An Investigation into the use of Genetic Algorithms in RTS Games

BSC (HON) COMPUTER GAMES TECHNOLOGY

Finlay McGrath

2017

Contents

[i. Abstract 2](#_Toc480669630)

[ii. Abbreviations 3](#_Toc480669631)

[1. Introduction 4](#_Toc480669632)

[1.1. Research Question 5](#_Toc480669633)

[1.2. Aims and Objectives 5](#_Toc480669634)

[2. Literature Review 6](#_Toc480669635)

[2.1 Background 6](#_Toc480669636)

[2.2 AI Techniques 7](#_Toc480669637)

[2.3 Summary 9](#_Toc480669638)

[3. Methodology 10](#_Toc480669639)

[3.1 Gameplay 10](#_Toc480669640)

[3.2 Rule Based AI 12](#_Toc480669641)

[3.3 Genetic Algorithm AI 13](#_Toc480669642)

[4. Results 17](#_Toc480669643)

[5. Discussion 20](#_Toc480669644)

[5.1 Genetic Algorithm 20](#_Toc480669645)

[5.2 Rule Based AI 21](#_Toc480669646)

[5.3 Gameplay 22](#_Toc480669647)

[6. Conclusion and Future Work 24](#_Toc480669648)

[7. Appendices 26](#_Toc480669649)

[7.1 Units 26](#_Toc480669650)

[7.2 Buildings 26](#_Toc480669651)

[7.3 Technology 27](#_Toc480669652)

[7.4 Rule Based AI Priority List 27](#_Toc480669653)

[7.5 List of Actions 28](#_Toc480669654)

[8. References 29](#_Toc480669655)

[9. Bibliography 31](#_Toc480669656)

# i. Abstract

Game Artificial Intelligence (AI) is an area of the games industry that has frequently been underappreciated and underinvested in by developers. Modern games still use the same rule based Artificial Intelligence techniques for making computer controlled players as they did nearly 20 years ago.

This project will consider an alternate method for creating an AI by using genetic algorithms to create a more adaptable AI that plays fairly without resorting to cheating.

An AI that uses a genetic algorithm to learn, will be tested against a rule based AI using a Real-Time Strategy (RTS) game made for this project. The genetic algorithm AI will be able learn how to play the game through an iterative process that uses a victory based fitness function to find the most successful solutions.

The genetic algorithm AI was capable of defeating the rule based AI after playing a thousand games of the RTS against a rule based AI. The results showed that the victory based fitness function was very successful at identifying when the genetic algorithm AI was winning and continued to win.

This project has demonstrated how a genetic algorithm could be created in order to improve the AI used in Real Time Strategy games in order to provide a more adaptable opponent for a human player. The project was very successful as it met the aims of creating an AI that was superior to the methods currently used in Real Time Strategy games. There is a discussion how more work could be done to adapt the project to be more suitable in creating game AI that could be realistically used in a Real-Time Strategy game.

# ii. Abbreviations

AI – Artificial Intelligence

DPS – Damage Per Second

FPS – First Person Shooter

RTS – Real Time Strategy

# 1. Introduction

Artificial Intelligence (AI) as used in games to direct a computer controlled actor is an important part of how players interact with a game. This differs from AI as used in academia where the AI is the end goal of research, where as in games, the AI is one several tools a developer might use to create a better playing experience. However, frequently a computer controlled actor in games is underwhelming and can detract from the experience. This can be shown by a large spectrum of issues that range from the simple to solve such as AI actors running into walls to more complex issues such as predictability and being easy to exploit by a human player. This leads to situations where developers of games such as the Sid Meier’s Civilization series give the AI unfair advantages that the human player does not possess. This merely delays the problem as it forces players who wish to challenge the most difficult of opponents to rely entirely on exploiting weaknesses in the AI’s rules without the AI being able to adapt. This simply exacerbates the problem creating a cycle where a player finds an exploit, the developer gives the AI an unfair advantage therefore meaning the player must take advantage of the found exploit to keep up with the AI. At the same time, AI advantages can feel unfair to human players despite the natural advantages the human player possesses such as creativity and the ability to plan.

There are generally two main issues in creating good game AI. The first issue is in being able to teach an AI how to play the game well enough to provide a challenge to the player. When compared to a game such as chess which has only 32 pieces on the board and a limited number of moves, a Real-Time Strategy (RTS) game has thousands of actions and millions of possible permutations. Each action creates a new branch of possible variations of the game. Games that rely on the movement of a unit can be the most complex as each unit in the game can be moved to each location in the game creating countless numbers of decisions a player can make. The difficulty is that previous rule based systems are very limited as it requires the developer to predict each of those permutations for the AI to provide a response. This means that some kind of machine learning is necessary.

The second issue is in ensuring that the created AI is enjoyable for a human player to play against. This means that it must adhere to a human sense of fairness and it must be beatable yet still provide a challenge sufficient for the player to enjoy. One of the ways this is done in RTS games is by providing multiple levels of difficulty to cater to different skill levels. However, in the higher difficulties this frequently involves the AI receiving advantages that the human player does not. The advantages provided vary depending on the game but in a First-Person Shooter (FPS) it could auto-aim meaning the AI would never miss, or in an RTS the AI could be provided with additional units or given stronger units.

RTS games usually involve two or more players competing over resources with the goal being to destroy the opponent. Each player can gather resources which are used to build buildings and to train units from those buildings. A unit can travel across the game map, which is generally specially designed for that game. Units must attack the opposing player’s units and buildings to destroy them and eventually defeat their opponent. RTS games can be symmetrical or unsymmetrical where in symmetrical games each player has the exact same capabilities as the other player. In unsymmetrical games, such as Blizzard Entertainment’s Starcraft II 2009 it is possible for a player to have access to a completely different roster of units and buildings than the opposing player. The goal of this project is to create an RTS AI that is capable of learning how to win while only having the same information as a human player, without being exploitable. This would then be compared to a rule based AI such as used in Warcraft 3 2002 or Age of Empires II 1999 in which the AI follows a rigid set of rules that are pre-set by the designer and cannot change without direct human input.

A genetic algorithm could be used for this purpose as they are very good at finding near optimal solutions to a problem (Simon Mardle and Sean Pascoe, 1999) without a reliance on unfair advantages. A genetic algorithm uses a fitness function that defines success to learn how to do something. While doing so, it also creates many sub-optimal solutions meaning it can be very useful for creating an AI of varying abilities. This allows it to be used for multiple difficulties for players of different skill levels making it ideal for an AI in an RTS that can easily be retrained whenever changes are made to the game. The intention and expectation would be that the learning Genetic AI would be superior and would beat the rule based AI after a sufficient learning period. The gameplay would be symmetrical in nature with each player being given the exact same capabilities, units and buildings as the other player.

This project would aim to create an AI that is challenging yet beatable and operates on the same level as a human player without receiving unfair advantages. This will be done using a genetic algorithm that uses the fitness function to find the correct way of playing against another AI.

## 1.1. Research Question

* Can a genetic algorithm be used to create a more adaptable and enjoyable experience for a human player when facing a computer player?

## 1.2. Aims and Objectives

* Research the application and use of genetic algorithms.
* Investigate how AI techniques are used in current RTS games for AI.
* Create an AI that can learn how to play an RTS game using genetic algorithms.
* Compare the genetic algorithm to other methods of AI in games.

In section two this dissertation will discuss the sources used to build evidence for the most effective method for creating a more adaptable AI for RTS games that is superior to the rule based AIs that are generally used in RTS games. It will then show in section three how this can be demonstrated by creating a game where the genetic algorithm AI trains and learns from a rule based AI. Results of this process are shown in section four and there is a discussion of what the results mean and how effective this process has been in section five. A further analysis on the project has been made to debate the overall success of the project and how it could be improved on in the future as part of section six.

# 2. Literature Review

## 2.1 Background

Since the dawn of industrialisation, machines have been used to automate processes which previously would have been done by a human. The modern version of this is robots which are used in factories to carry out tasks at a far more reliable and relentless rate than was previously possible. These techniques, while effective within their operational parameters, do have their limitations. For example, if the robot is taken out of its operational environment, it would not be able to function as it is an unintelligent machine. A potential solution to this problem would be to find a way for the robots to adapt to its current environment through learning.

Artificial Intelligence (AI) is a highly active area of research in computing and robotics with a “'Partnership on AI' formed by Google, Facebook, Amazon, IBM and Microsoft” to advance public understanding of AI (The Guardian 2016). During 2016, Google’s Deepmind AI defeated the world Go champion in a best of five series of games (BBC 2016) using AI learning techniques that enabled it to learn how to play.

AI has been used to great success in several industries such as online shopping where Amazon uses AI both for their website and in the physical logistics of moving goods in a warehouse. Amazon uses machine learning to predict what customers want, sometimes before they know they want it. It is also used for creating a more efficient customer support, meaning the customer can solve their issue without contacting an actual member of customer support. On the physical side, Amazon has used machine learning to maximise efficiency of staff when they are retrieving items from the shelves in a warehouse by optimising routes between objects.

Figure 1 Starcraft 2. Blizzard Entertainment

In the games industry, AI has not advanced significantly since the 1990s with games like Starcraft (1997) and Starcraft II (2009, figure 1) using the same rule based AI despite a twelve-year time gap and enormous advances in other areas such as graphics. One of the main reasons for this is due to the complexity of many video games as they can contain hundreds of variables at a time. This means that a brute force algorithm such as used by IBM’s Deep Blue in 1997 to defeat the then chess grandmaster Garry Kasparov is not feasible. This is due to it requiring enormous processing power to the exclusion of all other aspects of gameplay and even then, it would not be possible for a modern RTS (Real Time Strategy) game.

## 2.2 AI Techniques

There are several AI techniques which have been used successfully in video games in the past. The ones which will be discussed are neural networks, fuzzy logic, rule based systems and genetic algorithms. The previously mentioned rule based AI is a system where the AI takes an action if a condition is true. For example, in an RTS game a rule could be that the AI only attacks after it has reached enough units. Ligêza, 2006 describes rule based systems as one of the most common ways of solving problems as they explicitly state what is and what is not possible, this makes is simple to define what actions an AI will carry out. The actions an AI can carry out are defined by the rule base and the AI cannot deviate from them. Due to this, machine learning is not possible and the AI cannot adapt to a changing situation. However, rule based AIs are comparatively simple to set up if the rules are known in advanced and this is one of the main reasons why they have been used in the games industry to such a large degree. Another use for rule based AI is that they can be a useful benchmark for repetitive testing of other AIs such as neural networks or genetic algorithms which require a great deal of training before they prove successful.

Figure 2 – Artificial Neural Network (CBurnett)

A neural network (figure 2) is a form of machine learning that is based on the human brain. A brain is made up of many thousands of neurons which are a linked together. Each neuron can be described at the most basic level as taking in an electrical input, doing some process and then outputting the result. This idea was given a mathematical basis in *A Logical Calculus of the Ideas Immanent in Nervous Activity* (McCulloch and Pitts, 1943). This process can be transferred to AI by replicating the connections between artificial neurons defined as functions and the network can learn through training on a model by example. The training phase adjusts weightings of actions that define the likelihood of an action being taken, based upon the example that it is training on. The main flaw of this methodology when applied to games is that the AI can only ever be a copy of the player that the AI is training against. This means that the if the player is bad at the game so will the AI and it also means that the AI will need to be retrained with every change to the game that is made. In addition, training times can prove extremely long and this can prove to be somewhat impractical.

Fuzzy logic as described by P Wang, D Ruan and E Kerre, is a form of AI which, as opposed to actions or choices being correct or incorrect, relies on the idea that not all the facts are known and there is an element of uncertainty to all decisions being made. It also allows for a degree of ambiguity in the information provided to the AI and the main idea behind it is to turn an analogue world into a system which a digital machine can understand. This can work well for competitive games as by their nature a player does not know what the other player is thinking, therefore they can only prepare for the most likely action. Each action is given a degree of truth between zero and one, the most appropriate action is given the highest weighting. The degree of truth is given through the membership functions which evaluate whether certain properties are true. In an RTS example, the decision could be how in game resources could be spent, this decision would be affected by a variety of factors defined by membership functions such as how many units a player controls, how early in the game it is or what the opposing player is doing. These factors would come together to produce the degree of truth and based upon that degree of truth an action would be carried out. However, a flaw with this method is that you either need to know in advance to what degree each membership function effects the overall degree of truth or have some method of machine learning for the AI to adjust this during gameplay.

**Initialize the population**

**Select individuals for the mating pool**

**Perform crossover**

**Fitness Insert offspring into the population**

**The End**

**Perform mutation**

**yes**

**no**

**Stop?**

Figure 3 – Genetic Algorithm Process

Genetic algorithms were initially described by Holland, 1975 and are based upon evolutionary theory. The process (figure 3) starts with a completely random set of choices from the AI defined as a chromosome that are evaluated by a fitness function with the most successful being crossbred. The fitness function is what defines the success parameters, it is application dependent and the ’fitter’ the solution is, the higher the chance of it being chosen. The effectiveness of the solution is defined by how effective the fitness function is, and changing the fitness function can make or break the success of the solution. The candidate solutions chosen by the fitness function are paired and then crossover between the pairs occurs. The fitness function may select multiple candidates for crossover and the crossover can be done in several ways. The method that usually occurs is that the algorithm takes a segment of the chromosome from each of the chosen candidates and cuts them together to form a new candidate. An example of this would be if in a situation where eight candidates were chosen, the first half of the chromosome from candidate one and the second half of candidate two would be taken to form a new candidate. This would be repeated for candidates three and four, five and six, and seven and eight. This would create four new candidates and a mixture of the old and the new candidates is used to create a new generation. Each generation has a degree of mutation decided by probabilities to ensure there is some variation between generations by rerandomising some parts of the chromosome. This means that they cannot end up in a situation where an optimal solution could be found but the algorithm cannot find it as it is trapped in a local minima. Upon creating the new generation, the solution is reattempted and the process begins again as shown in figure 3.

Genetic Algorithms have been studied in a variety of domains from data mining and bioinformatics (Maulik 2011) to finding the best methods of energy efficiency (Liu and Huang 2012). Directly related to games it has been used to make an AI that “is capable of evolving a team's behaviours and optimizing the commands in a shooter game” (Liang 2013). This paper created an AI for Quake III Arena that was designed to be competitive with humans. It discusses the original rule based AI in Quake and how it limits the quality of the AI. Later it goes into detail about their use of a finite state machine to simplify the complex systems in a first-person shooter to ensure the AI can understand. An FPS has many variables that can affect AI performance, the game exists in a 3D game level that the AI must correctly navigate through quickly and effectively to find opposing players or objectives. The AI must correctly identify the most important objective on the map while avoiding enemy fire and maintaining the ability to shoot at enemies accurately. There can be multiple types of guns, some of which are more effective than others in addition to “powerups” on the map which effect gameplay. All these factors mean that rather than making a general-purpose AI, it is easier to create multiple states in a finite machine, each of which can handle an individual task. It operates through assigning scores to different states such as winning a game, capturing the flag and killing an enemy. These scores are fed into the fitness function and used to calculate the most successful solution of that generation. This allows the AI to learn how to select the most beneficial actions by passing through states as quickly as possible until it wins.

A different approach to the problem was taken by Mora et al (2012) and Fernández-ares et al (2013) who worked together to create a bot for the game Planet Wars which was part of the Google AI challenge. It is a turn based multiplayer game where each player controls a planet and have a certain number of troops. Also on the map are a large number of uncontrolled planets. The object of the game is to take over all your opponent’s planets by sending troops to defeat defending enemy troops, neutral planets can be taken over in the same way. The more planets you have the more troops you have, meaning you can launch more invasions and eventually win. The first paper (Mora et al 2012) uses the genetic algorithm to tune the weightings of a rule based AI to provide an initial base. Then, the AI is trained against other AIs that follow different methods of learning in order to provide more precise and accurate results.

The second paper (Fernández-ares et al 2013) starts with what was produced from the previous year’s work but takes a different route in improving the quality of the AI. Here they created multiple methods of determining success by using three different fitness functions that operate differently and are “based upon victories and numerical performance”. They mapped the output of the different versions of the genetic algorithm using behaviour trees in order to understand and map the choices that the AI made. Their results showed that the best method for creating the most effective genetic algorithm was through making the fitness function victory orientated. This meant that winning the game of Planet Wars was the most important characteristic when compared to any other parameter such as planets controlled or resources gathered.

## 2.3 Summary

There are many different AI techniques that can be used in games but the crucial factor is having an AI that can learn and adapt to a changing situation while providing a more enjoyable experience for the player. The most commonly used technique is the rules based system so a comparison should be made with a rule based AI. A genetic algorithm was chosen as it allows for an AI that unlike a rule based or fuzzy logic system is capable of learning. This is necessary for the game to learn how to play and adapt to a certain scenario to overcome its opponent. A genetic algorithm also has the advantage compared to a neural network in that it does not merely copy the example provided. Instead, the fitness function is the definition of success rather than whatever the neural network example is and is far easier to implement, train and run in real time. This means the fitness function is the most important factor in the genetic algorithm as it is likely that a genetic algorithm would become superior to the player that is being used to train it and that by controlling the fitness function the challenge the genetic algorithm AI represents can be changed.

# 3. Methodology

Based on the research carried out, the chosen methodology is an AI that uses a genetic algorithm to adapt and learn how to win against a rule based AI. The rule based AI will have a fixed ruleset and cannot change what actions it carries out between games but the genetic algorithm AI can. The genetic algorithm AI has a list of initially randomised actions and both AIs use a finite state machine to act upon the action chosen. Then after several games have been played against the rule based AI, the fitness function will evaluate which versions of the genetic algorithm AI are most successful and breed them together. This creates a new list of actions that the AI will carry out during gameplay. The fitness function will take victory over the rule based AI as the single most important factor in choosing how successful the AI was in a game though other factors are considered.

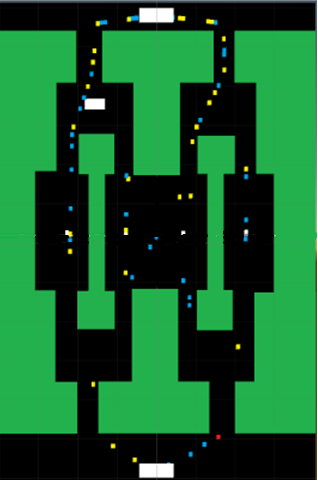
This project has three sections, a genetic AI, a rule based AI, and an RTS game created for the purposes of testing the AIs. The RTS segment is loosely based upon (Age of Empires II 1999), taking the core gameplay such as building units, gathering resources and attacking the enemy in order to provide sufficient complexity to test the AIs. The gameplay is just a canvas upon which the AIs can be tested and is not the focus of the project, instead gameplay is used to provide meaningful comparisons of the AIs.

## 3.1 Gameplay

The gameplay has three main components, player actions, unit movement and unit interactions. Each player can carry out the same types of actions, however the method in which they are carried out is different. Here detail will be provided on the types of actions and the requirements needed for those actions to be carried out. One of the main requirements for most actions is for a player having enough resources to carry out that action. More resources can be obtained by increasing the number of worker units a player has or by controlling more resource points on the map (Figure 4). Many adjustments were made to the gameplay during development of the project to make it more suitable for testing a genetic algorithm and making it more like an RTS. The original version of the game consisted of two players that controlled a headquarters. They could train workers to gain more resources and train warriors to attack the opposing player. This simpler design was steadily expanded upon during development until it reached the version described in the methodology.

There are many other unit types other than worker units however the worker unit is the only unit that can be trained at the start of the game. Other units require a resource investment by constructing buildings before they can be trained. These buildings include the barracks which trains melee units that only attack at close range, the archery range which trains ranged units and the stables which train fast hard hitting cavalry. Different units and building cost different numbers of resources and generally the most expensive a unit is, the more powerful it is in combat. There is also a tiered structure where a barracks is required before an archery range, and an archery range before a stable. A further step in the chain is that for the most powerful units to be built, a player must invest resources in research to unlock the ability to train that unit. The research itself is only available after a stable has been built. Other building opportunities for the player include walls and towers. These can be placed at specific chokepoints on the map that protect the player’s headquarters and are impassable to enemy units. The requirements of building walls and towers are that a player must control the area around the building site and have enough resource to construct the building. A player can capture base expansion points which allow the construction of an additional resource gathering building. These resource gathering buildings allow for a higher worker count as the Headquarters building only allows for twenty workers to gather resources at once. Each additional expansion point once built increases the cap by twenty units meaning that a player can gather more resources allowing for more units to be trained. A full list of units, buildings and technologies can be found in appendices one to three.

Rule AI



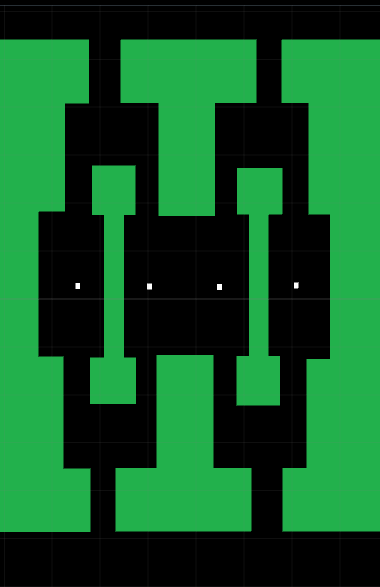
Genetic AI

Expansion/Resource Points

Figure 4 – Game Map

The unit interactions take place between neutral objects and enemy objects. Neutral objects include the terrain of the map and building points that can be captured. The tower bases can be captured and become friendly by having the most units near it. A friendly tower base can be turned into a tower by expending resources causing it to deal damage to nearby enemy units and must be destroyed to prevent this. The terrain of the map is a physical blockage that the units cannot travel through and acts as the boundaries of the map. The terrain is indicated by the green blocks shown on figure 3. Interaction with enemy objects involves attacking nearby enemies until either unit dies. These interactions follow the game rules as opposed to direct control by the player, the AI trains the unit then the actual attacking is resolved automatically based upon the proximity of units. A player can create walls that are passable to friendly units but unpassable to enemy units. These walls can be damaged and rebuilt similarly to towers.

The final part is the movement of units across from one side of the map to the other which are represented by coloured squares in figure 4. The yellow squares are infantry units including warriors, swordsmen and spearmen, the blue squares are ranged units including archers, crossbowmen and catapults while red squares represent cavalry like knights and horsemen. This uses a system of waypoints where a unit is given a list of directions by the player and an initial vector location to head towards. This is the only control the player has over player movement. For example, a unit could be told to go left and is given the vector location of the first waypoint. The unit would then travel towards the waypoint and upon reaching it the unit is given the location of the next waypoint. This continues until either the unit has died or it has reached the final waypoint of the opposing player’s headquarters. Figure 4 shows green blocks as the walls of the map, the coloured squares are units and white squares are buildings. Figure 5 indicates the directions in which units may travel, there are paths which can be branching, such as at the beginning where a unit can be sent either left or right. The arrows show the valid paths a unit can take when travelling from their headquarters at the top or bottom. Each unit can only travel in one direction, the rule based AI can travel down the map and the genetic algorithm AI can travel up the map. The arrows are multidirectional to represent that the two players travel along the same paths but in opposite directions. Changing the direction counts as a game action and is global for all units, however if a unit goes left initially it does not mean it will always go left. Upon reaching a branching waypoint the waypoint queries the unit’s owner on the current global direction of all units and sends the unit in the corresponding direction. A waypoint is branching if there is more than one valid direction that a unit can be sent and these have been indicated on figure 5 with a B.



B

B

B

B

B

B

B

B

B

B

Figure 5 – Game Map with directions

## 3.2 Rule Based AI

Initially a basic rule based AI was created that can play the game at a beginner level. It was given a list of orders to use in a situation, for example early on it focuses on building a strong economy then steadily builds up units to attack while advancing through research. The rule based system has no adaptability and is not capable of learning. The goal of the game is to destroy the opposing player’s headquarters building and the game will end when either player loses their headquarters. A new game will begin with the same rule based AI but the genetic algorithm will have a different set of actions to follow.

The main limiting factor on what actions can be carried out is the number of resources that a player has. Therefore, to resolve this, a fixed priority list was created that causes actions to be carried out in a specific order dependant on how high they are in priority. Programmatically this was implemented as a series of IF ELSE statements in C# with only one action being selected per update, with the code travelling down a series of checks until one is true. A basic overview of the rule based AI priority list is that it trains worker units until it has trained twenty units, then it builds a barracks to allow training of warrior units, then begins to train warriors while saving up enough money to build an archery range. The AI works its way down the priority list until an action is possible, if no actions are possible then no action will be taken. If an action further up the priority list becomes possible while resources are too low for the next action, the higher priority action will be carried out instead on the next update. One difference between the rule based AI and the genetic AI is that the rule based AI changes the global unit direction whenever a unit is trained and this is not a discrete action. When an action has been selected, it utilises the same finite state machine as the genetic algorithm AI does. Table 1 demonstrates the full rule AI priority list.

Table 1 – Rule AI Priority List

|  |  |
| --- | --- |
| Priority | Rule |
| 1 | Build workers if has less than 10 and headquarters not busy |
| 2 | Build expansion if controls an expansion point, has less than three expansions and had not reached the worker cap |
| 3 | Build stables if it doesn’t own a stable and has an archery range |
| 4 | Build archery range if it doesn’t own an archery range, it can afford it and has a barracks |
| 5 | Build barracks if it doesn’t own a barracks and can afford one |
| 6 | Build warrior if there is a free barracks and can afford it |
| 7 | Build workers if below cap and headquarters isn’t busy |
| 8 | build knight if stables isn't busy and can afford |
| 9 | Build horseman if stables isn’t busy and can afford |
| 10 | build crossbowman if archery range isn't busy, has the tech for it and can afford |
| 11 | build catapult if archery range isn't busy, can afford and has the tech for it |
| 12 | build spearman if barracks isn't busy and has tech for it and can afford |
| 13 | build swordsman if barracks isn't busy and has tech for it and can afford |
| 14 | Research advanced technology if doesn’t already have it and can afford |
| 15 | Build stables if there are no free stables, can afford and an archery range has been built |
| 16 | build archery range if there are no free archery ranges, a barracks has been built and can afford |
| 17 | build barracks when it can be afforded and there are no free barracks |
| 18 | Research sword technology if can afford and doesn’t already know |
| 19 | Research spear technology if can afford and doesn’t already know |
| 20 | Research crossbowman technology if can afford and doesn’t already know |
| 21 | Research catapult technology if can afford, doesn’t already know and has advanced tech |
| 22 | Research knight technology if can afford, doesn’t already know and has advanced tech |
| 23 | Build tower if controls a tower point |
| 24 | Build wall if controls a wall point |

## 3.3 Genetic Algorithm AI

The genetic algorithm AI chooses the action it is going to carry out in a different way from the rule based system. At the beginning of the initial training, a list of actions is created for the genetic algorithm for each of the original genetic AIs. The list corresponds to the idea that genetic algorithms represent chromosomes and that actions are individual genes. Each action in the list has a position and a value, the position refers to the order in which the actions will be carried out and the value refers to the type of action that the AI will attempt to process. This list is randomised using the Unity Random.Range function which generates an integer when given two other integers. Each integer represents an action, for example, if one was the result then the corresponding action would be to build a worker. If an action proves invalid for any reason, the AI will move onto the next action without trying to repeatedly attempt the failed action. Full details of the types of actions available can be seen in appendix 5. An initial list of five thousand actions is created and if this list runs out then more actions are generated randomly. The list of five thousand actions is carried out in order. Each action in the list has a position and a value, the position relates to the order it will be carried out in, while the value represents the action to be carried out such as if the value is a one, the action is to build a worker. An example of what a segment of this list looks like can be seen in figure 6 which shows three different lists for actions 124-132, this means they will be carried out in order with 124 being the one hundred and twenty fourth action that the AI tries to carry out. If an action succeeds or fails for any reason, it will move onto the next action which in this case would be action 125. After ten games have been played, using different randomly generated lists the fitness function will evaluate the effectiveness of the list of actions to select individuals for the mating pool. Each candidate is given a fitness factor that is used to compare candidates with the candidates with the highest fitness factor being chosen for crossbreeding by using the calculations below.

if (total money / number of actions > cost of most expensive unit/building

fitness factor = fitness factor - (total money / number of actions Count - cost of most expensive unit/building)

if the AI won

Fitness factor = fitness factor + 10,000

Fitness factor = fitness factor – game time in seconds

If the AI lost

Fitness factor = fitness factor + game time in seconds

Fitness Factor = Fitness Factor + damage dealt during game by genetic AI / game length in seconds

Fitness Factor = Fitness Factor – enemy player’s headquarters’ health (full health is 5000)

The calculations were decided based upon observations made throughout the development of the genetic algorithm. The reason 10,000 is added to the fitness factor when the genetic algorithm AI wins is to reflect the importance of victory to the fitness function, it just needs to be a large number, the exact value is unimportant. The other calculations are designed to be of a similar scale with one another as they are of similar importance.

The fitness function has winning the game as the most important factor when evaluating candidates (Fernández-ares et al 2013). In an RTS having excess resources but not spending or having no way to spend them, is inefficient as the resources could be used to create more military units to aid in the AI’s victory. Therefore, having more money than the most expensive unit results in a lower fitness factor. Other factors include how long the game took; with the faster wins accounting for more, dealing damage to the enemy headquarters, surviving for a longer game and how much damage per second (DPS) was dealt on average through the game. These factors were chosen to try to ensure that the longer lasting versions of the AI would be most likely to be chosen, but if the AI wins it will be chosen regardless of other factors. The fitness function works by giving each candidate a score based upon the above factors then the two highest scoring candidates are added to the mating pool to be bred together during the crossover stage to create the child candidates for the next phase of games. The crossover occurs by taking alternating actions from the parent candidates to create a new child candidate that has properties of both the parents. An example of this process for the range 124 to 132 can be seen in table 2.

Table 2 - Genetic AI Crossover stage

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Candidate | …Action 124 | Action 125 | Action 126 | Action 127 | Action 128 | Action 129 | Action 130 | Action 131 | Action 132… |
| Parent 1 | 12 | 9 | 4 | 2 | 1 | 5 | 16 | 12 | 15 |
| Parent 2 | 5 | 2 | 9 | 11 | 15 | 17 | 18 | 14 | 7 |
| New candidate | 12 | 2 | 4 | 11 | 1 | 17 | 16 | 14 | 15 |

A small percentage of the new candidate’s actions is mutated to prevent stagnation as otherwise this could lead to a situation where the path the genetic algorithm chooses leads to a dead end. This is known as a local optimum. The mutation is carried out by re-randomizing 2% of the actions in the list using the method that was used to create the initial population. This is carried out ten times to create a new population. The new population then attempts to defeat the rule based AI and the process repeats as shown in figure 6.

Initialize the Population with ten candidates

For each candidate play a game

Evaluate candidates using fitness function

Perform crossover

Generate new population

Perform mutation on population

Figure 6 – Genetic algorithm process

This method of crossbreeding and mutating candidates is different from that described in the literature review. This was done because the normal methodology of genetic algorithms was designed in terms of Academic AI as opposed to real world applications for AI. The aim of this project was to create an AI for RTS games and the crossbreeding method used in this project was changed so that it would be more suitable for games. The reason for this being so important is that the chromosome of the genetic algorithm is the same as the list of actions carried out by the AI meaning the order of the actions is just as important as the actions themselves. The early parts of the chromosome correspond to the actions that the AI will carry out at the beginning of the game, and the later parts of the chromosome correspond to the actions for the later parts of the game. Therefore, changing the order by cutting out segments of the chromosome would not create a suitable crossbreeding. Instead, by merging the actions alternatively as shown in figure 6 it takes qualities from both candidates at every chronological point of the gameplay. The object of this is to increase the chance that the randomly chosen actions will be correct and the object of the fitness function is to evaluate if the randomly actions are correct.

Once the genetic AI can beat the rule based AI consistently, more complexity was added to the game with more unit types with varying combat abilities and different buildings to train them from. In addition to this, it has been made more difficult for a player to build the more powerful units by adding research requirements that mean the player must make an additional investment of resources. Carrying out research occurs from a building and means that building cannot train units during that time. This means that the player will be at a temporary military disadvantage during gameplay while doing research, therefore the genetic algorithm must only carry out research when it will not lose. Similarly, it is necessary for the genetic algorithm to find the most optimal timing to build an expansion. If the expansion is built too quickly, they will be rapidly overrun by the opposing player’s units and lose the game. If it is built too slowly, then the opposing player will have more resource and therefore more units than the genetic algorithm AI. This timing is a critical area for which the genetic algorithm must find the best expansion timing or they will ultimately lose.

To further increase complexity and reduce brute force luck, the map was changed to provide multiple avenues of attack and defence, meaning there is more that the AI must get correct to win. Other changes were made such as walls and towers which respectively provide a temporary blockage for enemies or attack enemy units. More complexity means more choices for the AI therefore the initially random choices made by the genetic algorithm are less likely to be correct initially and it must learn how to play through evolution by being evaluated using the fitness function.

# 4. Results

This section will discuss the results produced from the application created as described in the methodology. Data will be shown that represents the success or failure of the genetic algorithm in learning how to defeat the opposing player as well as the effectiveness of the fitness function. To begin with, one hundred iterations of the genetic algorithm were carried out, where each iteration involves the genetic AI playing the rule based AI for ten games. After each ten games the genetic algorithm evaluates using the fitness function which versions of the AI proved most successful by assigning a fitness factor as described in the methodology. The two most successful candidates are bred and mutated to create a new generation and this process as shown in figure 6 is called an iteration. The rule based AI carried out the precise same actions every game without changing as it is designed to do and there are no results to show for that. Each iteration produces two successful candidates. Figure 7 shows the fitness factor over time for these successful candidates side by side with there being two hundred candidates for the one hundred iterations carried out during testing.

Figure 7 – Fitness factor over time

As can be seen, early on the fitness factor is negative to represent that the genetic AI is failing to meet any of the fitness function’s success criteria that were described in the methodology. As time goes by the most successful versions of the AI are identified and the fitness factor begins to rise over time. The large spikes on the graph represent the genetic AI winning a game as that is the most important factor and the main means of defining success. The smaller spikes represent the algorithm finding other factors which from observation of the gameplay, were found to be the genetic AI dealing damage to the opposing player’s headquarters. Towards the end, the algorithm has identified the actions which align with the fitness function’s success criteria and it is consistently winning games against the rule based AI.

Figure 8 shows how the genetic algorithm was steadily learning and moving towards a more successful outcome early during the testing. This graph is a zoomed in version of figure 7 and shows the same data but only for the first sixty iterations. It starts with a large negative number to reflect that no damage was being dealt to the opposing player’s headquarters then proceeds to steadily improve towards a higher fitness factor. This reflects that despite neither winning nor dealing damage towards the opposing player’s headquarters, it was still managing to meet some of the success criteria. Like figure 7, it shows the fitness factor for both parent candidates therefore there are two fitness factors per iteration.

Figure 8 – Early Fitness Factor

Figure 10 shows the win rate of all games played per iteration of the genetic algorithm. One hundred iterations were carried out with ten games being played for each iteration. Figures 8 and 9 only show the fitness factor for the successfully chosen candidates with the highest fitness factor but figure seven shows the win rate for all ten games. A 0% win rate would mean that no games were won by the genetic algorithm AI while a win rate of 100% means that the genetic algorithm won all ten games in that iteration. As can be seen, early on the genetic algorithm does not win any games therefore the win rate is 0%. Towards the end of the hundred iterations carried out during testing, the genetic algorithm AI proved more successful with a steadily higher chance of winning throughout the last 150 games played. The genetic algorithm reached a peak of a 90% win rate towards the end, meaning it won nine out of the ten games that it played during that iteration.

Figure 9 – Win Rate

During the course of gameplay, several observations were taken on the actions made by the genetic algorithm AI. Early iterations would tend to result in the genetic algorithm AI losing quickly against the opposing rule based AI without surviving for long or providing a meaningful challenge against the rule based AI. As time passed the genetic algorithm AI would last longer and deal more damage over the course of a game. One particularly notable observation is that the genetic AI would predict the direction of the rule AI’s units by sending its own units to attack in that direction. For example, the rule based AI attacks from the left side at the very start of the game. To counter that, the genetic algorithm AI would send its own units towards the left pathways of the map (figure 3) preventing attacks from that direction meaning it survived for longer. A further side effect of this was that because the genetic algorithm AI was sending units down the left pathway, sending even more units along that path would lead to the AI damaging and eventually destroying the opposing player’s headquarters. This was an example of a chain of actions that showed how surviving in the short term leads to ultimate victory for the genetic algorithm AI.

As the genetic algorithm AI initially had a completely random set of actions to begin with, it could clearly be observed that it was training more resource gathering units than was optimal. The gameplay works that for every expansion building or headquarters a player controls, they can have twenty workers effectively gathering more resources to be spent on units, giving a cap of twenty workers at the start. However, the genetic algorithm AI was training far too many workers wasting resources. As the testing progressed, the wasted resources decreased as the number of workers increased more slowly when above the worker cap which is equal to twenty workers for every headquarters or expansion a player owns. After reaching the cap the player no longer gains any resources for workers above the cap. Notably in the games where the genetic algorithm built an expansion meaning a higher cap of forty or sixty, it would build more workers.

Other observations made include that the genetic algorithm AI was more successful when it built more archer units from the archery range and less infantry from the barracks. It tended not to build an expansion in the vast majority of games, though there was a noticeable increase in the chance it would build one expansion in a game it was winning. It never built more than one expansion.

# 5. Discussion

## 5.1 Genetic Algorithm

This section will discuss the results that were expected and explain how they may differ from those produced by the project and why. It will also discuss how the work is relevant to previous pieces of work on the same topic. Success of the project will be measured by the genetic algorithm being able to identify the successful gameplay choices made by the AI and then be able to replicate them in future games without much change occurring. Ideally, the genetic AI would utilise all the features of the game, training a variety of different units and buildings. The AI should focus on the economic aspect by building expansions and training workers, as well as the military aspect by training combat units and constructing military buildings. However, as the fitness function’s largest factor is whether the genetic AI has won, and using all the game features is not a factor at all, it was expected that this will not happen. Instead, it was expected that the genetic algorithm would favour fast victories that rely very little on defence. This was expected because the fitness function takes victory as the most overwhelmingly important factor.

One of the original factors for quantifying the success of a candidate was comparing the amount of damage dealt by the genetic algorithm AI to the rule AI’s units and buildings. This method proved as an unsuccessful method as it just created a large number without taking into account other things going on in a game and was replaced by two other factors. The first factor was found by measuring the average DPS dealt by the genetic algorithm AI where high damage gave a higher rating in the fitness function. Increasing the magnitude of this factor appeared to have a positive effect on the success of the AI, with the more successful versions of the AI having a higher DPS. However, it is it difficult to ascertain whether this is a correlation or causation as it is possible that while more successful candidates did deal more damage, it is not clear as to why. In some circumstances, it was noticeable that the DPS was particularly high in short but otherwise successful games suggesting that perhaps the AI was dealing a high volume of damage to opposing player’s headquarters, inflating the amount of damage dealt. This would be considered a positive quality.

The other factor added to replace the total damage dealt was subtracting the remaining health of the rule based AI’s headquarters as part of creating the fitness factor in the fitness function. This is the reason why in figures 6 and 7 the fitness factor is a negative number for the majority of the time. As the genetic algorithm identifies the more successful candidates, any damage dealt to the opposing player’s headquarters amounts to a large change in the fitness factor causing a snowballing effect. This leads to a situation to where the better it plays, the more likely it will deal more damage to the opposing player’s headquarters meaning it will have an even higher fitness factor. It can be seen in figure 8 in the results section that there was a large increase to the fitness factor at certain stages of the testing, through observation this was noted to be the times where damage was being dealt to the opposing headquarters. This shows that there is a clear correlation between victory and dealing damage to the opposing team’s headquarters. It should be noted that victory itself accounts for twice as much in the fitness function as dealing damage to the opponent’s headquarters.

From the results, it is clear that the victory component to the fitness function only triggers when the genetic algorithm AI wins the game. This means that during early iterations of the AI, it is entirely reliant on other factors before reaching the point where it can win. Dealing damage to the headquarters is one of those factors but this also is affected by the same issue in that if the genetic algorithm AI loses quickly it won’t have any way of measuring success. The other parts of the fitness function successfully help alleviate this issue as shown by figure 8 in the results. This graph shows that the fitness factor rose over time without input from the most important sections of the fitness function but also that it takes a long time before the genetic algorithm AI was good enough to win. Instead, the victory based component (Fernández-ares et al 2013) ensured that once the genetic algorithm had learned how to win, it maintained that ability to find victory.

The effect of the game length on the fitness function worked different at different stages of development of the project. The original method was that when the genetic algorithm AI was winning, longer games would result in a lower fitness factor as the idea was that winning quickly was a desirable trait. This did in fact happen, with the genetic algorithm AI tending towards shorter games when it won. The side effect of this process was that the AI would focus entirely on an offensive strategy, with only the most minimal defence needed to survive long enough to destroy the rule based AI’s headquarters. This all out aggressive strategy while effective at winning the game, is not ideal in terms of playability were a human player involved. It almost seemed like the genetic algorithm was learning how to exploit a rule based AI like a human player would. This remarkably humanlike behaviour is interesting as one of the goals of the project was for more humanlike behaviour from an AI. However, this kind of humanlike behaviour would not be conductive to an enjoyable experience if a human player were to play the game.

Another method to use game length as a measurement of how successful the AI performed in a game was taken by accounting a longer game as a positive factor for success when the genetic algorithm AI lost the game. This decision was made after testing and finding the problem mentioned above. Longer games would mean that the genetic algorithm AI has survived for a longer period of time therefore must be doing better than if it lost quickly. The downside of this approach could be that the AI could be locked into a local minima where it pursues a defensive strategy but is unable to win the game. Between this and preferring quicker game times when it wins, a balance was found that allows it to avoid this issue though it still preferred the shorter more aggressive playstyle. It is possible that the defensive strategy was also avoided due to the negative fitness factor from not dealing damage to the opposing player’s headquarters as in early testing candidates acted defensively. This further emphases the importance of the refinement of the fitness factor in order to make a successful genetic algorithm.

All these methods proved to work successfully together, some to greater extents than others. The victory based component and the opposing player’s headquarters’ health component were both useful for identifying success when it occurred. However, when neither of these factors came into play it was the DPS and time components combined with mutation that led to the possibility of the genetic algorithm AI winning a game. This meant that it took a long time for the genetic algorithm AI to find the correct actions but when a useful mutation was generated it would be quickly identified and replicated throughout subsequent candidates. To improve on the time taken for the genetic algorithm to succeed, further methods of determining success such as number of workers could be implemented.

For the genetic algorithm AI to win, the AI had to find the correct actions and then carry them out in the correct order. One of the things noticed during testing that as the order of the actions is just as important as the actions themselves the genetic algorithm relies on the mutation to some extend to find the right order. In the future in may be useful to have a higher initial mutation level between generations of candidates then as the genetic algorithm AI proves more successful, to reduce the rate of mutation. Another option would be to entirely rely on mutations and remove the crossover stage, this would mean the fitness function would be evaluating the viability of different mutations. These would be an adaption of the genetic algorithm theorised by Holland, 1975 that could be a way to improve on the methodology specifically for RTS games. At the same time, it may also be an option to add another factor to the fitness function that takes into account the order of actions more directly showing that the fitness function is critical to the optimal operation of the genetic algorithm.

## 5.2 Rule Based AI

The rule based AI’s fixed ruleset meant that as expected it could not adapt to any changes in tactics by the genetic algorithm AI. Therefore, during development the rule based AI’s ruleset was constantly changed and updated to make it more flexible and be able to utilise any new features that were added to the game. The rule set of the rule AI has constructing buildings as one of the lowest priorities and is only carried out in a situation where all other unit producing buildings are occupied. This proved successful in ensuring that the rule AI was using its resources as effectively as possible in additional to maximising the output of all buildings. Another area which proved successful was in ensuring the AI always trained workers to the maximum effective number, this meant that at the very beginning of a game the rule based AI had more resources to spend than the genetic algorithm AI which would sometimes not build workers at the most optimal times.

One of the priorities for the rule based AI was to maximise resources output, to do so, it would save up resources until it could construct an expansion building. This is a large investment of resources that could otherwise be used to obtain a short term gain of additional military units. The rule based AI seeks to build up to two expansion buildings when it controls any of the four expansion locations on the map (figure 3). Building up to two expansion buildings was chosen as there are two expansion points next to each player and during testing it was shown that the genetic algorithm AI would destroy any expansions built by the rule based AI next to the genetic algorithm AI’s headquarters. The building of expansions allows for more worker units to gather resources which in turn can be spent on more military units, meaning a short-term loss for a long-term gain. This was the main loophole in the rule based AI’s defence as over a long game it could outproduce the genetic algorithm AI in units but in the short term it allowed the genetic algorithm AI to play aggressively to destroy the headquarters of the rule AI. This again proves that a rule based system is only as good as the rules that have been set.

## 5.3 Gameplay

The decision to automate pathfinding using a waypoint system proved very successful using the same design described by Liaw et al 2013 for the FPS Quake. This meant the genetic algorithm could focus on the macroscale which is an RTS term meaning in the training of units and constructing of buildings while allowing the micromanagement of individual units to be carried out by a separate system. By letting the genetic algorithm to concentrate on the bigger picture it meant that it was better targeted on a more specific problem and could find the more effective solutions faster with less incorrect solutions being found. The rule AI used the same waypoint system as the genetic algorithm AI as this was considered one of the game rules as opposed to an AI decision. The impact the players had on the unit movement was limited to the direction on the map that units were sent. Units always moved towards the opposing player’s headquarters, meaning there was no backtracking on the map. This helped ensure the game was always moving forward and that the players were always in conflict, unable to hold units in defence. This likely benefitted the genetic algorithm AI as it was not designed to handle the micromanagement of many units.

Against a human player the optimal solution may not be ideal as an unbeatable AI is not fun to play against, therefore it may be necessary to add a deliberate level of error into the AI by selecting a suboptimal solution. This is done by outputting the list of actions to a file, which can be read from and used to recreate that list of actions at a later date. Commonly in RTS games there is the option for different difficulty settings, this could be done by using different solutions to represent easy, medium and hard. This will hopefully create a more realistic computer controlled player that can make mistakes as a human player can, but also capable of winning. The current version may be suitable for this purpose as it is still imperfect and loses some of the time after significant training, though further testing would need to be carried out against a human player. One solution would be to have a human player play a few times against it and record the actions that the human player carried out. The genetic algorithm AI could then train against the recorded actions several times to create different versions of the genetic algorithm AI.

A change made late in development was the ability to build more than one of a building type, for example they could have three barracks rather than just one. At the same time the resource costs of all units was increased and the resources generated by each worker was increased. This change was made to create a situation where to maximise unit production, a player would always need to ensure they had sufficient worker units gathering resources and have enough buildings to spend the resources. It allows for more flexibility in strategy for both AI and helped to identify any flaws in the balancing of effectiveness of units. One such flaw found during development was that ranged units were entirely ineffective in combat. This led to a situation where the rule based AI would train ranged units as the design was that they would be effective when protected by other units. Instead, the range was too short and melee units would destroy ranged units too effectively. A change was made to make melee units only deal damage when in immediate proximity to other units and buildings while significantly increasing the distance from which a ranged unit could attack from. As this change was made, it provides alternate strategies and more options for the players to make, creating a more realistic simulation of an actual RTS.

To summarise, the project proved successful with the main aims being met with the research and investigation elements being carried out before development began. The development of a game that allowed an AI using genetic algorithm to learn how to play against a rule based system used in an RTS was successful. The genetic algorithm learned how to win against the fixed rule set of a rule based system using a fitness function that defined victory as the most important factor. During development, it was shown that all the factors were important in ensuring that the genetic algorithm could identify candidates that were more successful than others.

# 6. Conclusion and Future Work

The project proved successful meeting the aims and objectives stated in the introduction having created a genetic algorithm that could be used to teach an AI how to play an RTS game. This strongly suggests that in a real situation a genetic algorithm could be used in an RTS game to train an AI how to play. The genetic algorithm was capable of starting with a completely random set of actions in a random order then the fitness function would evaluate the candidates and it would find the most successful candidates over time. The game itself was made to be similar to the game Age of Empires to simulate how effective a genetic algorithm was in a real game and was an accurate simulation from the point of view of testing the effectiveness of game AI.

One of advantages of the application developed when applied to games when compared to the typical version of genetic algorithms described by Holland, 1975 is that the chromosome and the list of actions the AI carries out are the same thing. This means that is very simple to implement new types actions for the AI as each action is represented as a number and it is the game code that decides what those numbers mean. From a game development perspective, this is very useful as it ensures there is a layer of separation between AI and the gameplay that was implemented using the finite state machine. As this methodology is different from the more typical version of genetic algorithms it may be useful in future works to describe it as an evolutionary algorithm instead of a genetic algorithm as while it does share properties with genetic algorithms it may not be the most accurate label.

The next step in the process of using genetic algorithms in RTS games would be create a game where a human player is involved in the real-time decision-making process. This would mean that rather than the genetic algorithm facing against a rule based AI as was done in this project, it could then train against a person. As the learning stage of the genetic algorithm as carried out in this project took a thousand played games of the RTS it would be impractical for a human player to play all thousand games. Instead, two human players could play against each other, and the genetic algorithm plays against copies of each of those two players. This allows for a more realistic training partner while not requiring someone to play a thousand or more games. In a currently existing RTS such as Starcraft II players play on the internet against other players every day and it would possible to use those games as the copies for training. It also has the additional advantage of providing a very large pool of data for which to train the genetic algorithm on to make it more multipurpose and to create a perfectly optimised AI. However, perfection is not desired as human players are not perfect and an unbeatable AI is not wanted so suboptimal solutions would be necessary.

The current version of the project while effective with its parameters, would not be suitable without changes to make it more functional in a real situation. Rather than training a genetic algorithm against a rule based AI as done in this proof of concept, it may be more useful and accurate to train the genetic algorithm against other genetic algorithms. Different genetic algorithms could have different weightings in the fitness function and more experimentation could be carried out to what is the most effective way of evaluating success when the genetic algorithm does not win. Upon training several different versions of the genetic algorithm AI the most successful ones could be trained against a human player to find which genetic algorithm AI can play the best. It could also be possible to swap the AI and the list of actions with another AI partway through gameplay to create a dynamic difficulty that is more unpredictable. Doing this would provide useful information in evaluating the best method of creating a suitable AI for an RTS.

One of the conclusions of this project is that the victory based component is a valuable contribution to creating the genetic algorithm. Currently the victory based and the damage dealt to the enemy headquarters components of the fitness function are very good at identifying how well the AI is playing. However, these components only come into play when the genetic algorithm AI has already found some success. This suggests that future work could be done to evaluate the other measures of success during gameplay, perhaps by identifying patterns of gameplay that are effective like building workers at the start of the game which is always correct. A method of evaluating success when the genetic algorithm AI has not won could be to implement rules of what actions are successful together, such as if the genetic algorithm AI has built an expansion it would be evaluated more highly if it immediately trained workers. Another example would be by evaluating if buildings were being used to their full capacity, if the AI has built many unit production buildings like barracks but does not use them to train units it should be evaluated to be less effective. Further research should be carried out to maximise the potential of genetic algorithms in RTS games and to find other methods of identifying what should be considered in the fitness function other than victory.

# 7. Appendices

## 7.1 Units

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Unit Name** | **Health** | **Speed** | **Damage** | **Special** | **Requirement** | **Building** |
| Worker | Low | Medium | Low | Can gather resources | None | HQ |
| Warrior | Medium | Low | Low | None | None | Barracks |
| Spearman | Medium | Low | Medium | Good versus stable units | Barracks Spears | Barracks |
| Swordsman | High | Low | Medium | Expensive | Barracks  Swords | Barracks |
| Archer | Low | Medium | Low | Can attack targets at range | None | Archery Range |
| Crossbowman | Low | Medium | High | Can attack targets at range  Expensive | Archery Crossbows | Archery Range |
| Horseman | Medium | High | Medium | Fast | None | Stable |
| Knight | High | High | High | Best Unit | Stables Feudalism | Stable |
| Catapult | Low | Low | Low | Can attack targets at range  Deals bonus damage to buildings | Archery Range Ballistics | Archery Range |

## 7.2 Buildings

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Builds** | **Special** | **Requirement** |
| Headquarters | Workers  Catapult | If this is destroyed, you lose  Building Upgrades | None |
| Barracks | Warriors  Swordsman  Spearman | Infantry Upgrades | None |
| Archery Range | Archer  Crossbowman  Catapult | Archery Upgrades | Barracks |
| Stables | Horseman  Knight | Cavalry Upgrades | Archery Range |
| Expansion | Nothing | Can gather more resources here with workers | None |
| Walls | Nothing | Blocks Enemy Units  Can be upgrades with towers | none |
| Tower | Nothing | Shoots Enemy units | None |

## 

## 7.3 Technology

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Building** | **Requirement** | **Unlocks** |
| Advanced Research | Headquarters | Stables | Towers, Swords, Ballistics and Feudalism |
| Spears | Barracks | None | Spearman |
| Swords | Barracks | Spears  Advanced Research | Swordsman |
| Crossbows | Archery Range | None | Crossbowman |
| Ballistics | Archery Range | Crossbows  Advanced Research | Catapults |
| Feudalism | Stables | Advanced Research | Knights |

## 7.4 Rule Based AI Priority List

|  |  |
| --- | --- |
| Priority | Rule |
| 1 | Build workers if has less than 10 and headquarters not busy |
| 2 | Build expansion if controls an expansion point, has less than three expansions and had not reached the worker cap |
| 3 | Build stables if it doesn’t own a stable and has an archery range |
| 4 | Build archery range if it doesn’t own an archery range, it can afford it and has a barracks |
| 5 | Build barracks if it doesn’t own a barracks and can afford one |
| 6 | Build warrior if there is a free barracks and can afford it |
| 7 | Build workers if below cap and headquarters isn’t busy |
| 8 | build knight if stables isn't busy and can afford |
| 9 | Build horseman if stables isn’t busy and can afford |
| 10 | build crossbowman if archery range isn't busy, has the tech for it and can afford |
| 11 | build catapult if archery range isn't busy, can afford and has the tech for it |
| 12 | build spearman if barracks isn't busy and has tech for it and can afford |
| 13 | build swordsman if barracks isn't busy and has tech for it and can afford |
| 14 | Research advanced technology if doesn’t already have it and can afford |
| 15 | Build stables if there are no free stables, can afford and an archery range has been built |
| 16 | build archery range if there are no free archery ranges, a barracks has been built and can afford |
| 17 | build barracks when it can be afforded and there are no free barracks |
| 18 | Research sword technology if can afford and doesn’t already know |
| 19 | Research spear technology if can afford and doesn’t already know |
| 20 | Research crossbowman technology if can afford and doesn’t already know |
| 21 | Research catapult technology if can afford, doesn’t already know and has advanced tech |
| 22 | Research knight technology if can afford, doesn’t already know and has advanced tech |
| 23 | Build tower if controls a tower point |
| 24 | Build wall if controls a wall point |

## 7.5 List of Actions

|  |  |
| --- | --- |
| **Action** | **Value** |
| None | 0 |
| Train Worker | 1 |
| Build Barracks | 2 |
| Train Warrior | 3 |
| Train Archer | 4 |
| Build Archery Range | 5 |
| Train Knight | 6 |
| Build Stables | 7 |
| Build Tower | 8 |
| Send units to Left waypoints | 9 |
| Send units to Right waypoints | 10 |
| Research Advanced Research | 11 |
| Train Swordsman | 12 |
| Train Spearman | 13 |
| Train Crossbowman | 14 |
| Train Horseman | 15 |
| Train Catapult | 16 |
| Train Wall | 17 |
| Build Expansion | 18 |
| Research Swords | 19 |
| Research Spears | 20 |
| Research Catapults | 21 |
| Research Knights | 22 |
| Research Crossbowman | 23 |

# 8. References

Aizawa, K*.* 2013.Warren McCulloch’s Turn to Cybernetics: What Walter Pitts Contributed *Interdisciplinary Science Reviews.* 27 (3). Pp. 206-217.

BBC. 2016. [online]. Google’s AI wins final Go challenge. Available from: <http://www.bbc.co.uk/news/technology-35810133> [Accessed March 2016]

Blizzard Entertainment. 2002. Warcraft III: Reign of Chaos. [disk]. PC. Blizzard Entertainment.

Blizzard Entertainment. 1997. Starcraft. [disk]. PC. Blizzard Entertainment.

Blizzard Entertainment. 2010. Starcraft II. [disk]. PC. Blizzard Entertainment.

Burnett, C. 2006. [online image]. Available from: <https://commons.wikimedia.org/wiki/File:Artificial_neural_network.svg> Access April 2017.

Ensemble Studios. 1999. Age of Empires II: Age of Kings. [disk]. PC. Microsoft Corporation.

Fernández-ares, A., et al. 2013. [online]. Analysing the influence of the fitness function on genetically programmed bots for a real-time strategy game. *Entertainment Computing*. Volume 18. Pp.15-29. Available from: <http://www.sciencedirect.com.libproxy.abertay.ac.uk/science/article/pii/S1875952116300222>

Holland, J.H. 1975. Adaptation in natural and artificial systems. USA. University of Michigan Press.

King, D. 2016. *Genetic Algorithms.* [online image]. Dundee. Abertay University. Available from: <https://blackboard.abertay.ac.uk/webapps/blackboard/execute/content/file?cmd=view&content_id=_388273_1&course_id=_6156_1> Accessed October 2016.

Liaw, C. et al 2013. [online]. Evolving a team in a first-person shooter game by using a genetic algorithm. *Applied Artificial Intelligence.* 27(3). Pp.199-212. Available from:

<http://www.tandfonline.com.libproxy.abertay.ac.uk/doi/abs/10.1080/08839514.2013.768883>

Ligêza, A. 2006. Logical Foundations for Rule-Based Systems. USA. Springer.

Liu, C.H. and Huang, D.H. 2012. Reduction of power consumption and carbon footprints by applying multi-objective optimisation via genetic algorithms. *International Journal of Production Research.*

Mardle, S., and Pascoe, S. 1999. An overview of genetic algorithms for the solution of optimisation problems. *Computers in Higher Education Economics Review.* 13(1). Available from: <https://www.economicsnetwork.ac.uk/cheer/ch13_1/ch13_1p16.htm> [Accessed October 2016]

McCulloch, W. and Pitts, W. 1943. A Logical Calculus of the Ideas Immanent in Nervous Activity. *The Journal of Symbolic Logic.* Volume 5. Pp. 115-133.

[Maulik](http://link.springer.com.libproxy.abertay.ac.uk/search?facet-creator=%22Ujjwal+Maulik%22), U., [Bandyopadhyay](http://link.springer.com.libproxy.abertay.ac.uk/search?facet-creator=%22Sanghamitra+Bandyopadhyay%22), S., [Mukhopadhyay](http://link.springer.com.libproxy.abertay.ac.uk/search?facet-creator=%22Anirban+Mukhopadhyay%22), A. 2011. Multiobjective Genetic Algorithms for Clustering. Berlin. Springer-Verlag Berlin Heidelberg.

Mora, A., et al. 2012. Effect of Noisy Fitness in Real-Time Strategy Games Player Behaviour Optimisation Using Evolutionary Algorithms. *Journal of Computer Science and Technology.* 27(5). Pp.1007-1023. Available from: <http://search.proquest.com.libproxy.abertay.ac.uk/docview/1197182214?OpenUrlRefId=info:xri/sid:primo&accountid=8159>

The Guardian. 2016. [online]. 'Partnership on AI' formed by Google, Facebook, Amazon, IBM and Microsoft. Available from: <https://www.theguardian.com/technology/2016/sep/28/google-facebook-amazon-ibm-microsoft-partnership-on-ai-tech-firms> [Accessed October 2016]

Wang, P., Ruan, D. and Kerre, E.. 2007. Fuzzy Logic: A Spectrum of Theoretical & Practical Issues. *Studies in Fuzziness and Soft Computing.* Volume 215. Pp 1-13.

# 9. Bibliography

Amazon. 2017. [online]. Amazon Machine Learning. Available from: <https://aws.amazon.com/machine-learning/> [Accessed February 2017]

Firaxis Games. 2010. Sid Meier’s Civilization V. [disk]. PC. 2K Games

IBM. Frequently Asked Questions: Deep Blue. [Online]. Available from: <https://www.research.ibm.com/deepblue/meet/html/d.3.3a.html> [Accessed October 2016]