Evolutionary Algorithm Animator.

Developing a GUI based animator for illustrating evolutionary algorithms in operation.

Oana Furtuna

16025091

**STATEMENT OF ORIGINALITY**

**CS3D661 Individual Project**

This is to certify that, except where specific reference is made, the work described within this project is the result of the investigation carried out by myself, and that neither this project, nor any part of it, has been submitted in candidature for any other award other than this being presently studied.

Any material taken from published texts or computerized sources have been fully referenced, and I fully realize the consequences of plagiarizing any of these sources.

Student Name (Printed) OANA MADALINA FURTUNA

Student Signature Oana Furtuna

Registered Course of Study BSc Computer Games Development

Date of Signing 29/03/2019

Abstract

This paper presents a set of methods and techniques used to create an evolving entity by applying Genetic Algorithms (GA’s). GA’s are based on Darwin’s theory of evolution and they have been first implemented in an artificial environment in the 1960’s by John Holland. Over the years, they have been used and proved to be successful in difficult optimization problems which involve a large search space or varying input values. This paper looks at how different implementation techniques of genetic algorithms can be used to evolve individuals and their behaviours. This is done using a graph crossover approach and by applying a roulette wheel selection. This selection method is used to let the creatures make their own decisions when it comes to choosing the best behaviours in response to certain events. Results show that over a number of generations, the average fitness of each population increases, and the individuals develop new behaviours.

Table of Contents

[1 Introduction, Aim and Objectives 5](#_Toc4792152)

[1.1 Introduction 5](#_Toc4792153)

[1.2 Aim 6](#_Toc4792154)

[1.3 Objectives 6](#_Toc4792155)

[1.4 Deliverables 6](#_Toc4792156)

[2 Background Research 7](#_Toc4792157)

[2.1 Background 7](#_Toc4792158)

[2.2 Optimization 7](#_Toc4792159)

[2.3 Genetic Algorithms 8](#_Toc4792160)

[2.4 Genetic Operators 8](#_Toc4792161)

[2.5 Genetic Algorithms Paradigms of Evolutionary Computation 9](#_Toc4792162)

[2.6 Genetic Algorithm Encoding 10](#_Toc4792163)

[2.7 Structure of a Genetic Algorithm 11](#_Toc4792164)

[2.7.1 Initial Population 11](#_Toc4792165)

[2.7.2 Fitness Function 12](#_Toc4792166)

[2.7.3 Selection 12](#_Toc4792167)

[2.7.4 Crossover 14](#_Toc4792168)

[2.7.5 Mutation 16](#_Toc4792169)

[2.7.6 New Population 17](#_Toc4792170)

[2.8 Genetic Algorithms in Video Games 17](#_Toc4792171)

[3 Methodology 18](#_Toc4792172)

[3.1 Technologies 18](#_Toc4792173)

[3.1.1 Software 18](#_Toc4792174)

[3.2 Code Design 18](#_Toc4792175)

[3.2.1 Critter 19](#_Toc4792176)

[3.2.2 Herbivore and Carnivore controllers 19](#_Toc4792177)

[3.2.3 Vegetable, Tree and Dirt Controllers 20](#_Toc4792178)

[3.2.4 Age Controller 20](#_Toc4792179)

[3.2.5 Breeding Controller 20](#_Toc4792180)

[3.2.6 AI 21](#_Toc4792181)

[3.2.7 Seek 22](#_Toc4792182)

[3.3 Implementation 22](#_Toc4792183)

[3.3.1 Finite State Machine 22](#_Toc4792184)

[3.3.2 Genetic Algorithm 24](#_Toc4792185)

[4 Testing and Results 27](#_Toc4792186)

[4.1 Population Size 27](#_Toc4792187)

[4.2 Average Fitness Score 28](#_Toc4792188)

[4.3 Highest Fitness Score 29](#_Toc4792189)

[4.4 Available Behaviours 30](#_Toc4792190)

[5 Conclusion 32](#_Toc4792191)

[6 Future Work 33](#_Toc4792192)

[7 Bibliography 34](#_Toc4792193)

[8 Appendices 36](#_Toc4792194)

[8.1 Appendix 1: Ethics Form 36](#_Toc4792195)

[8.2 Appendix 2: Plan Milestone 1 40](#_Toc4792196)

[8.3 Appendix 3: Plan Milestone 2 41](#_Toc4792197)

[8.4 Appendix 4: Unity Assets Sources 42](#_Toc4792198)

[8.5 Appendix 5: Meeting Notes 42](#_Toc4792199)

# Introduction, Aim and Objectives

## Introduction

In 1975, John Holland applied the fundamental concepts of Darwin’s Theory of Evolution to a number of optimization problems. His implementation has then been studied and further developed by one of his students, Goldberg, and applied to numerous strings containing a wide variety of symbols. This work is what is called today a Genetic Algorithm. In 1992, Koza applied the same principles to a set of programs represented through hierarchical trees. His work is referred to as genetic programming. These two techniques are very similar, as they both use crossover and mutation as genetic operators to solve optimization problems.

Genetic algorithms attempt to recreate the natural evolution in an artificial environment, their goal being to generate a highly fit solution to the given problem. This approach has been proved to be highly successful when applied to optimization problems, and are especially effective when they involve a large, complex search space. This done using something similar to the following algorithm (Al Globus, 2000):

1. Randomly generate a population of individual potential solutions.

2. Evaluate each individual using a fitness function.

3. For each new generation, repeatedly select parent individuals at random with a bias towards better individuals and create children by applying transmission operators. Transmission operators may include:

1. Crossover: each of two parents is divided into two parts and one part from each parent is combined into a child.

2. Mutation: a single ‘parent’ is randomly modified to create a child.

3. Reproduction: a single ‘parent’ is copied into the new generation.

4. Continue until an acceptable solution is found or exhaustion sets in.

The purpose of this project is to research genetic algorithms and to try and use them to evolve a finite state machine (graph), by adding or deleting certain states (vertices) or possible events (edges).

## Aim

To evolve creatures and their behaviours using Genetic Algorithms.

## Objectives

1. Research relevant design techniques

2. Examine tools suitable for creating the application

3. Research genetic algorithms and various implementation approaches

4. Design appropriate coding pattern

5. Implement the Genetic Algorithm features

6. Test the application

7. Evaluate and review the application

## Deliverables

This thesis aims to deliver a working simulation of an ecosystem in which the creatures and their behaviours evolve over time. This process is implemented using genetic algorithm techniques.

# Background Research

## Background

Solving problems involving the use of mathematical expressions has been extensively studied over the last few decades. One of the biggest challenges that scientists faced was finding answers to problems without having to program them completely. The main idea was to have a system that takes in information about the given problem and produces a program that solves it, without having the programmers tell the computer how to do it (Koza, et al., 1999).

Problem solving using evolutionary methods has its roots in the 1950s when several researchers took over the evolutionary metaphor of nature and applied it to solve various problems from different domains.

In 1948, Alan Turing published the essay “Intelligent Machinery” in which he talked about the possibility of implementing human intelligence in a machine. Through his analogy with the human brain, Turing explained how the machine can evolve over time if it is provided with suitable information. Furthermore, he promoted the idea of a computer solving a given problem without having to tell it exactly how to do it, but merely just “educating” it, using rewards and punishments (Turing, 1948).

Two years later, in 1950, Alan Turing published another paper, “Computing Machinery and Intelligence” in which he looked at how an intelligent machine could benefit from having different elements from genetics, evolution and natural selection (Turing, 1950).

Turing’s attempts to create evolving programs were overlooked until the 1960s when John Holland began working on what is called today a Genetic Algorithm. The method consisted of having a number of individuals evolving over time, based on a certain set of rules, with the ultimate goal of improving their fitness (Haupt & Haupt, 1998). His work was further developed by one of his students, David Goldberg, who received an NSF Presidential Young Investigator Award for his dissertation on studying how genetic algorithms and classifier systems can be used to control gas-pipeline transmissions (Goldberg, 1989).

## Optimization

In mathematics, optimization is defined as “a mathematical technique for finding a maximum or minimum value of a function of several variables subject to a set of constraints, as linear programming or systems analysis.” (Optimization, n.d.) In other words, optimization is the process of choosing the right inputs in order to find the best possible outputs.

In most cases, a problem that requires optimization will have more than one possible answer. However, not all of them have the same quality and sometimes the solution can be ambiguous. Some problems, like adding up two integers or determining the size of a football field, have exact solutions and clearly defined methods used to achieve the best possible answer. Other problems, like determining the best novel, have only optimal solutions without a clear, definitive answer. The results to these problems are also known as optimal points or extrema and are characterized by having minimized or maximized output values. (Haupt & Haupt, 1998)

## Genetic Algorithms

Genetic algorithms (GA) are a sub-class of evolutionary algorithms, based on Darwin’s Theory of Evolution. In his book “On the Origin of Species”, 1859, Darwin introduces the ideas of natural selection, evolution and survival of the fittest, those concepts becoming the basis of GA.

Genetic algorithms work with a population of possible solutions, represented through a vector, array or list. The individuals of the initial population are randomly generated and with the use of selection and other genetic operators, they evolve and eventually turn into optimized solutions to the given problem. The entities in the population are evaluated based on a fitness (quality) function, which represents life expectancy. The better an individual performs, the more likely it is to have a longer life expectancy and a higher fitness function.

The individuals in the population are subject to genetic manipulation. The genetic operators used depend on the representation, the most popular ones being crossover and mutation. In most cases, through crossover, two parents produce two offspring which inherit traits, characteristics and behaviours from both parents. Through mutation, a small alteration of an individual’s genetic code is applied. The individuals have a certain chance of being chosen for the application of genetic operators, the way and order of which they are selected varying.

For a program to be considered a genetic algorithm, it must cover three basic principles (Goldberg, 1989):

1. Variation – There must exist a population consisting of individuals with a variety of traits and characteristics. If the individuals are very similar, their offspring will be the same and the evolution will be very slow or non-existent.
2. Heredity – When an offspring is created, it inherits the traits and characteristics from its parents. By having the individual live enough to reproduce, it will pass on its good properties thus giving its offspring a good chance in life.
3. Selection – There must be a system so that the most skilled individuals are selected for breeding. In time, the old individuals are removed from the population so if the offspring have good properties, the population’s average fitness grows.

## Genetic Operators

Genetic operators are methods that operate on the individuals in the population. There are two main genetic operators:

* A unary transformation operator called mutation, which applies a small change in an offspring’s genetic code;
* A stronger operator called crossover, which takes two or more parents and combines their genetic code in order to create new individuals.

After a couple of generations, the algorithm converges, and the fittest individual reaches a value as close as possible to the optimal solution.

Switching from one generation to the next one can be done in two ways (Champandard, 2004):

* Generational algorithm, where the new population is made of offspring only;
* Steady-state algorithm, where the offspring and the most gifted individuals from the old generation are brought together to form a new generation.

It is also possible to create a few offspring that will replace the least fit individuals. As a general rule, it is preferable to use an elitist model that keeps the fittest individuals from the current generation and automatically adds them to the next one. This is so that if the fittest individual is not used in the selection process or if its qualities are lost when applying the genetic operators, the system will automatically add it to the next generation thus preserving the best traits in the population. This method is very useful for solving a wide range of optimization problems. However, the fundamental components of an evolutionary algorithm are the encoding methods used for representing the data and the genetic operators. Those are the main elements used to differentiate between the different paradigm variants of evolutionary systems (Dasgupta & Michalewicz, 1997).

## Genetic Algorithms Paradigms of Evolutionary Computation

As the name suggest, the foundation of genetic algorithms is borrowed from the genetic branch of biology. Some fundamental principles of natural genetics are borrowed and used artificially to create robust search algorithms that only need minimum information about the problem in order to solve it.

The array of possible solutions kept by the algorithm represents the population, while the individuals represent the chromosomes. Chromosomes contain units named genes that are arranged in a straight line. The genes dictate which characters the offspring inherit. The genes are placed in certain locations in the chromosome, named locus, and the possible values of a gene form the gene’s set of alleles (Goldberg, 1989).

The general format of the genetic code is represented through a genome. A genotype is the specific information in the genome of a particular individual in a very compact format. A group of individuals from the same species will have identical genomes and similar genotypes, with each gene taking the same position in the chromosome. Figure 1 (Champandard, 2004) shows how the genotype is laid out.

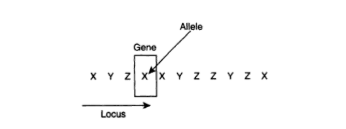


Figure 1 Conceptual representation of a genotype, displayed as a sequence of letters representing genes. (Champandard, 2004, p. 425)

The phenome is the overall structure of an individual’s body, describing the physical, physiological and biochemical traits displayed due to genetic and environmental constraints. The phenotype is the particular values that a phenome can take and unlike the genotype, it changes over time (Champandard, 2004). The interaction between the genes in a chromosome is called epistasis and it can affect the phenotypes (Oliveira, et al., 2007). Table 1 shows how the phenotype relates to the phenome.

|  |  |
| --- | --- |
| **Phenome** | **Phenotype** |
| Height | 1.75m |
| Weight | 70kg |
| Eye colour | Blue |
| Shoe size | 7 |

Table 1 Phenome and corresponding phenotype

The genotype and phenotype are similar concepts, but they must be carefully distinguished. The phenotype is used when determining the fitness or the success of an individual, while the genotype must be referred to at the reproduction stage. An offspring inherits the genotype from its parents and has genetic operators manipulating it (Rothlauf, 2006).

The following table summarizes the correspondence between the natural and artificial terminology (Goldberg, 1989):

|  |  |
| --- | --- |
| **Natural** | **Genetic Algorithm** |
| Chromosome | String |
| Gene | Feature, character or detector |
| Allele | Feature value |
| Locus | String position |
| Genotype | Structure |
| Phenotype | Parameter set, alternative solution, a decoded structure |
| Epistasis | Nonlinearity |

Table 2 Comparison of Natural and GA Terminology (Goldberg, 1989, p. 22)

## Genetic Algorithm Encoding

Once the initial population is created, every individual is evaluated and attributed a fitness function. The fitness function and the encoding methods are, usually, the only components of the genetic algorithm that depend on the problem to solve.

Encoding is the process of creating chromosomes in a data structure that can be saved and used on a computer (Bourg & Seemann, 2004). The variables representing the phenotype can be encoded through different methods, like binary numbers, whole numbers, floating points, alphabetic encoding, trees, etc. For an encoding system to be considered good, it needs to satisfy the following conditions:

* Have relative independence of the genes;
* Include as many limitations of the problem as possible;
* Have low complexity.

Solving the encoding problems is usually considered part of the evaluation function and the precision of the evaluation function is part of the problem description. Specifying the evaluation function can involve a simulation and it can be approximative or partial.

The main encoding techniques are (Kumar, 2013):

1. Binary Encoding

Binary encoding is the simplest and most popular encoding method for genetic algorithms. It consists of representing an individual using a string of bits (1s and 0s), represented in Figure 2. The advantages of using this method are that it is relatively simple to implement, every optimization problem can be encoded using this method, and operations with binary numbers are space-efficient, meaning that the algorithm will run smooth on machines that don’t have much memory.

A picture containing object, clock

Description automatically generated

Figure 2 Binary Encoding

1. Real Value Encoding

It consists of representing a gene through symbols, values or strings, as shown in Figure 3, and it is mostly used for optimization in a continuous search space. This technique is convenient because it doesn’t need to follow any encoding or decoding methods due to the fact that it uses real values as representation.

A close up of a logo

Description automatically generated

Figure 3 Real Value Encoding

1. Order Encoding

It consists of representing a chromosome using a sequence of elements. This encoding method is very useful for solving ordering problems like the travelling salesman’s problem (see **Error! Reference source not found.** for a detailed description). In TSP, the order of the visited cities is represented through a string of numbers or letters (Figure 4).



Figure 4 Order Encoding

1. Tree Encoding

It consists of representing the gene in the form of a tree of objects and it is mainly used in genetic programming for evolving expressions and programs.

A close up of a clock

Description automatically generated

Figure 5 Binary Tree Encoding

## Structure of a Genetic Algorithm

### Initial Population

A genetic algorithm starts with an initial population of randomly generated individuals which represents a set of solutions to the given problem. The size of the population needs to be carefully determined because if it is too small, the algorithm goes into premature convergence due to the lack of diversity. Similarly, if the population is too large, the algorithm would become slow.

The two most used techniques for initializing the population in a genetic algorithm are:

1. Random Initialization – every individual is randomly generated.
2. Heuristic Initialization – individuals are generated using a heuristic.

The entire population should not be initialized using the heuristic method alone, because if that is the case, the population will start with very similar individuals and very little diversity. The best way to do it is to initialize a couple of chromosomes using the heuristic technique and then fill out the population with random individuals. This way, the population will have enough diversity in order to optimize the solutions (Haupt & Haupt, 1998).

### Fitness Function

Every individual in the population represents a potential solution to the given problem. The quality of the solution is measured using a fitness function. This function works by taking in the phenotype of the individual and returns its fitness score represented as a floating-point number (Champandard, 2004).

Each optimization problem has a its own individual way of calculating the fitness. The quality of each chromosome depends on how well its phenotype behaves in the environment. The fitness score does not need any constrains for the scale as long as the individuals are assessed against each-other.

For a fitness function to be considered good, the following need to be applied (Mallawaarachchi, 2017):

* Clear definition for easier understanding.
* Efficient implementation – a bad fitness function leads to a bad genetic algorithm.
* Quantitative assessment – the fitness scores must be comparable
* Intuitive results – the best fitness should be assigned to the best performing individual.

### Selection

An important role is held by the selection operator due to the fact that it is used to decide which individuals in the population will contribute in creating the next generation. The goal of selection is to ensure better chances of breeding for the individuals with higher fitness, thus, over a couple of generations, rising the average fitness level. There is a large number of selection methods available, and the person implementing the genetic algorithm can also create its own individual one.

Here are the most simple and common ones (Haupt & Haupt, 1998):

1. Fitness Proportionate Selection

This type of selection is the most popular one. When using this method, the probability of selection for each individual is direct proportionate with their fitness function. This means that it gives the fitter individuals a better chance of being selected for the breeding process and thus passing on their features to the next generations. Through this method, the same individual can be chosen more than once so having an individual that is a lot fitter than the rest can lead to premature convergence.

The information in Table 3 will be used to aid describing two types of fitness proportionate selection. Both methods are implemented using a pie chart, by dividing the wheel into the number of individuals in the population, each of them getting a piece that is proportionate to their fitness score.

|  |  |
| --- | --- |
| **Individual** | **Fitness Score** |
| A | 8.2 |
| B | 3.2 |
| C | 1.4 |
| D | 1.2 |

Table 3 Fitness Proportionate Selection Individuals and Fitness Score Example

* Roulette Wheel Selection (RWS)

This type of selection is made by choosing a fixed point on the chart’s edge then spinning the wheel. The individual corresponding to the part that stops under the fixed point will be included in the breeding process.

Figure 6 Roulette Wheel Selection

* Stochastic Universal Sampling (SUS)

This method is very similar to the Roulette Wheel Selection, but this time, two fixed points are being chosen instead of one. Because of this, the individuals with a lower fitness score still have a pretty good chance at being chosen. This is desirable if the individuals with the highest fitness scores are very similar because it introduces a better diversity for the offspring.

Figure 7 Stochastic Universal Sampling

1. Rank Selection

This selection method consists of calculating the fitness values for each individual, on each generation, and sorting them in descending order of these values. Each individual then gets assigned a probability to be selected, which depends on its rank in the population. When using this technique, the fittest individuals lose their considerable advantage over the least fit because they all get almost the same share of the pie. Because of this, it should only be used when all the individuals in the population have a very similar fitness score in order to prevent stagnation (Kumar, 2012). It is worth noting that this method also works with negative values.

|  |  |  |
| --- | --- | --- |
| **Individual** | **Fitness Score** | **Rank** |
| A | 3.9 | 3 |
| B | 3.8 | 4 |
| C | 4 | 2 |
| D | 4.05 | 1 |

Table 4 Rank Selection Individuals and Fitness Score Example

Figure 8 Rank Selection

1. Tournament Selection

The tournament selection consists of the direct comparison between two or more randomly chosen chromosomes and the selection of the one with the highest fitness. The bigger the size of the tournament, the smaller chances there are for the low fitness individuals to be chosen for reproduction.

A screenshot of a cell phone

Description automatically generated

Figure 9 Tournament Selection

### Crossover

The crossover operator resembles the natural crossover between chromosomes. It works by taking information from the parents chosen through the selection process and combining it to create new individuals (offspring). The crossover in a genetic algorithm is dependent on the problem and the encoding method (Umbarkar & Sheth, 2015).

Given the major importance of this operator, several techniques have been developed:

1. Single-point and multi-point crossover

A screenshot of a cell phone

Description automatically generatedLet *r* be the length of the chromosome. A crossover point is an integer *k* ∈ {1, 2, 3, …, r-1}. The integer *k* indicates the position inside the chromosome where the sequence breaks. The two segments obtained are combined with the other segments originating from the second parent, thus creating a new offspring with traits from both individuals.

Considering two chromosomes:

Figure 10 Single-Point Crossover Parents

A screenshot of a cell phone

Description automatically generatedFollowing crossover, the strings on the right side of the crossover point *k* are exchanged between the chromosomes, resulting in two new offspring:

Figure 11 Single-Point Crossover Offsprings

The multi-point crossover method is very similar to the single-point one, only in this case, two or more crossover points are generated.

A screenshot of a cell phone

Description automatically generatedConsidering the same *x* and *y* individuals, but with two crossover points, *k1* and *k2*:

Figure 12 Two-Point Crossover Parents

A screenshot of a cell phone

Description automatically generatedFollowing the recombination process, the sequences between k1 and k2 are exchanged, resulting in two new off-springs:

Figure 13 Two-Point Crossover Offsprings

1. Shuffle crossover

This method consists of choosing one or more crossover points, like in the techniques described previously. However, this time, before combining the chromosome segments, the variables in the parents’ genetic code are shuffled (the same order in both individuals). After completing the crossover process, the values are put back into their initial positions. This method is used in order to remove any possible positional advantage that the variables could have (Ho, et al., n.d.).

It should be noted that the crossover doesn’t generate random off-springs. Although it is unlikely that every crossover between two individuals in the population will generate solutions that are more promising, the chances of creating a more gifted individual are greater than in the case of random searches. From a single-point crossover between two binary strings, only two different individuals can be created which will consist of combined bits from both parents; the created off-spring solutions are at least as promising.

### Mutation

Mutation is the second most popular and most important genetic operator. As described previously, the crossover operator is mainly responsible for the search aspect of the genetic algorithm. On the other hand, mutation is used for keeping a good diversity in the population and therefore it can introduce new individuals to the population, which couldn’t be created though other methods (Goldberg, 1989).

The mutation process consists of altering the value of one or more genes in the offspring’s genetic material. A gene has a certain probability of mutating, also named mutation rate. The mutation rate needs to be carefully determined, because if it is too low, it doesn’t have a good-enough impact on the population thus the search space stays minimal and premature convergence appears. Additionally, if the mutation rate is too high, the algorithm becomes a random search.

There are multiple variants of the mutation operator:

A close up of a clock

Description automatically generated

1. Bit flip mutation (Figure 14) – used for binary coded genetic algorithms, it consists of randomly choosing one or more bits and flip them (turn 0s into 1s and vice-versa).

Figure 14 Bit Flip Mutation

A drawing of a person

Description automatically generated

1. Swap mutation (Figure 15) – most common in permutation-based encoding, consists of choosing two bits at random and swapping their positions.

Figure 15 Swap Mutation

A picture containing text

Description automatically generated

1. Scramble mutation (Figure 16) – also common in permutation-based systems, involves choosing a subset of variables from the chromosome and randomly shuffling the values.

Figure 16 Scramble Mutation

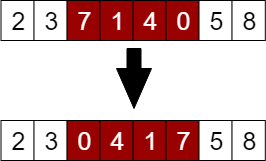


Figure 17 Inversion Mutation

1. Inversion mutation (Figure 17) – consists of selecting a subset of variables form the chromosome and inversing the entire string.

### New Population

The new population can be created in different ways, the two most widely used techniques are:

* Generational – where each new generation only contains the offspring created through selection and the genetic operators.
* Stead-state – also known as Incremental, is where whenever an offspring is created, it replaces an old individual in the population.

When a new generation is created, the algorithm needs to decide which individuals will be kicked out and which ones will be kept. In order to maintain a good diversity in the population, it is good to keep some of the members with a lower fitness function. At the same time, it wouldn’t be desirable to lose the best individuals because that could lower the population’s average fitness.

In order to solve these problems, before creating the new population, some genetic algorithms take the fittest individual and preserve it for the next generation. This process is known as Elitism.

The simplest survivor selection policy is to kick out random individuals and replace them with the offspring. However, this method could lead to convergence problems. To avoid this, a number of methods were developed, two of the most popular ones being: Age based selection, where the oldest individuals are thrown out, and the Fitness-based selection, where the offspring replace the least fit individuals (Goldberg, 1989).

## Genetic Algorithms in Video Games

Video games constitute one of the largest branches of the entertainment industry, consumers spending around $36 billion, in 2017, on things like content, hardware and accessories (Entertainment Software Association, 2018). In the last couple of years, in order to attract more users, the industry started creating dynamic games, consisting of intelligent opponents, good story lines, different goals and changing game environments (Frade, et al., 2008).

Implementing a genetic algorithm into a game can provide the players with a better user experience by achieving more realistic simulations and keeping things interesting by generating different terrains and levels for each stage.

Many level generation techniques only consist of having a rule-based system which builds the levels by following a number of production guidelines. These guidelines are very limited when it comes to creating unique environments because they need to be conceived by the designers and they are always constrained by the rule set. Additionally, there aren’t any standard libraries that can be used for terrain generation, so the algorithms have to be implemented from scratch. These aspects can lead to developers having to spend more time creating a suitable algorithm than it would take them to create the levels one by one (Sorenson & Pasquier, 2010).

# Methodology

This section describes the details for a 3D simulation based on the idea of “survival of the fittest”. The game consists of different species of wild (wolves) and domesticated (sheep) animals that can interact with each other in a natural manner and thus simulate the real-world evolution.

The simulation is created in the Unity engine using the C# programming language. The models for the animals and other objects are mainly obtained from the Unity Assets Store and links to all the resources can be found in the appendix.

## Technologies

### Software

One of the most important decisions that had to be made when creating this project was choosing which technologies to use for developing the simulation. After careful consideration, the two most viable options were: GALib with C++ and OpenGL or Unity and C#.

GALib is a C++ library specially built to aid the implementation of genetic algorithms. It contains a number of classes that can be used in evolutionary computation projects to easily generate the genome types and the genetic operators. The library was built so that it can be used on most of operating systems, being tested and successfully compiled on Windows, MacOS and UNIX systems. This makes it relatively simple to build a custom and more complex genetic algorithm by using the built-in features like the four different random number generators, and the replacement, selection and termination techniques. The library also contains implementations of different mutation and crossover operators which can be easily customized to fit into any genetic algorithm variant (Wall, 1996).

Unity was released in 2005 and became one of the most popular game engines for beginners and veterans alike. One of its main features is the editor which simplifies the game development process by making everything visual and allowing the user to pause the game at any time to inspect the scene and make live changes. Furthermore, the game engine promotes the relationship between entities and components, making it simple to add the desired scripts to certain game objects.

Unity has its own active support platform where developers can ask and answer questions, and due to the large community, it is very likely that other users have encountered the same problems and have already found a solution for them, thus speeding up the development process.

GALib is an easy to use library with good documentation and a large number of built-in techniques. However, using it meant that all the graphics and physics for the simulation had to be built from scratch. Due to time constraints, this was not the best solution as additional time was needed to focus on gathering more information about evolutionary computation. Similarly, Unity has excellent documentation while also having the built-in physics controller. The visual editor makes the graphical side of the development more straightforward while allowing simpler modifications to the project. When using Unity, every aspect of the genetic algorithm had to be implemented from the ground up which allowed for more development flexibility and the opportunity to create something unique.

## Code Design

Every creature in the game has a number of components attached to it. By combining them, the game object becomes an intelligent entity. The most important components are described below.

### A screenshot of text Description automatically generatedCritter

Every individual in the scene has the Critter component attached to it. This class contains and manages all the entity’s attributes, traits and characteristics. It holds the type of the creature (Herbivore, Carnivore, Tree, Dirt or Vegetable), its stats and controls its abilities to alarm, breed challenge. The available traits of each creature have been chosen with the purpose of using them together with the finite state machine (see 3.4.1) to give each individual the ability to evolve over time, then pass on its characteristics to future generations. This list contains 10 different trait types: walking speed, running speed, view radius, view angle, attack points, threat points, rank points, voice strength, beauty points and acting points.

The functions EncodeTraits() and GenerateRandomTraits() are called in the Awake() function in order to setup the creature before it spawns and becomes part of the simulation.

### Herbivore and Carnivore controllers

These controllers are attached to the corresponding creature type. What they do is setup the entities’ target types and default stats (energy, health, resources.) and, if it’s not a child, populate the available behaviours correspondingly. They do this by going through all the possible behaviours and adding them to the creature’s available behaviours list thus allowing it to use them. For the herbivores, every basic behaviour has a 90% chance of being selected, while the social and enemy-encounter behaviours have a probability of 50%. The carnivores are setup similarly, only difference being that the only enemy-encounter behaviour they get is Flee, with a 90% chance.

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

### Vegetable, Tree and Dirt Controllers

Similar to the herbivore and carnivore controllers, they setup the default stats of the objects. The Tree and Dirt controllers also manage how and when these entities produce the default food type (vegetables). This works by checking if the object has been harvested – If not, it stays visible and holds inactive default food types as children. However, if it has been harvested, the controller makes the entity non-harvestable and invisible to the all the other creatures, makes its children active and detaches them in order to let the herbivore consume them. After some time, the entity becomes harvestable again.

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

### Age Controller

A screenshot of a cell phone

Description automatically generatedThe carnivore and herbivores in the simulation have the Age Controller component attached to them. This script’s responsibility is to manage the entity’s age and to update its life stage and object size accordingly. The four life stages and sizes are:

* Baby (0.2, 0.2, 0.2)
* Teenager (0.4, 0.4, 0.4)
* Adult (1.0, 1.0, 1.0)
* Elder (0.8, 0.8, 0.8)

### Breeding Controller

The breeding controller is assigned only to the females in each species. It contains references to the mother and father (set during the breeding process), manages the genetic operators, and creates the new offspring. The behaviours crossover and mutation are done by going through every possible behaviour and generating a random number between 0 and 100:

* A screenshot of a cell phone

  Description automatically generatedIf the random number generated is lower than, or equal to 100-mutation\_rate/2, and the mother has that behaviour, the child inherits it from the mother.
* If it is higher than 100-mutation\_rate/2 but lower than or equal to 100-mutation\_rate, and the father has that behaviour, the child inherits it from the father.
* Otherwise, if the number is higher than 100-mutation\_rate, the algorithm checks if either the mother or father has that certain behaviour:
* If neither of the parents have it, the offspring will gain it as a mutation.
* If either, or both parents have it, the offspring will not get it.

The genetic operators applied to the traits are similar, only that when mutation happens, a random value is generated and attributed to that specific trait.

A screenshot of text

Description automatically generated

### AI

The AI script contains all the information needed to create an autonomous individual. It holds all the functions that check for events and make the transitions between states in the finite state machine. It has references to the animator, the creature’s stats and information, the navigation agent, the seek component and holds information about the current state. Its main purpose is to call the Update function in the finite state machine in order to check for changes and make the transition between states.

### Seek

The seek component holds information about the current and last known targets. There are 7 different target types and they have been all arranged in an array, based on their importance:

1. A screenshot of text

   Description automatically generatedEnemy – a type of creature that the individual is afraid of, either by nature or because of its threat points.
2. Challenger – a creature from the same species that has challenged the individual in order to try and move higher on the social ladder.
3. Mate – a creature from the same species that has accepted to breed with the individual.
4. Courter – a creature from the same species that is trying to convince the individual to breed with it.
5. Food – a viable food source.
6. Potential Mate – a creature from the same species that is ready to reproduce and that the individual will try to convince into breeding with it.
7. Opponent - a creature from the same species and having the same gender as the individual, that will be challenged in order for the individual to move higher on the rank ladder.

The FindVisibleTargets() function implements a field of view for the creature, using its view radius and view angle from the Critter component. All the objects with an attached collider found in the field of view are saved in an array. The array is then searched in the functions called in the Update() method in order to get a target for food, enemy, potential mate and opponent. All the other target types are being set straight from the state machine when a specific event occurs.

## Implementation

### Finite State Machine

Every state in the FSM has an Enter and Exit function which are executed when the owner goes into a state or when it leaves one, respectively. These functions can be left empty, however in this implementation, they are both used. The Enter state methods are the ones that start playing the animations and set the initial destination of the navigation agent, if there is a target. The Exit functions make sure that the owner stopped every possible running coroutine, while also resetting the navigation agent’s path, so that it stops whatever it was doing, or what it was moving towards, in that specific state.

The connection between the states is made through transitions. During a transition, the state machine runs the exit function of the current state, finishes the action associated with it, then executes the enter method of the following state, thus replacing the old state with the new one.

The transitions are triggered by events. In this implementation, the events are represented through boolean functions that check whether something is happening at that point. Examples of these functions can be: can the individual see a target? Is the target a potential mate? Is it close enough to interact with it? Depending on the answers to these questions, the finite state machine will decide which state to go next into, or if it needs more information from the owner.

All these tests are done in the Update function of the current state. Some of the implemented behaviours also use coroutines, in order to give the agent some time before forcing it to move into the next one. This extra time is needed for a number of different reasons, but most commonly it is to let the agent finish the current animation or to make it harder to switch to the next state thus making the current one less desirable in case there are enemies nearby.

**A close up of a map

Description automatically generated**The relationships between all the implemented states are shown in the following diagram:

Due to the large number of states and the connections between them, the diagram became unreadable. To make it easier to understand, the repeated conditions have been colour coded as such:

* Red: owner is attacked
* Green: owner lost target
* Blue: owner is dead

When a creature is created, it is unlikely that it will get every single behaviour. Some behaviours have a weight attached to them. This weight is represented by a certain trait of the individual. The states with a higher value are more desirable than the others, but if the creature only has one possible event response it will be forced into it, regardless of the corresponding trait value. For example, if a creature has a beauty level of 9 and a voice power of 1, when it encounters a potential mate, the individual is more likely to succeed in his interaction if it goes into the Impress state, rather than the Call Mate state. However, if the creature only has the Call Mate state, it will have to go through with it, even if it has a very low success chance.

The entity makes these decisions using a roulette wheel selection (see Figure 6). The algorithm used follows these 4 steps:

1. Create initial sum S1 = 0
2. Add the weight of every available behaviour to S1
3. Generate a random number between 0 and S1
4. Create new sum S2 = 0
5. Add the weight of every available behaviour to S2 until S2 is higher or equal to the random number generated in step 3
6. Set the last behaviour added to S2 as the chosen one.

This way, the states that are more fit have a higher chance of being selected, but at the same time, the less desirable ones are not completely ignored.

In order for the finite state machine to evolve over generations, it had to be somehow customized for each creature, instead of all of them having the same possible states. This was done by following a graph crossover approach. The initial idea was to make it possible to add or remove states, events and transitions, however, due to time constraints it has been decided to mainly just focus on the addition and removal of the available states. This technique was implemented by grouping all the available behaviours based on the events that can possibly lead to them, ending up with 24 states, arranged into 7 different lists based on the type of the behaviours:

1. Default: Idle, Wander, Chase, Attack, Eat, Breed, Get Up, Lay Down, Dead
2. Low Energy: Rest, Sleep
3. Food Source: Knock, Dig
4. Enemy Encounter: Alarm, Play Dead, Startle, Flee
5. Opponent Encounter: Threat, Aggress, Fight
6. Challenger Encounter: Submit, Watch
7. Possible Mate Encounter: Impress, Call Mate

The opponent and challenger encounter states can evolve and become more fit to be chosen due to the fact that they make use of their corresponding traits. If the interaction is successful, the individual gains some extra points in that certain trait. On the other hand, if the interaction fails, the creature loses points from that trait thus making it less likely to be chosen again at the next encounter and making more space for the other less desirable behaviours. Over time, individuals can max out their traits and therefore have a 100% chance of success with certain actions. If the creature gets to breed, its offspring will inherit part of its DNA and thus pass it on to the next generations.

Due to the close relationship between the states in the FSM and the creature’s traits, it can be stated that the finite state machine evolves with the individuals.

### Genetic Algorithm

The following steps are the standard procedure used for implementing a genetic algorithm:

1. Initialize population
2. Calculate Fitness Function
3. Select Parents
4. Crossover and mutate
5. Introduce new offspring into the population

The steps 2-5 are being repeated until an acceptable solution is found for the problem to solve. The aim of this project is to attempt to maximize the traits and behaviours of all the individuals involved in the simulation.

1. Initialize Population

The population contains two types of creatures: herbivores (sheep) and carnivores (wolves). At the beginning of the game, a set number of individuals from each category is spawned on the map. Every creature has its characteristics randomly generated. The traits are a random number from 0 (bad) to 10 (very good), encoded using the real values encoding technique, while the behaviours are selected from a pool of available behaviours. In order to keep a good diversity in the population and avoid premature convergence, every basic behaviour (Wander, Idle, Eat, Breed, etc.) has a 90% chance of being assigned to an entity, while each interactional behaviour (Startle, Play Dead, Impress and Call Mate, etc) has a probability of 50% of being selected.

In addition to these two types of creatures, the simulation also contains objects acting as three different food sources:

* Vegetables – this type is assigned to all the very basic food sources that can be eaten by any herbivores (hay, apples, peanuts). This type of food does not require additional actions to be taken in order to make it an herbivore’s meal.
* Dirt – this type is assigned to the Dirt Piles in the game. It requires the individual to have the Dig state in order to gain a basic food source (in this case, peanuts).
* Tree – this type is assigned to all the Trees in the scene. In order to get a basic food source from them (apples), the creatures need to have access to the Knock state.

The user can spawn a certain entity by selecting it from the provided inventory then clicking on a point on the map. The individuals created this way are setup the same way as the ones in the initial population. After being generated, every entity is added to a list that keeps track of the individuals in the population.

1. Calculate Fitness Function

Each frame, the simulation calculates every individual’s fitness score and arranges them in descending order based on the results, using a bubble sort algorithm. The fitness function used in this simulation is represented by the average of every trait value, age and the number of available behaviours.

Consider:   
 n = number of traits   
 = value of each trait  
 a = age of current individual  
 b = number of available behaviours of current individual

The fitness score is used for two things: breeding and fighting for a higher spot on the social ladder. When a creature is ready to go in either of these states, it has to pick the most likely action that will grant its victory. The weight of each possible action is represented by a specific trait level and/or the individual’s fitness score. If the critter succeeds, it gains extra points on the trait that it just used. However, if the critter fails the interaction, it loses points. When performing a social rank interaction, the critters also win or lose a point in their rank.

1. Select Parents

When an individual in the population reaches maturity, it unlocks the ability to breed. If it comes into contract with a potential partner, the individual will try to convince it into breeding with it. If successful, the female will call the breeding function in order to create and setup a new offspring.

1. Crossover and mutate

In this simulation, the crossover and mutations are happening at the same time. The creation of a new entity is done using the uniform crossover and the random resetting mutation techniques. The algorithm works using the following steps:

1. Generate a random number between 0 and 100:

* If the number is less than 100-mutation\_rate/2, the offspring gets the trait from the mother.
* If the number is higher than 100-mutation\_rate/2 but lower than 100-mutation\_rate, the offspring gets the trait from the father.
* Otherwise, the trait will mutate.

1. Repeat for every trait and behaviour.

By following these rules, both parents have equal chances of passing on their characteristics.

When mutation happens, if the algorithm is currently looking at a trait, it will replace that gene with a randomly generated number between 0 and 10. If it is looking at a potential available behaviour, it will check if either of the parents have it. If this is a new behaviour, the offspring will get it, but if one or both of the parents have it, the child will not.

The gender of the offspring is randomly chosen.

1. Insert new offspring in the population

This is done by instantiating a prefab of either the male or female creature, as a baby, at a position close to the mother. Once the offspring is alive, the parents go into a breed timer where they can’t breed again for another 10 seconds.

In order to avoid overpopulation and the surviving of bad genes, the less fit individuals need to be excluded from the population. This is done in the following ways:

* Old age – once the individual becomes an elder, every frame there is a 30% chance of it dying.
* Starvation – if the creature does not eat for a while it will start to lose health points. Once the health level is 0, it will die and become potential free food for predators. This can happen for different reasons, like not finding food, not being able to eat the food because it doesn’t have that certain behaviour, or just making the wrong life choices.
* Fighting – when fighting, both the predators and the prey can attack each other. The damage they deal is based on their attack points. There is a chance that a herbivore can kill a carnivore but that is unlikely because of the way the attack points are generated.

# Testing and Results

The simulation has been tested to check for correct and efficient outputs. The expected results are:

* the population size grows over time due to the breeding process of the adult creatures
* the average fitness score of each species gets higher over time
* the highest fitness score of each species gets higher over time
* the creatures start having more behaviours

A fully optimized creature means all its traits are maxed out and its finite state machine contains all the implemented behaviours. Due to the nature of genetic algorithms, imperfect solutions are also accepted.

The simulation was first tested with 20 herbivore and 20 carnivore creatures (10 males and 10 females), starting as babies, each species having a 10% chance of mutation.

## Population Size

After letting it run for several minutes, it has been observed that the number of entities in the sheep population has drastically increased, while the wolves population only gained a small number of new individuals. The following chart shows how the numbers changed over time.

Running the program for the second time shows that both species have steadily increased over 33 generations. The number of herbivores was slightly higher compared to the carnivores between generations 8 and 12 after which the sheep population declined slightly reaching 30 individuals, compared to 39 for the wolves. From that point, the herbivore numbers began to rise again, overtaking at generation 24 and finishing with an excess of 52 members over the carnivore population.

By running the program twice with the same values, it has been noted that increase of the population over time is not dependant on the starting number of individuals. It can also be observed that, both times, the sheep population ascended steadily, this being due to the multiple food sources available to them.

## Average Fitness Score

When it comes to the fitness of each species, it has been recorded that the average fitness score of the carnivores drastically ascended, while the herbivores line has only slightly changed over time. The wolves started with a lower average fitness than the sheep, however, by the 9th generation, they caught up, the two average fitness levels being very similar (24.13692 for sheep and 24.38623 for wolves). After generation 10, the carnivores got ahead and kept ascending until the simulation was stopped. The herbivores average fitness points had a slight decline between generations 12 and 16, then continued with very small fluctuations between 23.15 and 25.17. These values are shown in the following chart:

Running the program for the second time suggests that after generation 12, when the average fitness score of the sheep population rises, the wolves’ falls, and vice-versa. The carnivore average fitness score value peaks at 27.32 at generation 25, while the herbivores have theirs at 26.57, generation 11, right before the wolves take over.

After analysing the results, it can be stated that, both times the program ran, the evolutions of the values were fairly similar. The major differences were the starting average fitness score of the herbivores, which was slightly higher in the second run (19.60 vs 21.38) and the growth of the values which was more substantial in the first run. Additionally, the carnivores had a significant drop at the end of the second run, finishing with 14.08 points less than in the first one.

## Highest Fitness Score

Looking at the highest fitness score in each population, the following diagram shows that the best individuals in the carnivore group have strongly evolved over time, changing by an average of +0.78 with every new generation. Meanwhile, the herbivores had a good start, however, between generations 12 and 13 a big discrepancy has been observed, their best individual only having a fitness score of 32.27, compared to the 44.29 one noted before.

The second time the program was ran, the results were very similar, with the decrease of the herbivore highest fitness between generation 11 and 13. A major difference is the dramatic fall of the value at generation 29, for both species, this leading to a lower score at the end, compared to the first time the program ran.

## Available Behaviours

The technique used to assign the available states to the herbivores was that each basic behaviour had a 90% chance of being assigned, while the interactional behaviours only had a probability of 50%. The carnivores went through a similar process, the only difference being that from the Enemy-Encounter behaviours, they all got the Flee state.

In the first generation it has been observed that the most common basic behaviours for the herbivores were Attack, Breed and Lay down which were assigned to 100% of the individuals, while the most popular ones for carnivores were Rest and Flee, with an assignment of 100%, followed by another 4 which have been assigned to 19 out of 20 wolves. It must be noted that none of the predators have any Enemy encounter behaviours, except for the default Flee state.

By generation 6, there has been a constant rise for the most common behaviours from generation 1, 30 out of 31 wolves having the Wander, Breed and Submit states, while 100% of them can go into the Rest and Flee behaviours. When it comes to the herbivore population, the Breed and Lay Down behaviours were available to 100% of the population while the Aggress state became less common, with only 43% of the sheep being able to perform it, compared to 60% in the first generation.   
By this point, the carnivores have started to mutate and develop new behaviours like Knock, Dig, Alarm, Play dead and Startle.

By generation 30, compared to the herbivores, the carnivore population had a higher percentage of individuals who can execute the Play Dead state, this being 74% and 64%, respectively. The substantial rise in the number of the wolves with a mutated behaviour is very significant, considering that none of them had any at the start of the simulation.

Based on the information presented in this chapter, one can state that the creatures in the population and their behaviours evolve over time.

# Conclusion

This project addressed the problem of having a creature and its behaviours evolve over time using genetic algorithms.

A discussion of different ways this can be implemented has been provided in the Background Research Chapter. This section presented various ways in which data can be encoded, it looked at how different methods can affect the results and when each of them should be favoured.

Various suitable software and technologies have been examined and reasoning has been given for why one has been chosen over the other.

An in-depth description of the implementation has been provided, making it relatively easy for anyone with minimal coding knowledge to reimplement this project. Every important aspect of the simulation has been discussed and they have all been presented with a visual diagram of the implementation of each method and technique.

Details about the implementation of the genetic algorithm have been supplied in order to show how the techniques discussed in the research chapter have been understood and implemented.

The thesis examined the different, consistent testing results which show how specific individuals and their population evolve over time. These tests look at the number of individuals at each generation, the average fitness score of each species, the highest fitness value recorded in each population and the number of available behaviours at different times in the simulation.

The information presented in this document indicates that all the objectives have been met and the implementation of the evolving creatures is a success.

# Future Work

Due to time constraints, some test scenarios have been left out of this document and planned to be investigated in the near future instead. A single test bed can take hours to create, present and analyse, and thus it is very time consuming. Future work on this involves looking at how the values of each critter develop over time and investigating the results when using a different fitness function.

Currently, the simulation is flawed in the sense that there are some gaps between events and behaviours, due to the technique used when implementing the graph crossover. The creatures can get stuck in a loop between two states and thus just die of starvation or be attacked by an enemy, without it being able to respond. This can be avoided by adding in more behaviours and making sure that every creature has assigned at least one viable response for each possible event.

Another issue that needs to be looked at is the fact that sometimes, the simulation becomes slow and unresponsive making Unity crash. Initially it seemed to be caused by the bubble sort algorithm used to order the creatures based on their fitness score, specifically because of its O(n^2) complexity. It has been optimized by stopping the algorithm from running if there’s no changes in the inner loop, however the problem persisted. Another possible reason for this was the number of creatures spawned being too large for the computer’s resources. This has also been ruled out because after closely watching the resources used by Unity, and the number of individuals in each population, there was no clear pattern. It would happen with as little as 60 creatures while also running just fine with 600. This issue remains to be investigated and solved in the future.

Furthermore, the initial plan for the development of this projects also contained the implementation of some game aspects. Because getting the simulation to work properly was a more important task than adding in game mechanics, it has been decided to leave them out for now.

# Bibliography

Al Globus, 2000. *JavaGenes: Evolving Graphs with Crossover.* s.l., NASA Ames Research Center.

Anon., 2018. *r/Unity3D.* [Online]   
Available at: https://www.reddit.com/r/Unity3D/comments/70ra7b/after\_surprising\_response\_from\_the\_first\_thread\_i/  
[Accessed 13 03 2019].

Blickle, T. & Thiele, L., 1996. A Comparison of Selection Schemes used in Genetic Algorithms. *Evolutionary Computation,* 4(4), pp. 361-394.

Bourg, D. M. & Seemann, G., 2004. *AI for Game Developers.* 1st ed. Sebastopol: O'Reilly.

Champandard, A. J., 2004. AI Game Development. In: *AI Game Development.* Indianapolis: New Riders, pp. 421-448.

Dasgupta, D. & Michalewicz, Z., 1997. *Evolutionary Algorithms in Engineering Applications.* Berlin: Springer.

Entertainment Software Association, 2018. *Essential Facts About the Computer and Video Games Industry.* [Online]   
Available at: http://www.theesa.com/wp-content/uploads/2018/05/EF2018\_FINAL.pdf  
[Accessed 25 01 2019].

Frade, M., Vega, F. F. & Cotta, C., 2008. Modelling Video Games ’ Landscapes by Means of Genetic Terrain Programming- A New Approach for Improving Users ’ Experience. In: G. M. e. al., ed. *Applications of Evolutionary Computing. EvoWorkshops 2008. Lecture Notes in Computer Science.* Berlin: Springer.

Goldberg, D. E., 1989. *Genetic Algorithms in Search, Optimization & Machine Learning.* Boston: Addison-Wesley.

Haupt, R. L. & Haupt, S. E., 1998. *Practical Genetic Algorithms.* 1st ed. New York: Wiley.

Ho, Y. S., Tang, V. & Chin, L., n.d. *Genetic Algorithm: A Learning Experience.* [Online]   
Available at: http://www.cse.unsw.edu.au/~cs9417ml/GA2/index.html  
[Accessed 27 01 2019].

Koza, J. R., Bennet, H. F., Andre, I. D. & Keane, M. A., 1999. Genetic Programming: Turing's Third Way to Achieve Machine Intelligence.

Kumar, A., 2013. Encoding Schemes in Genetic Algorithm. *International Journal of Advanced Research in IT and Engineering,* 2(3), pp. 1-7.

Kumar, R., 2012. Blending Roulette Wheel Selection & Rank Selection in Genetic Algorithms. *International Journal of Machine Learning and Computing,* 2(4), pp. 365-370.

Mallawaarachchi, V., 2017. *How to define a Fitness Function in a Genetic Algorithm?.* [Online]   
Available at: https://towardsdatascience.com/how-to-define-a-fitness-function-in-a-genetic-algorithm-be572b9ea3b4  
[Accessed 25 01 2019].

Mitchell, M. & Forrest, S., 1994. Genetic Algorithms and Artificial Life. *Artificial Life,* 1(3), pp. 267-289.

Oliveira, P., Andrade, M., Krieger, J. & Soler, J., 2007. *Epistasis Model Selection Using the Genetic Algorithm.* São Paulo-SP, s.n.

Rothlauf, F., 2006. *Representations for Genetic and Evolutionary Algorithms.* 2nd ed. Heidelberg: Springer.

Sorenson, N. & Pasquier, P., 2010. Towards a Generic Framework for Automated Video Game Level Creation. In: C. Chio, et al. eds. *Applications of Evolutionary Computation.* Berlin: Springer, pp. 131-140.

Turing, A. M., 1948. *Intelligent Machinery,* s.l.: National Physics Laboratory.

Turing, A. M., 1950. Computing Machinery and Intelligence. *Mind,* 01 October, LIX(236), pp. 433-460.

Umbarkar, A. J. & Sheth, P. D., 2015. Crossover Operators in Genetic Algorithms: A Review. *ICTAC Journal on Soft Ccomputing,* 06(01), pp. 1083-1092.

Unity, n.d. *Unity Answers.* [Online]   
Available at: https://answers.unity.com/index.html

Wall, M., 1996. *GAlib: A C++ Library of Genetic Algorithm Components.* [Online]   
Available at: http://lancet.mit.edu/ga/dist/galibdoc.pdf  
[Accessed 15 02 2019].

# Appendices

## Appendix 1: Ethics Form

**SECTION A: Project Definition**

**FOR UNDERGRADUATE & TAUGHT POSTGRADUATE ONLY**

**Complete the following table with full and relevant information relating to your research.**

|  |  |
| --- | --- |
| Student Name | Oana – Madalina Furtuna |
| Student Number | 16025091 |
| Student E-mail Address (please use University e-mail) | 16025091@students.southwales.ac.uk |
| Name of Principal Project Supervisor | Ian Wilson |
| Project Title | Evolutionary Algorithm Animator. Develop a GUI based animator for illustrating evolutionary algorithms in operation. |
| Briefly describe the project, being sure to identify any aspects that are relevant to the Ethical Evaluation in Section B.  NOTE: A project determined to be High Risk will need to include additional information in Section B to fully-specify the risks and mitigations. | The project consists of researching Evolutionary Algorithms, exploring their use in video games and creating a simulation illustrating them in operation. |
| Please add an explanation of your study in plain English, with particular focus on any parts of your study which involve human participants. No more than 100 words. This is to help the Faculty Research Ethics Committee (FREC) to understand the project. | My study does not involve human participants. |

**SECTION B: Ethical Evaluation**

**FOR UNDERGRADUATE & TAUGHT POSTGRADUATE ONLY**

Consider the following points to determine the level of ethical risk your research presents:

1. Involves those who are considered vulnerable such as:

* Children under 16.
* Adults with learning difficulties.

Unless in an accredited setting, accompanied by a carer or professional with a duty of care.

1. Involves those who are considered highly vulnerable such as:

* Adults or children with diagnosed mental illness/terminal illness/dementia/in a residential care home.
* Adults or children in emergency situations.
* Adults or children with limited capacity to consent

1. Involves those who are “dependent” on others (such as teacher or lecturer to student). Unless in an accredited setting associated with normal working conditions or routines and within normal operating hours, such as a cultural institution, pre-school, school, or youth club where the research is carried out as part of professional practice such as curriculum development.
2. Requires full NHS ethical approval via the Integrated Research Application System.
3. Requires a Human Tissue Act license.
4. Involves “covert” procedures as in covert observation studies.
5. Involves anything considered “sensitive”. For example, does not carry a risk of those involved disclosing information which compromises the research (e.g., illegal activities; activities where moral opinion may differ, potential professional misconduct – work errors).
6. Induces significant psychological stress or anxiety or produce humiliation or cause more than fleeting harm / negative consequences beyond the risks encountered in the normal life of the participants (and where the potential for fleeting “harm” is clearly detailed in the participant information sheet). If in doubt regarding definition of the above terminology, please contact the research governance office.
7. Involves administration of drugs, placebos or other substances (such as food substances or vitamins) as part of this study.
8. Involves invasive procedures (not limited to blood sampling, collection of biological samples, or passing current through a participant’s body, etc.).
9. Offers any financial inducements to participate in the study.
10. Intends to recruit serving prisoners or serving young offenders via Her Majesty’s Prison & Probation Service.

For your course, there may be specific requirements in **addition** to these, depending on the nature of the subject and how your project is assessed. You must also complete those requirements.

If **none** of the 12 points above apply, then the research can be considered **Low Risk**, *unless your course identifies additional criteria relevant to your subject that would render it High Risk*. This Section is then signed off by yourself and your supervisor and held on file for review by FREC.

If **any** of the 12 points applies, then the research is considered **High Risk** and students must bring the matter to the attention of their research supervisor immediately. **Research cannot then commence until mitigations for the risk are agreed by FREC**. Seek advice from your Supervisor, who can help you identify mitigations of the risk or redesign as a Low Risk project.

**All students must complete the section below, in collaboration with their supervisor.**

Please strike through the statement that **does not** apply.

1. An ethics review has been completed, and the project has been identified as Low Risk.
2. ~~An ethics review has been completed, and a High Risk was identified. I agree to explain how they may be mitigated below and agree to abide by any conditions identified at this stage, by my Project Supervisor, the School or the Faculty. I understand that High Risk projects can only proceed with approval from the Faculty Research Ethics Committee.~~

|  |
| --- |
| Issues: (Include as much information as possible to help FREC members to understand the issues. Extend onto additional pages as necessary.) |
| Proposed mitigations: (Include as much information as possible to help FREC members to understand the mitigations. Extend onto additional pages as necessary.) |
| Student’s Signature:  Date: |
| **Supervisor’s statement:** I have ensured due diligence and accountable decision making by the student. I have sought appropriate advice where required to support my judgment in this.  Supervisor’s Signature:  Date: |
| **Any false or mis-represented information contributing to this Ethical Evaluation, including attempting to pass off a High Risk project as a Low Risk project, is subject to the Student Misconduct Regulations and may also have legal repercussions.** |

Both signatures are **required** for all projects, both Low Risk and High Risk.

## Appendix 2: Plan Milestone 1

## Appendix 3: Plan Milestone 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Main Task | Subtask | Due Date | Started | Completed |
| Gather Resources |  | ongoing | x |  |
| Research Tools |  | 20/10/2018 |  | x |
| Create Game Design Document | Game Description | 18/10/2018 |  | x |
| Characters | 22/10/2018 |  | x |
| Traits and Characteristics | 01/02/2019 |  | x |
| Tasks and Achievements | 28/02/2019 |  |  |
| Inventory | 28/02/2019 |  |  |
| Game Currency | 28/02/2019 |  |  |
| Create and/or Gather Assets |  | 01/02/2019 |  | x |
| Create Game Environment | Create Map | 01/02/2019 |  | x |
| Implement Finite State Machine | 07/02/2019 | x |  |
| Implement Traits and Characteristics | 14/02/2019 | x |  |
| Implement Genetic Algorithm | 21/02/2019 | x |  |
| Add In-Game Currency | 01/03/2019 |  |  |
| Add Tasks and Achievements | 01/03/2019 |  |  |
| Write-Up | Abstract | 28/03/2019 |  |  |
| Introduction | 28/03/2019 |  |  |
| Aims and Objectives | 23/11/2018 |  | x |
| Background Research | 01/02/2019 |  | x |
| Methodology | 08/03/2019 | x |  |
| Testing and Results | 22/03/2019 |  |  |
| Conclusion | 28/03/2019 |  |  |
| Testing |  | 22/03/2019 |  |  |
| Poster Presentation |  | 03/05/2019 |  |  |
| Coursework | Milestone 1 | 23/11/2018 |  | x |
| AI for Games Dev | 07/12/2018 |  | x |
| Game Engines Design | 14/12/2018 |  | x |
| Real-Time Rendering Techniques | 11/01/2019 |  | x |
| Milestone 2 | 01/02/2019 |  | x |
| Milestone 3 | 29/03/2019 |  |  |
| Game Engines Design | 05/04/2019 |  |  |
| Parallel and Concurrent Programming | 19/04/2019 |  |  |
| AI for Games Dev | 02/05/2019 |  |  |
| Milestone 4 | 03/05/2019 |  |  |

## Appendix 4: Unity Assets Sources

Low Poly Animals: <https://assetstore.unity.com/packages/3d/characters/animals/lowpoly-animals-107755>

Cartoon Farm Crops <https://assetstore.unity.com/packages/3d/vegetation/plants/cartoon-farm-crops-79777>

Water Effect Fits For Low Poly Style <https://assetstore.unity.com/packages/vfx/shaders/water-effect-fits-for-lowpoly-style-87810>

Low Poly Fruits Pickups <https://assetstore.unity.com/packages/3d/props/food/low-poly-fruit-pickups-98135>

## Appendix 5: Meeting Notes

**24th September 2018**

Write Aim and Objectives.

Planning the project, including other coursework, and give a deadline to every task.

Look into GAlib.

Terrarium – game based on genetics and evolution, created by Microsoft.

SWOC – strengths, weaknesses, opportunities, challenges.

**8th October 2018**

I presented a random character creation tool that I worked on. It takes random body parts and puts them together to create an individual.

Create pool of genetic behaviours and characteristics for the characters.

Look into different methods of encoding the traits and characteristics of each individual in the population. (binary, trees, etc.).

GAlib manual.

**22nd October 2018**

Feedback on the list of objectives – needs redoing, with less details.

I presented the work I’ve done on the application (spawning individuals that move around randomly looking for food).

The game characters need to be animated, had a look at Maximo for bipeds.

Create a storyboard for the project.

**5th November 2018**

RPC – Remote Procedure Call, to make the game multiplayer. (each player introduces a new creature?)

Decided to use binary encoding to represent the individuals.

Need to focus more on the Genetic Algorithm than on the gameplay for now.

**19th November 2018**

Got feedback on my background research write-up:

* Know everything you write about so you can answer questions
* Write more on the fitness function because it is very important
* Try using more graphs and diagrams for easier understanding
* Write in your own words
* Shorter phrases with less commas.
* DO NOT recycle work from other assignments

Write a small program to see if sorting the individuals based on their fitness has any impact on the Roulette Wheel selection.

**5th December 2018**

Discussed how to write code design and testing:

* Precondition
* Postcondition
* Description
* Examples

Discussed how the dissertation document can be made better by adding in citations, using the Word built-in headers and the table of contents.

**10th January 2019**

Discussed how the fitness function can be better integrated in the simulation, by making the critters have and see different colours (for example a red critter would be more likely to be attacked by a predator). Also talked about fitness scaling (change with age) and survivability.