BIRKBECK COLLEGE

MSc COMPUTER SCIENCE PROJECT PROPOSAL

Reinforcement Learning and Video Games: Implementing a Platformer AI with Evolutionary Methods

Author:
Monty WEST

Supervisor:
Dr. George MAGOULAS

Abstract

Placeholder.

Contents

1 Introduction

Artificial intelligence (AI) is a core tenant of video games, traditionally utilised as adversaries or opponents to human players. Likewise, game playing has long been a staple of AI research. However, academic research has traditionally focused mostly on board and card games and advances in game AI and academic AI have largely remained distinct.

The primary focus of game AI is enhance the experience and entertain. Investing time and resources into advanced AI research is infeasible and wasteful when simpler systems act well (if not perfectly). Furthermore, the games industry is driven by users, most of whom are not interested in advanced AI techniques.[1]

The first video game opponents were simple rule-based, discrete algorithms, such as the computer paddle in *Pong*. In the late 1970s video game AIs became more advanced, utilising search algorithms and reacting to user input. In *Pacman*, the ghost displayed distinct personalities and worked together against the human player [2]. In the mid 90s, Finite State Machines (FSMs) emerged as a dominant game AI technique, as seen in games like *Half-Life* [3]. Later, in the 2000s, Behaviour Trees gained preeminence, as seen in games such as *F.E.A.R.* [4] and *Halo 2* [5]. These later advances borrowed little from contemporary development in academic AI and remained localised to the gaming industry.

However, with increases in processing power and the complexity of games over the last ten years many academic techniques have been harnessed by developers. For example, Monte Carlo Tree Search techniques developed for use in Go AI research has been used in *Total War: Rome II* [6] and in 2008's *Left 4 Dead*, Player Modelling was used to alter play experience for different users [7, p. 10]. Furthermore, AI and related techniques are no longer only being used as adversaries. There has been a rise in intelligent Procedural Content Generation in games in recent years, in both a game-world sense (for example *MineCraft* and *Terraria*) and also a story sense (*Skyrim's* Radiant Quest System) [8].

Moreover, games have recently enjoyed more consideration in academic research. Commercial games such as *Ms. Pac Man*, *Starcraft*, *Unreal Tournament* and *Super Mario Bros.* and open-source games like *TORCS* [24] and *Cellz* [10] have been at the centre of recent competitions and papers [11] [12].

These competitions are the forefront of research and development into reinforcement learning techniques in video games, and will be explored in more detail in section 2.3.

The aim of this project is to explore the topic of reinforcement learning in video games. This will be realised through the implementation of a game-playing AI.

2 Background

Reinforcement learning has long been a staple of academic research into AI and Dynamic Programming, especially in robotics and board games. However, it has also had success in more niche problems, such as helicopter control [15] and human-computer dialogue [16].

2.1 Concept Definitions

At this point it is useful to introduce some high level descriptions/definitions of some concepts key to this project.

2.1.1 Game Specific

Rules Sets   [TODO] [Is this needed?]

Behaviour Trees (BTs)   Behaviour Trees are a construct which encodes progressively more specific actions. From the top of the tree broad behaviours are broken down in to subtrees. BTs are executed by traversing the tree and executing nodes.

Nodes of the tress can either be *control* nodes or *leaf* nodes. *Control* nodes affect how their children will be executed, for example a **Sequence** node asserts that it's children be executed in order from left to right (akin to AND) and a **Selector** node executed children in order from left to right until one succeeds (akin to OR). *Leaf* nodes can be **Conditions**, which succeed if the game state passes the condition and **Actions**, which carry out a set of moves or decisions. [32]

2.1.2 Learning

Online/Offline

Offline An offline (or batch) learner trains on an entire dataset before applying changes.

Online A online learner reacts/learns from data immediately after each datapoint.

(reference?)

Reinforcement Learning   A reinforcement learning agent focuses on a learning problem, with it's goal to maximise *reward*. Given a current *state* the agent chooses an *action* available to it, which is determined by a *policy*. This action maps the current *state* to a new *state*. This *transition* is then evaluated for it's *reward* . This *reward* often affects the *policy* of future iterations, but *policies* may be stochastic to some level. [13, s. 1.3]

Genetic Algorithms (GAs)   Genetic Algorithms are an subset of Evolutionary Methods and model the solution as a *population* of *individuals*. Each *individual* has a set of *chromosomes*, which can be thought of as simple pieces of analogous information (most often in the form of bit strings). Each *individual* is assessed by some *fitness function*. This assessment is used to cull the *population*, akin to survival of the fittest. Then a new *population* is created (possibly containing the fittest from the previous *population*) using *crossover* of *chromosomes* from two (or more) *individuals* (akin to sexual reproduction), *mutation* of *chromosomes* from one *individual* (akin to asexual reproduction) and *re-ordering* of *chromosomes*. Each new *population* is called a *generation*. [14, p. 7]

Grammatical Evolution (GE)   The solution in Grammatical Evolution is a program or program fragment. This program is described by a context-free grammar. The search space of the problem consists of integer strings, which are normally evolved using a GA. These integer strings encode a program tree using the context-free grammar. Decoding starts with the start symbol or expression and continues with the left most nonterminal. Imagining each symbol has an ordered list of possible choices, the next integer in the string is calculated modulo the length of the choice list, this value is then the index of the symbol's replacement. This continues until the grammar is in a terminal state. [32]

2.2   Reinforcement Learning and Commercial Games

Desirability

Ventures in utilising reinforcement learning in commercial video games have been limited and largely ineffectual. However, there are many reasons why good execution of these techniques is desirable. Firstly, modern games have large and diverse player bases, having a game that can respond and personalise to a specific player can help cater to all. Secondly, learning algorithms produce AI that can respond well in new situations (over say FSMs or discrete logic), hence making new content easy to produce or generate. Lastly, humans must learn and react to environments and scenarios during games. Having non-playable characters do the same may produce a more believable, immersive and relatable AI, which is one of the key criticisms with current games. [11, p. 7, p. 13]

Issues

The main issue with constructing effectual learning (or learnt) AI in game is time and money. Game development works on strict cycles and have limited resources to invest into AI research. Furthermore, one player playing one game produces a very small data set, making learning from the player

challenging. Moreover, AI that is believably human is a field still in it's infancy. [17]

2.3 Reinforcement Learning and Game AI Competitions

Despite the lack of commercial success, video games can act as great benchmark for Reinforcement Learning AI. They are designed to challenge humans, and therefore will challenge AI methods; games generally have some level of learning curve associated with playing them (as a human); games mostly have some notion of scoring suitable for a fitness function and they are generally accessible to students, academic and the general public alike. [11, p. 9] [12, p. 1] [21, p. 2]

Over the last few years several game based AI competitions have begun, over a variety of genres. These competitions challenge entrants to create an agent that plays a game and is rated according to the competitions specification. They have attracted both academic [21, p. 2] and media interest [12, p. 2]. Hence, several interesting papers concerning the application of Reinforcement Learning in video games have recently been published. Approaches tend to vary widely, modelling and tackling the problem very differently and combining and specialising techniques in previously unseen ways. [21, p. 11]

[Include part about evolutionary learning and game from D. Perez]

Some brief details of the competitions which are of relevance to this project are compiled in to Table 1. The Mario AI Competition is also explored in more detail below.

2.3.1 The Mario AI Competition

The Mario AI Competition, organised by Sergey Karakovskiy and Julian Togelius, ran between 2009-2012 and used an adapted version of the opensource game Infinite Mario Bros. From 2010 onwards the competition was split into four distinct 'tracks'. We shall focus on the unseen Gameplay track, where agents play several unseen levels as Mario with the aim to finish the level (and score highly). [12] [21]

Infinite Mario Bros. Infinite Mario Bros (IMB) [28] is an open-source clone of Super Mario Bros. 3, created by Markus Persson. The core gameplay is described as a *Platformer*. The game is viewed side-on with a 2D perspective. Players control Mario and travel left to right in an attempt to reach the end of the level (and maximise score). The screen shows a short section of the level, with Mario centred. Mario must navigating terrain and avoid enemies and pits. To do this Mario can move left and right, jump, duck and speed up. Mario also exists in 3 different states, *small*, *big* and *fire* (the latter of which enables Mario to shoot fireballs), accessed by finding

Genre	Game	Description
The Simulated Car Racing Competition		
Racing	TORCS (Open-source) [24]	Competitors enter drivers, that undergo races against other entrants which include qualifying and multi-car racing. The competition encourages the use of learning techniques (but doesn't ban non-learning agents). [25]
The 2K BotPrize		
First Person Shooter (FPS)	Unreal Tournament 2004	Competitors enter 'bots' that play a multi-player game against a mix of other bots and humans. Entrants are judged on Turing test basis, where a panel of judges attempt to identify the human players. [26]
The Starcraft AI Competition		
Real Time Strategy (RTS)	Starcraft	Competitors play against each other in a 1 on 1 knockout style tournament. Implementing an agent involves solving both micro objectives, such as path-planning, and macro objectives, such as base progression. [27]
The Mario AI Competition		
Platformer	Infinite Mario Bros (Open-source)	Competitors submit agents that attempt to play (as a human would) or create levels. The competition is split into 'tracks', including Gameplay, Learning, Turing and Level Generation. In Gameplay, each agent must play unseen levels, earning a score, which is compared to other entrants. [21]

Table 1: This table summarises some recent game AI competitions [23]

powerups. Touching an enemy (in most cases) reverts Mario to a previous state. Mario dies if he touches an enemy in the *small* state or falls into a pit, at which point the level ends. Score is affected by how many coins Mario has collected, how many enemies he has killed (by jumping on them or by using fireballs or shells) and how quickly he has completed the level. [21, p. 3]

Suitability to Reinforcement Learning   The competitions adaptation of IMB (known henceforth as the 'benchmark') incorporates a tuneable level generator and allows for the game to sped-up. This makes it a great testbed for reinforcement learning. The ability to learn from large sets of diverse data makes learning a much more effective technique. [21, p. 3]

Besides that, the Mario benchmark presents an interesting challenge for reinforcement learning algorithms. Despite only a limited view of the "world" at any one time the state and observable space is still of quite high-dimension. Though not to the same extent, so too is the action space. Any combination of five key presses per timestep gives a action space of 2^5 [21, p. 3]. Hence part of the problem when implementing a learning algorithm

for the Mario benchmark is reducing these search spaces. This has the topic of papers by Handa [29] and Ross and Bagnell [30].

Lastly, there is a considerable learning curve associated with Mario. The simplest levels could easily be solved by agents hard coded to jump when they reach an obstruction, whereas difficult levels require complex and varied behaviour. For example, traversing a series of pits may require a well placed series of jumps, or passing a group of enemies may require careful timing. Furthermore, considerations such as score, or the need to backtrack from a dead-end greatly increase the complexity of the problem. [21, p. 3, p. 12]

2.4 Agent-based Approaches

2.4.1 Overview

2.4.2 REALM

The REALM agent, developed by Slawomir Bojarski and Clare Bates Congdon, was the winner of the 2010 Mario AI competition, in both the unseen and learning Gameplay tracks. REALM stands for **R**ule Based **E**volutionary Computation **A**gent that **L**earns to Play **M**ario. REALM went through two versions (V1 and V2), with the second being the agent submitted to the 2010 competition.

The REALM agent utilised a Genetic Algorithm to evolve a rule set. In V1 these rules mapped conditions directly to key presses each time step. V2 adopted a two tier system. First, rules mapped conditions high-level behaviour. Second, the desired behaviour and the current environment was modelled as a search problem to produce the required key-presses. [31]

The ruleset developed from REALM V2 was entered into the 2010 unseen Gameplay track. It not only scored the highest overall score, but also highest number of kills and was never disqualified (by getting stuck in a dead-end). Competition organisers note that REALM dealt with the more difficult levels better than other entrants. [21, p. 10]

3 Aim and Objectives

3.1 Aim

The aim of the project is to explore the use of reinforcement learning techniques in creating a game playing AI. This will be achieved by producing an agent that plays the Mario AI benchmark. The project will pose this as a learning problem and not as a search or planning problem.

3.2 Objectives

1. **Benchmark Software**
   Prepare the available Mario AI benchmark software for use in this project.

2. **Observation Space**
   Design, implement and test a suitable strategy for representing the sensory information available from the benchmark into a manageable form.

3. **Handcrafted Agent**
   Design, implement and test a customisable handcrafted (non-learning) agent for the benchmark that utilises the strategy from Objective 2.

4. **Learning Framework**
   Integrate a Reinforcement Learning framework into the software.

5. **Agent Learning**
   Explore use of learning to construct a procedure that evolves an agent using the hand-crafted version created in Objective 3 as a template.

6. **Agent Extension**
   Investigate expansion of the agent template into a two-tiered system, separating high-level and low-level behaviour.

7. **Agent Evaluation**
   Evaluate the fitness of the agent produced and the effectiveness of the learning process.

3.3 Limitations

Due to the limitations of this project, it is unlikely that the final agent will be able to compete at the level of the Mario AI competition. Expansions and explorations into the template functionality of the agent will focus on increasing it's learning ability rather than it's final fitness.

[Note about Objective 6 being a stretch goal]

4 Methodology

The project will attempt to customise and expand on the REALM V1 agent, with influence from other entrants to the 2010 Mario AI Competition, such as D. Perez et al. It will adhere to the following stipulations:

- The agent will follow the rules of the 2010 Mario AI competition, which ban the use of reflection and limit access to classes within the software.

- As the project is concerned primarily with learning concepts there will be no use of search algorithms such as A*.

- No simulation of the game engine to predict enemy movements.

The agent will utilise offline reinforcement learning in the form of genetic programming. The evolution of the agent, and it's ability to play unseen levels will form the basis for it's evaluation.

4.1 Objective 1: Benchmark Software

The intention is that Scala will be the main language of this project. The Mario AI benchmark is coded in Java. Given that Scala is backwards compatible with Java there should not be a significant time penalty in integrating these two languages. In the event that this proves impossible or counter productive the codebase will be updated to Java 8 instead.

Moreover, build tools such as *sbt* or *Maven* and testing frameworks such as *Scala Mock* or *Mockito* will be used.

4.2 Objective 2: Observation Space

As this project will adhere to the rules of the 2010 Mario AI competition, information about Mario and his surroundings is restricted and prescribed by the software.

4.2.1 Mario Benchmark Sensory API

The benchmark provides the *Environment* interface. This acts as the source of sensory information for the agent. The rules of the competition state that this is to be the only source of game information available to agents. It includes:

- A 22x22 receptive field, representing the game screen, which encodes whether or not an enemy, a block or terrain is present in that square.

- A list of enemy positions (on the visible screen) with pixel resolution.

- State information about Mario, e.g. current state (*small*, *big*, *fire*), is on the ground, is able to jump etc.

Inspiration

E. Speed's entry into the 2009 Mario Ai Competition demonstrated the importance of translating sensory information. His agent attempted to use the entire 22x22 grid as the observation space for his learning algorithm, which lead to his agent running out of memory during the competition. [22, pp. 6-7]

REALM incorporated a simple but effective technique of translating the information available to a set of binary variables (and one ternary variable). For example, these included MAY_MARIO_JUMP, IS_PIT_AHEAD and IS_ENEMY_CLOSE_LOWER_RIGHT. This distilled the most important information in a way that significantly reduced the observation space, allowing for a more effective learning process. [31, p. 85]

Approach

The approach of transforming the *Environment* interface into a manageable set of conditions will be adopted in this project. Due to the importance of this step, and it's effect over not only the core functionality of the agent, but also the learning process, Objective 2 may be revisited often, especially during the Objectives 3, 5 and 6.

4.3   Objective 3: Handcrafted Agent

Once again it is important to look at the benchmark software, which provides a framework for creating agents.

4.3.1   Mario Benchmark Agent API

Creating an agent constitutes implementing the *Agent* interface, the core of which is the *getAction* method.

```
public interface Agent {
      boolean[] getAction();
      void integrateObservation(Environment environment);
      ...
}
```

The returned boolean array maps to key presses (in array order): [←] - Move left, [→] - Move right, [↓] - Duck, [**A**] - Jump (if possible), [**B**] - Run (if combined with moving left or right) and/or shoot fireball (when in *fire* state). Any combination of these is allowed.

The *integrateObservation* method is entry point for the sensory information for the agent.

The benchmark runs at constant 25 frames per second (fps), which allows 40ms for the agent controller to decide on an action each time step.

Inspiration

REALM implemented their core agent as a ruleset, with rules stating a preference over conditions, resulting in an action. Rule preference over binary conditions are

either TRUE, FALSE or DONT_CARE [31, p. 85]. Each time step a rule is chosen that best fits the current condition, with ties being settled by rule order [31, p. 86]. Actions in the first version were simply key-presses, whereas in version two they are high-level plans. These plans were passed to a simulator, which reassessed the environment and used A* to produce the key-press combination. This was done in part to reduce the search space of the learning algorithm. [31, pp. 85-87]

Approach

The handcraft AI will follow the ruleset approach. Rules will map conditions (as determined in Objective 2) to explicit key-press combinations. Hence, the agent will be single tiered, where the result of following a rule is the result of the *getAction* method. Furthermore, rulesets will be handled abstractly by the *Agent* interface. The details of a ruleset will be held externally to the interface in order to allow for customisation in the learning phase.

The translation of the observable space to the set of conditions will be implemented into the *integrateObservation* method.

The aim of this handcrafted agent is not to perform well, but to offer a baseline agent for the learning process and for later comparison. With this in mind only minimal time will be taken in constructing the rules for the handcrafted agent, with more focus being aimed at the abstract implementation.

4.4   Objective 4: Learning Framework

This project will focus on evolutionary methods as it's learning procedure. The integration of a learning framework must consider what is available in terms of external libraries and also what is offered from the benchmark software.

4.4.1   Libraries

There are several Genetic Programming libraries available for Java (and therefore Scala). This project will consider the use of two: ECJ [34] and JGAP [35]. The decision on which to use will be made on ease of inclusion and on features pertinent to this project.

4.4.2   Mario Benchmark

The Mario benchmark contains several useful features to encourage the use of learning techniques, below are the two most important.

Level Playing   The benchmark enables agents to play levels, returning important information about an agents performance on completion or death. This process can be automated and from a reinforcement learning perspective, form the basis of agent fitness.

The software also allows for the reliance on the system clock to be turned off, as well as the rendering of the GUI. This allows for levels to be run through several thousands times faster than otherwise. This is essential for implementing an effective learning strategy, as high numbers of playthroughs take minimal time. [21, p. 3]

Tuneable Level Generator Another essential element for effective learning is level generation. Without the reliance on human designed levels many different levels can be learnt from. The benchmark expands on the level generator from IMB, making it tuneable by over 20 parameters, including:

Seed Allows levels to be recreated.

Difficulty Controls the complexity of the level, e.g. pit size, height change.

Creatures Controls presence and numbers of specific enemies.

Length Controls the length of the level.

This not only allows for different agents to learn from the same data set, useful for evaluation and comparison (a draw back from truly random generation), it also allows agents to learn of a particular flavour of level, e.g. short difficult levels with many enemies. [21, p. 4-5]

Approach

4.5   Objective 5: Agent Learning

4.6   Objective 6: Agent Extension

4.7   Objective 7: Agent Evaluation

4.7.1   Mario AI Competition Scoring

The scoring mechanism is included in the Mario AI Benchmark software. The 2009 variant is contained in the *CompetitionScore* class and the 2010 variant in the *GamePlayTrack* class. Each scorer takes in a seed to initialise the levels used.

In 2009, the Gameplay track was scored by agents playing 40 unseen levels with total distance travelled being the competition score. Tie-breakers were settled on game-time, number of kills, Mario's state at the end of each level.

The 2010 competition's primarily addition was increased difficultly of levels (in terms of complexity, pit size, number of enemies etc.), which may include dead-ends (where the agent must back-track to continue). It also increased the number of levels played to 512.

Approach

The agent will be evaluated in the style of the unseen Gameplay track of the 2010 Mario AI Competition. After the learning, the best individual will chosen to be evaluated by the *GamePlayTrack* class. Here the agent will play 512 unseen levels (generated by a random seed). This will allow to compare against the results of the 2010 Competition (available from [21]), although the levels played will be different.

As the evaluation class allows multiple agents evaluation over the same levels (by passing the same seed) further comparisons can be drawn against other available agents:

- The hand-crafted base-line agent constructed as Objective 3. This will show whether learning has a significant effect on the proficiency of the full agent.

- The benchmark software includes several example agents, including a simple, hard-coded agent called *ForwardJumpingAgent*, which was used for similar comparisons in the 2009 competition. [22]

- Baumgarten's A* agent, which contains no learning and was the winner in 2009. It is open-source and available online [36].

The agent will also be assessed by the increase in fitness over time during learning. A steeper increase in fitness shows an effective learning process.

5   Timeline

Work will begin on June 10th, and run until the 14th of September, when the report will be submitted. The writing of the report will be a continual process over the 96 days.

June 10th – July 1st

1 day Integration of the existing Java codebase with the Scala language.

10 days Rule set interpretation testing and implementation, including the creation of a hand-made agent.

10 days Evolutionary Methods library integration and learning set-up.

July 1st – September 7th

- Decision on conditions to be used for rule sets and their translation from the *Environment* interface.

- Experimentation and assessment of evolutionary methods to evolve rulesets. This may include reassessing conditions.

- Possible expansions as spelled out in subsection **??**.

August 24th – September 7th

- Evaluation of agent(s) as specified in subsection **??**.

September 7th – September 14th

- Finishing touches to the project report.

References

[1] Siyuan Xu, *History of AI design in video games and its development in RTS games*, Department of Interactive Media & Game development, Worcester Polytechnic Institute, USA, `https://sites.google.com/site/myangelcafe/articles/history_ai`.

[2] Chad Birch, *Understanding Pac-Man Ghost Behaviour*, `http://gameinternals.com/post/2072558330/understanding-pac-man-ghost-behavior`, 2010.

[3] Alex J. Champandard, *The AI From Half-Like's SDK in Retrospective*, `http://aigamedev.com/open/article/halflife-sdk/`, 2008.

[4] Tommy Thompson, *Facing Your Fear*, `http://t2thompson.com/2014/03/02/facing-your-fear/`, 2014.

[5] Damian Isla, *GDC 2005 Proceeding: Handling Complexity in the Halo 2 AI*, `http://www.gamasutra.com/view/feature/130663/gdc_2005_proceeding_handling_.php`, 2005.

[6] Alex J. Champandard, *Monte-Carlo Tree Search in TOTAL WAR: Rome II Campaign AI*, `http://aigamedev.com/open/coverage/mcts-rome-ii/`, 2014.

[7] Georgios N. Yannakakis, Pieter Spronck, Daniele Loiacono and Elisabeth Andre, *Player Modelling*, `http://yannakakis.net/wp-content/uploads/2013/08/pm_submitted_final.pdf`, 2013.

[8] Matt Bertz, *The Technology Behind The Elder Scrolls V: Skyrim*, `http://www.gameinformer.com/games/the_elder_scrolls_v_skyrim/b/xbox360/archive/2011/01/17/the-technology-behind-elder-scrolls-v-skyrim.aspx`, 2011.

[9] *The Berkeley Overmind Project*, University of Berkeley, California. `http://overmind.cs.berkeley.edu/`.

[10] Simon M. Lucas, *Cellz: A simple dynamical game for testing evolutionary algorithms*, Department of Computer Science, University of Essex, Colchester, Essex, UK, `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.96.5068&rep=rep1&type=pdf`.

[11] G. N. Yannakakis and J. Togelius, *A Panorama of Artificial and Computational Intelligence in Games*, IEEE Transactions on Computational Intelligence and AI in Games, `http://yannakakis.net/wp-content/uploads/2014/07/panorama_submitted.pdf`, 2014.

[12] Julian Togelius, Noor Shaker, Sergey Karakovskiy and Georgios N. Yannakakis, *The Mario AI Championship 2009-2012*, AI Magazine 34 (3), pp. 89-92, `http://noorshaker.com/docs/theMarioAI.pdf`, 2013.

[13] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction* The MIT Press, Cambridge, Massachusetts, London, England, Available: `http://webdocs.cs.ualberta.ca/~sutton/book/ebook/the-book.html`, 1998.

[14] Melanie Mitchell, *An Introduction to Genetic Algorithms*, Cambridge, MA: MIT Press, 1996.

[15] Andrew Y. Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger and Eric Liang, *Inverted autonomous helicopter flight via reinforcement learning*, International Symposium on Experimental Robotics, `http://www.robotics.stanford.edu/~ang/papers/iser04-invertedflight.pdf`, 2004.

[16] S. Singh, D. Litman, M. Kearns and M. Walker, *Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJFun System*, Journal of Artificial Intelligence Research (JAIR), Volume 16, pages 105-133, 2002, `http://web.eecs.umich.edu/~baveja/Papers/RLDSjair.pdf`.

[17] Alex J. Champandard, *Making Designers Obsolete? Evolution in Game Design*, `http://aigamedev.com/open/interview/evolution-in-cityconquest/`, 2012.

[18] James Wexler, *A look at the smarts behind Lionhead Studio?s ?Black and White? and where it can and will go in the future*, University of Rochester, Rochester, NY 14627, `http://www.cs.rochester.edu/~brown/242/assts/termprojs/games.pdf`, 2002.

[19] Thore Graepal (Ralf Herbrich, Mykel Kockenderfer, David Stern, Phil Trelford), *Learning to Play: Machine Learning in Games*, Applied Games Group, Microsoft Research Cambridge, `http://www.admin.cam.ac.uk/offices/research/documents/local/events/downloads/tm/06_ThoreGraepel.pdf`.

[20] *Drivatar^{TM} in Forza Motorsport*, `http://research.microsoft.com/en-us/projects/drivatar/forza.aspx`.

[21] Sergey Karakovskiy and Julian Togelius, *Mario AI Benchmark and Competitions*, `http://julian.togelius.com/Karakovskiy2012The.pdf`, 2012.

[22] Julian Togelius, Sergey Karakovskiy and Robin Baumgarten, *The 2009 Mario AI Competition*, `http://julian.togelius.com/Togelius2010The.pdf`, 2010.

[23] Julian Togelius, *How to run a successful game-based AI competition*, `http://julian.togelius.com/Togelius2014How.pdf`, 2014.

[24] *TORCS: The Open Racing Car Simulation*, `http://torcs.sourceforge.net/`.

[25] Daniele Loiacono, Pier Luca Lanzi, Julian Togelius, Enrique Onieva, David A. Pelta, Martin V. Butz, Thies D. Lnneker, Luigi Cardamone, Diego Perez, Yago Sez, Mike Preuss, and Jan Quadflieg, *The 2009 Simulated Car Racing Championship*, IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES, VOL. 2, NO. 2, 2010.

[26] *The 2K BotPrize*, `http://botprize.org/`

[27] Michael Buro, David Churchill, *Real-Time Strategy Game Competitions*, Association for the Advancement of Artificial Intelligence, AI Magazine, pp. 106-108, `https://skatgame.net/mburo/ps/aaai-competition-report-2012.pdf`, 2012.

[28] *Infinite Mario Bros.*, Created by Markus Perrson, `http://www.pcmariogames.com/infinite-mario.php`.

[29] H. Handa, *Dimensionality reduction of scene and enemy information in mario*, Proceedings of the IEEE Congress on Evolutionary Computation, 2011.

[30] S. Ross and J. A. Bagnell, *Efficient reductions for imitation learning*, International Conference on Artificial Intelligence and Statistics (AISTATS), 2010.

[31] Slawomir Bojarski and Clare Bates Congdon, *REALM: A Rule-Based Evolutionary Computation Agent that Learns to Play Mario*, 2010 IEEE Conference on Computational Intelligence and Games (CIG?10) pp. 83-90.

[32] D. Perez, M. Nicolau, M. O?Neill, and A. Brabazon, *Evolving Behaviour Trees for the Mario AI Competition Using Grammatical Evolution*, Proceedings of EvoApps, 2010, pp. 123?132.

[33] E. R. Speed, *Evolving a mario agent using cuckoo search and softmax heuristics*, Proceedings of the IEEE Consumer Electronics Society?s Games Innovations Conference (ICE-GIC), 2010, pp. 1?7.

[34] *ECJ: A Java-based Evolutionary Computation Research System*, `https://cs.gmu.edu/~eclab/projects/ecj/`.

[35] *JGAP: Java Genetic Algorithms Package*, `http://jgap.sourceforge.net/`.

[36] Robin Baumgarten, *A* Mario Agent*, `https://github.com/jumoel/mario-astar-robinbaumgarten`.