Birkbeck College

Msc Computer Science Project Proposal

Reinforcement Learning and Video Games: Implementing a Evolutionary Agent

Author:
Monty West

Supervisor:
Dr. George Magoulas

MSc Computer Science project proposal, Department of Computer Science and Information Systems, Birkbeck College, University of London 2015

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 are deemed to act proficiently. [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, approaches became more 'agent' based. 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* [27] 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 agents in video games. This will be realised through the implementation of an agent-based game-playing AI, which presents an interesting reinforcement learning challenge.

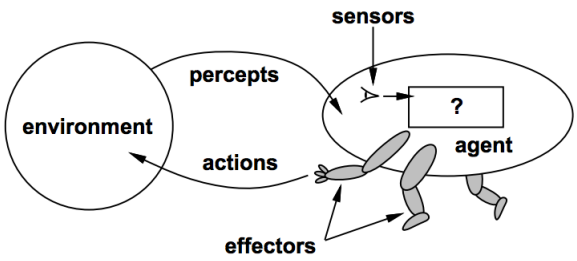

Figure 1: Illustration of an intelligent agent, taking from [15, p. 32]

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 [18] and human-computer dialogue [19].

Similarly the intelligent agent model is a popular approach to AI problems. It is seen in commercial applications, as mentioned above, as well as academic applications, such as board game AI.

The agent model's autonomous nature makes it particularly suited to utilising reinforcement learning.

2.1 Concept Definitions

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

2.1.1 Intelligent Agents (IAs)

An intelligent agent is an entity that uses *sensors* to perceive it's *environment* and acts based on that perception through *actuators* or *effectors*. In software, this is often realised as an autonomous program or module that takes it's perception of the *environment* as input an returns *actions* as output. [14, p. 34]

Figure 1 shows the basic structure of an intelligent agent.

2.1.2 Rule-based systems

A rule-based system decides *actions* from *inputs* as prescribed by a *ruleset* or *rule base*. A *semantic reasoner* is used to manage to the relationship between input and the ruleset. This follows a *match-resolve-act* cycle, which first finds all rules matching an input, chooses one based on a conflict strategy and then uses the rule to act on the input, usually in the form of an output. [16, pp. 28-29]

2.1.3 Behaviour Trees (BTs)

Behaviour Trees are a construct which encodes tiered behaviour. From the top of the tree broad behaviours are broken down in to subtrees, which pertain to specific actions. 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. [35]

2.1.4 Online/Offline Learning

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]

2.1.5 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]

2.1.6 Genetic Algorithms (GAs)

Genetic Algorithms are an subset of evolutionary algorithms 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 *reordering* of *chromosomes*. Each new *population* is called a *generation*. [17, p. 7]

2.2 Reinforcement Learning Agents 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 agents 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) agent AI in game is risk versus reward. Game development works on strict cycles and there are limited resources to invest into AI research, especially if the outcome is uncertain. 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. [20]

2.3 Reinforcement Learning Agents and Game AI Competitions

Despite the lack of commercial success, video games can act as great benchmark for reinforcement learning agents. They are designed to challenge humans, and therefore will challenge learning methods. Games naturally have some level of learning curve associated with playing them (as a human). Also, games require quick reactions to stimulus, something not true of traditional AI challenges such as board games. Most games have some notion of scoring suitable for a fitness function.Lastly, they are generally accessible to students, academic and the general public alike. [11, p. 9] [12, p. 1] [24, p. 2]

Over the last few years several game based AI competitions have begun, over a variety of genres. These competitions challenge entrants to implement an agent that plays a game and is rated according to the competitions specification. They have attracted both academic [24, p. 2] and media interest [12, p. 2]. The competition tend to encourage the use of learning techniques. Hence, several interesting papers concerning the application of reinforcement learning agents in video games have recently been published. Approaches tend to vary widely, modelling and tackling the problem very

Genre	Game	Description
Racing	TORCS (Open-source) [27]	**The Simulated Car Racing Competition** Competitors enter agent drivers, that undergo races against other entrants which include qualifying and multi-car racing. The competition encourages the use of learning techniques. [28]
First Person Shooter (FPS)	Unreal Tournament 2004	**The 2K BotPrize** 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. [29]
Real Time Strategy (RTS)	Starcraft	**The Starcraft AI Competition** Agents 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. [30]
Platformer	Infinite Mario Bros (Open-source)	**The Mario AI Competition** 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. [24]

Table 1: This table summarises some recent game AI competitions [26]

differently and combining and specialising techniques in previously unseen ways. [24, p. 11]

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 open-source 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] [24]

Infinite Mario Bros. Infinite Mario Bros (IMB) [31] 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 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. [24, 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. [24, 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 [24, 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 and Ross and Bagnell [33] separately addressed this issue in their papers [32] and [33] respectively.

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. [24, p. 3, p. 12]

2.4 Previous Learning Agent Approaches

Agent-based AI approaches in commercial games tend to focus on finite state machines, behaviour trees and rulesets, with no learning component. Learning agents are more prevalent in AI competitions and academia, where it is not only encouraged, but viewed as an interesting research topic [12, p. 1]. Examples from both standpoints are compiled in Table 2.

2.4.1 Evolutionary Algorithms

Evolutionary algorithms are a common choice of reinforcement learning methods used in game-playing agents. D. Perez et al. note in their pa-

per [35, p. 1] that they are particular suitable for game environments:

> *'Their stochastic nature, along with tunable high- or low-level representations, contribute to the discovery of non-obvious solutions, while their population-based nature can contribute to adaptability, particularly in dynamic environments.'*

The evolutionary approach has been used across several genres of video games. For example, *neuroevolution*, a technique that evolves neural networks, was used in both a racing game agent (by L. Cardamone [28, p. 137]) and a FPS agent (by the UT∧2 team [29]). Perhaps the most popular approach was to use genetic algorithms (GAs) to evolve a more traditional game AI agent. R. Small used a GA to evolve a ruleset for a FPS agent [41], T. Sandberg evolved parameters of a potential field in his Starcraft agent [37] , *City Conquest's* in-game AI used an agent-based GA-evolved build plan [20] and D. Perez et al. used a grammatical evolution (a GA variant) to produce behaviour trees for a Mario AI Competition entry [35].

2.4.2 Multi-tiered Approaches

Several of the most successful learning agents take a multi-tiered approach. By splitting high-level behaviour from low-level actions agents can demonstrate more a complex, and even human-like, performance. For example, COBOSTAR, an entrant in the 2009 Simulated Car Racing Competition, used offline learning to determine high-level parameters such as desired speed and angle alongside a low-level crash avoidance module [28, p. 136]. UT∧2 used learning to give their FPS bot broad human behaviours and a separate constraint system to limited aiming ability [29]. Overmind, the winner of the 2010 Starcraft Competition, planned resource use and technology progression at a macro level, but used A* search micro-controllers to coordinate units [9].

One learning agent that successfully utilised both an evolutionary algorithm and a multi-tiered approach is the Mario agent REALM, which is explored in more detail below.

2.4.3 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.

Rule-based

Rules map carefully chosen conditions (a simplification of the available environment information) to actions in a simple ruleset. Rule preference over binary conditions are either TRUE, FALSE or DONT_CARE [34, p. 85]. Each time step a rule is chosen that best fits the current condition, with ties being settled by rule order [34, p. 86].

Actions in V1 were explicit key-press combinations, whereas in V2 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 two-tier approach was designed in part to reduce the search space of the learning algorithm. [34, pp. 85-87]

Learning

REALM starts with a random ruleset and evolves it using a GA over 1000 generations. The best performing rule set from the final generation was chosen to act as the agent for the competition. Hence, REALM is an agent focused on offline learning. [34, pp. 87-89]

Population Populations have a fixed size of 50 individuals, with each individual being a rule set. Each rule represents a genome and each individual has 20. Initially rules are randomised, with each condition having a 40%, 30%, 30% chance to be DONT_CARE, TRUE, FALSE respectively.

Evaluation Individuals are evaluated by running through 12 different levels. The fitness of an individual is a modified score, averaged over the levels. Score focuses on distance, completion of level, Mario's state at the end and number of kills. Each level an individual plays increases in difficulty. Levels are predictably generated, with the seed being recalculated at the start of each generation. This is to avoid over-fitting and to encourage more general rules.

Breeding The breeding phase takes the five best individuals from the population and produces 9 offspring each. These parents are then also included in the next generation. Offspring are exposed to: **Mutation**, where rule conditions and actions may change value; **Crossover**, where a rule from one child may be swapped with a rule from another child and **Reordering**, where rules are randomly reordered. These occur with probabilities of 10%, 10% and 20% respectively.

Performance

The REALM V1 agent saw a larger improvement over the evolution, but only scored 65% of the V2 agents score on average. It is noted that V1

Name	Game/Competition	Approach
M. Erickson [24]	2009 Mario AI Competition	A crossover heavy GA to evolve an expression tree.
S. Polikarpov [25, p. 7]	2009-10 The Mario AI Competition	Ontogenetic reinforcement learning to train a neural network with action sequences as neurons.
REALM [34]	2010 Mario AI Competition	GA to evolve rulesets mapping environment to high-level behaviour.
D. Perez et al [35]	2010 Mario AI Competition	Grammatical evolution with a GA to develop behaviour trees.
FEETSIES [36]	2010 Mario AI Competition	"Cuckoo Search via Lévy Flights" to develop a ruleset mapping an observation grid to actions.
COBOSTAR [28, p. 136]	2009 Simulated Car Racing Competition	Covariance matrix adaptation evolution strategy to map sensory information to target angle and speed. Online reinforcement learning to avoid repeating mistakes.
L. Cardamone [28, p. 137]	2009 Simulated Car Racing Competition	Neuroevolution to develop basic driving behaviour.
Agent Smith [41]	Unreal Tournament 3	GAs to evolve very simple rulesets, which determine basic bot behaviour.
UT∧2 [29]	2013 2K Botprize	Neuroevolution with a fitness function focused on being 'human-like'.
T. Sandberg [37]	Starcraft	Evolutionary algorithms to tune potential field parameters.
Berkeley Overmind [9]	The Starcraft AI Competition	Reinforcement learning to tune parameters for potential field and A* search.
In-game Opponent AI [20]	City Conquest	GAs to evolve build plans.
In-game Creature AI[21]	Black & White	Reinforcement Learning applied to a neural network representing the creatures desires.
In-game Car AI [22]	Project Gotham Racing	Reinforcement learning to optomise racing lines.

Table 2: This table summarises learning based agent approaches to game AI

struggled with high concentrations of enemies and large pits. The creators also assert that the V2 agent was more interesting to watch, exhibiting more advanced and human-like behaviours. [34, pp. 89-90]

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. [24, 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 transformation strategy of 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

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.

Furthermore, due to time constraints and the exploratory nature of Objective 5, Objective 6 will be seen as stretch goal, whose completion may not be fully realised at the end of the project.

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.
- No simulation of the game engine to predict enemy movements. This technique has been used in the Mario AI competition, but isn't always applicable to a game AI challenge.

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.

4.2.2 Previous Agents

E. Speed's entry into the 2009 Mario AI Competition /cite[pp. 7-8]2010the 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. [25, 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. [34, 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.

4.3.2 Previous Agents

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 [34, p. 85]. Each time step a rule is chosen that best fits the current condition, with ties being settled by rule order [34, p. 86]. Actions in the first version (V1) were simply key-presses, whereas in version two (V2) they are high-level plans. The REALM V1 agent saw a larger improvement over the evolution, but only scored 65% of the V2 agents score on average [34, pp. 89-90].

Approach

The handcrafted AI will follow the ruleset approach. Rules will map conditions (as determined in Objective 2) to explicit key-press combinations as it has shown to be more greatly affected by the learning phase, and therefore is more fitting for this project's aim. 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 (developed in Objective 2) will be implemented into the *integrateObservation* method.

The aim of this handcrafted agent is not necessarily 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. Evolutionary learning has seen many successes in agent-based AI approaches (as seen in 2.4). Furthermore, the search space of this project is a ruleset, in which it is difficult to define the notion of a *neighbour*. The concept of *neighbours* is essential to many Reinforcement Learning techniques, such as Simulated Annealing. In this way, evolutionary methods are more easily applied to the non-continuous ruleset search space of this project as they are largely stochastic.

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 [38] and JGAP [39]. 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 played several thousands times faster than otherwise. This is essential for implementing an effective learning strategy, as high numbers of playthroughs take minimal time. [24, p. 3]

Tuneable Level Generator Another essential element for effective learning is level generation, in lieu of the lack of a data set. 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.

The level generator facilitates agents learning from many different levels and also allows agents to learn of a particular flavour of level, e.g. short difficult levels with many enemies. The **Seed** allows different agents to learn from the same data set, useful for evaluation and comparison (a draw back from truly random generation). [24, p. 4-5]

Approach

A package will be created that ties the genetic programming library with the benchmark software, allowing it to access the level generation and level playing portions of the software for the purpose of fitness evaluation. Furthermore, the package will be able hold a population of rulesets, with the

power modify them for the purpose of breeding. The parameters of level generation, fitness calculation and generation breeding will be easy to adjust, as it is important for Objective 5. The package will log results of each generation, which will be the basis of evaluating the learning process. Lastly, it will persist the last generation of rulesets, in order for one to be chosen as the final agent.

4.5 Objective 5: Agent Learning

Objective 5 is the first exploratory stage of the project and as such it is difficult to determine a specific approach. Most of the work done to reach this objective will be done planning and tuning the parameters of the learning framework.

The approach to this objective will once again draw inspiration from REALM.

4.5.1 Previous Agents

REALM starts with a random ruleset and evolves it using a GA over 1000 generations. The best performing rule set from the final generation was chosen to act as the agent for the competition. Hence, REALM is an agent focused on offline learning. [34, pp. 87-89]

Population Populations have a fixed size of 50 individuals, with each individual being a rule set. Each rule represents a genome and each individual has 20. Initially rules are randomised, with each condition having a 40%, 30%, 30% chance to be DONT_CARE, TRUE, FALSE respectively.

Fitness Individuals are evaluated by running through 12 different levels. The fitness of an individual is a modified score, averaged over the levels. Score focuses on distance, completion of level, Mario's state at the end and number of kills. Each level an individual plays increases in difficulty. Levels are predictably generated, with the seed being recalculated at the start of each generation. This is to avoid overfitting and to encourage more general rules.

Breeding The breeding phase takes the five best individuals from the population and produces 9 offspring each. These parents are then also included in the next generation. Offspring are exposed to: **Mutation**, where rule conditions and actions may change value; **Crossover**, where a rule from one child may be swapped with a rule from another child and **Reordering**, where rules are randomly reordered. These occur with probabilities of 10%, 10% and 20% respectively.

Another source of inspiration is R. Small's paper on an agent that plays Unreal Tournament [41]. Small used a GA to evolve a simplistic ruleset. The fitness function of the GA was tuned specifically to distil characteristics on the agent, in this case aggressiveness.

4.6 Objective 6: Agent Extension

Objective 6 is the second exploratory stage of the project. Time permitting, the goal here is to emulate the success shown by two-tier approaches, splitting high level behaviour from low level actions. The ruleset developed in the bulk of the project is a good basis for deciding high level behaviour, but a new approach may have to be investigated for the translation of this behaviour to low level actions. Furthermore, it is likely that Objective 2 will have to be revisited.

4.6.1 Previous Agents

Approaching this objective will involving reassessing previous approaches and relevant literature. Two previous agents of immediate interest are REALM and D. Perez et al's agent. Both utilise the high-level low-level split and are documented well in their reviews [34] and [35].

REALM V2 used a ruleset to decide on a general goal. This goal was passed to a simulator, which reassessed the environment and used A* to produce the explicit key-press combination. Not only did this change improve the agent by 65%, it also reduced the search space of the learning algorithm [34, pp. 85-87]. The creators also assert that the V2 agent was more interesting to watch, exhibiting more advanced and human-like behaviours [34, pp. 89-90]. As previously mentioned, this agent won the 2010 competition with the most kills and no disqualifications [24, p. 10].

D. Perez et al's approach uses Behaviour Trees, a common game AI approach to handling a behaviour-action split. Each behaviour tree is made up of subtrees that capture a specific goal. These are specifically made to be simplistic, comprised of condition nodes followed by action nodes. The process of Grammatical Evolution is used to evolve these behaviour trees, which are represented by a context-free grammar and encoded as integer strings. Although there was no use of search algorithms the agent was able to demonstrate planning behaviour [35]. It finished 4th in the 2010 Mario AI Competition, with the second highest kill count and no disqualifications [24, p. 10].

4.7 Objective 7: Agent Evaluation

Following the rules of the Mario AI Competition allows the agent to be evaluated in the same way as it's entrants. Furthermore, the evaluation used in the competition is available and adaptable.

4.7.1 Mario AI Competition Scoring

The scoring mechanism used in the competitions is included in the 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 [24]), although the levels played will be different.

However, as it is not the aim of the project to produce the best possible agent, it may be advantageous to adapt the evaluation. Possible changes could include lowering the difficulty or removing dead-ends. This process weakens the comparison against the 2010 results. However, the evaluation class allows multiple agents to be evaluated over the same levels (by passing the same seed), so 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. [25]
- Baumgarten's A* agent, which contains no learning and was the winner in 2009. It is open-source and available online [40].

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.

5.1 Phase 1

June 10th – July 10th

4 days Integration of the existing Java codebase with the Scala language and introduction of testing and build tools (as discussed in 4.1)

10-12 days Environment-condition translation scheme testing and implementation (as discussed in 4.2).

10-12 days Rule set interpretation testing and implementation (as discussed in 4.3).

2 days Construction of the handcrafted AI.

5.2 Phase 2

July 10th – September 7th

20 days Integration of the evolutionary learning library with the benchmark software as described in 4.4.

10-30 days Experimentation and assessment of evolutionary methods to evolve rulesets, including revisiting the condition scheme (Section 4.5).

10-30 days Investigating possible extensions of the agent as spelled out in 4.6.

5.3 Phase 3

August 24th – September 14th

14 days Evaluation of agent(s) as specified in 4.7.

7 days 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] Stuart J. Russell, Peter Norvig, *Artificial Intelligence: A Moden Approach (3rd ed.*, Upper Saddle River, New Jersey, 2009.

[15] Stuart J. Russell, Peter Norvig, *Artificial Intelligence: A Moden Approach (1st ed.*, Upper Saddle River, New Jersey, 1995.

[16] Anoop Gupta, Charles Forgy, Allen Newell, and Robert Wedig, *Parallel Algorithms and Architectures for Rule-Based Systems*, Carnegie-Mellon University Pittsburgh, Pennsylvania, ISCA '86 Proceedings of the 13th annual international symposium on Computer architecture, pp. 28-37, 1986.

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

[18] 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.

[19] 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`.

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

[21] 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.

[22] 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.

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

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

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

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

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

[28] 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.

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

[30] 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.

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

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

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

[34] 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.

[35] 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.

[36] 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.

[37] Thomas Willer Sandberg, *Evolutionary Multi-Agent Potential Field based AI approach for SSC scenarios in RTS games*, University of Copenhagen, 2011.

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

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

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

[41] Ryan Small, *Agent Smith: a Real-Time Game-Playing Agent for Interactive Dynamic Games*, GECCO?08, July 12?16, 2008, Atlanta, Georgia, USA. `http://www.cs.bham.ac.uk/~wbl/biblio/gecco2008/docs/p1839.pdf`.