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.

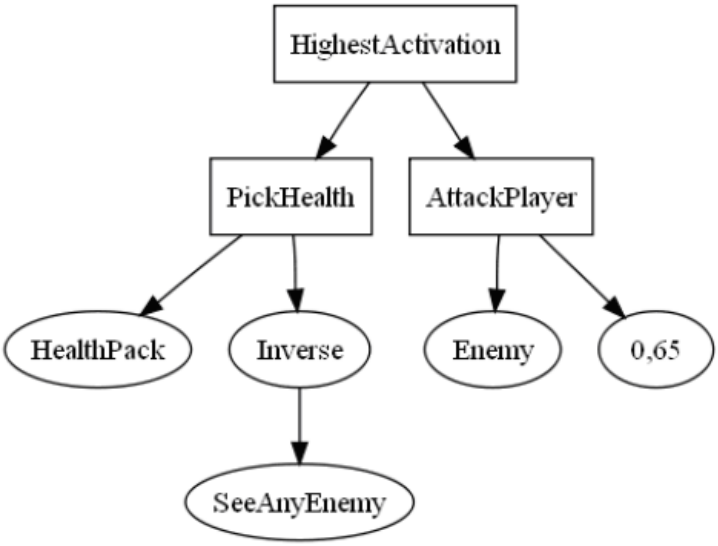

Figure 2 Behavior trees of best individuals for the DeathMatch experiment (courtesy of Rudolph Kadlec (2008)). The bot exhibits two types of behaviors. The bot attacks the enemy player (node AttackPlayer) with the priority fixed to 0.65 or it picks health packs (node PickHealth). The priority of PickHealth 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.

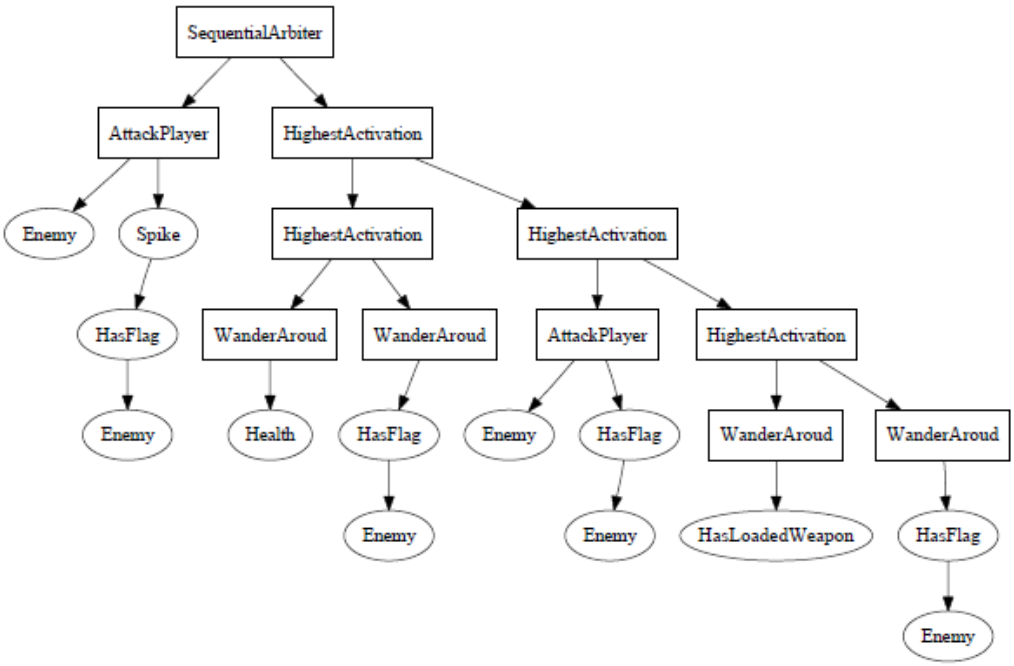

Figure 3 Behavior trees of best individuals for the Capture the Flag experiment (courtesy of Rudolph Kadlec (2008)). Although the tree looks complex the resulting behavior is simple: the bot wanders around (moves randomly around the map) if it does not see enemy flag carrier or shoots the enemy player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