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

Artificial intelligence in video games has long shunned the use of machine learning in favour of a handcrafted approach. However, the recent rise in the use of video games as a benchmark for academic AI research has demonstrated interesting and successful learning approaches. This project aims to follow this research and explore the viability of a game-playing learning AI. Considering previous approaches, an evolutionary agent will be created for a platform game based on Mario Bros.

The project will build on top of software developed for the Mario AI competition, which provides the game-engine and agent interface, as well as several other pertinent features. The code will be written in two modules. The basic agent being constructed first and then a learning framework built to improve it, utilising a genetic algorithm. The project will follow an agile methodology, revisiting design by analysing learning capability. The aim is to produce an agent the shows meaningful improvement during learning.

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.

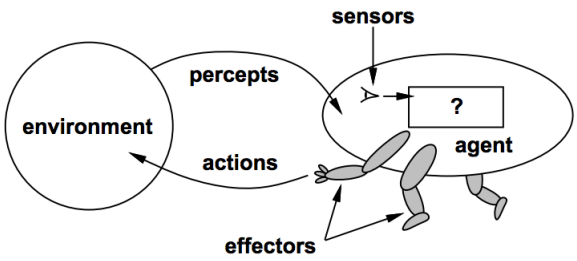

Figure 1: Illustration of an intelligent agent, taking from [15, p. 32]

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.

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 limit 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

Each time step REALM creates a list of binary observations of the current scene, for example IS_ENEMY_CLOSE_LOWER_RIGHT and IS_PIT_AHEAD. Conditions on observations are mapped to actions in a simple ruleset. These conditions are ternary (either TRUE, FALSE or DONT_CARE) [34, p. 85]. A rule is chosen that best fits the current observations, with ties being settled by rule order, and an action is returned [34, p. 86].

Actions in V1 are explicit key-press combinations, whereas in V2 they are high-level plans. These plans are passed to a simulator, which reassesses the environment and uses 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 30%, 30%, 40% chance to be TRUE, FALSE or DONT_CARE 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 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]

Name	Game/Competition	Approach
M. Erickson [24]	2009 Mario AI Competition	A crossover heavy GA to evolve an expression tree.
E. Speed [25]	2009 Mario AI Competition	GA to evolve grid-based rulesets. Ran out of memory during competition.
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

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 agent-based AI. This will be achieved by producing an intelligent agent that plays the Mario AI benchmark. The project will pose this as a reinforcement learning problem.

3.2 Objectives

1. **Preparation**
   Assess the available Mario benchmark software and carry out a feasibility study in regards to language, testing frameworks, build tools and logging.

2. **Design**
   Influenced by previous approaches, design an agent and a reinforcement learning procedure that can demonstrate meaningful improvement during learning.

3. **Implementation**
   Implement agent and learning designs in a modular, customisable and test-driven manner, utilising external libraries and features of the benchmark software.

4. **Testing**
   Provide significant test coverage to the functionality of the implementation.

5. **Learning**
   Grant the agent a large and diverse testbed from which to learn as well as ample time and resources to do so.

6. **Evaluation**
   Evaluate both the competence and learning capability of the agent and compare to alternative approaches.

3.3 Limitations

It is unlikely that the final agent will be able to compete at the level of the previous Mario AI competitions. The focus of the project is on an effective learning procedure rather than pure attainment of the agent.

4 Methodology

The two central components to this project, the agent and the learning process, will be designed, implemented and tested as two separate modules. This presents a clean separation of concerns. The agent module will be completed first, followed by the learning module.

It is foreseeable that certain parts of the project will be exploratory and as such evaluation may inform a required redesign of the either the agent or the learning modules. Hence it will be important to take an agile and iterative approach to production.

4.1 Objective 1: Preparation

The Mario benchmark software is open-source[1] and will form the initial codebase for this project. It is written in Java and contains the game-engine and useful features such as level generation and level playing classes. It also provides an agent interface, which will act as the link between my agent module and the game-engine. The source files will imported into an Eclipse IDE project.

I intend to use Scala as the main language of this project as I believe that it's functional and concurrent features will suit the problem. Scala and Java are widely compatible (as they both compile JVM bytecode) so this should no require a significant amount of work. However, the codebase will be assessed for the suitability of this approach, and Java 8 will be used if Scala proves to be counterproductive.

Furthermore the benchmark will be assessed for the inclusion of build tools, such as *sbt* and *Maven*; testing frameworks, such as *ScalaMock* and *Mockito* and logging, such as *log4j* and Scala's inbuilt logging package.

4.2 Objective 2: Design

The design of both the agent and learning module will follow the approach of the REALM agent, discussed in 2.4.3.

The agent module will be a rule-based system that maps environment based conditions to explicit key presses. This project will explore changing the conditions of REALM's V1 agent. The available sensory information will be assessed and distilled into a list of simple observations. As in REALM these will likely be binary values with ternary conditions encoding preference in the rules, but other approaches will be considered.

The learning module will utilise an offline genetic algorithm with the same configuration seen in 2.4.3. Rulesets will be individuals, with rules as genomes. Fitness of an individual will be determined by playing generated levels. However, different approaches to mutation, crossover and reordering

[1] Available at http://www.marioai.org/gameplay-track/getting-started

will be explored, as well as careful consideration of calculating fitness from level playing statistics.

If time permits, a redesign of the agent, to incorporate a multi-tiered approach, will be explored. This will involve a reflection on previous approaches such as REALM V2 [34] and D. Perez et al. [35].

4.3 Objective 3: Implementation

The agent module will implement the *Agent* interface including in the benchmark, which will allow for the agent to used in other areas of the benchmark software, such as the game-playing and evaluation classes.

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

The specific ruleset will be persisted external and loaded into memory on the startup of the agent module. The getAction() method will be implemented as the semantic reasoner for the ruleset, returning a boolean array of key presses as prescribe by the chosen rule.

The mapping of the sensory information to conditions will be handled by the integrateObservation() method. The *Environment* interface provides the sensory information to the agent. It includes a 22x22 grid charting terrain and box locations, pixel-resolution of enemy positions and information about Mario's state. As discussed in 4.2, this will be simplified to a list of conditions, which will be stored in memory and compared to the ruleset by the semantic reasoner.

The learning module will utilise an external library alongside features of the benchmark software. Many evolutionary computation libraries exist for Java (and therefore Scala), *ECJ* [38] and *JGAP* [39] will both be considered for use in this project.

The implementation will manage the integration of this library with both the agent module and the relevant features of the benchmark software. For the breeding phase it will hold and edit several rulesets at a time. It will then inject these rulesets into an instance of the agent and pass it to sped-up level playing class in the benchmark. The statistics returned from that class will form the basis of the fitness function. This phase will also include generation of the levels. The breeding, fitness and level generation parameters will be held externally to allow for easy tuning, which will assist the agile approach of the project. Lastly, fitness results will be logged to provide a basis for evaluating the learning of the agent.

4.4 Objective 4: Testing

Testing of the agent module will provided on two levels. Unit tests will be written to test the semantic reasoner and the environment observation methods. To achieve this, the Environment interface and rulesets will be mocked. Black-box integration tests will test the agent module as a whole. One focus of these tests will be response time, the benchmark game-engine runs at a constant 25fps, therefore the agent must respond within 40ms when deciding an action each time step.

Due to the stochastic nature of genetic algorithms, testing of the learning module will be limited. However, the breeding can be tested by verifying that children stay within parameters. Some evolutionary libraries, such as JGAP, provide several unit tests. If available these will become the primary source of test coverage for the module.

4.5 Objective 5: Learning

The learning testbed will be generated levels. The benchmark software offers the *LevelGenerator* class, which is tuneable by over 20 parameters including level complexity, enemy density and level length. The level generation is controlled by a seed, allowing levels to be regenerated if necessary. A new seed will be chosen for each generation and several levels will be generated with differing difficulties, length etc. In this way each individual of a generation will play the same levels, but over the course of the evolution a large number of levels will have been played, providing a large and diverse testbed.

Optimising the testbed parameters, as well as the parameters of the breeding phase and fitness function, will be an important stage of the project. Assessment of the parameters used in previous agent such as REALM (discussed in 2.4.3) D. Perez et al. [35] and Agent Smith [41] will inform the initial values. From there learning module logs will be analysed for improvements. For example, if there is a lot of variation in fitness then perhaps mutation should be restricted, or if the average fitness does not eventually level out then the further generations should be created.

The design of the agent can also influence the effectiveness of the learning algorithm. The size of the search space is determined by the conditions and actions of the rulesets, the reduction of which could improve evolutionary capability. Hence, the learning phase of the project may inform a redesign of the agent, which is one of the main reasons this project will take an agile approach.

The learning itself is likely to be a time consuming, computationally heavy procedure. To assist in providing ample resources to this process the project will have access to two 8 core servers, as well as a laptop with an Intel i7 processor.

4.6 Objective 6: Evaluation

The benchmark software provides classes for evaluating agents. The best ruleset will be chosen from the final generation of the genetic algorithm and tested using the *GamePlayTrack* class. Here the agent will play 512 unseen levels of varying type and difficulty. These levels are initiated by a seed, and therefore multiple agents can be scored on the same levels, which allows agents to be directly compared. Due to the limitations of the project discussed in 3.3, editing this class to make the trial less demanding will be explored as it may produce a more useful evaluation.

The primary comparison will be with a handcrafted ruleset, which will assess the significance of the agent evolution. Other comparisons can be drawn against agents that are included as example in the benchmark, such as the *ForwardJumpingAgent* that was used for similar comparisons in the 2009 competition. As the *GamePlayTrack* class was the method of evaluation for the 2010 Mario AI Competition, if the unedited class is used further comparisons can be drawn against it's entrants.

The second part of this objective is to evaluate the learning procedure. Figures such as average and maximum generation fitness can provide an insight into the effectiveness of the genetic algorithm. Furthermore, a baseline for these values can be provided by having the handcrafted agent play the levels alongside the evolving agent. The final evaluation report will provide an analysis of these figures.

5 Time Allocation

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.

This section aims to break each objective into tasks and allocate a number of days to their completion. This time allocation, along with the objectives, will form the basis of a critique of the project, to be included in the final report.

5.1 Objective 1: Preparation

$\frac{1}{2}$ a day	Importing of benchmark software into Eclipse IDE.
1 day	Assessment of use and possible inclusion of the Scala language.
$\frac{1}{2}$ a day	Inclusion of build tools, testing frameworks and logging in to project files.

5.2 Objective 2: Design

2-3 days	Design of structure and content of observation list (discussed in 4.2).
2-3 days	Design of crossover, mutation and reordering procedures used within the genetic algorithm.
7-10 days	Re-design of the agent to adopt a multi-tier system, including reexamining previous approaches (Non-core task).

5.3 Objective 3: Implementation

4-7 days	Implementation of translation of *Environment* to observation list in agent module (discussed in 4.3).
4-7 days	Implementation of semantic reasoner for ruleset in agent module.
5-8 days	Implementation of external ruleset persistence, including read and write access.
5-10 days	Implementation of agent re-design (Non-core task).

10-14 days	Integration of evolutionary Java library with the benchmark software.
3-6 days	Implementation of crossover, mutation and reordering of rulesets in the genetic algorithm.
3-6 days	Implementation of evolution fitness tracking and logging.

5.4 Objective 4: Testing

2-3 days	Writing of unit tests for agent module.
2 day	Writing black-box integration test for agent module.
4-7 days	Assessing and writing testing strategy for learning module.

5.5 Objective 5: Learning

10-20 days	Offline evolution of the agent.
3 days	Tuning of learning parameters (discussed in 4.5).

5.6 Objective 6: Evaluation

2-3 days	Evaluating capability of the final agent.
5 days	Collecting and charting data pertaining to agent evolution.
7 days	Finishing touches to the project report.

5.7 Total Allocation

The completion order of tasks will not follow the order presented here. This project is adopting an agile appraoch, revisiting tasks as it becomes necessary. The allocation given here is the time expected to be spend on each task in total.

Core task day allocation sums to between 70 and 106 days. This therefore allows a possible 26 days for non-core tasks (in 5.2 and 5.3) related to reengineering the agent to adopt a multi-tier approach.

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`.