of the National Conference on Artificial Intelligence AAAI-11*, 2011.

16 P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on System Sciences and Cybernetics*, SSC-4(2):100–107, 1968.

17 R. Holte, M. Perez, R. Zimmer, and A. MacDonald. Hierarchical A*: Searching Abstraction Hierarchies Efficiently. Technical report, University of Ottawa, TR-95-18, 1995.

18 Marcelo Kallmann, Hanspeter Bieri, and Daniel Thalmann. Fully dynamic constrained delaunay triangulations. In G. Brunnett, B. Hamann, H. Mueller, and L. Linsen, editors, *Geometric Modelling for Scientific Visualization*, pages 241–257. Springer-Verlag, Heidelberg, Germany, first edition, 2003. ISBN 3-540-40116-4.

19 M.M. Khorshid, R.C. Holte, and N. Sturtevant. A polynomial-time algorithm for non-optimal multi-agent pathfinding. In *Proceedings of the Fourth Annual Symposium on Combinatorial Search (SoCS)*, pages 76–83, 2011.

20 R. Korf. Real-Time Heuristic Search. *Artificial Intelligence*, 42(2-3):189–211, 1990.

21 D. Kornhauser, G. Miller, and P. Spirakis. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *Proceedings of the 25th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 241–250, 1984.

22 Alex Kring, Alex J. Champandard, and Nick Samarin. DHPA* and SHPA*: Efficient Hierarchical Pathfinding in Dynamic and Static Game Worlds. In *Proceedings of the AI and Interactive Digital Entertainment Conference AIIDE-10*, pages 39–44, 2010.

23 Ramon Lawrence and Vadim Bulitko. Database-driven real-time heuristic search in video-game pathfinding. *IEEE Trans. Comput. Intellig. and AI in Games*, 5(3):227–241, 2013.

24 Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius. Artificial and Computational Intelligence in Games (Dagstuhl Seminar 12191). *Dagstuhl Reports*, 2(5):43–70, 2012.

25 Ryan Luna and Kostas E. Bekris. An efficient and complete approach for cooperative path-finding. In *Proc. of the National Conference on Artificial Intelligence AAAI-11*, 2011.

26 Gabriele Röger and Malte Helmert. Non-optimal multi-agent pathfinding is solved (since 1984). In *Proceedings of the Symposium on Combinatorial Search SoCS-12*, 2012.

27 Jagan Sankaranarayanan, Houman Alborzi, and Hanan Samet. Efficient query processing on spatial networks. In *ACM Workshop on Geographic Information Systems*, pages 200–209, 2005.

28 Guni Sharon, Roni Stern, Meir Goldenberg, and Ariel Felner. The increasing cost tree search for optimal multi-agent pathfinding. In *Proceedings of the International Joint Conference on Artificial Intelligence IJCAI-11*, pages 662–667, 2011.

29 D. Silver. Cooperative pathfinding. *AI Programming Wisdom*, 2006.

30 Trevor Scott Standley. Finding optimal solutions to cooperative pathfinding problems. In *Proceedings of the National Conference on Artificial Intelligence AAAI-10*, 2010.

31 Trevor Scott Standley and Richard E. Korf. Complete algorithms for cooperative pathfinding problems. In *Proceedings of the International Joint Conference on Artificial Intelligence IJCAI-11*, pages 668–673, 2011.

32 N. Sturtevant, A. Felner, M. Barer, J. Schaeffer, and N. Burch. Memory-based heuristics for explicit state spaces. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 609–614, 2009.

33 Nathan Sturtevant. Memory-efficient abstractions for pathfinding. *Proceedings of the International Conference on Artificial Intelligence and Interactive Digital Entertainment AIIDE-07*, pages 31–36, 2007.

34 Nathan Sturtevant and Michael Buro. Partial Pathfinding Using Map Abstraction and Refinement. In *Proceedings of the National Conference on Artificial Intelligence AAAI-05*, pages 47–52, 2005.

35 Nathan Sturtevant and Michael Buro. Improving collaborative pathfinding using map abstraction. In *Proceedings of the International Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pages 80–85, 2006.

36 W. van der Sterren. Path Look-Up Tables – Small is Beautiful. In S. Rabin, editor, *AI Game Programming Wisdom 2*, pages 115–129, 2003.

37 K.-H. C. Wang and A. Botea. Fast and Memory-Efficient Multi-Agent Pathfinding. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 380–387, 2008.

38 K.-H. C. Wang and A. Botea. MAPP: a Scalable Multi-Agent Path Planning Algorithm with Tractability and Completeness Guarantees. *Journal of Artificial Intelligence Research JAIR*, 42:55–90, 2011.

39 Peter Yap, Neil Burch, Robert C. Holte, and Jonathan Schaeffer. Block A*: Database-Driven Search with Applications in Any-Angle Path-Planning. In *Proceedings of the National Conference on Artificial Intelligence AAAI*, 2011.

Learning and Game AI

Héctor Muñoz-Avila[1], Christian Bauckhage[2], Michal Bida[3], Clare Bates Congdon[4], and Graham Kendall[5]

1 **Department of Computer Science, Lehigh University, USA**
`hem4@lehigh.edu`
2 **Fraunhofer-Institut für Intelligente Analyse und Informationssysteme IAIS, Germany**
`christian.bauckhage@iais.fraunhofer.de`
3 **Faculty of Mathematics and Physics, Charles University in Prague, Czech Republic**
`michal.bida@gmail.com`
4 **Department of Computer Science, The University of Southern Maine, USA**
`congdon@usm.maine.edu`
5 **School of Computer Science, University of Nottingham, UK and Malaysia**
`graham.kendall@nottingham.ac.uk`

Abstract

The incorporation of learning into commercial games can enrich the player experience, but may concern developers in terms of issues such as losing control of their game world. We explore a number of applied research and some fielded applications that point to the tremendous possibilities of machine learning research including game genres such as real-time strategy games, flight simulation games, car and motorcycle racing games, board games such as Go, an even traditional game-theoretic problems such as the prisoners dilemma. A common trait of these works is the potential of machine learning to reduce the burden of game developers. However a number of challenges exists that hinder the use of machine learning more broadly. We discuss some of these challenges while at the same time exploring opportunities for a wide use of machine learning in games.

1998 ACM Subject Classification I.2.m Artificial Intelligence, miscellaneous

Keywords and phrases Games, machine learning, artificial intelligence, computational intelligence

Digital Object Identifier 10.4230/DFU.Vol6.12191.33

1 Introduction

Machine learning seeks to improve the performance of a system (e.g., a computer player agent) on a given gaming task (e.g., defeat an opponent). Typically, machine learning algorithms can do this **online** or **offline**. The former takes place while playing the game while the latter takes place from data collected in previous games. For example, online learning enables game-playing algorithms to adapt to the current opponent strategy while offline learning enables eliciting common game-playing strategies across multiple games.

In this chapter, we explore learning aspects in current computer games, challenges, and opportunities for future applications. Our intention is to look at the broad issues of incorporating learning into games, independently of languages and platforms.

We begin our study by discussing research and applications of machine learning to game AI. We first provide a quick overview of a number of research and applications of machine

Artificial and Computational Intelligence in Games. *Dagstuhl Follow-Ups*, Volume 6, ISBN 978-3-939897-62-0.
Editors: Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius; pp. 33–43

Dagstuhl Publishing
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany

learning to games. Then we examine carefully the use of evolutionary computation in gaming tasks.

Next we examine a number of challenges that makes it difficult to apply machine leaning more broadly in games including (1) the need for algorithms to explain their decisions and gain the user's trust, and (2) some lingering issues such as difficulty of pointing to a specific solution and the need for gaming data, and (3) Making the game enjoyable for the player. The latter is a difficult yet crucial one. It is clear that we want to make games more enjoyable but it is unclear how we can formalize the notion of "fun" in machine understandable form.

Finally, we examine opportunities for machine learning applications which we believe are within reach of current techniques. These include (1) balancing gaming elements, (2) balancing game difficulty, and (3) finding loopholes in games.

2 Sample State-of-the-Art Applications and Research

Our discussion of the state of the art is divided into two parts: first we give a overview of a number of applications of machine leaning and in the second part we discuss in depth how evolutionary computation can be used to build sophisticated AI.

2.1 Machine Learning for Game AI

It would be difficult to give a complete overview of research and applications on machine learning for Game AI. We discuss some of these works to give the reader an overview of the topic.

There are a number of well-documented success stories such as the use of induction of decision trees in the commercial game Black and White [1]. There are a number of noncommercial applications of machine learning to game such as the use of reinforcement learning to play Backgammon [29]. The use of machine learning to find patterns from network and log data has demonstrated to be significant [8]. Also there is significant research, demonstrating the use of learning approaches such as evolutionary computation to evolve rules for high-performance for arcade games.

In [6], the authors used a Q-learning based algorithm to simulate dog training in an educational game. In [10], the authors used coevolution to evolve agents playing Capture-the-Coins game utilizing rtNeat and OpenNERO research platform. In [22], the authors used learned self-organizing map to improve maneuvering of team of units in real-time strategy game Glest. In [24], car racing track models are learned from sensory data in car racing simulator TORCS. In [14], the authors used cgNEAT to evolve content (weapons) in Galactic Arms Race game. In [20], tunsupervised learning tecniques are used to learn a player model from large corpus of data gathered from people playing the Restaurant game. In [31], evolutionary algorithms are used for automatic generation of tracks for racing games. In [21], a neural network learning method combined with a genetic algorithm is used to evolve competitive agent playing Xpilot game. In [19], the authors use a genetic algorithm to evolve AI players playing real time strategy game Lagoon. In [7], artificial neural networks are used to control motorbikes in Motocross The Force game. In [28], the authors used genetic algorithms to evolve a controller of RC racing car. In [26], the authors introduced the real-time neuro-evolution of augmenting topologies (rt-NEAT) method for evolving artificial neural networks in real time demonstrated on NERO game. In [32], the authors present evolution of controllers for a simulated RC car. In [9], parameters are evolved for bots playing first person shooter game Counter Strike. In [34], the authors used a genetic algorithm to optimize parameters for a simulation of a Formula One car in Formula One Challenge '99-'02

racing game. In [15], the authors reported on the impact of machine learning methods in iterated prisoner's dilemma. In [4], the authors used real-time learning for synthetic dog character to learn typical dog behaviors. In [17], genetic algorithm are used to develop neural networks to play the game of Go. In [35], the authors used neuro-evolution mechanisms to evolve agents for a modified version of Pac-Man game. In [23], tagents playing Quake 3 are evolved. In [12], authors provided a technical description of the game Creatures where they used neural networks for learning of the game characters. In [2], data from recorded human game play is usedd to train an agent playing first person shooter game Quake 2. In [30], the authors used pattern recognition and machine learning techniques with data from recorded human game play to learn an agent to move around in first person shooter game Quake 2. In [18], the authors report on using tabular $Sarsa(\lambda)$ RL algorithm for learning the behavior of characters in a purpose-built FPS game.

2.2 A Discussion of Evolutionary Computation Applications

We concentrate on evolutionary systems, which is reflective of the potential and some of the difficulties of fielding machine learning techniques in commercial games.

Evolutionary rule-based systems have been demonstrated to be a successful approach to developing agents that learn to play games, as in [25, 5, 11]. In this approach, the agent's behavior is dictated by a set of if-then rules, such as "if I see an opponent, and I have low health, then collect health". These rule sets are subject to evolutionary learning, which allows one to start with random behaviors and to have the evolutionary learning process conduct the search for individual rules that are strong as well as complete rule sets that are successful.

This approach has been used with good results on a variety of arcade and video games, including Unreal Tournament 2004, Mario, and Ms. Pac-Man, in the competition environments provided at IEEE conferences. Small and Congdon [25] demonstrated learning in the environment of the Unreal Tournament 2004 Deathmatch competition at IEEE Congress on Evolutionary Computation. In this competition setup, agents (bots) played head-to-head in this dynamic first person shooter. Bojarski and Congdon [5] demonstrated learning in the environment of the Mario AI Championship 2010 competition at the IEEE Computational Intelligence and Games conference (CIG). In this competition setup, the agent was allowed 10,000 times to play a novel level (providing time to learn the level) and was scored on its performance on the 10,001st play. Gagne and Congdon [11] demonstrated learning in the environment of the Ms. Pac-Man vs. Ghosts Competition for CIG 2012. In this competition setup, a simulated version of Ms. Pac-Man is used, allowing entrants to submit agents for the role of Ms. Pac-Man in the game or to submit a ghost team. The Gagne and Congdon work describes an evolutionary rule-based system that learns to play the ghost team role.

These approaches have in common the use of "Pittsburgh-style" rule sets, in which the individuals in the evolutionary computation population are each a rule set. (This is contrasted with "Michigan-style" rule sets, in which each individual in the evolutionary computation population is a single rule, and the entire population constitutes a rule set.) Learning occurs both through mutation of the conditions for the rules and via crossover, in which rules swap their conditions. While the learning takes time (e.g., a week or two), the agents learn to get better at playing their respective games.

Kadlec [16], uses evolutionary computation techniques for controlling a bot. These techniques are used to learn both strategic behavior (e.g., planning next steps such as which weapons to find) as well as reactive behavior (e.g., what to do under attack). The testbed uses was Unreal Tournament 2004. Unreal Tournament is a first-person shooter game (see Figure 1) where bots and human-controlled characters compete to achieve some objectives.

Figure 1 Snapshot of Unreal Tournament 2004 (courtesy of Rudolph Kadlec (2008)).

Kadlec's work uses genetic programming to evolve the strategic knowledge while neural networks was used to generate reactive behavior.

Experiments were performed in two kinds of games: death match (where participants increase their score by killing enemies) and capture the flag, where players increase their score by capturing enemy flags and bringing them to their home base. Figures 2 and 3 show sample behavior trees learned for the death match and capture the flag experiments respectively. The experiment demonstrated the feasibility to learn complex behavior but on the other hand the resulting behaviors were not as complex as the ones followed by humans or hand-coded bots.

Behavior trees consisted of three types of nodes - functional nodes that help to decide which behavior should be selected, behavior nodes that code specific type of behavior and sensory nodes that output the value of a specified sensor. Each behavior in a tree has its priority either computed by a function or fixed. The functional nodes then decide which behavior will be activated based on these priorities. In the trees there are two types of functional nodes - highest activation that simply selects the behavior with the highest activation and sequential arbiter that returns the first behavior if its activation is higher than 0.1 or when the activation of the second behavior is lower than 0.1, otherwise it returns the second behavior result.

Kadlec [16] also reports on a second experiment using the algorithm reported by Stanley and Miikkulainen [27]. This experiment demonstrated the capability to learn reactive behavior but again the resulting bots didn't exhibit human-like or hard-coded performance. While the results were interesting, there is still room for research on these techniques before they can be deployed.

Bauckhage et al. [3] used neural network architectures to learn behavior of bots in the first-person-shooter game Quake II. As an input for learning, they used post-processed network data of the game, coding the changes in the state from the game. This allowed them to use recordings of gameplay to train the neural networks. Authors were successful in learning basic moving and aiming behavior by this approach, proving it is possible to learn human-like behavior by analyzing the data gathered from actual human game play.

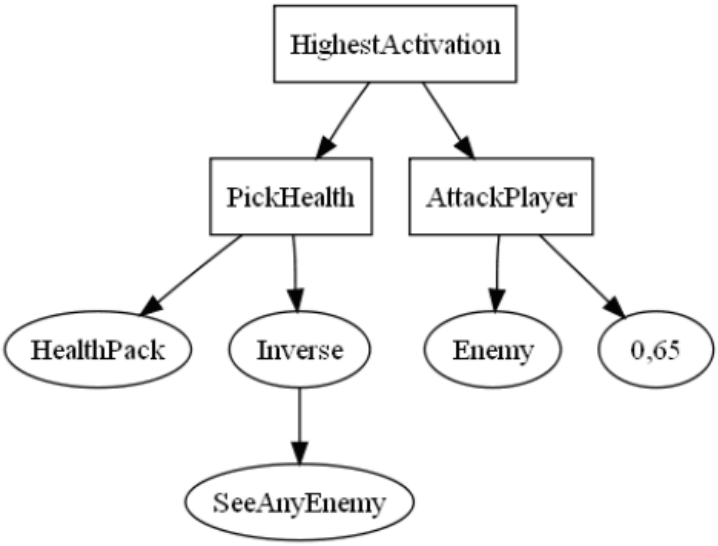

Figure 2 Behavior trees of best individuals for the DeathMatch experiment (courtesy of Rudolph Kadlec (2008)). The bot exhibits two types of behaviors. The bot attacks the enemy player (node AttackPlayer) with the priority fixed to 0.65 or it picks health packs (node PickHealth). The priority of PickHealth behavior is an inverse of the SeeAnyEnemy sensor. This means if the bot actually sees any enemy, the inverse function will inverse this sense to be false and the behavior priority will be 0, so the bot will select AttackPlayer behavior.

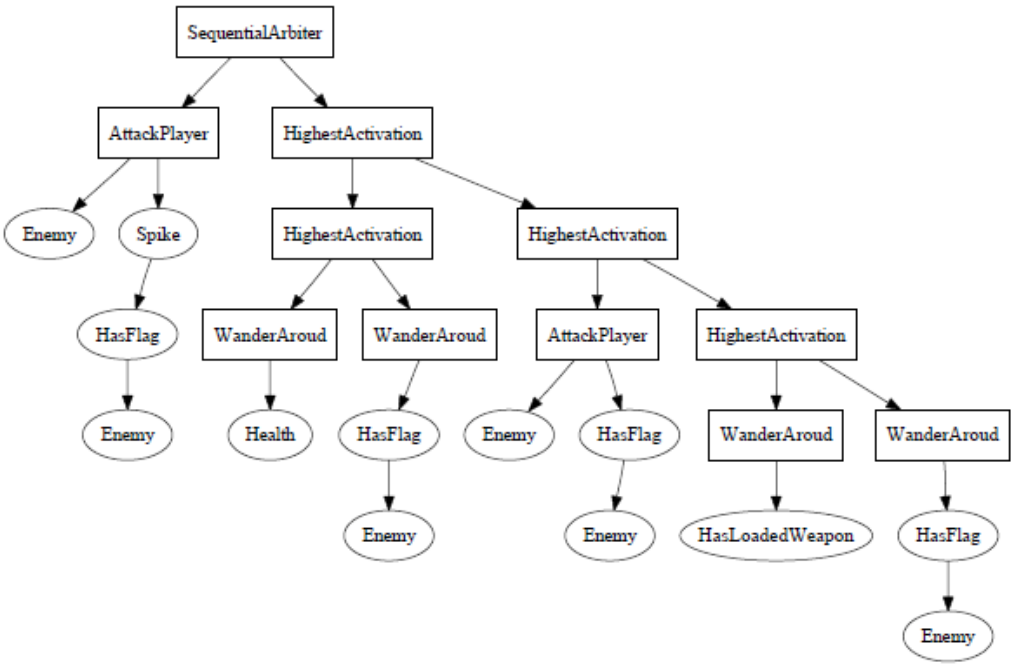

Figure 3 Behavior trees of best individuals for the Capture the Flag experiment (courtesy of Rudolph Kadlec (2008)). Although the tree looks complex the resulting behavior is simple: the bot wanders around (moves randomly around the map) if it does not see enemy flag carrier or shoots the enemy player if the player is holding the flag.

3 Challenges

We identify two challenges in adding learning to commercial games: The need to explain decisions and gaining user's trust and issues with using machine learning algorithms.

3.1 Need to Explain Decisions and Gaining User's Trust

It will be desirable for machine learning algorithms to explain their decisions. Unfortunately, it is often difficult to devise algorithms that are capable of automatically generating explanations that are meaningful for the user. In the context of computer games, explanations can help game developers understand the reasoning behind gaming decisions made by automated players and thereby correct potential flaws in the reasoning process. They also help understand capabilities and limitations in the decision making process and tune scenarios accordingly.

The lack of a capability to explain itself can lead to its decisions not being trusted, making it more difficult for machine learning techniques to be accepted. Game companies are often reluctant to add learning elements to their games for fear of "losing control" over the gameplay that might emerge with learning. Since there are many examples in machine learning of systems honing in on an exception or exploiting unanticipated by the developer of the system, this concern is not baseless. Examples of potentially problematic learning would include a non-player character (NPC) that is essential to the storyline learning something that destroys the storyline or an NPC that discovers an exploit, revealing it to human players. In the first case, the behaviors subject to learning would need to be controlled. In the second case, one can argue that human players will eventually discover exploits anyway, so this may be a non issue.

It's good to remember that the primary reason to add learning to a game would be that the learning could make the game more enjoyable for the player. One facet of this might be to relieve monotony, and another might be to adapt the play level to the player to make the game appropriately challenging (and possibly, to help the player learn to play the game). The issues of monotony might kick in with non-player characters (NPCs) having overly scripted actions; additionally, player enjoyment is often heightened when the player does not know for certain that they are playing against a bot. While adding an NPC that can adapt during gameplay has the potential to lessen predictability of the NPC, a concern is that an adaptive NPC could also learn ridiculous or malicious behaviors. However, if the facets of behavior that the agent is allowed to learn over are controlled, adaptivity has the potential to increase player enjoyment. For example, an NPC opponent that adapts its "skill level" to the human player, allowing just enough challenge. Whether opponent or teammate, an NPC with adaptive skill levels could in effect work with the player to help the player improve their own gameplay. The key to controlling the learning is to limit the facets of behavior that it is applied to. Part of the difficulty is that some of the learning mechanisms such as evolutionary computation (see Section 2) make intricate computations that are difficult to explain to the game developer. For example, the game developer might see that some of the resulting strategies are effective but might be puzzled by other strategies generated. Without even a grasp of how and why these techniques work, it will be difficult for the game developer to adopt such a technique. This is doubtless a significant challenge but one that if tackled can yield significant benefits.

3.2 Issues with Using Machine Learning Algorithms

Another challenge is that there is no simple answer if a game developer asks the question about which machine learning approach to use. Take for example, classification tasks, a subfield of machine learning where an agent must learn a function mapping input vectors to output classes. There is no such a thing as the best classifier algorithms; some classifier algorithms work well in some data sets but not in others. The sheer number of potential algorithms to use for classification tasks can be overwhelming for game practitioners that might be looking for a quick fix.

Another difficulty is obtaining the data for input to test the machine learning algorithms. There is no clear value added for a commercial company to gather and share the data. This is reminiscent of a "chicken and egg" problem, whereby the data is needed to demonstrate potential capabilities but without some proof that these techniques might work it is difficult to collect the data needed.

Part of the problem is that if a gaming company has money to invest in a game, aspects such as graphics will get prioritized simply because it is unclear what the benefit is from investing in machine learning.

Finally, some games are designed for the player to spend a certain number of hours. Having adaptable AI can make the game replayable for a long time and hence it might be undesirable for those kinds of games.

3.3 Reward versus Having Fun

One of the challenges of applying machine learning is the difficulty of eliciting adequate target functions. Machine learning algorithms frequently have an optimality criterion defined by a target function. For example, in reinforcement learning, an agent is trying to maximize the summation of its future rewards (this target function is called the return in reinforcement learning terminology). In an adversarial game, the reward can be defined as the difference in utility $U(s) - U(s')$ between the current state s and some previous state s'. One way to define the utility of an state is by computing $Score(ourTeam) - Score(opponent)$. Intuitively, by defining the return in this way, the reinforcement learning agent is trying to maximize the difference in score between its own team and its opponent.

While clearly having such a target is sensible in many situations such as a machine versus machine tournament, such target functions omit an crucial aspect in games: players want to have fun and this is not necessarily achieved by playing versus an opponent optimized to defeat them. "Having fun" is an intangible goal, difficult to formalize as a target function. This is undoubtedly one of the most interesting and challenging issues of applying machine learning to games.

4 Opportunities

We identify three opportunities for machine learning techniques including:

4.1 Balancing Gaming Elements

Many games have different elements such as factions in a real-time strategy games (e.g., humans versus orcs) or classes in a role-playing game (e.g., mages versus warriors). Machine learning could help with balancing these elements. One example of games that could benefit from machine learning techniques is collectible card games with the most notable example Magic: The Gathering. These games often feature a complex set of rules and thousands of

different cards with different abilities that are then used by players in their strategies. The goal of the developers of these games is to balance the gaming elements so no particular strategy will work in all cases. Some of these developers released online versions of their games (e.g. Magic: The Gathering) that omit the need of the players to own real cards, moving everything to virtual world. These online versions of games could provide the developers with invaluable statistics of a) the trends in the game - e.g. which strategy is used the most, b) strength of the card, e.g. when the player plays "the blue Viking" in the second turn he has 60% probability to win the game or c) general patterns in player strategies that could then be used to train competitive AI for these games. This would help the developers to improve the game in terms of gameplay and potentially make the game more desirable for players. While points a) and b) are more data mining and statistics processing, the point c) could benefit from one of the machine learning algorithms that are currently available.

4.2 Balancing Game Difficulty

In games such as those that are open-ended such as massive multiplayer online (MMO) games, a difficulty is how to tailor the game simultaneously towards dedicated players (e.g., players who play 20+ hours per week) and casual players (e.g., players who play 10 hours or less a week).

An important potential for adding learning to games is in adjusting the game difficulty to the player. Players will lose interest in a game that is markedly too easy or too hard for them, so the ability for an element of the game to adapt to the player will reasonably increase player engagement. Additionally, the same mechanism would allow a game to be enjoyed by different family members, with different profiles and histories for each. Furthermore, an adaptive approach to game difficulty includes the potential to help develop player skills, allowing a player an extended enjoyment of the game.

4.3 Finding Design Loopholes in Games

Pattern recognition techniques can be used to detect common patterns in game logs and then use these patterns to detect outliers. Such techniques will enable developers to detect anomalies (e.g. exploits in MMOs) much faster than it is currently done, which is done manually for the most part. MMORPG games often feature complex worlds with rich sets of rules. In these worlds it is often hard to predict general trends in the means of player strategies or economy prior to launch of the game. More often than not, these kinds of games need tweaking and balancing after launch preventing the exploitation of features not intended by game developers. These problems are often detected "by hand", by honest players reporting the issue, or by dedicated game developers, who monitor the game and check for these kinds of exploits. However, these processes could be partially automated by applying a) simulation, b) data mining and c) machine learning algorithms. For example the algorithm would gather data such as "gold gain per hour per level" for all of the players. Then all the players that would exceed certain threshold over average value would be tagged as suspicious and developers would be notified to further check the issue. This approach can be extended to almost any feature of the game, such as quest completion, difficulty of the enemies, etc. Moreover, methods of auto-correction by machine learning methods could be applied, e.g. I see that this player defeats that enemy every time, but this is not supposed to be, so we increase the difficulty of this particular enemy for this particular player.

4.4 Making Timely Decisions

One of the most difficult challenges of applying AI to games is twofold. First, that Game AI is typically allocated comparatively little CPU time. Most CPU time is devoted to other processes such as pathfinding or maintaining consistency between the GUI and the internal state of the game. Second, the time for developing the game AI is comparatively short; other software development tasks such as graphics and level design take precedence. This makes it very difficult to design and run a deep Game AI. As a result frequently game AI is generally not as good as it can be [13].

Machine learning offers the possibility to learn and tune capable Game AI by analyzing logs of game traces (e.g., player versus player games during beta testing). Indeed in Section 2.1, we discussed some of systems. For example, [29] reports on a system capable of eliciting game playing strategies that were considered novel and highly competent by human experts in a board game. [33] reports on a learning system that controls a small squad of bots in an FPS game and rapidly adapts to opponent team's strategy. In these and other such learning systems, the resulting control mechanism is quite simple: it basically indicates for every state the best action(s) that should be taken. Yet because it captures knowledge from many gameplay sessions it can be very effective.

5 Conclusions

In this work, we have explored the state of the art in machine learning research and challenges and opportunities in applying machine learning to commercial games. For the state of the art we have explored research on evolutionary computation, as an example of a machine learning technique that shows a lot of promise while at the same time discussing limitations. We explored three basic challenges: (1) lack of explanation capabilities which contribute to a lack of trust on the results of the machine learning algorithms, (2) other issues with machine learning such the difficulty of getting the data needed because of perceived cost-benefit tradeoffs, and (3) modeling "fun" in machine learning target functions. Finally, we explored opportunities for machine learning techniques including using machine learning techniques for (1) balancing game elements, (2) balancing game difficulty, (3) finding design loopholes in the game, and (4) making timely decisions.

Acknowledgments. This chapter presents an extension of the discussions that a group of researchers had on the topic of machine learning and games at the Dagstuhl seminar on the topic Artificial and Computational Intelligence in Games that took place in May 2012. This work is funded in part by National Science Foundation grants number 1217888 and 0642882.

References

1 Jonty Barnes and Jason Hutchens. *AI Programming Wisdom*, chapter Testing Undefined Behavior as a Result of Learning, pages 615–623. Charles Media, 2002.

2 Christian Bauckhage, Christian Thurau, and Gerhard Sagerer. Learning human-like opponent behavior for interactive computer games. *Pattern Recognition*, LNCS 2781:148–155, 2003.

3 Christian Bauckhage, Christian Thurau, and Gerhard Sagerer. Learning human-like opponent behavior for interactive computer games. *Pattern Recognition*, pages 148–155, 2003.

4 Bruce Blumberg, Marc Downie, Yuri Ivanov, Matt Berlin, Michael Patrick Johnson, and Bill Tomlinson. Integrated learning for interactive synthetic characters. *Proceedings of the*

29th annual conference on Computer graphics and interactive techniques – SIGGRAPH '02, 21(3):417–426, 2002.

5 Slawomir Bojarski and Clare Bates Congdon. REALM: A rule-based evolutionary computation agent that learns to play mario. In *IEEE Computational Intelligence and Games*, pages 83–90, 2010.

6 C Brom, M Preuss, and D Klement. Are educational computer micro-games engaging and effective for knowledge acquisition at high-schools? A quasi-experimental study. *Computers & Education*, 57(3):1971–1988, 2011.

7 Benoit Chaperot and Colin Fyfe. Improving Artificial Intelligence In a Motocross Game. *2006 IEEE Symposium on Computational Intelligence and Games*, pages 181–186, May 2006.

8 Gifford Cheung and Jeff Huang. Starcraft from the stands: Understanding the game spectator. In *CHI*, 2011.

9 Nicholas Cole, Sushil J. Louis, and Chris Miles. Using a genetic algorithm to tune first-person shooter bots. *Proceedings of the 2004 Congress on Evolutionary Computation (CEC 2004)*, 1:139–145, 2004.

10 Adam Dziuk and Risto Miikkulainen. Creating intelligent agents through shaping of co-evolution. *IEEE Congress on Evolutionary Computation (CEC 2011)*, pages 1077–1083, 2011.

11 David J. Gagne and Clare Bates Congdon. Fright: A flexible rule-based intelligent ghost team for ms. pac-man. In *IEEE Computational Intelligence and Games*, 2012.

12 Stephen Grand, Dave Cliff, and A Malhotra. Creatures: Artificial life autonomous software agents for home entertainment. *The First International Conference on Autonomous Agents (Agents '97)*, pages 22–29, 1997.

13 Stephen Grand, Dave Cliff, and A Malhotra. A gamut of games. *AI Magazine*, 2001.

14 Erin J. Hastings, Ratan K. Guha, and Kenneth O. Stanley. Evolving content in the Galactic Arms Race video game. *2009 IEEE Symposium on Computational Intelligence and Games*, pages 241–248, September 2009.

15 Philip Hingston and Graham Kendall. Learning versus evolution in iterated prisoner's dilemma. *Evolutionary Computation, 2004. CEC2004.*, 1:364–372, 2004.

16 Rudolf Kadlec. Evolution of intelligent agent behavior in computer games. Masters thesis, Charels University in Prague, Czech Republic, 2008.

17 George Konidaris, D Shell, and N Oren. Evolving neural networks for the capture game. *Proceedings of the SAICSIT 2002 Post-graduate Research Symposium*, 2002.

18 Michelle McPartland and Marcus Gallagher. Reinforcement learning in first person shooter games. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(1):43–56, Mar 2011.

19 Chris Miles, Juan Quiroz, Ryan Leigh, and Sushil J. Louis. Co-Evolving Influence Map Tree Based Strategy Game Players. *2007 IEEE Symposium on Computational Intelligence and Games*, pages 88–95, 2007.

20 Jeff Orkin and Deb Roy. The restaurant game: Learning social behavior and language from thousands of players online. *Journal of Game Development*, 3:39–60, 2007.

21 Matt Parker and Gary B. Parker. The Evolution of Multi-Layer Neural Networks for the Control of Xpilot Agents. *2007 IEEE Symposium on Computational Intelligence and Games*, pages 232–237, 2007.

22 Mike Preuss, Nicola Beume, Holger Danielsiek, Tobias Hein, Boris Naujoks, Nico Piatkowski, Raphael Stuer, Andreas Thom, and Simon Wessing. Towards intelligent team composition and maneuvering in real-time strategy games. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(2):82–98, 2010.

23 Steffen Priesterjahn, Oliver Kramer, Alexander Weimer, and Andreas Goebels. Evolution of human-competitive agents in modern computer games. *Evolutionary Computation, 2006. CEC 2006*, pages 777–784, 2006.

24 Jan Quadflieg, Mike Preuss, Oliver Kramer, and Gunter Rudolph. Learning the track and planning ahead in a car racing controller. *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, pages 395–402, August 2010.

25 Ryan K. Small and Clare Bates Congdon. Agent smith: Towards an evolutionary rule-based agent for interactive dynamic games. In *IEEE Congress on Evolutionary Computation*, pages 660–666, 2009.

26 Kenneth O. Stanley, Bobby D. Bryant, and Risto Miikkulainen. Evolving neural network agents in the NERO video game. *Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games (CIG'05)*, pages 182–189, 2005.

27 Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

28 Ivan Tanev, Michal Joachimczak, and Katsunori Shimohara. Evolution of driving agent, remotely operating a scale model of a car with obstacle avoidance capabilities. *Proceedings of the 8th annual conference on Genetic and evolutionary computation – GECCO '06*, pages 1785–1792, 2006.

29 Gerald Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68, March 1995.

30 Christian Thurau, Christian Bauckhage, and Gerhard Sagerer. Synthesizing movements for computer game characters. *Pattern Recognition*, LNCS 3175:179–186, 2004.

31 Julian Togelius, Renzo De Nardi, and Simon M. Lucas. Towards automatic personalised content creation for racing games. *2007 IEEE Symposium on Computational Intelligence and Games*, pages 252–259, 2007.

32 Julian Togelius and Simon M. Lucas. Evolving Controllers for Simulated Car Racing. *Proceedings of the 2005 Congress on Evolutionary Computation (CEC 2005)*, 2:1906–1913, 2005.

33 Lee-Urban S. Vasta, M. and H. Munoz-Avila. Retaliate: Learning winning policies in first-person shooter games. In *Proceedings of the Seventeenth Innovative Applications of Artificial Intelligence Conference (IAAI-07). AAAI Press.*, 2007.

34 Krzysztof Wloch and Peter J. Bentley. Optimising the performance of a formula one car using a genetic algorithm. *Parallel Problem Solving from Nature-PPSN VIII*, pages 702–711, 2004.

35 GN Yannakakis and J Hallam. Evolving opponents for interesting interactive computer games. *From Animals to Animats*, 8:499–508, 2004.

Player Modeling

Georgios N. Yannakakis[1], Pieter Spronck[2], Daniele Loiacono[3], and Elisabeth André[4]

1 **Institute of Digital Games, University of Malta, Msida, Malta**
`georgios.yannakakis@um.edu.mt`
2 **Tilburg Center of Cognition and Communication, University of Tilburg, The Netherlands**
`p.spronck@uvt.nl`
3 **Politecnico di Milano, Italy**
`daniele.loiacono@polimi.it`
4 **University of Augsburg, Germany**
`andre@informatik.uni-augsburg.de`

Abstract

Player modeling is the study of computational models of players in games. This includes the detection, modeling, prediction and expression of human player characteristics which are manifested through cognitive, affective and behavioral patterns. This chapter introduces a holistic view of player modeling and provides a high level taxonomy and discussion of the key components of a player's model. The discussion focuses on a taxonomy of approaches for constructing a player model, the available types of data for the model's input and a proposed classification for the model's output. The chapter provides also a brief overview of some promising applications and a discussion of the key challenges player modeling is currently facing which are linked to the input, the output and the computational model.

1998 ACM Subject Classification I.2.1 Applications and Expert Systems: Games

Keywords and phrases User modeling, Computer Games, Computational and Artificial Intelligence, Affective Computing,

Digital Object Identifier 10.4230/DFU.Vol6.12191.45

1 Introduction

Digital games are dynamic, *ergodic* media (i.e., a user interacts with and alters the state of the medium). They are designed to be highly engaging and embed rich forms of user interactivity. Collectively, the human-computer interaction (HCI) attributes of digital games allow for high levels of player incorporation [9]. As such, they yield dynamic and complex emotion manifestations which cannot be captured trivially byf standard methods in affective computing or cognitive modeling research. The high potential that games have in affecting players is mainly due to their ability of placing the player in a continuous mode of interaction, which develops complex cognitive, affective and behavioral responses. The study of the player in games may not only contribute to the design of improved forms of HCI, but also advance our knowledge of human experiences.

Every game features at least one user (i.e., the player), who controls an avatar or a group of miniature entities in a virtual/simulated environment [9]. Control may vary from the relatively simple (e.g., limited to movement in an orthogonal grid) to the highly complex (e.g., having to decide several times per second between hundreds of different possibilities in a highly complex 3D world). The interaction between the player and the game context is

Artificial and Computational Intelligence in Games. *Dagstuhl Follow-Ups*, Volume 6, ISBN 978-3-939897-62-0.
Editors: Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius; pp. 45–59

Dagstuhl Publishing
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany

of prime importance to modern game development, as it breeds unique stimuli which yield emotional manifestations to the player. Successful games manage to elicit these emotions in a manner that is appreciated by the player, and which form the main reason that the player is willing to engage in the game [50].

The primary goal of player modeling and player experience research is to understand how the interaction with a game is experienced by individual players. Thus, while games can be utilized as an arena for eliciting, evaluating, expressing and even synthesizing experience, we argue that one of the main aims of the study of players in games is the understanding of players' cognitive, affective and behavioral patterns. Indeed, by the nature of what constitutes a game, one cannot dissociate games from player experience.

This chapter focuses on experience aspects that can be detected from, modeled from, and expressed in games with human players. We explicitly exclude the modeling of non-player characters (NPCs), as in our view, player modeling involves a *human player* in the modeling process. We also make a distinction between *player modeling* [10, 26] and *player profiling*. The former refers to modeling complex *dynamic* phenomena during gameplay interaction, whereas the latter refers to the categorization of players based on *static* information that does not alter during gameplay — that includes personality, cultural background, gender and age. We will mainly focus on the first, but will not ignore the second, as the availability of a good player profile may contribute to the construction of reliable player models.

In summary, player modeling — as we define it in this chapter — is the study of computational means for the modeling of player cognitive, behavioral, and affective states which are based on data (or theories) derived from the interaction of a human player with a game [78]. Player models are built on *dynamic* information obtained during game-player interaction, but they could also rely on *static* player profiling information. Unlike earlier studies focusing on taxonomies of *behavioral* player modeling — e.g., via a number of different dimensions [58] or direct/indirect measurements [56] — we view player modeling in a holistic manner including cognitive, affective, personality and demographic aspects of the player. Moreover, we exclude approaches that are not directly based on human-generated data or not based on empirically-evaluated theories of player experience, human cognition, affect or behavior. The chapter does not intend to provide an inclusive review of player modeling studies under the above definition, but rather a high-level taxonomy that explores the possibilities with respect to the modeling approach, the model's input and the model's output.

The rest of this chapter provides a taxonomy and discussion of the core components of a player model which are depicted in Figure 1. That includes the computational model itself and methods to derive it as well as the model's input and output. The chapter illustrates a few promising applications of player modeling and ends with a discussion on open research questions for future research in this field.

2 Computational Model

Player modeling is, primarily, the study and use of artificial and computational intelligence (AI and CI) techniques for the construction of computational models of player behavior, cognition and emotion, as well as other aspects beyond their interaction with a game (such as their personality and cultural background). Player modeling places an AI umbrella to the multidisciplinary intersection of the fields of user (player) modeling, affective computing, experimental psychology and human-computer interaction.

One can detect behavioral, emotional or cognitive aspects of either a human player or

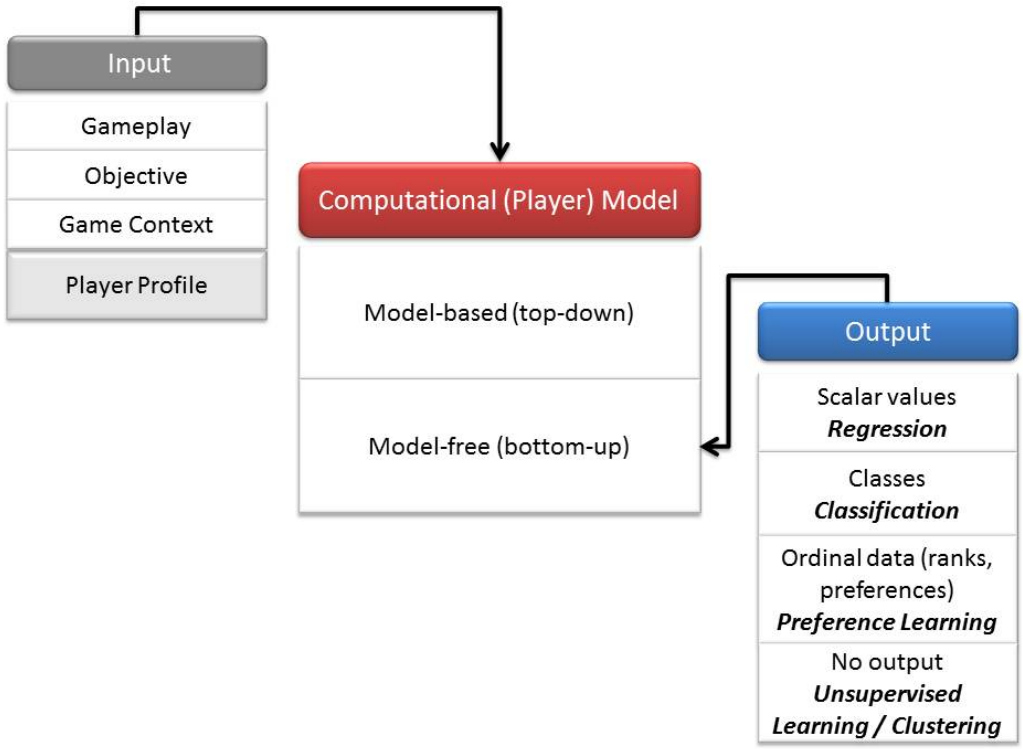

Figure 1 Player Modeling: the core components.

a non-player character (NPC). In principle, there is no need to model an NPC, for two reasons: (1) an NPC is coded, therefore a perfect model for it exists in the game's code, and is known by the game's developers; and (2) one can hardly say that an NPC possesses actual emotions or cognition. However, NPC modeling can be a useful testbed for player modeling techniques, by comparing the model discovered with the actual coded one. More interestingly, it can be an integral component of adaptive AI that changes its behavior in response to the dynamics of the opponents [6]. Nevertheless, while the challenges faced in modeling NPCs are substantial, the issues raised from the modeling of human players define a far more complex and important problem for the understanding of player experience.

By clustering the available approaches for player modeling, we are faced with either *model-based* or *model-free* approaches [80] as well as potential hybrids between them. The remaining of this section presents the key elements of both model-based and model-free approaches.

2.1 Model-based (Top-down) Approaches

According to a *model-based* or top-down [80] approach a player model is built on a theoretical framework. As such, researchers follow the modus operandi of the humanities and social sciences, which hypothesize models to explain phenomena, usually followed by an empirical phase in which they experimentally determine to what extent the hypothesized models fit observations.

Top-down approaches to player modeling may refer to emotional models derived from emotion theories (e.g., cognitive appraisal theory [21]). Three examples are: (1) the emotional dimensions of arousal and valence [19], (2) Frome's comprehensive model of emotional

response to a single-player game [22], and (3) Russell's circumplex model of affect [51], in which emotional manifestations are mapped directly to specific emotional states (e.g., an increased heart rate of a player may correspond to high arousal and therefore to excitement). Model-based approaches can also be inspired by a general theoretical framework of behavioral analysis and/or cognitive modeling such as usability theory [30], the belief-desire-intention (BDI) model, the cognitive theory by Ortony, Clore, & Collins [47], Skinner's model [57], or Scherer's theory [53]. Moreover, theories about user affect exist that are driven by game design, such as Malone's design components for 'fun' games [40] and Koster's theory of 'fun' [35], as well as game-specific interpretations of Csikszentmihalyi's concept of Flow [14]. Finally, several top-down difficulty and challenge measures [2, 28, 45, 59, 60, 70, 75] have been proposed for different game genres as components of a player model. In all of these studies, difficulty adjustment is performed based on a player model that implies a direct link between challenge and 'fun'.

Note, however, that even though the literature is rich in theories on emotion, caution is warranted with the application of such theories to games (and game players), as most of them have not been derived from or tested on ergodic media such as games. Calleja [9], for instance, reflects on the inappropriateness of the concepts of 'flow', 'fun' and 'magic circle' (among others) for games.

2.2 Model-free (Bottom-up) Approaches

Model-free approaches refer to the construction of an unknown mapping (model) between (player) input and a player state representation. As such, model-free approaches follow the modus operandi of the exact sciences, in which observations are collected and analyzed to generate models without a strong initial assumption on what the model looks like or even what it captures. Player data and annotated player states are collected and used to derive the model. Classification, regression and preference learning techniques adopted from machine learning or statistical approaches are commonly used for the construction of a computational model. Data clustering is applied in cases where player states are not available.

In model-free approaches we meet attempts to model and predict player actions and intentions [64, 65, 76] as well as *game data mining* efforts to identify different behavioral playing patterns within a game [15, 43, 61, 62, 72]. Model-free approaches are common for facial expression and head pose recognition since subjects are asked to annotate facial (or head pose) images of users with particular affective states (see [55] among others) in a crowd-sourcing fashion. The approach is also common in studies of psychophysiology in games (see [68, 79] among others).

2.3 Hybrid Approaches

The space between a completely model-based and a completely model-free approach can be viewed as a continuum along which any player modeling approach might be placed. While a completely model-based approach relies solely on a theoretical framework that maps a player's responses to game stimuli, a completely model-free approach assumes there is an unknown function between modalities of user input and player states that a machine learner (or a statistical model) may discover, but does not assume anything about the structure of this function. Relative to these extremes, the vast majority of the existing works on player modeling may be viewed as hybrids between the two ends of the spectrum, containing elements of both approaches.

3 Input

The model's input can be of three main types: (1) anything that a human player is doing in a game environment gathered from *gameplay* data (i.e., behavioral data); (2) *objective* data collected as bodily responses to game stimuli such as physiology and body movements; and (3) the *game context* which comprises of any player-agent interactions but also any type of game content viewed, played, and/or created. The three input types are detailed in the remainder of this section. We also discuss static information on the player (such as personality and gender) that could feed a player model.

3.1 Gameplay Input

The main assumption behind the use of behavioral (gameplay-based) player input is that player actions and real-time preferences are linked to player experience as games may affect the player's cognitive processing patterns and cognitive focus. In the same vein, cognitive processes may influence player experience. One may infer the player's present experience by analyzing patterns of his interaction with the game, and by associating his emotions with context variables [12, 24]. Any element derived from the interaction between the player and the game forms the basis for gameplay-based player modeling. This includes detailed attributes of the player's behavior (i.e., game metrics) derived from responses to system elements (i.e., NPCs, game levels, or embodied conversational agents). Game metrics are statistical spatio-temporal features of game interaction [17]. Such data is usually mapped to levels of cognitive states such as attention, challenge and engagement [12]. In addition, both general measures (such as performance and time spent on a task) and game-specific measures (such as the weapons selected in a shooter game — e.g., in [25]) are relevant.

A major problem with interpreting such behavioral player input is that the actual player experience is only indirectly observed by measuring game metrics. For instance, a player who has little interaction with a game might be thoughtful and captivated, or just bored and busy doing something else. Gameplay metrics can only be used to approach the likelihood of the presence of certain player experiences. Such statistics may hold for player populations, but may provide little information for individual players. Therefore, when one attempts to use pure gameplay metrics to make estimates on player experiences and make the game respond in an appropriate manner to these perceived experiences, it is advisable to keep track of the feedback of the player to the game responses, and adapt when the feedback indicates that the player experience was gauged incorrectly.

3.2 Objective Input

Games can elicit player emotional responses which, in turn, may affect changes in the player's physiology, reflect on the player's facial expression, posture and speech, and alter the player's attention and focus level. Monitoring such bodily alterations may assist in recognizing and synthesizing the player's model. As such, the objective approach to player modeling incorporates access to multiple modalities of player input.

Within objective player modeling, a number of real-time recordings of the player may be investigated. Several studies have explored the interplay between physiology and gameplay by investigating the impact of different gameplay stimuli to dissimilar physiological signals. Such signals are usually obtained through electrocardiography (ECG) [79], photoplethysmography [68, 79], galvanic skin response (GSR) [41, 49], respiration [68], electroencephalography (EEG) [44], electromyography (EMG) and pupillometry [7]. In addition to physiology one

may track the player's bodily expressions (motion tracking) at different levels of detail and infer the real-time affective responses from the gameplay stimuli. The core assumption of such input modalities is that particular bodily expressions are linked to basic emotions and cognitive processes. Motion tracking may include body posture [52] and head pose [55], as well as gaze [4] and facial expression [48].

While such objective multimodal measurements are usually more meaningful than the gameplay inputs discussed in the previous subsection, a major problem with most of them is that they are viewed by the player as invasive, thus affecting the player's experience with the game. This affect should be taken into account when interpreting measurements.

3.3 Game Context Input

In addition to both gameplay and objective input, the game's context is a necessary input for player modeling. Game context refers to the real-time parameterized state of the game. Player states are always linked to game context; a player model that does not take context into account runs a high risk of inferring erroneous states for the player. For example, an increase in galvanic skin response (GSR) can be linked to a set of dissimilar high-arousal affective states such as frustration and excitement; thus, the cause of the GSR increase (e.g., a player's death or level completion) needs to be fused within the GSR signal and embedded in the model.

3.4 Player Profile Information

Differences between players lead to different playing styles and preferences. Player profile information includes all the information about the player which is static and it is not directly (nor necessarily) linked to gameplay. This may include information on player personality (such as expressed by the Five Factor Model of personality [13]), culture dependent variables, and general demographics such as gender and age. Such information may be used as static model input and could lead to the construction of more accurate player models.

A typical way of employing player profile information is the stereotype approach [34]. The approach attempts to (automatically) assign a player to a previously defined subgroup of the population, for which key characteristics have been defined. Each subgroup is represented by a stereotype. After identifying to which subgroup the player belongs, the game can choose responses appropriate for the corresponding stereotype. This approach has been used by Yannakakis & Hallam [73] and by Thue [63]. They define stereotypes in terms of a player's gaming profile, rather than the gamer's characteristics outside the realm of gaming.

Van Lankveld et al [69, 71] look beyond pure gaming behavior, attempting to express a player's personality profile in terms of the Five Factor Model. While they achieve some success in modeling players in terms of the five factors, they notice that gameplay behavior not necessarily corresponds to "real life" behavior, e.g., an introverted person may very well exhibit in-game behavior that would typically be assigned to an extroverted person. Therefore, we can recognize two caveats which should be taken into account when aiming to use a real-life personality profile of a player to construct a model of the player inside a game: (1) the player's behavior inside the game not necessarily corresponds to his real-life behavior and vice versa; and (2) a player's real-life profile not necessarily indicates what he appreciates in a game.

4 Output

The model's output is usually a set of particular *player states*. Such states can be represented as a class, a scalar (or a vector of numbers) that maps to a player state — such as the emotional dimensions of arousal and valence or a behavioral pattern — or a relative strength (preference). The output of the model is provided through an annotation process which can either be driven by self-reports or by reports expressed indirectly by experts or external observers [80]. However, there are instances where reports on player states are not available; output then must be generated by unsupervised learning (see [15, 17] among others).

The most direct way to annotate a player state is to ask the players themselves about their playing experience and build a model based on these annotations. Subjective player state annotations can be based on either a player's free-response during play or on forced data retrieved through questionnaires. Free-response naturally contains richer information about the player's state, but it is often unstructured, even chaotic, and thus hard to analyze appropriately. Forcing players to self-report their experiences using directed questions, on the other hand, constrains them to specific questionnaire items which could vary from simple tick boxes to multiple choice items. Both the questions and the answers provided may vary from single words to sentences. Questionnaires can either involve elements of the player experience (e.g., the Game Experience Questionnaire [29]), or demographic data and/or personality traits (e.g., a validated psychological profiling questionnaire such as the NEO-PI-R [13]).

Alternatively, experts or external observers may annotate the player's experiences in a similar fashion. Third-person annotation entails the identification of particular player states (given in various types of representation as we will see below) by player experiences and game design experts (or crowd-sourced via non-experts). The annotation is usually based on the triangulation of multiple modalities of player and game input, such as the player's head pose, in-game behavior and game context [55]. Broadly-accepted emotional maps such as the Facial Action Coding System [18] provide common guidelines for third-person emotion annotation.

Three types of annotations (either forced self-reports or third-person reports) can be distinguished. The first is the *rating*-based format [41], which labels player experience states with a scalar value or a vector of values (found, for instance, in the Game Experience Questionnaire [29]). The second is the *class*-based format, which asks subjects to pick a user state from a particular representation which could vary from a simple boolean question (*was that game level frustrating or not? is this a sad facial expression?*) to a user state selection from, for instance, the Geneva Emotion Wheel [54]. The third is the *preference*-based format, which asks subjects to compare an experience in two or more variants/sessions of the game [77] (*was that level more engaging that this level? which facial expression looks happier?*). A recent comparative study has exposed the limitations of rating approaches over ranking questionnaire schemes (e.g., pairwise preference) which include increased order of play and inconsistency effects [74].

Beyond annotated player states or player profiles (such as personality traits), player models may be constructed to predict attributes of gameplay (e.g., in [39, 65] among others) or objective manifestations of the experience [3].

5 Applications

In this section we identify and briefly illustrate a few promising, and certainly non inclusive, applications of player modeling to game design and development, that range from adapting the challenge during the game to personalizing the purchasing model in free-to-play games.

5.1 Adaptive Player Experience and Game Balancing

As already mentioned in section 2.1 there exist several theoretical frameworks investigating the relationship between experience and game interaction that have been either built with primarily games in mind (e.g. the player immersion model of Calleja [9]; the theory of 'fun' of Koster [35]) or derived from other fields (e.g. psychology) and domains and tailored to games (e.g. the theory of flow [14] adopted for games [11]). Essentially what unifies all these theories is that ultimate player experience is achieved when elements of the game (mechanics, storyline, challenges) are in some sort of right balance with the general skills of the player. A common, but rather simplistic, approach to balance between game challenges and different player skills in order to match a broader range of players consists of providing a small, predefined set of difficulty levels which the player can pick from. A more sophisticated approach consists of adapting the game's challenge in response to the actions of the players and to the state of the game environment; relevant examples of this approach are the *AI director* [8] of *Left 4 Dead* (Valve, 2008) and the *race script* [23] framework used in *Pure* (Black Rock Studio, 2008).

Most of the game balancing approaches currently used rely on simple in-game statistics to estimate the state of the players and make strong assumptions on what kind of experience the players are looking for in the game (e.g., the basic assumption behind the AI director in *Left 4 Dead* is that players enjoy dramatic and unpredictable changes of pace). While, in general, such assumptions hold for a large number of players (as the designers usually have a good idea of their players), they are not universally applicable and may actually exclude large groups of potential players that would be willing to play a game if it would offer an experience more to their liking. Reliable player modeling techniques have the potential to adapt the challenge level of a game (and other gameplay elements) in a manner that suits each individual player [75, 1].

5.2 Personalized Game Content Generation

Procedural content generation (PCG) aims to deliver a large amount of game content algorithmically with limited memory resources. Nowadays PCG is mainly used to cut the development cost and, at the same time, to face the increasing expectations of players in terms of game content. Recently, the problem of automating the generation of high-quality game content has attracted considerable interest in the research community (see [78] and the corresponding chapter in this volume). In particular, research showed that search algorithms can be combined successfully with procedural content generation to discover novel and enjoyable game content [38, 66, 67]. Accordingly, the automation of content creation offers an opportunity towards realizing player model-driven procedural content generation in games [80]. The coupling of player modeling with PCG approaches may lead to the automatic generation of personalized game content.

Player models may also inform the generation of computational narrative (viewed here as a type of game content [80]). Predictive models of playing behavior, cognition and affect can drive the generation of individualised scenarios in a game. Examples of the coupling between player modeling and interactive narrative include the affect-driven narrative systems met in *Façade* [42] and *FearNot!* [5], the emotion-driven narrative building system in *Storybricks* (Namaste Entertainment, 2012), and the affect-centred game narratives such as the one of *Final Fantasy VII* (Square Product, 1997).

5.2.1 Towards Believable Agents

Human player models can inform and update believable agent architectures. Behavioral, affective and cognitive aspects of human gameplay can improve the human-likeness and believability of any agent controller — whether that is ad-hoc designed or built on data derived from gameplay. While the link between player modelling and believable agent design is obvious and direct the research efforts towards this integration within games is still sparse. However, the few efforts made on the imitation of human game playing for the construction of believable architectures have resulted in successful outcomes. Human behavior imitation in platform [46] and racing games [31] have provided human-like and believable agents while similar approaches for developing *Unreal Tournament* bots (e.g. in [33]) recently managed to pass the Turing test in the 2k BotPrize competition.

5.3 Playtesting Analysis and Game Authoring

While aiming at creating a particular game *experience*, designers can only define and alter the game *mechanics* [11, 27] (i.e., game rules) that, in turn, will affect the playing experience. From the mechanics arise the *game dynamics*, i.e., *how* the game is actually played. The dynamics lead to *game aesthetics*, i.e., what the player experiences during the game.

Even when a game is tested specifically to determine whether it provides the desired player experience, it is usually difficult to identify accurately which are the specific elements of the mechanics that work as intended. Player modeling, which yields a relationship between player state, game context, and in-game behavior, may support the analysis of playtesting sessions and the identification of what appeals to a particular player, and what does not (see [15, 43, 72] among many).

Player modeling provides a multifaceted improvement to game development as it does not only advance the study of human play and the enhancement of human experience. Quantitative testing via game metrics — varying from behavioral data mining to in-depth low scale studies — is improved as it complements existing practices [78].

Finally, user models can enhance authoring tools that, in turn, can assist the design process. The research field that bridges user modeling and AI-assisted design is in its infancy and only a few example studies can be identified. Indicatively, designer models have been employed to personalise mixed-initiative design processes [37, 36]. Such models drive the procedural generation of designer-tailored content.

5.4 Monetization of Free-to-Play Games

Recent years have seen an increasing number of successful free-to-play (F2P) games on diverse platforms, including Facebook, mobile devices and PC. In F2P games, playing the game itself is free. Revenues come from selling additional contents or services to the players with in-game *microtransactions* [20]. In order to be profitable, these games require developers to constantly monitor the purchasing behavior of their players [20]. Player modeling may improve the understanding of the players behavior [16] and help with the identification of players who are willing to pay. In addition, the information provided by player modeling might be used to customize the market content and mechanisms, eventually leading to increased profits.

6 The Road Ahead: Challenges and Questions

In this section we list a number of critical current and future challenges for player modeling as well as promising research directions, some of which have been touched upon in the previous sections.

- Regardless of the line of research in player modeling chosen, the biggest obstacle right now is lack of proper and rich data publicly available to the researchers. What is required is a rich multimodal corpus of gameplay and player data as well as player descriptions. Such a corpus must include detailed gameplay data for several games for a large number of players, including actions, events, locations, timestamps as well as biometrical data, that are trivial to obtain in large volumes (e.g., camera images and eye-tracking). Demographic data for the players must be available, as well as player information in the form of several questionnaires and structured interview data. Not all this data needs to be available for every subject in the database; several large datasets of gameplay data already exist, and it would be beneficial to include those in the database too.
- The use of procedural content generation techniques for the design of better games has reached a peak of interest in commercial and independent game development [78], which is showcased by successful (almost entirely procedurally generated) games such as *Minecraft* (Mojang, 2011) and *Love* (Eskil Steenberg, 2010). Future games, in general, are expected to contain less manual and more user-generated or procedurally-generated content, as the cost of content creation and the content creation bottleneck are key challenges for commercial game production. As the number of games that are (partially or fully) automatically generated grows, the challenge of modeling players in never-ending open worlds of infinite replayability value increases substantially.
- Nowadays, several modalities or player input are still implausible within commercial game development. For instance, existing hardware for physiological measurements requires the placement of body parts (e.g., head or fingertips) to the sensors, making physiological signals such as EEG, respiration and skin conductance rather impractical and highly intrusive for most games. Modalities such as facial expression and speech could be technically plausible in future games: even though the majority of the vision-based affect-detection systems currently available cannot operate in real-time [81], the technology in this field is rapidly evolving [32].

  On a positive note, recent advances in sensor technology have resulted in low-cost unobtrusive biofeedback devices appropriate for gaming applications. In addition, top game developers have started to experiment with multiple modalities of player input (e.g., physiological and behavioral patterns) for the personalization of experience of popular triple-A games such as *Left 4 Dead* (Valve, 2008) [1]. Finally, recent technology advances in gaming peripherals such as the PrimeSense camera showcase a promising future for multimodal natural interaction in games.
- Comparing model-based and model-free approaches to player modeling, we note that model-based inherently contains argumentation and understanding for the choices of the model, which model-free lacks. However, practice shows that model-based approaches often fail to encompass relevant features because of a lack of insight of the model builders. The model-free approach has the advantage of automatically detecting relevant features; however, it is also prone to detecting meaningless relationships between user attributes, game context and user experience. In computer games an extensive set of features of player behavior can be extracted and measured. At the same time there is, usually, lack of insight in what these features actually mean, at least at present. Therefore, in the current state of research, model-free approaches seem most suitable. Domain-specific knowledge, feature extraction and feature selection are necessary to achieve meaningful models of players .
- As mentioned before, player characteristics within a game environment may very well differ from the characteristics of the player when dealing with reality. Thus, validated personality

models such as psychology's Five Factor Model might not fit well to game behavior. An interesting direction in player modeling research is to determine a fundamental personality model for game behavior; such a model will have some correspondence with the Five Factor Model, but will also encompass different characteristics. Moreover, the behavioral clues that can be found in game behavior may be considerably different from those that can be found in reality.

- Experience has shown that diligent application of data mining techniques may provide insight into group behaviors. However, it remains difficult to make predictions about individuals. As such, player models can usually only give broad and fuzzy indications on how a game should adapt to cater to a specific player. One possible solution is to define several possible player models and classify an individual player as one of them (the stereotyping approach). Then, when gameplay is going on, the model can be changed in small steps to fit the player better. I.e., the player model is not determined as a static representation of the player, used to determine how the game should be adapted; rather it is a dynamic representation of a group of players, that changes to highlight the general characteristics of a specific player, and drives the game adaptation dynamically. In our view, this step-wise approach to game adaption by means of player modeling has the potential to lead to quick, good results in creating games that offer a personalized form of engagement to the player.

References

1 M. Ambinder. Biofeedback in gameplay: How valve measures physiology to enhance gaming experience. In *Game Developers Conference*, 2011.

2 Gustavo Andrade, Geber Ramalho, Hugo Santana, and Vincent Corruble. Extending reinforcement learning to provide dynamic game balancing. In *Proceedings of the Workshop on Reasoning, Representation, and Learning in Computer Games, 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 7–12, August 2005.

3 S. Asteriadis, K. Karpouzis, N. Shaker, and G.N. Yannakakis. Does your profile say it all? using demographics to predict expressive head movement during gameplay. In *Proceedings of the 20th conference on User Modeling, Adaptation, and Personalization (UMAP 2012), Workshop on TV and multimedia personalization*, 2012.

4 Stylianos Asteriadis, Kostas Karpouzis, and Stefanos D. Kollias. A neuro-fuzzy approach to user attention recognition. In *Proceedings of ICANN*, pages 927–936. Springer, 2008.

5 Ruth Aylett, Sandy Louchart, Joao Dias, Ana Paiva, and Marco Vala. Fearnot!–an experiment in emergent narrative. In *Intelligent Virtual Agents*, pages 305–316. Springer, 2005.

6 Sander Bakkes, Pieter Spronck, and Jaap van den Herik. Opponent modeling for case-based adaptive game ai. *Entertainment Computing*, 1(1):27–37, 2009.

7 A. Barreto, J. Zhai, and M. Adjouadi. Non-intrusive physiological monitoring for automated stress detection in human-computer interaction. In *Proceedings of Human Computer Interaction*, pages 29–39. Springer, 2007.

8 Michael Booth. The ai systems of left 4 dead. In *Keynote, Fifth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE09)*, 2009.

9 G. Calleja. *In-Game: From Immersion to Incorporation*. The MIT Press, 2011.

10 D. Charles and M. Black. Dynamic player modelling: A framework for player-centric digital games. In *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education*, pages 29–35, 2004.

11 Jenova Chen. Flow in games (and everything else). *Communications of the ACM*, 50(4):31–34, April 2007.

12 C. Conati. Probabilistic Assessment of User's Emotions in Educational Games. *Journal of Applied Artificial Intelligence, special issue on "Merging Cognition and Affect in HCI"*, 16:555–575, 2002.

13 Paul T. Costa and Robert R. McCrae. Domains and facets: hierarchical personality assessment using the revised NEO personality inventory. *Journal of Personality Assessment*, 64(1):21–50, 1995.

14 Mihaly Csikszentmihalyi. *Flow: the Psychology of Optimal Experience*. Harper Collins, 1990.

15 A. Drachen, A. Canossa, and G. N. Yannakakis. Player Modeling using Self-Organization in Tomb Raider: Underworld. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 1–8, Milan, Italy, September 2009. IEEE.

16 A. Drachen, R. Sifa, C. Bauckhage, and C. Thurau. Guns, swords and data: Clustering of player behavior in computer games in the wild. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pages 163–170, 2012.

17 Anders Drachen, Christian Thurau, Julian Togelius, Georgios N. Yannakakis, and Christian Bauckhage. Game data mining. In Magy Seif El-Nasr, Anders Drachen, and Alessandro Canossa, editors, *Game Analytics, Maximizing the Value of Player Data*, pages 205–253. Springer-Verlag, 2013.

18 P. Ekman and W.V. Friesen. Facial action coding system: A technique for the measurement of facial movement. palo alto. *CA: Consulting Psychologists Press. Ellsworth, PC, & Smith, CA (1988). From appraisal to emotion: Differences among unpleasant feelings. Motivation and Emotion*, 12:271–302, 1978.

19 L. Feldman. Valence focus and arousal focus: Individual differences in the structure of affective experience. *Journal of Personality and Social Psychology*, 69:53–166, 1995.

20 Tim Fields and Brandon Cotton. *Social Game design: monetization methods and mechanics*. Morgan Kaufmann, 2011.

21 N. Frijda. *The Emotions*. Cambridge University Press, Engelwood cliffs, NJ, 1986.

22 J. Frome. Eight ways videogames generate emotion. In *Proceedings of DiGRA 2007*, 2007.

23 Iain Gilfeather. Advanced racing game ai in pure. *GDC Europe*, 2009.

24 J. Gratch and S. Marsella. Evaluating a computational model of emotion. *Autonomous Agents and Multi-Agent Systems*, 11(1):23–43, 2005.

25 Erin Hastings, Ratan Guha, and Kenneth O. Stanley. Evolving content in the galactic arms race video game. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 241–248, 2009.

26 Ryan Houlette. Player modeling for adaptive games. In Steve Rabin, editor, *AI Game Programming Wisdom 2*, pages 557–566. Charles River Media, Inc, 2004.

27 Robin Hunicke, Marc LeBlanc, and Robert Zubek. Mda: A formal approach to game design and game research. In *Proceedings of the AAAI Workshop on Challenges in Game AI*, pages 04–04, 2004.

28 Hiroyuki Iida, N. Takeshita, and J. Yoshimura. A metric for entertainment of boardgames: its implication for evolution of chess variants. In R. Nakatsu and J. Hoshino, editors, *IWEC2002 Proceedings*, pages 65–72. Kluwer, 2003.

29 W. IJsselsteijn, K. Poels, and YAW de Kort. The game experience questionnaire: Development of a self-report measure to assess player experiences of digital games. *TU Eindhoven, Eindhoven, The Netherlands*, 2008.

30 K. Isbister and N. Schaffer. *Game Usability: Advancing the Player Experience*. Morgan Kaufman, 2008.

31 German Gutierrez Jorge Munoz and Araceli Sanchis. Towards imitation of human driving style in car racing games. In Philip Hingston, editor, *Believable Bots: Can Computers Play Like People?* Springer, 2012.

32 David Kadish, Nikolai Kummer, Aleksandra Dulic, and Homayoun Najjaran. The empathy machine. In Marc Herrlich, Rainer Malaka, and Maic Masuch, editors, *Entertainment Computing - ICEC 2012*, volume 7522 of *Lecture Notes in Computer Science*, pages 461–464. Springer Berlin Heidelberg, 2012.

33 Igor V Karpov, Jacob Schrum, and Risto Miikkulainen. Believable bot navigation via playback of human traces. In Philip Hingston, editor, *Believable Bots: Can Computers Play Like People?* Springer, 2012.

34 A. Kobsa. User modeling: Recent work, prospects and hazards. *Human Factors in Information Technology*, 10:111–125, 1993.

35 Raph Koster. *A theory of fun for game design*. Paraglyph press, 2005.

36 Antonios Liapis, Georgios Yannakakis, and Julian Togelius. Sentient sketchbook: Computer-aided game level authoring. In *Proceedings of ACM Conference on Foundations of Digital Games*, 2013. In Print.

37 Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Adapting models of visual aesthetics for personalized content creation. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(3):213–228, 2012.

38 Daniele Loiacono, Luigi Cardamone, and Pier Luca Lanzi. Automatic track generation for high-end racing games using evolutionary computation. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(3):245–259, 2011.

39 T. Mahlmann, A. Drachen, J. Togelius, A. Canossa, and G.N. Yannakakis. Predicting player behavior in tomb raider: Underworld. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 178–185. IEEE, 2010.

40 Thomas W. Malone. What makes things fun to learn? heuristics for designing instructional computer games. In *Proceedings of the 3rd ACM SIGSMALL symposium and the first SIGPC symposium on Small systems*, pages 162–169, 1980.

41 R. L. Mandryk, K. M. Inkpen, and T. W. Calvert. Using Psychophysiological Techniques to Measure User Experience with Entertainment Technologies. *Behaviour and Information Technology (Special Issue on User Experience)*, 25(2):141–158, 2006.

42 Michael Mateas and Andrew Stern. Façade: An experiment in building a fully-realized interactive drama. In *Game Developers Conference, Game Design track*, volume 2, page 82, 2003.

43 Olana Missura and Thomas Gärtner. Player modeling for intelligent difficulty adjustment. In Johannes Fürnkranz Arno Knobbe, editor, *Proceedings of the ECML–09 Workshop From Local Patterns to Global Models (LeGo–09)*, Bled, Slovenia, September 2009.

44 Anton Nijholt. BCI for Games: A State of the Art Survey. In *Proceedings of Entertainment Computing - ICEC 2008*, pages 225–228, 2009.

45 Jacob Kaae Olesen, Georgios N. Yannakakis, and John Hallam. Real-time challenge balance in an RTS game using rtNEAT. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 87–94, Perth, Australia, December 2008. IEEE.

46 Juan Ortega, Noor Shaker, Julian Togelius, and Georgios N Yannakakis. Imitating human playing styles in super mario bros. *Entertainment Computing*, 2012.

47 A. Ortony, G. L. Clore, and A. Collins. *The Cognitive Structure of Emotions*. Cambridge University Press, 1988.

48 Maja Pantic and George Caridakis. Image and video processing for affective applications. In Paolo Petta, Catherine Pelachaud, and Roddy Cowie, editors, *Emotion-Oriented Systems: The Humaine Handbook*, pages 101–117. Springer-Verlag Berlin Heidelberg, 2011.

49 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*, 2005.

50 N. Ravaja, M. Salminen, J. Holopainen, T. Saari, and J.J.A. Laarni. Emotional response patterns and sense of presence during video games: potential criterion variables for game design. In *Proceedings of the third Nordic Conference on Human-Computer Interaction*, 2004.

51 J. A. Russell. Core affect and the psychological construction of emotion. *Psychological Rev.*, 110:145–172, 2003.

52 N. Savva, A. Scarinzi, and N. Berthouze. Continuous recognition of player's affective body expression as dynamic quality of aesthetic experience. *IEEE Transactions on Computational Intelligence and AI in Games*, 2012.

53 K. R. Scherer. Studying the emotion-antecedent appraisal process: An expert system approach. *Cognition and Emotion*, 7:325–355, 1993.

54 K.R. Scherer. What are emotions? and how can they be measured? *Social science information*, 44(4):695–729, 2005.

55 N. Shaker, S. Asteriadis, G. Yannakakis, and K. Karpouzis. A game-based corpus for analysing the interplay between game context and player experience. In *Affective Computing and Intelligent Interaction*, pages 547–556. Springer, 2011.

56 M. Sharma, M. Mehta, S. Ontanón, and A. Ram. Player modeling evaluation for interactive fiction. In *Proceedings of the AIIDE 2007 Workshop on Optimizing Player Satisfaction*, pages 19–24, 2007.

57 B. F. Skinner. *The Behavior of Organisms: An Experimental Analysis*. Cambridge, Massachusetts: B. F. Skinner Foundation, 1938.

58 A.M. Smith, C. Lewis, K. Hullet, and A. Sullivan. An inclusive view of player modeling. In *Proceedings of the 6th International Conference on Foundations of Digital Games*, pages 301–303. ACM, 2011.

59 Nathan Sorenson and Philippe Pasquier. Towards a generic framework for automated video game level creation. In *Proceedings of the European Conference on Applications of Evolutionary Computation (EvoApplications)*, volume 6024, pages 130–139. Springer LNCS, 2010.

60 Pieter Spronck, Ida Sprinkhuizen-Kuyper, and Eric Postma. Difficulty Scaling of Game AI. In *Proceedings of the 5th International Conference on Intelligent Games and Simulation (GAME-ON 2004)*, pages 33–37, 2004.

61 S. Tekofsky, P. Spronck, A. Plaat, H.J. van den Herik, and J. Broersen. Psyops: Personality assessment through gaming behavior. In *Proceedings of the FDG 2013*, 2013.

62 Ruck Thawonmas, Masayoshi Kurashige, Keita Iizuka, and Mehmed Kantardzic. Clustering of Online Game Users Based on Their Trails Using Self-organizing Map. In *Proceedings of Entertainment Computing - ICEC 2006*, pages 366–369, 2006.

63 D. Thue, V. Bulitko, M. Spetch, and E. Wasylishen. Learning player preferences to inform delayed authoring. In *AAAI Fall Symposium on Intelligent Narrative Technologies*. AAAI Press, Arlington, 2007.

64 David Thue, Vadim Bulitko, Marcia Spetch, and Eric Wasylishen. Interactive storytelling: A player modelling approach. In *The Third Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pages 43–48, Stanford, CA, 2007.

65 Christian Thurau, Christian Bauckhage, and Gerhard Sagerer. Learning human-like Movement Behavior for Computer Games. In S. Schaal, A. Ijspeert, A. Billard, Sethu Vijayakumar, J. Hallam, and J.-A. Meyer, editors, *From Animals to Animats 8: Proceedings of the* 8^{th} *International Conference on Simulation of Adaptive Behavior (SAB-04)*, pages 315–323, Santa Monica, LA, CA, July 2004. The MIT Press.

66 Julian Togelius, Mike Preuss, Nicola Beume, Simon Wessing, Johan Hagelbäck, and Georgios N. Yannakakis. Multiobjective exploration of the starcraft map space. In *Proceedings*

of the IEEE Conference on Computational Intelligence and Games, pages 265–272, Copenhagen, Denmark, 18–21 August 2010.

67 Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. Search-based procedural content generation. In *Proceedings of the European Conference on Applications of Evolutionary Computation (EvoApplications)*, volume 6024. Springer LNCS, 2010.

68 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*, pages 321–328. IEEE, 2010.

69 Giel van Lankveld. *Quantifying Individual Player Differences*. PhD thesis, Tilburg University, The Netherlands, 2013.

70 Giel van Lankveld, Pieter Spronck, and Matthias Rauterberg. Difficulty Scaling through Incongruity. In *Proceedings of the 4th International Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 228–229. AAAI Press, 2008.

71 Giel van Lankveld, Pieter Spronck, Jaap van den Herik, and Arnoud Arntz. Games as personality profiling tools. In *2011 IEEE Conference on Computational Intelligence in Games (CIG'11)*, pages 197–202, 2011.

72 Ben Weber and Michael Mateas. A Data Mining Approach to Strategy Prediction. In *IEEE Symposium on Computational Intelligence in Games (CIG 2009)*, pages 140–147, Milan, Italy, September 2009.

73 G. Yannakakis and J. Hallam. Game and player feature selection for entertainment capture. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 244–251, 2007.

74 G. Yannakakis and J. Hallam. Rating vs. preference: a comparative study of self-reporting. *Affective Computing and Intelligent Interaction*, pages 437–446, 2011.

75 Georgios N. Yannakakis and John Hallam. Towards Optimizing Entertainment in Computer Games. *Applied Artificial Intelligence*, 21:933–971, 2007.

76 Georgios N. Yannakakis and Manolis Maragoudakis. Player modeling impact on player's entertainment in computer games. In *Proceedings of the 10^{th} International Conference on User Modeling; Lecture Notes in Computer Science*, volume 3538, pages 74–78, Edinburgh, 24–30 July 2005. Springer-Verlag.

77 G.N. Yannakakis. Preference learning for affective modeling. In *Affective Computing and Intelligent Interaction and Workshops, 2009. ACII 2009. 3rd International Conference on*, pages 1–6. IEEE, 2009.

78 G.N. Yannakakis. Game AI Revisited. In *Proceedings of the 9th conference on Computing Frontiers*, pages 285–292. ACM, 2012.

79 G.N. Yannakakis, H.P. Martínez, and A. Jhala. Towards affective camera control in games. *User Modeling and User-Adapted Interaction*, 20(4):313–340, 2010.

80 G.N. Yannakakis and J. Togelius. Experience-driven procedural content generation. *Affective Computing, IEEE Transactions on*, 2(3):147–161, 2011.

81 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*, 31(1):39–58, 2009.

Procedural Content Generation: Goals, Challenges and Actionable Steps

Julian Togelius[1], Alex J. Champandard[2], Pier Luca Lanzi[3], Michael Mateas[4], Ana Paiva[5], Mike Preuss[6], and Kenneth O. Stanley[7]

1 **Center for Computer Games Research, IT University of Copenhagen, Copenhagen, Denmark**
julian@togelius.com
2 **AiGameDev.com KG, Vienna, Austria**
alexjc@aigamedev.com
3 **Department of Electronics and Information, Politecnico di Milano, Milano, Italy**
lanzi@elet.polimi.it
4 **Center for Games and Playable Media, University of California, Santa Cruz, California, USA**
michaelm@cs.ucsc.edu
5 **Intelligent Agents and Synthetic Characters Group, INESC-ID, Lisboa, Portugal**
ana.paive@inesc-id.pt
6 **Department of Computer Science, Technical University of Dortmund, Dortmund, Germany**
mike.preuss@cs.tu-dortmund.de
7 **Department of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, USA**
kstanley@eecs.ucf.edu

Abstract

This chapter discusses the challenges and opportunities of procedural content generation (PCG) in games. It starts with defining three grand goals of PCG, namely multi-level multi-content PCG, PCG-based game design and generating complete games. The way these goals are defined, they are not feasible with current technology. Therefore we identify nine challenges for PCG research. Work towards meeting these challenges is likely to take us closer to realising the three grand goals. In order to help researchers get started, we also identify five actionable steps, which PCG researchers could get started working on immediately.

1998 ACM Subject Classification I.2.1 Applications and Expert Systems: Games

Keywords and phrases procedural content generation, video games

Digital Object Identifier 10.4230/DFU.Vol6.12191.61

1 Introduction

Procedural content generation (PCG) refers to the algorithmic generation of game content with limited or no human contribution. "Game content" is here understood widely as including e.g. levels, maps, quests, textures, characters, vegetation, rules, dynamics and structures, but not the game engine itself nor the behaviour of NPCs. PCG has been part of published games since the early eighties, with landmark early examples being the runtime

Artificial and Computational Intelligence in Games. *Dagstuhl Follow-Ups*, Volume 6, ISBN 978-3-939897-62-0.
Editors: Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius; pp. 61–75

Dagstuhl Publishing
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany

generation of dungeons in *Rogue* and the extremely compressed representation of hundreds of star systems in *Elite*. Prominent recent examples of PCG include the ubiquitous *SpeedTree* system, used for generation of trees, grass and other types of vegetation in hundreds of commercial games, and the generation of dungeons, weapons and items in the *Diablo* series of games. PCG can be used for a variety of reasons, including providing variety, reducing development time and development costs, saving space in transmission or on disk, augmenting human creativity and enabling adaptivity in games.

The academic research community around PCG has formed only within the last few years. While a number of important papers were published in venues dedicated to AI or graphics research, the first workshop devoted entirely to PCG took place in 2009 and the first journal special issue devoted to the topic came out only in 2011[1]. The solutions to content generation problems that have been explored by academic researchers have tended to be focused more on adaptable and controllable (and sometimes highly complex) algorithms, whereas the types of algorithms that have so far been used in published games tend to be simpler, faster and less controllable.

The workgroup on PCG decided to look at the field from the perspective of what research would be most important to do in the future in order to ensure that real breakthroughs are made, and modern PCG techniques will be able to add value to the games industry by enabling new types of games as well as new ways of developing games. We took a top-down perspective and started with thinking about what sort of things we would ultimately want PCG to be able to achieve; the grand *goals* of the research field. We then tried to identify the most important *challenges* that need to be overcome in order to be able to realise these goals, and finally identified a handful of *actionable steps* that could make progress towards overcoming these challenges, and which could be taken already today by someone – perhaps you? – interested in contributing to the field.

2 Goals

The following is a list of what we consider the most important goals of PCG research. In each case, the goal is currently not obtainable and it would require significant further research effort leading to some sort of breakthrough in order to be able to realise the goal. However, in each case there is already some work done that directly addresses this goal, so for each item we list what we consider the most relevant previous work.

2.1 Multi-level Multi-content PCG

Imagine being able to push a button and generating a complete game world: terrain, vegetation, roads, cities, people, creatures, quests, lore, dialogue, items, vehicles; polygons, graphs, textures, text. For example of such a fully fledged game world, think of the *Elder Scrolls V: Skyrim* (Bethesda) game world, complete with all that which makes it so immersive and exciting – except for the underlying game engine. And then press the same button again, and you get a fresh new world, different from the previous world in every respect: the details, the overall structure, the look and feel. Maybe it's a sci-fi world threatened by invasion, or a murder mystery in a contemporary industrial city. The only limits for the expressive space

[1] The PCG workshop runs annually since 2010, co-located with the Foundations of Digital Games Conference. The autumn 2011 issue of IEEE Transaction on Computational Intelligence and AI in Games is entirely devoted to PCG. A discussion group for the PCG community can be found at `https://groups.google.com/group/proceduralcontent/`

are the capabilities of the underlying game engine, which provides primitives for movement, social interactions, combat, scoring etc. The system we imagine should be so flexible so that it would be possible to plug in a new game engine, complete with a new set of rules and dynamics, and go on to generate complete game worlds that fit with that engine.

In other words we are here envisioning a system that can generate multiple types of quality content at multiple levels of granularity in a coherent fashion while taking game design constraints into consideration. Nothing like this exists yet, and while it might not be possible to achieve such a system in the foreseeable future, we see much room for progress in this direction.

Almost all existing approaches to content generation generate a single type of content for a single game, and the results all too often look uninspired and generic. There are some interesting examples of trying to generate several types of content so that they fit in with each other. *Dwarf Fortress* (Bay 12 Games) generates many aspects of the game world, including its geology and backstory, all the way down to the socks on each foot of individual dwarfs. However, for each type of content the generation process is very simple, and the generated game worlds show little variation. A similar problem is tackled by Hartsook et al., though that paper employs a "waterfall" model where one type of content (stories) are generated first and the second (maps) afterwards, with no feedback between process [9]. Another approach to multi-level multi-content generation is that of the *Sketchaworld* system by Smelik et al. [23]. This system allows for mixed initiative generation of various aspects of a landscape, including topology, vegetation, and the placement of roads and buildings. All objects have semantics and defined interdependencies, which together with a conflict resolution system allows editing on one level (e.g. changing the flow of a river) to have effects on entities on other levels (e.g. a bridge is automatically created over the river to allow a pre-existing road to pass through). However, that system does not address game-specific design considerations such as balance or challenge.

2.2 PCG-based Game Design

Imagine a game where procedural content generation was a central mechanic, without which the game could not exist at all; in fact, the whole genre to which the game belongs could not exist without procedural content generation. Imagine that the game was truly endless, and that exploration of an infinite range of content was a central part of the gameplay, indeed the main reason the game was so appealing.

Almost all existing approaches to PCG focus on generating content for an existing game, where the core game itself could exist without the PCG mechanism. PCG is done to facilitate development, or to allow for runtime adaptation. Even in games such as *Rogue*, *Spelunky* and *Diablo*, where a key feature of the game is the endless variation in game content, all of the levels could in principle have been generated offline and presented to the player without taking player choice into account. Creating games where a PCG algorithm is an essential part of the game design requires innovations both in game design, where parameters to the PCG algorithm need to be meaningfully based on player actions, and in PCG, where the algorithm needs to be reliable and controllable beyond the capacities of most current algorithms.

A few games have made progress towards realising this vision. *Galactic Arms Races* and *Petalz* both feature the evolution of new content as a core part of the game, where the players respectively evolve weapons to defeat enemies or evolve flowers to impress friends as the game is being played [10, 19]. *Endless web* is a platform game that lets the player explore different dimensions of content space, and generates new levels in response to the

player's actions[26]. In *Infinite Tower Defence*, the role of PCG is to create new levels and enemies that match the current strategy of the player so as to force the player to explore new strategies [2]. Another example is *Inside a Star-filled Sky*, which features a "zooming" mechanic, where the player character can enter enemies and explore new levels inside them; this yields an apparently endless hierarchy of nested levels. While these games are addressing the challenge of PCG-based game design, they are still variations on well-known genres rather than examples of genres that could only exist because of PCG.

2.3 Generating Complete Games

Imagine a PCG system that could create complete games. Not just content for an existing game, but the whole game from scratch, including the rules and game engine. At the press of a button, starting from nothing, the system will create a game no-one has played before and which is actually enjoyable to play. This would involve automating the arguably most central and "AI-complete" aspects of game design, including estimating how a human player would experience interacting with a complete system of rules and affordances. Perhaps the system accepts parameters in the form of a design specification for the game. For the example, a human might task the system with designing a game that features fog of war, that promotes collaboration or that teaches multiplication.

Several attempts have been made to generate game rules, sometimes in combination with other aspects of the game such as the board. Of particular note is Cameron Browne's *Ludi* system which generated a board game of sufficient novelty to be sold as a boxed product through searching through a strictly constrained space of board games [4]. This system built on evolutionary computation, as did earlier [32] and later [6] attempts to evolve simple arcade-style games. Other attempts have build on symbolic techniques such as logic programming [25, 34]. While these examples have proven that generation of playable game rules is at all possible, they all generate only simple games of limited novelty.

3 Challenges

Analysing what would be needed in order to reach the grand goals discussed above, the workgroup arrived at a list of eight research challenges for procedural content generation. Successfully meeting any of these challenges would advance the state of the art in PCG significantly, and meeting all of them would probably render the goals described above attainable. Addressing any of these challenges could make a good topic for a PhD thesis.

3.1 Non-generic, Original Content

Generated content generally looks generic. For example, looking at the dungeons generated for a roguelike game such as those in the *Diablo* series, you easily get the sense that this is just a bunch of building blocks hastily thrown together with little finesse – which is just what it is. Re-generate the level and you get a superficially very different but ultimately equally bland level. Most generated levels lack meaningful macro-structure and a sense of progression and purpose. Very rarely would you see a generated level about which you could say that it was evidence of skill or artfulness in its creator, and even more rarely one which showcases genuine design innovation. Compare this with the often masterfully designed levels in comparable games such as those in the *Zelda* franchise. The Zelda levels are aesthetically pleasing in several ways, providing a clear sense of place and progression, and often offering some original and unique take on the design problems of action adventure games.

There are a few examples of PCG systems having come up with what could be called genuine inventions. In the *Galactic Arms Race* game, several of the weapons that were generated (such as the tunnel-maker and hurricane) were surprising to players and designer alike and unlike anything the designers had seen before, yet were effective in the game and opened up for new playing styles, as if they had been designed by a human designer [10]. As discussed above, Browne's *Ludi* system managed to come up with a game (*Yavalath*) that was sufficiently novel to be sold as a boxed product. However, we do not know of a system that has exhibited sustained creativity, or that (in the language of Margaret Boden [3]) has displayed transformational rather than just exploratory creativity.

The challenge, then, is to create content generators that can generate content that is purposeful, coherent, original and creative. This challenge is quite broad, as interpretations of these adjectives could vary – this only means that there are many different ways of approaching the challenge.

3.2 Representing Style

Directly connected to the previous challenge but somewhat more specific is the challenge to create a content generator that can create content in a particular style that it has somehow learned or inferred. Human artists of various kinds can observe artefacts produced by another artist, and learn to imitate the style of that artist when producing new artefacts – e.g., a skilful painter could study a number of Picasso paintings and then produce new paintings that were recognisably in the same style (while presumably not exhibiting the same creativity), and then go to on to study Mondrian paintings and produce Mondrian-like paintings of her own. An analogous capacity in PCG could be a level generation system that could study the seminal level designs of Shigeru Miyamoto in the *Zelda* and *Super Mario* series, and produce similar designs automatically; the same generator could, after being presented with John Romero's significantly different designs for *Doom* levels, learn to imitate that style too. Perhaps the generator could be presented with artefacts that were not game levels, for example architectural designs by Frank Lloyd Wright, and learn to reproduce that style within the constrained design space of levels for a particular game. Similarly competent texture generators, game rule generators and and character generators could also be imagined. It is important to note that the challenge is not only to imitate the surface properties of a set of designs (such as recurring colours and ornamentation) but also the deeper features of the design, having to do with expression of ideas and emotions in interplay with the player and game design.

Some attempts have been made to model player preferences in level generators [16] or designer preferences in interactive evolution for content generation [15]. However, the preferences modelled in these experiments are still very indirect, and the generated artefacts still exhibit the style of the generator more than that of the player or designer.

3.3 General Content Generators

Almost all existing PCG algorithms generate a single type of content for a single game. Reusability of developed PCG systems is very limited; there is no plug-and-play content generation available. This can be contrasted with the situation for other types of game technology, where game engines are regularly reused between games and even some aspects of game AI (such as pathfinding) now being available as middleware. The only PCG middleware that is actually in use in multiple games is *SpeedTree*, but this again generates only a single type of content (vegetation) and that type is of little functional significance in most games,

meaning that the risks of generating content are rather low; ugly shrubbery is ugly but tends not to break the level. The lack of readily available PCG systems that could be used without further development to generate for example levels or characters for a new game is probably holding back adoption of PCG techniques in the game industry.

It is already the case that certain techniques underly a number of different PCG systems. For example, L-systems [18] are at the core of both *SpeedTree* and techniques for creating landscapes [1] and levels [7]. Similarly, compositional pattern-producing networks (CPPNs) [29] are the basis for both the spaceships in Galactic Arms Race [10], the flowers in Petalz [19] and the pictures in PicBreeder [20]. However, in each case significant engineering effort was required to make the method work with the particular type of content in the particular game.

A general content generator would be able to generate multiple types of content for multiple games. The specific demands, in term of aesthetics and/or in-game functionality, of the content should be specified as parameters to the generator. A game designer armed with such a tool would just need to properly specify the requirements for the content that should be part of a new game in order to promptly have a means of automatically generating content for it. One idea for how to achieve this is to treat PCG algorithms as content themselves, and generate them using other PCG methods so as to fit the particular content domain or game they are meant to be applied to [14].

3.4 Search Space Construction

If content is to be generated then it must be be situated within a search space. The structure of the search space determines what content can be reached from small perturbations of any particular instance in the search space. For a content generator to be successful, the structure of the search space needs to have certain forms of locality. In general, small perturbations in the underlying representation should not lead to radical changes in the appearance or functionality of the content itself. For example, a table should not turn into a mushroom in a single small mutation. Rather, the table might become a little shorter, a little taller, or a little rounder, but it would remain a recognisable table.

This search space structure is ultimately determined by the selected underlying representation for the class of content being generated. Existing representations include the L-sytems [18] and CPPNs [29] discussed in the previous section, as well as logic-based representations such AnsProlog code snippets [24] and more direct representations where individual numbers correspond to individual features of the artefact. Such representations bias the generator towards producing certain qualitative themes, such as fractals in L-systems or symmetry in CPPNs. These biases are a reflection of the structure of the search space induced by the representations – fractals tend to be reachable from many different parts of a search space induced by L-systems. Thus the representation becomes an implicit means of structuring which types of content neighbour which, and therefore which artefacts can lead to which others.

This relationship between representation and search space structure highlights the significant challenge of designing a generator for a specific type of content: If a type of content is to be generated – say vehicles or buildings – then the engineers designing the generator must carefully construct a representation that induces a reasonable structure on the search space. One would not want airplanes to change in one step into wheelbarrows. To ensure such a smooth a tightly coupled landscape, the designer must intimately understand the relationship between the underlying representation and the structure of the space it induces, a relation which is not necessarily intuitive. For this reason, designing the search space to

make a satisfying generator requires significant skill and insight, and there are currently few general principles known to the research community for undertaking such a task.

In the future, it is possible that tools can be built to aid in adapting a particular representation for a particular class of content. For example, as noted in the previous section, CPPNs have encoded pictures, weapon systems, and flowers by interpreting their outputs and choosing their inputs carefully. It would be most helpful if a tool could help to automate such choices so that a particular representation could be adapted to a particular content class more easily.

3.5 Interfaces and Controllability for PCG Systems

Most existing PCG systems are not easy for a human to interface with and control. Many classic PCG implementations, for example the dungeon generators in many roguelike games, have no parameters of control at all; taking a random seed as input, a level is generated as output. In very many cases, you as a user (or as a game) would need to have more control of the generated artefact. Like controlling how difficult a level is, whether it should be more suited to speed runners or to explorer-type players, how much treasure and how many puzzles it should contain, and whether it should include a green flagpole at a particular location or perhaps five pixels to the left of that position. Or the age of a generated flower, the intuitiveness of a ruleset or the hipness of a car. There are many possible types of control that could be desirable, depending on the game and the designer.

Some classic constructive algorithms such as L-systems offer ways for the designer to specify aspects of the generated content, such as the "bushiness" of a plant. Search-based approaches allow the designer to specify desirable properties of the content in the form of objectives, but encoding the desired qualities in a fitness function is often far from straightforward and there is no guarantee that content with high values on these objectives can be found in the search space. Other PCG paradigms such as solver-based PCG using e.g. Answer Set Programming [24] offer complementary ways of specifying objectives, but again, it is not easy to encode the desired qualities for a non-expert. The mixed-initiative PCG systems Sketchaworld [23] and Tanagra [27] explicitly address this problem by allowing the user to interact with the PCG system by moving and creating objects in physical space, and thus imposing constraints on how what can be generated where. These systems clearly show a viable way forward, but so far only some aspects of control has been achieved (physical location) at the cost of some limitations in what sort of underlying PCG methods can be used.

What would it mean to allow users (be they designers or players, or perhaps some other algorithm such as a challenge balancing system) complete control over the content generation algorithm? Presumably it would mean that they could at any point during the generation process change any aspect of the content: making the level more blue or less scary or just making all of the gaps except the fifth one contain apples. Then the generator responds by implementing the changes, perhaps introducing new features or removing others, but still respecting what the user has specified wherever possible, and intelligently resolving conflict between specifications (e.g. the apples in the gaps could make the level more difficult). One could imagine something like *Adobe Photoshop*'s extreme array of expressive tools, including brushes, filters and abilities to only have modifications apply to particular layers, but all the way taking game design into account and autonomously generating appropriate content suggestions. It should be emphasised that this is not only a formidable challenge for PCG algorithms, but also for human-computer interaction. It is not even obvious how to represent aspects of game content that depend on being played to be experienced (such as game rules)

in an editor; a recent attempt at mixed-initiative generation of game rules mostly highlighted the problem [30].

3.6 Interaction and Opportunistic Control Flow Between Generators

Closely related to the previous challenge, and crucial for the goals of multi-level PCG and generating complete games, is the challenge to devise workable methods for communication and collaboration between algorithms working on generating different aspects or layers of the same artefact. For example, when a system generates the rules for a game, the physical environments for the same game and the characters or creatures that feature in it, the various generative algorithms must be able to communicate with each other. The simplest way of implementing this would probably be a "waterfall" model where the rules are generated first, positing requirements on the terrain/levels generator, in turn further constraining the space available for the creature/character generators. But this rules out any innovations in rules which are dependent on, and initiated by, uncommon solutions and crazy ideas in level or character design. In fact, as the rule generator cannot know what sort of characters the character generator will be able to produce (unless the latter's search space is severely constrained), the rules will have to be constrained to be very bland and workable with pretty much any characters. For these reasons, games developed by teams of humans are often developed in a much more opportunistic way, where opportunities or problems discovered at any content layer could affect the design of the other layers (e.g. the invention of a new type of enemy spurs the invention of new rules).

How can we replicate such an opportunistic control flow in a completely (or mostly) algorithmic environments, where algorithms (or perhaps some algorithms and some humans) collaborate with each other? One could imagine a system where constraints are posted in a global space, but this requires that a language and/or ontology be constructed to make such constraints comprehensible across generators, and also that a system is devised for working out priorities and solving conflicts between constraints. Going further, one could imagine equipping the content generators with models of themselves so as to provide a level of introspection, allowing them to exchange models of the bounds of their generative spaces.

3.7 Overcoming the Animation Bottleneck

In modern 3D computer games, animation is a major concern. Almost everything needs to be animated: creatures, characters, vehicles and features of the natural world such as vegetation and water. Creating believable animations even if the underlying characters are not procedurally generated is a huge challenge. In particular:

- Motion capture or hand animation is very expensive to acquire, and requires either the use of specialised motion capture facilities or an extensive team of animators.
- Data-heavy forms of animation such as motion capture or hand-animation also costs a significant amount of time, and are often a bottleneck for improving character behaviour.
- Animation systems based on data require significant runtime overheads for shifting around the data, decompressing it and generating runtime poses via blending.

This makes current animation techniques a bottleneck in three different ways, each as important as the other. Procedural techniques are already starting to resolve each of these three different issues, and the games industry is highly interested in the results [5].

There are many domain-specific problems to generating compelling animations for characters that were hand defined, but harder still is the problem of animating procedurally

generated creatures. Being able to generate an artefact does not mean that one automatically is able to animate it, and if one is not able to convincingly animate an artefact it is more or less useless in-game, as it would break the suspension of disbelief.

The big challenge of procedural animation is to match the high expectations of human observers, without having to resort to stylisation as a solution. This solution will involve subtle combinations of data and code that are crafted and assembled together masterfully by skilled technical animators, and new algorithms to make this possible.

3.8 Integrating Music and Other Types of Content

While most computer games feature music, the whole audio component usually only serves the task of supporting the game flow or emphasising the general atmosphere of the game. This is often done either by producing complete sound tracks as in the movie industry or by designing a very simple generative system (as in the Google app *Entanglement*) that repeatedly recombines single parts in order to stretch the available music and prevent boredom. Games that actively use the music as source of information to create game content or, vice versa, use game events for adjusting or even creating music are still rare.

In many well-known games based on music (e.g. Guitar Hero or SingStar), the interaction between music and game events is completely fixed and has been manually created. An overview of the different possibilities is given in [17]. Some examples of more complex interaction are:

- Rez (Sega)[2] from 2001, a rail shooter that tightly binds the visual impression (and appearance of enemies) to the composed music and also feedbacks user actions acoustically.
- Electroplankton by Nintendo[3] from 2005, where the player interacts in various ways with the game to create audio and visual experiences.
- The turntable-like game Scratch-Off [8] that requires user interaction matching the rhythm of the music while blending over to another piece of music.
- The Impossible Game is a graphically simple but challenging platform game for consoles and mobile phones that requires the user to cope with very different obstacles that are generated in accordance with current music events.
- The Bit.Trip series by Gaijin Games features the main character Commander Video who has to cope with game events mostly based on the rhythm of the played music in 6 very different games (rhythm shooter, platformer).
- Wii Music automatically generates melodies based on how the player moves the controller.
- BeatTheBeat [12] features three mini-games (rhythm-based tap game, shooter, and tower defence) that each use the available music as background and its analysed feature events as source for game event creation.

Music informatics has made great strides in recent years, including the emergence personalised music selection systems (including learning personal preferences) as well as complex music generation systems. Given this progress, it should be possible to establish game/music systems that interact in a much more complex way than what we are used to see in most games. In principle, one could take an existing simple game and attach it to a recommendation or generating system. Some thought has to be put into designing clever interfaces, but the problem appears per se as solvable. Using existing music as input source

[2] http://www.sonicteam.com/rez
[3] http://www.mobygames.com/game/electroplankton

for realtime production of game content just becomes possible now as modern computers are powerful enough to analyse the music online. This scheme can enhance a game considerably because the user can modify it by selecting the music that best reflects the current mood or just some very different music to create a new experience.

However, this strategy is only applicable for simple games with a restricted number of (partly repetitive) game components. For more complex games, interaction in the other direction (game events influence the music played) may make more sense. The components to achieve this are available, simple forms of modulation would be changing volume, speed, or key of the music. One could also think of online remixing by changing the volume of single instrument tracks, an approach that has been tried some years ago by several musicians by means of online tools (e.g. Peter Gabriel, William Orbit), but not in the games context. The feasibility of this approach highly depends on access to the single music and games components, but technically appears to be rather simple. A more sophisticated approach would apply a music generating system in order to modify or re-create the currently played music. However, for achieving a believable connection between game events and music, the semantic of the game events needs to be defined in a transferable fashion, for example through their effects on the player's mood.

3.9 Theory and Taxonomy of PCG Systems

While PCG research has been steadily increasing in volume in the last few years, there has been a lack of unifying theory to guide the research or even to help relating the disparate contributions to each other. New algorithms have been proposed and case studies carried out in a number of game domains, but it is seldom clear in what ways a new approach is better (or worse, or just different) to existing approaches. A theory and taxonomy of PCG would explain the relative advantages of different approaches, why some content generation problems are harder than others, and which approaches are likely to work on what problems. It would also help situate and analyse any new algorithms proposed. A recent paper makes an initial attempt to provide a taxonomy for PCG, but covers mostly search-based approaches, in particular those based on evolutionary computation [33].

4 Actionable Steps

The challenges listed above are somewhat abstract and are mostly long-term projects. For some of them, it is not even clear how to approach them. To make matters more concrete, and to provide a set of example projects for anyone wishing to contribute to advancing the state of the art of procedural content generation, we devised a number of *actionable steps*. These are more specific research questions around which projects with a limited scope could be formulated, and for which the technical prerequisites (in terms of algorithms and benchmarks) already exist. You could start working on any of these steps already today.

4.1 Atari 2600 Games

Inventing a system that could generate a complete game with the complexity and scope of a modern AAA game might be a tall task – after all, developing these games often takes hundreds of person-years. Fortunately, not all games are equally complex. Those games that were made for early computers and game consoles were substantially less complex, as available memory size, processing speed, graphics resolution and development teams were all fractions of what they are today. The *Atari 2600* games console, released in 1977, had

4 kilobytes of RAM, a 1.2 MHz processor and was home to classic games such as *Pitfall*, *Breakout*, *Adventure* and *Pac-Man*. All of them had two-dimensional graphics that adhered to certain constraints regarding e.g. the number of movable objects that were dictated by the system's hardware design.

One could realistically try to directly approach the third of the grand goals outlined above, that of generating complete games, working within the much constrained space of games that would be possible on a system such as the Atari 2600. The limited space makes it much more tractable to search for good games, regardless of which search-mechanism would be used. The limited computational requirements of running these games would also make it possible to test them, for example by simulating playthroughs, at a much faster pace than real-time. A successful game generation system for Atari 2600 games should be able to re-invent classic games such as *Breakout* and *Pac-Man*, but also to invent enjoyable games that have never before been seen and which differ in some interesting way from all existing games. Developing such a system would require devising a description language for this type of video games, that is complete enough to cover essentially all interesting such games, but which still enables the space to be effectively searched; for some thoughts on how this could be done, please see the chapter *Towards a Video Game Description Language* in this volume.

Relates directly to challenges: general content generators, search space construction, interaction and opportunistic control flow.

4.2 Procedural Animation for Generated Creatures

One way of approaching the challenge of bringing PCG and procedural animation together is to develop a creature generator which generates creatures together with suitable procedural animation. The most impressive attempt to do something similar is probably the Creature Creator in *Spore*, but that is an editor rather than an automatic content generator, and imposes certain constraints on the generated creatures that should be avoided in order to be able to search a more complete creature space.

Relates directly to challenges: overcoming the animation bottleneck, interfaces for PCG systems.

4.3 Quests and Maps

The computational generation and adaptation of narrative is a specific type of PCG which enjoys its own research field, and a number of promising approaches have been presented, most of them based on planning algorithms [35]. Another domain of PCG which has seen much attention is the generation of maps of different kinds. Maps are automatically generated in games such as *Civilization* and *Diablo*, and a number of recent studies have investigated techniques such as cellular automata [11], grammars [7, 1] and evolutionary computation [31] for generating interesting and/or balanced maps for different types of games.

Generating maps and quests together could potentially be an excellent showcase for multilevel PCG (the first of the grand goals outlined above), as the best-designed games often feature quests and maps that interact and reinforce each other in clever ways – the map helping the game tell the story, and the story helping the player explore the map – and there are workable approaches to generating each type of content separately. However, there is surprisingly little work done on generating quests and maps *together*. One of the few examples is Dorman's use of grammars for generating Zelda-style dungeons and quests together, achieving good results by severely limiting the domain [7]. Another is Hartsook et

al.'s generation of matching maps and quests, by simply generating the map after the quest so that the former fits with the latter [9].

There is plenty of scope for taking this further and generating maps and quests that are perfect fits for each other. The interested investigator could start by taking one of those algorithms that has been proven to work well for one domain (maps or quests) and try to encode the other domain within the first, or by devising a scheme where map and quest generators take turns to respond to each other, or perhaps by trying to invent a completely new algorithm for this task. One could also try to allow human intervention and input at any phase of the quest/map generation.

Relates directly to challenges: interaction and opportunistic control flow, general content generators, interfaces for PCG systems.

4.4 Competent Mario Levels

The *Mario AI Benchmark* is a popular testbed within the CI/AI in games community, based on an open source clone of Nintendo's classic platformer *Super Mario Bros.* The benchmark has been used for a series of competitions focused on developing AI controllers that play the game proficiently [13], but in 2010 and 2012 it was also used for a competition where entrants submitted level generators capable of generating personalised levels for particular players [22]. A number of PCG systems were submitted to this competition, and a number of other PCG experiments using the Mario AI Benchmark have been published following the competition [21, 28].

However, comparing the quality of the generated levels with those that can be found in the real Super Mario Bros game, or with human-designed levels in any other high-quality platformer, makes any PCG aficionado disappointed. The generated levels typically lack a sense of progression, or any other macro-structure for that matter. Unlike the real Super Mario Bros levels, there is no sense that they are telling a story, trying to teach the player a skill, or hiding a surprise. Furthermore, the generated levels frequently feature items and structures that make no sense, unexplainable difficulty spikes and repeated structures that seem to be taken straight from a structure library. A high priority for someone interested in procedurally generating platform levels should be to devise an algorithm that can create levels with a sense of purpose. Using the Mario AI Benchmark as a testbed means that there is no shortage of material for comparisons, both in the form of level generators and in the form of professionally designed levels.

Relates directly to challenges: non-generic content, representing style.

4.5 Player-directed Generation with Model-based Selection

A final intriguing possibility is that player-directed generation in the style of Galactic Arms Race [10] could be enhanced by combining it with model-based selection such as in [16]. In a game like Galactic Arms Race, the game generates new content based on content players have liked in the past (as evidenced by e.g. using it). This idea works to ensure that new content appearing in the world derives from content that players appreciated.

However, as the number of players climbs higher, the amount of content generated will also climb because player behaviour generally leads to new content spawning in the world. With a relatively small population of players, this dynamic poses few problems because the probability of any player in the game eventually experiencing a reasonable sampling of the diversity of generated content is high. However, with many players, the consequent content explosion means that most players will see only a small fraction of the diversity of

content that the game is able to produce. In that situation, the question arises whether the overall search for content might benefit from trying to expose players to content that they are likely to find interesting. That is, the game might try to *model* the type of content that individual players prefer and thereby avoid wasting effort presenting newly-generated instances to players who are unlikely to be interested in them. If such mismatches occur on a large scale, then pockets of the search space that could have been explored fruitfully might be abandoned simply because the players who would have been interested in such content never had to the opportunity to observe it.

By combining player modelling with player-directed content generation, it is a possible that a synergistic effect could accelerate the search and also produce a more diverse set of content. When players are exposed to candidate content that they are likely to find interesting, their discernment in principle can help to explore the subspace of that particular type of content with more fidelity than would be possible through the overall player population.

Relates directly to challenges: representing style, search space construction.

5 Conclusion

This chapter presents three grand goals for procedural content generation, and presents several challenges that should be addressed in order to realise these goals, and a sample of actionable steps that could get you started towards the challenges. Obviously, these are not the only conceivable actionable steps nor even the only challenges for PCG. We believe PCG presents a rich and fertile soil for research and experimentation into new techniques, with obvious benefits both for industry and for the science of game design.

References

1 Daniel A. Ashlock, Stephen P. Gent, and Kenneth M. Bryden. Evolution of l-systems for compact virtual landscape generation. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 2005.

2 Pippa Avery, Julian Togelius, Elvis Alistar, and Robert Pieter van Leeuwen. Computational intelligence and tower defence games. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 2012.

3 M. Boden. *The creative mind: Myths and mechanisms*. Routledge, 2003.

4 Cameron Browne. *Automatic generation and evaluation of recombination games*. PhD thesis, Queensland University of Technology, 2008.

5 Alex J. Champandard. Procedural characters and the coming animation technology revolution. *AIGameDev.com*, 2012.

6 Michael Cook and Simon Colton. Multi-faceted evolution of simple arcade games. In *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2011.

7 Joris Dormans. Adventures in level design: Generating missions and spaces for action adventure games. In *Proceedings of the FDG Workshop on Procedural Content Generation*, 2010.

8 N. Gillian, S. O'Modhrain, and G. Essl. Scratch-Off: A gesture based mobile music game with tactile feedback. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, Pittsburgh, June 4–6 2009.

9 Ken Hartsook, Alexander Zook, Sauvik Das, and Mark O. Riedl. Toward supporting stories with procedurally generated game worlds. In *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2011.

10 Erin J. Hastings, Ratan K. Guha, and Kenneth 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.

11 Lawrence Johnson, Georgios N. Yannakakis, and Julian Togelius. Cellular Automata for Real-time Generation of Infinite Cave Levels. In *Proceedings of the ACM Foundations of Digital Games*. ACM Press, June 2010.

12 Annika Jordan, Dimitri Scheftelowitsch, Jan Lahni, Jannic Hartwecker, Matthias Kuchem, Mirko Walter-Huber, Nils Vortmeier, Tim Delbrügger, Ümit Güler, Igor Vatolkin, and Mike Preuss. Beatthebeat – music-based procedural content generation in a mobile game. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, 2012.

13 S. Karakovskiy and J. Togelius. The mario ai benchmark and competitions. In *IEEE Transactions on Computational Intelligence and AI in Games*, volume 4, pages 55–67, 2012.

14 Manuel Kerssemakers, Jeppe Tuxen, Julian Togelius, and Georgios Yannakakis. A procedural procedural level generator generator. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, 2012.

15 A. Liapis, G. 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, 2012.

16 Chris Pedersen, Julian Togelius, and Georgios N. Yannakakis. Modeling Player Experience for Content Creation. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(1):54–67, 2010.

17 Martin Pichlmair and Fares Kayali. Levels of sound: On the principles of interactivity in music video games. In Baba Akira, editor, *Situated Play: Proceedings of the 2007 Digital Games Research Association Conference*, pages 424–430, Tokyo, September 2007. The University of Tokyo.

18 Przemyslaw Prusinkiewicz. Graphical applications of l-systems. In *Proceedings of Graphics Interface / Vision Interface*, pages 247–253, 1986.

19 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 *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2012.

20 Jimmy Secretan, Nicholas Beato, David B. D'Ambrosio, Adelein Rodriguez, Adam Campbell, and Kenneth O. Stanley. Picbreeder: Evolving pictures collaboratively online. In *CHI'08: Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1759–1768, New York, NY, USA, 2008. ACM.

21 N. Shaker, G. N. Yannakakis, and J. Togelius. Crowd-sourcing the aesthetics of platform games. *IEEE Transactions on Computational Intelligence and Games, Special Issue on Computational Aesthetics in Games, (to appear)*, 2012.

22 Noor Shaker, Julian Togelius, Georgios N. Yannakakis, Ben Weber, Tomoyuki Shimizu, Tomonori Hashiyama, Nathan Sorenson, Philippe Pasquier, Peter Mawhorter, Glen Takahashi, Gillian Smith, and Robin Baumgarten. The 2010 Mario AI championship: Level generation track. *IEEE Transactions on Computational Intelligence and Games*, 2011.

23 R.M. Smelik, T. Tutenel, K. J. de Kraker, and R. Bidarra. A declarative approach to procedural modeling of virtual worlds. *Computers and Graphics*, 35:352–363, 2011.

24 Adam Smith and Michael Mateas. Answer set programming for procedural content generation: A design space approach. *IEEE Transactions on Computational Intelligence and AI in Games*, 2011.

25 Adam M. Smith and Michael Mateas. Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, 2010.

26 G. Smith, A. Othenin-Girard, J. Whitehead, and N. Wardrip-Fruin. Pcg-based game design: creating endless web. In *Proceedings of the International Conference on the Foundations of Digital Games*, pages 188–195. ACM, 2012.

27 Gillian Smith, Jim Whitehead, and Michael Mateas. Tanagra: Reactive planning and constraint solving for mixed-initiative level design. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):201–215, 2011.

28 Nathan Sorenson and Philippe Pasquier. Towards a generic framework for automated video game level creation. In *EvoApplications (1)*, pages 131–140, 2010.

29 Kenneth 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.

30 Julian Togelius. A procedural critique of deontological reasoning. In *Proceedings of DiGRA*, 2011.

31 Julian Togelius, Mike Preuss, Nicola Beume, Simon Wessing, Johan Hagelbäck, and Georgios N. Yannakakis. Multiobjective exploration of the starcraft map space. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, 2010.

32 Julian Togelius and Jürgen Schmidhuber. An experiment in automatic game design. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG)*, 2008.

33 Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. Search-based procedural content generation: a taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3:172–186, 2011.

34 Mike Treanor, Bryan Blackford, Michael Mateas, and Ian Bogost. Game-o-matic: Generating videogames that represent ideas. In *Proceedings of the FDG Workshop on Procedural Content Generation*, 2012.

35 R Michael Young, Mark O Riedl, Mark Branly, Arnav Jhala, RJ Martin, and CJ Saretto. An architecture for integrating plan-based behavior generation with interactive game environments. *Journal of Game Development*, 1(1):51–70, 2004.

General Video Game Playing

John Levine[1], Clare Bates Congdon[2], Marc Ebner[3], Graham Kendall[4], Simon M. Lucas[4], Risto Miikkulainen[6], Tom Schaul[7], and Tommy Thompson[8]

1 **Department of Computer and Information Sciences, University of Strathclyde**
john.levine@strath.ac.uk
2 **Department of Computer Science, University of Southern Maine**
congdon@usm.maine.edu
3 **Universitat Griefswald, Institut fur Mathematik und Informatik**
marc.ebner@uni-greifswald.de
4 **School Of Computer Science, University of Nottingham**
gxk@cs.nott.ac.uk
5 **School of Computer Science and Electrical Engineering, University of Essex**
sml@essex.ac.uk
6 **Department of Computer Science, University of Texas at Austin**
risto@cs.utexas.edu
7 **Courant Institute of Mathematical Sciences, New York University**
schaul@cims.nyu.edu
8 **School of Computing and Mathematics, University of Derby**
t.thompson@derby.ac.uk

Abstract

One of the grand challenges of AI is to create general intelligence: an agent that can excel at many tasks, not just one. In the area of games, this has given rise to the challenge of General Game Playing (GGP). In GGP, the game (typically a turn-taking board game) is defined declaratively in terms of the logic of the game (what happens when a move is made, how the scoring system works, how the winner is declared, and so on). The AI player then has to work out how to play the game and how to win. In this work, we seek to extend the idea of General Game Playing into the realm of video games, thus forming the area of General Video Game Playing (GVGP). In GVGP, computational agents will be asked to play video games that they have not seen before. At the minimum, the agent will be given the current state of the world and told what actions are applicable. Every game tick the agent will have to decide on its action, and the state will be updated, taking into account the actions of the other agents in the game and the game physics. We envisage running a competition based on GVGP playing, using arcade-style (e.g. similar to Atari 2600) games as our starting point. These games are rich enough to be a formidable challenge to a GVGP agent, without introducing unnecessary complexity. The competition that we envisage could have a number of tracks, based on the form of the state (frame buffer or object model) and whether or not a forward model of action execution is available. We propose that the existing Physical Travelling Salesman (PTSP) software could be extended for our purposes and that a variety of GVGP games could be created in this framework by AI and Games students and other developers. Beyond this, we envisage the development of a Video Game Description Language (VGDL) as a way of concisely specifying video games. For the competition, we see this as being an interesting challenge in terms of deliberative search, machine learning and transfer of existing knowledge into new domains.

1998 ACM Subject Classification I.2.1 Applications and Expert Systems: Games

Keywords and phrases Video games, artificial intelligence, artificial general intelligence

Digital Object Identifier 10.4230/DFU.Vol6.12191.77

Artificial and Computational Intelligence in Games. *Dagstuhl Follow-Ups*, Volume 6, ISBN 978-3-939897-62-0.
Editors: Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius; pp. 77–83

Dagstuhl Publishing
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany

1 Motivation

The field of Artificial Intelligence is primarily concerned with creating agents that can make good decisions [15]. In the context of games, this definition is particularly relevant, as the player of a game will typically have to make a large number of accurate decisions in order to achieve a favourable outcome (a win over an opponent, or a high score). Games have thus always formed a fertile source of benchmarks and challenges for the AI researcher since the earliest days of the field [16].

Great progress has been made with specific games, such as chess, backgammon and poker (to name but three of the many games that AI researchers have effectively mastered). However, each AI program is written to tackle a specific game and the techniques used for one game do not necessarily transfer to other games. Humans can thus still claim to be, in some sense, more intelligent than these programs because they can play multiple games and learn how to play and excel at new ones: their intelligence is more *general*.

In AI, we are always striving for theories concerning intelligence in general rather than intelligence in the specific. If we come up with a theory of good decision making and use a turn-taking game as *an example* of how that theory works, then we have made a better contribution than someone who writes a bespoke computer program just to play that game. In the best case, our theory of intelligence should be able to be applied to any situation in the real world where decisions are taken.

These observations, combined with the success of computer programs at turn-taking games such as chess and Othello, have given rise to the challenge of General Game Playing (GGP) [5], which has now become an annual competition at the AAAI conference [22]. This competition focuses on turn-taking board games. The logic of the game is specified in a language called GDL (Game Description Language) [21] which in turn is inspired by the single agent Planning Domain Definition Language (PDDL) [24] used in the AI planning community.

Turn-taking board games are only one genre of games at which humans excel. A turn-taking board game is usually quite slow-paced, and so deliberative reasoning about the consequences of actions and the likely responses of opponents are possible using seconds or even minutes of CPU time. Humans also excel at other games where decisions have to be made much more quickly and where the environments are noisy, dynamic and not always predictable. Video games are a very good example of this genre and form an interesting challenge for AI researchers.

Classic arcade video games (such as Pac-Man, Asteroids and Space Invaders) are very good examples of the kind of activities we are seeking to cover in this work. In his book *Half-Real* [11], Juul offers a six-part definition of what a game is: (1) games are rule-based; (2) they have variable outcomes; (3) the different outcomes have different values; (4) the player invests effort to influence the outcome; (5) the player is attached to the outcome; and (6) the game has negotiable consequences. Looking specifically at points (3), (4) and (5), our core assumptions in this work are that the value of the outcome achieved by the player of a video game (point 3) is a direct reflection of the amount of intelligence (intellectual effort) brought to bear on the task (point 4); and that this can be further used as an indicator of the player's general level of intelligence (point 5).

In this work, we seek to extend the idea of General Game Playing into the realm of video games, thus forming the area of General Video Game Playing (GVGP). In GVGP, computational agents will be asked to play video games that they have not played before. The agent will have to find out how the game reacts to its actions, how its actions affect

its rewards and therefore how to win the game (or maximise its score). At the minimum, the agent will be given the current state of the world and told what actions are applicable. Every game tick the agent will have to decide on what action to take. The state will then be updated, taking into account the actions of the other players in the game and the game physics. We envisage that the rules and the physics of the game can be encoded using a Video Game Description Language (VGDL). We have analysed a selection of classic arcade games with a view to representing them in VGDL and the results of this work are presented in another paper in this volume.

The aims of this work are:

- to define what General Video Game Playing is and our motivation for this investigation;
- to review some of the previous work that has influenced and inspired this work;
- to discuss what video games we would initially like to tackle in this work;
- to speculate on the computational methods that might be appropriate for GVGP;
- to present a proposal for a GVGP framework, based on existing software for the Physical Travelling Salesman Problem (PTSP);
- to outline our proposal for a GVGP competition.

The remainder of the chapter is structured as follows. In the next section, we look at the background behind GVGP, in particular the GGP competition and the work of the Atari 2600 group at Alberta. We then examine what sort of games would like to tackle in GVGP and the mechanisms that agents might use in order to play these games competently. We then examine how we would build a GVGP framework, based on the Physical Travelling Salesman (PTSP) work at the University of Essex. Finally, the paper ends with a proposal for a GVGP competition, with tracks based on the form of the state (frame buffer or object model) and whether or not a forward model of action execution is available.

2 Background

General video game playing is an excellent tool to test artificial intelligence algorithms. If a human player is sitting in front of a video game and is playing the game, then they need to make observations about the state of the game:

- Where is the avatar of the player (i.e. which screen object is 'me')?
- Where are the opponents or enemies?
- What are the current options (direction of next move, firing of a weapon, jump, etc.)?
- Which option will most likely bring the own avatar towards a particular goal?

One of the main advantages of general video game playing is that it is of intermediate complexity. It is more complex than simple board games, e.g. movement options may be continuous and the game may also employ simulated physics. However, general video game playing is not as complex as developing human-like robots.

In developing human-like robots, researchers have resorted to highly constrained settings. Sample tasks include: vacuuming the floor [10], cleaning windows [4], moving objects or persons from location A to location B [8], or giving tours in a museum [19]. Even though these problems have all been beautifully addressed by these researchers the settings are constrained, i.e. algorithms are usually only transferable to similar domains. In some cutting edge areas of robotics (search for the holy grail of artificial intelligence) the goals are not at all clearly defined. Should a robot be able to play chess, learn to play the piano, wash the dishes or go shopping for groceries? Should a robot first learn to play chess and then how to go shopping or vice versa? That aside, robot actuators and input sensors may not allow

handling of completely different problem classes such as driving on the road and playing chess.

In contrast, in general video game playing, the goal of the player is clearly defined. Moreover, there are several different goals (one for each game). The goals are not taken from an artificial laboratory setting but correspond to actual goals that have been addressed by numerous game playing enthusiasts around the world. Goals include:

- Navigating a yellow object through a maze while collecting all white dots and at the same time avoiding colored ghosts (Pac Man).
- Landing a space ship on the moon (Lunar Lander).
- Navigating a space ship through a field of asteroids (Asteroids).
- Driving a racing car (Pole Position).
- Climbing to the top of a platform environment while avoiding falling objects (Donkey Kong).

While each game has its unique story and goal which should be achieved by the player, the method of how the player interacts with the game is the same across all these games. All possible moves of the player are defined through the game controller (e.g. the Atari 2600 had a joystick with a single button resulting in 18 possible moves for each time step). The number of possible movements for each time step are sufficiently small and discrete. However, given the continuous nature of the game, i.e. a new decision has to be made for each time step, the game is sufficiently complex.

We believe that addressing the diverse problems of general video game playing will allow us to gain new insights in working towards the holy grail of artificial intelligence, i.e. development of human-like intelligence.

2.1 General Game Playing

The General Game Playing competition at AAAI has now been running since 2005 and focuses on turn-taking board games. The programs which enter the competition are not given the details of the games in advance; instead, the rules of the game to be played are specified using GDL, the Game Description Language [21]. Given the unseen nature of the games, there are two common approaches used to construct players, namely deliberative reasoning (e.g. minimax search) or policy learning (e.g. reinforcement learnng). One of the most successful players to date is Cadia Player [2], which is a deliberative approach based on Monte Carlo Tree Search [12, 3].

2.2 The Atari 2600 Games Group

The work described in our paper has a strong connection to the work of the Atari 2600 games group at Alberta, and much of our discussion was influenced by their work [1]. They have built what they call The Arcade Learning Environment (ALE), based on the games for the classic Atari 2600 games console. A variety of different genres of game are available for this console: shooters, adventure, platform, board games, and many others. An emulator for 2600 games is used in the ALE, and the player of these games has to interact with this emulator to find out what action to take next. Two approaches to creating players are investigated, based on reinforcement learning [18] and Monte Carlo Tree Search [12]. Other recent work in this area by Hausknecht *et al.* has investigated the use of evolutionary neural networks for creating general video games players [6].

The aim of the work cited in this section is similar to ours: to create general artificial intelligence agents for playing video games. Our proposal complements this work in two ways:

firstly, by proposing an alternative and potentially more general framework for creating the games, including a Video Game Description Language; and secondly, by proposing a General Video Game Playing competition for evaluating different approaches to this problem.

2.3 Artificial General Intelligence

The field of artificial general intelligence (AGI) addresses the problem of artificial intelligence in its most broad sense. The intelligent agents that it aims to build should be able to (learn to) act well in a very broad range of environments, and under a broad range of constraints. This contrasts with narrower AI research that focuses on a single application domain. A large portion of the recent work in AGI is based on Hutter's theoretical framework [9]. Recently, games were proposed as the specific ingredient for measuring this form of general intelligence [17]; namely, an (infinite) set of unseen games were proposed to play the role of the "broad range of environments".

3 Games for General Video Game Playing

What kind of games are suitable for General Video Game Playing research? Since we want to start our investigations with a feasible challenge, we think that classic 2-dimensional arcade games offer the right balance. If this is successful, we then would envisage moving into 3-dimensional games, which offer more complexity (not least in terms of the effort required to create the game environment). In the companion paper in this volume, we present Video Game Description Language definitions for three diverse arcade games: Space Invaders, Lunar Lander and Frogger. Other suitable candidates might include simple first person shooting games such as Maze War [23], albeit represented in a 2-dimensional form.

All these games are characterised by relatively simple but fast-paced gameplay, with the need for the player to make accurate decisions and adopt high performing strategies in order to win. The emulators for such games can be created more easily than those complex 3-dimensional games but will still allow for sufficient experimentation for the results to be transferrable to more complex game environments.

4 A Competition for General Video Game Playing

One of the main challenges of General Video Game Playing is to create a software framework that allows for games to be designed and representated and different game-playing agents tested via some form of long-running competition. Competitions have been very useful in promoting research in creating AI players of games: examples include the long-running Ms Pac-Man competition [13], the 2K Botprize [7] and the Physical Travelling Salesman Problem [14].

In order to support research in General Video Game Playing we propose using software created for the Physical Travelling Salesman Problem as the basis of a General Video Game Playing framework. In order to do this, we would need to extend the framework to add more agents and more game objects and create a general state model for the system. The framework would be adapted to read 2-dimensional games written in VGDL.

We propose to hold a GVGP competition, in a number of tracks. These tracks will be based on variations in what data is available to the player:

- What data will be provided to the player to characterise the state? A frame buffer or an object model?
- Will the player have access to the entire state or just a first-person perspective?

- Will the player be able to use the simulator as a forward model to be able to do deliberative reasoning?

Since these three features are mutually exclusive, the competition could, in theory at least, run in 8 independent tracks (e.g. frame buffer model, entire state, no forward model). In practice, only some of these variations will be interesting: in particular, the decision to use a the entire state or a first person perspective will be dictated by the genre of game we are playing. However, the use of frame buffer vs object model and the use of a forward model vs no forward model are both interesting dimensions, thus resulting in at least 4 variations for each game.

5 Potential Applications

While creating general video game players is an interesting academic activity in itself, we think that this area of research could find applications in real video games. Our eventual aim would be to enable the creation of high performance AI players without any extra code being written: the game environment would be designed and then handed over to the general AI agent, which would then work out how to play the game competently. In the shorter term, we think that general video game playing techniques could be used both to test game environments, including those created automatically using procedural content generation [20] and to find potential loopholes in the gameplay that a human player could exploit.

References

1 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.

2 Yngvi Björnsson and Hilmar Finnsson. Cadiaplayer: A simulation-based general game player. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(1):4–15, 2009.

3 Cameron Browne, Edward J. Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Trans. Comput. Intellig. and AI in Games*, 4(1):1–43, 2012.

4 M. Farsi, K. Ratcliff, P. J. Sohnson, C. R. Allen, K. Z. Karam, and R. Pawson. Robot control system for window cleaning. In D. A. Chamberlain, editor, *Proceedings of the 11th International Symposium on Automation and Robotics in Construction XI*, pages 617–623. Elsevier Science, 1994.

5 Michael Genesereth and Nathaniel Love. General game playing: Overview of the AAAI competition. *AI Magazine*, 26:62–72, 2005.

6 Matthew Hausknecht, Piyush Khandelwal, Risto Miikkulainen, and Peter Stone. Hyperneat-ggp: A hyperneat-based atari general game player, 2012.

7 Philip Hingston. The 2k botprize. `http://botprize.org/`, 2012. [Online; accessed 27-August-2012].

8 Yuji Hosoda, Masashi Koga, and Kenjiro Yamamoto. Autonomous moving technology for future urban transport. *Hitachi Review*, 60(2):100–105, April 2011.

9 Marcus Hutter. *Universal artificial intelligence: Sequential decisions based on algorithmic probability*. Springer Verlag, 2005.

10 iRobot Corporation. *Roomba. Vacuum Cleaning Robot*. 8 crosby Drive, Bedfard, MA 01730, 2011.

11 Jesper Juul. *Half-Real: Video Games Between Real Rules and Fictional Worlds*. MIT Press, 2005.

12 Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *ECML*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer, 2006.

13 Simon M. Lucas. Ms pac-man competition. `http://cswww.essex.ac.uk/staff/sml/pacman/PacManContest.html`, 2012. [Online; accessed 27-August-2012].

14 Diego Perez, David Robles, and Philipp Rohlfshagen. The physical travelling salesman problem competition. `http://www.ptsp-game.net/`, 2012. [Online; accessed 27-August-2012].

15 Stuart J. Russell and Peter Norvig. *Artificial Intelligence – A Modern Approach (3rd edition)*. Pearson Education, 2010.

16 Jonathan Schaeffer and Jaap van den Herik, editors. *Chips Challenging Champions: Games, Computers, and Artificial Intelligence*. Elsevier Science Inc., New York, NY, USA, 2002.

17 T. Schaul, J. Togelius, and J. Schmidhuber. Measuring intelligence through games. *Arxiv preprint arXiv:1109.1314*, 2011.

18 Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

19 Sebastian Thrun, Maren Bennewitz, Wolfram Burgard, Armin B. Cremers, Frank Delaert, Dieter Fox, Dirk Hähnel, Charles Rosenberg, Nicholas Roy, Jamieson Schulte, and Dirk Schulz. Minerva: A second-generation musuem tour-guide robot. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1999–2005. IEEE, 1999.

20 Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. Search-based procedural content generation. In Cecilia Di Chio, Stefano Cagnoni, Carlos Cotta, Marc Ebner, Anikó Ekárt, Anna Esparcia-Alcázar, Chi Keong Goh, Juan J. Merelo Guervós, Ferrante Neri, Mike Preuss, Julian Togelius, and Georgios N. Yannakakis, editors, *EvoApplications (1)*, volume 6024 of *Lecture Notes in Computer Science*, pages 141–150. Springer, 2010.

21 Wikipedia. Game description language — wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Game_Description_Language`, 2012. [Online; accessed 27-August-2012].

22 Wikipedia. General game playing — wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/General_Game_Playing`, 2012. [Online; accessed 27-August-2012].

23 Wikipedia. Maze war — wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Maze_War`, 2012. [Online; accessed 27-August-2012].

24 Wikipedia. Planning domain definition language — wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Planning_Domain_Definition_Language`, 2012. [Online; accessed 27-August-2012].

Towards a Video Game Description Language

Marc Ebner[1], John Levine[2], Simon M. Lucas[3], Tom Schaul[4], Tommy Thompson[5], and Julian Togelius[6]

1 **Institut für Mathematik und Informatik, Ernst Moritz Arndt Universität Greifswald**
marc.ebner@uni-greifswald.de
2 **Department of Computer and Information Science, University of Strathclyde**
john.levine@strath.ac.uk
3 **School of Computer Science and Electrical Engineering, University of Essex**
sml@essex.ac.uk
4 **Courant Institute of Mathematical Sciences, New York University**
schaul@cims.nyu.edu
5 **School of Computing and Mathematics, University of Derby**
t.thompson@derby.ac.uk
6 **Center for Computer Games Research, IT University of Copenhagen**
julian@togelius.com

Abstract

This chapter is a direct follow-up to the chapter on General Video Game Playing (GVGP). As that group recognised the need to create a Video Game Description Language (VGDL), we formed a group to address that challenge and the results of that group is the current chapter. Unlike the VGDL envisioned in the previous chapter, the language envisioned here is not meant to be supplied to the game-playing agent for automatic reasoning; instead we argue that the agent should learn this from interaction with the system.

The main purpose of the language proposed here is to be able to specify complete video games, so that they could be compiled with a special VGDL compiler. Implementing such a compiler could provide numerous opportunities; users could modify existing games very quickly, or have a library of existing implementations defined within the language (e.g. an Asteroids ship or a Mario avatar) that have pre-existing, parameterised behaviours that can be customised for the users specific purposes. Provided the language is fit for purpose, automatic game creation could be explored further through experimentation with machine learning algorithms, furthering research in game creation and design.

In order for both of these perceived functions to be realised and to ensure it is suitable for a large user base we recognise that the language carries several key requirements. Not only must it be human-readable, but retain the capability to be both expressive and extensible whilst equally simple as it is general. In our preliminary discussions, we sought to define the key requirements and challenges in constructing a new VGDL that will become part of the GVGP process. From this we have proposed an initial design to the semantics of the language and the components required to define a given game. Furthermore, we applied this approach to represent classic games such as Space Invaders, Lunar Lander and Frogger in an attempt to identify potential problems that may come to light. Work is ongoing to realise the potential of the VGDL for the purposes of Procedural Content Generation, Automatic Game Design and Transfer Learning.

1998 ACM Subject Classification I.2.1 Applications and Expert Systems: Games

Keywords and phrases Video games, description language, language construction

Digital Object Identifier 10.4230/DFU.Vol6.12191.85

Artificial and Computational Intelligence in Games. *Dagstuhl Follow-Ups*, Volume 6, ISBN 978-3-939897-62-0.
Editors: Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius; pp. 85–100

Dagstuhl Publishing
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany

1 Motivation

Recent discussions at the Dagstuhl seminar on Artificial and Computational Intelligence in Games addressed the challenge of extending the principles of General Game Playing (GGP) to video games. While the rationale and challenges facing research in General Video Game Playing (GVGP) is the focus of another report in these proceedings, the need for a Game Description Language (GDL) designed to suitably express the key concepts and mechanics inherent in video games became apparent. This has led to this report exploring the challenges and potential pitfalls faced in creating a Video Game Description Language (VGDL) that will work as part of the GVGP process.

The focus of traditional GDLs is to express components expected in the state of a game, and the rules that induce transitions, resulting in a state-action space. Typically, GDLs for use in GGP competitions conform to a set-theoretic language that expresses atomic facts in predicate logic. One of the more popular examples: the GDL component of the Stanford General Game Playing Competition employs this approach to define discrete games of complete information [4]. While this provides means to define and subsequently generate games that conform to this particular set, the language is ill-suited for defining video game environments. This is due to a number of factors:

- Nondeterministic behaviour in video games as a result of non-player characters (NPCs) or elements of chance are common.
- Decisions for actions in video games can be attributed to player input or prescribed NPC behaviour. These decisions are no longer turn-based and may occur simultaneously at any step of the game.
- Video game environments may employ continuous or temporal effects, real-time physics and context-based collisions or interactions between combinations of players and artefacts. Part of the player's task is learning to predict those dynamics.
- Typical video games use much larger environments than board games for the player to interact with. However, the player-environment interactions are sparse, meaning that a part of the environment complexity is rarely or never influencing decision making.

These factors among others present an interesting research challenge in its own right: to try and maintain a balance between simplicity and generality in a description language whilst ensuring it is suitably expressive. Driven by these factors and the challenge they represent, we have raised key goals that shape our aspirations:

- To define a GDL that supports the core mechanics and behaviour expected of classical 2D video games.
- To ensure the VGDL defined will support the work being developed in the GVGP discussion group.
- Define a language that is sufficient not only to represent a game to the human reader, but also for a compiler to generate an instance of the game.
- Provide a VGDL that is unambiguous, extensible and tractable that could provide opportunities for Procedural Content Generation (PCG).

In this report, we consolidate the key points raised by the authors as we explored the challenges and impact of expanding the current research in Game Description Languages for Video Games to further these design goals. We highlight the current state-of-the-art in the field, the core criteria and considerations for the design and subsequent construction of a VGDL, the challenges the problem domain will impose, the potential for future research and collaboration and finally highlight preliminary steps in developing a VGDL prototype.

2 Previous Work

Several attempts have been made in the past to model aspects of both simple arcade games and board games. Most of them had the purpose of being able to generate complete games (through evolutionary computation or constraint satisfaction) but there have also been attempts at modelling games for the purposes of game design assistance.

The Stanford General Game Playing Competition defines its games in a special-purpose language based on first-order logic [4]. This language is capable of modelling a very large space of games, as long as they are perfect information, turn-taking games; those games that can practically be modelled with the Stanford GDL tend to be board games or games that share similar qualities. The language is very verbose: an example definition of Tic-Tac-Toe runs to three pages of code. We do not know of any automatically generated game descriptions using this GDL, but even if they exist, their space is unlikely to be suitable for automatic search, as any change to a game is likely to result in an unplayable and perhaps even semantically inconsistent game.

One noteworthy attempt at describing and evolving board games is Browne's *Ludi* language and the system powered by this language that evolves complete board games [1]. The Ludi system is a high-level language, and expressly limited to a relatively narrow domain of two-player "recombination games", i.e. turn-based board games with a restricted set of boards and pieces, and with perfect information. The primitives of the Ludi GDL are high-level concepts such as "tiling", "players black white" etc. As the language was expressly designed to be used for automatic game generation it is well suited to search, and most syntactically correct variations on existing game descriptions yield playable games.

Also related is the Casanova language developed by Maggiore [6], now an open source project [2]. Casanova is a high-level programming language with particular constructs that make certain aspects of games programming especially straightforward, such as the update/display loop, and efficient collision detection. However, it is significantly lower level than the VGDL developed here and its complex syntax would means that it is unsuitable as it stands for evolving new games.

Moving from the domain of board games to video games that feature continuous or semi-continuous time and space, only a few attempts exist, all of them relatively limited in scope and capabilities. Togelius and Schmidhuber created a system which evolved simple game rules in a very confined rule space [11]. The game ontology featured red, blue and green things (where a thing could turn out to be an enemy, a helper, a power-up or perhaps something else), and a player agent. These could move about in a grid environment and interact. The rules defined the initial number of each thing, the score goal and time limit, how each type of thing moved, and an interaction matrix. The interaction matrix specified when things of different colours collided with each other, and with the player agent; interactions could have effects such as teleporting or killing things or the agent, or increasing and decreasing the score. Using this system, it was possible to generate very simple Pac-Man-like games.

Nelson and Mateas created a system that allowed formalisation of the logic of dynamical games, and which was capable of creating complete *Wario Ware*-style micro-games [9], i.e. simpler than a typical Atari 2600 game. That system has not yet been applied to describe or generate anything more complex than that.

Togelius and Schmidhuber's system inspired Smith and Mateas to create Variations Forever, which features a somewhat expanded search space along the same basic lines, but a radically different representation [10]. The rules are encoded as logic statements, generated using Answer Set Programming (ASP [5]) in response to various specified constraints. For

example, it is possible to ask the generator to generate a game which is winnable by pushing a red thing into a blue thing; if such a game exists in the design space, ASP will find it. Another system that took the same inspiration to another direction is Cook et al.'s ANGELINA, which expands the design space by generating not only rules but also other aspects of the game, particularly levels, concurrently [3]. The Inform system [8] permits specifying text adventure games, coded in natural language. Further, Mahlmann et al.'s work on a Strategy Game Description Language, used for representing and generating turn-based strategy games, should also be mentioned [7].

As far as we are of, no game description language can represent even simple arcade games of the type playable on an Atari 2600 with any fidelity. For example, the physics of Lunar Lander, the destructible bases of Space Invader or the attachment of the frog to the logs in Frogger cannot be modelled by existing GDL's.

Aside from generating rules, there are a number of applications of evolutionary computation to generate other types of content for games; see [12] for a survey.

3 Criteria and Considerations

During our initial discussions, we identified what core criteria would be required for any resulting VGDL. These criteria are instrumental in ensuring the original goals introduced in Section 1:

- **Human-readable:** We want to ensure a simplicity in the structure of the language that ensures a human reader can quickly formulate new definitions or understand existing ones through high level language.
- **Unambiguous, and easy to parse into actual games:** We intend for the software framework attached to the language to be able to instantly generate working prototypes of the defined games through a simple parsing process. The game mechanics and concepts transfer between games and, where possible, between classes of games.
- **Searchable/tractable:** A representation of the game components in a concise tree structure allows for generative approaches like genetic programming, specifically leading to natural cross-over operators.
- **Expressive:** The language must be suitably expressive to represent the core mechanics and objects one expects of classical 2D video games. Most of these aspects are by design encapsulated, simplifying the language and drastically reducing the code required to define them.
- **Extensible:** Many game types, components and dynamics can be added any time, by augmenting the language vocabulary with new encapsulated implementations.
- **Legal & viable for randomly generated games:** all game components and dynamics have 'sensible defaults', which make them as likely as possible to inter-operate well with a large share of randomly specified other components.

These criteria are considered as we move into the subsequent section, in which we discuss how we envisage the structure of the VGDL, and how components necessary to describe video games are expressed.

4 Language Structure

Our discussion focused on the core components required in order to represent a simple video game, this was separated into the following categories:

- **Map:** Defines the 2D layout of the game, and also the initialization of structures as obstacles or invisible entities such as end of screen boundaries and spawn points.
- **Objects:** Objects that exist within the game. These can be different types of entities, such as non-player characters (NPC), structures, collectibles, projectiles, etc.
- **Player Definitions:** Determines the number of players, and which ones are human-controlled.
- **Avatars:** Player-controlled object(s) that exist on the map. When a player passes input to the game, it will have an impact on the state of the avatar.
- **Physics:** Definition of the movement/collision dynamics for all or some object classes.
- **Events:** Control events sent from the player via input devices, timed or random events, and object collision triggered events that affect the game.
- **Rules:** Any event can be specified to lead to game-state changes like object destruction, score changes, another event, etc. The rules also specify the game objectives, and termination.

4.1 Map

The map of the game defines the (initial) two-dimensional layout, in particular the positioning of the obstacles and the starting positions of the objects (including the avatar). Initially, we see maps as string representations, that can be parsed directly into a game state by mapping each ASCII character to an instance of the corresponding object class at its position.

4.2 Objects

The fundamental components of a game are the *objects* that exist in the 2D space. Every object has a set of (x,y) coordinates, a bounding polygon (or radius), and is represented visually on-screen. Objects are part of a hierarchy that is defined by the VGDL and permits the inheritance of properties like physics, or collision effects with a minimal specification. The player-controlled avatar is a specific type of object, and most games dynamics revolve around the object interactions. A permanent static object, such as a wall, collectable (power pills, health packs, coins etc.), portals or spawn/goal location is referred to as a 'structure'. These static objects will not require any update logic for each game tick. Meanwhile dynamic objects, such as avatars, NPCs, elevators and projectiles will specify how they change state on subsequent ticks of the game.

4.3 Physics

The temporal dynamics of non-static objects are called their 'physics', which include aspects like gravitational pull, friction, repulsion forces, bouncing effects, stickiness, etc. Furthermore, more active NPC behaviors like fleeing or chasing are included under this header as well. Those dynamics operate on (specified or default) properties of the objects, like mass, inertia, temperature, fuel, etc. The player control inputs affect the dynamics of the avatar object(s), and those mappings to movement and behavior are encapsulated in the avatar class used (e.g. spaceship, car, frog, pointer).

4.4 Events

We recognise that in many games, there are instances of state-transition that are not the result of an established agent committing actions. As such, we have identified the notion of

an *event*: a circumstance that at an undetermined point in time will result in change to one or more objects on the next time step of the game.

To give a simple example, an NPC's behaviour is defined by its own rationale. As such, at any given time step, the action the NPC will commit is determined by pre-written code that may factor elements from the environment as well as whether the current time step bears any significance in-terms of the NPCs lifespan or any cyclical behaviour. Meanwhile, a human player's actions are not inferred from the entity itself. Rather, it is achieved by the system polling the input devices that are recognised by the game software. In the event that the input devices present a signal, we recognise this as an event, given that this signal will appear at an indeterminable point in time.

We have recognised a range of circumstances we consider events, namely:

- Signals from players via recognised input devices.
- Object collisions (with each other, or with map boundaries).
- Actions occur at specific points of time in the game (e.g. spawning).
- Actions that will occur with a given probability.

For any game we wish to define, we must identify the range of events that can occur at any point. As is seen later in Sections 6, we identify the list of events required for a handful of games.

4.5 Rules

The game engine detects when the bounding boxes of two objects intersect and triggers a collision event. Besides the default effect of ignoring the collision, common effects that could occur include: bouncing off surfaces/objects, destruction of one or both objects, sticking together, teleportation and the spawning of a new object. Outside of game entities, collisions can also refer to the player leaving the 'view' of the game, as such we require rules for recognising instances of scrolling, panning and zooming of the game view. Lastly, collisions can also trigger non-local events, this can affect the current score, end a given level either as a result of success or termination of the players avatar, move the game to the next level or indeed end the game entirely.

5 Representing the Language: Preliminary Syntax

We propose a preliminary syntax as close to human-readable as possible, in order to invite game contributions from as broad an audience as possible, but definitely including game designers and other non-programmers. Our proposed VGDL needs to suitably represent a tree structure, which we propose to encode in Python-like white-space syntax. The other syntax features are arrows (>) representing *mappings*, e.g. from tile characters to objects names, from object names to object properties, or from collision pairs to their effects. Object properties are assigned by terms combining property name, an '=' symbol, and the value. We require naming properties to make the code more readable and options ordering-independent. Example: "Rectangle (height=3, width=12)". Parentheses and commas can be added for readability, but are ignored by the parser.

6 Examples of Classic Games

The preceding sections discussed the purpose, structure and format of the VGDL that the authors have devised. In this section we showcase examples that were conceived during

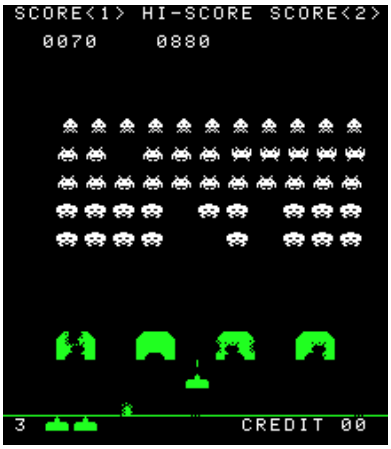

Figure 1 A screenshot of the Space Invaders game [15].

discussion. For this language to work in conjunction with the GVGP research, it is desirable to suitably express the entities and mechanics of games that one would expect to find on the Atari 2600. This section therefore focuses on three games reflective of that period: Space Invaders, Lunar Lander and Frogger; chosen for their dissimilar mechanics and design. These examples were selected for two purposes: to provide context to the design decisions we made during discussion – thus providing definitions that reflect that we had envisaged – and secondly as an attempt to identify potential complications that would need to be resolved. What follows is our attempt to define the entities, physics, collision matrix and event table that reflect the game as succinctly as possible.

6.1 Space Invaders

Space Invaders, originally released in 1978 in arcades and in 1980 for the Atari 2600, is one of the most influential games in the industry both critically and commercially [15]. The mechanics of Space Invaders are relatively straightforward; the player must defend against waves of enemy characters (the space invaders) who are slowly moving down the screen. The enemies begin at the top of the screen and are organised into rows of a fixed size. The group will collectively move in one direction along the x-axis until the outmost invader collides with the edge of the screen, at which point the group moves down a set distance and alternates their horizontal direction. While moving down the screen, enemies will at random open fire and send missiles towards the bottom of the screen. These missiles either eliminate the player upon contact, or can damage one of bunkers situated between the invaders and the player. The player can hide behind these bunkers to strategically avoid being destroyed out by the invaders. In addition to these invaders, a flying saucer can appear at random intervals which traverses along the top of the screen, opening fire on the player below. To retaliate against these threats, the player controls a cannon at the bottom of the screen which can be moved left or right within the bounds of the game world. The player can fire missiles at the invaders that will eliminate them upon contact. The player must eliminate all enemy invaders before they reach the bottom of the screen in order to win. Figure 1 shows a screenshot from the Space Invaders game. As previously described, we can see the player-controlled cannon at the bottom of the screen as the waves of enemies move down in their grouped 'typewriter' formation.

Referring to the initial language structure in Section 4, we identify the core components of Space Invaders as follows:

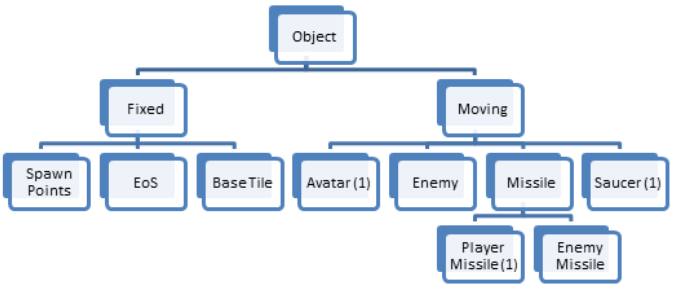

Figure 2 An object hierarchy of the entities found within the Space Invaders game. Note that the Avatar, Player Missile and Saucer have been identified as singletons.

Table 1 The Space Invaders collision matrix, identifying the behaviour of the game in the event that two entities collide in the game.

	Avatar	Base	Enemy Missile	Player Missile	Enemy	End of Screen	Saucer
Avatar			life lost		game over	avatar stops	
Base			base damaged; missile disappears	base damaged; missile disappears	base tile removed		
Enemy Missile				both destroyed		missile destroyed	
Player Missile					destroys enemy	missile destroyed	saucer destroyed
Enemy						moves down, all enemies change dir.	
End of Screen							disappears
Saucer							

Table 2 The event listing for Space Invaders, showcasing not only instances which happen with a certain probability or time, but also the relation of input devices to avatar control.

Event	Action
Joystick left	Avatar left
Joystick right	Avatar right
Joystick button	Avatar fires missile
Time elapsed	Saucer appears
Probability event	Random enemy fires a missile

- **Map:** An empty map save for the tiles that represent the bases the player uses for cover. Locations are annotated for spawn points NPC waves, the human player and the structures that present the boundary of the screen.
- **Objects:** These are separated in 'Fixed' and 'Moving' objects. The former is indicative of spawn points and the tiles that represent a base. Meanwhile, the latter denotes entities such as NPCs and missiles. A hierarchy of these objects is shown in Figure 2.
- **Player Definition:** At present we consider this a one-player game, with the controls identified in Table 2.
- **Avatars:** Player controls one sole avatar, the missle cannon.
- **Physics:** Each moving object in the game moves a fixed distance at each timestep. There are no instances of physics being employed beyond the collisions between objects.
- **Events:** The events of the game can be inferred both from the collision matrix in Table 1 and the event listings in Table 2.
- **Rules:** Rules of the game, such as player scoring, the death of either enemy or player, and winning the game can be inferred from the event table and collision matrix.

We feel that this information, which was gathered as part of our groups discussion, is representative of the Space Invaders game. The object tree in Figure 2 identifies each type of object which is used in the game. From this we can begin to dictate rules for collision as well as assign events to these objects. Note that we have identified some objects as singletons, given that only one of these objects at most should exist in the game world at any given timestep. The collision matrix in Table 1, highlights the behaviour and events that are triggered as a result of objects colliding within the game. Meanwhile the events shown in Table 2 not only shows the actions that the user can have via the avatar, but circumstances such as enemy saucers or missiles that are triggered by time or probability.

6.2 Lunar Lander

Lunar Lander is an arcade game released by Atari in 1979. This game was selected due to its greater emphasis on precision when controlling the avatar, which proved popular at the time of release [14]. Control of the avatar is confounded by gravitational forces and momentum gathered from movement, which is an element that we did not have to consider in Space Invaders. As shown in Figure 3, the player is tasked with piloting the lunar module towards one of the predefined landing platforms. Each platform carries a score multiplier and given the rocky terrain of the moon surface, they are the only safe places to land the module; landing the module in any location other than the pads results in death. The player can rotate the lander within a 180 degree range such that the thrusters of the lander can range from being perpendicular and parallel to the ground. The aft thruster must be used to slow the descent of the lander given the gravitational forces bearing down upon it. However, the player has two further considerations: the control must be applied proportionally to provide adequate strength to the thrusters. Secondly, applying the thrusters will consume the limited fuel supply. The player scores multiples of 50 points for a successful landing, with the multiplier identified on the landing platform. Failed attempts also receive score based on how smoothly the lander descended.

Once again we discussed how the game could be formulated within the proposed VGDL, summarised as follows:

- **Map:** The map file would identify the terrain of the world as structures, with unique identifiers for landing platforms based on their score multipliers. The map will also

Figure 3 A screenshot of Lunar Lander taken from [14] showing the player trying to land on either of the highlighted platforms.

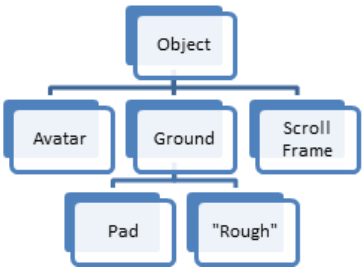

Figure 4 An object hierarchy of the entities found in Lunar Lander, complete with reference to the scroll pane used for viewing the game.

Table 3 Lunar Lander collision matrix.

	Lunar Lander	Landing Pad	Rough	Close to EOS
Lunar Lander		IF upright and low speed then win else ship explodes	ship explodes	scroll/wrap

Table 4 Lunar Lander events.

Event	Action
Joystick Left	Rotate Left Stepwise
Joystick Right	Rotate Right Stepwise
Joystick Up	Increase Thrust
Joystick Down	Decrease Thrust
Landing Altitude Reached	Zoom Scroll Pane

identify where the avatar will spawn in the game world at the beginning of a landing (typically the top-left).

- **Objects:** The object tree, shown in Figure 4 indicates the avatar itself, the ground being separated into landing pad and the 'rough': i.e. all other parts of the terrain. One unique object in this tree is the Scroll Frame. Having played Lunar Lander we recognised the scroll frame's view follows the lander should it move towards the left and right edges of the screen. Furthermore, once the lander has descended to an altitude of approximately 500, the scroll frame zooms closer to the lander to allow for more precise control.
- **Player Definition:** This a one-player game, with controls as shown in the event list in Table 2.
- **Avatars:** The lander module.
- **Physics:** The game must model the gravitational force that exerts upon the lander, and the continuous variables needed to calculate the thrust speed. We would intend for the language's multiple physics models to contain some that represent these phenomena. These models will be parameterised to customise the model for the game.
- **Events:** With exception of the player input, the only event shown in Table 2 is the zooming of the scroll pane once the final descent altitude is reached.
- **Rules:** Rules of the game, such as player scoring, the death of the player, and winning the game can be inferred from the event table and collision matrix.

Again we have encapsulated the core game mechanics to be represented in the VGDL. We believe these definitions can be expressed succinctly within the proposed language. One component of the game that has not been explored as yet is the varying difficulties that Lunar Lander offers to the player. There are four difficulty settings, Training, Cadet, Prime and Command, differ in the strength of gravitational force applied, applying atmospheric friction to the movement of the lander and in some cases added rotational momentum. This could be achieved given our intention to apply parameterised physics models as part of the language.

6.3 Frogger

The last game covered in our initial discussions was Frogger, an arcade game developed by Konami and published by Sega in 1981 [13]. The game's objective is to control a collection of frogs one at a time through a hazardous environment and return to their homes. In order to reach the goal, the player must navigate across multiple lanes of traffic moving at varying speeds, followed by a fast flowing river full of hazards. While the first phase of the trip relies on the player using any available space on the road whilst avoiding traffic, the latter constrains the users movements to using floating logs and turtles – strangely, despite being an amphibian entering the water proves fatal. What proved interesting to players at release and subsequently to the discussion group is the range of hazards that the player must overcome. Each 'lane' of hazards moves independently across the screen at its own speed. Furthermore, some groups of floating turtles in the river would submerge after a set period of time, meaning the player had to quickly move to another safe zone.

Below we highlight the key features of the game that we would implement in the VGDL:

- **Map:** The map identifies where the initial location of the vehicles and logs. Furthermore, it must assign the goal locations, the surrounding screen edge to prevent the player moving off the level and the water hazards.

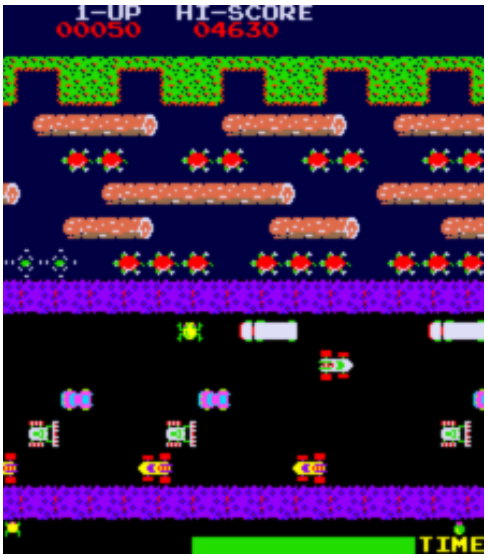

Figure 5 A screen of the Frogger game from [13], with the frog avatar crossing the freeway with only the river of logs and turtles keeping him from the lilypad goals at the top of the screen.

- **Objects:** The object tree in Figure 6 indicates the range of fixed and moving objects that exist in the game. While this tree looks similar to that shown in Figure 2, we create a hierarchy for the floating platforms in the game, using the Log as the supertype. This is of relevance to the collision matrix (Table 5), as we only define the event of colliding with the supertype. This has been introduced as collisions with the subtypes are conditional and driven by context. This issue would be addressed as a rule in the game.
- **Player Definition:** Frogger is a two-player game, however only one player is in control of the frog at any given time. The controls for the player are shown in Table 6.
- **Avatars:** The frog currently on screen.
- **Physics:** Frogger does not adopt any physics models in the movement or behaviour of any objects in game. The game is driven by the collisions between the frog and other objects, be they hazard or otherwise.
- **Events:** Table 6 shows that outside of player controls, we must consider the appearance of a fly, which provides bonus score, and the creation of new moving vehicles or logs on screen.
- **Rules:** Outside of standard scoring and life counts, the rules must carry extra information regarding the collisions that take place in game. While the collisions shown in Table 5 are similar to those shown previously, there are conditions on whether this results in death for the player. Firstly, should the player land on a floating turtle, then it is safe and as the collision matrix shows, the frog will stick to that surface. However should the turtle be submerged then the frog collides with the water, resulting in death for the player. Secondly, it is safe for a frog to collide with a crocodile provided it is not near its mouth. In the event of landing on the mouth, the player dies. In addition, there are rules that define how the player scores: should the frog reach the lily pad having landed on either a fly or female frog, then the score earned has a bonus of 200 points.

The features defined are similar to that which we have seen in the previous two games, but we addressed the unique elements of this game by defining rules for specific contexts, whilst retaining a simplicity for the design. This issue of context-specific collisions is one

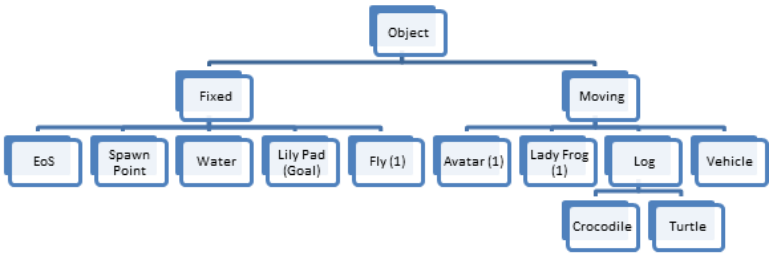

Figure 6 The proposed object hierarchy for the Frogger. Note the separation of fixed and mobile objects used previously in Figure 2. Furthermore, the crocodile and turtle objects have been placed under logs in the hierarchy given that they are also moving platforms for the frog. The difference is they carry conditions that dictate whether the frog has safely landed upon them.

Table 5 The Frogger collision matrix. Note we only consider collisions with a log type, as collisions with the log subtypes are determined at runtime based on their current state.

	Frog	Vehicle	Log	Fly	Goal Pad	Water	EOS
Frog		dies	sticks	frog eats fly	fills position, new frog created	player dies	blocks movement
Vehicle							disappears/wraps
Log							disappears/wraps
Fly							disappears/wraps
Goal Pad							
Water							
EOS							

Table 6 Frogger events.

Event	Action
Joystick Left	Jump Left
Joystick Right	Jump Right
Joystick Up	Jump Up
Joystick Down	Jump Down
Probability Event	Fly Appears
Probability Event	Lady Frog Appears
Time Elapsed	New Logs/Vehicles appear

that must be addressed early on in the language. Games that we would wish to define in the VGDL have collisions with enemies that are dependent upon state and/or location. For example, in the game Mario Bros. the player will be killed should they collide directly with a moving crab. Should the player land on the top of a 'Koopa' tortoise, then the enemy is stunned; while side-on contact proves fatal. Given our efforts on Frogger we believe this will not prove taxing when working in the formal language.

7 Prototype and Roadmap

Upon reaching the end of our time in Dagstuhl we had reached a consensus on what we expected of the Video Game Description Language. This of course is reflected by the concepts and examples shown throughout Sections 3 to 6. At the time of writing, a working prototype of the language has been developed in Python.[1] The language incorporates many of the considerations raised in Section 3 and is expressed in accordance with our expectations set out in Section 4. The language is implemented atop Pygame: a collection of Python modules that are designed to support creation of games.[2] The language compiler can take a user-defined game written in VGDL and construct a working pygame implementation. While a work in progress, the prototype is already capable of constructing games that mimic those discussed throughout our sessions. Figure 7 shows a working implementation of Space Invaders that has been developed using the prototype.

Referring back to the core language components in Section 4 and how we envisaged Space Invaders to be represented using this approach in Section 6.1, the game is currently implemented as follows:

- **Map:** The map is defined separate from the game logic and is shown in Figure 8. As discussed in Section 6.1, we have represented the screen bounds, the avatar, base tiles and spawn points for the aliens. The map definition assigns specific symbols to each of these objects, with the corresponding mapping in the game definition listing in Figure 7 (lines 11-13).
- **Objects:** Game objects are defined in the *SpriteSet* section of Figure 7. Note that upon defining the type of sprite and the class it adheres to, we introduce properties that are relevant to that entity. A simple example is the base tiles which are identified as immovable white objects. Meanwhile the aliens express the probability of fire, their movement speed and the type of behaviour they will exhibit. Note keywords such as 'Bomber and 'Missile' which are pre-defined types in the language. The indentation in the language can be used to denote hierarchical object definitions, e.g. both *sam* and *bomb* inherit the properties from *missile*, which allows statements like in line 22 that indicate the collision effect of any subclass of missile with a *base* object. Also, the *sam* missiles which are launched by the avatar are rightfully declared as singletons.
- **Player Definition & Avatars:** The player is acknowledged in as the avatar (line 3). Furthermore, it is classified as a FlakAvatar, a pre-defined type, that provides a working implementation of the control scheme and movement constraints required for Space Invaders play.
- **Events:** The collision matrix shown in Table 1 has been implemented in the *InteractionSet* within Figure 7. Note that in the majority of instances, the behaviour is to kill the sprites

[1] The source code is open-source (BSD licence), and the repository is located at `https://github.com/schaul/py-vgdl`.
[2] Pygame is freely available from `http://www.pygame.org`.

```
BasicGame
    SpriteSet
        avatar   > FlakAvatar   stype=sam
        base     > Immovable    color=WHITE
        missile  > Missile
            sam  > orientation=UP    color=BLUE singleton=True
            bomb > orientation=DOWN  color=RED  speed=0.5
        alien    > Bomber stype=bomb prob=0.01 cooldown=3 speed=0.75
        portal   > SpawnPoint   stype=alien  delay=16   total=20
    LevelMapping
        0 > base
        1 > avatar
        2 > portal
    TerminationSet
        SpriteCounter   stype=avatar    limit=0 win=False
        MultiSpriteCounter stype1=portal stype2=alien limit=0 win=True
    InteractionSet
        avatar  EOS  > stepBack
        alien   EOS  > turnAround
        missile EOS  > killSprite
        missile base > killSprite
        base missile > killSprite
        base   alien > killSprite
        avatar alien > killSprite
        avatar bomb  > killSprite
        alien  sam   > killSprite
```

Figure 7 A definition of the Space Invaders game from a working prototype of the Video Game Description Language. While a work in progress, this definition satsifies many of the considerations from our discussions in Sections 4 and 6.1.

```
wwwwwwwwwwwwwwwwwwwwwwwwwwwwww
w                            w
w2                           w
w000                         w
w000                         w
w                            w
w                            w
w                            w
w                            w
w    000       000000      000   w
w   00000     00000000    00000  w
w   0   0     00    00    00000  w
w                1               w
wwwwwwwwwwwwwwwwwwwwwwwwwwwwww
```

Figure 8 A representation of the Space Invaders environment. Each character represents either the screen boundary, a component of a bunker base, the player and spawn points for the enemy players. While the game map is constructed here, the relation to the game entities is defined in the *LevelMapping* section of the game definition in Figure 7.

that are involved in the collision. Naturally this would need to handle more complex interactions in future games. Referring back to the event table shown in Table 2, the player inputs have been modelled within the *FlakAvatar* type in the language, meanwhile the probability that drives enemy fire has been declared as a property of the alien entity.

- **Rules:** The *TerminationSet* defines the conditions for winning and losing the game. The game states that having zero avatars will result in failure. Meanwhile should the number of aliens and unused spawn points reach zero,the game will declare the player as winner.

While this is a work in progress, the prototype already provides playable implementations of Space Invaders and Frogger, with definitions of games with continuous physics such as Lunar Lander are in their early stages. It is our intent to continue developing this prototype prior to developing the build that will operate in the GVGP framework. It is encouraging that we have reached this stage within a relatively short period of time since our discussions in Dagstuhl, and we will continue to expand the language until it satisfies our goals.

References

1 C. Browne. Evolutionary game design. *IEEE Transactions on Computational Intelligence and AI in Games*, pages 11–21, 2011.

2 Casanova. Casanova project page. `http://casanova.codeplex.com`, 2012.

3 Michael Cook and Simon Colton. Multi-faceted evolution of simple arcade games. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, 2011.

4 Michael Genesereth and Nathaniel Love. General game playing: Overview of the aaai competition. *AI Magazine*, 26:62–72, 2005.

5 Vladimir Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1):39–54, 2002.

6 Giuseppe Maggiore, Alvise Spanò, Renzo Orsini, Michele Bugliesi, Mohamed Abbadi, and Enrico Steffinlongo. A formal specification for casanova, a language for computer games. In *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems*, pages 287–292. ACM, 2012.

7 T. Mahlmann, J. Togelius, and G. Yannakakis. Towards procedural strategy game generation: Evolving complementary unit types. *Applications of Evolutionary Computation*, pages 93–102, 2011.

8 Graham Nelson. *The Inform Designer's Manual*. Placet Solutions, 2001.

9 Mark Nelson and Michael Mateas. Towards automated game design. In *Procedings of the 10th Congress of the Italian Association for Artificial Intelligence*, 2007.

10 Adam M. Smith and Michael Mateas. Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, 2010.

11 Julian Togelius and Jürgen Schmidhuber. An experiment in automatic game design. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG)*, 2008.

12 Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. Search-based procedural content generation: a taxonomy and survey. *IEEE Transactions on Computational Intelligence and Games*, 3:172–186, 2011.

13 Wikipedia. Frogger. `http://en.wikipedia.org/w/index.php?title=Frogger`, 2012.

14 Wikipedia. Lunar lander. `http://en.wikipedia.org/w/index.php?title=Lunar_lander_(arcade_game)`, 2012.

15 Wikipedia. Space invaders. `http://en.wikipedia.org/w/index.php?title=Space_Invaders`, 2012.

Artificial and Computational Intelligence for Games on Mobile Platforms

Clare Bates Congdon[1], Philip Hingston[2], and Graham Kendall[3]

1 **Department of Computer Science, The University of Southern Maine, USA**
congdon@usm.maine.edu
2 **School of Computer and Security Science, Edith Cowan University, Australia**
p.hingston@ecu.edu.au
3 **School of Computer Science, University of Nottingham, UK and Malaysia**
graham.kendall@nottingham.ac.uk

Abstract

In this chapter, we consider the possibilities of creating new and innovative games that are targeted for mobile devices, such as smart phones and tablets, and that showcase AI (Artificial Intelligence) and CI (Computational Intelligence) approaches. Such games might take advantage of the sensors and facilities that are not available on other platforms, or might simply rely on the "app culture" to facilitate getting the games into users' hands. While these games might be profitable in themselves, our focus is on the benefits and challenges of developing AI and CI games for mobile devices.

1998 ACM Subject Classification I.2.m Artificial Intelligence, miscellaneous

Keywords and phrases Games, mobile, artificial intelligence, computational intelligence

Digital Object Identifier 10.4230/DFU.Vol6.12191.101

1 Introduction

Games are an appealing application to showcase AI (Artificial Intelligence) and CI (Computational Intelligence) approaches because they are popular and ubiquitous, attracting a diverse range of users.

Mobile games are easier to bring to market than commercial (large scale) video games. This makes them a practical choice for development and study in an academic environment, using relatively small teams of academics and students, who are able to work on relatively low budgets. For example, the small screen size and lack of powerful graphics hardware typical of mobile devices means that simple graphics, often only 2 or 2.5 inches, are expected, so that large teams of highly skilled artists and 3D modellers are not required.

Mobile devices usually provide a wider variety of input data (touch, location, images, video, sound, acceleration, orientation, personal data, data from/about other users etc.) than is normally available on a desktop or laptop computer and offer a full range of output options (images, video, animation, sound, vibration, wireless, bluetooth, infrared) as well. In addition, the popularity of mobile devices allows developers to recruit large numbers of casual users, whose interactions provide another potentially large data source for game data mining, using techniques such as those described in [6]. Novel game mechanics and interaction methods might be made possible by processing these input data using AI and CI methodologies.

Computational power and battery life present two potential obstacles to intensive AI/CI-based games, and some potential designs will require offloading some of the computation

Artificial and Computational Intelligence in Games. *Dagstuhl Follow-Ups*, Volume 6, ISBN 978-3-939897-62-0.
Editors: Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius; pp. 101–108

Dagstuhl Publishing
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany

to servers. It might also be difficult to implement large-scale, complex game worlds due to the limited resources that are available. There are also significant challenges in developing AI/CI libraries that can work with low memory, limited battery power etc., adapting or developing AI/CI methods to work effectively in games that are played in short bursts, using unreliable communications, and providing real-time responses. However, these constraints provide significant research opportunities.

Mobile devices are still "young" enough to provide opportunities for developers to implement innovative products without having to employ large specialist teams (e.g. graphic designers, musicians etc.), although some specialists are still required of course. However, devices are becoming more capable – for example, the original iPhone had a screen resolution of 480x320 pixels, a single 2 Megapixel still camera, and a storage capacity of 4–8 GB, while the iPhone 5 is 1136x640 pixels, has two 8 Megapixel cameras and can record 1080p HD video at 30 fps, has a storage capacity of 16–64 GB, and has in-built voice recognition. Applications are also becoming more sophisticated – for example, technologies like Web3D and game engines like Unity3D are bringing 3D graphics to mobile platforms [5]. Inevitably, game players will come to expect more and more from mobile games, so the opportunity for small players and enthusiasts will not last long (perhaps several years, in our estimation). Those who are interested in this area might want to explore, and capitalize, on those opportunities now. Moreover, CI/AI both provide significant opportunities both in terms of research challenges and also to make the games more interesting and more fun to play. We would like to see the research community take up the challenge to showcase what can be done with the limited resources available on mobile devices, but also utilizing the larger number of sensors (e.g. movement detection) and other options (e.g. location awareness) which are not available on traditional "living room" game consoles.

The aim of this chapter is to outline the limitations of mobile computing, with respect to utilizing AI/CI, but also draw out some of the potential advantages that could be exploited now (or certainly in the years to come) as the convergence of technology continues and offers greater opportunities than are available at present.

The rest of the chapter is presented as follows. In the next section, we present the (limited) work that has been carried out on AI/CI for mobile devices. In Section 3 we lay out what we believe are the defining characteristics of mobile environments. In Section 4 we outline the challenges faced when using mobile devices. Section 5 presents the opportunities that arise when using a mobile device, rather than a desktop, console, or other *stationary* computer. In Section 6 we provide some insight as to what AI/CI can offer mobile computation. We also outline some possible projects that would be feasible at this time, as well as some thoughts as to what might be possible in the next 5–10 years. Section 7 concludes the chapter.

2 Prior Work

We were able to find only a limited amount of work that considers AI/CI in mobile games and there seems to be limited scientific literature about using AI/CI on mobile devices at all. In this section, we summarize the few papers we did find, on AI/CI for games as well as for non-games on mobile devices.

In one gaming example, Aiolli and Palazi [1] adapted an existing machine learning algorithm to enable it to work within the reduced computational resources of a mobile phone. Their target was the game "Die guten und die bösen Geister", which is a board game requiring the player to identify which game pieces (Ghosts) are "good" and which are "bad". Therefore, an AI opponent for the game would need to be able to perform a simple

classification task. The more usual classification algorithms were rejected on the basis of requiring too much memory or too much computation. Instead the authors opted for a very simple system based on two prototype feature vectors, one for good and one for bad ghosts. Unfortunately, they did not report any comparison of performance of this simple scheme over more complex classifiers, but the point is that for such applications, there is a trade-off to evaluate between accuracy and computational resource requirements. There was also no evaluation of the different schemes in terms of player satisfaction.

In a more recent example by Jordan et al. [10], the authors report on a research prototytpe *BeatTheBeat*, in which game levels are matched to background music tracks based on features extracted from the audio signal, and these are allocated to cells on a game board using a self-organising map.

In a paper discussing the potential uses of AI methods in serious mobile game, Xin [13] suggests that, while AI methods could add value to such games, computational requirements might require a client-server solution, offloading the AI to a server.

Although not focusing on games, Kruger and Malaka [11] argue that AI has a role in solving many of the challenges of mobile applications, including

- Location awareness
- Context awareness
- Interaction metaphors and interaction devices for mobile systems
- Smart user interfaces for mobile systems
- Situation adapted user interfaces

This paper introduces a special issue of the journal Applied Artificial Intelligence containing articles describing the state of the art as it was in 2004. Many of these same challenges may provide opportunities for novel mobile game concepts based on AI/CI.

In [2], Baltes et al. describe their experience with implementing high-level real-time AI tasks such as vision, planning and learning for small robots, using smart phones to provide sensing, communication and computation. Although their aims are different from ours, many of the research challenges in terms of implementing AI solutions with limited computational resources are similar. Their robots' agent architectures are based on behaviour trees described using a XML schema, and translated off-line into efficient C code. Vision is based on fast approximate region detection. A standard algorithm was found to be too slow and was modified to take advantage of specific domain knowledge (e.g. expected object colors). Another high-level task that they tackled was multi-agent simultaneous location and mapping (SLAM). Once again, the task was simplified by taking advantage of the structured environment (robot soccer). BlueTooth was used to share information between agents. A particle filter method was used to maintain estimates of the robots' poses, with a limited particle population size dictated by the available memory. We see that the researchers used a variety of strategies to cope with the limitations of the computing platform: offline pre-processing, modification and simplification of algorithms for specific tasks and environments, and sharing of information between mobile devices. We expect that some of the same strategies and even some of the same algorithms will be applicable in both the robotics and games domains.

3 Characteristics of a Mobile Environment

Our working definition of a mobile device for game playing is a device that is networked, and is small enough to be mobile, yet still provides a platform for general computation. In the future, one might imagine that many kinds of mobile devices might be used in games.

For example, a car's GPS system might be used in a futuristic version of a scavenger hunt car rally (scavenger hunt games for mobile phones already exist – e.g. SCVNGR, textClues). However, at the present time, we are chiefly thinking of smart phones and tablets.

While computational resources (CPU, memory, persistent storage) are available on these devices, they are all limited in comparison to standard platforms, and limited battery power is an additional (and major) consideration.

On the plus side, these devices usually have a number of other features that are often not available, and especially not all together, on "standard" gaming platforms:

- **location services** – whether by GPS, WiFi or cell tower triangulation;
- **personal ownership** – generally one person is more or less the sole user of a particular device.
- **Internet access** – to data, services and other users;
- **multiple modes of connectivity** – WiFi, Bluetooth, 3G/4G may be provided, and it is expected that connectivity will not be continuously available.
- **a range of non-standard sensors** – touch screen, camera (for image capture and subsequent processing), microphone will probably be provided, and others may be, such as a gyroscope, accelerometer and video camera;
- **non-standard outputs** – a small screen, some sound, possibly vibration.
- **other app data** – apps may be able to share data, especially with social media platforms.

Also, usage patterns for these devices are often different from those on standard game platforms such as PCs – games are often played in short bursts (waiting for a meeting, on a bus/train etc.), and gameplay may be interruptible.

In designing and implementing games for mobile devices, these differences combine both to provide challenges, which AI and CI have the potential to solve, and to provide opportunities for novel game concepts based on or supported by AI and CI methods.

4 Challenges When Using AI/CI on Mobile Devices

Mobile devices introduce a number of constraints to game design:

- Limited CPU and memory in some ways harken back to the days of early video games.
- Small screen size limits graphical complexity.
- The reality that these devices are typically used when running on a battery further encourages limiting CPU and memory usage beyond what is physically available on these devices.
- However, real-time responses are often called for with mobile devices.
- Connectivity issues must be kept in mind, as devices may lose signal either while out of range of a cell tower or due to a user opting not to pay for wi-fi access at a given location.

It is our thinking that these challenges provide interesting constraints when designing AI/CI-based games, as will be discussed in the next section.

5 Opportunities When Using AI/CI on Mobile Devices

There are some limitations to using mobile devices for gaming (such as small screen size, limited battery life, less powerful processors etc.) but there are also many opportunities for utilizing mobile devices, which are not present on static devices. We briefly mentioned some of these in the introduction, but in this section we discuss these opportunities in a little more detail.

5.1 Small Screen

Having a smaller screen could be seen as a limitation but it could also be viewed as an opportunity. Having limited graphic capabilities means that the programmers may not have to focus as much on this aspect of the system as would be the case if you were designing a system that had a large screen, high resolution and a powerful graphics processor to assist with the processing required in rendering the screen (although screen resolutions are improving and mobile phone GPUs are becoming more powerful). If a programmer's (or researcher's) skills are in AI/CI, then having a platform which is relatively easy to program could be an advantage as you are able to focus on the AI/CI, without having to be so concerned about the graphics. This may also reduce the need for artists on the project team. Of course, as technology continues to develop, the advantages that we outline here will gradually diminish, and the quality of graphics and art will become a higher priority.

5.2 Location Awareness

A static computer, by its nature, is stationary, and this could be seen as one of its major limitations. A gaming device that is able to be in different geographical locations at different times, opens up a range of possibilities that were not available even a few years ago. It is obvious that having devices that can be moved around offers many opportunities but the focus of this chapter is to look at those opportunities from an AI/CI point of view. AI/CI could be utilized in a variety of ways. As the player roams around the game (both physically and within the game world) the AI/CI agent could tailor the game playing experience to meet the expectations of the players.

5.3 Interaction with Other Players

Having a capability such as Bluetooth provides opportunities to meet with other players that are in a similar location, but you were not aware that they were there. This would be useful in locations such as a city center but imagine how many people are potentially with a few feet of you at a sporting event or a concert. Once the application had identified potential game 'buddies' the AI/CI could be used to validate the other person's skill level, whether they are actually a match for you to play with etc. A lot of innovation in gameplay is taking place in the mobile market. A couple of examples are Fingle (a bit like the classic ice-breaking game, Twister, but for hands on a tablet – `http://fingleforipad.com/`) and Swordfight (an example of a Phone-to-Phone Mobile Motion Game [15]).

5.4 Social Media

Mobile platforms already take advantage of the many social platforms that are available. Facebook and Twitter are probably the most well known but there are hundreds, if not thousands, of other platforms that offer users the ability to communicate with one another. Indeed many people, we suspect, use their phone more for texting and updating their status rather than for making phone calls. If a networked, mobile platform is used for game playing, users might want to update their various social networking sites with the games they are playing, their progress, their high scores, who they are playing with etc. This could place a burden on the user who does not have the time to disseminate all this information, but still wishes it to be known. AI/CI could be used to learn when/what the user wishes to update to social networking sites. For example, a user might always tweet a new high score, but not update their facebook page. Another user might keep a certain person regularly

updated about their progress through a game via social media messages aimed at just that user. The challenge is to learn what to update and when, and provide the API (Applications Programming Interface) to the various social media feeds, many of which already exist.

5.5 AI/CI Libraries for Use in Mobile Games

The limited CPU and memory resources typically available on mobile devices suggest the need for AI and CI libraries specifically designed for mobile applications. Two approaches come to mind. Firstly, for applications that require execution on the device itself, stripped down and simplified implementations of common algorithms would be useful. On the other hand, for applications where a client-server model is appropriate, cloud or web service based implementations would be a good solution.

In the academic literature, we could not find any examples of the first kind of any substance. However, there are many examples of small libraries from the open-source community that could provide a good starting point. Many of these examples are implemented in Lua, a scripting-like language with object-oriented capabilities that is commonly used for games. Some examples are Abalhas, which is a PSO implementation in Lua (by Alexandre Erwin Ittner, available at `http://ittner.github.com/abelhas/`), LuaFuzzy, a fuzzy logic library written in Lua (`http://luaforge.net/projects/luafuzzy/`) and LuaFann, a fast artificial neural net implementation (`http://luaforge.net/projects/luafann`). One could perhaps envisage a collection of small, modular library components, written in Lua, and covering these AI and CI technologies, along with others such as evolutionary algorithms, a Lua version of OpenSteer, an A^* implementation, a lightweight rule-based system library perhaps based on CLIPS, and so on.

Of course, this is only one possible development path. For example, web-based development using JavaScript in conjunction with native code, as discussed by Charland et al [4] is another possibility. There are also existing open-source AI and CI codes, such as JMLR MLOSS (`http://jmlr.csail.mit.edu/mloss/`), implemented in various languages such as C++, Java or Python. While there may be issues such as size and portability to overcome, much of this could also be utilised : we point out the Lua pathway as one that might work particularly well for mobile games.

There are also examples of cloud-based implementations of AI and CI technologies that might be utilised in a client-server approach for mobile games. For example, there is Merelo et al.'s cloud-based evolutionary algorithm [7], Li's cloud-based fuzzy system [12] and Haqquni et al.'s cloud-based neural network system [9]. Apple's SIRI is an example where local processing is combined with higher performance cloud-based processing to solve an AI problem – speech recognition.

6 What Can AI/CI Offer for Games on Mobile Devices

6.1 Procedural Content Generation

Using AI/CI methods for Procedural Content Generation (PCG) in games is an active research area with some notable successes in recent years. Spore is one high-profile example in the commercial arena. We argue that several factors make mobile games well suited for PCG. Firstly, in terms of typical length of play sessions and complexity of typical game environments, mobile games are smaller in scale than many standard video games. This should mean that PCG is achievable with limited computational resources, and could be done locally on the device, without having to offload the task to a server. Second, some

of the more interesting AI/CI methods for PCG make use of player preferences, either in real-time or offline. Mobile games with many players would have a ready source for the training data needed to drive these systems.

For example, Interactive Evolutionary Computation (IEC) is a CI technique that could be very well suited for mobile games. Hastings et al. have applied this technique successfully in *Galactic Arms Race* [8]. This game features weapons defined by particle systems controlled by a kind of neural network called a *Compositional Pattern Producing Network*, and these are evolved using *cgNEAT*, a version of *Neuro-Evolution by Augmenting Topologies*, where fitness is determined by popularity of player choices in the game. The authors coined the term *collaborative content evolution* to describe this approach.

The mobile game-playing population could provide an ideal environment for collaborative content evolution, with a large pool of players, playing many short game sessions, providing a very large number of judgements to feed into fitness calculations. Crowd-sourcing used in this way should enable content to evolve rapidly, giving game players a constantly novel, changing game experience, guided by the preferences of the players themselves.

6.2 Personalisation and Customisation

Recently, CI techniques are being used to adapt gameplay to optimise player satisfaction in real time. For example, Yannakakis and Hallam reported success with using neural network based user models adjusted in real time to improve player satisfaction in "playware" games [14]. Using the kind of lightweight libraries proposed in 5.5, this kind of gameplay adaptation and other customisation could be added to mobile games, and neural networks and other machine learning methods have already been proven to be effective for adaptation in other, non-mobile games.

6.3 Ubiquitous Games etc.

The terms *ubiquitous or pervasive* computing have been in use for some time now. As far back as 2002, these terms were also applied to games (see e.g. [3]). There's obviously a considerable overlap between these kinds of games and mobile games – mobile devices provide the means of achieving ubiquity/pervasiveness. A related concept is that of the *augmented reality game*. Here too, modern mobile devices have the camera, audio, and display capabilities to support augmented reality applications. For ubiquitous games, real-time adaptation with CI algorithms running on the device, could be combined with periodic sychronisation with a cloud-based repository, so that the learned personal profile can be shared across locations and devices. For augmented reality games, either a generic light-weight augmented reality library or perhaps some application specific implementation in the style of Baltes et al. [2], could be used.

7 Conclusions

Mobile platforms are already widespread and their use is largely for interacting with social media sites and for tweeting. Some people also use them for what they were originally designed for, making phone calls. Game playing is becoming more widespread on these devices, more so on phones than tablets, with around a third of mobile phone owners reportedly playing mobile games (see, for example `http://www.infosolutionsgroup.com/popcapmobile2012.pdf`). Computational Intelligence and Artificial Intelligence are not often present in these games, or if present, are unsophisticated. However, there is a window of opportunity where we are

able to integrate these technologies into these games, with less of the now usual overhead of having to work with graphic designers, musicians, plot design etc. As mobile platforms develop the complex, large teams associated with console based game design are likely to converge such that it may be more difficult, if not impossible, to enter this market.

In this chapter, we have outlined some of the opportunities and challenges in introducing CI/AI onto mobile platforms. We hope that the research community will take up the many research challenges that exist in this exciting, fast moving area.

References

1 F. Aiolli and C. E. Palazzi. Enhancing Artificial Intelligence on a Real Mobile Game. *International Journal of Computer Games Technology*, 2009.

2 Jacky Baltes and John Anderson. Complex AI on Small Embedded Systems: Humanoid Robotics using Mobile Phones. In *AAAI Spring Symposium Series*, 2010.

3 Staffan Björk, Jussi Holopainen, Peter Ljungstrand, and Regan Mandryk. Special issue on ubiquitous games. *Personal and Ubiquitous Computing*, 6:358–361, 2002.

4 Andre Charland and Brian Leroux. Mobile application development: web vs. native. *Commun. ACM*, 54(5):49–53, May 2011.

5 Kevin Curran and Ciaran George. The future of web and mobile game development. *International Journal of Cloud Computing and Services Science (IJ-CLOSER)*, 1(1):25–34, 2012.

6 Anders Drachen, Christian Thurau, Julian Togelius, Georgios N Yannakakis, and Christian Bauckhage. Game data mining. In *Game Analytics*, pages 205–253. Springer London, 2013.

7 Juan Julián Merelo Guervós, Maribel García Arenas, Antonio Miguel Mora, Pedro A. Castillo, Gustavo Romero, and Juan Luís Jiménez Laredo. Cloud-based evolutionary algorithms: An algorithmic study. *CoRR*, abs/1105.6205, 2011.

8 Erin J. Hastings, Ratan K. Guha, and Kenneth 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.

9 A.A. Huqqani, Xin Li, P.P. Beran, and E. Schikuta. N2Cloud: Cloud based neural network simulation application. In *2010 International Joint Conference on Neural Networks (IJCNN'10)*, pages 1–5, 2010.

10 A. Jordan, D. Scheftelowitsch, J. Lahni, J. Hartwecker, M. Kuchem, M. Walter-Huber, N. Vortmeier, T. Delbrugger, U. Guler, I. Vatolkin, and M. Preuss. BeatTheBeat music-based procedural content generation in a mobile game. In *2012 IEEE Conference on Computational Intelligence and Games (CIG'12)*, pages 320–327, 2012.

11 A. Kruger and R. Malaka. Artificial Intelligence Goes Mobile. *Applied Artificial Intelligence: An International Journal*, 18(6):469–476, 2004.

12 Zhiyong Li, Yong Wang, Jianping Yu, Youjia Zhang, and Xu Li. A novel cloud-based fuzzy self-adaptive ant colony system. In *2008 Fourth International Conference on Natural Computation (ICNC'08) – Volume 07*, pages 460–465, Washington, DC, USA, 2008. IEEE Computer Society.

13 Chen Xin. Artificial Intelligence Application in Mobile Phone Serious Game. In *First International Workshop on Education Technology and Computer Science (ETCS'09)*, volume 2, pages 1093–1095, 2009.

14 G.N. Yannakakis and J. Hallam. Real-time game adaptation for optimizing player satisfaction. *IEEE Trans. on Computational Intelligence and AI in Games*, 1(2):121–133, 2009.

15 Zengbin Zhang, David Chu, Xiaomeng Chen, and Thomas Moscibroda. Swordfight: enabling a new class of phone-to-phone action games on commodity phones. In *MobiSys*, pages 1–14, 2012.