Exploring How Factors in Evolutionary Algorithms Affect Games AI

Harry Johnson
JOH11209181

BSc (Hons) Computer Science
The University of Lincoln

1st May 2014

Exploring How Factors in Evolutionary Algorithms Affect Games AI

By Harry Johnson

Acknowledgements

Many thanks to my supervisor Prof. Tom Duckett for the support and guidance throughout the entire project.

Thank you to all the staff and students at the University of Lincoln and school of computer science for all the help and support.

Abstract

Evolutionary algorithms solve problems by evolving a population of candidate solutions, it does this by selecting the best individual solutions, breeding them to form a new population and repeating the process over many generations to form a robust and efficient set of final solutions. The project seeks to understand how altering some factors will affect the algorithm's ability to solve problems

The problem in this case, is competing in a simple platform-game, and the population of solutions represent differently optimised game AI. The game rules are simple; one player must jump on the opponent's head, before the opponent jumps on theirs.

Some of the factors being tested include the addition of elite individuals, populations of different sizes, different game modes, and island population, populations that compete against each other but cannot inter-breed.

The results of the experiments show how the problem of optimising evolutionary algorithm is a hard one, and that the effects of altering factors in a virtual environment bear striking resemblance to the effects of changes in ecological environments. One of the most interesting observations made is the apparent predator-prey relationship between two separately evolving populations, where each population takes turns in out-evolving the other.

Contents

List of Tables and Figures

Chapter 1: Introduction

1.1 Overview

The following report documents the research completed for the degree of Computer Science BSc (Hons). The project demonstrates the implementation of an artificial intelligence (AI), created using evolutionary processes, for a simple competitive video-game.

The aim of the project is to theorise on the effects of certain factors on game-AI. Each factor's effect will be measured with a series of specifically optimised evolutionary algorithms.

Evolutionary algorithms explore solutions using survival of the fittest, it searches for the best suited candidate solutions out of a population of other solutions to reproduce and refine over generations. After enough time, the population usually converges on a near ideal solution.

The evolutionary algorithm designed for this project was created to evolve a population of game AI, the game being a simple platform game, where one player must jump on the opponent's head to win. By altering the way that the game evaluates success, and optimising the evolutionary algorithm in different way, the performance of the system changes in interesting ways.

The project explores not only how the changes in simple variables like population sizes affect the way the evolutionary algorithm works, but also how implementing unique processes like elite individuals, and island populations can have their own effects.

The main objective of, and idea behind evolutionary systems is to emulate the extremely adaptive process of evolution and natural selection. Looking at the solutions that the biological world has generated to fit almost every niche, it is clear that its' ability to solve problems is unmatched by even our most advanced technology.

1.2 Research Question

- How do factors in evolutionary algorithms and game programming affect the ability to generate an efficient game AI?

This question will be referenced and kept in mind throughout the span of the project's research, development and analysis stages. The conclusion will attempt to answer this question and the success of the project will depend on the final ability to answer the question.

1.3 Aims and Objectives

1.3.1 Aims

- Implement an evolutionary algorithm to dynamically create a video game AI
- Record how modifying factors affects the AI's effectiveness
- Attempt to draw conclusions on the nature of evolutionary algorithms based on the results gathered.

1.3.2 Objectives

- Research existing games, evolutionary algorithms and forms of game AI.
- Adapt the components to implement a working system.
- Develop a robust evolutionary algorithm with a selection of potential factors and operators to test.
- Develop an experimentation strategy that measures the AI's effectiveness in response to changing factors.
- Conduct a wide range of experiments.
- Analyse how the factors affect each other and the result through comparison and statistical evaluation.

1.4 Definition of Key Terms and Concepts

Term	Definition
Evolutionary Algorithm	Generic term for population-based optimisation algorithm. Uses mechanisms such as reproduction, mutation, recombination and selection by fitness.
Genetic Algorithm	A subset of the evolutionary algorithm, individuals in a population are stored as raw data.
Genetic Programming	A subset of the evolutionary algorithm, individuals in a population are stored as a computer program.
Population	A collection of potential solutions.
Genes	Representation of an individual's traits in a population.
String	The physical representation of a solution as an array of binary digits.
Genotype	Representation of an individual's combined traits in a population.
Crossover	Reproduction method in evolutionary algorithms. Combination of 2 individual solution's genes to create a new solution.
Mutation	Random changes in genes of individual in population.

Fitness	Value representing a solution's success in a population.

1.5 Overview of Chapters

- Chapter 2: Literature Review
    - This section will look into researching existing work and attempting to find factors which may lead to changes in an evolutionary algorithm's performance.
- Chapter 3: Methodology
    - This section outlines which project, software development and experimental methodologies will be carried out to ensure the success of the project.
- Chapter 4: Design and Implementation.
    - The design and implementation of new systems are detailed in this section, as well as the adaptation of existing components.
- Chapter 5: Design of Experiments
    - In this section the experiments are design, reported and analysed.
- Chapter 6: Conclusion
    - A summary of the results and findings are mentioned here.
- Chapter 7: Reflective Analysis
    - This section looks at how the project process went, what could've been improved, and the potential for future work.

Chapter 2: Literature Review

2.1 Introduction

Evolutionary algorithms (EA) were first thought to have been developed by Nils Aall Barricelli in 1954 (Fogel, 2006). Barricelli created simulations to substantiate his alternative theories of evolution; the simulations only used 5 kilobytes of memory each but despite this constraint, they exhibited complex behaviours such as parasitic infiltration, natural disasters and stagnation (Paul Prudence, 2012).

Lawerence J. Fogel was another pioneer in EA, his dissertation titled “On the Organisation of Intellect” was the basis for the book “Artificial Intelligence through Simulated Evolution”, which was one of the first in the field of evolutionary computing.

Evolutionary algorithms are often computationally intensive. In the past this factor has limited the potential to solve real world problems but as technology has improved, the horizon for EA solutions has widened.

Sean Luke (1998) used EA techniques as a tool to teach robots how to play soccer. The ‘softbots’ had access to certain senses such as ball position and goal position; they also had a set of commands like kicking a ball or yelling a message. By corresponding senses with commands, behaviours were formed.

Past research indicates that users of EA techniques often encounter optimisation problems. This is often due to the large amount of factors that go into most genetic programs. Games, being rule-based, have a multitude of factors within them as well; this presents a further level of complexity for genetic programs which seek to optimise game-playing techniques.

2.2 Motivation & Justification

Evolutionary algorithms are a powerful tool for finding solutions in uncertain problem domain such as quantum physics, space anomalies, biology, electronic design, game strategies and psychology. Between 1967 and 1989, Goldberg (1987) documented 82 different applications of genetic programming across 11 fields.

Bill Gates (2014) claims that “the ultimate is computers that learn”, when asked about the most promising field in computer science. Deep learning is a set of algorithms in machine learning that aims to give technology the tools to learn. EA is closely tied to machine learning and advances in either field may benefit the other.

Lee Spector et al (1999) applied evolutionary algorithm methods to quantum computing to evolve quantum algorithms and push the field forward in other areas. EA excels in the quantum environment due to its ability to solve black-box problems.

Natural genetic systems have also been modelled through the use of EA. The understanding of DNA and the way that it influences nature is a large area for interest; cancer-based gene expression has been classified through the use of EA by Jin-Hyuk Hong *et al* (2006).

Gargolinski (2005) describes the issues of implementing EA into game AI. EA's use in games is described as "computationally expensive" but it is becoming easier to deal with as hardware speeds up, Gargolinski also states that the processing can be done offline however, which would circumvent the issue of hardware limitation.

Despite Gargolinski's doubts, one of the earliest examples of evolutionary methods in game AI is the use of learning agents in the video-game Quake 3 (id Software, 1999). A combination of the Q-learning algorithm and the genetic algorithm were used in the implementation of the system (Bonse *et al,* 2004). Despite the difficulty of the task, it was found that learned agents have almost double the chance of winning versus preprogramed opponents. The findings of the research display the potential for evolutionary methods to be used in AI design; Bonse *et al* (2004) stated that the experiment depended on choosing the right input vector and settings as well as the used environment to create the ideal AI, suggesting the importance of optimisation.

Whilst there exists many examples of EA, optimising it for specific uses is less understood, this is due to the large amount of possible applications of EA and the amount of possible configurations. Gargolinski (*op cit*) explains that tweaks/optimisations will most likely be very problem specific, when referring to the use of EA in games. The project aims to understand why the algorithm reacts differently to certain problems, instead of simply finding the most efficient solution. With this information, more organic and intelligent AI could be created, and due to the robustness of EA, it is likely that solutions found in a game environment could be carried over into other areas.

2.3 Factors

2.3.1 The Issue of Optimisation

In order to attempt to answer the research question, many different optimisations of the game and evolutionary algorithm will need to be implemented. Due to the vast amount of factors, testing every one of them will be unrealistic. The factors tested will be impossible to test completely. This is because some areas in the fitness landscape will require extremely specific optimisations of factors to reach, and only a few optimisations will be selected.

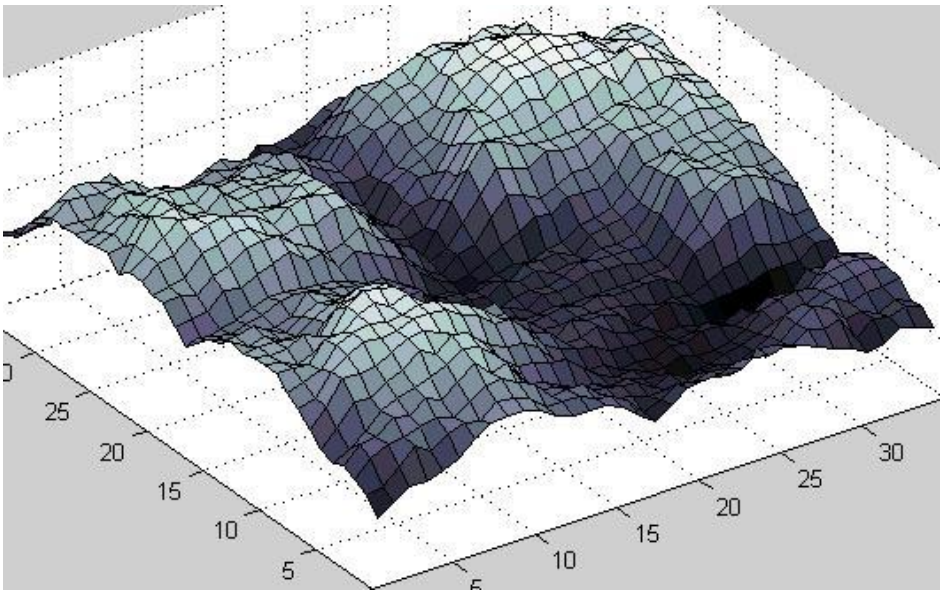

Figure 1 (Elena Popovici, 2003)

To explain the theory behind the fitness landscape, figure 1 shows an example, where the x and y values represent two factors and the point at which they intersect represents that optimisation's fitness. Altering just one x value or y value in the graph won't always lead to the highest peak (highest fitness), it requires an optimisation of both values. The complexity in finding the highest peak increases as the ways to traverse the fitness landscape increases; these are the factors in game design and the evolutionary algorithm.

Fogel (1994) explores the process of optimisation in the paper "An Introduction to Simulated Evolutionary Optimization", it concludes that in order to advance the GP field, the natural process of evolution must be closely observed and abstracted. To highlight how evolutionary algorithms emulate natural evolution, Fogel (1994) explains how stagnation in a population's fitness can occur when the environment is static and Eiben *et al* (2003) explains how incest prevention positively affects computer-simulated evolution. So perhaps, In order to match the level of optimisation that natural evolution accomplishes, EA should follow the natural world's method as closely as possible.

However, complete optimisation shouldn't be the goal of an EA project, Goldberg (1989) states "we can only strive to improve", this is due to the complexity that EA problems have in their solutions. Because of the complexity and randomness, complete confidence in the algorithm is impossible, although by running the program with different starting parameters, confidence can be increased (Lynch, 2006).

2.3.2 Factors in Game Design

2.3.2.1 Environment

To create a truly robust AI, it should be able to perform successfully in different environments, to improve it further; it should work in a dynamic environment. Looking back at Fogel's (1998) point about a static environment stagnating a population, it suggests that not only does changing a population's environment make them more robust, it can actually nudge the population to work better in its original environment.

The main issue with evolutionary adapted AI is that it can ignore its environment entirely. It forms behaviours that work in certain environments but it does not actually store information about its surroundings. The result of this is that the AI relies heavily on the environment in order to function; Bonse *et al* (2004) encountered this problem when evolving AI for Quake 3 (id Software, 1999).

Bonse's problem was that, in a square arena, AI would often get stuck in corners; they would also do nothing if they could not see the enemy immediately, Bonse solved this problem by implementing an arena without any corners. It is possible to avoid the pitfalls of the environment if the population was given more time to evolve, as shown in Graham's (2003) research. Bonse's AI was created using a neural network-type design, neural networks can be prone to searching unrelated solutions, which may have meant the chance of finding the solution to the pitfalls in the environment was too low to occur, combined with the fact that Bonse was unable to speed up the game time, it was unlikely he would have been able to find the solution.

2.3.2.2 Game Rules

The rules that determine which player wins will greatly affect the behaviour of an evolved AI, This is because winning or losing is usually the main variable that determines fitness. Behaviours evolved in games with altered rules could positively affect an AI's fitness with normal settings. For example, if the rules were changed so that the AI had to beat an opponent 3 consecutive times instead of just once, it could influence it to behave more carefully instead of opting for a 'kamikaze' strategy.

Altering the game rules may also be an efficient method of directing the evolutionary algorithm to increase an AI's fitness is a specific area. Geijtenbeek et al (2013) evolved virtual bipedal creatures to walk across different terrain, in different directions, and through obstacles, the AI was first evolved to walk in a straight line before optimising it for the advanced experiments, which may have saved time evolving.

2.3.3 Factors in Evolutionary Algorithms

2.3.3.1 Initial Population

The initial Population can drastically alter the effectiveness of an evolutionary algorithm; this is due to the balance between selecting a population size that's large enough to accommodate many solutions whilst being small enough to use an acceptable amount of processing power (Diaz-Gomez et al, 2007).

Storn *et al* (1997) used a method of generating the initial population by applying a uniform distribution of random individuals. Storn also mentions that when a nominal solution is available; it can be used to generate an initial population by using normally distributed random deviations from the solution.

2.3.3.2 Population Size

Different population sizes can affect an algorithm depending on many other factors including the problem domain, the resources available and time constraints. A Non-finite population size is also an option and may result in population with a higher fitness, Segura *et al* (2013) claims a phenomenon called genetic drift can occur in finite population sizes where new individuals, superior to their parents, aren't generated.

Fixed population sizes can be structured in a number of ways to increase robustness and efficiency. Preselection, the replacement of existing individuals is a method that Cavicchio (1970) first implemented in order to encourage niches. By replacing an inferior parent, a new individual is able to maintain a healthy and varied population. Cavicchio claimed that this technique was most useful in maintaining diversity in small populations. Preselection, the replacement of existing individuals is a method that Cavicchio (1970) first implemented in order to encourage niches. By replacing an inferior parent, a new individual is able to maintain a healthy and varied population. Cavicchio claimed that this technique was most useful in maintaining diversity in small populations.

2.3.3.3 Genotype Size

Similarly to the population size, the genotype length can be altered too. Increasing the genotype allows more information to be stored, if the information is useless however, it can decrease the individual's fitness. Increasing the genotype can add more complexity, in AI, extra or nested loops can be added with a longer genotype, extra complexity in AI behaviour could result in more specialised or robust behaviour. A higher genotype length, however, will take longer to process

(as there will be more data to decode), so processing and time limitations will need to be taken into account.

Harvey (1992) claims that variable length genotypes must be used and unrestricted in problems where the problem space is not pre-defined. It is stated by Harvey that using a variable length genotype allows a potentially infinite number of solutions to be explored; the only restriction is the time and processing power available. The game AI environment could work well with variable length genotypes because of the large amount of possible behaviours, however research is lacking on the effects of different genotype length on game AI.

Manos *et al* (2006) was able to improve the design of micro structured optical fibre by implementing variable-length genotypes. Manos claims that variable genotype lengths allowed for superior, non-symmetrical designs and also that the algorithm gained the ability to change various attributes of the type of structure evolved. More complexity is added when variable length genotypes are implemented because increasing or decreasing the genotype length alters the number of ways that they can be combined; it also introduces a new problem of what to do when combining genotypes of different lengths.

Altenberg (1994) designed a method of incrementally increasing the genome size which aims to better represent biological evolution, the method works by only accepting added genes that increase the fitness of an individual. The method showed success in the NK model that Altenberg used to test the theory, similar results may arise in a game AI environment.

2.3.3.4 Fitness Evaluation

Fitness is calculated by testing a solutions success. In multiplayer games, the success of an individual relies on their ability to defeat an opponent. It is unrealistic and unreliable to test each solution's viability against a human opponent due to time-constraints and human error, therefore methods such as tournaments and pre-programmed opponents can be used to evaluate fitness against.

Belal Al-Khateeb *et al* (2009) tested which tournament method was most efficient for evolving checkers AI. The results of a round-robin selection method were compared to individual and social learning algorithms to distinguish which generates a better AI. The round robin method was found to be the most efficient in enhancing the evolutionary checkers algorithm, it is unclear however, if round-robin would enhance fitness similarly in different game-styles.

Conversely, Sipper *et al* (2006) used an elimination tournament method in which individuals were repeatedly paired up against other randomly selected individuals to battle up to 1000 consecutive

times (depending on the game), after each pair fought, the loser would be removed from the tournament and the process would then continue until one remained.

It would seem as though a round robin tournament would be more effective at fairly determining fitness because each individual's skills would always be measured up against each other. The elimination tournament method could be exploited by an individual if they happened to match up against opponents that they played best against. However, in a population of the same size, a round robin tournament will always require more matches than an elimination tournament, as a result, in large populations where processing time is a major constraint, an elimination tournament may be more appropriate. Also, as the elimination tournament method should in theory take less time, more matches can be played between matches pairs, which should result in a more reliable outcome.

Round robin, elimination and contest tournaments were compared and analysed by Dmitry Ryvkin (2008). It was found that the population's size, distribution of skills and noise level should be taken into account when choosing a tournament method as its efficiency relies directly on the type of population it is testing.

Dmitry's (2008) implemented contest tournament method paired each individual against the same opponent once; in a game environment, fitness can only be calculated by pairing two players against each other. In order to get fair results using the contest method, the same opponent will need to be played against each individual. In Amit Benbassat's *et al* (2011) research into evolving board-game players, one method partnered evolving AI's against randomly playing opponents, the other against simultaneously-evolving AI's. The same methods of fitness calculation were used in Tiago Francisco's *et al* (2008) 2D spaceship genetic program. Unlike Benbassat *et al*, Francisco *et al* would like to improve the Random Opponent Evolution approach; to get better human competitive results.

Besides human-controlled and the aforementioned random and simultaneously evolving opponents, pre-programmed is the only other viable opponent type, due to the immense number of fitness calculations required, human-controlled opponents would be too time-consuming. Pairing evolving AI's against pre-programmed opponents has proven its success in a number of projects already (Sipper, 2011; Bonse, 2004; Hui, 2011).

Implementation of several, separately evolving AI's is also possible and may have advantages. Sean Luke (1998) used this method; the result was AI's developing different behaviours within a

group. In the context of team based games, players taking different roles increase the fitness for the entire team; the potential for an increase in individual fitness is unclear however.

Implementation of a fitness sharing function could help a population avoid premature convergence by encouraging specialisation. Goldberg and Richardson's (1987) implementation of a sharing function worked by dividing an individual's fitness by the amount of individuals it was sharing with, the neighbourhood and degree of sharing was determined by an individual's similarity to others.

2.3.3.5 Multi-Objective Fitness Evaluation

In a single-criterion problem, a population's fitness is improved by seeking to increase/decrease a single objective utility/cost. Multi-objective optimisation is the problem that occurs when many fitness identifiers are available, but they are incomparable to each other (Goldberg, 1989).

In single-criterion problems, the optimal solution can be difficult to find, for multi-criterion problems; a single-criterion approach won't encapsulate the entire problem and so won't return a robust and efficient solution. In games, there can be many utilities/costs; maximising fitness doesn't always just mean the maximisation of a single objective, multi-criterion optimisation algorithms can be used to find a balance between the objectives and form an optimal strategy (Michael Benisch, 2006).

Coello *et al* (1999) reviewed many techniques to calculate fitness in multi-criterion problems. The weighted sum approach represents the fitness measurement for each individual as the sum of each objective function, each using a different weighting coefficient. Coello claims the main strength of this method is in efficiency, however, the method does not provide a way in which to determine the objective weighting and so it must be estimated or calculated physically.

Goal programming and goal attainment are methods that use predetermined goals to steer the objective weighting optimisation. Due to the heavy reliance on decision maker intervention, the algorithm can only be as effective as the goals set. The algorithm is also more likely to fail due to human error; a more autonomous method would counteract problems in human judgement (Coello *et al*, 1999).

VEGA (Vector Evaluated Genetic Algorithm) is a method developed by David Schaffer (1984) which involves evaluating and selecting each objective function separately and then combining and shuffling the subpopulations to form the next generation. The danger of using this technique is that it heavily favours speciation, individuals that are excel in many areas won't be evenly judged against another that is proficient in just one objective (Ghosh *et al*, 2004).

Mumford-Valenzuela (2005) developed SEAMO (a Simple Evolutionary Algorithm for Multi-objection Optimisation), it improves on VEGA by performing a domination check to retain non-dominated (non-specialised) solutions and a niching mechanism to exterminate closely packed solutions. Unlike goal programming, it does not require intense human-intervention which makes it a good candidate method for testing.

2.3.3.6 Selection

The process of choosing which solutions to generate the next population of solutions is called selection. The standard implementation of selection uses the fitness value of each individual to determine the likelihood that their genes will make it into the next generation. Fitness scaling and elitism alter the way that fitness determines the chance of an individual's selection; on the other hand, mating techniques such as positive assortative mating and crowding use genotype similarity to affect an individual's chance of selection.

Fitness scaling is a technique that involves the resorting of evaluated individuals in a population. The technique is used in genetic algorithms because there can be issues moving between multiple plateaus of solutions that have similar fitness, fitness scaling overcomes the issue by rewarding the more fit solutions to a high precision (Sadjadi, 2004). Fitness scaling can help avoid premature convergence (Goldberg, 1989); it can also help in problems where rank is the only worthwhile identifier of fitness (Kreinovich, 1993).

Mathworks (2014) implemented two fitness scaling functions into their genetic algorithm, rank scaling and top scaling. Rank scaling works by first sorting the population according to each individual's raw fitness, the actual fitness of each individual is determined by their sorted position. Rank scaling can run with different scaling procedures, linear scaling attempts to keep the average scaled fitness equal to the average raw fitness (Goldberg, 1989), exponential scaling is a technique commonly used in robots, power law scaling is often used in machine vision applications (Kreinovich, 1993), and sigma scaling attempts to balance between maintaining wild variations in early generations and similar members in later generations (Gargolinski, 2005).

The top scaling implementation by Mathworks (2014) is a form of elitism, working by only selecting the fittest 40% of a population. Implementing an elitist selection method sacrifices a global perspective for improved local search (Goldberg, 1989). Fernando (2005) claims that elitism can positively affect the search for better solutions when mutation is likely to produce a low fitness solution. Steady state selection works similarly to elitism, except instead of protecting the fittest members of a population, steady state selection always removes the least fit members of a population (Gargolinski, 2005).

Specialisation can be induced into a population in a number of ways, positive assortative mating is a popular technique, which is widespread in the natural world. By implementing a constraint where individuals in a population favour mating with individuals with similar genotypes, self-maintained groups appear to form with their own talents and niches. Negative assortative mating works in the opposite was as positive assortative mating, instead of preferring closely related individuals, individuals try to mate with others that are most different to them, Fernandes *et al* (2001) claims that negative assortative mating results in a population that is less likely to become trapped in local optima.

Crowding is a method that combines positive assortative mating and preselection (see population size); in crowding, new individuals will replace others in a random subpopulation, which is decided by to genetic similarity and fitness (De Jong, 1975).

2.3.3.7 Crossover and Mutation

Crossover and mutation are the techniques that spawn the next generation of solutions from the fittest individuals of the last. Crossover uses building blocks in existing solutions to create new ones; mutation gives the algorithm the ability to think outside the box by harnessing the power of randomness. The classic method of crossover is completed by splicing two existing solutions together at a randomly selected cross point to create two new solutions. The most common way of implementing mutation is done by including the chance that a random bit inside the string that represents a solution will invert (Goldberg, 1989).

Some alternative ways of crossing and mutating solutions can be done through displacement mutation, insertion mutation, inversion mutation, and displaced inversion mutation. Displacement mutation inserts a random section of a solution's string into a random location in the partner's string. Insertion mutation works in the same way as displacement mutation but only at the gene level. Inversion mutation reverses a random string portion and displaced inversion mutation works similarly to displacement mutation, except it also reverses the displaced string portion (Gargolinski, 2005).

2.4 Control Factors

In order to test the effects of each factor, a control group will need to be created to compare the results to. Lynch (2006) theorises that to successfully implement changes to the genetic algorithm, it is best to start with a robust system, and then slowly modify it. It should therefore be a priority to attempt to create a basic, robust system as the control group. The control mutation rate, population size, and other variables will be decided by optimising the evolutionary algorithm to generate a successful opponent, within an acceptable timeframe.

The initial optimisation will be decided by looking at similar systems for a rough guide. Sean Luke (1998) and Sipper *et al* (2006) both implemented evolutionary algorithms to create AI for game techniques using a decision tree structure; therefore their optimisations should be well suited for a similar artefact. Gargolinski (2005) also stated a rough guide of what optimisation parameters to use for an initial evolutionary algorithm in game AI.

	Population Size	Mutation Rate	Crossover Rate
Sean Luke	128	0.3	0.7
Sipper *et al* (Backgammon)	128	0.25	0.65
Sipper *et al* (Chess)	80	0.15	0.5
Sipper *et al* (Robocode)	256	0.05	0.95
Gargolinksi	150-250	0.01	0.7

Figure 2

Figure 2 displays the suggested or implemented optimisations for the previously mentioned projects. Sean Luke's generation number was excluded as his project was under strict time constraints and as a result, the generation number had to be reduced. To produce an initial optimisation for the artefact, the average will be taken from the data for each variable (the middle value will be used for ranges), rounded to within 2 decimal points. The result of this can be seen in figure 3.

Population Size	Mutation Rate	Crossover Rate
158	0.15	0.7

Figure 3

2.5 Population Analysis

Various details about the state of the population at each generation will need to be measured. Details regarding the fitness will tell how successful the population is at different stages, measuring the number of shared schema will express how similar a population is and which building blocks are most effective, it may also help identify if and at which point the population converged.

Gustafson (2004) mentions two methods of measuring diversity, the number of unique genotypes (Koza, 1992; Langdon, 1998) and the number of unique fitness values (Rosca, 1995). The measurement of diversity in a population method is important to look out for as a lack of diversity is the main cause of premature convergence (Leung, 1997).

Once the population has been measured at each generation, values such a maximum and average fitness can be compared visually using graphs, as shown in an example graph in figure 4.

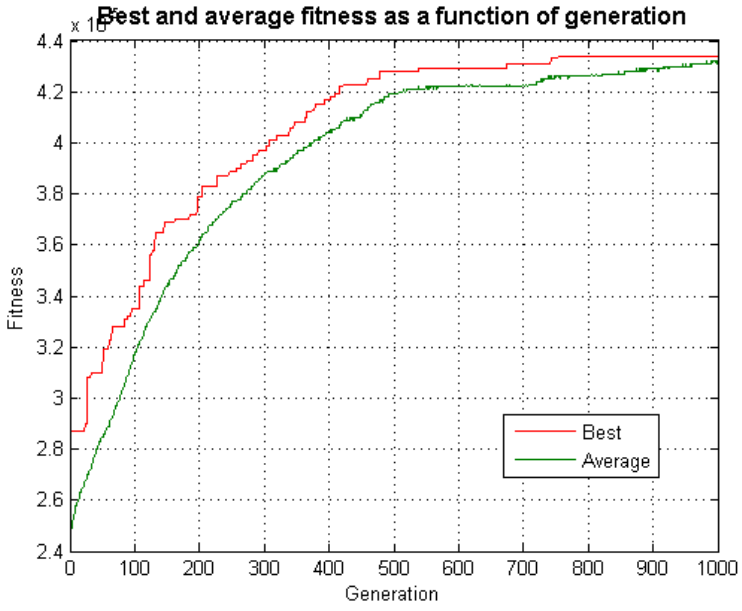

Figure 4

The final, most fit individual in each testing category will be tested and compared. The final population will be the end result of the genetic algorithm and will therefore be used to determine if the optimisation was successful or not.

2.6 Conclusion

Factors within the evolutionary algorithm such as population size and genotype length have limitless potential configurations and as a result, the number of combined possible configurations is even higher. When extra implementation techniques such as fitness sharing (Goldberg *et al*, 1987) or crowding (De Jong, 1975) are added into the algorithm, the ways in which it can be optimised increases even higher. The literature review process has highlighted how the act of optimising evolutionary algorithms is an incredibly complicated process and as a result, optimisations may only be unique to the problems they were designed for.

However, it is also clear that some definite, or at least, reliable truths exist for evolutionary algorithms. For example, the need for diversity for a fit population and the methods that can enhance the probability of a diverse population, such as fitness scaling (Goldberg, 1989), fitness sharing (Goldberg *et al*, 1987), preselection (Cavicchio, 1970), positive/negative assortative mating (Fernandes *et al*, 2001), or crowding (De Jong, 1975). Also, it was found that complete confidence in a solution is impossible, evolutionary algorithms should strive to improve, but not

for perfection (Goldberg, 1989). It has also been found that the most effective methods used in evolutionary algorithms are the ones that are also used in nature (Fogel, 1994).

Additionally, the literature review process has helped in the selection of the game to generate AI for, the type of evolutionary algorithm to implement, which factors to test, and the parameters to initialise the evolutionary algorithm with.

Chapter 3: Methodology

3.1 Project Management

Turbit (2004) says that a project management methodology breaks down the project into phases, with plans in place before each phase begins; it also defines roles and responsibilities, and the budget. In order to select an appropriate project management methodology, the nature of the project should first be well understood.

The aims of the project are to implement an evolutionary algorithm to dynamically create a video game AI, to record how modifying factors affects the AI's effectiveness, and to attempt to draw conclusions on the nature of evolutionary algorithms based on the results gathered. From this, we can gather that the large percentage of time spent will be on experimentation and analysis, therefore a project management methodology that caters experiments should be most suitable.

The requirements of the chosen methodology also depend on what resources and limitations the project depends on. In terms of costs, time will be the most valuable resource, as there is a strict deadline for completion; therefore a methodology that favours timekeeping will be highly regarded. The largest drain on time resources for evolutionary algorithm projects is computation time, for example, Sean Luke's (1998) project was first estimated to take over a year to find a solution. Processing time should not be an issue however, as the University of Lincoln allows powerful computer resources to be accessible to all computer science students. The project is relatively small-scale and so, human and monetary resources are not very significant. Finally, the project scope could potentially be very large, given that so many factors alter evolutionary algorithms, consequently the chosen methodology will be required to intelligently select which factors to experiment.

With the aforementioned points in mind, the "design of experiments" project management methodology was decided to be most suitable. Design of experiments (DoE) works by investigating how the controlled variation of inputs affects the output from a process (Souza *et al*, 2013); this fits well with the aims of the project.

DoE's applicability to the project can be noted further by looking at the objectives of DoE. Park (2007, p. 309) outlines the objectives of DoE as the following:

1. "To measure or evaluate characteristic values without analysing them statistically.

2. Identification through experimentation of factors that are significant to the responses and determination of how large their impact is. The results of experimentation are analysed statically.
3. Statistical identification of factors with little influence.
4. Determination of the values of significant factors. Then, the optimum condition can be found."

Park (2007) also provided a diagram of the processes to be carried out by a DoE project, which can be seen in figure 5.

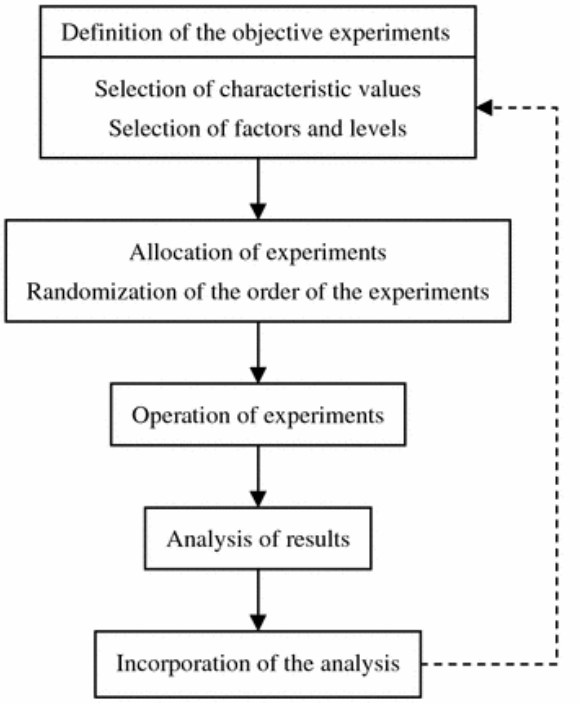

Figure 5 (Park, 2007)

The iterative component of the DoE process can be seen where the results are incorporated into the analysis to help in the selection of the next set of experiments. This ability to dynamically adjust, may allow the project to intelligently select appropriate experiments.

3.2 Software Development

Whilst the DoE project methodology will allow for the intelligent selection of experiments, it does not outline the actual process for the design, or implementation of experiments, evolutionary algorithm, game coding, or game AI. All of these components are vital to the success of the project.

In order to choose a software development methodology that meets the needs of the project, the needs and nature of the project should be well defined, so it may help to look back again to the projects' aims.

- Implement an evolutionary algorithm to dynamically create a video game AI
- Record how modifying factors affects the AI's effectiveness
- Attempt to draw conclusions on the nature of evolutionary algorithms based on the results gathered.

The technology for the implementation of a genetic algorithm, game and game AI already exist and are well documented. Therefore, as it will not affect the results of the experiments which will attempt to answer the research question, it would be most appropriate to use existing software, if possible.

The component-based approach is therefore a natural choice as Crnkovic *et al* (2006) defines the approach as the building of systems from already existing components. Taking on the component-based approach however will mean accepting several consequences for the system lifecycle, as Crnkovic *et al* (2006, p. 2) explains:

- "Separation of development processes. The development processes of component-based systems are separated from development processes of the components; the components should already have been developed and possibly have been used in other products when the system development process starts.
- A new process: Component Assessment. A new, possibly separated process, finding and evaluating the components appears. Component assessment (finding and evaluation) can be a part of the main process, but many advantages are gained if the process is performed separately. The result of the process is a repository of components that includes components' specifications, descriptions, documented tests, and the executable components themselves.
- Changes in the activities in the development processes. The activities in the component-based development processes are different from the activities in non-component-based approach. For the system-level process, the emphasis will be on finding the proper components and verifying them. For the component-level process, design for reuse will be the main concern."

In context to the project, the consequences mean that each component will need to have been developed beforehand through a new process of component assessment; also, there will be

different processes for the overall system and for the individual components. The component-based lifecycle can be seen in the diagram by Crnkovic *et al* (2006) in figure 6.

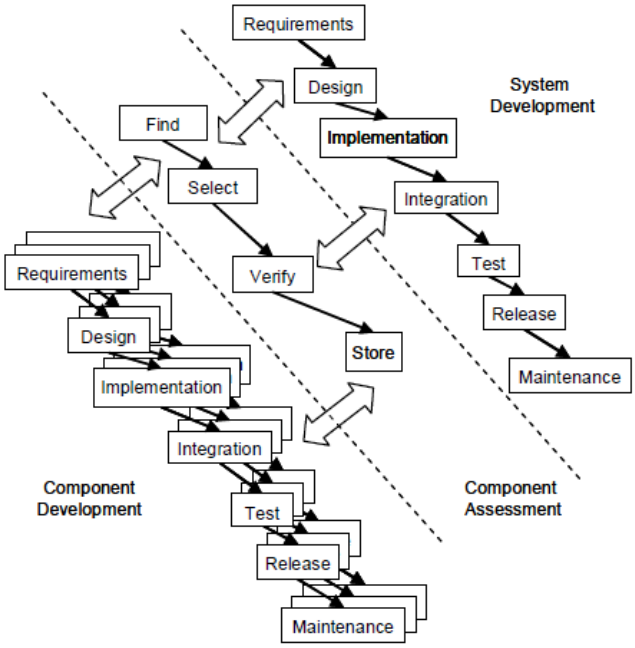

Figure 6 (Crnkovic, 2006)

Some of the benefits to using a component-based approach are the decrease of production costs and shorter completion time (Lau, 2007). Lau also states how the component-based approach relies on each component's ability to easily be reused, in other words, there's no use using an existing component if it takes longer to adapt it, than it is to create the component itself. Theref0re, if a component isn't adaptable, it may be more worthwhile to create the individual component.

From the literature review process, it is clear that the software for each component already exists or the implementation is highly documented. Therefore, in the case that a hybrid approach is needed for the construction or adaption of one or more of the components, the design process should be straightforward as it would only involve recreating an existing component in a more adaptive form.

Given the straightforward nature of any components needing to be specifically implemented, a waterfall approach would seem to be most suitable. Khan *et al* (2011) explains that the incremental model is an evolution of the waterfall method however, which may suggest that it is an improvement on waterfall. Also, Laplante (2004) makes a case against the waterfall method, claiming that it is an outmoded practise; Laplante claims that the waterfall method is usually unsuitable because flaws in the initial assumptions become more realised as more information

about the problem is gathered. The incremental approach outlined by Khan *et al* (2011) works off the same processes as waterfall, but with added flexibility, which makes debugging easier, changes simpler to implement, and risks handled better. The processes within Khan's *et al* (2011) model can be seen in figure 7.

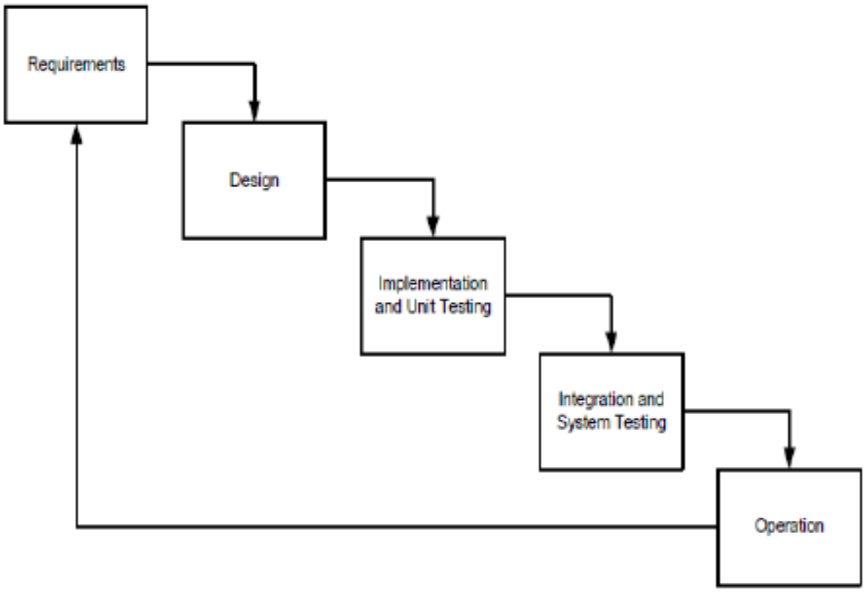

Figure 7 (Khan *et al*, 2011)

One of the main disadvantages outlined by Khan *et al* (2011) is that clients must learn a how to use the new system after each iteration, which won't be a problem for the project due to the lack of clients. Khan also stated there may be difficulty integrating additional functions after each increment without degrading past work, but with a clearly defined program structure, code will rarely be rewritten. Additionally, it was stated that in projects where the requirements aren't fully gathered or the planning isn't well done, problems can arise, however as the planning and requirements are fully understood, these disadvantages shouldn't arise.

As the incremental approach appears to have all the advantages of the waterfall method, but with added flexibility and a low chance of problems, it appears to be most suitable to apply to the project with the component-based approach.

3.3 Toolsets and Machine Environments

Because the project depends on each component's ability to work together and also because the evolutionary algorithm will need to process quickly to create a solution in a reasonable time, the computation speed for the components will be a highly deciding factor. Because of the emphasis on speed, it will be preferential to use the same programming language for each component.

3.3.2 Game

3.3.2.1 Requirements

In order to select an appropriate game to apply EA methods to, the requirements and limitations of the project should be clearly understood. Due to time constraints, simplicity should be a priority.

Considering that AI alone can be intensive to compute, it should be taken into account that evolutionary algorithms must simulate fitness up to hundreds of times for each generation, which can lead into the hundreds itself, for this reason, A game with high computation speed is a requirement. A reflection of the scale in EA programs is shown in Figure 8 which exhibits the control population size and generation count from Sipper *et al*'s (2006) research.

	Backgammon	Chess	Robocode
Population Size	128	80	256
Generation Count	500	150-250	100-200

Figure 8

Thought must be put into the programming language and optimisation of the game; these factors contribute to the game's overall processing speed.

There should also be powers to speed up the game's clock, the quicker a game runs, means more fitness tests can be calculated, which can allow for testing larger population sizes and generations. The adaptability of the game will contribute towards this.

Finally, only multiplayer games should be considered in order to test different tournament modes. Candidate games must determine fitness by competing two individuals or more against one-another.

3.3.2.2 Game Genre

Figure 9 displays some of the games that were considered to apply evolutionary programming methods to.

Genre	Description	Plausibility
Ball and paddle	Each player controls a paddle which can be moved across a 2D plane. Players score points by deflecting a physics-based ball behind their opponent's paddle.	Very simple to implement. An optimum AI can be easily achieved and there is a lack of possible strategies.
Platform game	Each player controls a character capable of moving and jumping across a 2D environment.	Simple to implement. Not a lot of potential for AI behaviours but game genre is very adaptable, so presents a

	A player wins by jumping on the top of an opponent.	lot of possible experiments.
Fighting game	Each player controls a character capable of moving and jumping across a 2D environment. Players can choose to attack dealing damage to opponents nearby. An opponent must be attacked multiple times to defeat them.	Slight variation on platform game genre. Similarly simple to implement. Also has a lack of potential strategies but has potential for different adaptations.
Multi-directional shooter	Each player controls a character capable of driving, turning and shooting. Each player must dodge obstacles and shoot the opponent to win.	Simple to implement. Room for multiple strategies to work.
First-person shooter	The player controls a character from their point of view, in a 3D environment. Players must shoot one another with weapons in order to win.	Somewhat difficult to implement. Would present a variety of experimentation options if done correctly however.
Real-time strategy	Players control a variety of different characters from a top-down perspective. Players must maintain an economy whilst using their forces to defeat the enemy.	Very difficult to implement. Large amount of potential strategies to flourish, lots of room for AI to grow.

Figure 9

Due to the simplicity, a 2D platform game was considered most appropriate and a safe decision. The genre is complex enough to have the potential for multiple behaviours to work but also, it is simple enough to analyse the AI. In one of the more complex games, comparing AI's effectiveness would be a difficult process. The genre is also very fast paced, which would allow for many fitness evaluations.

The 2D platform game's simplicity also means the game rules can be changed easily, which could give further insight into any developed AI's adaptability.

3.3.2.3 Existing Games

Figure 10 presents some existing 2D platform game implementations.

Game	Language	Description	Plausibility
"Simple platformer engine" (Jon, 2010)	Python	The aim is to move the players' position to a goal position.	Very simple and small. The code is very basic and would require a small amount of alteration to work with multiple players. Only requires the pygame library to run.
Smash Battle (Jeroen Groeneweg, 2011)	C++	Up to four players can fight with a variety of different weapons,	Requires a lot of prerequisite libraries in order to function. Lots of built in functions and game

		across different arenas.	modifiers add potential for experiments. Code is somewhat complex.
SuperTux (Bill Kendrick, 2013)	C++	The player must traverse a landscape full of enemy creatures to reach the end.	Code is somewhat complex and requires some extra libraries. Would require alteration to work with multiple players. Extensive documentation.

Figure 10

3.3.2.4 Conclusion

Jon's (2010) "Simple platformer engine" was selected as most suitable. The code is very easy to understand and is very fast. This game was chosen, despite the alteration needed, because simplicity was a priority. Inspection of each game's code uncovered that Smash Battle and SuperTux would have required extensive research into each game's documentation to understand, Jon's game is understandable almost immediately.

Despite this, the chosen game component will depend on the available evolutionary algorithm components. The most appropriate game will be the one that adapts best to the available components.

3.3.3 Evolutionary Algorithm

Within evolutionary algorithms, several subsets and alternative implementations exist, the most popular being genetic programming (GP) and genetic algorithms (GA). Whilst both techniques are appropriate for use in game AI, due to the way that each of them work, they each have different levels of complexity and robustness in certain scenarios.

The main difference between GA and GP is that GA seeks to evolve a set of solutions, whereas GP seeks to evolve a function to create a solution. In the context of AI, GA would seek to optimise and alter variables within an AI framework; GP would seek to create the most optimal AI framework itself. From this point of view, genetic programming appears to offer much more flexibility and unique solutions.

Sah *et al* (2008) compared GP and GA in their abilities to generate photo-mosaic images. The findings of the research suggest that GP was more successful than GA; GP's success is credited to its ability to narrow the search space down to the area of best possible solutions. GP was able to do this because of its development flexibility and ability to experiment; this suggests that GP would excel in creating more organic and unpredictable strategies in a game environment. Sah's findings go against the original hypothesis that GA would outperform GP; Sah explains that GA's failure was unexpected and might have been a result of using a two-point crossover method, GA

was thought to be more suitable for the problem due it being a fixed-length problem (well understood).

GA's are more useful in situations where the problem is already somewhat understood. Game AI is a well-researched and documented area, it may be better to use existing AI frameworks if the existing model just needs to be optimised. The conditions for success in video game strategies are also inherently understood, due to the rule-based nature of games.

Regarding the complexity of implementing each method, GA will require extra effort to create the AI framework, GP will simply evolve it. GP however requires more complex crossover and mutation processes, it can also require more time before solutions start to appear, Sah (2006) found that GA's fitness was generally initially higher than GP's until it is eventually overtaken. With all the advantages and disadvantages in mind, implementing a system using genetic algorithm methods would be more appropriate, due to the understood problem domain and simplicity required.

3.3.3.1 Existing Evolutionary Algorithms

Figure 11 presents some existing evolutionary algorithm implementations.

Name	EO (Geneura Team, 2014)	ECJ Sean Luke *et al*, 2014)	MOEA (Dave Hadka, 2014)	JGAP (Klaus Meffert and Neil Rotstan, 2014)	Inspyred (Python community, 2014)	Open Beagle (Christian Gagn'e and Marc Parizeau, 2007)	Watchmaker Framework (Daniel Dyer, 2013)	DEAP (François-Michel De Rainville *et al*, 2012)
Language	C++	Java	Java	Java	Python	C++	Java	Python
Evolutionary Algorithm	GA	GA, GP	GA, GP	GA, GP	GA	GA, GP	GA	GA, GP
Variable Population Length	X	X						
Variable Genotype Length	X	X						
Tournament Methods	Elimination	Round-Robin, Elimination	Elimination	Round-Robin	Elimination		Elimination	Elimination
Island Populations		X	X		X	X	X	
Fitness Sharing	X							
Multi-objective		SPEA2, NSGA-II	NSGA-II, OMOPSO, SMPSO, MOCHC		NSGA-II, PAES	NSGA-II		NSGA-II, SPEA2
Fitness Scaling	Linear	Sigma					Linear, Sigma	
Elitism	X	X					X	X
Steady-State Selection							X	
Crowding			X		X			X

Figure 11

3.3.3.2 Conclusion

Across the considered evolutionary algorithms, ECJ (2014) overall had the most amount of relevant features although, generally, most of the considered evolutionary algorithms had similar implementations and so, none of them functionally stood out from the rest.

An evolutionary algorithm containing all the features wanted was unlikely to be available due to the huge scope of possible functions that could be implemented. Therefore, adaptation and the implementation of extra features are inevitable for any of the evolutionary algorithms.

In order to communicate to the game component efficiently, the evolutionary algorithm component should be written in the same language; therefore Inspyred (2014) and DEAP (2012) were considered the most appropriate out of the candidate games. However, close inspection of the code revealed that the program structure of each package to be quite restrictive; many of the factors could only be exclusively implemented. As a result, because the design of experiments methodology is being used, the validity of the results attained depends on the amount of factors being tested against each other and so, the implementation of a new evolutionary algorithm may be most appropriate in order to attain valid results.

3.3.4 AI Structure

Rabin (2002) discusses several methods of designing game AI, such as finite-state machines, production systems and decision trees. Rabin illustrates that each method has differing advantages, disadvantages, and complexities, therefore an AI's structure should be chosen carefully depending on which game genre it inhabits.

In illustration to Rabin's point about optimising AI methods for different applications, Sipper *et al* (2006) applied genetic programming methods to create AI for 3 games, backgammon, chess, and robocode. Each one of the AI was structured differently and their success may have relied on it. For the backgammon and chess programs, decision making was calculated in artificially segmented trees, the backgammon tree separated 'contact' and 'race' behaviours and the chess tree separated behaviours depending on the level of advantage the player had. Unlike the other games, robocode's decision making wasn't turn-based, it was event based, seemingly because the AI was required to work in real time.

3.3.4.1 Decision Trees

Sean Luke (1998) used the method of decision trees to control the behaviour for a game of robots playing football. Each decision tree evaluated which action to do, the first tree was responsible for moving around, and the second was responsible for kicking the ball. It was concluded that the implementation of several decision trees, which forces behaviours known to be required from the problem domain, was beneficial and optimises the evolutionary algorithm to only search worthwhile areas.

Sean Luke was restricted to simulating the robot AI in real-time, which meant the evolution process was slower than ideal. A decision tree-based evolutionary AI with a high run-time should therefore be successful, considering Sean Luke's success regardless of the time-constraint.

3.3.4.2 Reactive programming

Finite-state machines are a common and popular form of reactive AI, when used with an evolutionary algorithm, problems can occur however, as shown in Spears *et al* research (2000). Spears found that evolved finite-state machine had difficulties making full use of newly accessible states that have never been seen before. A method of merging and deleting states was suggested that would solve the issue however, whilst finite-state machines have been proven worthy in games such as Thief 3 (Ion Storm, 2004), or Pac-Man (Namco, 1980), the success of evolutionary finite-state machine AI is unclear.

Reactive computer behaviour is not restricted to use in game AI, it is used widely across other computer applications. Power planning systems are under strict time constraints, which make reactive programming methods very appropriate; with a reactive system in place, emergency actions can always be prioritised above anything else. Kwang (1998) implemented 3 types of EA into a reactive system which was designed to manage power; the implementation was successful, which shows promise for EA in designing reactive AI.

3.3.4.3 Neural networks

In the detection of breast cancer, neural networks have had a classification accuracy rate of 96% (Land *et al,* 1998), neural networks have an excellent ability to solve data classification problems but their current use in commercial games is very limited (Charles, 2004).

Charles (2004) explains that most existing implementations in games of strategy suffer from being slow-paced; the most successful implementations have been in games of learning and social interaction such as Black & White (Lionhead Studios, 2001) and The Sims (Electronic Arts, 2001).

Graham (2003) shows the strengths of using neural networks by using them to generate real-time path-finding in games, the use of neural networks allowed for dynamic representation of the virtual world, the path-finding algorithm was more robust than conventional implementations because it was able to generalise what it had learnt.

Fogel (2000) explains that "the vast complexity of biological information processing systems and a lack of detailed understanding restricted the possibility for success in creating a real artificial "brain"", when discussing the implementation of neural networks in game AI.

3.3.4.4 Conclusion

Three popular methods of implementing game AI are described by Maxim Likhachev (2011), decision trees, finite-state machines and behaviour trees. It is summarised that decision trees are easy to implement, intuitive, and fast, with these points and the previous research in mind, a decision tree structure would appear to be a safe choice for the game AI framework. The only disadvantage Maxim mentions is that it requires careful manual planning, although learning trees would counteract that drawback, as Maxim also states.

Finite-state machines are a very simple and orthodox way of implementing game AI, it comes with too many drawbacks however to make a good candidate for evolutionary algorithm implementation. Alex J. Champandard (2007) backs this statement up by declaring that finite-state machines are too low-level and "their logic is limited", things like counting are impossible or difficult to do, limiting the types of behaviours that can be achieved.

Neural networks have an immense ability to tackle complex problems but this strength also limits is ability in well under-stood problem domains, as it will often attempt seemingly random solutions. Decision trees are able to search vast areas of solutions like neural networks but are more easily restricted where the problem is understood.

3.4 Experiment Methods

All data collected will be from the results of experiments which showcase the effectiveness of an AI solution playing a game against another solution. Each experiment will be built through a different evolutionary algorithm optimisation, to understand the effects of each optimisation on the AI, the resulting data from each experiment will be compared and analysed.

The data being compared from the process are the data representation of each AI solution in a population, and the fitness value of each solution. The outcome of any given game is either that one player wins and one player loses, or they both draw, depending on whether the player wins, loses or draws decides on the fitness value they receive. In order for this system to work, the difference between winning, losing and drawing must be measurable and comparable, so the fitness data is considered interval (Valerie J. Easton *et al*, 1997) and quantitative (Colin Neville, 2007). The strings that represent solutions themselves are also interval data, as they can be represented as discrete numbers and compared to be higher or lower to each other.

The data collected is said to be objective, because it is observable, measurable and unbiased (Old Dominion University, 2013). The tests are repeatable, as evolutionary algorithms are simply the complex representation of mathematical equations.

In order to make conclusions on the effects of an evolutionary algorithm across many generations, a lot of data will need to be analysed. Every solution in the population can be represented across each generation with the use of a scatter graph, or a table that summarises population data into meaningful data.

In order to understand how the data will be translated into a visual representation, the independent and dependant variables need to be defined. For each experiment, the fitness value for each individual is a dependant variable because it changes depending on the individual's string. The strings that evolve from an evolutionary algorithm in an experiment are dependent on the optimisation of factors. The factors that influence an evolutionary algorithm are the only true independent variables.

The between-group approach would be the most suitable as an experimentation method, as opposed to a within-group approach. This is because the between-group approach will focus on noticing differences between the groups and noticing correlations that may be related to their success or failure. In the context of the project, this means comparing the different experiments and noticing correlations between certain factors and the state of the AI generated. The state of the AI will be measured using the dependant measures (average fitness, number of shared schema, etc). The within-group approach usually has the advantage of having less extraneous variables, this is because the same group of individuals (Price *et al*, 2008), in a computer environment however, the same individuals can be used at the start of each experiment.

A statistical analysis of the data might be appropriate, as trends are can appear in the results of evolutionary algorithm experiments. For instance, a linear model of fitness improving can potentially appear. However, due to the sporadic nature of evolutionary algorithms, a simple statistical analysis might not be possible.

Chapter 4: Design and Implementation

4.1 System Overview

Several features were added to Jon's (2010) "simple platformer engine", including the implementation of an extra player, player collision, player score, a time limit, and an AI generator. The system was first optimised with the researched control factors in figure 3, and then optimised further to reach a comfortable level of efficiency. The implementation of Inspyred (2010) as a component was unsuccessful; instead a new evolutionary algorithm was implemented to suit the needs of the project.

4.2 Game Implementation

During development of the game, the simplicity was kept in mind as a requirement, as each game would be required to compute at least once per individual in the population. Each game was limited to a maximum of 1000 computations before a draw was decided, this would give each game a max duration of 1 second, providing the system can maintain 1000 frames per second. The short game duration was decided because to simulate a population of 158 individuals against random or pre-programmed opponents, it would take at least 2 minutes and 38 seconds, to evolve the population over 283 generations it would take 12.4 hours.

As the game duration was required to be kept quite short, the game rules would have to be simplified as to reach a winner in the short time. The resulting win conditions were to simply jump on the opponents head. The environment was set as blank, also. A visualisation of the game (arena.py) can be seen in figure 12. The text on the screen describes each player's left x position, right x position, top y position and current movement values.

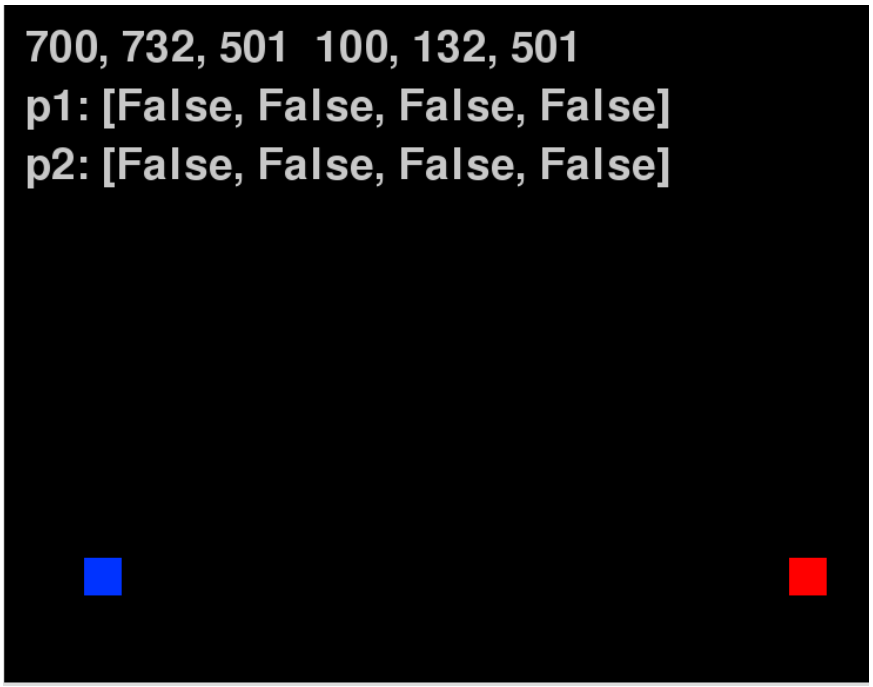

Figure 12

Each player is capable of moving left or right, and jumping. Each player starts either near the left side or right side.

Two other adaptations of the game were created; testarena.py allows the user to control the movement of one of the players, to play against an entered AI string.

The other adaption, multiarena.py takes in 3 strings and plays a match between them. The match ends when 2 players have been defeated, or when the timer runs out. A section of each string is used to select the colour for each player.

4.3 AI Implementation

A decision tree approach was implemented which translated the binary string representing each individual into a binary tree containing if statements or actions. A simple following and jumping behaviour was implemented as 100010000101000001100000, which is visualised in figure 13.

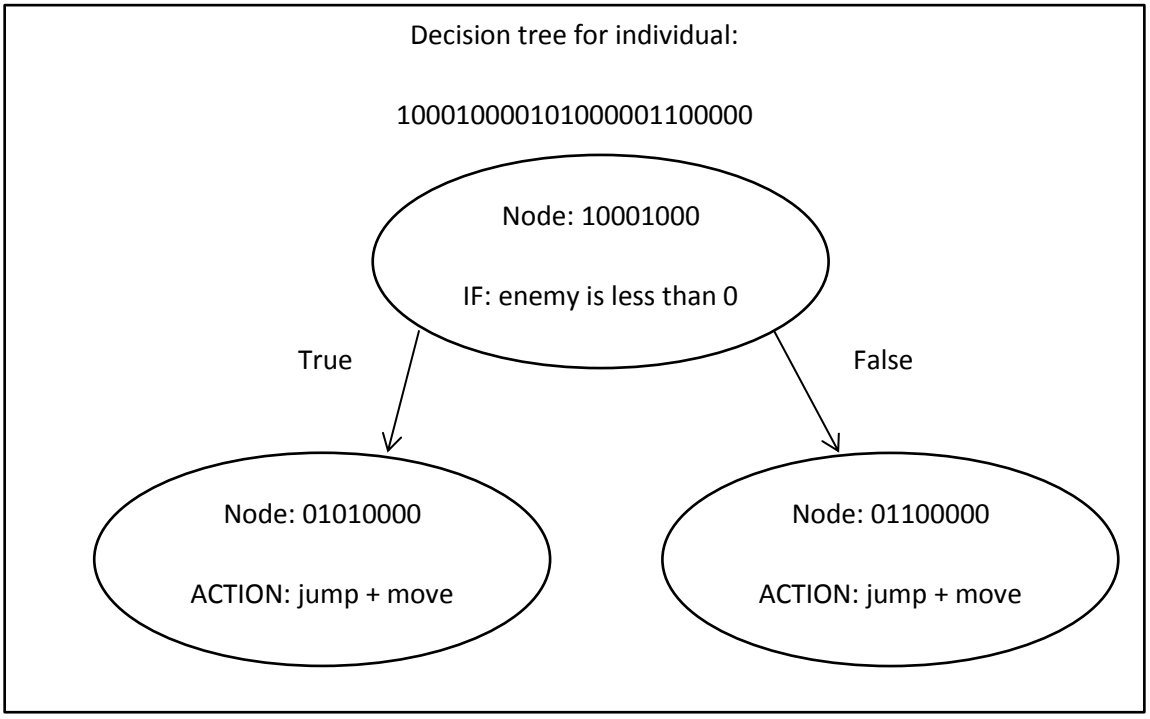

Figure 13

The system deciphers a decision tree from the string through a series of steps. The first step separates the whole string of binary digits into a list of bytes. The system then starts a recursive function of continuously adding bytes onto the tree, starting at the root. If the node is an if-statement then it will translate the next byte in the list as its' true condition, once that node is added, the next available byte is translated as the false condition. If the byte is an action, no recursive the node is simply added as the root, true condition or false condition.

One potential flaw in the system is that it can be difficult balance the tree. If a series of if statements are added in a row, for example, then one branch will stretch out in one direction. The efficiency of the tree is reduced when it is unbalanced as it is more likely to take longer to reach the solution.

The process of translating a byte into either an if-statement or action is quite simple. If the first digit is 0, the node is an action, else if the digit is a 1 then it is an if-statement. The next 7 digits describe the nature of the if-statement or action, which is outlined in figure 14.

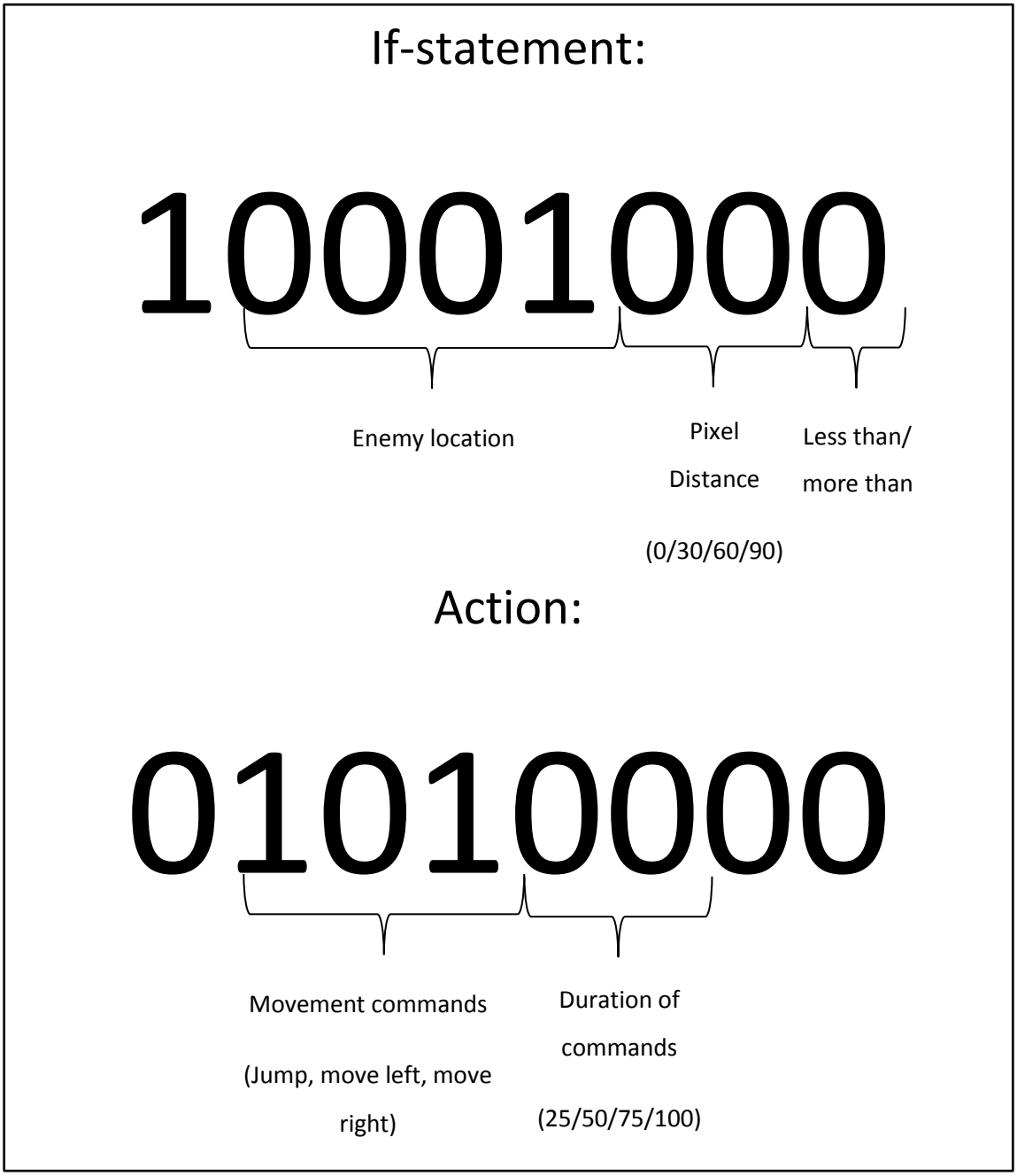

Figure 14

4.4 Evolutionary Algorithm

To reduce implementation time, David Goldberg's (1989) book "Genetic Algorithms in Search, Optimization, and Machine Learning" was used as a guide for the implementation of the evolutionary algorithm; it contains a detailed explanation and examples of code.

The evolutionary algorithm was implemented using the iterative and incremental methodology, each of the iterations consisted of the following stages: requirements, design, implementation and testing, and evaluation. Khan's *et al* (2011) model was used as a base, but the implementation and unit testing process was merged with the integration and system testing, and

operation was changed to evaluation, a process for determining the requirements of the next iteration.

More thorough testing and documentation was completed for the final iteration.

The iterations were repeated until a robust evolutionary algorithm was implemented, capable of testing multiple factors against each other and exporting the results in a suitable format.

4.4.1 Iteration 1

4.4.1.1 Requirements

The initial requirements were to build an evolutionary algorithm that could successfully complete each of the main evolutionary processes, population generation, evaluation, selection, crossover and mutation.

4.4.1.2 Design

1. Generate random population
2. Match half the population against the other half. Fitness is calculated by winning minus duration, losing plus duration, or drawing.
3. Use roulette selection to pick the next generation
4. Crossover each string at a random point
5. Implement the chance for a gene to flip, for each string.

4.4.1.3 Implementation and Testing

Black box testing was used, as the next iteration's requirements firstly depended on the success of the current iteration. The success of the iteration could be tested by observing the effectiveness of the population at generation 100, which can be seen in figure 15.

Test ID	Individual	Observed Behaviour
A1	0110000010000111010110111000000101100111010000101100010001000101111010001111010100011011001011010101000111011111101111111111011111100100001101110100111111101101110	Moves left and jumps
A2	0011000010010011010110111000001101100111010001111100111000101011100101000110101111111110110010101111000000110010011110110000001101010111101111010100111001000110	Moves right
A3	0110101000111000111101001000001101001101010111011001110111000101100010001111001111100001011001111101001100010110101101011111100011001010010111101000000100011100001	Moves left and jumps
A4	0110100000111000111101001000001101011101010110011001110111000101100010001111010100110011010011010101000111011111101111111111011111100100001101110100111110	Moves left and jumps

	1111111	
A5	0010000010000111010110111100010111100111010000101101111000110101110110101011010100011011001110111111000100011110010101010010000000101000011011101001111101101110	Moves left
A6	0111001000101110011101001000000001110111010000101101011001110101110010101011010101011010001011101010000111011111101111111110111111100100001101110100111110110111 0	Moves right and jumps
A8	0110100110000011110010001000000101100111011000101100010001111011101001010100101001101001101010100011101111110111111111101111110010000110111010011111011011110	Moves left and jumps
A9	0110000000111010100101001000000101111101010110111101111110001011000100011111100001100111100110101010001010111010111011100110111001011010111101001000010001100100	Moves left and jumps
A10	1111001010111000111101011100100101101111010100101101011001111101010101001010110000010111010100001010010111000011010110000111101010000110111010011111011111 0	Moves right and jumps if the opponent jumps or is left.
A11	0110000010000111010110111000000010110011101000010110001000101011101100100110111101100101100010101010001110111011111011101111110001100101001011110100000011011000 01	Moves left and jumps
A12	0110000010000011110110111000000001010111011000101101001000111010001000111000100011011001010110111001101001100000101011011000111101001010101010000010001100010	Moves left and jumps
A13	0110100000111000111101001000001101011101010111110011110011001110001010111100000011011011001101000010001101101111111011000011110100000011011101001111111111010	Moves left and jumps
A14	0110000110011101110110111000000110110011101000010111111011110001000100011110101001100110100110101010001110111011111011101111110001100101001011110100111110110111 0	Moves left and jumps
A15	0111011010011011110110111010110000100101010110101101100010010111001010001101011111101101100101011100000000100100111101100000011010101111011111010100111011001 10	Moves right and jumps
A16	0110000010000111010110111000000010110011101000010110001000101111010001111010100011011001110111011001001101110000010111101000001110110001101110100111110111101 0	Moves left and jumps
A17	0110001000111000111101011000000001101001010010111001110010110011001101100000010110011001101101110010000110100011010110011001010100101101011010100010110111000000	Moves left and jumps
A18	1111001010111000111101011100100101101111010100101101011001111101010101001010110000010111010100001000101101010110101100010111111010001010111110000110011001100	Moves right and jumps
A19	0110000010000111010110111100000010110011101000000110001100101011101111010010100010100110010111011100001010010110111010110000011110101000010111101000111101 0	Moves left and jumps

	1100100	
A20	0110100110000011110010001000000101111101010111111011101000110101110010001111000100110011010011010101000111011111101111011111100011001011010111101000111010011100	Moves left and jumps
A21	1110011010111000110101001000000101111101010111111011100001001001110001010111010000011011001001101000110000001100011011101100001111111011000110111010011100011111010	Jumps for a short duration if the opponent jumps.
A22	0110100000111000111101001000001101011101010110011001110111000101100010001111001111100001011001111101001100010110100110111100110111100100001101110100111110111011	Moves left and jumps
A23	0110000010000111010110111000000010110011101000010110001000101111010001111010100011011001110110111001101111101111011111100011001011010111101000111010011100	Moves left and jumps
A24	0110001000111000111101011000000001101001010010111001110001011001100111001101100000001011001100110111011001010011001011000011110101000011011101001111101111110	Moves left and jumps
A25	0110100110000011110010001000000101100111011000101100100001111011101000011110001000110110010101101110011010011000001010110110001111010010101010100000010001100010	Moves left and jumps
A26	0110100110000011110010001000000101111101010111111011000110010011100010101110000001101101100110000101000110101011111101100001111111000000110111010111111011101110	Moves left and jumps
A27	0110000000111010100101001000000000100101010110101100111000101000111010001101011111101101100101011100000001001001111011000000110101011110111101010011101100110	Moves left and jumps
A28	0110000011000001110110111100010101000111010000101101000001101011101100100110111101100101100010101010001110111011110111111000110010010101111110000010001110010	Moves left and jumps
A29	0110110010000011110110111010110000100101010111101111000110010011100010101110000001101101100110100001000110101011111101100001111111010111101111010100111011001100	Moves left and jumps
A30	0110100000111000111101001000001101011101010110011001110111000101100010001111010100011011001110110111001100011011100000101111010000010100101010111111100010001100001	Moves left and jumps
A31	0110000010000111010110111100000101100111010000001100011010111010110101010010001010011001011101110100011011010111111011000011111101000011011101001111101101110	Moves left and jumps

Figure 15

4.4.1.4 Evaluation

The testing revealed that the most popular strategy was to either always move and jump left, or always move and jump right. This could be identified as premature convergence as the strategy developed could be easily beaten a non-moving opponent, that weren't facing them. Therefore,

to encourage the algorithm to develop behaviours for winning against a non-moving opponent, on both sides, the next iteration should calculate fitness over two games, each game featuring the two opponents in different positions. Also, the next iteration should force strings to begin with an if-statement, instead of having the chance to begin with an action, as AI's which simply repeat an action are stagnating the population from advancing.

Also, the best control factors, learned through the literature review should be implemented. To measure the performance of the algorithm, the average fitness and maximum fitness should be implemented

4.4.2 Iteration 2

4.4.2.1 Requirements and Design

- Implement fitness evaluation through two games against a non-moving opponent.
- Implement the control factors in figure 3.
- Implement code to measure the average and maximum fitness.
- Implement strings to begin with if-statement.

4.4.2.2 Implementation and Testing

Figure 16 describes the average and max fitness for the population over 100 generations. It can be seen that the best solutions began appearing at generation 47 but were wiped from the population at generation 73.

Appendix 10.1 shows the results of the test for each generation.

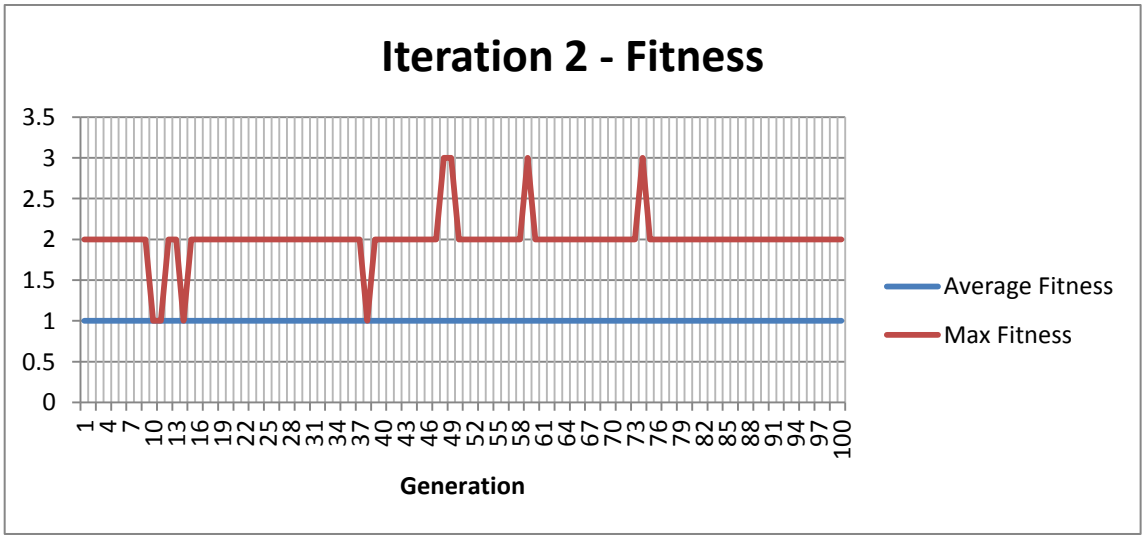

Figure 16

Figure 17 displays a collection of individuals generated in iteration 2 with a fitness of 3.

Test ID	Individual	Observed Behaviour
B1	10010000101010000101110101101100 10100011001101000101101010010001	Moves left and jumps, if the opponent moves within a close distance it begins moving and jumping in the direction of opponent.
B2	11010110111100000101110101101101 00010100010010101000010000000110	Moves towards the player and jumps.
B3	11111110110101000101110101101111 01010110010000101000010000000100	Moves towards the player and jumps.
B4	11011010110101100111011011010001 01110100110100100110110100010100	Moves towards the player and jumps.
B5	11010110100101100111001111110101 00010000100100110110010100010100	Moves towards the player and jumps.

Figure 17

4.4.2.3 Evaluation

We can gather that the best solutions aren't surviving to pass down their genes well enough. This may be because the best solutions aren't rewarded with enough fitness, or maybe mutation is altering the new solutions before they can spread through the population, there are many different factors that could potentially be affecting the performance. It should be a focus, then to implement measures to allow for the easy manipulation of factors.

The next iteration should also use a predetermined seed to generate the initial population. This will increase the validity of any comparisons made between experiments, as they will all be under the same input.

4.4.3 Final Iteration

4.4.3.1 Requirements

- Implement extra evolutionary algorithm operators.
- Add the capability to switch operators easily.
- Implement a seed before using any random variables

4.4.3.3 Operations

4.4.3.3.1 generatePopulation()

Returns a randomly generated population of strings

4.4.3.3.2 simpleCrossoverRandom(pos, population)

Requires a population of strings and the string to be crossed-over's position in the population. Generates a random number between 0 and 1, if the number is lower than the crossover rate then crossover is performed, crossover selects a random partner from the population and splices the two strings together at a random point to form a new string. Returns the crossed individual.

4.4.3.3.3 simpleCrossover(parent1, parent2)

Requires two strings. Generates a random number between 0 and 1, if the number is lower than the crossover rate then crossover is performed, splices the two strings together at a random point to form 2 new strings. Returns the crossed individuals.

4.4.3.3.4 singleFlipMutation(individual)

Randomly flips a random position in a string, depending on mutation rate, from 0 to 1, or from 1 to 0. Returns the mutated individual.

4.4.3.3.5 evaluationNonMoving(population)

Returns the fitness for each string in a given population. Each string competes twice in a game against a non-moving opponent, each time in a starting different position. Fitness increases by 1 each time the individual beats the non-moving opponent.

4.4.3.3.6 randomEvaluation(population)

Returns the fitness for each string in a given population. Each string competes twice in a game against a another evolving AI, each time in a starting different position. An individual's score increases by 1 when they defeat the opponent.

4.4.3.3.7 randomEvaluation(population, population)

Returns the fitness for each string in the given populations. Each string competes twice in a game against an AI from the other population, each time in a starting different position. Fitness increases by 1 each time an individual beats another opponent.

4.4.3.3.8 random3PlayerEvaluation(population)

Returns the fitness for each string in a given population. Each string competes three times in a game against a two other AIs, each time in a starting different position. An individual's score increases by 1 when they defeat an opponent.

4.4.3.3.9 rouletteSelection(population, populationFitness)

Returns a mating pool from a given population, depending on the populationFitness. Mating pool is decided by randomly selecting individuals from the population. Inidividuals with higher fitness have a higher likelihood of being selected.

4.4.3.3.10 recordStatistics(population, populationFitness)

Collects a series of statistics about the population and saves them to a text file, also records the population and population fitness. Calculates the average fitness, standard deviation of fitness, max fitness, min fitness, median fitness, number of unique fitness values, and number of unique genotypes.

4.4.3.3.11 addElites(population, populationFitness)

Returns a list of elites out of a given population.

4.4.3.3.12 steadyStateRemoval(population, populationFitness)

Returns a population and population fitness with the lowest ranking individuals removed.

4.4.3.4 Testing

White-box testing was used to test each of the system functions (figure 18).

generatePopulation()			
Test ID	Input	Expected output	Actual output
C1	None.	A list of randomly generated strings.	As expected
simpleCrossover(parent1, parent2)			
Test ID	Input	Expected output	Actual output
C2	'aaaaaaaaaaaaaaaaaaa'. 'bbbbbbbbbbbbbbbbbbb'.	Two strings, each containing a portion of the passed in strings.	'aaaaaaaaaaaabbbbbbbb' 'bbbbbbbbbbbbaaaaaaaa'
singleFlipMutation(individual)			
Test ID	Input	Expected output	Actual output
C3	'0000000000'.	A random bit has been changed from '0' to '1'.	'0010000000'
C4	'1111111111'.	A random bit has been changed from '1' to '0'.	'1111110111'
evaluationNonMoving(population)			
Test ID	Input	Expected output	Actual output
C5	A population of non-moving strings ('00000000').	No one scores any points; function returns fitness for each individual as 1.	As expected.
C6	A population of moving left and jumping strings ('01100000').	Each individual scores a point; function returns fitness for each individual as 2.	As expected.
C7	A population of chasing and jumping AIs ('10001000010100000110 0000').	Each individual scores 2 points; function returns fitness for each individual as 3.	As expected.
randomEvaluation(population)			
Test ID	Input	Expected output	Actual output
C8	A population of non-moving strings ('00000000').	No one scores any points; function returns fitness for each individual as 1.	As expected.
randomEvaluation(population, population)			
Test ID	Input	Expected output	Actual output
C9	A population of moving left and jumping strings ('01100000') and a population of non-moving strings ('00000000').	Each individual in the moving population scores a point, no one scores in the non-moving population; function returns fitness for each individual as 2 for	As expected.

		the moving population and 1 for the non-moving population.	
C10	A population of chasing (‘10001000010100000110 0000’) and a population of non-moving strings (‘00000000’).	Each individual in the moving population scores two points, no one scores in the non-moving population; function returns fitness for each individual as 3 for the moving population and 1 for the non-moving population.	As expected.
random3PlayerEvaluation(population)			
Test ID	Input	Expected output	Actual output
C11	A population of non-moving strings (‘00000000’).	No one scores any points; function returns fitness for each individual as 1.	As expected.
rouletteSelection(population, populationFitness)			
Test ID	Input	Expected output	Actual output
C12	A population with equal fitness.	The returned population accounts for most of the entered population. Low chance that any individuals appear noticeably more than others.	As expected.
C13	A population with one individual with twice the fitness as all the others.	The individual with the highest fitness appears more than once in the new population. Low chance that it won’t.	As expected.
recordStatistics(filename, population, populationFitness)			
Test ID	Input	Expected output	Actual output
C14	Population of varying values.	Population, population fitness, average fitness, standard deviation of fitness, max fitness, min fitness, median fitness, number of unique fitness values, and number of unique genotypes are saved to a text file.	As Expected
addElites(population, populationFitness)			
Test ID	Input	Expected output	Actual output
C15	A population where the first two individuals have the highest fitness. Number of elites set to 0.	Returns nothing.	As expected.
C16	A population where the first two individuals have the highest fitness. Number of elites set to 1.	Returns the first individual.	As expected.
C17	A population where the first two individuals have the highest fitness. Number of elites set to 2.	Returns the first two individuals.	As expected.
steadyStateRemoval(population, populationFitness)			
Test ID	Input	Expected output	Actual output

C18	A population where the first two individuals have the lowest fitness. Number of steady state removals set to 0.	Returns nothing.	As expected.
C19	A population where the first two individuals have the lowest fitness. Number of steady state removals set to 1.	Returns the first individual.	As expected.
C20	A population where the first two individuals have the lowest fitness. Number of steady state removals set to 2.	Returns the first two individuals.	As expected.

Figure 18

Figure 19 shows the average and max fitness for the population.

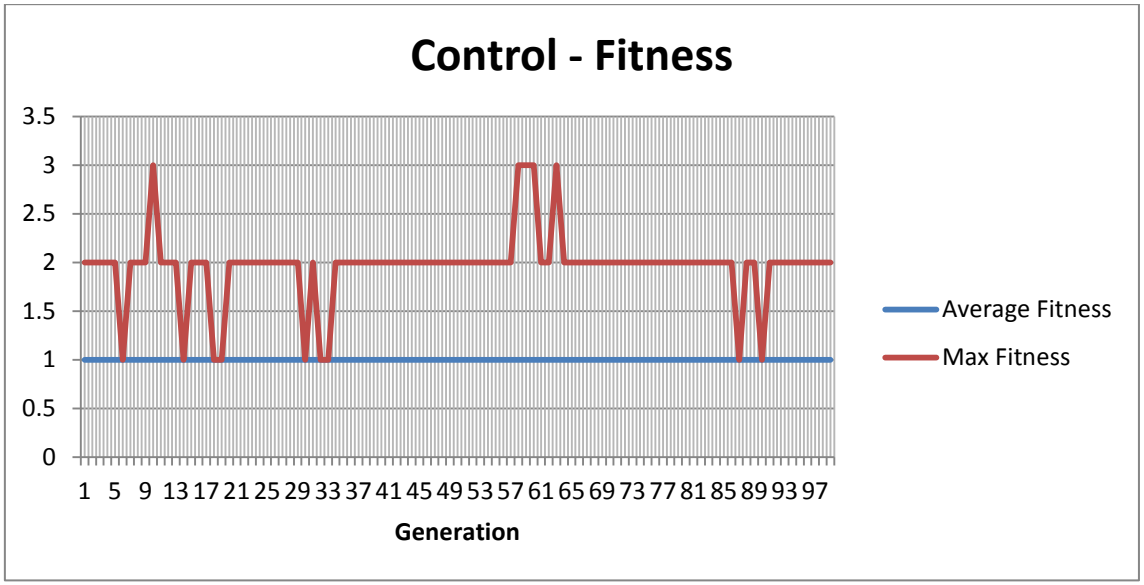

Figure 19

4.4.3.5 Evaluation

The testing results report that the system is fully operational and robust. A large amount of extra operators have been implemented, which should enable many factors to be tested.

It is clear however, that the system is not performing optimally. Despite generating with an high achieving individual early on at generation 10 with a fitness value of 3, the genes are not moving through the population. As we know that the inner working of the evolutionary algorithm work through black box testing, it is highly likely that the optimisation of factors is causing the problem.

Generations through 58 to 60, all maintain a max fitness of 3, this shows hope that the system can maintain good solutions and that the passing down of fit genes operates on some level, it also shows improvement from the first iteration as the best solutions have been observed to last 1 generation longer. Because of this, the system was deemed suitable for experimentation, with no more iterations required.

Chapter 5: Design of Experiments

5.1 Study Design

The experiments being conducted are being carried out in order to better understand the workings of the evolutionary algorithm. Within evolutionary systems, different processes are carried out to generate the population of solutions, in the control group; a generic process is being carried out at each step. Each study will alter a process by inserting an alternative implementation to determine how each implementation can affect the outcome.

For each experiment, statistical data is being measured from each generation, including the population representations, fitness values, and fitness statistics such as average and max. The data will be primarily analysed through quantitatively though data analysis but also qualitatively by spotting trends in the data over time.

To analyse an individual AI, the behaviour is observed through entering the string into the testarena.py code.

The implemented functions allow for the testing of many different factors. The factors that have been selected for testing are population size, evaluation methods, elitism, steady state removal, and island populations. A generic implementation will be used as a control group to compare the results against, as each experiment will share the same factors optimisation with it, except the factor being tested.

- Study 1: Exploring the effects of population size on the evolutionary algorithm's ability to generate an efficient AI.
    - Half-population size
    - Quarter population size
- Study 2: Exploring the effects of simultaneously evaluating AIs on the evolutionary algorithm's ability to generate an efficient AI.
    - Competing AI – 1 versus 1
    - Competing AI – 1 versus 1 versus 1
- Study 3: Exploring the effects of elitism and steady state removal on the evolutionary algorithm's ability to generate an efficient AI.
    - 5 elites
    - 5 steady state removals
    - 5 elites and 5 steady state removals

- Study 4: Exploring the effects of separated populations on the evolutionary algorithm's ability to generate an efficient AI.
    - Competing island populations

The largest issue for the control group has been in maintaining good members of the population. Based on this, the first hypothesis will be that implementing elites into a population results in a better overall population, than the control group. The reasoning behind this prediction is that elites protect the best individuals from leaving the population, so the randomness of the roulette elimination method doesn't affect them.

Evaluating each individual by competing them against one-another to determine fitness will always, at best, result in at least half of the population losing, because for one individual to win, another has to lose, whereas against non-moving AI, the entire population could potentially win. Also, because even a randomly moving opponent is more difficult to defeat than a non-moving one, the second hypothesis will be that competing AI result in a less overall fitness than the control group.

The third hypothesis will be that reducing the population size will promote population stagnation. The reasoning behind this is that it is easier to spread a good solution through to the entire population when there aren't many to spread to.

The fourth and final hypothesis will be that implementing two competing island populations will result in uneven fitness levels between groups.

5.2 Experiment Description

Each experiment is running off the same python file (experiments.py). The experiment is selected by altering the autorun code at the end of the file, which selects a function to run.

5.2.1 Control Group

The control group will generate an AI through standardised, generic steps, in order for comparisons to be made. The taken in generating AI through the control group are as follows:

1. Generate initial population.
2. Evaluate using non-moving opponent evaluation method.
3. Select mating pool using roulette selection method.
4. Apply simple crossover to each individual.
5. Apply four simple flip mutations to each individual.
6. Restart from step 2 until 99 generations have passed.

The system will be optimised with the following factors:

- Population size: 90
- Genotype length: 64
- Mutation rate 0.5
- Crossover rate: 0.5

Within experiments.py, control() runs the control experiment.

5.2.2 Population Size

The procedure for the population size experiments is the same as the control group, except for the half population size experiment, the population size was set to 45 and for the quarter population size experiment, and the population size was set to 22.

Within experiments.py, control_halfPopulationSize () runs the half population size experiment and control_quarterPopulationSize() runs the quarter population size experiment.

5.2.3 Simultaneous Evaluation

For the competing AIs experiment, the evaluation method has been changed from evaluationNonMoving(population) to randomEvaluation(population). The implemented evaluation method matches each individual in a game against another individual to determine its' fitness.

In the 1 versus 1 versus 1 competing AI experiment, the random3PlayerEvaluation(population) method is used. In this method, individuals are evaluated in matches where three different AI's compete. Each individual gains a point for each opponent they beat. The game ends after a duration, or when one individual remains.

Within experiments.py, competingAI() runs the 1 versus 1 competing AI experiment and competingAI_3PlayerEvaluation() runs the 1 versus 1 versus 1 competing AI experiment.

5.2.4 Elitism and Steady State Removal

In the 5 elites experiment, the number of elites was set to 5, in the 5 steady state removals experiment, the number of steady state removals was set to 5, and in the 5 elites 5 steady state removals experiment, both factors was set to 5.

Within experiments.py, control_5Elites() runs the 5 elites experiment, control_5SteadyStateRemovals() runs the 5 steady state removals experiment, and control_5Elites_5SteadyStateRemovals () runs the 5 elites 5 steady state removals experiment.

5.2.5 Island Populations

In the competing island populations experiment, each step, except the evaluation step is completed twice, once each for each separate population. The randomEvaluation(population, population) method is being used also to evaluate the two populations in games where each individual is against another from the other population. The population size has been set to 45 for the total combined population size to equal the usual 90.

Within experiments.py, competingAI_islandPopulations() runs the competing island populations experiment.

5.3 Results and Analysis

5.3.1 Control Group

Appendix 10.2 contains the results in full.

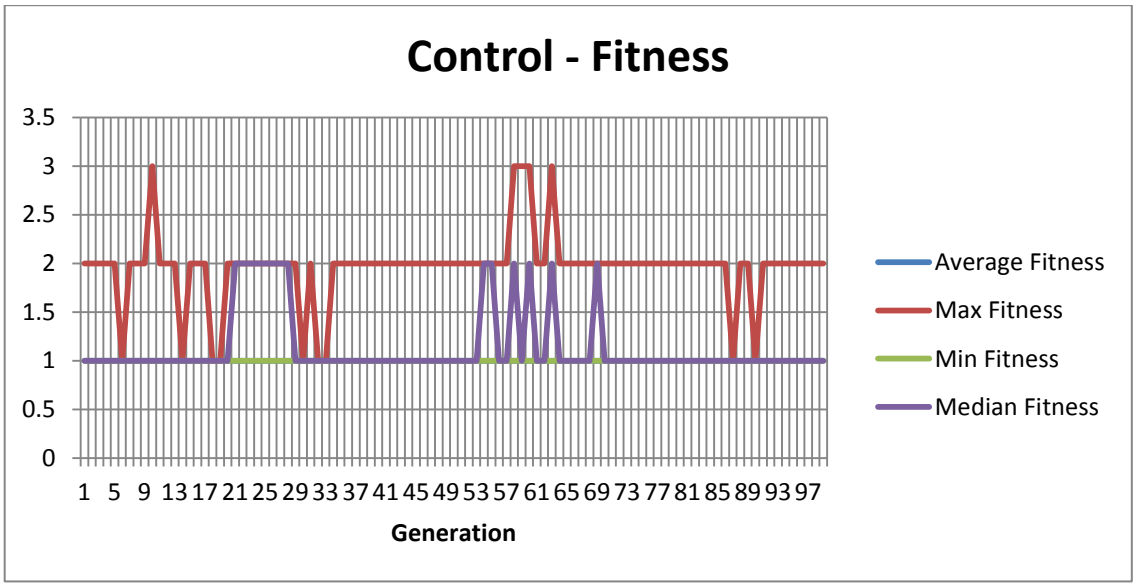

Figure 20

Average Statistics			
Average Fitness	Max Fitness	Min Fitness	Median Fitness
1	1.95959596	1	1.141414141

Figure 21

5.3.1.1 Analysis

In the final population for the control population doesn't appear to be particularly impressive. In generation 99, the only individual to defeat an opponent is individual "110001011000110101010111100010000010000101101010001110100010000 0". This suggests a lack of progress in the game AI's evolution. The last individual to achieve 2 wins was

"10001110101111010110000000010010100110110010000001011011111101010" at generation 63, this individual evolved a following and jumping behaviour, but for some reason it wasn't carried into the next generation.

5.3.2 Population Size

5.3.2.1 Half Population Size

Appendix 10.3 contains the results in full.

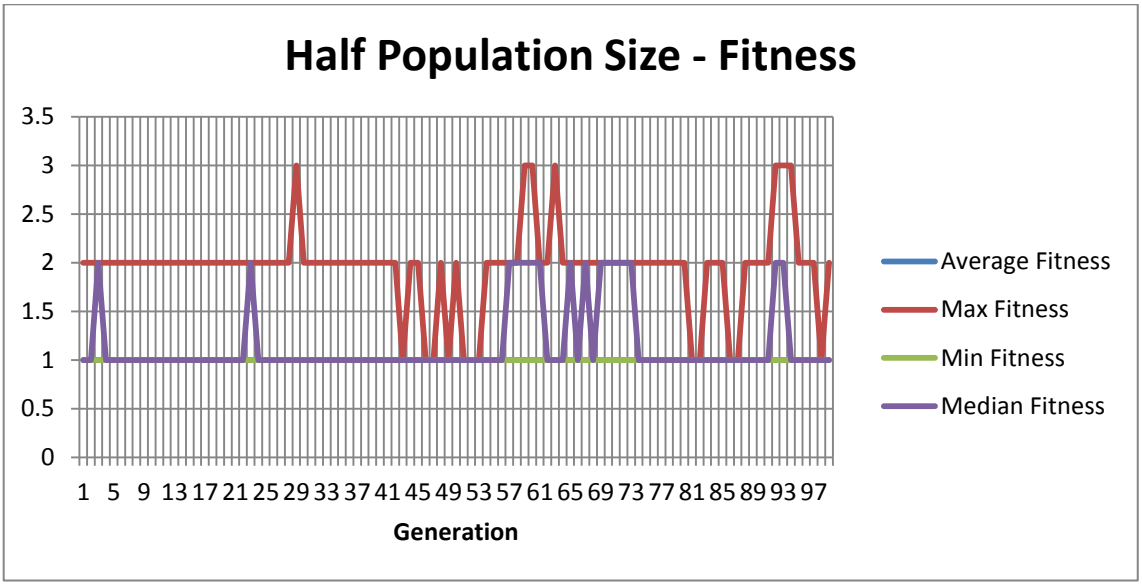

Figure 22

5.3.2.2 Quarter Population Size

Appendix 10.4 contains the results in full.

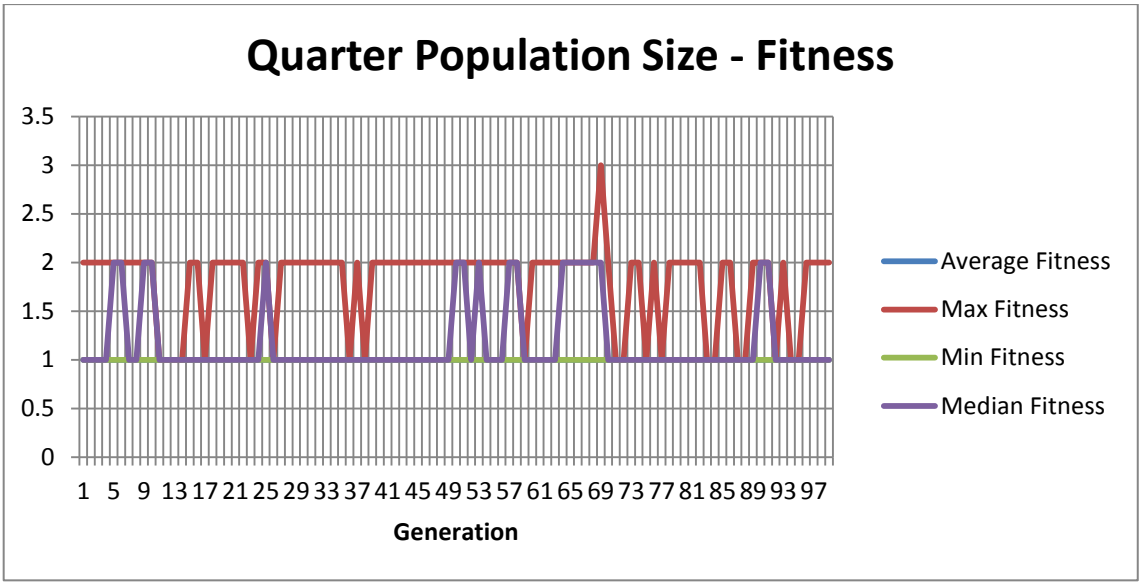

Figure 23

5.3.2.3 Analysis

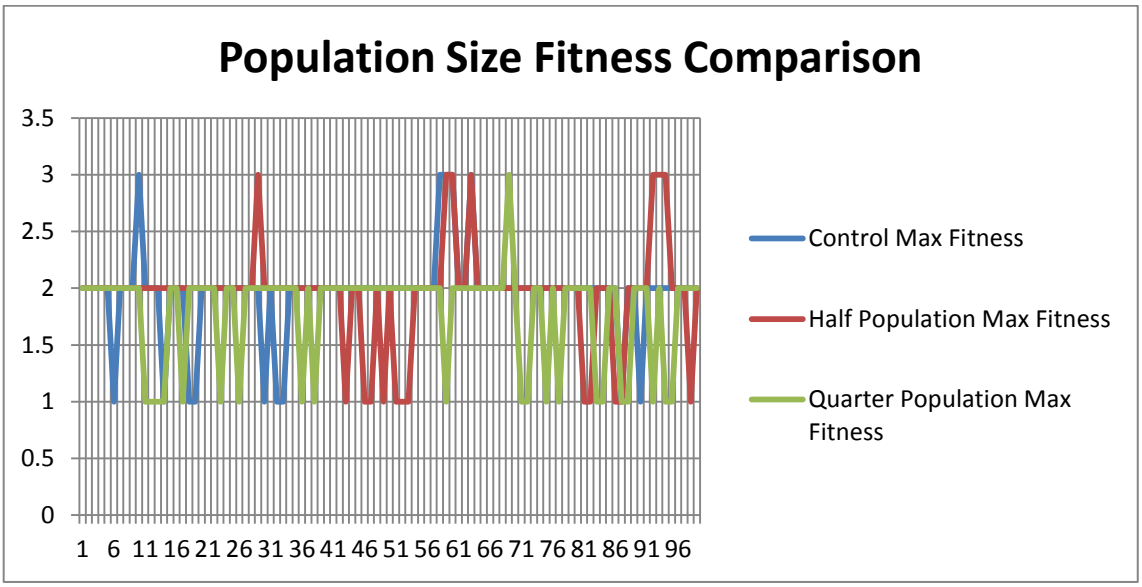

Figure 24

Figure 24 displays a comparison between the control, half population and quarter population max fitness. Quantitatively, looking at the comparison table in figure 25, not a lot stands out from either of the groups, except quarter population average max fitness is the lowest; however it also has the highest median.

	Average Statistics			
	Average Fitness	Max Fitness	Min Fitness	Median Fitness
Control	1	1.95959596	1	1.141414141
Half Population	1	1.94949495	1	1.161616162
Quarter Population	1	1.7979798	1	1.181818182

Figure 25

The decrease in max fitness and increase in median fitness might suggest that the population is more converged than the other groups. This theory aligns with the hypothesis that reducing the population size will promote population stagnation, as premature convergence can lead to stagnation. Given that the difference in values is very small, and that the max fitness fluctuates so drastically, it is unlikely that the reduced population size has made a drastic change.

Similarly to the control group, both half population size and quarter population size experiments were able to develop AI's capable of following and jumping on the opponent, but they weren't able to keep their genes in the population despite this. Given that final population for both experiments only contained AI's that moves in one direction and jumped, or didn't do anything

advantageous, it can be said that the experiments were unsuccessful at improving the control experiment's optimisation.

5.3.3 Simultaneous Evaluation

5.3.3.1 Competing AI – 1 versus 1

Appendix 10.5 contains the results in full.

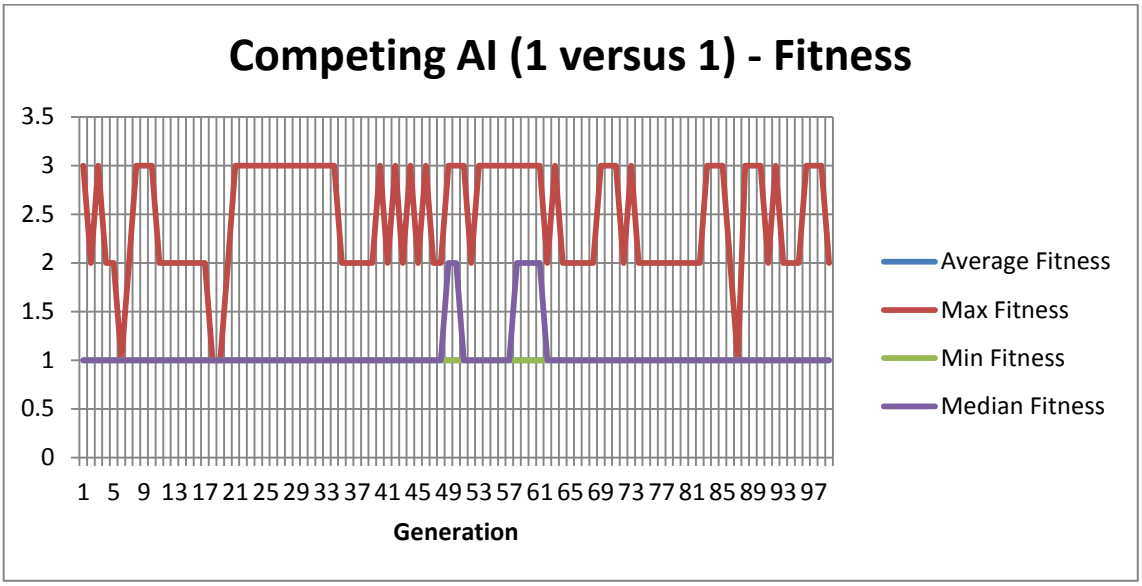

Figure 26

5.3.3.2 Competing AI – 1 versus 1 versus 1

Appendix 10.6 contains the results in full.

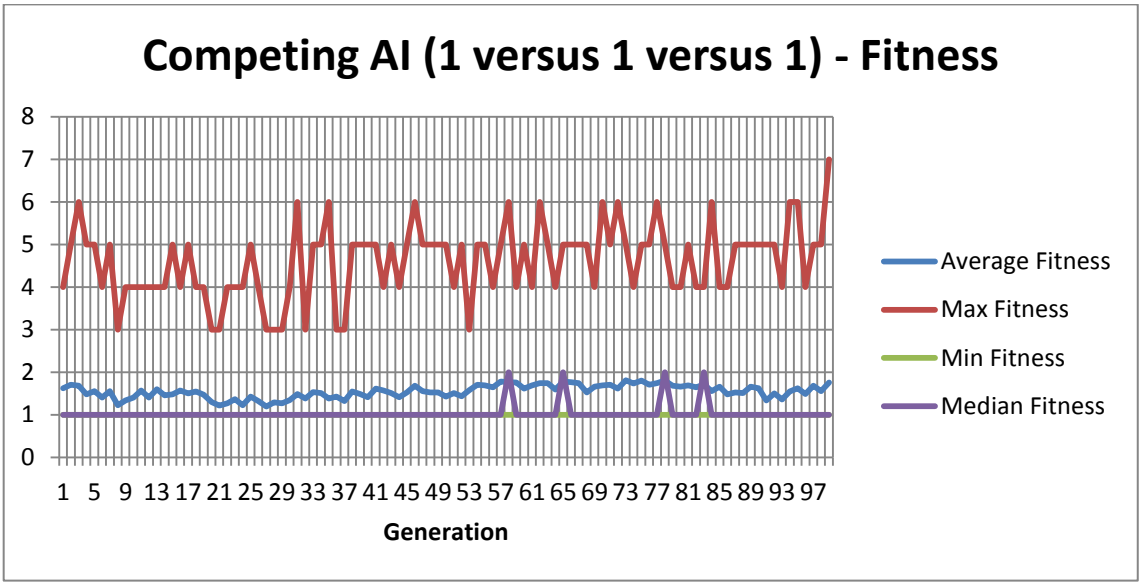

Figure 27

5.3.3.3 Analysis

The fitness graph for the 1 versus 1 versus 1 shows that max fitness never declines lower than 4 after generation 53, it can also been seen that after this point, the median fitness spikes on 4 different occasions. If the final max fitness recording of 7 is also taken into account, it the graph shows a slight positive trend towards a higher fitness.

Individuals in the 1 versus 1 versus 1 experiment can potentially reach fitness of 9, if they play perfectly in each of their evaluation matches. Therefore it doesn't make for a fair comparison, to compare raw fitness values, as individuals in other experiments can only reach a fitness of 3. It is interesting however to observe how, despite having the natural advantage, the median fitness never surpasses the control or 1 versus 1 experiments (figure 28).

The 1 versus 1 versus 1 experiment also appears to show a lot more detail than other experiments, looking at the average fitness in figure 27, a lot more variation between generations can be noticed, compared to other experiments where the average fitness usually stays on 1.

	Average Statistics			
	Average Fitness	Max Fitness	Min Fitness	Median Fitness
Control	1	1.95959596	1	1.141414141
Competing AI (1 versus 1)	1	2.46464646	1	1.060606061
Competing AI (1 versus 1 versus 1)	1.545679012	4.60606061	1	1.04040404

Figure 28

It is interesting how the overall max fitness for the 1 versus 1 experiment is significantly higher than the overall max fitness for the control experiment, as it goes against the hypothesis that was set. The hypothesis expected the overall fitness to be lowest in the competing AI group because at least half the population will always lose. One possible explanation could be that having a varied range of opponents led to a more hostile environment, where only the fittest solutions could survive, in contrast to a non-moving opponent environment, where a solution only has to jump and move in one direction to achieve an adequate fitness. The difference in population fitness is contrasted visually in figure 29.

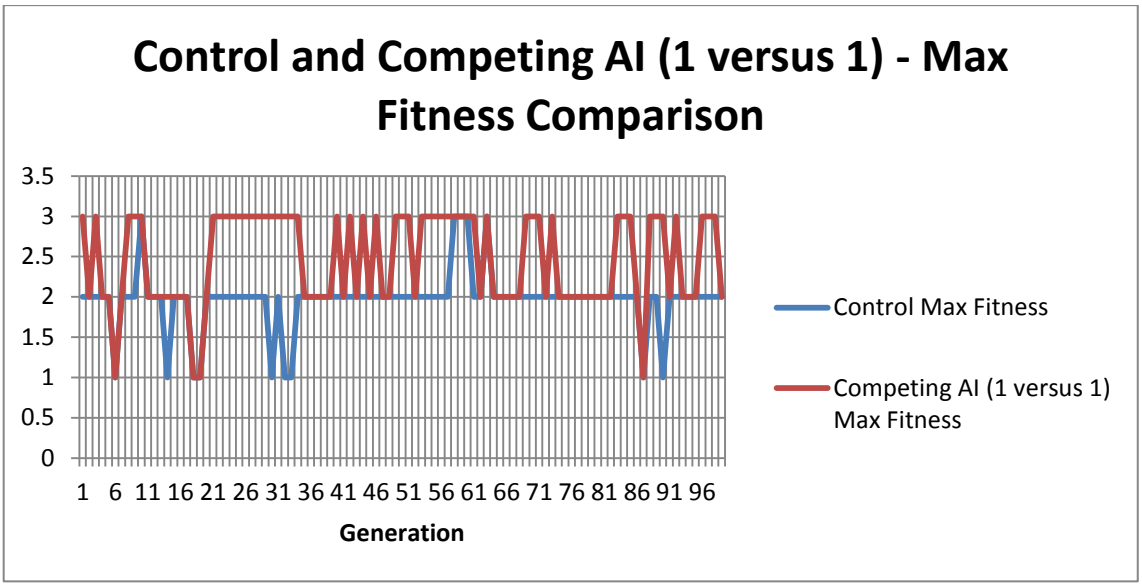

Figure 29

Despite the difference in fitness over time, the end result for both populations in the control experiment and 1 versus 1 experiment are very similar. The best individuals to be evaluated in the 1 versus 1 experiment exhibit the simplistic move in one direction and jump strategy.

It is interesting to notice that some members in the 1 versus 1 competing group that were evaluated with a fitness of 3, didn't use the following and jumping tactic, but the strategy that usually results in fitness of 2, the moving in one direction and jumping strategy. This can be explained by looking at the behaviour of the matched up AI, in generation 98, a moving right and jumping AI was able to defeat its opponent twice because the opponent's strategy was to just move right. This unexpected behaviour allowed an unfavourable AI to be rewarded; it is likely that this scenario happened more than once throughout the experiment, in which case, the 1 versus 1 competing AI's max fitness could be a reflection of that, since it is generally higher than the control's.

The unfair evaluation of fitness also reflects in the 3 player competing AI experiment, in the final round of evaluations, one individual was able to score a point in each round. The individual only employed a move right and jump strategy, it was able to win because both of its' opponents either did the same or simply moved right. In order to fairly evaluate fitness, AIs with equally successful strategies should be evaluated fairly; the system of replaying games in different positions might work against non-moving AI opponents, but competing AI appear to be able to exploit it.

5.3.4 Elitism and Steady State Removal

5.3.4.1 5 Elites

Appendix 10.7 contains the results in full.

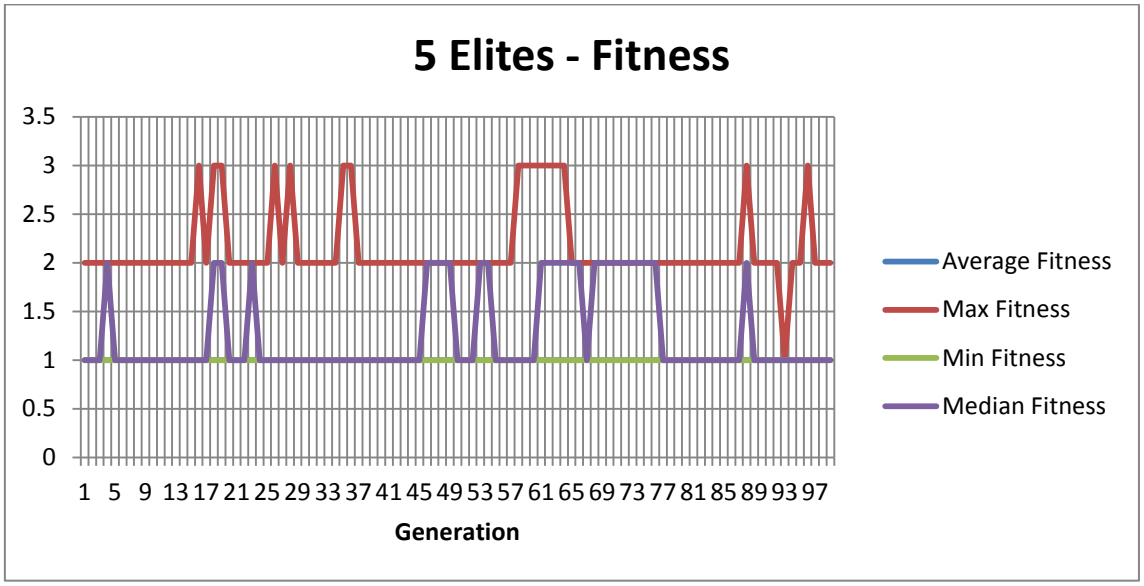

Figure 30

5.3.4.2 5 Steady State Removals

Appendix 10.8 contains the results in full.

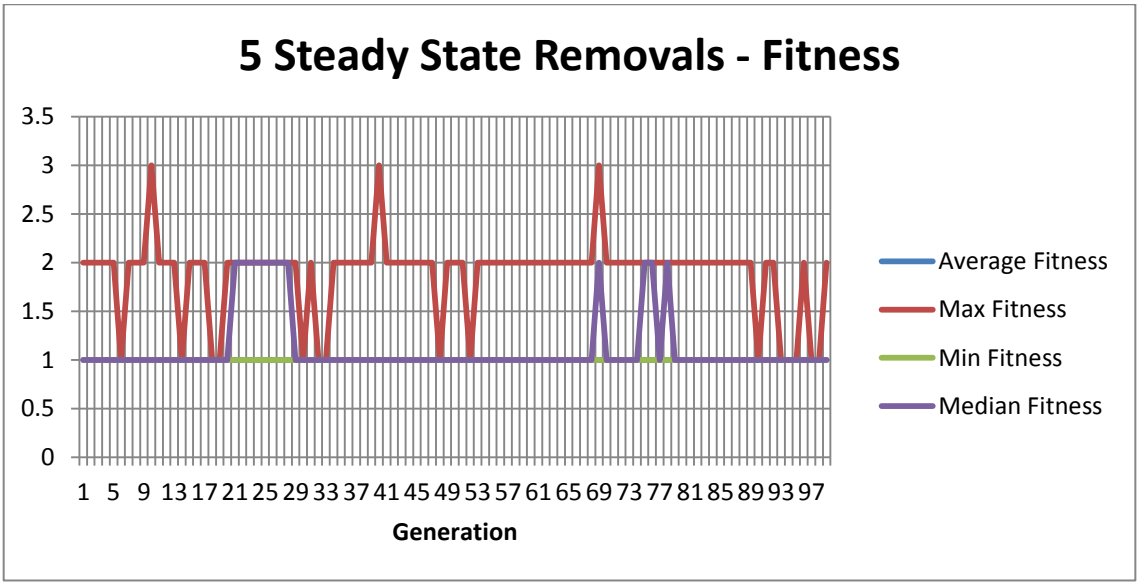

Figure 31

5.3.4.3 5 Elites 5 Steady State Removals

Appendix 10.9 contains the results in full.

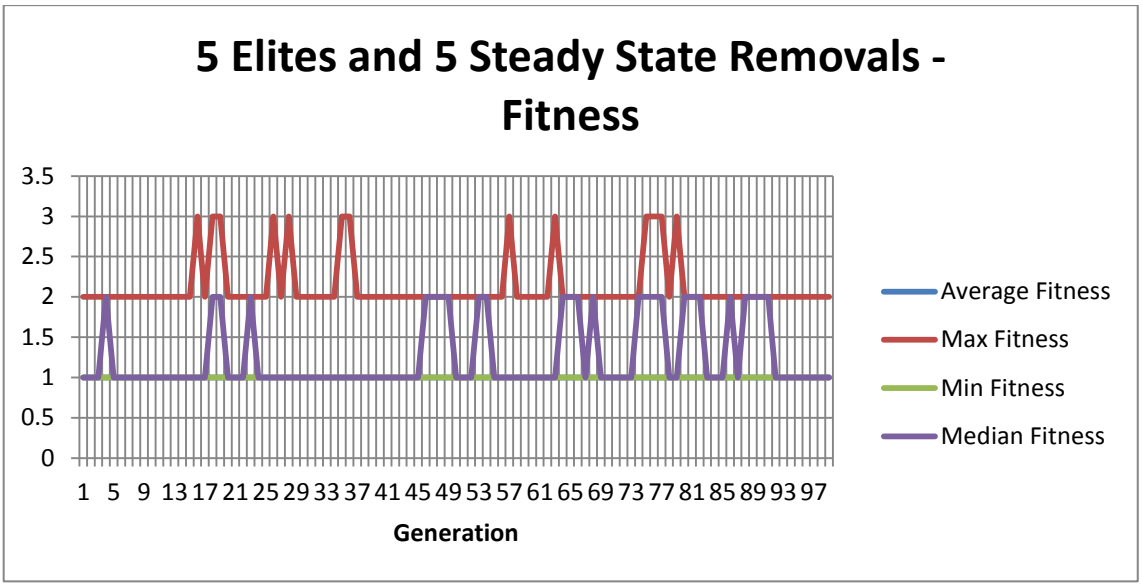

Figure 32

5.3.4.4 Analysis

Looking at figure 30, it appears as though implementing 5 elite individuals into a population results in highly fit individuals staying in the population for longer, which can especially be seen between generations 59 and 65 where a max fitness of 3 is maintained for 6 generations. This observation fall in line with the hypothesis that implementing elites will result in a higher fitness than the control group.

Between the control, elite and steady state experiments, the experiments which implemented elites are noticeably more success. Figure 31 shows that the implementation of 5 steady state removals has had very little effect on the overall fitness, compared to the control experiment; this is further highlighted in the comparison between 5 elites and 5 elites 5 steady state removals, where steady state removals have also made little impact, from the statistical overview in figure 33. Steady state removal's lack of effect may be because the roulette selection method is already quite capable of removing unfit individuals.

	Average Statistics			
	Average Fitness	Max Fitness	Min Fitness	Median Fitness
Control	1	1.95959596	1	1.141414141
5 Elites	1	2.15151515	1	1.262626263
5 Steady State Removals	1	1.87878788	1	1.121212121
5 Elites 5 Steady	1	2.13131313	1	1.262626263

State Removals				

Figure 33

Again, the final population for each experiment still lacks highly fit individuals, despite evolving them several times throughout the generations.

5.3.5 Island Populations

Appendix 10.10 contains the results in full.

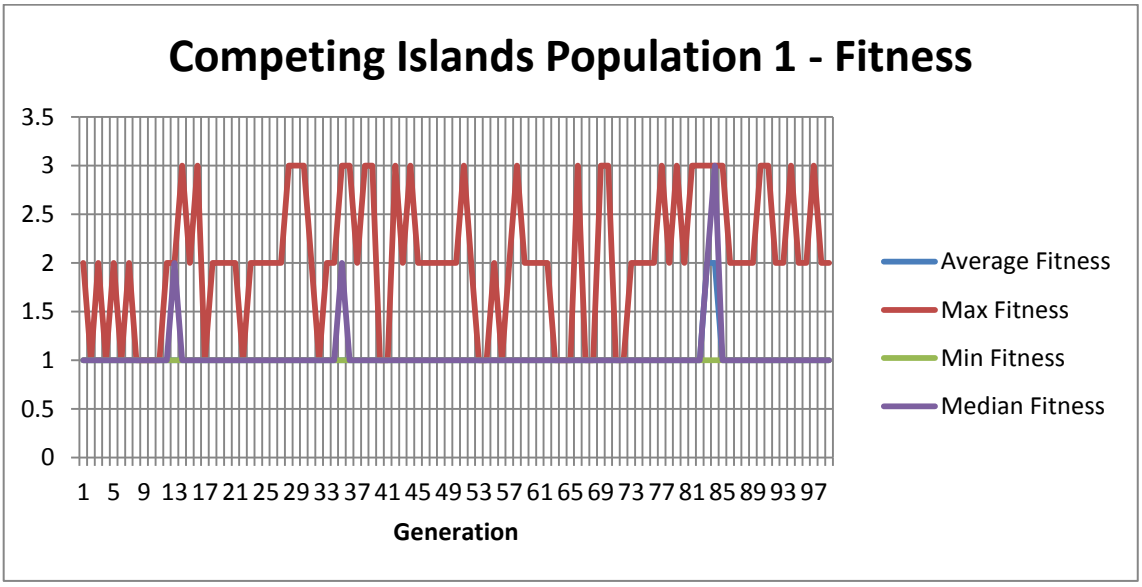

Figure 34

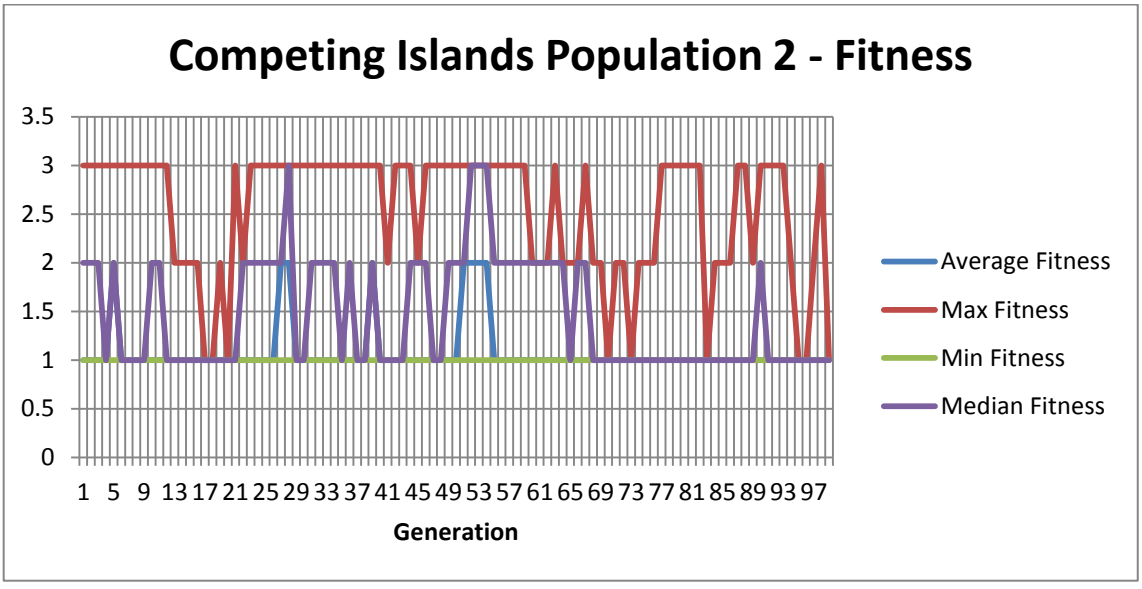

Figure 35

5.3.5.1 Analysis

	Average Statistics			
	Average Fitness	Max Fitness	Min Fitness	Median Fitness
Competing AI (1 versus 1)	1	2.46464646	1	1.060606061
Population 1	1.02020202	2.05050505	1	1.050505051
Population 2	1.080808081	2.54545455	1	1.454545455

Figure 36

From a statistical outlook, it is clear that population 2 has had a significantly higher median and max fitness throughout the generations (figure 36). It is also interesting that the regular competing AI evaluation method has a higher max and median fitness than population 1, but not population 2.

One possible explanation to this anomaly is that population 2 developed a highly fit set of individuals first, which was able to repeatedly defeat population 1. This explanation fits in with the population comparison in figure 37 and it also fits in with the hypothesis. From a qualitative perspective, a correlation may be apparent between population 1's max fitness and population 2's. This can be seen especially in the first 10 generations, where population 1's max fitness remains low whilst population 2's is constantly set at 3, then, population 1's max fitness rises to 3 just as population 2's max fitness falls to 1.

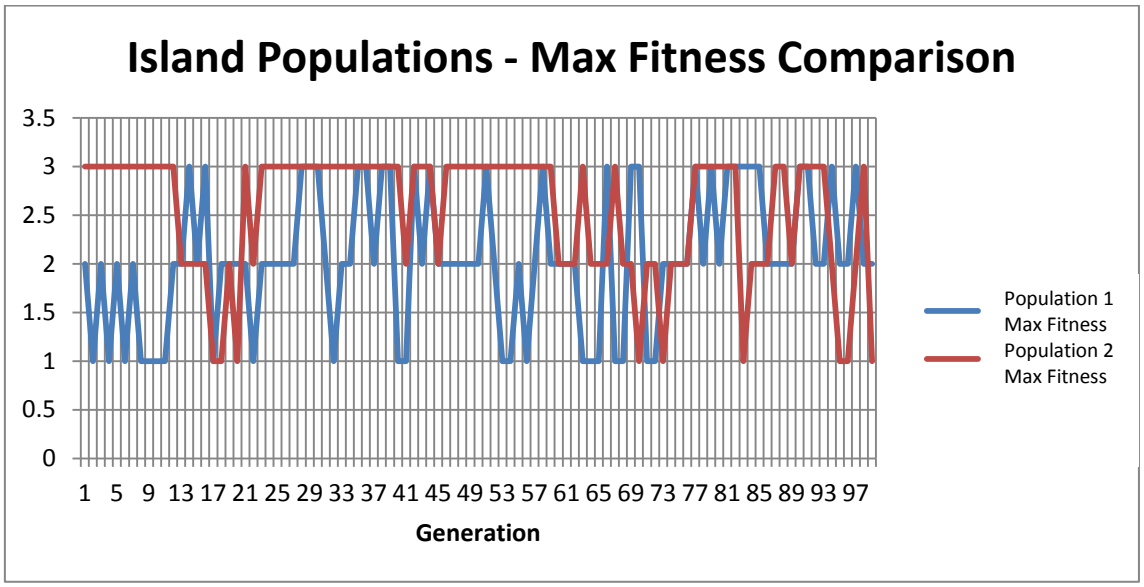

Figure 37

Analysing individuals for the separate population experiments appears to bring up the most interesting results, although like many of the other experiments, the final population remains unimpressive.

At moments such as the shift in fitness at generation 10, where population 1's max fitness overtakes population 2's, adaptation can be observed. Before generation 10, the solutions scoring a fitness of 3 in population 2 simply moved in one direction and jumped because it was a reliable strategy against individuals in population 1, who moved in one direction without jumping. At generation 11, 3 individuals scored a fitness of 3, at generation 12, only 1 individual scored a fitness of 3. Then, in the space of one generation, in population 1, generation 14, an individual won by using the same strategy of jumping and moving right that individuals in population 2 were using.

In theory, individuals that never scored a higher than average fitness, would likely be removed from the population eventually, it appears as though this might have been the case for the island populations experiment. Moving AI's may have repeatedly lost against jumping and moving AI's so roulette selection removed them to the point where population 2's tactic of moving right and jumping had a low chance of winning, and so, as they weren't able to achieve higher than average fitness, so their numbers dwindled as well.

In the study of ecosystems, the balance of nature is an idea which states that ecological systems are usually in a stable equilibrium (Pimm, 1991). Predator-prey equations are equations which model the interactions between predator populations and prey populations (Berryman, A. A., 1992). In a study of ocean life (The Ottawa Citizen, 2008), a chaotic balance of nature is observed in populations of sea creatures where populations multiply and decline in a cycle that seems relatable to the cycles that might exist in the island population experiment.

Chapter 6: Conclusion

The results of the project have given some insight into how evolutionary systems can share similarities with evolution and natural selection, it has also shown how complex that system is, by highlighting how difficult it can be just to evolve simplistic game AI. It is unfortunate that the final generation for each experiment displayed an uninspiring population but that doesn't mean the experiments were a failure, the aim was to gain insight, not optimise a system.

Even simplistic evolutionary algorithms, like the one implemented in this project have the power to display complex relationships. The most apparent and interesting being the potential predator-prey relationship in the island population experiment, in which population 2's fitness deterministically but also seemingly chaotically, raised and declined in response to population 1's. Further work should be based on harnessing these natural relationships to create highly adapted and self-maintained populations.

The largest obstacle in the way of the evolutionary algorithm was in accurately measuring an individual's fitness. Individuals were able to appear highly fit to the algorithm, when in fact, their opponents were just highly unfit. This is most apparent in the 1 versus 1 competing AI experiment, where bad strategies like moving in one direction and jumping were rewarded because the opponents practically gave them the win by moving underneath them. Projects in the future must be able to accurately measure fitness, as evolutionary algorithms prefer the easiest solution which is often to exploit the system.

The implementation of elite individuals displayed clear results, that the inclusion of 5 elites improved the overall fitness. Whether or not elite individuals are preferable in other problems will depend on the problem, but perhaps the findings can be transferred onto similarly defined problems. The qualities that made elites so preferable in the project's context was that it had issues keeping the best members in the population, the reasons it had this issue could be because of many different factors.

It is likely that a multitude of reasons caused good solutions to die out early, as is the case with evolutionary systems, in some cases, the best solutions are found be continuing to search through seemingly random solutions.

In conclusion, the project scope never would have enabled the research question to be answered fully. This is simply because each factor would need to be tested against every other which would require an unrealistic number of experiments. Another issue with testing evolutionary systems is that they could be simulated indefinitely, there is no set number of generations for a population

to be evolved for, and it is always possible that the best possible solution is just one generation away. This doesn't mean the project wasn't a success, or that the question wasn't answered. As we can see from the results, the populations reacted in unique ways to the different optimisations and in some cases, correlations and relationships could be postulated. The project has shown the potential for simulating accurate evolutionary systems, and also in its ability to come up with interesting and adaptive strategies.

Chapter 7: Reflective Analysis

This section contains a personal reflection on the process of completing the project. It will look into what went well, and what didn't, it will also highlight where the project was limited, how the project could be improved, and what research should be conducted in the future.

7.1 Strengths and Limitations

Evolutionary systems are very visually accessible; the development of a population over many generations can be easily transferred into graph form. It would have given more insight, if more statistics were gathered from the population at each stage. The population's convergence is an interesting metric that could only be objectively measured using the number of unique genotypes and number of unique fitness values, which almost always remained the same. Also, the lack of a precise fitness evaluation method meant that it didn't feel like the whole population was being measured, as fitness would usually stay between 1 and 3.

The number of factors measured also limited what could be said and observed about the evolutionary algorithm. The factors implemented only scratched the surface of alternative implementations, as can be seen from the vast amount of factors researched in the literature review.

Also, the design of experiments experiment design could have been improved drastically. Each factor was only compared against itself, and the control. Further insight may have been possible if factors were tested in different combinations.

The decision to implement the evolutionary algorithm, the game AI and a large portion of the game meant that less time could be spent dedicated to the research and experimentation sections. If more time was spent conducting experiments, more could be conducted, or a more detailed analysis could be written.

The use of existing software was attempted and failed because of the software's limitations. Many of the factors that were to be tested could only be done in certain combinations with other factors, and so to adapt the system to work in a way in which all factors could be tested against each other, may have required more work than it is to recreate the system from scratch.

The selection of the platform game, whilst it may have been for the best, also limited what could be achieved. In the simple platform game that was adapted, not much could be done about determining fitness; a player won, lost or drew. If the game rules were more complex and how much a player won was measurable, then fitness could be more effectively compared. In iteration

2 of the development of the evolutionary algorithm, the duration of the game was taken into account when evaluating fitness and it didn't disrupt the system. The decision to remove duration as a fitness identifier was done because its' inclusion is in itself a factor, and so would need to be tested separately. Inclusion of the duration would have given more precise fitness values, or at least fitness values that could be compared easily.

The decision to use genetic algorithms, or a rigid AI structure, meant that the AI was limited in its abilities. If genetic programming was used instead, more desirable behaviours may have arisen due to genetic programming's freedom in creating solutions. The decision tree structure that was implemented meant that AI's only had access to certain information and certain actions, in the 1 versus 1 versus 1 competing AI experiment, for example, AI's could only "see" the opponent closest to them. The potential for more advanced behaviours can also be seen in figure 14; 2 bits from each action node aren't used.

7.2 Future Work

Future work should focus on evolving games in environments where fitness can be evaluated quickly and precisely. With the continued improvement in computation speed, evolutionary algorithms should be able to develop strategies as quick as ever, tackling more and more complex problems.

As the island population experiment showed such interesting results, the experimentation of different types of island populations could yield more results. A predator-prey relationship is just one that exists in the natural world, attempting to simulate other relationships such as herbivore-plant predation, mutualism, commensalism, and parasitism.

The games AI generated in the project aren't nearly as sophisticated as human players and so a compromise should be looked into, between simple structures like decision trees which are fast but short-sighted, and neural networks which are highly adaptable but unspecialised.

Chapter 8: References

Alex J. Champandard. (2007). *10 Reasons the Age of Finite State Machines is Over.* Available: http://aigamedev.com/open/article/fsm-age-is-over/. Last accessed 23/03/2014.

Al-Khateeb, B., & Kendall, G. (2009, September). Introducing a round robin tournament into Blondie24. In Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on (pp. 112-116). IEEE.

Altenberg, L. (1994, June). Evolving better representations through selective genome growth. In Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on (pp. 182-187). IEEE.

Angeline, P. J., Saunders, G. M., & Pollack, J. B. (1994). An evolutionary algorithm that constructs recurrent neural networks. *Neural Networks, IEEE Transactions on*, *5*(1), 54-65.

Benbassat, A., & Sipper, M. (2011, July). Evolving board-game players with genetic programming. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation* (pp. 739-742). ACM.

Berryman, A. A. (1992). The orgins and evolution of predator-prey theory. *Ecology*, *73*(5), 1530-1535.

Bill Gates. (2014). *Hello Reddit – I'm Bill Gates, co-chair of the Bill & Melinda Gates Foundation and Microsoft founder. Ask me anything.* Available: http://www.reddit.com/r/IAmA/comments/1xj56q/hello_reddit_im_bill_gates_cochair_of_the_bill/. Last accessed 16/02/2014.

Bill Kendrick. (2013). *SuperTux* (Version 0.1.3) [Computer program]. Available at: http://supertux.lethargik.org (Accessed 20/04/2014).

Bonse, R., Kockelkorn, W., Smelik, R., Veelders, P., & Moerman, W. (2004). *Learning agents in quake iii*. Technical report, University of Utrecht, Department of Computer Science.

Cavicchio, D.J. (1970). *Adaptive search using simulated evolution.* Unpublished doctoral dissertation, University of Michigan, Ann Arbor.

Charles, D., & McGlinchey, S. (2004). The past, present and future of artificial neural networks in digital games. In Proceedings of the 5th international conference on computer games: artificial intelligence, design and education. The University of Wolverhampton (pp. 163-169).

Chrisantha Fernando. (2005). *The Standard Genetic Algorithm.* Available: http://www.sussex.ac.uk/Users/ctf20/dphil_2005/Publications/GeneticAlgorithm.ppt. Last accessed 08/04/2014.

Christian Gagn'e and Marc Parizeau . (2007) *Open BEAGLE* (Version 4.0.0) [Computer program]. Available at http://code.google.com/p/beagle/ (Accessed 22/04/2014)

Coello, C. A. C. (1999). A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information systems*,*1*(3), 269-308.

Colin Neville. (July 2007). *Introduction to Research and Research Methods.* Available: http://www.brad.ac.uk/management/media/management/els/Introduction-to-Research-and-Research-Methods.pdf. Last accessed 23/04/2014.

Crnkovic, I., Chaudron, M., & Larsson, S. (2006, October). Component-based development process and component lifecycle. In *Software Engineering Advances, International Conference on* (pp. 44-44). IEEE.

Daniel Dyer. (2013) *Watchmaker Framework* (Version 0.7.1) [Computer program]. Available at http://watchmaker.uncommons.org/support.php (Accessed 22/04/2014)

Dave Hadka. (2014) *MOEA Framework* (Version 2.1) [Computer program]. Available at http://www.moeaframework.org/ (Accessed 22/04/2014)

David Goldberg (1989). *Genetic Algorithms in Search, Optimization & Machine Learning*. Boston, United States: Addison-Wesley. 1.

De Jong, K. A. (1975). An analysis of the behaviour of a class of genetic adaptive systems. (Doctoral dissertation, University of Michigan). *Dissertation Abstracts International 36*(10), 5140B. (University Microfilms No. 76-9391)

Diaz-Gomez, P. A., & Hougen, D. F. (2007). Initial Population for Genetic Algorithms: A Metric Approach. In *GEM* (pp. 43-49).

Dr. Price & Dr. Oswald. (2008). *Within-Subjects Designs.* Available: http://psych.csufresno.edu/psy144/Content/Design/Experimental/within.html. Last accessed 23/04/2014.

Dworman, G., Kimbrough, S. O., & Laing, J. D. (1996, July). Bargaining by artificial agents in two coalition games: A study in genetic programming for electronic commerce. In *Proceedings of the First Annual Conference on Genetic Programming* (pp. 54-62). MIT Press.

Eiben, A. E., & Smith, J. E. (2003). *Introduction to evolutionary computing*. springer.

Elena Popovici. (2003). *Understanding Landscapes.* Available: http://www.cs.gmu.edu/~eclab/papers/lecture-pres/UnderstandingLandscapes.pdf. Last accessed 30/04/2014.

Fernandes, C., Tavares, R., Munteanu, C., & Rosa, A. (2001, March). Using assortative mating in genetic algorithms for vector quantization problems. In *Proceedings of the 2001 ACM symposium on Applied computing* (pp. 361-365). ACM.

Fogel, D. B. (1994). An introduction to simulated evolutionary optimization. *Neural Networks, IEEE Transactions on*, *5*(1), 3-14.

Fogel, D. B. (2000). Applying Fogel and Burgin'sCompetitive goal-seeking through evolutionary programming'to coordination, trust, and bargaining games. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on* (Vol. 2, pp. 1210-1216). IEEE.

Fogel, D. B. (2006). Nils barricelli-artificial life, coevolution, self-adaptation. *Computational Intelligence Magazine, IEEE*, *1*(1), 41-45.

Francisco, T., & Jorge dos Reis, G. M. (2008, July). Evolving combat algorithms to control space ships in a 2D space simulation game with co-evolution using genetic programming and decision trees. In *Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation* (pp. 1887-1892). ACM.

François-Michel De Rainville, Félix-Antoine Fortin, Christian Gagné, Olivier Gagnon, Marc-André Gardner, Simon Grenier, Yannick Hold-Geoffroy, and Marc Parizeau. (2012) *deap* (Version 1.1.0) [Computer program]. Available at http://code.google.com/p/deap/ (Accessed 22/04/2014)

Geijtenbeek, T., van de Panne, M., & van der Stappen, A. F. (2013). Flexible muscle-based locomotion for bipedal creatures. *ACM Transactions on Graphics (TOG)*, *32*(6), 206.

Geneura Team. (2014) *Evolving Objects* (Version 1.0) [Computer program]. Available at http://eodev.sourceforge.net/ (Accessed 22/04/2014)

Ghosh, A., & Dehuri, S. (2004). Evolutionary algorithms for multi-criterion optimization: A survey. *International Journal of Computing & Information Sciences*, *2*(1), 38-57.

Goldberg, D. E., & Richardson, J. (1987, July). Genetic algorithms with sharing for multimodal function optimization. In *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms* (pp. 41-49). Hillsdale, NJ: Lawrence Erlbaum.

Graham, R., McCabe, H., & Sheridan, S. (2003). Pathfinding in computer games. ITB Journal, 8.

Gustafson, S. M. (2004). *An analysis of diversity in genetic programming*(Doctoral dissertation, University of Nottingham).

Harvey, I. (1992). Species adaptation genetic algorithms: A basis for a continuing SAGA. In *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life* (pp. 346-354).

Hong, J. H., & Cho, S. B. (2006). The classification of cancer based on DNA microarray data that uses diverse ensemble genetic programming. *Artificial Intelligence in Medicine*, *36*(1), 43-58.

Hui, O. J., Teo, J., & On, C. K. (2011, October). Interactive evolutionary programming for mobile games rules generation. In Sustainable Utilization and Development in Engineering and Technology (STUDENT), 2011 IEEE Conference on (pp. 95-100). IEEE.

Jeroen Groeneweg. (2011). *Smash Battle* (Version 1.0) [Computer program]. Available at: http://smashbattle.demontpx.com/ (Accessed 20/04/2014.

Jon. (2010). 2D Platforming with Pygame. Available: http://www.nandnor.net/?p=64. Last accessed 18/03/2014.

Khan, A. I., Qurashi, R. J., & Khan, U. A. (2011). A Comprehensive Study of Commonly Practiced Heavy and Light Weight Software Methodologies. *International Journal of Computer Science Issues (IJCSI)*, *8*(4).

Klaus Meffert , Neil Rotstan. (2014) *JGAP* (Version 3.6.2) [Computer program]. Available at http://jgap.sourceforge.net/ (Accessed 22/04/2014)

Koza, J. R. (1990). Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Stanford University, Department of Computer Science.

Koza, J. R. (1992). Genetic Programming: vol. 1, On the programming of computers by means of natural selection (Vol. 1). MIT press.

Kreinovich, V., Quintana, C., & Fuentes, O. (1993). Genetic algorithms: what fitness scaling is optimal?. *Cybernetics and Systems*, *24*(1), 9-26.

Land, W. H., Albertelli, L., Titkov, Y., Kaltsatis, P., & Seburyano, G. (1998, May). Evolution of neural networks for the detection of breast cancer. InIntelligence and Systems, 1998. Proceedings., IEEE International Joint Symposia on (pp. 34-40). IEEE.

Langdon, W. B. (1998). Genetic programming and data structures: genetic programming+ data structures= automatic programming! (Vol. 1). Springer.

Laplante, P. A., & Neill, C. J. (2004). The demise of the Waterfall model is imminent. *Queue*, *1*(10), 10.

Lau, K. K., & Wang, Z. (2007). Software component models. *Software Engineering, IEEE Transactions on*, *33*(10), 709-724.

Lee, K. Y., & Yang, F. F. (1998). Optimal reactive power planning using evolutionary algorithms: A comparative study for evolutionary programming, evolutionary strategy, genetic algorithm, and linear programming. Power Systems, IEEE Transactions on, 13(1), 101-108.

Leung, Y., Gao, Y., & Xu, Z. B. (1997). Degree of population diversity-a perspective on premature convergence in genetic algorithms and its markov chain analysis. *Neural Networks, IEEE Transactions on*, *8*(5), 1165-1176.

Luke, S. (1998). Genetic programming produced competitive soccer softbot teams for robocup97. Genetic Programming, 1998, 214-222.

Lynch, B. (2006). Optimizing with Genetic Algorithms. Available: https://www.msi.umn.edu/sites/default/files/OptimizingWithGA.pdf. Last accessed 14/02/2014.

Manos, S., Poladian, L., Bentley, P. J., & Large, M. (2006, July). A genetic algorithm with a variable-length genotype and embryogeny for microstructured optical fibre design. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation* (pp. 1721-1728). ACM.

Mathworks. (2014). *Global Optimization Toolbox: User's Guide (r2011b).*Available: www.mathworks.com/help/pdf_doc/gads/gads_tb.pdf. Last accessed 07/04/2014.

Maxim Likhachev. (2011). *Intelligence I: Basic Decision-Making Mechanisms.* Available: http://www.cs.cmu.edu/~maxim/classes/CIS15466_Fall11/lectures/intelligenceI_cis15466.pdf. Last accessed 23/03/2014.

Michael Benisch. (2006). *Graduate AI Computational Game Theory.*Available: http://www.cs.cmu.edu/~ggordon/780-fall07/fall06/lectures/Game_theory_I.pdf. Last accessed 06/04/2014.

Mumford-Valenzuela, C. L. (2005). A simple approach to evolutionary multiobjective optimization. In *Evolutionary Multiobjective Optimization* (pp. 55-79). Springer London.

Old Dominion University. (2013). *Subjective vs Objective.* Available: http://www.lib.odu.edu/genedinfolit/1infobasics/subjective_vs_objective.html. Last accessed 23/04/2014.

Park, G. J. (2007). Design of experiments. *Analytic Methods for Design Practice*, 309-391.

Paul Prudence. (2012). Nils Barricelli's 5 Kilobyte Symbiogenesis simulations and 'molecule shaped numbers' – A precursor to DNA Computing. Available: http://www.dataisnature.com/?p=1448. Last accessed 14/02/2014.

Pimm, S. L. (1991). The balance of nature?: ecological issues in the conservation of species and communities. University of Chicago Press.

Python community. (2014) *inspyred* (Version 1.0) [Computer program]. Available at https://pypi.python.org/pypi/inspyred (Accessed 22/04/2014)

Rabin, S. (Ed.). (2002). AI game programming wisdom. Cengage Learning.

Roee Ben Halevi and Ronit Grinzaig. (2009). *Exercise 2.* Available: http://www.cs.bgu.ac.il/~grinzaig/ecal092/ex2/q1/ex2q1%20-%20main%20page.html. Last accessed 30/04/2014.

Rosca, J. P. (1995, March). Genetic Programming Exploratory Power and the Discovery of Functions. In *Evolutionary Programming* (pp. 719-736).

Rosca, J. P. (1996, July). Generality versus size in genetic programming. InProceedings of the First Annual Conference on Genetic Programming (pp. 381-387). MIT Press.

Sadjadi, F. (2004, October). Comparison of fitness scaling functions in genetic algorithms with applications to optical processing. In *Optical Science and Technology, the SPIE 49th Annual Meeting* (pp. 356-364). International Society for Optics and Photonics.

Sah, S. B. M., Ciesielski, V., D'Souza, D., & Berry, M. (2008). Comparison between genetic algorithm and genetic programming performance for photomosaic generation. In *Simulated Evolution and Learning* (pp. 259-268). Springer Berlin Heidelberg.

Schaffer, J. D. (1984). Some experiments in machine learning using vector evaluated genetic algorithms (artificial intelligence, optimization, adaptation, pattern recognition).

Sean Luke, Liviu Panait, Gabriel Balan, Sean Paus, Zbigniew Skolicki, Rafal Kicinger, Elena Popovici, Keith Sullivan, Joseph Harrison, Jeff Bassett, Robert Hubley, Ankur Desai, Alexander Chircop, Jack Compton, William Haddon, Stephen Donnelly, Beenish Jamil, Joseph Zelibor, Eric Kangas, Faisal Abidi, Houston Mooers, James O'Beirne, Khaled Ahsan Talukder, and James McDermott. (2014) *ECJ* (Version 21) [Computer program]. Available at http://cs.gmu.edu/~eclab/projects/ecj/ (Accessed 22/04/2014)

Segura, C., Coello, C. A. C., Miranda, G., & León, C. (2013). Using multi-objective evolutionary algorithms for single-objective optimization. *4OR*, *11*(3), 201-228.

Sipper, M., Azaria, Y., Hauptman, A., & Shichel, Y. (2006, September). Attaining human-competitive game playing with genetic programming. In *ACRI* (p. 13).

Souza, J. P. E., Alves, J. M., Damiani, J. H. S., & Silva, M. B. (2013). DESIGN OF EXPERIMENTS: ITS IMPORTANCE IN THE EFFICIENT PROJECT MANAGEMENT.

Spears, W. M., & Gordon, D. F. (2000). Evolving finite-state machine strategies for protecting resources. In *Foundations of Intelligent Systems* (pp. 166-175). Springer Berlin Heidelberg.

Spector, L., Barnum, H., Bernstein, H. J., & Swamy, N. (1999). 7 Quantum Computing Applications of Genetic Programming. *Advances in Genetic Programming*, *3*, 135.

Steve Gargolinski. (2005). *Genetic Algorithms for Game Programming.* Available: http://www.google.co.uk/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CC8QFjAA&url=http%3A%2F%2Fweb.cs.wpi.edu%2F~imgd1001%2Fc06%2Fsamples%2Fgargolinski.ppt&ei=jIMDU-HyDqar7Aa6voGgDw&usg=AFQjCNG5z05XX. Last accessed 18/02/2014.

Storn, R., & Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, *11*(4), 341-359.

The Ottawa Citizen (2008). Study of ocean life shows a "chaotic" balance of nature. Available: http://www.canada.com/ottawacitizen/news/story.html?id=96ce2ce4-0435-4905-b9d9-6be65efe773a&k=40047. Last accessed 30/04/2014.

Turbit, N. (2004). Project management & software development methodology.

Valerie J. Easton, John H. McColl, Stuart G. Young. (1997). *Statistics Glossary.* Available: http://www.stats.gla.ac.uk/steps/glossary/presenting_data.html. Last accessed 23/04/2014.

Chapter 9: Chapter 10: Appendices

10.1 Iteration 2 Testing

Generation	Average Fitness	Max Fitness	Generation	Average Fitness	Max Fitness	Generation	Average Fitness	Max Fitness
1	1	2	34	1	2	67	1	2
2	1	2	35	1	2	68	1	2
3	1	2	36	1	2	69	1	2
4	1	2	37	1	2	70	1	2
5	1	2	38	1	1	71	1	2
6	1	2	39	1	2	72	1	2
7	1	2	40	1	2	73	1	2
8	1	2	41	1	2	74	1	3
9	1	2	42	1	2	75	1	2
10	1	1	43	1	2	76	1	2
11	1	1	44	1	2	77	1	2
12	1	2	45	1	2	78	1	2
13	1	2	46	1	2	79	1	2
14	1	1	47	1	2	80	1	2
15	1	2	48	1	3	81	1	2
16	1	2	49	1	3	82	1	2
17	1	2	50	1	2	83	1	2
18	1	2	51	1	2	84	1	2
19	1	2	52	1	2	85	1	2
20	1	2	53	1	2	86	1	2
21	1	2	54	1	2	87	1	2
22	1	2	55	1	2	88	1	2
23	1	2	56	1	2	89	1	2
24	1	2	57	1	2	90	1	2
25	1	2	58	1	2	91	1	2
26	1	2	59	1	3	92	1	2
27	1	2	60	1	2	93	1	2
28	1	2	61	1	2	94	1	2
29	1	2	62	1	2	95	1	2
30	1	2	63	1	2	96	1	2
31	1	2	64	1	2	97	1	2
32	1	2	65	1	2	98	1	2
33	1	2	66	1	2	99	1	2
						100	1	2

10.2 Control Experiment Results

Generation	Average Fitness	Max Fitness	Min Fitness	Median Fitness	Unique Fitnesses	Unique Genotypes
1	1	2	1	1	2	90
2	1	2	1	1	2	90
3	1	2	1	1	2	90
4	1	2	1	1	2	90
5	1	2	1	1	2	90
6	1	1	1	1	1	90
7	1	2	1	1	2	90
8	1	2	1	1	2	90

9	1	2	1	1	2	90
10	1	3	1	1	3	90
11	1	2	1	1	2	90
12	1	2	1	1	2	90
13	1	2	1	1	2	90
14	1	1	1	1	1	90
15	1	2	1	1	2	90
16	1	2	1	1	2	90
17	1	2	1	1	2	90
18	1	1	1	1	1	90
19	1	1	1	1	1	90
20	1	2	1	1	2	90
21	1	2	1	2	2	90
22	1	2	1	2	2	90
23	1	2	1	2	2	90
24	1	2	1	2	2	90
25	1	2	1	2	2	90
26	1	2	1	2	2	90
27	1	2	1	2	2	90
28	1	2	1	2	2	90
29	1	2	1	1	2	90
30	1	1	1	1	1	90
31	1	2	1	1	2	90
32	1	1	1	1	1	90
33	1	1	1	1	1	90
34	1	2	1	1	2	90
35	1	2	1	1	2	90
36	1	2	1	1	2	90
37	1	2	1	1	2	90
38	1	2	1	1	2	90
39	1	2	1	1	2	90
40	1	2	1	1	2	90
41	1	2	1	1	2	90
42	1	2	1	1	2	90
43	1	2	1	1	2	90
44	1	2	1	1	2	90
45	1	2	1	1	2	90
46	1	2	1	1	2	90
47	1	2	1	1	2	90
48	1	2	1	1	2	90
49	1	2	1	1	2	90
50	1	2	1	1	2	89
51	1	2	1	1	2	89
52	1	2	1	1	2	90
53	1	2	1	1	2	90

54	1	2	1	2	2	90
55	1	2	1	2	2	90
56	1	2	1	1	2	90
57	1	2	1	1	2	90
58	1	3	1	2	3	90
59	1	3	1	1	3	90
60	1	3	1	2	3	90
61	1	2	1	1	2	90
62	1	2	1	1	2	90
63	1	3	1	2	3	90
64	1	2	1	1	2	90
65	1	2	1	1	2	90
66	1	2	1	1	2	90
67	1	2	1	1	2	90
68	1	2	1	1	2	90
69	1	2	1	2	2	90
70	1	2	1	1	2	89
71	1	2	1	1	2	90
72	1	2	1	1	2	90
73	1	2	1	1	2	90
74	1	2	1	1	2	90
75	1	2	1	1	2	90
76	1	2	1	1	2	90
77	1	2	1	1	2	90
78	1	2	1	1	2	90
79	1	2	1	1	2	90
80	1	2	1	1	2	90
81	1	2	1	1	2	90
82	1	2	1	1	2	90
83	1	2	1	1	2	90
84	1	2	1	1	2	90
85	1	2	1	1	2	90
86	1	2	1	1	2	90
87	1	1	1	1	1	90
88	1	2	1	1	2	90
89	1	2	1	1	2	90
90	1	1	1	1	1	90
91	1	2	1	1	2	90
92	1	2	1	1	2	90
93	1	2	1	1	2	90
94	1	2	1	1	2	90
95	1	2	1	1	2	90
96	1	2	1	1	2	90
97	1	2	1	1	2	90
98	1	2	1	1	2	90

99	1	2	1	1	2	90

10.3 Half Population Size Experiment Results

Generation	Average Fitness	Max Fitness	Min Fitness	Median Fitness	Unique Fitnesses	Unique Genotypes
1	1	2	1	1	2	45
2	1	2	1	1	2	45
3	1	2	1	2	2	45
4	1	2	1	1	2	45
5	1	2	1	1	2	45
6	1	2	1	1	2	45
7	1	2	1	1	2	45
8	1	2	1	1	2	45
9	1	2	1	1	2	45
10	1	2	1	1	2	45
11	1	2	1	1	2	45
12	1	2	1	1	2	45
13	1	2	1	1	2	45
14	1	2	1	1	2	45
15	1	2	1	1	2	45
16	1	2	1	1	2	45
17	1	2	1	1	2	45
18	1	2	1	1	2	45
19	1	2	1	1	2	45
20	1	2	1	1	2	45
21	1	2	1	1	2	45
22	1	2	1	1	2	45
23	1	2	1	2	2	45
24	1	2	1	1	2	45
25	1	2	1	1	2	45
26	1	2	1	1	2	45
27	1	2	1	1	2	45
28	1	2	1	1	2	45
29	1	3	1	1	3	45
30	1	2	1	1	2	45
31	1	2	1	1	2	45
32	1	2	1	1	2	45
33	1	2	1	1	2	45
34	1	2	1	1	2	45
35	1	2	1	1	2	45
36	1	2	1	1	2	45
37	1	2	1	1	2	45
38	1	2	1	1	2	45
39	1	2	1	1	2	45
40	1	2	1	1	2	45

41	1	2	1	1	2	45
42	1	2	1	1	2	45
43	1	1	1	1	1	45
44	1	2	1	1	2	45
45	1	2	1	1	2	45
46	1	1	1	1	1	45
47	1	1	1	1	1	45
48	1	2	1	1	2	45
49	1	1	1	1	1	45
50	1	2	1	1	2	45
51	1	1	1	1	1	45
52	1	1	1	1	1	45
53	1	1	1	1	1	45
54	1	2	1	1	2	45
55	1	2	1	1	2	45
56	1	2	1	1	2	45
57	1	2	1	2	2	45
58	1	2	1	2	2	45
59	1	3	1	2	3	45
60	1	3	1	2	3	45
61	1	2	1	2	2	45
62	1	2	1	1	2	45
63	1	3	1	1	3	45
64	1	2	1	1	2	45
65	1	2	1	2	2	45
66	1	2	1	1	2	45
67	1	2	1	2	2	45
68	1	2	1	1	2	45
69	1	2	1	2	2	45
70	1	2	1	2	2	45
71	1	2	1	2	2	45
72	1	2	1	2	2	45
73	1	2	1	2	2	45
74	1	2	1	1	2	45
75	1	2	1	1	2	45
76	1	2	1	1	2	45
77	1	2	1	1	2	45
78	1	2	1	1	2	45
79	1	2	1	1	2	45
80	1	2	1	1	2	45
81	1	1	1	1	1	45
82	1	1	1	1	1	45
83	1	2	1	1	2	45
84	1	2	1	1	2	45
85	1	2	1	1	2	45

86	1	1	1	1	1	45
87	1	1	1	1	1	45
88	1	2	1	1	2	45
89	1	2	1	1	2	45
90	1	2	1	1	2	45
91	1	2	1	1	2	45
92	1	3	1	2	3	45
93	1	3	1	2	3	45
94	1	3	1	1	3	45
95	1	2	1	1	2	45
96	1	2	1	1	2	45
97	1	2	1	1	2	45
98	1	1	1	1	1	45
99	1	2	1	1	2	45

10.4 Quarter Population Size Experiment Results

Generation	Average Fitness	Max Fitness	Min Fitness	Median Fitness	Unique Fitnesses	Unique Genotypes
1	1	2	1	1	2	22
2	1	2	1	1	2	22
3	1	2	1	1	2	22
4	1	2	1	1	2	22
5	1	2	1	2	2	22
6	1	2	1	2	2	22
7	1	2	1	1	2	22
8	1	2	1	1	2	22
9	1	2	1	2	2	22
10	1	2	1	2	2	22
11	1	1	1	1	1	22
12	1	1	1	1	1	22
13	1	1	1	1	1	22
14	1	1	1	1	1	22
15	1	2	1	1	2	22
16	1	2	1	1	2	22
17	1	1	1	1	1	22
18	1	2	1	1	2	22
19	1	2	1	1	2	22
20	1	2	1	1	2	22
21	1	2	1	1	2	22
22	1	2	1	1	2	22
23	1	1	1	1	1	22
24	1	2	1	1	2	22
25	1	2	1	2	2	22
26	1	1	1	1	1	22

27	1	2	1	1	2	22
28	1	2	1	1	2	22
29	1	2	1	1	2	22
30	1	2	1	1	2	22
31	1	2	1	1	2	22
32	1	2	1	1	2	22
33	1	2	1	1	2	22
34	1	2	1	1	2	22
35	1	2	1	1	2	22
36	1	1	1	1	1	22
37	1	2	1	1	2	22
38	1	1	1	1	1	22
39	1	2	1	1	2	22
40	1	2	1	1	2	22
41	1	2	1	1	2	22
42	1	2	1	1	2	22
43	1	2	1	1	2	22
44	1	2	1	1	2	22
45	1	2	1	1	2	22
46	1	2	1	1	2	22
47	1	2	1	1	2	22
48	1	2	1	1	2	22
49	1	2	1	1	2	22
50	1	2	1	2	2	22
51	1	2	1	2	2	22
52	1	2	1	1	2	22
53	1	2	1	2	2	22
54	1	2	1	1	2	22
55	1	2	1	1	2	22
56	1	2	1	1	2	22
57	1	2	1	2	2	22
58	1	2	1	2	2	22
59	1	1	1	1	1	22
60	1	2	1	1	2	22
61	1	2	1	1	2	22
62	1	2	1	1	2	22
63	1	2	1	1	2	22
64	1	2	1	2	2	22
65	1	2	1	2	2	22
66	1	2	1	2	2	22
67	1	2	1	2	2	22
68	1	2	1	2	2	22
69	1	3	1	2	3	22
70	1	2	1	1	2	22
71	1	1	1	1	1	22

72	1	1	1	1	1	22
73	1	2	1	1	2	22
74	1	2	1	1	2	22
75	1	1	1	1	1	22
76	1	2	1	1	2	22
77	1	1	1	1	1	22
78	1	2	1	1	2	22
79	1	2	1	1	2	22
80	1	2	1	1	2	22
81	1	2	1	1	2	22
82	1	2	1	1	2	22
83	1	1	1	1	1	22
84	1	1	1	1	1	22
85	1	2	1	1	2	22
86	1	2	1	1	2	22
87	1	1	1	1	1	22
88	1	1	1	1	1	22
89	1	2	1	1	2	22
90	1	2	1	2	2	22
91	1	2	1	2	2	22
92	1	1	1	1	1	22
93	1	2	1	1	2	22
94	1	1	1	1	1	22
95	1	1	1	1	1	22
96	1	2	1	1	2	22
97	1	2	1	1	2	22
98	1	2	1	1	2	22
99	1	2	1	1	2	22

10.5 Competitive AI – 1 versus 1 Experiment Results

Generation	Average Fitness	Max Fitness	Min Fitness	Median Fitness	Unique Fitnesses	Unique Genotypes
1	1	3	1	1	3	90
2	1	2	1	1	2	90
3	1	3	1	1	3	90
4	1	2	1	1	2	90
5	1	2	1	1	2	90
6	1	1	1	1	1	90
7	1	2	1	1	2	90
8	1	3	1	1	3	90
9	1	3	1	1	3	90
10	1	3	1	1	3	90
11	1	2	1	1	2	90
12	1	2	1	1	2	90

13	1	2	1	1	2	90
14	1	2	1	1	2	90
15	1	2	1	1	2	90
16	1	2	1	1	2	90
17	1	2	1	1	2	90
18	1	1	1	1	1	90
19	1	1	1	1	1	90
20	1	2	1	1	2	90
21	1	3	1	1	3	90
22	1	3	1	1	3	90
23	1	3	1	1	3	90
24	1	3	1	1	3	90
25	1	3	1	1	3	90
26	1	3	1	1	3	90
27	1	3	1	1	3	90
28	1	3	1	1	3	90
29	1	3	1	1	3	90
30	1	3	1	1	3	90
31	1	3	1	1	3	90
32	1	3	1	1	3	90
33	1	3	1	1	2	90
34	1	3	1	1	3	90
35	1	2	1	1	2	90
36	1	2	1	1	2	90
37	1	2	1	1	2	90
38	1	2	1	1	2	90
39	1	2	1	1	2	90
40	1	3	1	1	2	90
41	1	2	1	1	2	90
42	1	3	1	1	3	90
43	1	2	1	1	2	90
44	1	3	1	1	3	90
45	1	2	1	1	2	90
46	1	3	1	1	3	90
47	1	2	1	1	2	90
48	1	2	1	1	2	90
49	1	3	1	2	3	90
50	1	3	1	2	3	89
51	1	3	1	1	3	89
52	1	2	1	1	2	90
53	1	3	1	1	3	90
54	1	3	1	1	3	90
55	1	3	1	1	3	90
56	1	3	1	1	3	90
57	1	3	1	1	3	90

58	1	3	1	2	3	90
59	1	3	1	2	3	90
60	1	3	1	2	3	90
61	1	3	1	2	3	90
62	1	2	1	1	2	90
63	1	3	1	1	3	90
64	1	2	1	1	2	90
65	1	2	1	1	2	90
66	1	2	1	1	2	90
67	1	2	1	1	2	90
68	1	2	1	1	2	90
69	1	3	1	1	3	90
70	1	3	1	1	3	89
71	1	3	1	1	3	90
72	1	2	1	1	2	90
73	1	3	1	1	3	90
74	1	2	1	1	2	90
75	1	2	1	1	2	90
76	1	2	1	1	2	90
77	1	2	1	1	2	90
78	1	2	1	1	2	90
79	1	2	1	1	2	90
80	1	2	1	1	2	90
81	1	2	1	1	2	90
82	1	2	1	1	2	90
83	1	3	1	1	3	90
84	1	3	1	1	3	90
85	1	3	1	1	3	90
86	1	2	1	1	2	90
87	1	1	1	1	1	90
88	1	3	1	1	3	90
89	1	3	1	1	3	90
90	1	3	1	1	3	90
91	1	2	1	1	2	90
92	1	3	1	1	3	90
93	1	2	1	1	2	90
94	1	2	1	1	2	90
95	1	2	1	1	2	90
96	1	3	1	1	3	90
97	1	3	1	1	2	90
98	1	3	1	1	3	90
99	1	2	1	1	2	90

10.6 Competitive AI – 1 versus 1 versus 1 Experiment Results

Generation	Average Fitness	Max Fitness	Min Fitness	Median Fitness	Unique Fitness's	Unique Genotypes
1	1.622222222	4	1	1	4	90
2	1.7	5	1	1	5	90
3	1.677777778	6	1	1	6	90
4	1.477777778	5	1	1	5	89
5	1.555555556	5	1	1	5	89
6	1.4	4	1	1	4	90
7	1.555555556	5	1	1	5	90
8	1.222222222	3	1	1	3	90
9	1.333333333	4	1	1	4	90
10	1.4	4	1	1	4	89
11	1.566666667	4	1	1	4	90
12	1.4	4	1	1	4	90
13	1.6	4	1	1	4	90
14	1.455555556	4	1	1	4	90
15	1.477777778	5	1	1	5	90
16	1.566666667	4	1	1	4	90
17	1.5	5	1	1	5	90
18	1.544444444	4	1	1	4	90
19	1.466666667	4	1	1	4	90
20	1.3	3	1	1	3	90
21	1.211111111	3	1	1	3	90
22	1.266666667	4	1	1	4	90
23	1.366666667	4	1	1	4	90
24	1.222222222	4	1	1	4	90
25	1.422222222	5	1	1	5	90
26	1.322222222	4	1	1	4	88
27	1.188888889	3	1	1	3	90
28	1.288888889	3	1	1	3	90
29	1.266666667	3	1	1	3	90
30	1.344444444	4	1	1	4	90
31	1.488888889	6	1	1	5	90
32	1.377777778	3	1	1	3	90
33	1.533333333	5	1	1	5	90
34	1.511111111	5	1	1	5	90
35	1.377777778	6	1	1	5	90
36	1.422222222	3	1	1	3	90
37	1.322222222	3	1	1	3	90
38	1.544444444	5	1	1	5	90
39	1.488888889	5	1	1	5	90
40	1.411111111	5	1	1	5	90
41	1.611111111	5	1	1	5	90
42	1.566666667	4	1	1	4	90

43	1.511111111	5	1	1	5	90
44	1.411111111	4	1	1	4	90
45	1.522222222	5	1	1	5	89
46	1.677777778	6	1	1	6	90
47	1.555555556	5	1	1	4	90
48	1.522222222	5	1	1	5	90
49	1.522222222	5	1	1	5	90
50	1.422222222	5	1	1	5	90
51	1.511111111	4	1	1	4	90
52	1.433333333	5	1	1	5	90
53	1.577777778	3	1	1	3	90
54	1.7	5	1	1	5	90
55	1.688888889	5	1	1	5	89
56	1.644444444	4	1	1	4	90
57	1.777777778	5	1	1	5	90
58	1.766666667	6	1	2	6	89
59	1.755555556	4	1	1	4	89
60	1.611111111	5	1	1	5	90
61	1.688888889	4	1	1	4	90
62	1.744444444	6	1	1	6	90
63	1.744444444	5	1	1	5	90
64	1.588888889	4	1	1	4	90
65	1.777777778	5	1	2	5	90
66	1.766666667	5	1	1	5	90
67	1.744444444	5	1	1	5	90
68	1.522222222	5	1	1	5	90
69	1.655555556	4	1	1	4	90
70	1.688888889	6	1	1	5	90
71	1.7	5	1	1	5	90
72	1.611111111	6	1	1	5	89
73	1.811111111	5	1	1	5	90
74	1.733333333	4	1	1	4	90
75	1.8	5	1	1	5	90
76	1.7	5	1	1	5	90
77	1.744444444	6	1	1	5	90
78	1.8	5	1	2	5	89
79	1.677777778	4	1	1	4	90
80	1.655555556	4	1	1	4	90
81	1.688888889	5	1	1	5	90
82	1.644444444	4	1	1	4	90
83	1.711111111	4	1	2	4	90
84	1.555555556	6	1	1	5	90
85	1.655555556	4	1	1	4	90
86	1.477777778	4	1	1	4	90
87	1.522222222	5	1	1	5	90

88	1.511111111	5	1	1	5	89
89	1.655555556	5	1	1	5	90
90	1.622222222	5	1	1	5	90
91	1.333333333	5	1	1	4	90
92	1.5	5	1	1	5	90
93	1.355555556	4	1	1	4	90
94	1.544444444	6	1	1	6	90
95	1.622222222	6	1	1	6	90
96	1.488888889	4	1	1	4	89
97	1.677777778	5	1	1	5	90
98	1.555555556	5	1	1	5	89
99	1.755555556	7	1	1	7	89

10.7 5 Elites Experiment Results

Generation	Average Fitness	Max Fitness	Min Fitness	Median Fitness	Unique Fitnesses	Unique Genotypes
1	1	2	1	1	2	90
2	1	2	1	1	2	90
3	1	2	1	1	2	90
4	1	2	1	2	2	90
5	1	2	1	1	2	90
6	1	2	1	1	2	90
7	1	2	1	1	2	90
8	1	2	1	1	2	90
9	1	2	1	1	2	90
10	1	2	1	1	2	90
11	1	2	1	1	2	90
12	1	2	1	1	2	90
13	1	2	1	1	2	90
14	1	2	1	1	2	90
15	1	2	1	1	2	90
16	1	3	1	1	3	90
17	1	2	1	1	2	90
18	1	3	1	2	3	90
19	1	3	1	2	3	90
20	1	2	1	1	2	90
21	1	2	1	1	2	90
22	1	2	1	1	2	90
23	1	2	1	2	2	90
24	1	2	1	1	2	90
25	1	2	1	1	2	90
26	1	3	1	1	3	90
27	1	2	1	1	2	90
28	1	3	1	1	3	90

29	1	2	1	1	2	90
30	1	2	1	1	2	90
31	1	2	1	1	2	90
32	1	2	1	1	2	90
33	1	2	1	1	2	90
34	1	2	1	1	2	90
35	1	3	1	1	3	90
36	1	3	1	1	3	90
37	1	2	1	1	2	90
38	1	2	1	1	2	90
39	1	2	1	1	2	90
40	1	2	1	1	2	90
41	1	2	1	1	2	90
42	1	2	1	1	2	90
43	1	2	1	1	2	90
44	1	2	1	1	2	90
45	1	2	1	1	2	90
46	1	2	1	2	2	90
47	1	2	1	2	2	90
48	1	2	1	2	2	90
49	1	2	1	2	2	90
50	1	2	1	1	2	90
51	1	2	1	1	2	90
52	1	2	1	1	2	90
53	1	2	1	2	2	90
54	1	2	1	2	2	90
55	1	2	1	1	2	90
56	1	2	1	1	2	90
57	1	2	1	1	2	90
58	1	3	1	1	3	90
59	1	3	1	1	3	90
60	1	3	1	1	3	90
61	1	3	1	2	3	90
62	1	3	1	2	3	90
63	1	3	1	2	3	90
64	1	3	1	2	3	90
65	1	2	1	2	2	90
66	1	2	1	2	2	90
67	1	2	1	1	2	90
68	1	2	1	2	2	90
69	1	2	1	2	2	90
70	1	2	1	2	2	90
71	1	2	1	2	2	90
72	1	2	1	2	2	90
73	1	2	1	2	2	90

74	1	2	1	2	2	90
75	1	2	1	2	2	90
76	1	2	1	2	2	90
77	1	2	1	1	2	90
78	1	2	1	1	2	90
79	1	2	1	1	2	90
80	1	2	1	1	2	90
81	1	2	1	1	2	90
82	1	2	1	1	2	90
83	1	2	1	1	2	89
84	1	2	1	1	2	90
85	1	2	1	1	2	90
86	1	2	1	1	2	90
87	1	2	1	1	2	90
88	1	3	1	2	3	90
89	1	2	1	1	2	90
90	1	2	1	1	2	90
91	1	2	1	1	2	90
92	1	2	1	1	2	90
93	1	1	1	1	1	90
94	1	2	1	1	2	90
95	1	2	1	1	2	90
96	1	3	1	1	3	90
97	1	2	1	1	2	90
98	1	2	1	1	2	90
99	1	2	1	1	2	90

10.8 5 Steady State Removals Experiment Results

Generation	Average Fitness	Max Fitness	Min Fitness	Median Fitness	Unique Fitnesses	Unique Genotypes
1	1	2	1	1	2	90
2	1	2	1	1	2	90
3	1	2	1	1	2	90
4	1	2	1	1	2	90
5	1	2	1	1	2	90
6	1	1	1	1	1	90
7	1	2	1	1	2	90
8	1	2	1	1	2	90
9	1	2	1	1	2	90
10	1	3	1	1	3	90
11	1	2	1	1	2	90
12	1	2	1	1	2	90
13	1	2	1	1	2	90
14	1	1	1	1	1	90

15	1	2	1	1	2	90
16	1	2	1	1	2	90
17	1	2	1	1	2	90
18	1	1	1	1	1	90
19	1	1	1	1	1	90
20	1	2	1	1	2	90
21	1	2	1	2	2	90
22	1	2	1	2	2	90
23	1	2	1	2	2	90
24	1	2	1	2	2	90
25	1	2	1	2	2	90
26	1	2	1	2	2	90
27	1	2	1	2	2	90
28	1	2	1	2	2	90
29	1	2	1	1	2	90
30	1	1	1	1	1	90
31	1	2	1	1	2	90
32	1	1	1	1	1	90
33	1	1	1	1	1	90
34	1	2	1	1	2	90
35	1	2	1	1	2	90
36	1	2	1	1	2	90
37	1	2	1	1	2	90
38	1	2	1	1	2	90
39	1	2	1	1	2	90
40	1	3	1	1	3	90
41	1	2	1	1	2	90
42	1	2	1	1	2	90
43	1	2	1	1	2	90
44	1	2	1	1	2	90
45	1	2	1	1	2	90
46	1	2	1	1	2	90
47	1	2	1	1	2	90
48	1	1	1	1	1	90
49	1	2	1	1	2	90
50	1	2	1	1	2	89
51	1	2	1	1	2	89
52	1	1	1	1	1	90
53	1	2	1	1	2	90
54	1	2	1	1	2	90
55	1	2	1	1	2	90
56	1	2	1	1	2	90
57	1	2	1	1	2	90
58	1	2	1	1	2	90
59	1	2	1	1	2	90

60	1	2	1	1	2	90
61	1	2	1	1	2	90
62	1	2	1	1	2	90
63	1	2	1	1	2	90
64	1	2	1	1	2	90
65	1	2	1	1	2	90
66	1	2	1	1	2	90
67	1	2	1	1	2	90
68	1	2	1	1	2	90
69	1	3	1	2	3	90
70	1	2	1	1	2	89
71	1	2	1	1	2	90
72	1	2	1	1	2	90
73	1	2	1	1	2	90
74	1	2	1	1	2	90
75	1	2	1	2	2	90
76	1	2	1	2	2	90
77	1	2	1	1	2	90
78	1	2	1	2	2	90
79	1	2	1	1	2	90
80	1	2	1	1	2	90
81	1	2	1	1	2	90
82	1	2	1	1	2	90
83	1	2	1	1	2	90
84	1	2	1	1	2	90
85	1	2	1	1	2	90
86	1	2	1	1	2	90
87	1	2	1	1	2	90
88	1	2	1	1	2	90
89	1	2	1	1	2	90
90	1	1	1	1	1	90
91	1	2	1	1	2	90
92	1	2	1	1	2	90
93	1	1	1	1	1	90
94	1	1	1	1	1	90
95	1	1	1	1	1	90
96	1	2	1	1	2	90
97	1	1	1	1	1	90
98	1	1	1	1	1	90
99	1	2	1	1	2	90

10.9 5 Elites 5 Steady State Removals Experiment Results

Generation	Average Fitness	Max Fitness	Min Fitness	Median Fitness	Unique Fitnesses	Unique Genotypes

1	1	2	1	1	2	90
2	1	2	1	1	2	90
3	1	2	1	1	2	90
4	1	2	1	2	2	90
5	1	2	1	1	2	90
6	1	2	1	1	2	90
7	1	2	1	1	2	90
8	1	2	1	1	2	90
9	1	2	1	1	2	90
10	1	2	1	1	2	90
11	1	2	1	1	2	90
12	1	2	1	1	2	90
13	1	2	1	1	2	90
14	1	2	1	1	2	90
15	1	2	1	1	2	90
16	1	3	1	1	3	90
17	1	2	1	1	2	90
18	1	3	1	2	3	90
19	1	3	1	2	3	90
20	1	2	1	1	2	90
21	1	2	1	1	2	90
22	1	2	1	1	2	90
23	1	2	1	2	2	90
24	1	2	1	1	2	90
25	1	2	1	1	2	90
26	1	3	1	1	3	90
27	1	2	1	1	2	90
28	1	3	1	1	3	90
29	1	2	1	1	2	90
30	1	2	1	1	2	90
31	1	2	1	1	2	90
32	1	2	1	1	2	90
33	1	2	1	1	2	90
34	1	2	1	1	2	90
35	1	3	1	1	3	90
36	1	3	1	1	3	90
37	1	2	1	1	2	90
38	1	2	1	1	2	90
39	1	2	1	1	2	90
40	1	2	1	1	2	90
41	1	2	1	1	2	90
42	1	2	1	1	2	90
43	1	2	1	1	2	90
44	1	2	1	1	2	90
45	1	2	1	1	2	90

46	1	2	1	2	2	90
47	1	2	1	2	2	90
48	1	2	1	2	2	90
49	1	2	1	2	2	90
50	1	2	1	1	2	90
51	1	2	1	1	2	90
52	1	2	1	1	2	90
53	1	2	1	2	2	90
54	1	2	1	2	2	90
55	1	2	1	1	2	90
56	1	2	1	1	2	90
57	1	3	1	1	3	90
58	1	2	1	1	2	90
59	1	2	1	1	2	90
60	1	2	1	1	2	90
61	1	2	1	1	2	90
62	1	2	1	1	2	90
63	1	3	1	1	3	90
64	1	2	1	2	2	90
65	1	2	1	2	2	90
66	1	2	1	2	2	90
67	1	2	1	1	2	90
68	1	2	1	2	2	90
69	1	2	1	1	2	90
70	1	2	1	1	2	90
71	1	2	1	1	2	90
72	1	2	1	1	2	90
73	1	2	1	1	2	90
74	1	2	1	2	2	90
75	1	3	1	2	3	90
76	1	3	1	2	3	90
77	1	3	1	2	3	90
78	1	2	1	1	2	90
79	1	3	1	1	3	90
80	1	2	1	2	2	90
81	1	2	1	2	2	90
82	1	2	1	2	2	90
83	1	2	1	1	2	89
84	1	2	1	1	2	90
85	1	2	1	1	2	90
86	1	2	1	2	2	90
87	1	2	1	1	2	90
88	1	2	1	2	2	90
89	1	2	1	2	2	90
90	1	2	1	2	2	90

91	1	2	1	2	2	90
92	1	2	1	1	2	90
93	1	2	1	1	2	90
94	1	2	1	1	2	90
95	1	2	1	1	2	90
96	1	2	1	1	2	90
97	1	2	1	1	2	90
98	1	2	1	1	2	90
99	1	2	1	1	2	90

10.10 Island Populations Experiment Results

10.10.1 Population 1

Generation	Average Fitness	Max Fitness	Min Fitness	Median Fitness	Unique Fitnesses	Unique Genotypes
1	1	2	1	1	2	45
2	1	1	1	1	1	45
3	1	2	1	1	2	45
4	1	1	1	1	1	45
5	1	2	1	1	2	45
6	1	1	1	1	1	45
7	1	2	1	1	2	45
8	1	1	1	1	1	45
9	1	1	1	1	1	45
10	1	1	1	1	1	45
11	1	1	1	1	1	45
12	1	2	1	1	2	45
13	1	2	1	2	2	45
14	1	3	1	1	3	44
15	1	2	1	1	2	45
16	1	3	1	1	3	45
17	1	1	1	1	1	45
18	1	2	1	1	2	45
19	1	2	1	1	2	45
20	1	2	1	1	2	45
21	1	2	1	1	2	45
22	1	1	1	1	1	45
23	1	2	1	1	2	45
24	1	2	1	1	2	45
25	1	2	1	1	2	45
26	1	2	1	1	2	45
27	1	2	1	1	2	45
28	1	3	1	1	3	45
29	1	3	1	1	3	45

30	1	3	1	1	3	45
31	1	2	1	1	2	45
32	1	1	1	1	1	45
33	1	2	1	1	2	45
34	1	2	1	1	2	45
35	1	3	1	2	3	45
36	1	3	1	1	3	45
37	1	2	1	1	2	45
38	1	3	1	1	3	45
39	1	3	1	1	2	45
40	1	1	1	1	1	45
41	1	1	1	1	1	45
42	1	3	1	1	3	45
43	1	2	1	1	2	45
44	1	3	1	1	3	45
45	1	2	1	1	2	45
46	1	2	1	1	2	45
47	1	2	1	1	2	45
48	1	2	1	1	2	45
49	1	2	1	1	2	45
50	1	2	1	1	2	45
51	1	3	1	1	3	45
52	1	2	1	1	2	45
53	1	1	1	1	1	45
54	1	1	1	1	1	45
55	1	2	1	1	2	45
56	1	1	1	1	1	45
57	1	2	1	1	2	45
58	1	3	1	1	3	45
59	1	2	1	1	2	45
60	1	2	1	1	2	45
61	1	2	1	1	2	45
62	1	2	1	1	2	45
63	1	1	1	1	1	45
64	1	1	1	1	1	45
65	1	1	1	1	1	45
66	1	3	1	1	3	45
67	1	1	1	1	1	45
68	1	1	1	1	1	45
69	1	3	1	1	3	45
70	1	3	1	1	3	45
71	1	1	1	1	1	45
72	1	1	1	1	1	45
73	1	2	1	1	2	45
74	1	2	1	1	2	45

75	1	2	1	1	2	45
76	1	2	1	1	2	45
77	1	3	1	1	3	45
78	1	2	1	1	2	45
79	1	3	1	1	3	45
80	1	2	1	1	2	45
81	1	3	1	1	3	45
82	1	3	1	1	2	45
83	2	3	1	2	3	45
84	2	3	1	3	3	45
85	1	3	1	1	3	45
86	1	2	1	1	2	45
87	1	2	1	1	2	45
88	1	2	1	1	2	45
89	1	2	1	1	2	45
90	1	3	1	1	3	45
91	1	3	1	1	3	45
92	1	2	1	1	2	45
93	1	2	1	1	2	45
94	1	3	1	1	3	45
95	1	2	1	1	2	45
96	1	2	1	1	2	45
97	1	3	1	1	3	45
98	1	2	1	1	2	45
99	1	2	1	1	2	45

10.10.2 Population 2

Generation	Average Fitness	Max Fitness	Min Fitness	Median Fitness	Unique Fitnesses	Unique Genotypes
1	1	3	1	2	3	45
2	1	3	1	2	3	45
3	1	3	1	2	3	45
4	1	3	1	1	3	45
5	2	3	1	2	3	45
6	1	3	1	1	3	45
7	1	3	1	1	3	45
8	1	3	1	1	3	45
9	1	3	1	1	3	45
10	1	3	1	2	3	45
11	1	3	1	2	3	45
12	1	3	1	1	3	45
13	1	2	1	1	2	45
14	1	2	1	1	2	45
15	1	2	1	1	2	45
16	1	2	1	1	2	45

17	1	1	1	1	1	45
18	1	1	1	1	1	45
19	1	2	1	1	2	45
20	1	1	1	1	1	45
21	1	3	1	1	3	45
22	1	2	1	2	2	45
23	1	3	1	2	3	45
24	1	3	1	2	3	45
25	1	3	1	2	3	45
26	1	3	1	2	3	45
27	2	3	1	2	3	45
28	2	3	1	3	3	45
29	1	3	1	1	3	45
30	1	3	1	1	3	45
31	1	3	1	2	3	45
32	1	3	1	2	3	45
33	1	3	1	2	3	45
34	1	3	1	2	3	45
35	1	3	1	1	3	45
36	1	3	1	2	3	45
37	1	3	1	1	3	45
38	1	3	1	1	3	45
39	2	3	1	2	3	45
40	1	3	1	1	3	45
41	1	2	1	1	2	45
42	1	3	1	1	3	45
43	1	3	1	1	3	45
44	1	3	1	2	3	45
45	1	2	1	2	2	45
46	1	3	1	2	3	45
47	1	3	1	1	3	45
48	1	3	1	1	3	45
49	1	3	1	2	3	45
50	1	3	1	2	3	45
51	2	3	1	2	3	45
52	2	3	1	3	3	45
53	2	3	1	3	3	45
54	2	3	1	3	3	45
55	1	3	1	2	3	45
56	1	3	1	2	3	45
57	1	3	1	2	3	45
58	1	3	1	2	3	45
59	1	3	1	2	3	45
60	1	2	1	2	2	45
61	1	2	1	2	2	45

62	1	2	1	2	2	45
63	1	3	1	2	3	45
64	1	2	1	2	2	45
65	1	2	1	1	2	45
66	1	2	1	2	2	45
67	1	3	1	2	3	45
68	1	2	1	1	2	45
69	1	2	1	1	2	45
70	1	1	1	1	1	45
71	1	2	1	1	2	45
72	1	2	1	1	2	45
73	1	1	1	1	1	45
74	1	2	1	1	2	45
75	1	2	1	1	2	45
76	1	2	1	1	2	45
77	1	3	1	1	2	45
78	1	3	1	1	2	45
79	1	3	1	1	3	45
80	1	3	1	1	3	45
81	1	3	1	1	3	45
82	1	3	1	1	3	45
83	1	1	1	1	1	45
84	1	2	1	1	2	45
85	1	2	1	1	2	45
86	1	2	1	1	2	45
87	1	3	1	1	3	45
88	1	3	1	1	2	45
89	1	2	1	1	2	45
90	1	3	1	2	3	45
91	1	3	1	1	3	45
92	1	3	1	1	3	45
93	1	3	1	1	3	45
94	1	2	1	1	2	45
95	1	1	1	1	1	45
96	1	1	1	1	1	45
97	1	2	1	1	2	45
98	1	3	1	1	3	45
99	1	1	1	1	1	45