Using Fitness Functions and Genetic Algorithms to Enhance the Performance of NPCs

Acknowledgements

I would first like to thank others who provided help when needed and without whom the project would turned out a lot worse.

- A big thank you to Grzegorz Cielniak, the project supervisor for providing solid guidance and judgement throughout the entire process.
- The Creation Kit Wiki for compositing the scripting functions of the Creation Kit so that I didn't have to rely on guess work.
- Ian Patterson, Stephen Abel and Paul Connelly without whom the development would have been a lot more confusing and slowed down greatly.
- Bethesda for providing free mod tools for all users.
- Kieran Charles Hicks for providing his report as an example.

Digitally attached to this submission will be the modification file made as a result of the project. By opening it with the Creation Kit, all created content can be accessed.

Abstract

The video game industry typically uses finite state machines for its AI, while overlooking potentially worthwhile techniques such as genetic algorithms. This project will attempt to determine if genetic algorithm can compete with finite state machines and to see if they can be used to optimise the strength and performance of NPCs with a video game. By examining the data that results, conclusions about game balance can be drawn to hopefully result in a fairer and more fun experience for the plater, as well as saving time on the programming process. The project will attempt by using a quantitative methodology of measuring NPC performance and comparing it to those with differing equipment. The outcome of this would suggest that there is potential for both fitness functions and genetic algorithms to optimise the game experience for the player.

Contents

1. Project Background 5
1.1 Introduction 5
1.1.1 Finite State Machines 5
1.1.2 Genetic Algorithms 6
1.2 Motivations 6
2. Literature Review 7
2.1 AI in the Games Industry 7
2.2 Genetic Algorithms 8
2.3 Finite State Machines 9
2.4 Evolutionary Computing 10
2.4.1 History 12
2.4.2 Evolutionary Algorithms in Games 14
2.4.3 Case-injected Genetic algorithms and other variants 17
2.5 Intelligent AI and Human-Like Behaviour 18
2.6 Summary 18
3. Methodology 19
3.1 Development Methodology 20
3.2 Evaluation Methodology 22
3.2.1 Qualitative Evaluation 22
3.2.2 Quantitative Evaluation 23
4. Tools Appraisal 24
4.1 Engine Selection 24
4.2 Numerical Computing Software Selection 26
5. Aims and Objectives 27
5.1 Aims 28
5.2 Objectives 28
6. Project Management 29
6.1 Project Planning 29
6.2 Risk Assessment 31
6.3 Backups 32

7. Implementation 33
7.1 Outline and Requirements 33
7.2 Population Implementation 35
7.2.1 NPC Implementation 36
7.2.2 Enemy Implementation 37
7.2.3 Data Gathering 38
7.3 Coding Implementation 39
7.3.1 Scripting in Papyrus 39
7.3.2 Data Gathering Implementation 41
8. Evaluation 53
8.1 Evaluation Methodology 53
8.2 Results 54
9. Conclusion 62
9.1 Future Works 63
9.2 Critical Reflection 64
10. Bibliography 67

1. Project Background

1.1 Introduction

This introduction will briefly cover what genetic algorithms (GAs) are, why they are underused and have potential for becoming more common. The UK-based video games industry alone was valued at £1.72 billion in 2014 (Nesta, 2014) and with such huge value being present in the industry, any optimisation or improvement in the development process can result in even bigger profits, and any time saved in production is a huge reduction in costs, more so considering programmers earn as much as $124,000 in some areas of the USA (Big Fish Games, 2015). Countless video games feature non-player controlled game agents, both friendly characters and enemies. The non-player characters (NPCs) are typically designed to simulate living agents, with modern games such as *Skyrim* and *GTA V* featuring unscripted interactions between non-player characters, simulating a real, living world:

https://youtu.be/krvSnHfOOVM

Stories such as these help fuel player interest in a game, resulting in changing emergent gameplay that can keep players interested even years after a game comes out, with users being able to have completely unique experiences:
http://us.battle.net/wow/en/forum/topic/3870837624

With AI resulting in such emergent, interesting gameplay, games that feature excellent AI, such as Alien: Isolation, are critically acclaimed and respected for the interesting scenarios and opponents that feature.

1.1.1 Finite State Machines

The industry uses finite state machines (FSMs) for their AI in the vast majority of cases. FSMs are a flawed systems, with skilled players being able to find holes in the programmed rules and exploiting them over and over again. Predictable, robot-like behaviour can also result, possibly resulting in a less believable game world and a

reduction in fun (Geisler, 2004, 54). Finite state machines are used to model and code a character's behaviour in a limited or finite number of states, with transitions between states happening as a result of input from the game (Zana, 2013). This means that they perform well in their pre-programmed situations, acting exactly as the developers desire, but undesired behaviour can result in odd situations.

1.1.2 Genetic Algorithms

However, despite the prevalence of FSMs, it is important to ask: is there a better method? Other AI systems including evolutionary computing are the subject of many different research projects. Indeed, evolutionary algorithms in games are not new: researchers such as Cook, Cole and Miles have experimented with replacing the default AI with genetic based intelligence. A genetic algorithm comprises of a population, in this case it will be a population of game agents, with a set of biologically inspired operators defined over the population. As in biology, the most suited element in the population will survive and pass on its information. There are currently only three commercial games that have feautures evolutionary computing, and there currently exists no game that evolves novel content as the game is played (Guha et al, 2009). Genetic programming works by running a program, then comparing its behaviour to an ideal. Usually, a fitness value is calculated by collecting data on several metrics that and those that do well are chosen to breed and produce new agents for the next generation, then repeating this process until a close to ideal solution is reached (Langdon et al, 2008, 1-2).

Genetic algorithms are becoming a more popular topic for research, but most researchers focus on the use of GAs in other industries, or on evolving map generation, or RTS AI. Little research as been done into use of GAs in first-person games, but with Louis and Miles proving that GAs can compete with FSMs and save time, they hold potential value in the games industry.

1.2 Motivation

Why then, are genetic algorithms, a subsidiary of evolutionary computing, a good choice for use in future video games? With video game worlds and interactions becoming more and more complex, it is becoming exponentially harder to code finite state machines, which rely on the programmer inputting every state and interaction possible, which takes an enormous amount of time (Cole et al, 2004, 139). As previously said, with the huge amount of employees in a large video games development company, and the salaries being almost $125,000, any time saved in the programming of AI can result in huge monetary savings. It is regarded that advancements in AI are lagging behind other aspects of game such as world-building and graphics (Miikkulainen, 2007), and this is partly due to the same method of AI being used in almost every game for the last 30 years - finite state machines. If genetic algorithms work as an alternative to finite state machines, not only could there be a saving in time and money, but more realistic game worlds and human-like NPCs and enemies could result. Genetic algorithms can work in almost any situation as they will work towards the best solution without human input, which makes them well suited to automated design and complicated tasks with multiple solutions. They are rarely used, partly because they are expensive in terms of computational power and can often take longer than other systems to reach a solution.Within games, genetic algorithms have primarily been used on NPCs and AI, such as Black and White. It is an underused system, but can work as well as the tried and tested methods, so it warrants further research. GAs are mostly used to create changing systems within games, usually NPC and enemy behaviours. Because this is a relatively unexplored topic, a lot can be discovered within the scope of this project, and any conclusions that are drawn will hopefully be of value to the field of AI and games.

2. Literature Review

2.1 AI in the Games Industry

The video game industry has made a very clear push within recent years towards virtual environments that accurately mimic real-world settings, with a large focus on believability of game elements such as physics or graphics. Despite this, the industry still relies on finite

state machines for the computer-controlled characters within these environments, also known as non-player characters (NPCs). As a result, NPCs lack the same degree of believability as their surroundings due to the robotic behaviour that FSMs invoke. NPCs normally allow the player to interact them and simulate decision-making and intelligence. In 2013, the global video game industry was valued at $93 billion (Gartner, 2013). Video games are the perfect platform for testing new AI techniques such as genetic algorithms, partly due to the lack of recent development of AI in games, but also because of the millions of players acting as testers, and also because of the minimal risk to human life video games have, as well as the benefits to the video game industry that can result: increased longevity of games, with emergent, changing gameplay, and decreased production costs. There are also benefits to having learning agents outside of the video game industry - such as potential improvements in education and training with computers. Stanley, Bryant and Miikkulainen suggested machine learning based upon natural evolutionary methods were the way forward and worked on allowing game agents to learn and evolve in real time (Bryant et al, 2005, 653).

2.2 Genetic Algorithms

The prime question that should be answered: what is a genetic algorithm, and broadly, what is evolutionary computing? The idea of evolutionary computing is to is a technique that allows a computer to automatically solve a problem without the user first knowing the answer or form of the solution in advance. Additionally, evolutionary algorithms are designed to emulate aspects of the natural world, specifically things like evolution and breeding. Genetic programming works by running a program, then comparing its behaviour to an ideal. Usually, a fitness value is calculated by collecting data on several metrics that and those that do well are chosen to breed and produce new agents for the next generation. Further, some genetic operators can be used, such as crossover - the combination of random elements of two “parents” to create a child agent, and mutation - the creation of a new agent by random elements, optionally from a parent (Langdon et al, 2008, 1-2). A genetic algorithm can be described as something that "emulates biological evolutionary theories to solve optimisation problems" (Alippi et al, 1994).Genetic algorithms have several uses within computing, primarily within automated design and

learning computer intelligence systems (Aguirre et al, 1997). They can work in almost any situation as they will work towards the best solution without human input, which makes them well suited to automated design and complicated tasks with multiple solutions. A genetic algorithm usually follows the following format:

1. Creation of a "population".
2. Evaluation of each individual.
3. Selection of best individuals.
4. Genetic manipulation to create new population from best individuals.

2.3 Finite State Machines

Now that the topic of what a evolutionary algorithm is, their main competitors in the game industry shall be covered to give a broader understanding of AI techniques within the industry. The artificial intelligence systems of video games have historically relied on finite state machines to provide a decision-making mechanism for the computer-controlled characters. These finite state machines are implemented in virtually all computer games that have been developed and released (Miles and Tashakkori, 2010, 32-33).
Part of the reason finite state machine (FSMs) are used is that they are very simple to make and implement, and they can be adapted to work in many ways. For example, Musse and Thalmann used a hierarchical FSM architecture to model the behaviour of crowds of characters, while Devillers and Donikian developed a scenario language that used FSMs to specify most of the instructions (Rudomın et al, 2005, 742). Finite state machines are used to model and code a character's behaviour in a limited or finite number of states, with transitions between states happening according to input from the game. FSMs are very versatile and can be either simple or complex, they are easy to understand and code, and they help to make an algorithm behave in a deterministic way. To provide an example of a FSM from the iPhone game Unstoppable Jake:

"Each instance of the Terror Drone in the level, holds his own state variables, when the TerrorDrone is constructed, the mState variable, which keeps track of its current state is initialized to TerrorDroneState.MOVE, and the update function is called in an endless loop,

performing different things according to the current state.

```
function TerrorDrone:Update()

-- Update current time
    self.mCurMS = gw.getCurrentMs()

    -- get self current position
    self.x, self.y = go.getPosition(self.ptr)
    self.px, self.py = gw.getPlayerPosition()

    ------------------
    -- State Machine --
    ------------------

    if (self.mState == TerrorDroneState.MOVE) then
        self:UpdateMOVE()

    elseif (self.mState == TerrorDroneState.TURN) then
        self:UpdateTURN()

    elseif (self.mState == TerrorDroneState.CHASE) then
        self:UpdateCHASE()

    elseif (self.mState == TerrorDroneState.EXPLODE) then
        self:UpdateEXPLODE()

    end

end

function TerrorDrone:switchState(newState)
    app.logString("TerrorDrone Changed State: " .. self.mState .. " -> " .. newState)

    self.mPrevState = self.mState
    self.mState = newState
end"
```

(Zana, 2013), (Zana, 2012)

Geisler states that the downside of using this system is the “high level of predictability of opponents, and a large amount of work manually programming each rule”. To mimic human vs human involves a high amount of tuning for game balance, where for a skilled player, the AI is never “good enough”. It needs to keep challenging the player, to adapt to the player and if possible learn from the player. Instead, a good player will learn the behaviour of the enemy AI and begin exploiting it (Geisler, 2004, 54). While this does have its place in modern games, there should also still be challenge for skilled players.

2.4 Evolutionary Computing

To ensure quality opponents we can turn to machine learning, which is concerned with improving it's performance of a task through multiple generations and experience. While this is standard to “academics”, evolutionary computation methods such as genetic algorithms have only very recently started to be considered within video games. While in the past, these methods were considered too hardware-intensive on memory and CPU to be a common option, with the current hardware advancements it is very possible. In fact, these methods have been evidenced in earlier games such as *Black and White*, and *Creatures* (Geisler, 2004, 54-56). Cole et al said that authors of bots spend an enormous amount of time setting parameters, where more realistic bots results in exponentially more complicated and time-consuming parameter coding, with bots defined as computer-controlled rule-based expert robots. Lack of game balance can result from this as it is up to a programmer who may not be well versed in the game in question to accurately create the intended game balance (Cole et al, 2004, 139). Finite state machines therefore leave room for improvement, especially in the area of making AI behave unpredictably and “human-like”. Rather than spend time working on improving their AI, many game developers simply choose to make opponents “harder” by inflating their values, such as health - enemies are made more challenging, not with improved intelligence but with bigger guns (Laird and VanLent, 2001). As a result of this, many big games spend very little time innovating or improving on standard AI practices - it takes time, and with games now

costing as much $100 million or more to make (Huerta, 2014), innovation is a risk many developers do not want to take (Cook, 2015), especially in iterative franchises where AI assets can be re-used, such as Call of Duty. To further prove this point, Rock Paper Shotgun wrote on article about the future and past AI in games, saying:

“Players correlated intelligence with challenge – in other words, smarter enemies should be harder to defeat. Bungie noticed that if you simply doubled the health of an enemy, playtesters would report that they seemed more intelligent. The takeaway was a realisation that what the game was doing simply didn’t matter – what mattered was what the player *thought* the game was doing.” (Cook, 2015). While this works to make the game more challenging and the AI seem tougher, enemies are still open to having their weaknesses exploited and the same holes in their defence taken advantage of multiple times. By changing what the AI use and and composition of a population of enemies to be better suited to the task at hand, the game changes based on the player's actions, and the AI appear to learn - which, as said, is just as important as having “smarter” AI.

Cook goes on to say that the “graphics race” push for bigger and better game worlds is slowing down and about to run its course, and with AI becoming a strong point of comparison for modern games, with games like Alien: Isolation and No Man's Sky hinting at AI being applied at larger scales and more robustly than ever before (Cook, 2015). Miikkulainen said the current AI model does not work well in video games, due to the the large amount of agents running at once and also due to the large, changing environments that run in real-time. He goes on to say that computational intelligence, primarily evolutionary computation, is well suited for use within video games, making use of adapting, embedded intelligent agents, ultimately making progress towards intelligent machines. In the long run, it will lead to better games, with both reduced production costs, and less bugs. It will also allow for training games, for games to get better with the player, and always provide a fair challenge (Miikkulainen, 2007).

2.4.1 History

The idea of mimicking the mechanics of biological evolution to develop powerful

algorithms as opposed to other methods such as finite state machines has been around for more than 30 years, since innovate researchers in the US and Europe independently came up with the idea (Back and Schwefel, 1996, 20). Evolutionary programming was originally introduced as an attempt to create artificial intelligence, and to evolve finite state machines to predict events on the basis of former observations. In addition, they were designed with the goal of solving difficult discrete and continuous, mainly experimental parameter observational problems (Back et al, 1997, 3-4). This is well suited to the problem at hand, assessing the performance of NPCs based upon a set of parameters and then make changes based upon former events, and to investigate whether AI using evolutionary programming algorithms are more effective than other methods, such as FSMs, and whether they are appropriate for use in a modern games industry. Evolutionary algorithms mimic the process of natural evolution, which in itself is the combination of new genetic information and its evaluation and selection. The better an individual performs, the greater the chance of distributing genetic information and over time and generations, this leads to a population of individuals with above average fitness. At least three primary evolutionary algorithms have been distinguished : evolution strategies, genetic algorithms and evolutionary programming. From these three, countless variants exist, but the main differences lie in the representation of individuals, design of variation operators (mutation, crossover, recombination - more are still being researched) and the selection and reproduction mechanism (Back et al, 1997, 4-6). The search space is defined on a case by case basis depending on the needs and what is being observed, and observational metrics can be defined as a result of the search space, based upon the end goal or purpose of the algorithm, which represents genotypes and phenotypes. Evolutionary computing is adept at optimisation - the art of improving upon an idea. It consists of trying variations of an initial concept and using the information gained to improve on the idea. A computer excels at this as long as the idea can be numerically input and there can also be an output. A genetic algorithm is therefore using the computer for optimisation as opposed to using human-made values. This will save time and effort on inputting and fine-tuning values as would be done in a FSM system, and will also automatically reach an end goal, as opposed to a human-made end which may be sub-par. What, however, makes a good or even “best” solution? This implies that there is more than one solution, and that some are better than

others. The definition of these things is based upon the problem, the search space, and the method of solution. Some problems do have exact answers, while others only have minimum and maximum solutions known as optimal points (Haupt and Haupt, 2004, 1). In addition, an original solution the developer had not thought of may result (Miikkulainen, 2007). Genetic algorithms excel at this type of problem-solving: where there an ideal that can be iterated through based on initial data to try and get closer and closer to the answer, or optimal point. As such, a genetic algorithm is ideal for the current project - the goal to assess the performance of an actor in a population to reach an ideal point of high fitness.

According to Guha, Hastings and Stanley, machine learning, which genetic algorithms are a subsidiary of, have thus far been used in only three commercial video games: *Colin McRae Rally 2.0, Creatures 3*, and *Black and White 2,* although the AI brains in these games is generally trained before release. While "evolving game content is an emerging research area with great potential to contribute to the mainstream gaming industry", there currently exists no game that evolves novel content as the game is played (Guha et al, 2009), despite the fact that it has been shown that machine learning has a lot of use in developing video games. It has been said that part of the reason why genetic algorithms are seldom implemented is that they take numerous trials to to learn effective behaviour and that applying adaptive AI may result in uncontrollable and unpredictable AI behaviour. However, the ability to adapt to changing circumstances is considered an important feature for simulating human-like AI behaviour (Bakkes et al, 2009).

2.4.2 Evolutionary Algorithms in Games

There have been several other researchers that have attempted to do use evolutionary algorithms within video games, such as Cole, Louis and Miles in 2004, who used genetic algorithms to test AI behaviour in *Counter-Strike* to discover if genetic algorithms produced a marked improvement on the default finite state machine-based AI bots, using the ELO system for skill in lieu of a fitness value. They discovered that after only a few generations, genetic algorithms reached the level of a human-coded AI bot that was coded by a game expert. They even theorised that such a method could be extended to other first-person games and even other genres, and that it could reduce development time, in addition

to the programmer no longer needed high level knowledge of the game in question. (Cole et al, 2004). NERO (Neuro Evolving Robotic Operatives), a game produced at the Digital Media Collaboratory involves the player training digital agents to battle each other, whereupon they will evolve in real time (Miikkulainen, 2007). They ran into the issue that if evolution occurs at each generation, the behaviour of an entire population would change instantly, which would be obvious to the player. In addition, behaviour would remain the same for long periods of time, which makes it seem like there is no evolution at all. This was fixed replacing a single individual every few game ticks. Since there were no play testers for this project, and because making congruous, smooth gameplay was not a high priority, the latter solution was not implemented. It is however, a valid concern if evolving agents were put into a polished game, but not important to this project.

It has been said before that most modern games use finite state machine or rule-based systems for
their AI (Louis and Miles, 2005) and indeed, there are next to no examples of genetic algorithms
used in published games, let alone AI-based genetic algorithms. However, according to this paper,
genetic algorithms have a huge advantage over the standard systems – there are only a finite amount
of scenarios accounted for, and this can leave "holes" in the AI which a player can exploit. Genetic
algorithms allow game agents to "learn" from past experience in order to play better, which produces an organic, unique experience when interacting with NPCs, and also stops AI weaknesses
from being exploited. As previously stated, few games use genetic algorithms despite the advantages it provides, and upon implementing it into a game, it would reveal either the problems
with this method or prove their usefulness, depending on the outcome of the work. A special case-injected variant of genetic algorithms was used here. This seemed new and so more investigations

were performed. Essentially a case-injected variant of genetic algorithms seeds the population with
effective individuals from past trials in order to reduce the time taken to find an ideal solution, as
well as ensure some variance in the population. The theory behind it is that "it makes little sense to
start a problem-solving search attempt from scratch when previous searches may have yielded
useful information about the problem domain" and the result of this is that "Injecting cases with
higher fitness tends to lead to a quicker flattening out of the performance curve, of average or
maximum fitness versus time" (Louis and McDonnell, 2004).

Guha, Hastings and Stanley created evolutionary content generation for the video game Galactic Arms Race, where content is given a fitness value based on how much it is used by players after it is spawned into the game world. As with this project, they used a spawning pool that content is randomly selected from by random probability, ensuring diversity and that previous high fitness solutions might re-appear. The unique aspect of Galactic Arms Race is that weapons evolve based upon and randomly appear based on whether they were found to be favoured by players in the past – the individual weapons themselves in theory are not good or bad, as strength and speed does not vary. Rather, the pattern of particles fired changes, resulting in a complex multi-objective co-evolutionary landscape (Guha et al, 2009). This is similar to what has been undertaken in this project, the key difference being that part of the goal for this project is to find the best combination of different equipment, and also to find what equipment is best, though they were intended to be equal when created by the developers. The conclusion to Guha's work states that due to the constant evolution that is based on player tastes, the result is a constant stream of new content suited to what player's want, keeping them more engaged for longer and increasing replay value (Guha et al, 2009). This project is different in that evolution is

based upon AI actions and performance, and thus different conclusions and data can be drawn at its conclusion. In 2006, Aha and others used evolutionary learning to generate AI tactics in RTS game *Wargus*. They found that automatically evolved knowledge bases can adapt to many different strategies, and can also evolve high quality tactics (Aha et al, 2006, 83).

2.4.3 Case-injected Genetic algorithms and other variants

Some genetic algorithms use have additional features like case-injected systems. Case-injected systems can have advantages such as increasing efficiency and reducing the time taken until a good solution is found. However it can introduce selection bias and also reduce the variance in a population, and it is not usually used as it is a relatively new technique. In the past, a case-injected variant of genetic algorithms has been used which seeds the population with effective individuals from past trials in order to reduce the time taken to find an ideal solution, as well as ensure variance in the population (Louis and Miles, 2005). This prevents an entire population from dying out, which is good for obvious reasons. Most likely a form of this method will be used, though not identical. Within this experiment, an "ideal solution" is much harder or even impossible to find – within the game that will be chosen, the AI has near infinite situations to deal with, so there is no one perfect solution or set-up, although there are definitely superior ones, so saving time is not an issue. Specific cases will be injected into the population however, so that high variance can be achieved, which should prevent an entire population from dying out. This can result in selection bias as with standard case-injected systems, but this only ensures representation from all types of game agent, and this should not have a negative effect. Essentially, this behaves like mutation but it is selected as opposed to randomised. If an extinction event occurs different cases will be injected and the population restarted, which should give a different outcome.

A reinforcement learning mechanism was proposed in 2004, known as FALCON (Fusion Architecture for Learning, COgnition and Navigation), which works by providing sensory field for current states, an action field for current actions, and a reward field for

representing reinforcement values. This functions as a kind of hybrid between the traditional state based systems and learning fitness systems (Ng et al, 2009).

2.5 Intelligent AI and Human-Like Behaviour

Fogel wrote in 2006 that a calculator is not intelligent, everything it knows has been programmed by a person, they can never learn anything new, and outside of their domain, they are useless - they know what they know because a person has already done it. “The dream of the intelligent machine is the vision of creating something that does not depend on having people pre-program its problem-solving behaviour" (Fogel, D. B., 2006, 1). This is a direct parallel to the use of finite state machines within AI and video games, compared to the use of evolutionary computing and genetic algorithms. Once the conditions, metrics and end goal for a genetic algorithm has been set, the population will continue to adapt and change without any input from a programmer and in doing so they can be used to simulate a learning and organic environment. Soni and Hingston have said that while the question of what makes for an interesting or “fun” opponent is up for debate, it is widely accepted that both extreme sub-human and super-human levels of skill result in a lack of fun. Rather, for maximum enjoyment, the skill level of an opponent should generally be close to the human playing the game, and that humans prefer human-like opponents. One factor affecting this is the unpredictability of the AI - “ a more predictable opponent is easier to defeat, and predictability is generally thought to be an indicator that one's opponent is following a fixed set of instructions, computer-like” (Hingston and Soni, 2008, 363). Evolutionary computation, including genetic algorithms, help to make an NPC more efficient and more unpredictable by changing aspects of an NPC based upon fitness values and other metrics, and in turn this simulates learning and creates a new experience for the player. From these statements, it can be concluded that genetic algorithms can be better for simulated realistic and human-like AI, and that they have a lot of unexplored potential due to their minimal use in games.

2.6 Summary

From looking at many similar research papers, it is visible that genetic algorithms are a popular method for improving gameplay within many different genres of games. It has been concluded multiple times that evolutionary algorithms can save on development time and produce comparable or even better results compared to other popular methods of AI implementation, such as finite state machines. While FSMs are very popular, they have flaws that genetic algorithms can improve upon. Research is on-going as to how evolutionary computing can be implemented into modern video games, but it is widely believed that the changing, evolving content genetic algorithms can produce can keep players engaged and interested, as well as keep gameplay fresh, extending the life cycle of modern games. It is generally accepted that AI advancements have been suffering in recent years, and games that feature excellent AI, such as Alien: Isolation, are critically acclaimed and respected for the interesting scenarios and opponents that feature. Furthermore, the focus on graphics has been falling off and AI is looking more and more appealing as a way to enhance the believability of game worlds and NPCs.

3. Methodology

To help understand the ideal methodologies behind this project, it is helpful to reiterate what is

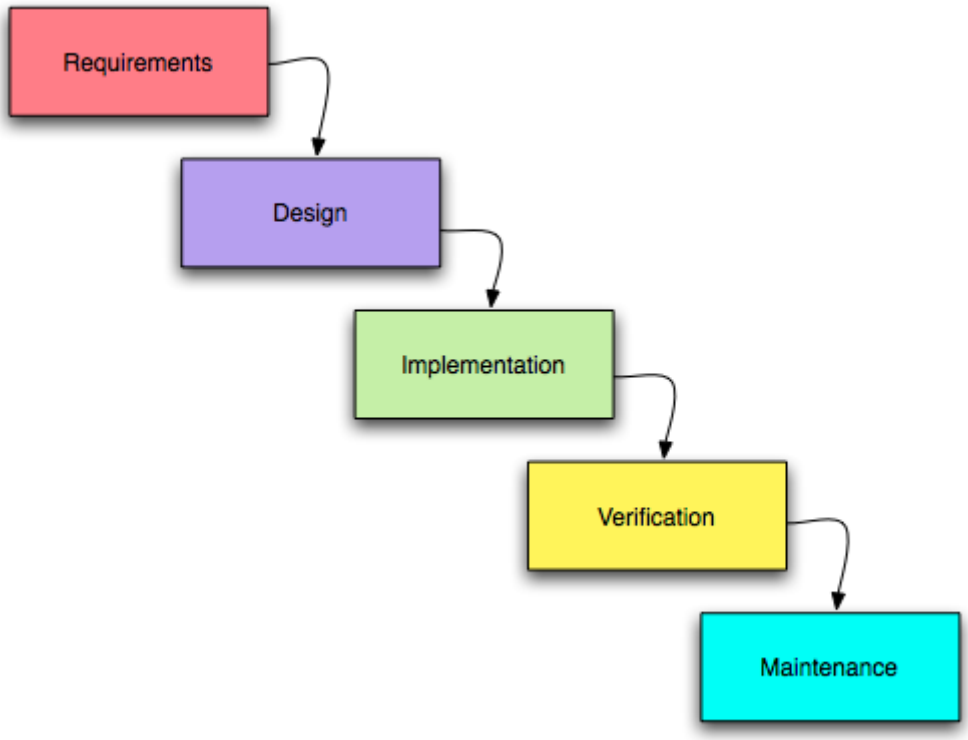
hoped to be accomplished. In the short term, the aim is to use a genetic algorithm within a popular game to see if it can enhance the "performance" or fitness of a population of NPCs within that game. Further conclusions can be drawn from this, such as the effectiveness of genetic algorithms as a method of making AI seem more realistic and human-like, or simulating intelligence. Further, this project also hopes to examine the viability of GAs within the modern game industry, and whether they can compete with or even replace other popular algorithms like finite state machines. To do this, a population of NPCs will be created within the popular role-playing game, *Skyrim*, using the free Creation Kit that it is bundled with for the creation of user content. The NPCs will be made to fight and several metrics of their performance as they fight will be recorded and the data will be analysed and used to affect the next generation of NPCs, forming the basics of a genetic algorithm. The success, ease of implementing, and the data collected will affect the conclusions that are drawn.

One of the reasons why genetic algorithms were chosen as the focus for this project is that they are a popular research choice for studies in AI and such, but lack the presence in video games, yet they may present a realistic choice for a relatively easy potential method to improve the behaviour of NPCs in games. Additionally, research has suggested that they can save time and in turn, money - with the games industry being valued at $93 billion (Gartner, 2013), this will result in huge savings and an increase in efficiency, as well as also improving the end experience for the user. There has been a surge of popularity recently for games with generated and changing content, and genetic algorithms have the potential to provide this.

3.1 Development Methodology

There's several different development methodologies that could have been chosen for the development of this project. The traditional method would be the Waterfall methodology, which divides the project into several distinct stages, each of which generally finishes before the next begins. The advantage of this is that progress is more easily measured, as the full scope of the work is known in advance, and the design can be completed before development work even begins, resulting in better software design that meshes well. What is expected from the project or artefact is usually defined right at the start, before work begins. The drawback of this method is that requirements can lack detail since they are defined at the start, or they may change. This usually applies more to those working for clients, but it is still useful to look at for this project. The other main methodology is the agile, or iterative methodology. This involves rapid delivery of an application in functional components. All work is planned a short while in advance and then performed in "sprints" which have a defined duration. As work is completed, it is continually reviewed by the client, or in this case, the project supervisor. This has a few advantages, mainly for the client, such as having frequent opportunities to see the work being delivered, and to make changes if necessary. This system can also quickly create basic working versions of the software. However, due to the nature of "sprints", it requires complete dedication from the workers. If work is not completed on time, additional sprints may be needed, taking extra time. Since this is an iterative process with less emphasis on the final product, there may be

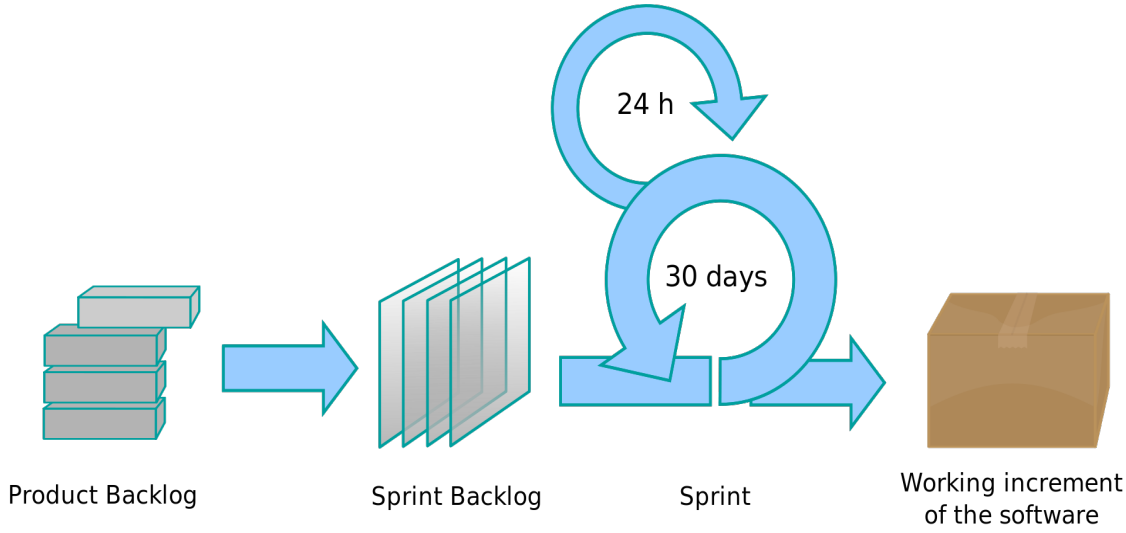
a reduction in overall quality (Lotz, 2013).



Example of Waterfall Methodology (The Smart Method, 2011)

There is also the scrum methodology, which is a subset of agile development. Agile development features twelve principles which place an emphasis upon communication and collaboration, functioning software, team self organisation, and the flexibility to adapt to emerging business realities. The philosophy behind scrum is to inspect and adapt in a loop within short cadences of work, using the sprint system already mentioned. The idea is to have a potentially shippable piece of software ready at all times. Scrums and many agile methodologies are however designed for use within teams, with clear roles such as a Product Owner and scrum masters. Since the work for this project is being performed by a single person, the scrum methodology is unsuitable for use on this project (James, 2009). The waterfall methodology would make the most sense for this project, as there is no client with specific requirements, and also because "sprints" of work are unrealistic when there is other work besides the project that must be done. Ultimately, waterfall was chosen because due to the clearly defined stages and goals that are ideal for making steady progress, and it

is helpful to plan out the order and work that must be done before having to do it. Sprints can be effective, but can lead to quick burnout and need dedication - due to the project taking place over many months, sprints would be a poor choice.



Cycle of Scrum Methodology (Lakeworks, 2009)

3.2 Evaluation Methodology

3.2.1 Qualitative Evaluation

There are a few different ways to evaluate the final results of this project. For example, an interview could be conducted asking people to play the game with and without the genetic AI systems, and see which they prefer, or perhaps only give them the modded version and ask what they liked and didn't like about it. A user study could be undertaken to find out which AI system people prefer, and which they find more believable. Without polling people's opinions, it is very difficult to find out what is more "believable". As said before, unpredictable and changing behaviour, as well as challenging opponents are all parts of what seems believable. However, it is extremely difficult to judge what is unpredictable behaviour, and hard to quantify changing behaviour - that is outside of the scope of this project. If the agents perform better with time, they can be considered to be becoming

more challenging and having changing behaviour, while ideally there would also be a user survey, this will suffice within the constraints of the project.

3.2.2 Quantitative Evaluation

Cole and other used the ELO system to compare the default agents to the genetically evolved agents (Cole et al, 2004). The ELO system is a method for calculating skill levels within a competitive game environment, originating in chess, by giving each player a skill rating that is adjusted after every win or loss, based upon the skill level of the teammates and the opponents. Classically it is used for 1v1 competitions but has been adapted for team competition, such as *League of Legends* (White, 2015) and *Counter-Strike: Global Offensive* (Namessar, 2014). Typically it is used for humans but works with bots as well. However, this would not have been appropriate for this project, as *Skyrim,* the chosen game, does not have a skill or ELO system, nor would much be gained from having one - fitness values work better for extended data gathering and use of fitness metrics, ELO systems are better for multi-player matches, measuring only wins and losses. Other researchers have evaluated NPC performance and AI systems by working out the average performance of 4 different configurations (with/without elitism and two different fitness functions) and found out the best combination, though this is more useful for determining the best version of a genetic algorithm. A quantitative approach shall be taken for this project, as it is the most common for research of this type. Additionally, part of the goal is to prove whether evolutionary algorithms can compete with more standard algorithms by comparing the results of both against each other, and quantitative data is best suited for this. For this project, only one fitness function will be used as the raw data from the metrics is normalised, this eliminated the need for weighting on each metric. A genetic algorithm with elitism is being used, which allows the most fit members of the population to pass on their data to the next generation. This will result in an improvement to be visible more quickly and also simulate real world genetics more closely, as based on research this seems to be optimal (Revello and McCartney, 2002). It is, however important to leave elements of random generation within the system, as this simulated mutation and avoids an extinction event in case of a weakness being exploited. Since the aim is to create adapting

game agents that change to create an ideal party, their effectiveness in combat is what is being measured. Metrics such as damage over time and damage taken in being measured. It is probable that the success of the project will be evaluated by comparing the fitness of an adapted population of NPCs against the initial population. The success will be based on whether the population of GA actors are higher in fitness than a default population, and how many generations it took to optimise the fitness.

4. Tools Appraisal

4.1 Engine Selection

A simple custom-made game could be have been made for this research, which is a good approach for testing one particular set of scenarios, and for creating a specific software suited to the specific needs of this project. This, however, is an unnecessary step if there already exists a software that can support the project, it would be wiser to use that, as producing a new custom-made game takes more time and effort. There are many games that could have been chosen for use in this project.

Many different game engines were examined when deciding what the platform for this project would be. The project requires support for teams of NPCs to work together against opponents and also allow for customisation of weapons, equipment and stats. Additionally, the ability to control the spawns of NPCs is required. Certain metrics must be measured, like damage dealt, so this must also be supported, whether it is in the game by default or must be coded in. As such, the ability to create custom code easily and quickly putting it within the game is vital to the project. Skyrim was the first software to be investigated initially as the platform for the project. On the surface, it appears to fit all the requirements. It comes with a free mod development tool (the *Creation Kit*) that can be used to customise many aspects of the game, and also has GUI elements for ease of use. However, after researching the Creation Kit it was found that the AI packages are not able to be altered within the tool or in any other way. Further research indicated that while this may limit some aspects of the project, it would not prove to be a major drawback and could be

worked around. Further research into other engines was done, in case a better solution presented itself. The REDkit used for the Witcher games developed by CD Projekt Red was examined, and while this may be suitable, it is intended more for the development of custom campaigns as opposed to editing the game itself, although it should be possible. However, there are few areas in the game suitable for two sides of NPCs fighting each other, so a new area may have to be created, taking away from dev time. Mount and Blade created by TaleWorlds Entertainment was also examined, though this does not have a full mod kit, but instead uses a system of Python scripts. There is official mod support, but no custom GUI made for the purpose of modding the game, instead prospective modder must use modules, which are comprised of a set of Python scripts (Yavuz, 2005). Because there is no fully implemented editor, making the project within Mount and Blade may be more difficult, however it should also allow for more in-depth changes within the engine. There is no real respawning within the game, but characters which fall unconscious can get up again, and there are systems to automatically change party equipment. Dragon Age: Origins also features mod support, however it lacks a detailed respawning system, so it may be unsuitable for this project.

Most Source engine games feature mod support via the Valve Hammer Editor, which works on games such as Half-life, Portal and Dota 2. Again, research suggests it lacks a full respawning system, or the ability to change NPC equipments automatically. However, Dota 2 would most likely be suitable, as that features 2 teams of 5 "heroes" out of over 100 fighting against each other, with bot support. The heroes are divided into categories, such as "carry", "support", "durable" and more (IGN, 2012). Instead of changing NPC equipment, the project could be carried out by changing which heroes are used to find what combination of categories is more effective. At the time of development, the official Dota workshop tools had not been released and official support for modding was limited. Additionally, the complexity of the Dota gameplay makes performance and success hard to define, resulting in a possible unreliable fitness value, especially since every character has different base stats, although this can produce unexpected results that or strategies humans have not yet noticed. Another source engine game, Counter-Strike, could have been chosen. Cole and others implemented a genetic algorithm in Counter-Strike to good results,

though the limited number of items in the game, as well as the limited number of players would reduce the amount of possible solutions and reduce the scope of the project, which is not desired. Additionally, strategies in Counter-Strike are heavily map dependent and reliant on map knowledge (Maple Apps, 2015), which means that one genetically optimised strategy may only work on one map, resulting in calibration being required for every new map.

Ultimately, the Creation Kit for Skyrim was chosen, as it has a GUI-based tool that allows for quick changes to be made for the game and is visually intuitive, with features such as drag and drop and pre-determined loadouts for NPCs and such (Fincher, 2011). It uses its own scripting language, Papyrus, that is intended to be simple and easy to learn. This could result in problems as it is a completely unique language that will have to be learned from scratch for the project, and may be capable of less than other languages, but will hopefully result in easier coding once it has been adapted to. Mount and Blade also uses a scripting language that we have little experience with, but does not have a graphical interface and we are also more familiar with the Creation Kit in general, having used it before as well as its predecessors in the previous Elder Scrolls games. The downside to choosing a commercial game is that there can be limitations on the modifications that can be made. However, it comes with the guarantee that it is functional - if a development team could use tool to create an entire commercial game, it is more than usable for the creation of a mod within game. On top of this, there are already pre-determined NPC loadouts, a game world that can be used for the mod, multiple types of enemies that can be used, and overall less content has to be created for the purposes of the project, saving time. In short, the CK (Creation Kit) supports everything necessary for the project to go ahead, and is user-friendly and intuitive.

4.2 Numerical Computing Software Selection

For the purposes of data collection, there are numerous tools that could be used. While software for data manipulation is not *required*, it will greatly speed up the process of gathering results and making bars and graphs to display them, and as such, such software

will be used. There is a huge amount of such tools available, like *Microsoft Excel*, *SageMath, SciLab, FlexPro,* and many more. Since the volume of choice is large, it is helpful to identify some requirements to rule out unsuitable or inferior software. First, since the data is being exported from the game engine in text format, the software must be capable of reading data in from text files. Second, ideally the software will allow for in-depth manipulation of the data, such as recognising and separating the different metrics into different variables, and allowing changes to be made. For display purposes and time saving, the tool should also be able to produce histograms from the data. Finally, the tool should be able to normalise the data. This still leaves several tools that can be used, though in the end, *MATLAB* was chosen. This was due to having a script-like command line system that allows for complex custom numerical equations. Additionally, it was recommended by the supervisor for the project and met all the needs outlined for the software.

5. Aims and Objectives

The goal of this project is to see if genetic algorithms can be used to enhance the performance of NPCs (Non-Player Characters, usually friendly) within a popular game, and to use genetic algorithms to optimise a population of NPCs within the game. To do this, a popular game will be selected for which a modification will be made to put genetic algorithms into the game so that the goal can be achieved. The modification will include a population of NPCs as well as a population of enemies for them to fight against, while the ability to gather information on the performance of the population of NPCs will be coded in, so that the performance of individual NPCs and the population as a whole can be measured in a numerical way, using several different metrics. By doing this, performance can be analysed and strength of individual loadouts and combinations can be determined. The aim is not to make the game agents "smarter" or "better", simply making them more effective without resorting to AI "cheats" like those that are often seen in certain games on harder setting, such as *Starcraft II*.

5.1 Aims

- To investigate if genetic algorithms can improve player engagement and improve the performance of agents within video games.
- To allow for the gathering of data on the performance of NPCs in a video game.
- To design and develop a genetic algorithm within the video game.
- To determine whether the genetic algorithm improves the performance of NPCs within the video game.

5.2 Objectives

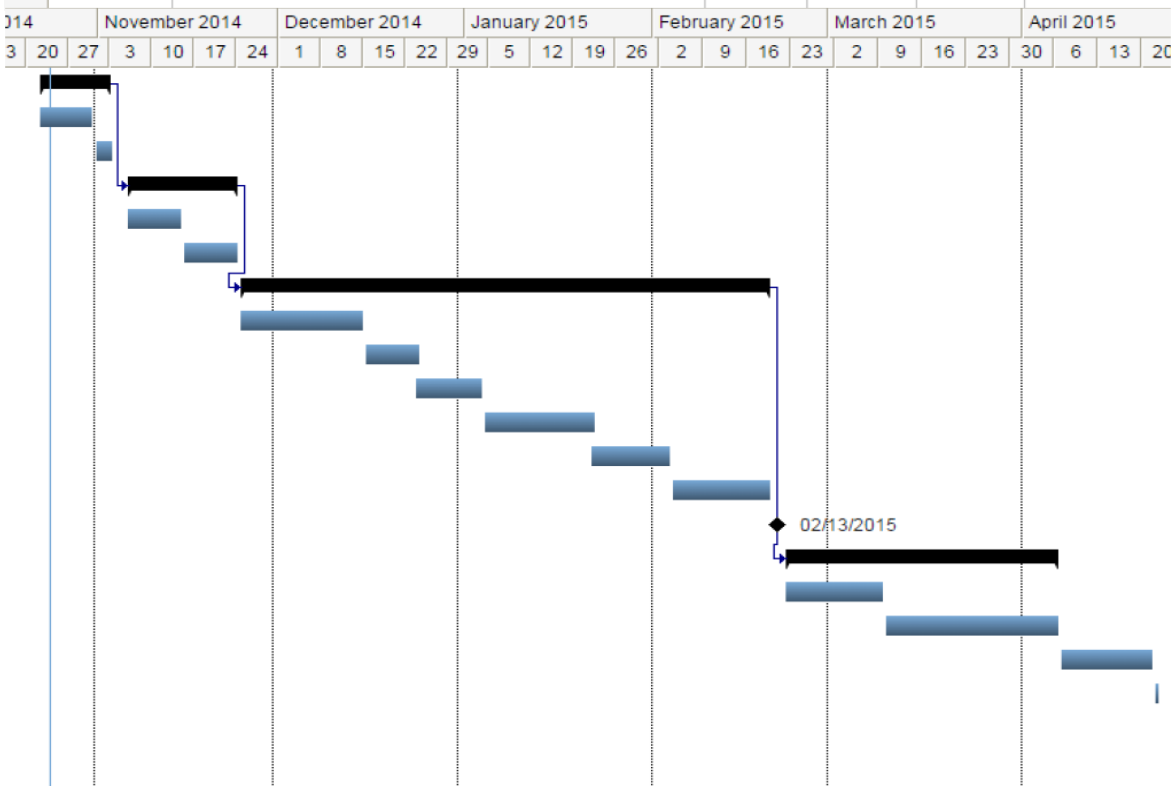
- Select a video game that supports modifications that will be used as the base for the project, or create one.
- Implement a population of NPCs as a test group, and a population of enemies for them to fight.
- Ensure different types of NPCs can spawn to ensure deviation.
- Develop a way to measure the actions of each NPC.
- Turn the actions of each NPC into a quantitative measurement.
- Use the actions of each NPC to create a numerical indicator of their performance, a fitness value.
- Implement a method to automatically publish data gathered within the game to an exterior text file that can be read by numerical software.
- Analyse the fitness value to identify the strongest NPCs or weaknesses in the calculated fitness value.
- Adjust the NPC population based upon each individual's fitness.
- Repeat the process of adjusting based on fitness until a close to optimal “strategy” or combination of different types of NPCs is reached.
- Evaluate whether the NPCs are more effective as a result of this.

6. Project Management

6.1 Project Planning

Enclosed is a Gantt Chart that details the timeline and the steps to take towards the completion of the project.

		Name	Duration	Start	Finish	Predecessors
2		**Research**	10d	**10/20/2014**	**10/31/2014**	
3		Further background research	7d	10/20/2014	10/28/2014	
4		Choosing of Game Engine	3d	10/29/2014	10/31/2014	
5		**Design**	14d	**11/03/2014**	**11/20/2014**	**2**
6		Fitness Function design	7d	11/03/2014	11/11/2014	
7		Algorithm Design	7d	11/12/2014	11/20/2014	
8		**Implementation**	61d	**11/21/2014**	**02/13/2015**	**5**
9		Creation of Population and In-game set-up	14d	11/21/2014	12/10/2014	
10		Break	7d	12/11/2014	12/19/2014	
11		Design and Implementation of Algorithm, Part 1	7d	12/19/2014	12/29/2014	
12		Christmas Break	14d	12/30/2014	01/16/2015	
13		Design and Implementation of Algorithm, Part 2	9d	01/16/2015	01/28/2015	
14		Additional Features	12d	01/29/2015	02/13/2015	
15		Finish Implementation	1d	02/13/2015	02/13/2015	
16		**Evaluation**	36d	**02/16/2015**	**04/06/2015**	**8**
17		Evaluate Results	16d	02/16/2015	03/09/2015	
18		Write Report	20d	03/10/2015	04/06/2015	
19		Extra Time	7d	04/07/2015	04/15/2015	
20		End	1d	04/16/2015	04/16/2015	



The project was divided into four main sections, Research, Design, Implementation and Evaluation, to be performed in that order. Allocated is the completion of each as a major milestone for the project. Research must be performed first as it dictates what will be done in the project proper and teaches valuable lessons. It is divided into two sections, background research which will vital for gaining a proper knowledge of the best way to go about the project and also the knowledge required to complete or even start the project. The second section is dedicated to selecting the correct game/SDK. This will be done second because background research will help dictate which mod development kit will be used. The design is the next major section and is also divided into two major parts. The fitness function may seem minor, but will greatly affect the way the NPCs evolve and therefore the results. Getting it right is extremely important as slight changes will have huge impact especially after several generations of evolution. The algorithm is of great importance as well, as so equal time has been allocated to designing it. Implementation is the next major section. It has been roughly divided up into the major tasks/objectives that are required for the genetic algorithm. Included is two breaks in this time as implementation contains the longest periods of work and also the most technical tasks. The additional features have been given a place because while they are technically not required for the algorithm, they are very helpful and will provide more stable and dependable results, though if time is short, they can be cut. These additional features cover things like elitism and mutation – useful, but the algorithm functions without them. Now that implementation has been finished, the final major part of the project is evaluation. This is probably the most important part, as good information is useless if it is not expressed or explained correctly. Over two weeks are dedicated to the gathering and evaluating of results. This is where the results of all previous work will be gathered, and it will determine if the project is a success or not. The final thing to do is write the report. Again, a lot of time has been allocated for this as if results are not expressed well, the information does not get across and the data gathered will be useless to everyone, making the project a waste. A week of "extra" time has been saved, though this is not spare time – it will be used to shore up any weak points in the project, double-check data and so on. It is effectively a safety net to ensure the project is as good as it can be.

6.2 Risk Assessment

Risk	Likelihood	Impact	Solution
Hard to adapt to new scripting language (Papyrus)	High	Medium, slows down work and may result in inefficient code	Practice, ask online for help, do tutorials and such
Struggle on implementation, due to lack of knowledge	Medium	Medium	Ask around university for people who know about them, ask online, read more research papers
Run out of time	Medium	Strong, may not finish or gather results, project may have to be changed	Try to keep to the plan outlined in the Gantt chart, if not then reduce project goals.
Stress	High	Medium, lack of work gets done	Take regular breaks, work on other things
Cannot gather enough data, or it gathers too slowly	Low	Medium, very long periods of data gathering that slows down the process and results	Let data gather for long periods, or re-write code to make it gather in different ways, or more quickly
Data does not give notable results	Low	High, hard to draw a conclusion and get results	Gather more data, change the fitness metrics

Loss of data or results	Low	High, have to start data gathering again	Keep lots of backups, re-gather the data if required and begin the process again

6.3 Backups

Since this project is completely technical and development based, the entirety of the work will be stored digitally. This naturally leads to problems such as data loss that would affect the project more severely than others. Many things could lead to the loss of data – hardware failure, power surges, theft, or even accidental deletion, and any of them would be disastrous for the project. If the data loss came late into work on the project, work may have to be rushed to catch up, resulting in poorer work and mistakes, or even complete project failure. Fortunately, data loss is easily avoidable. Mods made with the Creation Kit compress into small .esp files that take up very little space and work on every system, as long as the system has Skyrim installed. This makes data transfer very easy and work is ready to be transferred at all times - no need for export to a release version, or downloading the Creation Kit as long as no changes have to be made. This also makes it very easy to keep safe from data loss as transfer speeds and kept to a minimum and are very simple. To install the mod that the work is on simply requires the placement of the .esp file into a folder in the Skyrim directory, since the mod uses no custom textures or models, so installation is extremely easy. All work is primarily stored on a home PC, but secondary USB storage has been used that was updated
after every work session so at most only one session of work could have been lost. It is however very easy to lose USB drives so other methods of data protection are required. For online storage, several different services are available. Access to Google cloud storage is available that can be used to upload work on a regular basis, using the service Google Drive. In the event data loss occurs on Google cloud, it will not affect the project unless the other two methods of data storage both go down as well. If necessary though, other online storage services such as Dropbox are available. Github has a revision system that saves changes and allows for rollbacks to be made if required, as well as allowing for local changes without pushing to the server, and also allowing to different versions of the

project, to be merged together whenever desired - neither Google Drive nor Dropbox allows for this. Additionally, Github has a user client that allows all functionality from the desktop, without having to use a web browser, allowing for ease of use, and quick changes. Finally, we have greater experience with Github than the other two methods of online storage, so it was the natural choice. While ultimately there was no reason not to use all three, Github presented itself as the superior choice and offered everything the other options offered.

Working on the same thing for several months is extremely repetitive and can get very boring, so the risk of burnout is high. If interest is lost in the project then less time may be put into it, and lower quality work would result. This can be avoided by taking regular breaks from the project to work on other things or not work at all, so that it seems fresh when coming back to it. Conversely, setting strict deadlines for finishing certain parts of the project, so that even if motivation is lost, there will still be a timeline to stick to and work will still progress at a reasonable pace. Another option is to provide rewards for meeting deadlines or getting something done, so that there is an incentive to keep working hard.

7. Implementation

7.1 Outline and Requirements

One of the main goals for this project was to implement a genetic algorithm with the aim of finding the best combination of weapons within a group of NPCs. The implementation will be broken down into several sections including the creation of the population and their layout and items, then discussing the coding, including sections such as the gathering of data and creation of metrics. For the purposes of data collection, there can be no interference, a controlled environment must be created. The population that is being tested must be able to gather data by fighting with enemies for long periods of time with as little

as possible left to randomness. Since we are testing the effect of different equipment on NPC performance, the only variable that should be different between the NPCs is the equipment itself. Based upon gathering data from certain metrics, we can work out which type of weapon is most effective, for example. Our fitness value was based upon damage dealt, kill count, damage taken, hits dealt and hits taken. These metrics were chosen as they provide a good spread of data, and is similar to the method Cole and others used to determine the success of their Counter-Strike bots, which rewarded the bots for getting kills and securing the win for the team.

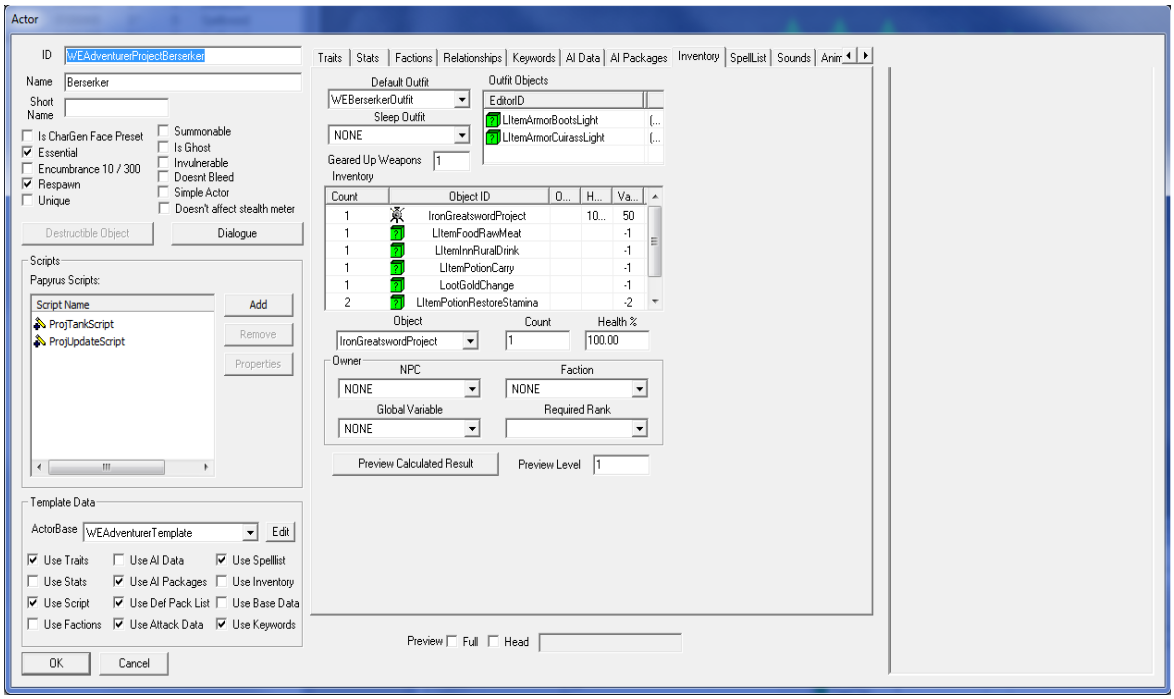
While these fitness metrics are simple, they are very effective for determining the fighting effectiveness of our game agents. Kills over time is the simplest form of this - a more effective agent should be able to get more kills in the same time than an ineffective agent. However, an agent with higher damage per hit is likely to get more kills, even if his damage over time is not higher, due to having an easier kill threshold to hit. This can produce misleading results, so damage over time must be measured as well. We must however still measure kills over time to ensure that the weapons with high kill thresholds are not under-valued again producing inaccurate results. We also measure hits over time for the inverse reason, to ensure high speed but low damage weapons are not under-represented. By using a combination of these metrics, we can find out if a weapon is stronger or weaker than others in its calibre. Damage taken is also a useful metric to track, as one NPC taking damage prevents others from taking damage, and thus increases the likelihood that the team will succeed. Some NPCs serve different roles within a group - for example, an archer should stay in the back and provide damage, and ideally should not take any damage at all. However, a shield-bearer would ideally take lots of hits as the shield will reduce the damage taken. As such, we also record the number of hits the NPC takes. Since some NPCs serve the role of taking damage, while others serve to deal it, hits taken may be either a positive of negative metric. However, it can be difficult to work out when it is a good and when it is bad, as the situation dynamically changes and there may not be a dedicated damage-taker, or “tank”, in the group. This unfortunately was unable to be investigated in the time given, but is an interesting point for further work.

The data gathered is output to a text file every so often, which can then be opened manually or with a mathematical tool - it is important to keep the format so it can be read automatically. The log file is created when it opened near the start of the script, and if there is already a log file by that name, a number convention will additionally be used. These log files are designed for bug finding and error reports, however they work just as well for publishing the information that is desired.

7.2 Population Implementation

The first thing that must be done when attempting to create a genetic algorithm is to establish a population. The CK has pre-built NPCs that have several features that can be altered as the user chooses. These features include such things as weapon, armour, items, health and other actor values, physical appearance, and AI behaviours such as aggression. Default NPCs were used as the base as these were considered good enough for release and presumably are balanced to satisfaction, resulting in more useful data about which equipment is better. Each NPC was made from a different template that came with the Creation Kit. The only difference that this resulted in was in stats. Each template appear at identical levels and fit into the same tier of difficulty and so can be assumed to be intended to be equally strong except for equipment. However, names were changed so that they could be recognised easily and were given unique weapons with scripts attached that allowed them to gather data, though the stats on the weapons were identical to the default that the game gives them.

The simulation in progress



7.2.1 NPC Implementation

Three NPCs were made for testing, all of them having the same level and the same tier of equipment, like the player would find within the game themselves. One of the NPCs was

given an iron great sword, another an iron short sword and the other an iron dagger. For the purposes of the testing they given infinite health so that data gathering could continue for long periods of time without interruption. If an NPC is copied, everything, including scripts and items is copied with it. The Creation Kit allows you to drag and drop NPCs into the game world at a location of the modder's choosing. A location in the wilderness was chosen for tests, away from anything that can interfere with results, such as wild animals or bandits. The population and their opponents were placed on flat ground so that pathing and crossing terrain would not be an issue, the population and their opponents could gather data uninterrupted and reducing the chance of AI or pathfinding errors.

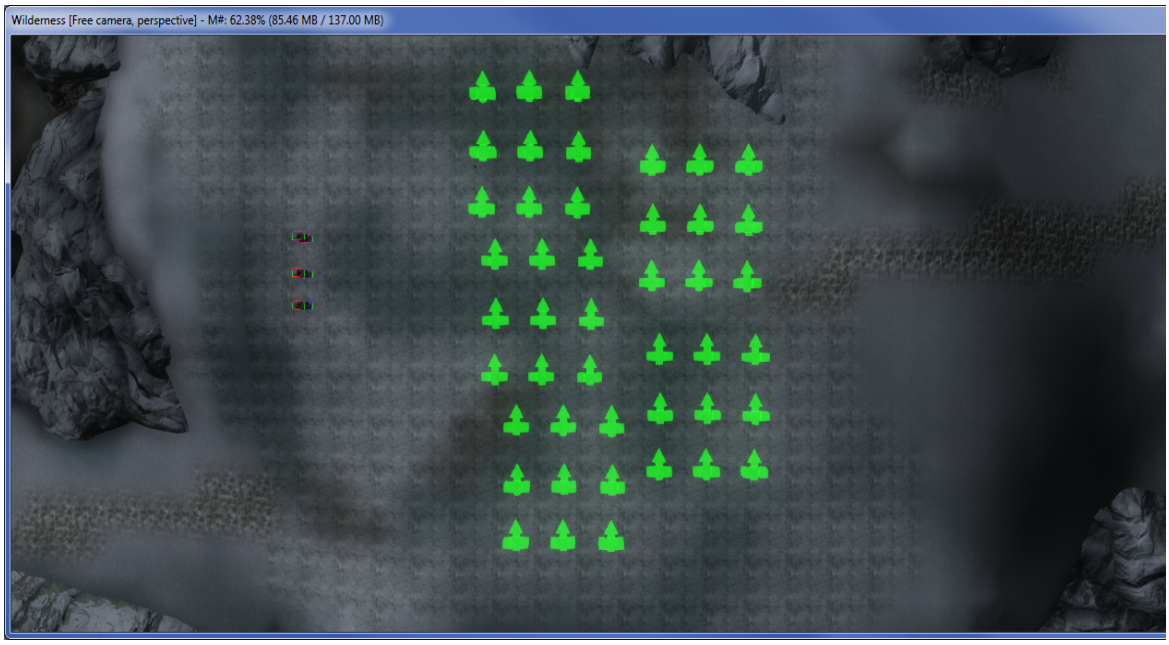
7.2.2 Enemy Implementation

The opponents that were chosen makes little difference, as long as the opponents are consistent to avoid changing elements and ensure a controlled group. As such, the group of opponents need to have the same weapons and equipment, otherwise it could produce inconsistencies with results due to NPCs attacking different enemies, or being attacked by different enemies and and taking different amounts of damage each time the test is run. Some degree of randomness is unavoidable, but to ensure an effective controlled group, the only variable that should change should be the variable that we are testing, the equipment of the NPCs. For the opponents “Draugr” were chosen as they are a common enemy in the game with a lot of variants that could be used if desired, for example different weapons. Draugr are undead enemies that appear at all stages of the game in a lot of locales, so they made sense as opponents. The Draugr used were ones with stats and equipment pre-made in the game, no changes to health or strength etc. were made. They all have an Ancient Nord Great sword and pose a small threat by themselves. Since for the purposes of testing, our control population of NPCs have infinite health, we can create a large amount of opponents for them to fight to decrease downtime between fighting and produce more consistent results, with an increase in the speed of data gathering. As such, 45 Draugr were created and placed close to our NPCs so that the fight starts immediately. The NPCs are all equidistant from their opponents so as not to give one NPC an advantage over another. The Draugr respawn after a short delay through a script.

Example of a Draugr (ferrebeekeeper, 2012)

7.2.3 Data Gathering

Gathering data was designed to be extremely easy, to speed up the iterative process and help with getting large amounts of data. A new, default player was created and the tutorials quest completed so that full play functionality was enabled. Upon completion of the tutorial, a console command was used to teleport the player to the cell where the populations were set up. A save and quicksave were immediately created so that it is very easy and quick to reload directly into an initialised state of data gathering. The save is before the data gathering actually begins, so all data is fresh. The location the player spawns in is close enough to begin the populations fighting, but not close enough that enemies or the NPCs will take notice of the player, therefore avoiding interrupting or polluting the results. If there are any changes made to the mod source file, the player can simply re open the game and press continue, and continue with the new changes.



7.3 Coding Implementation

7.3.1 Scripting in Papyrus

The Creation Kit uses its own scripting language, Papyrus. “It works by receiving Events from the game, and sending function calls to it” (Creation Kit Wiki, 2015). Scripting in Papyrus consists of several different concepts: Objects, Functions, Events, and Variables and Properties. Objects are a special type of script that recognises recognised by the engine and accepting functions, properties and events, that determine what functions can be used, what the script can be attached to, and how it will behave. A function is a section of code that are larger than a single expression, and take various parameters and return a value to their caller. They are extremely versatile and can be called and reused multiple times. An event is a special pre-defined function that the game calls when a certain event occurs. Variables and properties perform similar functions, holding data such as values and objects for when the developer needs them. A variable is private meaning that only that script can see, set and get the contents. A property has to be declared and set outside of a script, and is essentially a variable that other scripts can access, and have their contents read and changed by other scripts (Thomaskaira, 2012). If the variable or property holds a number value, getting and setting returns is value. If it holds an object, you can use it to access that

object's properties and functions. It is possible to “extend” a script so that it inherits a parent's events and functions, without having to modify the script itself. A script is extended by another script, and inherits all functions and events. If the child has their own version of a the function or event, it overrides the parent's. A script can only be extended by a single script, not multiple. When calling a non-global function on an object, you can only call functions that are available in that object's script. For example:

“**ScriptName TeleportOnBleedout extends Actor**

ObjectReference property TeleportMarker auto

Event OnEnterBleedout()
    MoveTo(TeleportMarker)
EndEvent

In plain English, "extends Actor" means "inherit all of Actor's properties and functions (including events)". If all I had in this script was that single line, then it would act as a duplicate of the Actor script. This means that, without declaring them myself, I have access to all of Actor's properties and functions, without having to add any extra identifiers to tell the compiler where to look. As well as being able to call any of Actor's functions directly, I can override them by defining them again in my script." (Cipscis, 2014)

While normally a script has to be attached for an object for it to be present in the game, certain pre-made scripts are automatically attached by the game engine, for example the Actor script is on all Actor references in the game (Creation Kit Wiki, 2015). Papyrus files are opened in the Creation Kit editor with the Papyrus Script Manager, but can also be opened in any text editor. Beginning a line with a semicolon will cause it be ignored, useful for commenting.

There are many different ways the gathering of information could be coded, for example a script could be attached to the enemies that the NPCs fight that records damage taken and finds the source of it, however this would create problems for accurately remembering who the aggressor as time goes on, as a different variable would have to be made for each NPC in the population that data is being recorded for. Another method would be to attach the script for data collection on to each of the NPCs we are examining. This method works well for the most part, however Papyrus does not have an event defined for an actor hitting another actor, therefore it would be impossible to detect when damage is dealt through this method. The final method would be to use an enchantment on the weapons of the NPCs, with a script as the effect for the enchantment. By doing this, the script can be used as a magic effect, which allows for the use of techniques not available to scripts attached to actors. The most useful of these is the ability to use the event OnEffectStart, which will detect when the enchantment is applied, which is whenever our NPC hits an enemy. This allows us to register the hits made and damage done by our NPCs, which we could not do with the other methods. However, using a script attached to the weapon leaves us unable to record the damage done the NPCs, as Magic Effect scripts do not support this. As such, there is a second script attached to each member of our population of NPCs for the purpose of recording damage received. Using this combination, we can gather data on the metrics that we want.

7.3.2 Data Gathering Implementation

When coding for the gathering of data, it was originally planned to use variables to record the values of damage done etc. However, it was found that Papyrus did not support this - using variables meant that whenever that script was used, which was every time one of our member's struck an enemy, the variables would be reset and thus could not be used to store data. Global variables can solve this problem, as they are initialised once as the game starts, and called be from any script, however the data will then not be unique to the NPCs

we are measuring - they will share data, which is unsuitable. Fortunately, A workaround to this problem was discovered shortly after. By adding an item to the inventory of the NPC in question, that represents the metric we are measuring, the data can be preserved and kept unique to each NPC. The items that were added are in the miscellaneous category; they do nothing and cannot be interacted with, they also only have physics while out of the inventory. They are weightless and value-less, they have the visual appearance however, of various objects that were used as the base, to allow them to have an icon. The only thing remaining is to ensure that there are if states that update our data only when we want to, and with the amount we want. For example, the Kill Counter should only go up if the target of our enchantment (the opponent our NPC is hitting) dies from the attack. In Skyrim, NPCs can perform executions if the enemy's health is low enough, entering into a special animation that will instantly kill the subject of their attack, and we need to take this into account when counting the kills of our NPC. Here is the result:

```
Scriptname ProjectEnchantScript extends activemagiceffect

float DamageCount
int HitCountVar
int TimerVar
int TankVar
```

Event OnEffectStart (Actor akTarget, Actor akCaster) *The caster is our NPC, and the target is an enemy*

```
    akCaster.AddItem(Pain, akCaster.GetEquippedWeapon().GetBaseDamage(), true)
    akCaster.AddItem(HitCount,1,true)

    if (akTarget.IsDead() != false && akTarget.GetKiller() == akCaster)
                akCaster.AddItem(Charkill, 1, true)
```

While we have now added these items to the inventories of our NPCs, we cannot actually do anything with them yet - there is no variable that represents them, and thus they cannot be used as an integer value as desired. Fortunately, this is simple to do:

```
  KillerCount = akCaster.GetItemCount(Charkill)
DamageCount = akCaster.GetItemCount(Pain)
HitCountVar = akCaster.GetItemCount(HitCount)
TankVar = akCaster.GetItemCount(TankCount)
  TimerVar = akCaster.GetItemCount(TimerObj)
```

```
MiscObject Property Charkill  Auto

MiscObject Property Pain  Auto

MiscObject Property HitCount  Auto

MiscObject Property TimerObj  Auto

MiscObject Property TankCount  Auto
```

Since we are using a game object as a variable in the script, we must “point” to it within the creation kit data, by declaring the variable as an property that points to the appropriate misc (miscellaneous) object. We could output the data to a text file now if we so desired, however this is done in another script. Earlier versions of this script featured both this and a message printed to the console, though these are now commented out:

```
;Debug.OpenUserLog("myUserLog")
```

1 ;Debug.TraceUser ("myUserLog", akCaster.GetActorBase().GetName() + " has killed " + KillerCount + " and dealt " + DamageCount + " and hit " + HitCountVar + " lasted " + TimerVar + " seconds" + " and been hit " + TankVar + " times")

2 ;Debug.Notification ("killed " + KillerCount + " and dealt " + DamageCount + " and hit " + HitCountVar)

For posterity, here is the script in its entirety:

```
Scriptname ProjectEnchantScript extends activemagiceffect

int KillerCount
float DamageCount
int HitCountVar
int TimerVar
int TankVar



Event OnEffectStart (Actor akTarget, Actor akCaster)

    ;Debug.OpenUserLog("myUserLog")
    akCaster.AddItem(Pain, akCaster.GetEquippedWeapon().GetBaseDamage(), true)
    akCaster.AddItem(HitCount,1,true)

    if (akTarget.IsDead() != false && akTarget.GetKiller() == akCaster)
        akCaster.AddItem(Charkill, 1, true)
        KillerCount = akCaster.GetItemCount(Charkill)
        DamageCount = akCaster.GetItemCount(Pain)
        HitCountVar = akCaster.GetItemCount(HitCount)
        TankVar = akCaster.GetItemCount(TankCount)
        TimerVar = akCaster.GetItemCount(TimerObj)

        ;Debug.TraceUser ("myUserLog", akCaster.GetActorBase().GetName() + " has killed " +
```

```
KillerCount + " and dealt " + DamageCount + " and hit " + HitCountVar + " lasted " + TimerVar + " seconds" + " and been hit " + TankVar + " times")

        ;Debug.TraceUser ("myUserLog", "\t" + TimerVar + "\t" +  KillerCount + "\t" + DamageCount + "\t" +  HitCountVar + "\t" + TankVar + "\n")

        ;Debug.Notification ("killed " + KillerCount + " and dealt " + DamageCount + " and hit " + HitCountVar)

    endif



    if (akCaster.IsInKillMove() == true)
        akCaster.AddItem(Charkill, 1, true)
        KillerCount = akCaster.GetItemCount(Charkill)
        HitCountVar = akCaster.GetItemCount(HitCount)
        DamageCount = akCaster.GetItemCount(Pain)
        TankVar = akCaster.GetItemCount(TankCount)
        TimerVar = akCaster.GetItemCount(TimerObj)

        ;Debug.TraceUser ("myUserLog", akCaster.GetActorBase().GetName() + " has killed " + KillerCount + " and dealt " + DamageCount + " and hit " + HitCountVar + " lasted " + TimerVar + " seconds" + " and been hit " + TankVar + " times")

        ;Debug.TraceUser ("myUserLog", "\t" + TimerVar + "\t" +  KillerCount + "\t" + DamageCount + "\t" +  HitCountVar + "\t" + TankVar + "\n")

        ;Debug.Notification ("killed " + KillerCount + " and dealt " + DamageCount + " and hit " + HitCountVar)
    endif


EndEvent

MiscObject Property Charkill  Auto
```

MiscObject Property Pain Auto

MiscObject Property HitCount Auto

MiscObject Property TimerObj Auto

MiscObject Property TankCount Auto

As mentioned earlier, some metrics cannot be recorded in this single script, such as the damage the NPC has taken. A second script was made to record metrics like these. Additionally, the time that the NPC has been fighting needs to be recorded, so that the data can take time into account. For example, it is more useful to find out the rate of damage than it is to get the total damage in a session, especially if we don't know how long the session lasted. Since the goal is to make a timer, we should add one instance of the the timer object every second. Within the Creation Kit, we must first register our scripts for updating with time, and then use the event that uses this update. It is only possible to register for updates once per script, but any value of time can be used, and both game time and real-time can be used. We register for updates when combat begins, which should be when the game opens, as the NPCs should immediately see and attack the enemy Draugr. We need to make sure that we only register for updates once, or problems may occur. Since the purpose is to make a timer that counts in real world time, that is the option that is used.

Event OnCombatStateChanged(Actor akTarget, Int aeCombatState)
if (IsInCombat() == true && DoOnce == false)
RegisterForUpdate(1.0)
DoOnce = true
EndIf

if (IsInCombat() == false)

```
UnregisterForUpdate()
DoOnce = false;

EndIf
EndEvent



Event OnUpdate()

if (IsInCombat() == true && IsDead() == false)
    AddItem(TimerCount, 1, true)
EndIf

;Debug.Notification (GetActorBase().GetName() + " fought for " + TimerVar + " seconds")
;Debug.TraceUser ("myUserLog", "\t" + TimerVar + "\t" +  KillerCount + "\t" +
DamageCount + "\t" +  HitCountVar + "\t" + TankCount + "\n")

EndEvent
```

Since the rest of this script follows the same principles as the first script written above, the script will be simply be posted in its entirety as it is assumed no further explanation is needed.

```
Scriptname ProjTankScript extends Actor

int TankCount
int TimerVar
int HitCountVar
```

```
int KillerCount
float DamageCount
bool DoOnce = false;



Event OnCombatStateChanged(Actor akTarget, Int aeCombatState)
if (IsInCombat() == true && DoOnce == false)
RegisterForUpdate(1.0)
DoOnce = true
EndIf

if (IsInCombat() == false)

UnregisterForUpdate()
DoOnce = false;

EndIf
EndEvent

Event OnHit (ObjectReference akAggressor, Form akSource, Projectile akProjectile, bool
abPowerAttack, bool abSneakAttack, bool abBashAttack, bool abHitBlocked)
AddItem (Tank, 1, true)
TankCount = GetItemCount (Tank)
;Debug.Notification (GetActorBase().GetName() + " has taken " + TankCount + " hits")
EndEvent

Event OnUpdate()

if (IsInCombat() == true && IsDead() == false)
    AddItem(TimerCount, 1, true)
EndIf

;Debug.Notification (GetActorBase().GetName() + " fought for " + TimerVar + " seconds")
```

```
;Debug.TraceUser ("myUserLog", "\t" + TimerVar + "\t" +  KillerCount + "\t" +
DamageCount + "\t" +  HitCountVar + "\t" + TankCount + "\n")

EndEvent

MiscObject Property Tank  Auto


MiscObject Property TimerCount  Auto

MiscObject Property Charkill  Auto

MiscObject Property HitCount  Auto

MiscObject Property Pain  Auto
```

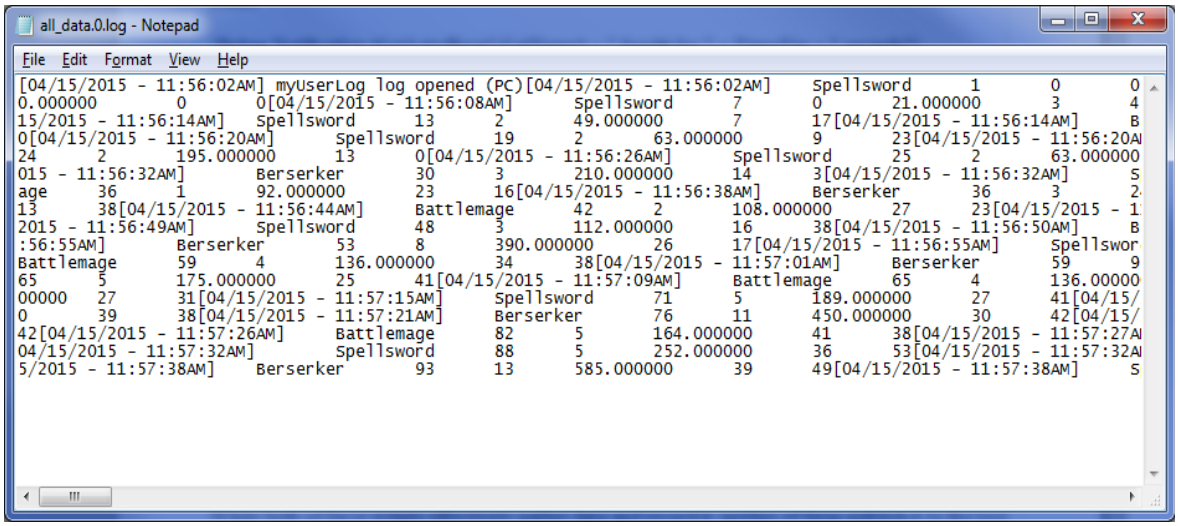
While both of these scripts effectively gather data and record it, neither of them publish it to the text file. A third script was created to do this, also attached to each of the NPCs in the population.

```
Scriptname ProjUpdateScript   Extends Actor

bool DoOnce
int TankCount
int TimerVar
int HitCountVar
int KillerCount
float DamageCount
```

```
Event OnActivate(ObjectReference akActionRef)
        RegisterForUpdate(1.0)
        Debug.OpenUserLog("myUserLog")
EndEvent


Event OnUpdate()

TimerVar = GetItemCount(TimerCount)
KillerCount = GetItemCount(Charkill)
HitCountVar = GetItemCount(HitCount)
DamageCount = GetItemCount(Pain)
TankCount = GetItemCount (Tank)
TimerVar = GetItemCount(TimerCount)
Debug.TraceUser ("myUserLog", "\t" + GetActorBase().GetName() + "\t" + TimerVar + "\t"
+  KillerCount + "\t" + DamageCount + "\t" +  HitCountVar + "\t" + TankCount + "\n")
EndEvent



MiscObject Property Charkill  Auto

MiscObject Property TimerCount  Auto

MiscObject Property HitCount  Auto

MiscObject Property Pain  Auto

MiscObject Property Tank  Auto
```

There is a function built-in to Skyrim to publish a line to a text file with a custom name which is being used for the publishing of data. These log files are designed for bug finding

and error reports, however they work just as well for publishing the information that is desired, and they publish with a time stamp of each line. The purpose of this script is to select the text file myUserLog in the Logs directory of the Skyrim game, then every one second push certain text information to that log file. The log file is created when it opened near the start of the script, and if there is already a log file by that name, a number convention will additionally be used. The resultant log file can be read by any text editor. This is the result of all of these scripts after several minutes in a single session:



While there is now a method to gather data, the session can only last as long as there are monsters to defeat. While Skyrim does have a method to reset a cell (section of game world) back to its default state, reviving the enemies. This will only work if the player is not in the cell, and the player must be in the cell for it to simulate and for the test to run (Skyrim UESP, 2014). As such, this method is unsuitable. As a way around this problem, an enemy in the population will respawn shortly after it dies. They do not use a respawn animation, they simply return to life ready for combat, so that an NPC will not gain an advantage by killing the same enemy repeatedly.

Scriptname ProjectRespawnScript extends Actor

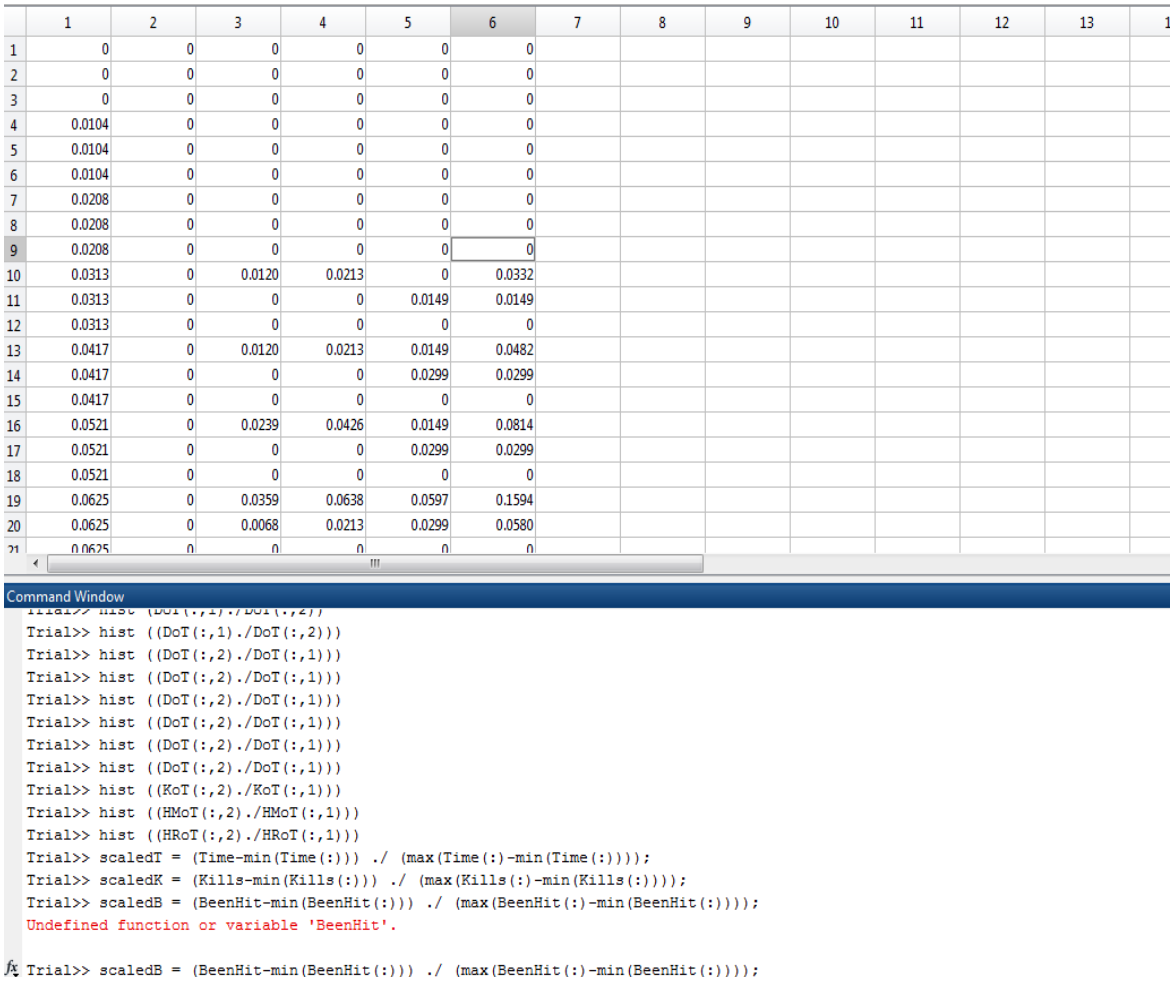
Event OnDeath (Actor akKiller)

Utility.Wait(50.0)
Resurrect()

EndEvent

This simply script triggers on the death of the enemy, where after fifty seconds it will come back to life, and can be multiple times, though only once per death.

7.4 MATLAB Implementation



	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0	0	0	0	0	0							
2	0	0	0	0	0	0							
3	0	0	0	0	0	0							
4	0.0104	0	0	0	0	0							
5	0.0104	0	0	0	0	0							
6	0.0104	0	0	0	0	0							
7	0.0208	0	0	0	0	0							
8	0.0208	0	0	0	0	0							
9	0.0208	0	0	0	0	0							
10	0.0313	0	0.0120	0.0213	0	0.0332							
11	0.0313	0	0	0	0.0149	0.0149							
12	0.0313	0	0	0	0	0							
13	0.0417	0	0.0120	0.0213	0.0149	0.0482							
14	0.0417	0	0	0	0.0299	0.0299							
15	0.0417	0	0	0	0	0							
16	0.0521	0	0.0239	0.0426	0.0149	0.0814							
17	0.0521	0	0	0	0.0299	0.0299							
18	0.0521	0	0	0	0	0							
19	0.0625	0	0.0359	0.0638	0.0597	0.1594							
20	0.0625	0	0.0068	0.0213	0.0299	0.0580							
21	0.0625	0	0	0	0	0							

Now that the data has been gathered, it is time to use it. The first thing that must be done is import the raw data into MATLAB, the chosen maths compuational software. The format is

which data is published to the log file makes this extremely easy - We can import the name of the NPC, the time, the number of kills, amount of damage done, hits made and hits taken and it can be formatted as a matrix. From here, the different data sets, for example time passed and damage down, can be split into separate variables and manipulated as desired. It is useful to make a histogram of the data, one for each NPC and one for the whole population, as this is an excellent way of displaying range of values and the frequency of them, allowing for an easy way of seeing outliers, averages and checking the normal distribution. To make this easier, MATLAB has a built-in histogram function. A histogram shows the frequency distribution of a set of data, so it is excellent at finding outliers or mistakes, as well as for showing if data fits a bell curve. As well as this, it is a good way of converting a lot of data into a visual format, where it would otherwise take a lot of time to examine or copy. To produce a fitness value all of the metrics were normalised so the values varied between 0 and 1, and then all the metrics except for time were added. Variants of the following command were used to normalise the metrics:

scaledK = (Kills-min(Kills(:))) ./ (max(Kills(:)-min(Kills(:))));

8. Evaluation

From the aims that were defined earlier, a population of NPCs and enemies has been made and a method to gather data from numerous metrics has been implemented. As part of the aims and to analyse the performance of the NPC population, an evaluation needs to be carried out.

8.1 Evaluation Methodology

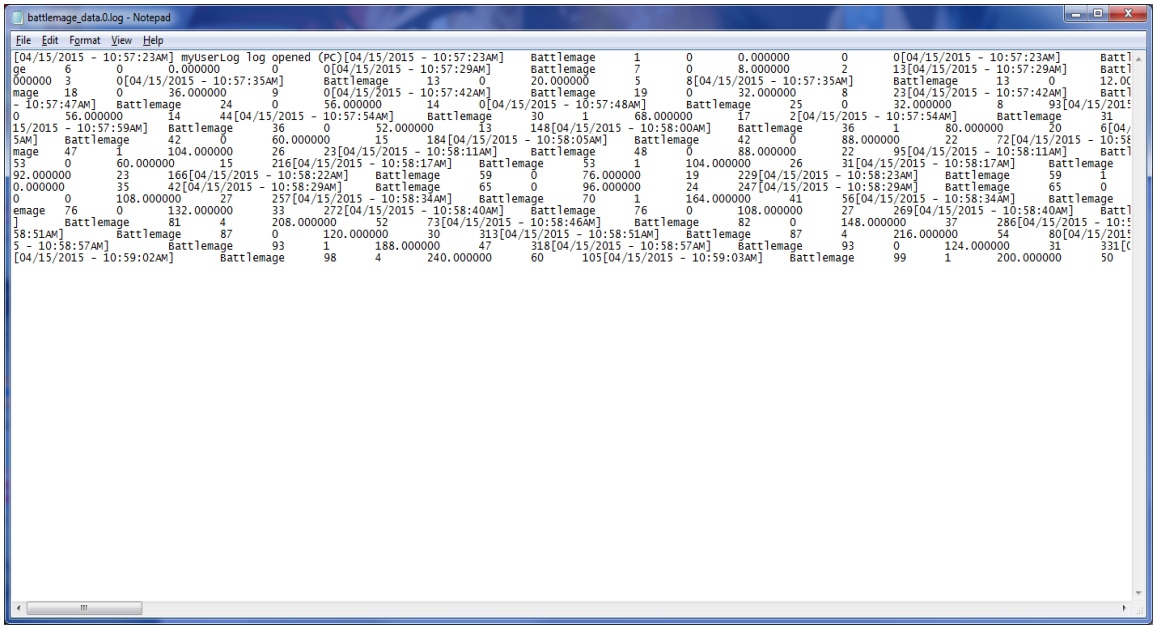
As mentioned previously, due to the goal of this project being to compare and analyse the performance of a population of NPCs by quantifying their actions. As such, it makes the most sense to use a quantitative evaluation method, which also fits with past research on this topic.

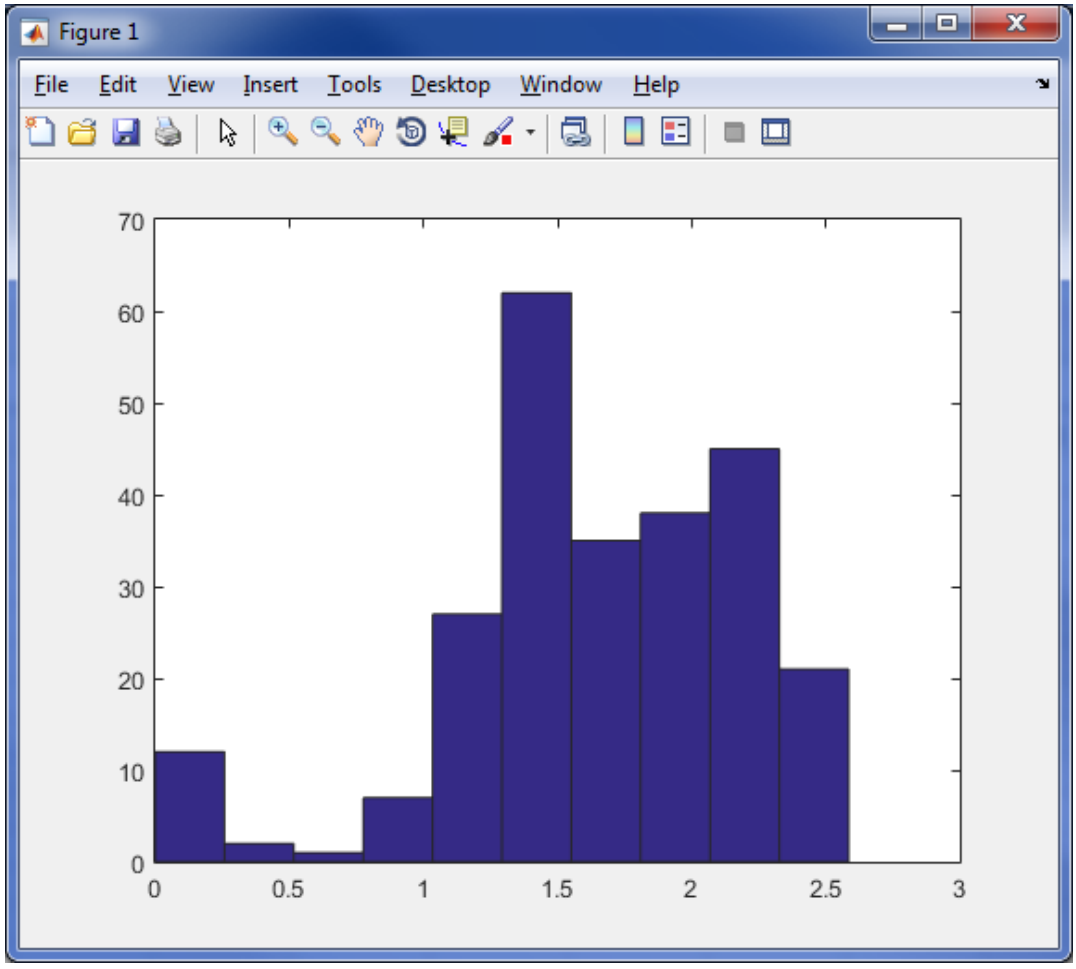
Unfortunately, the project did not reach completion - the process of automatically improving and reiterating through NPC generations was not implemented, although a fitness function was able to be created and imported into a mathematical computing

software. This was primarily due to Skyrim being unable to automatically change the equipment of an NPC based upon numerical values. To do this, the ability to result in randomised equipment in order to simulate an organic population would be required, where data can be measured on each piece of equipment that may randomly spawn. The chance of each equipment piece to spawn would change depending upon fitness values - for example, if a short sword were the most effective weapon in generation one, it would be more likely to spawn within the NPC population in generation two. While random weapon spawning is possible in the Creation Kit using levelled lists, it would result in an equal chance of each individual equipment piece. This is fine for generation one, but not for later generations as it would render the automatic optimisation of the population impossible. While the goal and premise of this project was sound, if the opportunity were presented again, the game engine that serves as the basis for the project would most likely be changed. The engine that would be chosen is uncertain. Since fitness values were able to be calculated, these will be evaluated instead, by comparing the fitness of each NPC's fitness value. The fitness value is attained by the normalisation then addition of all the metrics - kills made, damage done, hits made and hits received.

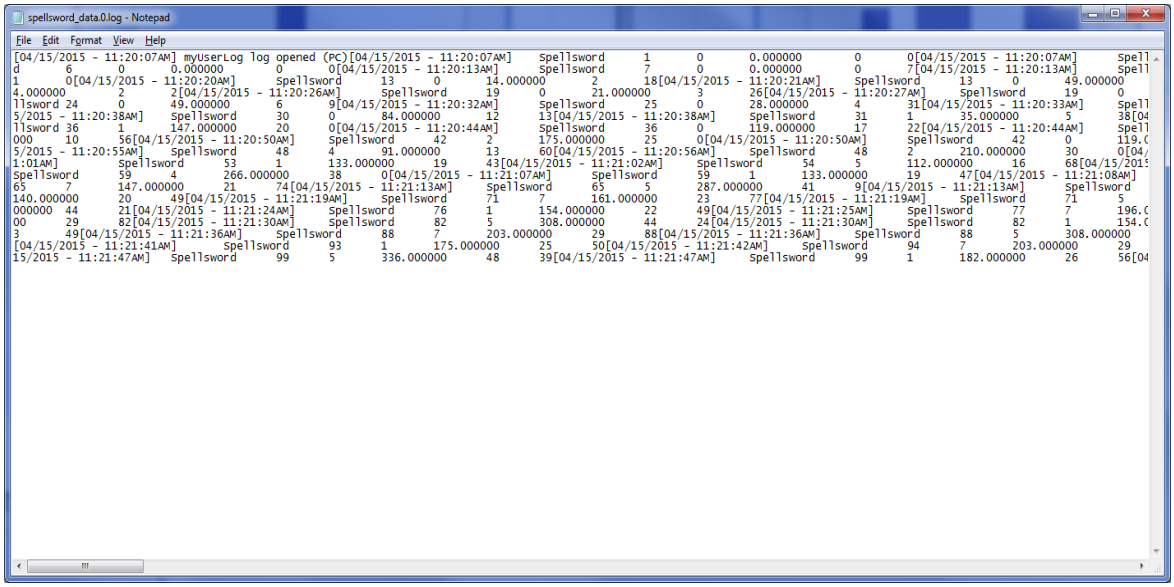
8.2 Results

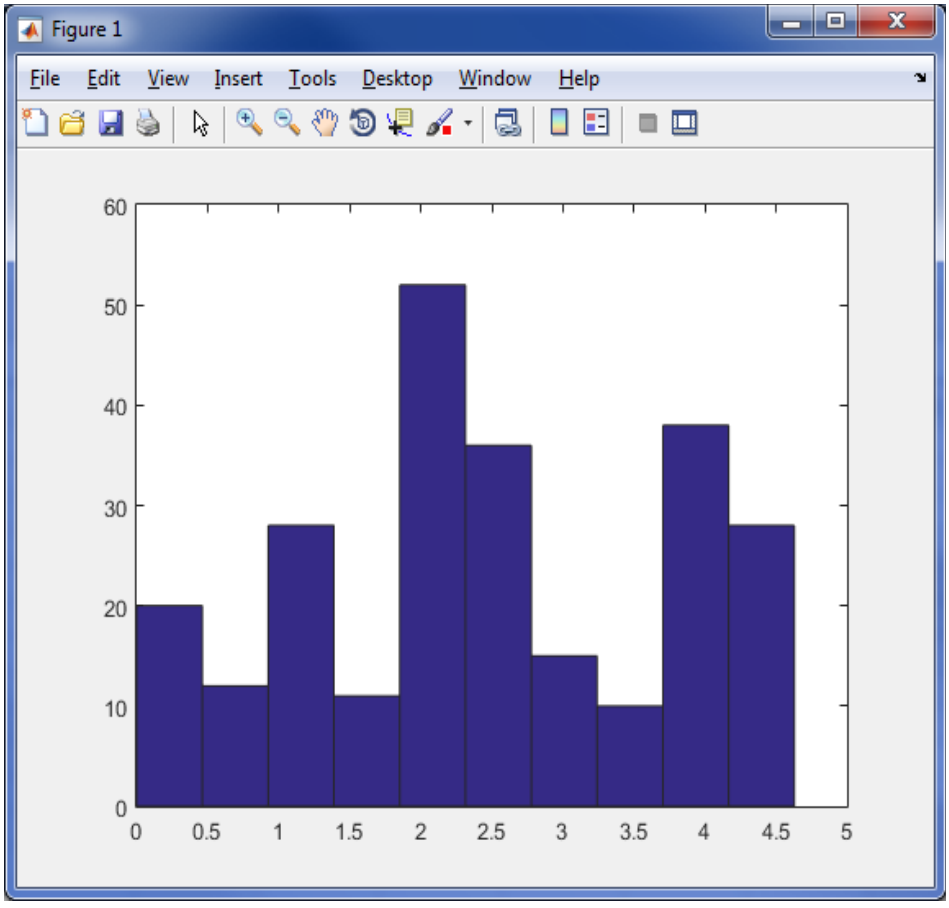
Firstly, data will be gathered from multiple sessions about the performance of the NPCs. Currently the population features three NPCs, all with different weapons. Several histograms will be made, each measuring damage inflicted over time, one for each type of NPC and one for one of each. When measuring a single type of NPC, three of the same type of NPC will be placed in the positions the default population (one of each type) would be in, to keep the AI interactions the same and minimise the impact of the change.



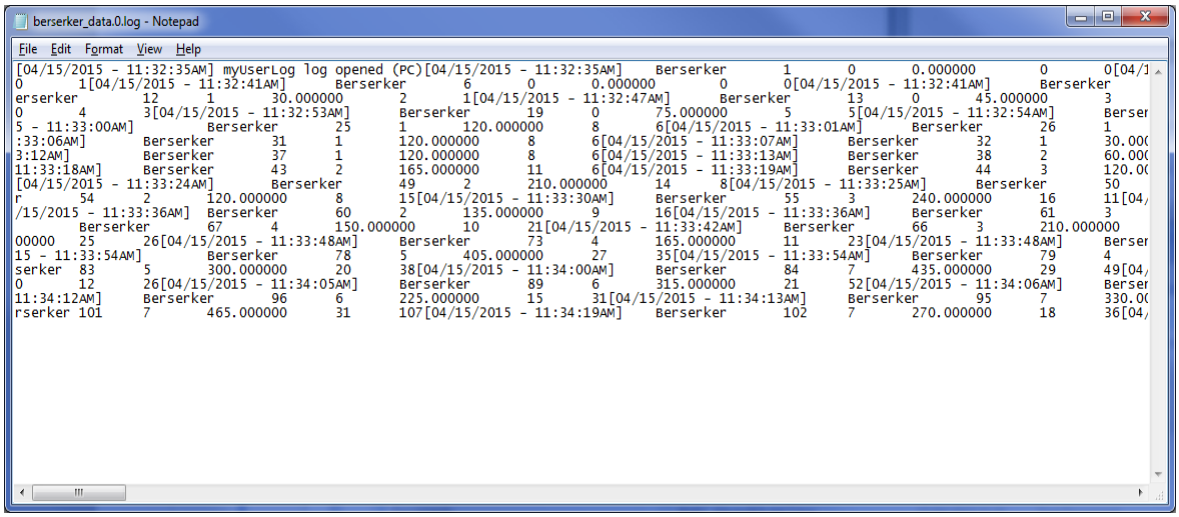


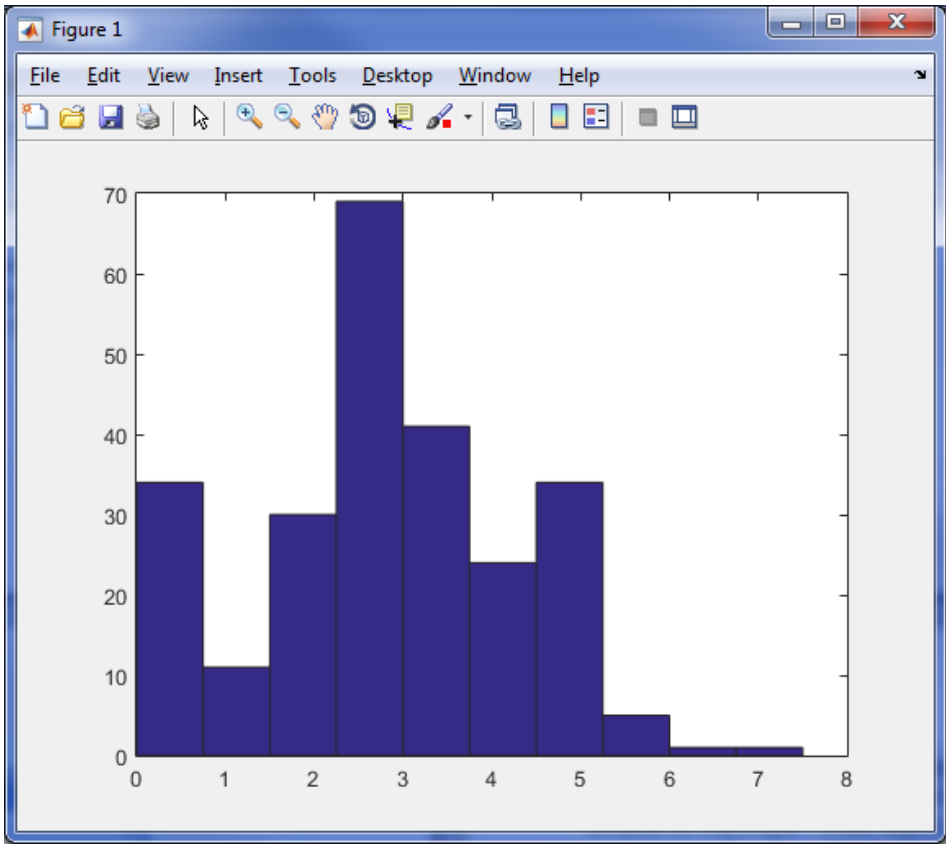
Histogram of Damage over Time of 3 Iron Dagger-wielding NPCs



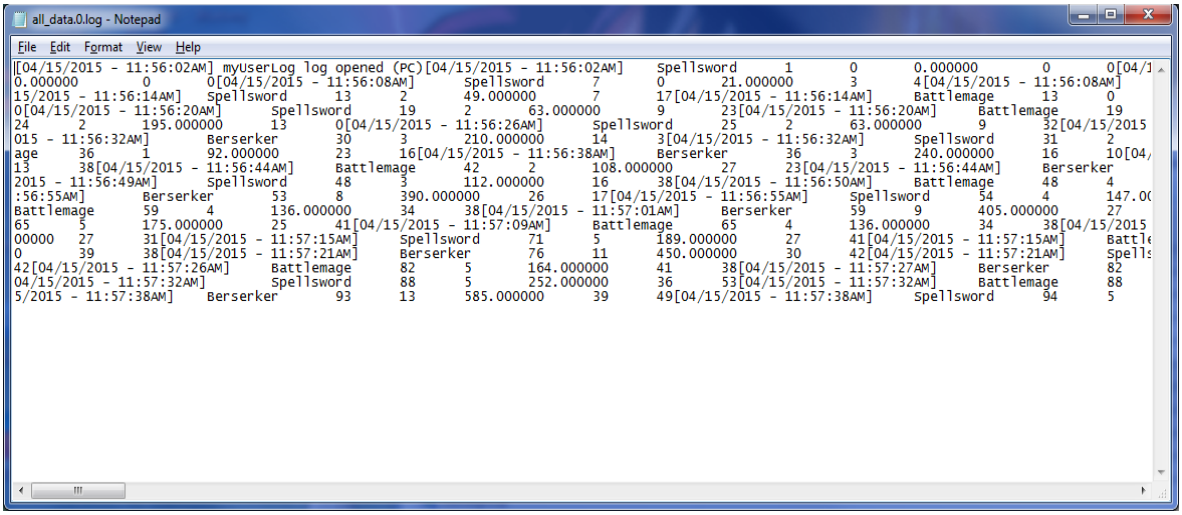


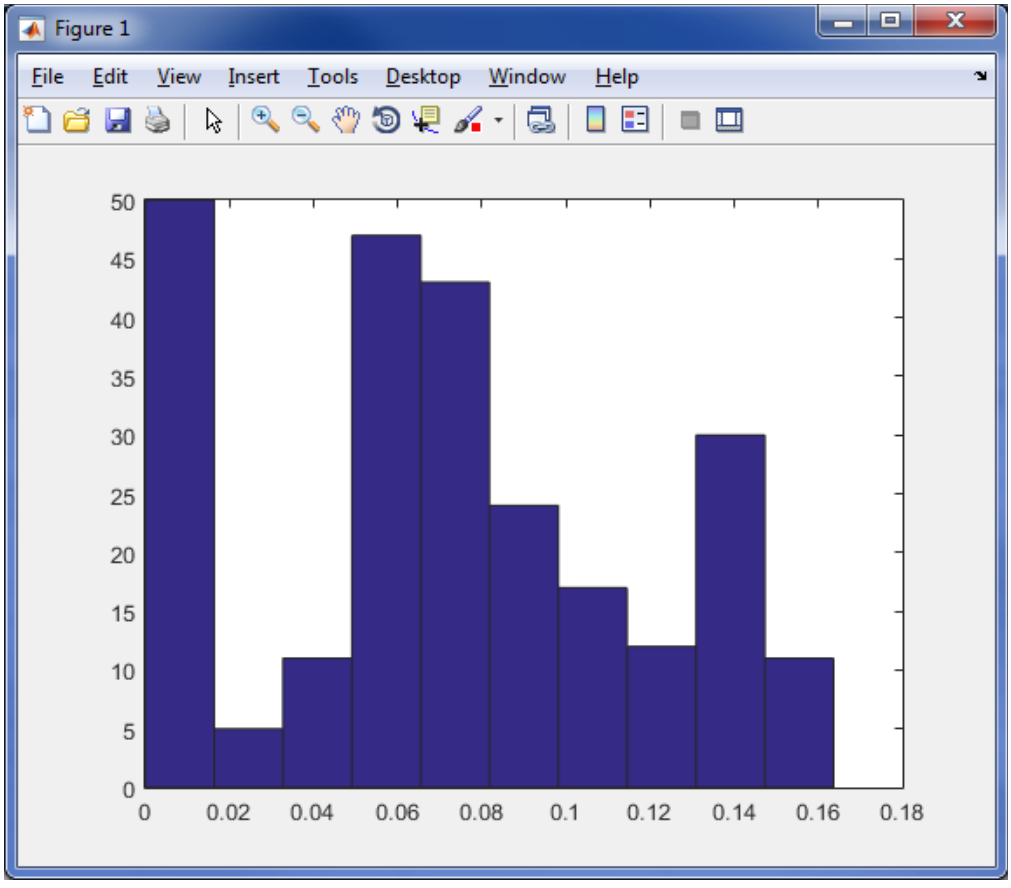
Histogram of Damage over Time of 3 Iron sword-wielding NPCs



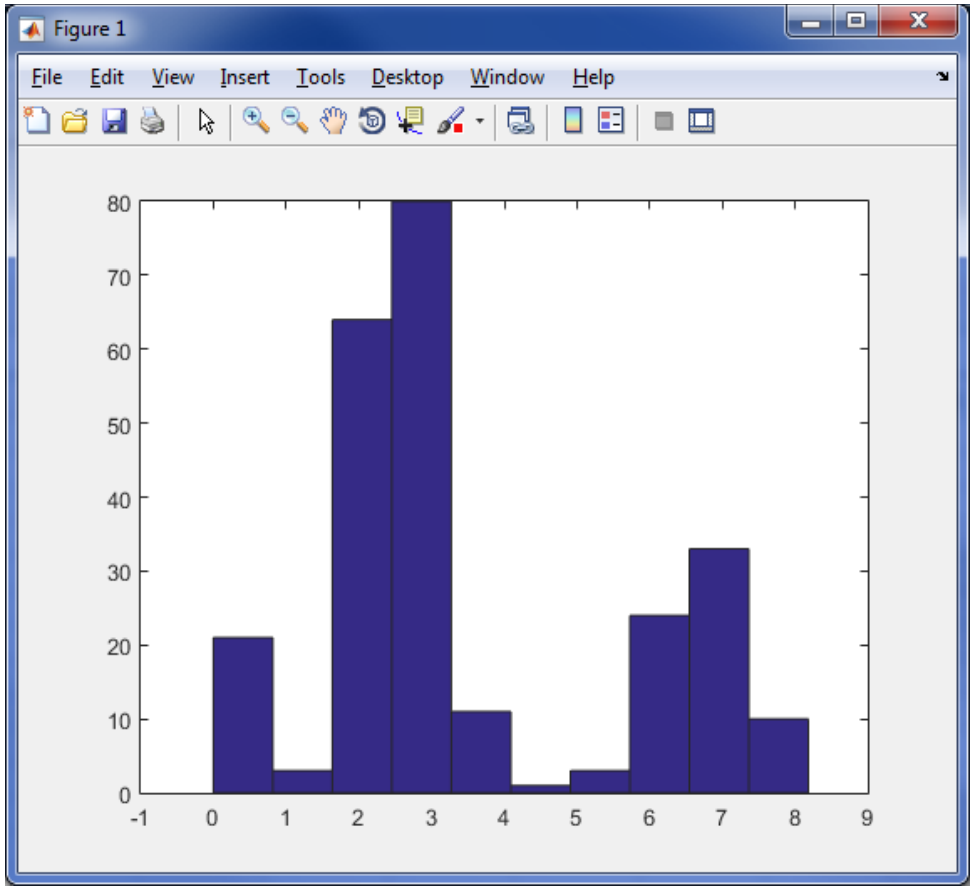


Histogram of Damage over Time of 3 Iron Greatsword-Wielding NPCs

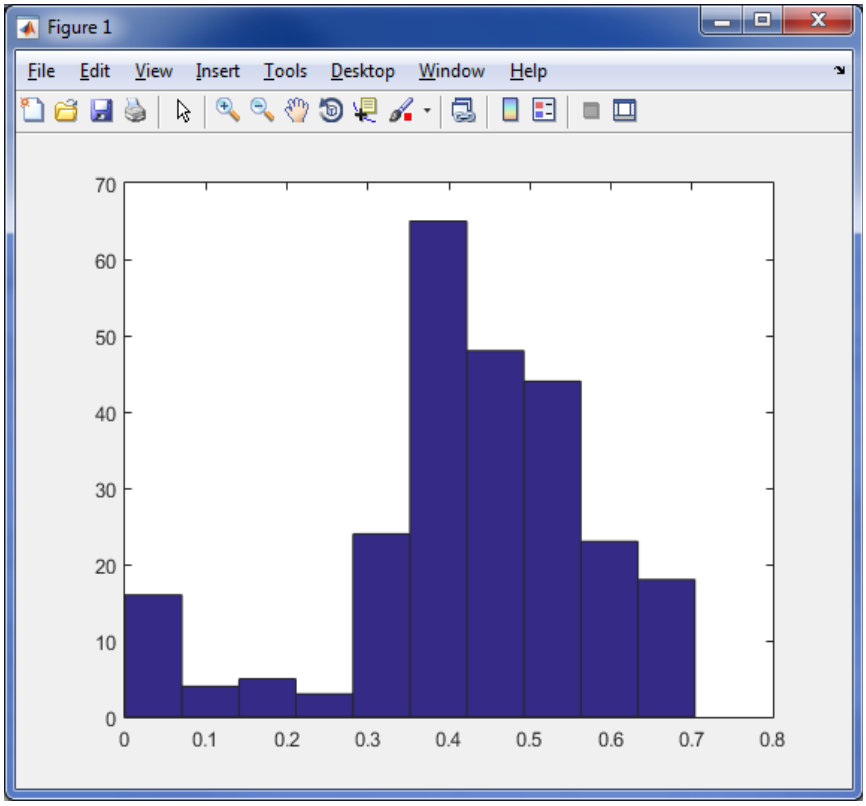




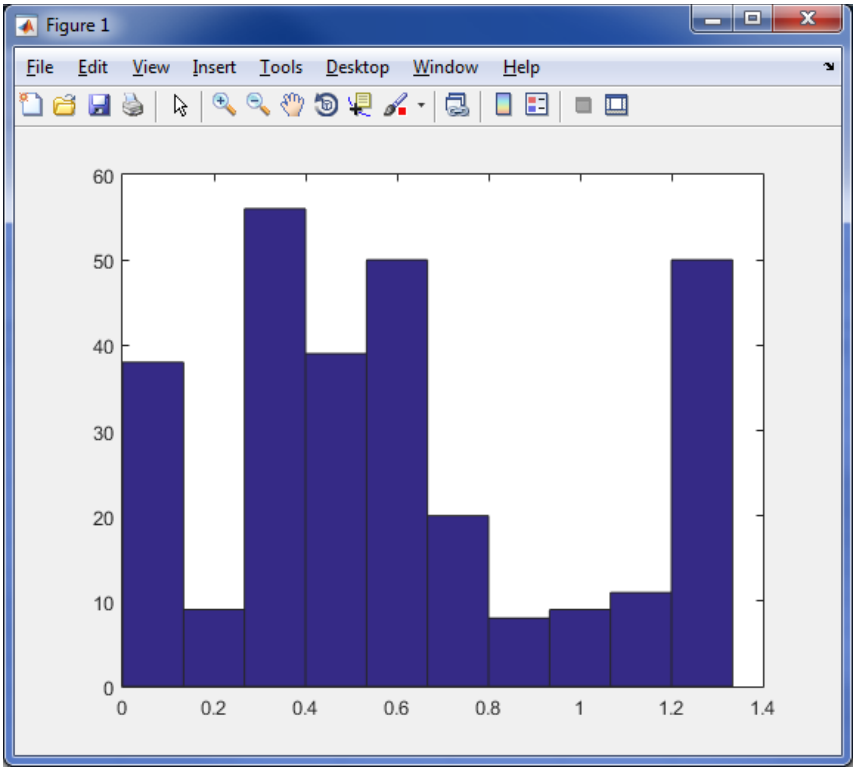
Histogram of Damage over Time of 1 of each type of NPC (3 total)



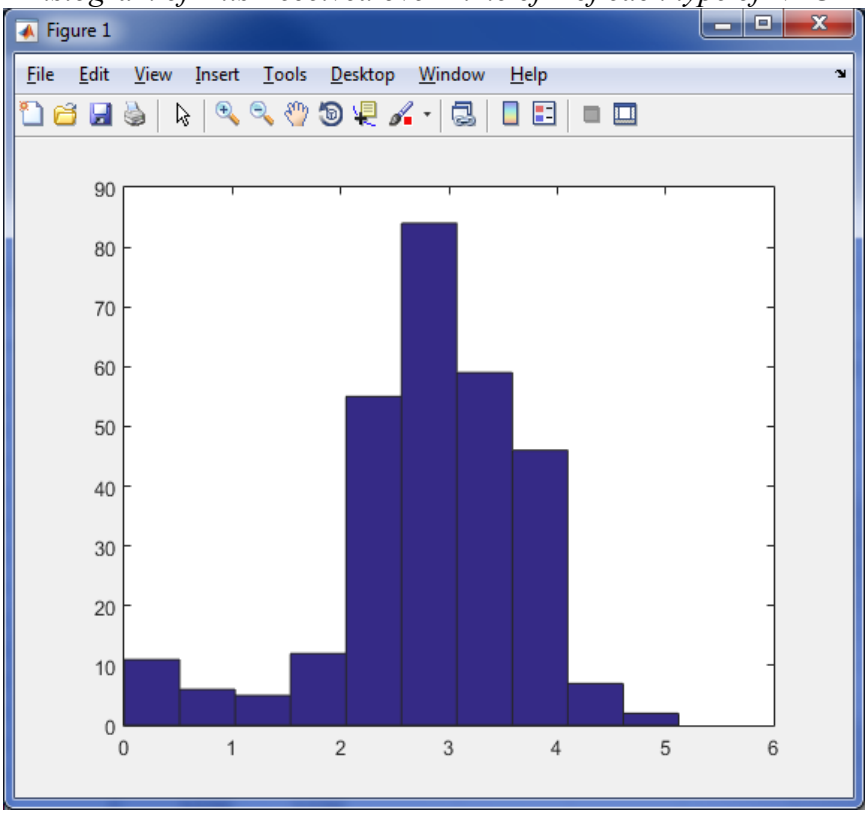
Histogram of Kills over Time of 1 of each type of NPC



Histogram of Hits Made over Time of 1 of each type of NPC



Histogram of Hits Received over Time of 1 of each type of NPC



Histogram of Fitness Value of 1 of each type of NPC

The fitness values above were calculated by normalising all of the metrics so that they all had equal effect on the fitness and so that scaling had no effect. All of metrics were added together except for time, then a histogram was made of the result against time.

It is extremely difficult to represent the vast amount of data that was created in a format that is visually appealing and easy to draw conclusions from. As such, only the most relevant data has been displayed. The data in its entirety shall be attached in the digital supporting files section. After analysing the data, it is clear to see that the great sword-wielding NPC has the highest damage and kills over time of all the NPCs. Since both regular swords and daggers can be dual-wielded, there is no reason to use dagger over swords, except for the extra damage multiplier when “sneak attacking” with the perk *Assassin's Blade*. As observed from the data swords do more damage and as such will gain more benefit from the sneak attack multiplier, which is the same without the aforementioned perk. A dagger in Skyrim offers no other benefits and thus there is no reason to use one in open combat. The NPC equipped with the great sword had roughly 10% more health by default than the other types of NPCs. As said, the stats for the NPCs came with the Creation Kit and were not altered except for the purposes of testing. This extra health is presumably to offset the fact that a two-hander (the great sword) cannot use a shield, though it would take more sessions with new NPCs to determine whether it is better to do and take more damage by using a two-hander, or if a one-handed sword and shield is more effective to deal and take less damage. Since for the most part, NPCs in Skyrim are unable to dual-wield, a two-handed great sword is the optimal strategy for doing melee damage.

9. Conclusion

The goal of this project was to determine if genetic algorithms could be used to improve the performance of NPCs within a video game. From investigation of the subject and the first aim of the project it was determined that genetic algorithms and evolutionary computing in general. It was found that genetic algorithms have value in the games

industry by researchers such as Cole, Louis and Miles, and Bryant, Miikkulainen and Stanley. They found that genetic algorithms can be adapted to deal with almost any situation, are as effective or better than the current industry standard, finite state machines, and can be faster to implement than FSMs.

From the second aim of the project was to select a video game to use in order to gather data on certain metrics of NPCs, so that a final rating of their performance, a fitness value, could be constructed. The selection of the metrics was based upon past research from the likes of Cole, Louis and Miles to create gather reliable data that is of use. A video game engine with modification software selected, *Skyrim*, that served as the base for the project.

Unfortunately, the further aims, based around the completion of a genetic algorithm were not able to be completed, due to the lack of support from the Creation Kit. It is important to note that the failure to meet certain aims of the project is not due to a problem on the part of genetic algorithms or that they are too complex to implement within modern games. Instead, the problem is due to the Creation Kit not being able to modify many game systems, to prevent users from exceeding the bounds of copyright - some aspects of the game are locked away from general users, but if developer access were available, it would be possible to fully realise all the goals of this project.

Because of this, the project had to be evaluated differently. The new evaluation was based upon the data gathered and conclusions were drawn as result of this, to analyse the performance of each NPC and what the implications of that were for weapon balance.

9.1 Future Works

Since the project was heavily time-constrained, the scope of the project suffered as a result - there is much yet to be discovered, were more time available. Within the same engine, more research would be done into the most effective equipment available and how this would impact balancing and could be used to impact player fun. By obtaining fitness

values on all weapons, it would be possible to find out which weapons are over and under powered, and adjust weapon statistics accordingly. This should result in a fairer, more satisfying experience for the player, instead of relying on what the programmer “feels” to be balanced. Additionally, by finding information on what is strong and combines well together, enemy encounters can be tuned in more ways than currently - tuned by equipment as opposed to just adjusting stats such as health and strength, which is not very realistic. This should result in a more immersive experience for the player - they can tell at a glance which enemies are stronger and act accordingly. Conversely, this information can be used to improve companion NPCs - the game can detect what weapon or armour the player is using based on their skills or what they have equipped, and then spawn complementary NPCs based upon data of strong combinations. Since the project cannot be taken deeper within this engine based upon the Creation Kit limitations on equipment spawning, another engine could be chosen, one that supports what is needed. This is a significant task, although it would be quicker than this first attempt due to the experience gained. Based upon the initial investigation into an appropriate engine, it is possible a Source Engine game would be chosen, as they are versatile and come equipped with mod tools for which completely game-changing mods have been made. However, since that initial investigation was flawed due to incorrectly identifying an appropriate engine, another tool investigation would be carried out before making a definite choice.

9.2 Critical Reflection

Upon reflection, I would consider the project as a whole a mixed success and were reasonable in scope and ambition. Research was carried out effectively and was informative for the purposes and designing and planning the rest of the project, although it was at times hard to find research that was helpful or related to the project matter, which slowed down initial progress, and made it harder to make an informed choice regarding the engine to be used for the project.

The planning phase of the project went quite well. Several different objectives, milestones and stages were noted and accounted for with regards to time. Initial work on the project

was slow due to initial lack of comfort with the Creation Kit as well as its scripting language, Papyrus. The creation of the population went quite smoothly, requiring no coding and being mostly GUI based. However, problems soon occurred when trying to create output to the screen, especially with displaying NPC names, as there no functionality for this within the Creation Kit. The solution for this was to the install the Skyrim Script Extender (SKSE), which adds several additional functions, which made data gathering a lot more precise and accurate. The next problem that the implementation faced was to code for the counting and summing of the data gathered, as opposed to the metrics resetting every time the script was called. Again, this was due to a limitation of the creation kit as it is rare for a group of NPCs to need to save a custom number individual to it, outside of inventory. This delayed the project for some time until the solution to use the inventory and its contents as a variable as a workaround was discovered. Progress went relatively smoothly after that, simply a matter of inventing and coding more metrics using the same method, which was quite quick. Pushing the results to a text file turned out to be relatively simple, initially it was thought that there was no in-built function for this, and mods were looked at in case any used this process. After it was the desired function was found, it was implemented smoothly, and now exterior tools could be used. However, to make gathering data session last longer, a change was implemented to allow the enemies to respawn, and the population of NPCs have infinite health. Overall, much time could be saved on implementation if it were redone with knowledge and experience that has been gained.

All MATLAB portions of the project went well, the tool proved invaluable for the displaying and computing of data.

More time would have been ideal for the evaluation portion of the project. It could have examined more metrics and the system tested against multiple types of NPCs. More time to analyse the data would also have been helpful. Due to the aforementioned problems with implementation, progress was much slower than expected. Fortunately, this was planned for by the use of buffers of free time in the Gantt chart. As a result, there was less effect on the project than might have resulted otherwise, though there was definitely a negative

result. Additionally, time was wasted attempting to implement the next stages of a genetic algorithm, before finding out it was not possible. This could have been avoided with better planning and investigation, and it did have a bad effect on the project. If that time were saved and instead used in a wider scope, to create a bigger population of NPCs with more variety and equipment etc., the evaluation would have provided more conclusive results. Nevertheless, it provides proof that this system can work - problems with the game balance were identified and if the system were extended, a much fairer game could result. This system works with most modern games, and so many games could benefit from this. Because of this, the project is considered a success despite the problems.

To further extend the evaluation, user tests and surveys could have been used, by comparing how much users enjoyed an optimised experience versus a default one. Ultimately, the design and carrying out of such a survey would have sapped too many resources and was not the highest priority.

Many of the problems experienced during the implementation period could have been fixed easily. Some of these problems were impossible to know beforehand, while others could have been avoided, so it is fair to say that this project has provided a good learning experience. A system to gather a lot of data was created and that data proved suitable for drawing conclusions about game balance and the strength of certain equipment, which was part of the original goal. While the goal to fully implement a genetic algorithm was not accomplished, the other aims were successful and overall the project can be considered a success.

10. Bibliography

Louis, S., and McDonnell, J. (2004) Learning with case-injected genetic algorithms. *Evolutionary Computation, IEEE Transactions* 8(4) 316-328.

Louis, S., and Miles, C. (2005) Playing to learn: case-injected genetic algorithms for learning to play computer games. *Evolutionary Computation, IEEE Transactions* 9(6) 669-681.

Miles, J. and Tashakkori, R. (2010) Improving Believability Of Simulated Characters. *Journal of Computing Sciences in Colleges,* 25(3) 32-29.

Hernandez, B., Millan, E. and Rudomin, I. (2005) Fragment shaders for agent animation using
finite state machines. *Simulated Modelling Practice and Theory*, 13 741-751.

Zana, G. (2013) *Modeling a Simple AI behavior using a Finite State Machine*. [online] Available from: http://web.archive.org/web/20130203032845/http://blog.manuvra.com/modeling-a-simple-ai-behavior-using-a-finite-state-machine/ [Accessed 30 March 2015].

Back, T., and Schwefel, H. P. (1996) Evolutionary computation: An overview. *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on,* 20-29.

Geisler, B. (2004). Integrated machine learning for behavior modeling in video games. *Challenges in game artificial intelligence: papers from the 2004 AAAI workshop,* 54-62.

Cole, N., Louis, S. J., and Miles, C. (2004) Using a genetic algorithm to tune first-person shooter bots. *Evolutionary Computation, 2004,* 1 139-145.

Bledsoe, W. (1986) I had a dream: AAAI presidential address. *AI Magazine*, *7*(1) 57.

Hingston, P. and Soni, B. (2008) Bots trained to play like a human are more fun. *Neural Networks, 2008 (IEEE World Congress on Computational Intelligence),* 363-369.

Unknown (1969) *Advances in Instrumentation,* 24(4) 691.

Fogel, D. B. (2006) *Evolutionary computation: toward a new philosophy of machine intelligence.* Third Edition. John Wiley & Sons.

Laird, J., and VanLent, M. (2001) Human-level AI's killer application: Interactive computer games. *AI magazine*, 22 (2) 15.

Cook, M. (2015) Electric Dreams, Part 1: The Lost Future Of AI. [online] Rock Paper Shotgun. Available from: http://www.rockpapershotgun.com/2015/02/13/electric-dreams-part-1-the-lost-future-of-ai/ [Accessed 30 March 2015].

Cook, M. (2015) Electric Dreams, Part 4: The Lost Art Of Dreaming. [online] Rock Paper Shotgun. Available from: http://www.rockpapershotgun.com/2015/03/27/electric-dreams-part-4-the-lost-art-of-dreaming/ [Accessed 30 March 2015].

Huerta, D. (2014) What is the average cost of development for video games like Halo, Call of Duty, or Assassin's Creed?. [online] Quora. Available from: http://www.quora.com/What-is-the-average-cost-of-development-for-video-games-like-Halo-Call-of-Duty-or-Assassins-Creed [Accessed 30 March 2015]

Back, T., Hammel, U., and Schwefel, H. P. (1997) Evolutionary computation: Comments on the history and current state. *Evolutionary computation,* 1 (1) 3-17.

Haupt, R., and Haupt, S. (2004) *Practical genetic algorithms*. John Wiley & Sons.

Bryant, B., Miikkulainen, R. and Stanley, K. (2005) Real-time neuroevolution in the NERO video game. *Evolutionary Computation, IEEE Transactions,* 9(6) 653-668.

Miikkulainen, R. (2007) Creating intelligent agents in games. *Frontiers of Engineering:: Reports on Leading-Edge Engineering from the 2006 Symposium,* 15.

Gartner (2013) *Gartner Says Worldwide Video Game Market to Total $93 Billion in 2013.* [online] Stamford: Gartner. Available from: http://www.gartner.com/newsroom/id/2614915 [Accessed 1st April 2015].

Guha, R., Hastings, E. and Stanley, K. (2009) Evolving content in the galactic arms race video game. *Computational Intelligence and Games, 2009* 241-248.

Aha, D., Muñoz-Avila, H., Ponsen, M. and Spronck, P. (2006) Automatically generating game tactics through evolutionary learning. *AI Magazine* 27(3) 75-84.

Zana, G. (2012) *Unstoppable Jake*. [online game] Tel-Aviv, Israel: Manuvra Games. [Accessed 30th March 2015].

Alippi, C., Ribeiro Filho, J., and Treleaven, P. (1994) Genetic-algorithm programming environments. *Computer*, 27(6) 28-43.

Poli, R., Langdon, W. and Mcphee, N. (2008) *A Field Guide To Genetic Programming.* Lulu.com

Aguirre, A., Christiansen, A., and Coello, C. (1997) Automated design of combinational logic circuits using genetic algorithms. *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms* 335-338.

Namessar, (2014) *Counter-Strike Global Offensive Rank System: What is ELO?*. [online] Turkey: Available from: http://namessar.blogspot.co.uk/2014/02/counter-strike-global-offensive-rank.html [Accessed 8th April 2015].

White, A. (2015) *What is Elo and why is it such a big deal in LoL?*. [online] Available from: http://www.gameskinny.com/8mry7/what-is-elo-and-why-is-it-such-a-big-deal-in-lol [Accessed 8th April 2015].

Lotz, M. (2013) *Waterfall vs. Agile: Which is the Right Development Methodology for Your Project?* [online] Segue Technologies. Available from: http://www.seguetech.com/blog/2013/07/05/waterfall-vs-agile-right-development-methodology
[Accessed 7th April 2015].

James, M. (2009) *Scrum Methodology* [online] Scrummethodology. Available from: http://scrummethodology.com/ [Accessed 8th April 2015].

IGN (2012) *Dota 2 - Game Overview.* [online] Available from: http://uk.ign.com/wikis/dota-2/Game_Overview [Accessed 10th April 2015].

Yavuz, A. (2005) *Part 1: Getting Started*. [online] Taleworlds. Available from: http://forums.taleworlds.com/index.php/topic,5408.0.html [Accessed 10th April 2015].

Fincher, J. (2011) *Skyrim devs releasing Creation Engine with new features for the modding community* [online] Gizmag. Available from: http://www.gizmag.com/skyrim-

creation-engine-modding/20737/ [Accessed 10th April 2015].

Creation Kit Wiki (2015) *Papyrus Introduction.* [online] Available from http://www.creationkit.com/Papyrus_Introduction [Accessed 11th April 2015].

Cipscis (2014) *Accessing External Functions and Properties in Papyrus.* [online] New Zealand: Available from: http://www.cipscis.com/skyrim/tutorials/externalaccess.aspx [Accessed 12th April 2015].

Bakkes, S., Spronck, P., and Van Den Herik, J. (2009). Rapid and reliable adaptation of video game AI. *Computational Intelligence and AI in Games*, 1(2) 93-104.

Maple Apps (2015) *How to aim better in CS:GO*. [online] Germany: Maple Apps. Available from: http://csgoskills.com/academy/aiming/ [Accessed 13th April 2015].

Ng, G., Subagdja, B., Tan, A. and Wang, D. (2009). Creating human-like autonomous players in real-time first person shooter computer games. *Proceedings, Twenty-First Annual Conference on Innovative Applications of Artificial Intelligence* 173-178.

Thomaskaira (2012) *Papyrus 101 Class 2: Properties* [online] Available from: http://tesalliance.org/forums/index.php?/topic/5039-class-2-properties/ [Accessed 14th April 2015].

Nesta (2014) *Games industry worth as much as £1.7bn – double previous estimate*. [online] Available from: http://www.nesta.org.uk/news/games-industry-worth-much-ps17bn-double-previous-estimate [Accessed 14th April 2015].

Big Fish Games (2015) *Top Gaming Studios, Schools & Salarie.* [online] Available from: http://www.bigfishgames.com/blog/video-gaming-industry-numbers-by-region/ [Accessed 14th April 2015].

UESP (2014) *Skyrim:Respawning*. [online] Available from: http://www.uesp.net/wiki/Skyrim:Respawn [Accessed 15th April 2015].

ferrebeekeeper (2012) *Draugar - Undead Viking Warriors*. [online] Available from: https://ferrebeekeeper.wordpress.com/2012/10/18/draugar-undead-viking-warriors/ [Accessed 15th April 2015].

SKSE (2015) *Skyrim Script Extender (SKSE)*. [online] Available from: http://skse.silverlock.org/ [Accessed 15th April 2015].

Lakeworks (2009) *File:Scrum process.svg*. [online] Available from: http://upload.wikimedia.org/wikipedia/commons/5/58/Scrum_process.svg [Accessed 15th April 2015].

The Smart Method (2011) *The Waterfall Development Methodology*. [online] Available from: http://learnaccessvba.com/application_development/waterfall_method.htm [Accessed 15th April 2015].

Hicks, K. (2014) *Investigating Using Social Media to Create Compelling Collaboratively Generated Content*. BSc. University of Lincoln.