Evolution in an open world RPG

Abstract

Abstract text

Table of Contents

[Chapter 1: Introduction 4](#_Toc307064383)

[Chapter 2: Background 6](#_Toc307064384)

[2.1 History of Table Top Role Playing Games 6](#_Toc307064385)

[2.2 General Features of a Table Top Role Playing Game 6](#_Toc307064386)

[2.3 History of Computerised Role Playing Games 7](#_Toc307064387)

[2.4 The General Features of a Computerised Role Playing Game 8](#_Toc307064388)

[2.4.1 Features of a Standard Computerised Role Playing Game 9](#_Toc307064389)

[2.4.2 Features of an Open World (aka Sandbox) Role Playing Game 10](#_Toc307064390)

[2.5 Primitive Artificial Intelligence 10](#_Toc307064391)

[2.6 Adaptive Behaviour 11](#_Toc307064392)

[2.6.1 Online and Offline Learning 11](#_Toc307064393)

[2.6.2 Supervised and Unsupervised 11](#_Toc307064394)

[2.7 Scripting 11](#_Toc307064395)

[2.8 Dynamic Scripting 12](#_Toc307064396)

[2.9 Finite State Machine 14](#_Toc307064397)

[2.10 Neural Networks 15](#_Toc307064398)

[2.11 Genetic Algorithms 16](#_Toc307064399)

[2.12 Rejection and roulette sampling 18](#_Toc307064400)

[2.13 Background on computer games RPG 19](#_Toc307064401)

[2.14 Individuality 21](#_Toc307064402)

[Chapter 3: Problem statement 22](#_Toc307064403)

[Chapter 4: Description of System 23](#_Toc307064404)

[4.1 Description of Animals 24](#_Toc307064405)

[4.2 Description on Environments 25](#_Toc307064406)

[Asdasdas 25](#_Toc307064407)

[4.3 A model that simulates simple life 27](#_Toc307064408)

[4.4 A model that can be used by game content creators 27](#_Toc307064409)

[4.5 Fleeing and knowing whom to flee from 27](#_Toc307064410)

[4.6 Breeding and generic fitness function 28](#_Toc307064411)

[4.7 Rejection sampling for breeding new generations 28](#_Toc307064412)

[4.8 How the map and the creature are stored 29](#_Toc307064413)

[Chapter 5: Empirical Results 32](#_Toc307064414)

[Chapter 6: Further Research 33](#_Toc307064415)

[6.1 Balancing the eco-system 33](#_Toc307064416)

[6.2 Potential of using a neural network to control how the creature behaves 34](#_Toc307064417)

[6.3 More in depth simulation of the creatures needs 34](#_Toc307064418)

[6.4 Simulating a scent to choose mating partners, being alerted to enemies and finding food 34](#_Toc307064419)

[References 37](#_Toc307064420)

# Chapter 1: Introduction

Video games have been around for over 50 years almost as long as the programmable computer, and in that time thanks to an almost global availability of computers, massive increases in hardware power and new computer languages, almost all aspects of a video game have become significantly more complex and interesting.

However is lieu of huge advancements in hardware leading to greatly improved real time graphics, sound and physics, artificial intelligence (AI) has not seen the same advancement. The areas that have seen the most advancement for AI in games is with the first-person-shooter (FPS) genre, which is one of the most popular markets and relies on decent AI in order to make the game worth playing at all. Many of the advancements have all been related to path-finding as these games are about the player’s position in relation to other players and often finding items.

One recent game that made use of more advanced AI for its bots is Monolith Productions F.E.A.R, where the AI uses a combination of GOAP (Goal Oriented Action Planning) [1] [2] and a NavMesh [2] to create goals and ways of achieving those goals. Another game that came out recently is id’s Rage, where they’ve taken this to the next level where the motions of each enemy is much more fluid and unpredictable and even varies depending on the type of enemy.

On the other hand, other genres have not had such work done with their AIs. The genre this thesis will be focusing on will be role-playing-games (RPG) and in particular open world role-playing-games. These are identified by their large in game worlds where the player is often free to do as they wish, slowly unlocking new areas through completing quests and obtaining new skills. The origin for this genre comes from playing table top RPGs such as Dungeons & Dragons, a game where players use a rule book, a story teller and their imaginations to interact with in-game characters and the world environment.

Many game elements of these table-top games have made their way into computer games, such as the rules for how combat, spells and nearly all other interactions take place. Some have expanded on these game elements to large deformable terrain in games like Bay 12’s Dwarf Fortress and Mojang’s Minecraft. Both of these feature very large procedurally generated worlds populated with a multitude of different environments and creatures.

However as the player causes changes to the world, often only they are affected by such changes, they have little to no effect on any other creature. For example if the player were to remove an important food source such as a population of rabbits, this would no impact on the wolf populations which also in turn would have no impact on any other populations that are affected by the wolves. Furthermore there is no room for natural selection such that a player could potentially influence the genetics and thus the behaviour of a population of creatures should they so choose to in an open world style game.

These aspects would normally fall to the story teller’s imagination to solve, however it is not found in computer games, often because of their perceived complexity and unknown gameplay value factor.

As such this thesis aims to present a computerised solution this problem that should result in a more dynamic world that allows for situations such as this to have a solution in some form. This makes use of parameterised scripts based on the genetic make-up of each creature to change their behaviour, allowing each creature its own individuality as well as the potential for adaptation and distinct populations to emerge.

ROADMAP!

# Chapter 2: Background

While the genre of RPG is very diverse, we are going to focus on the history of the typical modern RPG that has given us the standard class based characters, typical RPG stats and levelling, and quest lines.

## 2.1 History of Table Top Role Playing Games

The origin of nearly all modern RPGs can be traced back to Dungeons & Dragons, first released in 1974 by Gary Gygax and Dave Arneson. Dungeons & Dragons deviated from the other RPGs available at the time which were primarily military simulations. These military simulations consisted of the players controlling armies of units and were often short lived campaigns with a simple set of rules in a small world. Dungeons & Dragons however gave the players control of a single unit that was their avatar in the fantasy world and everything else in the world is controlled by a specific played called the Dungeon Master (DM). This shifted the focus from the generally short campaigns played in a small section of a world with a narrow goal, to a more open ended world that is far more dynamic now that it the product of the DM players imagination; such that they controlled the flow of the adventure with it [1]. This was an important change as now the game had a user specific goal and it was the DMs role to guild the other players through a world or to complement what they wanted to do and to make it feel like a real place. This means that player actions can have long lasting consequences that may or may not be immediate or obvious.

## 2.2 General Features of a Table Top Role Playing Game

As Dungeons & Dragons has a storying telling element and is controlled by the imaginations of the players, it does have rules, lore and a certain style for how the game is played. Most games at the high level fall into two categories, either a linear story or an open ended world. Linear stories are generally shorter but this is not always the case, but they are generally the product of the DM telling a story of some kind, and depending on how narrow the story is, the players actions will affect how the story progresses and the actual ending. In an open world setting, the players generally go find an adventure and when an appropriate time presents itself, the DM will add in a story that may just be a short misadventure or perhaps a long drawn out journey and many of these can happen within a given game. Also depending on the open world game, there is no definite end unless one in agreed upon as even death in most of these games is temporary and reversible. Whilst in a short linear story, the game ends when the players reach the end of that story, although a linear story can still have subplots.

Another major feature of these games is the non-player-characters, often referred to as NPCs. These NPCs often have knowledge about the situation or turmoil in the game world and depending on the personality of these NPCs, the players may have to extract the information out of them. As the NPCs are controlled by the DM, the players can interact with them as they would a human and form friendships, rivalries or become enemies. There are no exact rules for how a DM controls these NPCs, only that they should appear reasonably realistic and have motivations for any particular actions they perform. This fact that such an important factor in a typical Dungeons & Dragons game is primarily controlled by the imagination of a human player doesn’t translate well into a computerised form.

Players are generally able to perform whatever actions they want and the DM in turn generally should react to these actions by having the game world react accordingly. Such as, should a particular player be playing an evil character they may poison a town’s water supply, which to us as humans we can predict the potential outcomes of such an action. However to a computer unless this action was accounted for by the game’s developers it would probably not even be an action that could be performed, let alone having any meaningful outcome. This is another aspect that does not translate well into a computerised form, the ability to perform any physical action that affects the player or the game world at large.

Another important factor when playing these games compared to a computerised form, is the fact that each player interacts with the world in turn. Generally only if there is a particularly lively event, each player tells the DM (and the other players) what actions they are performing, to which the DM then has to calculate the outcome of these actions and respond back to the players with the consequences.

## 2.3 History of Computerised Role Playing Games

The first commercially available computerised RPG was Starpath’s Dragonstomper for the Atari 2600 released in 1982. This game had many similarities of a Dungeons & Dragons game, where the player controls a single unit in a world and must improve their characters stats and equipment in order to advance through the story. It has many features of a table top game such as similar numerical game rules, NPCs, a story and a large game world for the player to explore.

## 2.4 The General Features of a Computerised Role Playing Game

As by the nature of a computer, many if not all of the numerically based rules have been successfully translated into a computer form for the use in RPGs. And as these rules were originally designed to be calculated by human players within the context of playing a game which means they cannot be too complicated or laborious to calculate, they can be calculated quite easily by a computer at a much higher rate than by a human. This has an important consequence that unlike a table top RPG that is played in turns and can at times take quite a while to complete a combat encounter, that the games could be played in real-time, where the computer takes advantage of different stats of different creatures to calculate how often they can attack and how often the players can attack. It is also important to note that not all computerised RPGs are played real-time, there are still many out there that choose to be played in a turn-based fashion to give the players’ time to think about what they are going to do, however this is by design rather than a limitation of the medium.

These computerised RPGs also still feature NPCs, however they are generally much more limited that what one would expect from a table top RPG. They are generally there just for selling items to the player or for providing some simple and often inane dialog, usually in a town that could be populated with many NPCs; only a select few will be of any importance. This is mostly due to the limitations of the medium, where getting an NPC to act like a human beyond repeating scripted dialog and walking around scripted paths can add a lot of extra effort. However some games have implanted a system typically referred to as Living World, where the NPCs have lives, where the go to sleep at night, wake up and go to their jobs, mingle with other NPCs and talk about daily life and then go back to their homes at night. While much of this is still scripted and the NPCs don’t actually have needs, it can still add a nice level of realism to a village or city, rather than a town filled with people who are stuck in one spot and who only repeat the same few lines of dialog.

Often the only way the world will react to the player outside of combat or dialog is through scripted events, this is again due to limitations of the medium. Where if the player started a fire in the corner of a town in any given RPG; most of the time nothing would happen, unless the developers had put in checks for such an event.

However there have been aspects of these games that have taken advantage of the computer, such as the graphics, sound and networking. These have allowed for the games to be quite immersive and while some would argue that this is simply taking away the use of your imagination, it is similar to the difference between reading a book and watching a film, they each have their own place. The other aspect networking has also made it possible that many people can play the same RPG together over the internet which again would previously be either very tricky to do using phones or video streaming and would also be limited to a few people. This has brought about a particularly popular sub-genre the MMORPG (Massively Multiplayer Online Role Playing Game), these are typically played by thousands of players at once in a very large game world.

Despite these advancements in computerised RPGs, the AI side has not changed much in these years. If you look at a modern computerised RPG you will still find that many creatures are in one of a few states, such as ‘Idle’, ‘Attacking’ or ,’Fleeing’. NPCs have also not changed much in these years except in the dialog side, many games do now offer complex dialog trees and the player’s character’s abilities can have an effect on what the NPC may be willing to tell the player. These can range from the character’s bluff skill, to their ability to intimidate which could both offer a different set of dialog from the NPCs; these however are still scripted in but they are an improvement.

### 2.4.1 Features of a Standard Computerised Role Playing Game

While in a table top RPG the difference between a linear story and open world RPG is typically not so clear cut, within the context of a computerised RPG the distinction is clearer, although not in all cases.

What is considered the standard computerised RPG is for the most part a linear story RPG that is primarily a story telling device. The player generally has very little ability to stray from the normal path and what the player has to do next; or rather what they can do next is made reasonably clear. This does however mean that the stories can be much more involved as the writer’s do not have to take into account every possible action the players may make ahead of time, unlike a DM who can react to players as they perform the actions.

Some of the major games of note in this series are the BioWare’s games Baldur’s Gate game series and Dragon Age: Origins which are both known for their story telling ability.

### 2.4.2 Features of an Open World (aka Sandbox) Role Playing Game

Open world RPGs in a computerised form are for the most part like their table top counterpart, the player chooses a character they wish to play as and set out into a world. Depending on the game they may have a clear main story with a single or multiple endings, or they may have several different stories that all result in a different endings. These are typically defined by the fact that the player has much more freedom to go about what they do and can go where-ever they want within the game world for the most part. However because of this freedom they do tend to have issues with giving the player direction and in telling a story. This is why they generally go for a more ‘sandbox’ feel, where the players can just do whatever they feel like within this game world and are able to create their own fun through interacting with the world however they want.

Some games of note from this series are Dwarf Fortress, Minecraft, New World Computing’s Might and Magic game series and Bethesda’s The Elder Scroll game series. Dwarf Fortress and Minecraft both only use randomly generated worlds and for the most part let the players create their own stories out of their experiences in those worlds, which gives the players a lot of freedom in what they do, but for the most part have no provided story.

Might and Magic, and The Elder Scrolls games however both offer worlds where the player is given some background to the game world and then set free in it. They can then choose to follow the main story or they can simple explore the world and find all its secrets.

## 2.5 Primitive Artificial Intelligence

In none of these computerised RPGs however has there been much work in furthering the AI that was introduced from some of the early computerised RPGs; however this is not the case with all game genres. Some genres have advanced their AI considerably over the years to create agents that are able to navigate complex terrain, form goals and create opinions based on their experiences with the world.

## 2.6 Adaptive Behaviour

Adaptive behaviour is the ability to adapt to changes in game environment or player’s actions at run-time in order to appropriately deal with the new situation; preferably in interesting ways that are not easily predictable [2]. These adaptive techniques are often classified as online unsupervised learning, however this does not mean that the basis for these AI’s cannot be seeded using offline supervised techniques beforehand.

### 2.6.1 Online and Offline Learning

Offline learning, in the context of developing a piece of software, is where the AI is taught during the development stage and kept static once it is released. Online learning is where the AI learns while the program is running after it has been released [3]. Online learning does not mean that offline learning cannot be applied during development time so that the AI has a good starting point for when the user first uses the software. The important factor of online learning is the AI’s ability to adapt to changes; however how this is implemented can be computationally expensive.

### 2.6.2 Supervised and Unsupervised

Supervised learning is the task of inferring a solution from a set of supervised training data and examples; once the AI has been trained it can then be tested on real world data. Unsupervised learning takes place on real world data [3].

Both the online/offline and supervised/unsupervised properties have an important impact on the learning algorithm that is used and how it is used. As in an offline supervised learning environment, the algorithm can take it’s time to find an optimal solution for the current situation, however if there is no online learning part the algorithm will still not be adaptive.

## 2.7 Scripting

The most commonly used method for adding a way of giving any game a basic level of AI is to hand code exactly how the AI should behave through the use of scripts. A script can be anything from a simple set of rules to complex algorithms running in either native code or in a run-time interpreted language. Scripts generally form the backbone of all AI in directing their exact actions and retrieval of information from the game world and work using a combination of inputs from the world, the current state the script is in and a set of rules for those states [4]. Scripts allow for the programmers to give the AI very precise instructions on exactly how, when and where an action should be performed, which depending on the game can result in optimal performance. However if the game is such that an optimal performance by the computer players results in a near unbeatable opponent, then the player is not going to continue playing and then the fact that the AI is perfect is moot. This has the result that the programmers have to artificially limit the abilities of the AI in order to level the playing field to such a level that the AI only keeps track of a certain number of units at a time, can only perform a certain number of actions within a second or how fast they react to events in the world, such as spotting an enemy. However in their simplest form have no memory, no way of adapting over time and are usually quite predictable by the player; however unless the scripts themselves are extremely complex, they are for the most part very computationally efficient.

Scripting has been used to great effect in many games such as BioWare’s RPG Neverwinter Nights (NWN). NWN has proven itself a very competent RPG based on D&D’s 3.5 edition rules and is a good example of scripting, as it provides a feature rich scripting engine that can be used for creating events in the game and for controlling the AI. This is an example of a scripted AI that is complex enough to deal with many different situations and to provide an opponent that will in many cases use its own abilities to good effect. NWN’s AI still has the inherent flaw of any scripted AI that it is however predictable and will not change tactics if what it is currently trying is not working well and that particular creature may actually have the abilities that would be more affective that in given situation.

## 2.8 Dynamic Scripting

The next logical step with scripting is to be able to change what script is being executed at run-time based on the AI performance; this is possible through the use of a technique called dynamic scripting [2]. Dynamic scripting provides a way for the AI to learn and adapt to different situations based on its performance, this is achieved through keeping a database of different available rules and each AI agent keeps its own set of weights associated for each rules. These weights are then updated at run-time based on the AI’s performance. At the start of an encounter, these weights can either be all equal or they can be assigned a random value. Depending on the implementation of the system the way in which the weights are used to create the script that is actually used will change, however the simplest way is to simply sort the weights from highest to lowest, although this can lead to issues where if the situation changes quickly the rules used will remain the same for several passes until the lower weight rules overtake the previously highly weighted rules. So another way to use the weights is to have them based on a probability, the weights with the higher values are then executed more often, still leaving the opportunity for the AI to discover a rule that would perform even better for the current situation if its current script is not working as well as it could.

The performance of a script is determined by its fitness function which calculates how well the script performed during the last encounter or other important time frame; this is then used to adjust the weights of the rules that were used to make up the final script. The total amount of weight is always maintained, and each script has a *Wmax* and *Wmin* associated with them which determine the maximum and minimum values that the weight can have. When a rule is rewarded after it performed well in the last encounter, the other rules need to have their weights reduced by the same amount that keeps the total weight in balance. The same is also true when a rule is punished when it did not perform well, and its weight is reduced, the other rules need to have their weights increased. The reason behind increasing the weights of the rules that were not used is that, since the scripts there were used did not perform well, perhaps these other ones will [5]. The reward and punished values also have a maximum and minimum.

The weight adjustment formula in its general form:

Where *Pmax* and *Rmax* are the maximum the weights can be adjusted due to a punishment or reward, and where *b* is a break-even value where the weights will remain unchanged if *F* the fitness value equals *b*.

In a given encounter after the script has been created by the system, the overall performance is can be determined. If the script performed well, all the rules that were used have their weights increased and the remaining weights are decreased by the same amount. Again the reasoning is that since the script performed well without those other scripts, chances are those rules are not important [5].

Also presented in [2] is an example of adding dynamic scripting to NWN, where they have taken the static scripts provided with the game and broken them down into a rulebase with some variation to give each script a certain use condition. They present a mock combat situation of two teams of four combatants, a rouge, wizard, cleric and fight; one team is controlled by the original scripts and the other by their dynamical scripts. While at first the static team proved successful, the dynamic team quickly adapted from their randomly weighted scripts into a set of scripts that proved to be much more effective. This was done to demonstrate that should a human player always play using the same tactics that the dynamic scripts would adapt, putting them at a disadvantage.

Another technique called Reinforcement Learning (RL) behaves in a very similar way to dynamic scripts, in that it also has a set of actions that the AI can perform and each action has a weight assigned to it and is altered based on the perceived performance of using a given action. RL differs mostly in how the actions are evaluated and how the set of possible actions that the AI will then perform next is determined.

## 2.9 Finite State Machine

Finite state machines (FSM) when in the context of a game AI are a way of separating different groups of actions based on what state the AI is currently in. The finite state machine as a whole is made up of two important elements, the states and the transition. These different states often reflect simple high level actions such as ‘Idle’ or ‘In combat’, these different states often define what script is currently in use. The transitions define the conditions it takes to go from one state to another; this can often be as simple as ‘See enemy’ takes the AI from the ‘Idle’ state to the ‘In combat’ state.

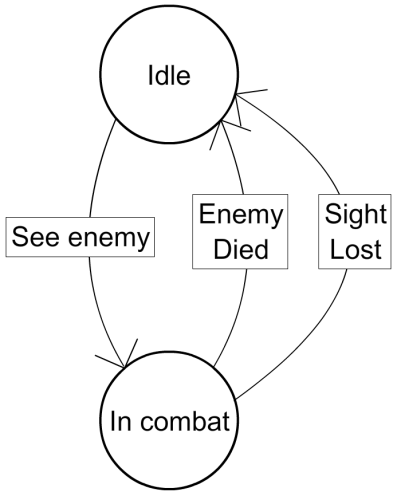


Figure 1: A simple FSM that shows the transitions between two states.

FSMs have been heavily used in games as a simple way to determine which script to run, while not adaptive, more complex FSMs can be designed in such a way that they can function with a basic memory and as such the second time the same action occurs the state can change which script is being used. Because of this, developers have used FSMs because they are simple to use, debug and implement.

One example of a game that has used FSMs is id’s Quake game series [6]. They show how the game AI can be represented by a series of different states, all of which are related to combat.

## 2.10 Neural Networks

Neural networks (NN) attempt to simulate the biological process that occurs with large networks of neurons found in a brain [7]. Each of these artificial neurons contains a weighted factor for each input and a condition for firing the neuron’s output and a weight for the value of the neuron’s output. These neurons are often grouped into layers, with one set for accepting the input and collecting the output and a varying number of hidden layers in between. All output from the neurons from one layer are connected to all other neurons in the next layer. These, when combined into a large network can be trained to match input data to training data, and to adjust their weights based on amount of error found between the expected output and the network’s output. There are several different learning algorithms that are used when training a neural network; the simplest and most commonly used is back propagation [8].

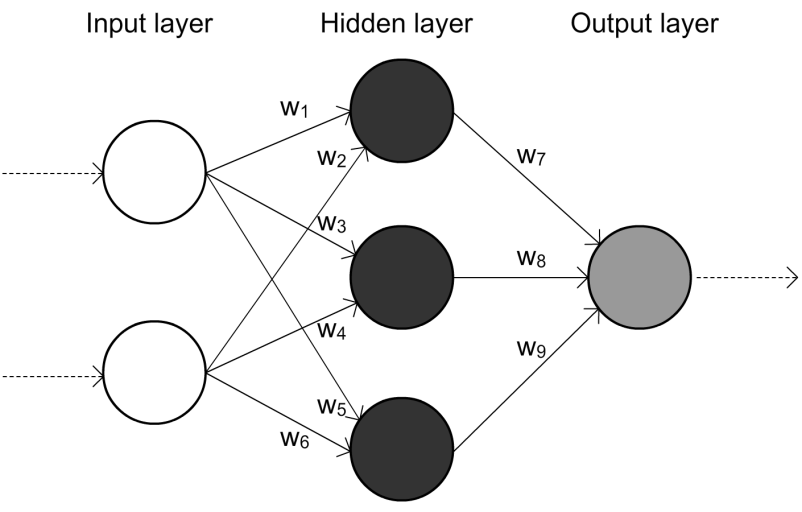


Figure 2: Simple neural network with one hidden layer. Showing how each connection has an associated weight.

While NNs are not often used in games for online learning, it has been successfully used in offline supervised learning. Such as Colin McRae Rally 2.0, the second game in a long series of realistic rally car driving simulators. These games focus on creating a realistic environment for which the cars are to be driven in where everything about the cars and the track is simulated. This presented a tough challenge for the developers of the AI to create something that could navigate these tracks proficiently and as such they turned to using a NN that was trained to drive and handle the roads and then used higher level rules for additional driving skills such as overtaking and recovering from a crash [9].

NNs offer a good way of adapting to a situation and learning to deal with complex input and output. However neural networks in the context as working similar to a gene pool where different genes are going to have a varying importance on whether or not a creature survives in a given environment; additionally in the context of a gene pool, neural networks are going to have less variation between different individuals resulting in poor environmental adaptation should something major occur.

## 2.11 Genetic Algorithms

Genetic algorithms (GA) provide another way to replicate biological processes to search for a solution to a problem. This is achieved by using a population (aka gene pool) of potential solutions, each encoded into a string of data called a chromosome [10] [11]. These chromosomes can simply be an array of values used to represent different parameters for a function or more complex data. Through natural selection where the selection is based off a fitness function that determines the genetic fitness of a chromosome. In the context of a game, the chromosome could be split up into an array of bytes that are used to control frequency or order of AI agent actions. The performance of these actions is used to determine the fitness of each chromosome. The algorithm then moves to the reproduction phase where based on the fitness of each chromosome, the ones with the highest fitness are chosen to be bread together to form the next generation of chromosomes; the two main operators at work during reproduction are crossover and mutation. The crossover operator at random will choose a crossover point where the data between two chromosomes are swapped. After crossover a mutation operator is applied which has a small chance to corrupt a small amount of data, this usually involves simply swapping a single bit somewhere in the new chromosome. Of course, depending on implementation, the number of crossover and mutation points can be more than one. Due to these crossover and mutation operators, the new child chromosomes have a chance to be both better in some aspects and worse in others. Due to the additional random nature of how the child chromosome come about, the algorithm can find novel solutions to the problem, as an important factor in the genetic algorithm is that it does not need to know why a chromosome is more fit, only that it is.

This random nature of the algorithm is both an advantage and disadvantage, in that one chromosome may result in a novel and interesting solution to the problem and in that same generation you may end up with a very poor performing chromosome. This chance of having a poor performing chromosome is generally what leads to game developers not using GAs as an online adaptive solution, however this also depends on how the GA is used to affect the AI.

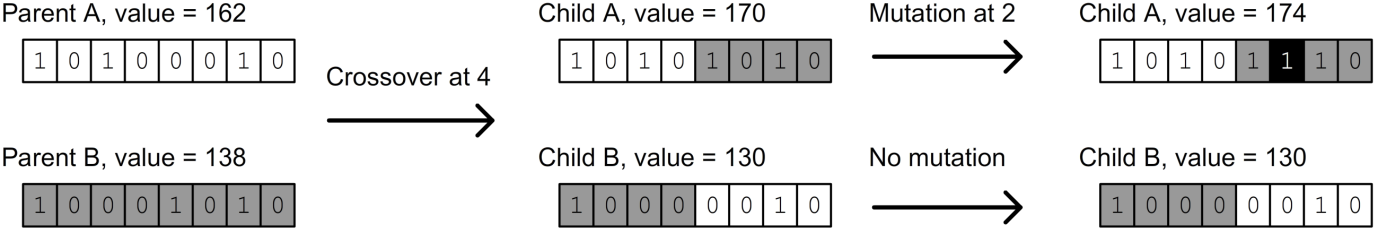


Figure 3: Shows the process of crossover and mutation between two chosen parents and how the values for the child  
 chromosomes come about. This shows a very simplistic data encoding model where the actual value of the  
 chromosome is directly based off the bit stream. This can work in many cases, however as shown the values  
 can fluctuate rather than trending towards a particular value.

A research paper into using supervised offline GAs to train an AI that could deal with tricky situations presented in an RTS that scripted rule based AI’s cannot, showed that it did not take much time for the trained AI to outperform the scripted AIs. These situations include (in the context of Wargus an open source engine of Blizzard’s Warcraft II RTS) the Footman rush, where the player will attack early on with the weakest land based unit in order to gain advantage and Knight rush, an attack comprising of the most powerful land based unit as soon as possible. These two strategies are often used by players to overwhelm enemy AI because their usual strategy is to build an economy and focus on expansion in order to be in a good position later on in the game. This exploit the flaw in the scripted AI that it is assuming how the player is going to play, and if the player uses a strategy outside of that, the AI typically does not perform very well [12]. Another problem with this exploit is that it can be hard for a developer to come up with a strategy to counter such an attack; the trained GA on the other hand was able to find a solution. This does have the unfortunate limitation that this had to be done offline and not online.

## 2.12 Rejection and roulette sampling

Rejection sampling is the process creating pseudo-random numbers using a function *f(x)* and comparing it again a number taken from uniformed distributed set *U(0, 1)*. This relies on *f(x)* having a codomain range the same as the range of *U*.

If we take the example of needing a random number from a range of *[min, max]*, first we need a function *f(x)* that plots the desired distribution over that range and a codomain over the range of *(0, 1)*. Then sample *x* from the set *[min, max]* and *u* from the set *U(0, 1)*. If *u < f(x)* then return *x*, else repeat with new values of *x* and *u*. This process has the advantage of being very flexible as the only requirements are that the codomain of *f(x)* matches the range of the set *U*, and it is very simple to implement. However as it relies on two random numbers giving values desired results and continuing to sample if they do not, the number of iterations it can take to get a number that matches the *u < f(x)* condition can vary greatly depending on how much of *f(x)* is a number that satisfies the condition.

Example *f(x)* function for use in rejection sampling process.

Where

Roulette sampling is the process where a pseudo-random number is taken from a distribution based on an array of probabilities. This lends itself well with rejection sampling where *f(x)* returns the value of the array at *x* which is its probability.

|  |  |
| --- | --- |
| Example array of probabilities.  Prob[0] = 2.0  Prob[1] = 1.1  Prob[2] = 1.1  Prob[3] = 3.5  Prob[4] = 2.7  Prob[5] = 4.3  Prob[6] = 5.1  Prob[7] = 1.5  Prob[8] = 1.8  Prob[9] = 4.0 |  |

## 2.13 Background on computer games RPG

Within the worlds of many open world RPGs there usually exists many varying populations of creatures. And while these worlds can be expansive and the variety in the types of creatures you encounter may be great, more often than not the AI that controls these creatures is either the same or extremely similar. Another feature of these open world RPGs is that the player can affect the world’s environment in some way; these can either be part of the usual story progression or through the player’s individual actions. However often in these cases the creatures of the world are unaffected by these changes unless they specifically made to react to these changes by the game designers. This lack of ability to react in any reasonable way gives the creatures a generally mechanical feel and makes them very predictable. Often the most complex behaviour is nothing more than the ability to attack on sight and to flee to a safe distance when injured. Part of this limited behaviour is the result of game developers and designers being unwilling to invest in unproven methods of gameplay and with the rise in many multiplayer focuses games. As such many of the solutions presented in this thesis are only targeted towards games which fit the right requirements of being an open world RPG or simulation and that attracts the kind of game players who are interested in the game’s AI.

Some of these types of games do exist, both from big game making companies and smaller indie groups. One of the biggest and most widely known of the big titles is Bethesda’s The Elder Scrolls, where many of the games titles in game worlds range in size from 25.9 to 41.4 square kilometres of detailed landscape, and another title which used terrain generation to create a landscape 487,000 square kilometres in area. The landscapes are filled with various points of interested and depending on the area the player will find different varieties of creatures. However all of these different creatures unfortunately all have the same behavior with the only differences generally relating to how it moves (either be walk or flying).

Some of the indie titles that have also been very successful in this category have been Dwarf Fortress from Bay 12 and Minecraft from Mojang. Both of these games procedurally generate their vast worlds, in the case of Minecraft it will generate new terrain as the player explores new areas, giving a Minecraft world only practical limits of file size and limitations from using 32 bit integers and floating point errors. Because of this the world size is capped at 3,600,000,000 square kilometres which is still approximately 7 times the surface area of Earth. Dwarf Fortress worlds on the other hand have a fixed world size (for that world) with the largest being approximately 25,900 square kilometres. Both of these very popular games along with the titles made by big companies show that there is an interest in games which simulate very large worlds for the players to explore. In both of these cases however the AI is still quite limited, with Minecraft using fairly primitive AI that allows creatures to follow another creature, attack or flee. Dwarf Fortress however does boast some relatively complex AI that is used for the Dwarves, that makes use of many different parameters to represent a Dwarfs mood and keeps a list of what the Dwarf recently observed or did. However there is no breeding of traits to pass down to the next generation, nor is there any of this more complex AI found in the general creatures found in the wild.

All of these examples show that while these games can create vast worlds for the players to explore, very little of what the player does to the environment has much effect on the creatures that inhabit that world.

Some of the reasons as to why more complex AI is not used can be put down to how creatures are generally dealt with in games. One major factor is that there is no concept of the individual creature. All of them are simply an instance of a base template for that creature. This paradigm for dealing with creatures has the benefit of reducing memory and CPU requirements as less data needs to be stored about the individual creatures themselves and creatures that are not within range of the player do not need to be simulated. However this solution does not allow for more complex behaviours to emerge as there is no way for individual growth of each creature.

## 2.14 Individuality

The concept of the unique individual in many games is not one that is explored frequently, often due to technical and time limitations of having to create potentially hundreds of unique individual creatures and the potential memory issues that would have to be taken into account. However there are games that do attempt to have unique individuals and these individuals are usually created procedurally, moving the work from the content creators to the algorithms to come up with unique combinations. One such game is Tale Worlds Mount & Blade, a medieval first and third person combat simulator where the player takes control of an army filled with different individuals. Each individual is randomly generated through the games own character creator and any changes made to that individual is persistent; for example if a knight loses their horse they are given a new horse, which can be obvious at times when they are given a pony. While these differences between each individual is purely cosmetic and does not affect gameplay, it gives a level of detail and immersion that would not be possible if the player were in control of one hundred or so identical units.

# Chapter 3: Problem statement

From the previous chapter we can see that there have been advancements in AI but only in certain areas, and that there are games that are built around the premise of being set in a large open ‘living’ world. However these large open ‘living’ worlds are in-fact very static and that change only occurs when it is specifically told to do so. And that on the surface a world that appears to feature many places to explore filled with varying creatures is filled with creatures that may all look different but they all act and behave the same. Never would the player find a pack of wolves where each wolf may react differently to your presence, where the player could find a particularly timid lone wolf that would run from you instead of simply fighting you to the death. Where the players actions such as killing all the wolves that attack them does not result in a local genetic change where only the timid wolves end up surviving resulting in a pack of timid wolves, simply as a consequence of the players actions.

# Chapter 4: Description of System

In order to start creating a model that allows for animals to live, first a game engine and game world had to be created. This game engine was built from the ground up in C++ making use of PDCurses, a cross platform curses library for the graphics and the Boost libraries for the regular expressions, case insensitive string comparisons and string to number lexical casting functions. The world is a 2D grid with each tile having its own properties relating to its transparency, pass-ability (does it obstruct movement), its food value and how fast food regrows on that tile. Each tile is considered to be 1 meter squared in area but there is no restriction on the number of animals that can be on one tile at a time. Every animal has the same set of properties and can be considered unique unless two share the exact same values for each property.

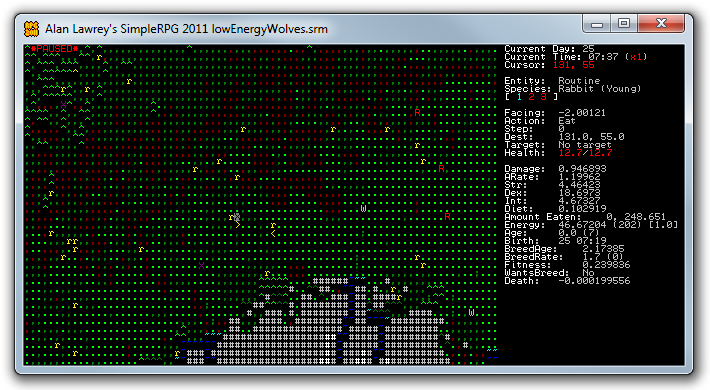


Figure 4 Example of a lush forest edge at the base of a mountain; contains lots of food for both herbivores and carnivores.

The game screen is split up into two sections, on the left is the game world screen and on the right is the menu screen. In the screen shot above the menu is currently showing a select set of properties of the animals under the cursor. The menu also provides functionality to move animals about and to change various in game options such as the time scale (x1) to speed up testing of gameplay.

## 4.1 Description of Animals

Each animal is made up of numerous properties that control what it wants to eat, when it wants to eat, how it deals with threats, its ability to deal with threats, when it wants to breed and what its current breeding fitness is. As well as more standard properties found in an RPG such as strength, dexterity, intelligence, combat related properties and health. Many properties are used as a base value and the actual value used during the game are augmented by other properties. Such as the running and walking speeds which both have a base value that is then augmented by the animals dexterity.

In this game world one of the most important factors is the food and each animal has a base requirement amount of kilo Calories (kCals) they need per day in order to survive. This base requirement of kCals does change based on the actions that an animal performs, walking uses up more than sitting still, running and attacking both require much more and giving birth requires much more again. Because of this if an animal wants to breed, first it must find enough food in order to survive giving birth and as such will eat much more than they normally require. Herbivores have an advantage in most situations as plant food is found in many places, however each food tile does not offer much in the way of kCal and regrows quite slowly as oppose to eating a creature which will give much more food.

## 4.2 Description on Environments

|  |  |
| --- | --- |
| **Graphic** | **Tile description** |
|  | Short, medium and long grass. Provides various food values for herbivores. |
|  | Small bush, pine tree and palm tree, all obstruct path. |
|  | Dirt provides no food value. Stone and lighter stone both obstruct path. |
|  | Shallow, medium and deep water. Medium and deep water obstruct path. |
|  | Sand with nothing, sand with small cactus and large cactus provide food. |
|  | Yellow rabbit and its facing arrows. |
|  | Red rabbit and its facing arrows. |
|  | White wolf and its facing arrows. |
|  | A purple X indicates a dead animal. An inverted X shows where the cursor is. |

## Asdasdas

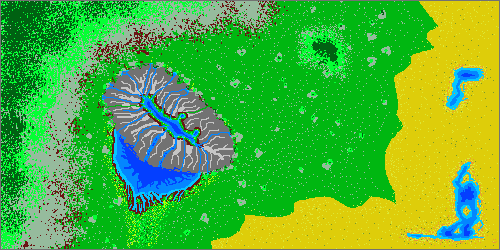


Figure 5: The map used for all the tests. Each colour is used to represent a different kind of tile.

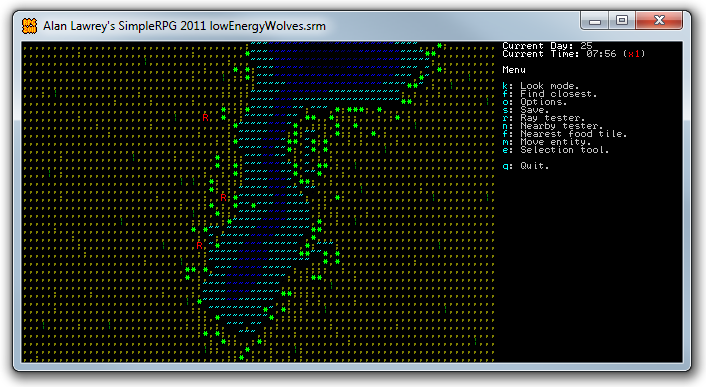


Figure 6: Example of a different harsher environment that does not have much food except around the oasis.

## 4.3 A model that simulates simple life

The first part of this system is to simulate the basic requirements for life; this includes requiring energy to live, dying of old age and being able to pass on genetic data to the next generation. Additional requirements such as shelter, water and sleep can also be simulated as to add further detail; however these were not taken into consideration due to time constrictions. These requirements give each individual creature a reason to be simulated and the removal of a need from the environment such as food results in either adaptation or death.

## 4.4 A model that can be used by game content creators

Part of a model that can be used by content creators for a game, requires that the content be designed, as such the model shown makes use of properties that are either a number or a string, each that has a known effect.

## 4.5 Fleeing and knowing whom to flee from

When faced with a threat of new creature, there are three basic actions that can be taken, to ignore, to attack or to flee. All of these require knowing what threat the new creature poses. Most often in the real world this would be done through a combination of visual and olfactory stimuli, and past experience. However all of these processes are very complicated and well beyond the scope of being simulated within a game environment. To simplify the system makes use of each creatures properties and comes up with a threat index of each creature with respect to the creature perceiving the threat. This threat model is designed to make use of as few identifying properties as possible for the sake of both speed and stability. The model also has to make some assumptions about how each creature would perceive another creature as a threat, such as any creature bigger than it is likely to be a threat and that any creature that is a carnivore is also likely to be a threat. While this is generally the case, there are real world examples that contradict the herbivore assumption, such as the Hippopotamus and the Elephant, where both can be known to be rather aggressive.

## 4.6 Breeding and generic fitness function

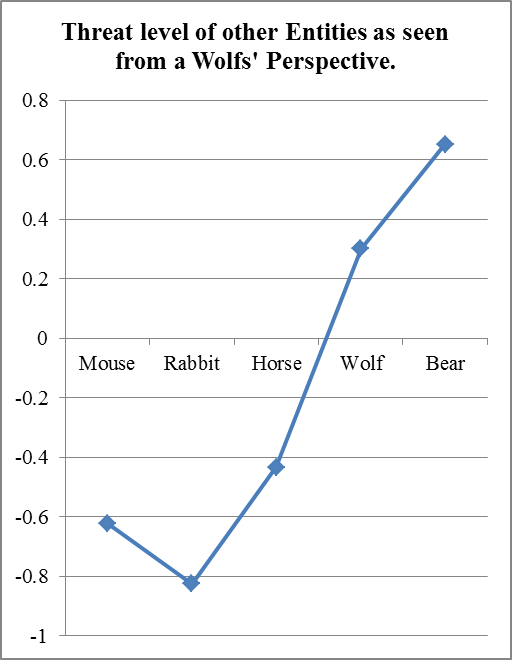
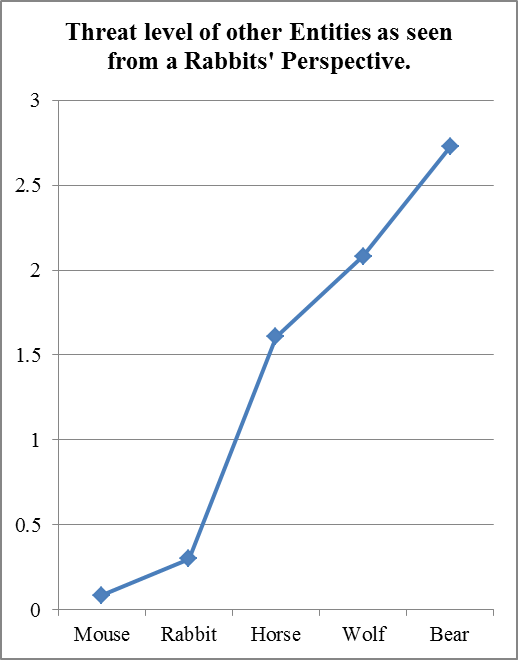


Figure 7: These two graphs show the difference perceptions in threat from both a Rabbits’ and Wolfs’ perspective

In the real world the ‘fitness’ function for choosing a mate is a complex matter and can vary greatly from creature to creature. However in nearly all cases the mate is healthy and young

## 4.7 Rejection sampling for breeding new generations

In a typical genetic algorithm solution, there are often new chromosomes that have a gene that has under gone a radical change that results in the most significant bit (MSB) being flipped, potentially changed the value 0 to the value 1,073,741,824 (2 ^ 31) within one mutation. While this may simply result in that particular gene having a very low fitness and will ‘weeded’ out in the next generation, in the situation where these genes have direct impact on gameplay, having an individual whose abilities completely differ from the others and potentially having significantly having more power than the others would negatively affect gameplay.

In order to overcome this disparity between how a typical genetic algorithm may deal with crossover, we used a system based on rejection sampling that samples over a range between the two parents genes with the

## 4.8 How the map and the creature are stored

All the data relating to the map its current state and the state of each creature and for the purpose of analysing the history of each in game day are saved in a single human readable text file. The format is loosely based on the JSON format where not all properties are required (although there are some exceptions to this rule) and the order of the properties is not important. Whitespace is also ignored unless it is between two quotation marks which denote a string. The format is split based on a regular expression that generally leaves most of the actual parsing up to the loading function.

// Multi-line comments.

"(/\\\*[^/\\\*]\*\\\*/)|"

// Single-line comments.

"(//[^\n]\*\n)|"

// Double quoted strings.

"(\"[^\"]\*\")|"

// Single quoted strings.

"('[^']\*')|"

// A single | character

"(\\|?)"

// Everything that's not whitespace.

"(\\S+)"

Apart from the single ' and double quote " marks along with the // and /\* \*/ tags, only the vertical bar | character is considered to be special.

These next two sections show how each area is divided up. After each heading denoted by a -- each section has its own parsing logic that deals with the proper loading. Here is an example (and simplified) map showing the main features and what a typical map may look like.

This page has been split into two columns for the sake of space.

-- Options

hud\_width 30

current\_time 300

current\_day 0

day\_length 600

-- Tiles

, 2 // Deep water

. 0 // Grass

; 5 // Short grass

^ 1 // Sand

~ 6 // Stone

\* 7 // Thick Trees

R 8 // Trees

r 9 // Dirt

T 10 // Lime Stone

d 11 // Shallow water

w 12 // Water

-- Map

RRRRRRRRRRRRRRRRRRRRRRRRRR

RdRddd.........^^TTTT^^,,R

RdRdrd^^.^...^^TTT^^,,,,,R

RdRdr^...^^^TTT^^^,,,,,^^R

RdRdr^.^^....^.^^,,,,,^^TR

Rdddr.....^^...^,,,,,^^TTR

Rrrrr..ww~~~~^.,,,,,,^^TTR

Rdddr.w~\*\*\*~~w.,,,,,,,^^TR

Rdwwww~\*\*\*\*\*~w..,,,,rdrrrR

Rdw~\*\*\*\*\*~~w....^,,,dddddR

R..ww~~~ww...~~TT^,;rddddR

R^.........~~~~~TT^^rddddR

RRRRRRRRRRRRRRRRRRRRRRRRRR

-- Entities

Animal

id 0

species "Wolf"

position 2.92859 3.30684

facing -3.08701

name "Fred"

current\_action

TargetAction

action Attack

completed 0

step 0

target @ 1

end

action\_history

Action

action Eat

completed 1

step 6

end

end

destination @ 3

health 10 10

strength 16

dexterity 15

intelligence 14

running\_speed 15

walking\_speed 3

turning\_speed 2

entity\_size 2.0

entity\_mass 48

diet 0.8

aggression 1.1

damage\_base 4

birthdate "0 35"

attack\_rate 1.6

end

Animal

id 1

species "Rabbit"

position 9.93701 4.02985

facing 1.60264

name "Jeff"

graphic yellow 1 r

current\_action

Action

action Idle

completed 0

step 2

end

action\_history

end

destination -1 -1

health 8.34 12.78

strength 4

dexterity 18

intelligence 5

running\_speed 15

walking\_speed 1.1

turning\_speed 2.6

entity\_size 0.3

entity\_mass 0.8

rest\_energy\_per\_day 200

hunger\_limits 0.4 0.8 1.2

life\_expectancy 7

birthdate "1 1"

breeding\_age 2

age 2

breeding\_rate 1 0

diet 0.1

damage\_base 1

attack\_rate 1.2

end

-- End // Not required but any text beyond this header is ignored.

The previous page shows a complete and working map file. The aim is to show how each animal is only made up of different properties and not all need to be defined, such as in the case of the wolf, it has not been given an age so it will start off as if it has just been born. There are also an addition number of properties not shown in the file which are generally less important such as cool down timers for different actions which will all start off at the default value of zero. This means that a new Animal entity can be added to the game with just two lines.

Animal

end

However the Animal would not have any distinct properties but it would survive (perhaps rather poorly).

# Chapter 5: Empirical Results

Results show that the various processes relating to the creatures surviving, ageing and breeding are all working, and that the different genetic make-ups do have an impact on the creatures behaviour. The biggest success is in the fact that the difference between a wolf and a rabbit is only in their genetic make-up.

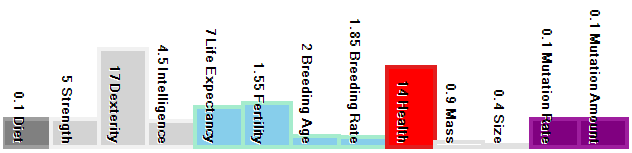


Figure 8: Averaged genetic make-up of some rabbits.

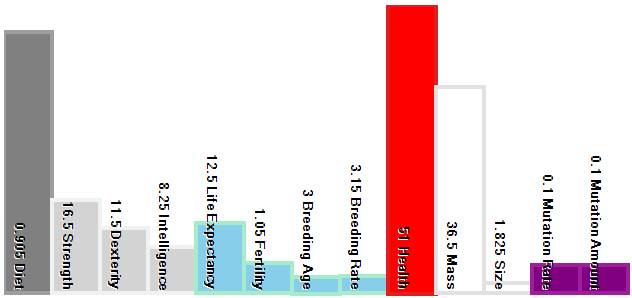


Figure 9: Averaged genetic make-up of some wolves.

# Chapter 6: Further Research

## 6.1 Balancing the eco-system

At the time of completion for this thesis, the eco-system was entirely self-regulated, requiring that each creature doesn’t overeat and that as a whole the populations need to stay near the systems carrying capacity. However due to limitations of the AI and the scale of the system it is currently quite difficult to get the system stabilised around the carrying capacity. In the context of a game it may be more practical to for the game to directly manage the population and reject requests to breed if the system is unstable (such as too many predators or not enough food tiles), this would use a model similar to the Predator-Prey model presented independently by Alfred James Lotka [15] and Vito Volterra [16]. This Lotka-Volterra model could be used to manipulate the birth and death rates to keep the system in balance, and depending on the scale of the world, could be done on a per population basis or on a global basis. However too much reliance of this could result in restrictions of the evolution as the general Lotka-Volterra model uses constant values for the birth and mortality rates and while these rates can change, some results can still be very unstable.

Figure 10: Example of a stable Lotka-Volterra algorithm using constant birth and mortality rates.

A more interesting approach would to be further extending the self-regulating aspect beyond the limited and already artificial approach that is currently taken.

## 6.2 Potential of using a neural network to control how the creature behaves

Currently the creatures are controlled using a simple state machine based on what action and what step in that action that they are current executing, this allows for a single action to be reasonably complex and to be able to move between steps as the situation changes. However without the use of the creatures parameters to change how the action executes the actions would be complete predictable in all cases.

The neural network could take all the creatures’ genes and some filtered data about the surrounding entities.

## 6.3 More in depth simulation of the creatures needs

To fit within the scope of the thesis’s time constraints, additional needs such as sleep and water requirements have not been simulated. These however are important aspects to all living beings and in general have a major impact on where creatures would be found and how they interact with the land.

More complex social interactions have also not been simulated. This could include methods of social creatures flocking and generally staying close to each other. And while each creature has an alignment for how it sees other species, which is currently only either neutral or if they’ve been close to a friend whose been attacked, negative. This does not extend to how they feel about other individuals of any species. This could open the possibilities to more complex interaction between individuals of the same species where two opposing forces where the need to stay together for protection and mating, is negated by potentially negative actions of each individual, such as stealing food or a mate.

## 6.4 Simulating a scent to choose mating partners, being alerted to enemies and finding food

To simplify certain aspects of this thesis, segmentation between different species came down to a using a string to represent the species name. While this simplified numerous aspects of the program, it did also limit the creatures’ ability to evolve and the potential to have new species arise. The scent would be based off a combination of the creatures’ parameters that represent its’ genes. This combined with a method of comparing similar scents would allow for a more natural way of speciation to arise. This could also be used to weed out any genetic extremes that may arise, as currently any creature that is alive and is of the same species has a chance to reproduce.

**Acknowledgements**

I would like to acknowledge my supervisor Dr. Andrew Skabar who helped me numerous aspects of this thesis. Such as the rejection sampling for the use of breeding individual properties for the animals and for keeping me focused when I got distracted with less important elements of the program.

The Boost C++ Library, which was used for the regular expression engine, case insensitive string comparisons and the string to floating point number lexical casting.

# References

|  |  |
| --- | --- |
| [1] | J. Orkin, "Applying Goal-Oriented Action Planning to Games," in *AI Game Programming Wisdom 2*, S. Rabin, Ed., Hingham, Massachusetts, Charles River Media, 2004, pp. 217-227. |
| [2] | J. Orkin, "Goal-Orient Action Planning," 30 03 2011. [Online]. Available: http://web.media.mit.edu/~jorkin/goap.html. |
| [3] | M. Cook, Dungeon & Dragons: Dungeon Master's Guide, 1st ed., Wizards of the Coast, 2003. |
| [4] | P. Spronck, "Dynamic Scripting," in *AI Game Programming Wisdom 3*, S. Rabin, Ed., Boston, Massachusetts, Charles River Media, 2006, pp. 661-675. |
| [5] | E. Alpaydin, Introduction to Machine Learning, 2nd ed., The MIT Press, 2004. |
| [6] | R. Schank and R. P. Abelson, Scripts, Plans, Goals, and Understanding, Psychology Press, 1977. |
| [7] | M. Ladebeck, *Apply Dynamic Scripting to "Jagged Alliance 2",* Technische Universität Darmstadt , 2008. |
| [8] | id. [Online]. Available: ftp://ftp.idsoftware.com/idstuff/source/q1source.zip. |
| [9] | J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *PNAS,* vol. 79, no. 8, pp. 2554-2558, 1 April 1982. |
| [10] | J. A. E. Bryson and Y.-C. Ho, Applied Optimal Control: Optimization, Estimation and Control, Revised ed., Taylor & Francis, 1975. |
| [11] | 24 April 2001. [Online]. Available: http://www.generation5.org/content/2001/hannan.asp. |
| [12] | J. H. Holland, Adaptation in natural and artificial systems., 2nd ed., The MIT Press, 1992. |
| [13] | D. E. Goldberg, Genetic algorithms in search, optimization, and machine learning., 1st ed., Addison-Wesley, 1989. |
| [14] | B. Marc Ponsen, *Improving adaptive game AI with evolutionary learning,* Delft: Delft University of Technology, 2004. |
| [15] | A. J. Lotka, "Contribution to the Theory of Periodic Reactions," *J. Phys. Chem.,* vol. 14, no. 3, pp. 271-274, 1910. |
| [16] | V. Volterra, "Variazioni e fluttuazioni del numero di’individui in specie animali conviventi," *Mem. R. Accad. Naz. dei Lincei,* vol. 2, pp. 31-113, 1926. |
| [17] | I. S.-K. E. P. Pieter Spronck, *Online adaptation of game opponent AI in simulation and in practice,* Universiteit Maastricht. |
| [18] | D. S. Miller and P. R. Payne, "Weight Maintenance and Food Intake," *The Journal of Nutrition,* no. 78, pp. 255-262, 1962. |
| [19] | L. S. João Leite, *Evolving characters in role playing games,* New University of Lisbon. |
| [20] | R. Evans, "Varieties of Learning," in *AI Game Programming Wisdom*, S. Rabin, Ed., Hingham, Massachusetts, Charles River Media, 2002, pp. 567-578. |
| [21] | M. E. Bratman, Intention, Plans, and Practical Reason, Standford, California: CSLI Publications, 1999. |
| [22] | G. Bell, Selection: The Mechanism of Evolution, 1st ed., Springer, 1996. |