

**Rudimentary Evolution Simulation using a Genetic Algorithm**

MComp Honours Computer Science (G405)

Supervisor: Dr Phillip Lord

May 2021

Word Count: 12270

Devon Foister (180282173)

**Abstract**

**Declaration**

“I declare that this dissertation represents my own work except, where otherwise stated.”

**Acknowledgements**

**Table of Contents**

1. [1 Introduction 9](#_Toc71761303)

[1.1 Topic 9](#_Toc71761304)

[1.2 Motivation 9](#_Toc71761305)

[1.2.1 The Problem 9](#_Toc71761306)

[1.2.2 My Approach 10](#_Toc71761307)

[1.3 Aims & Objectives 11](#_Toc71761308)

[1.3.1 Aim 11](#_Toc71761309)

[1.3.2 Objectives 11](#_Toc71761310)

1. [2 Background 12](#_Toc71761311)

[2.1 Research Sources 12](#_Toc71761312)

[2.2 Genetic Algorithms 12](#_Toc71761313)

[2.2.1 Overview 12](#_Toc71761314)

[2.2.2 Algorithm Selection 13](#_Toc71761315)

[2.2.3 Algorithm Crossover 15](#_Toc71761316)

[2.2.4 Algorithm Mutation 16](#_Toc71761317)

[2.2.5 Algorithm Survivors 16](#_Toc71761318)

[2.3 Physics Simulation 17](#_Toc71761319)

[2.3.1 First Law 17](#_Toc71761320)

[2.3.2 Second Law 17](#_Toc71761321)

[2.3.3 Third Law 17](#_Toc71761322)

[2.3.4 Calculating Movement 18](#_Toc71761323)

[2.4 Collision Detection and Resolution 20](#_Toc71761324)

[2.4.1 Overview 20](#_Toc71761325)

[2.4.2 Circle/Circle 20](#_Toc71761326)

[2.4.3 Rectangle/Rectangle 21](#_Toc71761327)

[2.4.4 Rectangle/Circle 21](#_Toc71761328)

[2.4.5 Collision Resolution 22](#_Toc71761329)

1. [3 Design & Implementation 23](#_Toc71761330)

[3.1 Project Plan 23](#_Toc71761331)

[3.1.1 Plan Explanation 24](#_Toc71761332)

[3.2 System Design 25](#_Toc71761333)

[3.2.1 System Requirements 25](#_Toc71761334)

[3.2.2 Simple UML Diagram 26](#_Toc71761335)

[3.2.3 Object Oriented Approach 26](#_Toc71761336)

[3.2.4 Completed System UML Diagram 27](#_Toc71761337)

[3.3 GUI Design 28](#_Toc71761338)

[3.4 Tools & Technologies 29](#_Toc71761339)

[3.4.1 C++17 29](#_Toc71761340)

[3.4.2 Visual Studio 2019 29](#_Toc71761341)

[3.4.3 Eigen 29](#_Toc71761342)

[3.4.4 SFML 29](#_Toc71761343)

[3.5 Application Development Methodology 30](#_Toc71761344)

[3.6 Algorithm Development 30](#_Toc71761345)

[3.6.1 Planning 30](#_Toc71761346)

[3.6.2 Algorithm Overview 31](#_Toc71761347)

[3.6.3 Selection Implementation 32](#_Toc71761348)

[3.6.4 Crossover Implementation 33](#_Toc71761349)

[3.6.5 Mutation Implementation 35](#_Toc71761350)

[3.6.6 Algorithm Output 37](#_Toc71761351)

[3.7 Organism Development 38](#_Toc71761352)

[3.7.1 The Chromosome 38](#_Toc71761353)

[3.7.2 Normal Movement 38](#_Toc71761354)

[3.7.3 Food and Poison 38](#_Toc71761355)

[3.8 Physics Engine Development 39](#_Toc71761356)

[3.8.1 High Level Explanation 39](#_Toc71761357)

[3.8.2 Rigid Bodies 39](#_Toc71761358)

[3.8.3 Integrating Acceleration 39](#_Toc71761359)

[3.8.4 Collision Detection/Resolution Within the Physics Engine 40](#_Toc71761360)

[3.8.5 Integrating Velocity 41](#_Toc71761361)

[3.9 Collision Detection Development 42](#_Toc71761362)

[3.9.1 Circle/Circle Collision 42](#_Toc71761363)

[3.9.2 Rectangle/Circle 43](#_Toc71761364)

[3.9.3 Rectangle/Rectangle 43](#_Toc71761365)

[3.10 Renderer Development 44](#_Toc71761366)

[3.11 Global Header 44](#_Toc71761367)

1. [4 Testing 45](#_Toc71761368)

[4.1 Overview 45](#_Toc71761369)

[4.2 Testing Solution 45](#_Toc71761370)

[4.2.1 Element Creation 45](#_Toc71761371)

[4.2.2 Selection Methods 45](#_Toc71761372)

[4.2.3 Crossover Methods 46](#_Toc71761373)

[4.2.4 Mutation Methods 47](#_Toc71761374)

[4.2.5 Collision Detection 51](#_Toc71761375)

1. [5 Evaluation 52](#_Toc71761376)

[5.1 Results 52](#_Toc71761377)

[5.1.1 Example One 52](#_Toc71761378)

[5.1.2 Example Two 54](#_Toc71761379)

[5.1.3 Example Three 56](#_Toc71761380)

[5.2 Requirements Evaluation 58](#_Toc71761381)

[5.2.1 Functional Requirement Evaluation 58](#_Toc71761382)

[5.2.2 Non-Functional Requirement Evaluation 59](#_Toc71761383)

1. [6 Conclusion 60](#_Toc71761384)

[6.1 Satisfaction of Aims & Objectives 60](#_Toc71761385)

[6.2 Summary 60](#_Toc71761386)

[6.3 Future Work 60](#_Toc71761387)

1. [7 References 61](#_Toc71761388)
2. [8 Appendices 62](#_Toc71761389)

[8.1 Appendix 1 62](#_Toc71761390)

[8.2 Appendix 2 68](#_Toc71761391)

[8.3 Appendix 3 74](#_Toc71761392)

[8.4 Appendix 4 81](#_Toc71761393)

**Table of Figures**

[Figure 1‑1 Nathan Rooy’s Solution [X] 9](file:///C:\Users\dev\source\repos\dfoister\Rudimentary-Evolution-Dissertation\Ds.docx#_Toc71761250)

[Figure 2‑1 Population, Chromosomes and Genes [X] 12](file:///C:\Users\dev\source\repos\dfoister\Rudimentary-Evolution-Dissertation\Ds.docx#_Toc71761251)

[Figure 2‑2 Roulette Wheel Selection [X] 14](#_Toc71761252)

[Figure 2‑3 Stochastic Universal Sampling [X] 14](#_Toc71761253)

[Figure 2‑4 Point Crossovers [X] 15](#_Toc71761254)

[Figure 2‑5 Uniform Crossover [X] 15](#_Toc71761255)

[Figure 2‑6 Circle/Circle Collision [X] 20](#_Toc71761256)

[Figure 3‑1 Project Plan Diagram 23](file:///C:\Users\dev\source\repos\dfoister\Rudimentary-Evolution-Dissertation\Ds.docx#_Toc71761257)

[Figure 3‑2 Simplified System Diagram 26](#_Toc71761258)

[Figure 3‑3 System UML Diagram 27](file:///C:\Users\dev\source\repos\dfoister\Rudimentary-Evolution-Dissertation\Ds.docx#_Toc71761259)

[Figure 3‑4 Console at Program Launch 28](#_Toc71761260)

[Figure 3‑5 Program GUI 28](#_Toc71761261)

[Figure 3‑6 Mutation Process Code 31](#_Toc71761262)

[Figure 3‑7 Roulette Wheel Selection Code 32](#_Toc71761263)

[Figure 3‑8 Tournament Selection Code 33](#_Toc71761264)

[Figure 3‑9 Uniform Crossover [X] 33](#_Toc71761265)

[Figure 3‑10 Single Point Crossover [X] 33](#_Toc71761266)

[Figure 3‑11 Multiple Point Crossover [X] 34](#_Toc71761267)

[Figure 3‑12 Scramble Mutation Code 35](#_Toc71761268)

[Figure 3‑13 Swap Mutation Code 36](#_Toc71761269)

[Figure 3‑14 Creep Mutation Code 36](#_Toc71761270)

[Figure 3‑15 Random Resetting Code 37](#_Toc71761271)

[Figure 3‑16 Algorithm Output 37](#_Toc71761272)

[Figure 3‑17 Chromosome Representation 38](#_Toc71761273)

[Figure 3‑18 Organism Movement Code 38](#_Toc71761274)

[Figure 3‑19 Acceleration Integration Code 39](#_Toc71761275)

[Figure 3‑20 (Physics Engine) Collision Detection Code 40](#_Toc71761276)

[Figure 3‑21 Collision Penetration Resolution 40](#_Toc71761277)

[Figure 3‑22 Collision Velocity Resolution 41](#_Toc71761278)

[Figure 3‑23 Velocity Integration Code 41](#_Toc71761279)

[Figure 3‑24 Circle/Circle Collision Code 42](#_Toc71761280)

[Figure 3‑25 Rectangle/Circle Collision Code 43](#_Toc71761281)

[Figure 4‑1 Program Start 45](#_Toc71761282)

[Figure 4‑2 Program Running with Numbered Elements 45](file:///C:\Users\dev\source\repos\dfoister\Rudimentary-Evolution-Dissertation\Ds.docx#_Toc71761283)

[Figure 4‑3 Uniform Crossover Test 46](file:///C:\Users\dev\source\repos\dfoister\Rudimentary-Evolution-Dissertation\Ds.docx#_Toc71761284)

[Figure 4‑4 Single Point Crossover Test 46](#_Toc71761285)

[Figure 4‑5 Multiple Point Crossover Test 47](#_Toc71761286)

[Figure 4‑6 Swap Mutation Test 47](file:///C:\Users\dev\source\repos\dfoister\Rudimentary-Evolution-Dissertation\Ds.docx#_Toc71761287)

[Figure 4‑7 Swap Mutation & Random Engine Test 48](file:///C:\Users\dev\source\repos\dfoister\Rudimentary-Evolution-Dissertation\Ds.docx#_Toc71761288)

[Figure 4‑8 Scramble Mutation Test 48](file:///C:\Users\dev\source\repos\dfoister\Rudimentary-Evolution-Dissertation\Ds.docx#_Toc71761289)

[Figure 4‑9 Inversion Mutation Test 49](#_Toc71761290)

[Figure 4‑10 Creep Mutation Test 50](#_Toc71761291)

[Figure 4‑11 Random Resetting Test 50](#_Toc71761292)

[Figure 4‑12 Collision Detection Test 51](#_Toc71761293)

[Figure 5‑1 Result One Table 52](#_Toc71761294)

[Figure 5‑2 Result One Average Fitness 53](#_Toc71761295)

[Figure 5‑3 Result One Best Fitness 53](#_Toc71761296)

[Figure 5‑4 Result Two Table 54](#_Toc71761297)

[Figure 5‑5 Result Two Average Fitness 55](#_Toc71761298)

[Figure 5‑6 Result Two Best Fitness 55](#_Toc71761299)

[Figure 5‑7 Result Three Best Fitness 56](#_Toc71761300)

[Figure 5‑8 Result Three Average Fitness 57](#_Toc71761301)

[Figure 5‑9 Result Three Best Fitness 57](#_Toc71761302)

# Introduction

## Topic

Newcastle University CSC3423 Genetic Algorithms Lecture [1]. This lecture gave me the inspiration to start this project and goes in depth into the history of genetic algorithms (GA) as well as in depth information on different selection methods, mutation/crossover methods and many different use cases. Along with the speaking aspect of the lecture, this gave me a solid groundwork to begin planning my own GA.

## Motivation

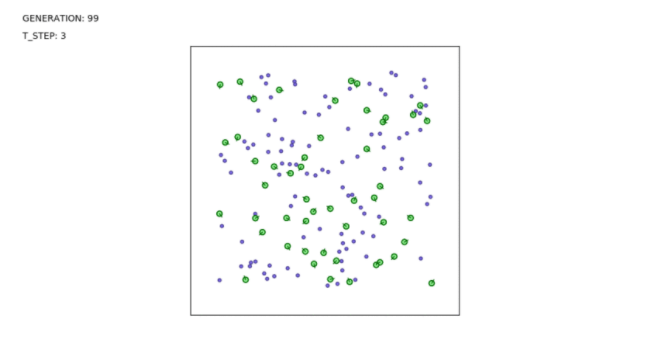
The motivation for this project began after I had taken a module at Newcastle University, titled Biologically Inspired Computing. After reading through the lecture notes for genetic algorithms [3] I wanted to see the algorithm used in the context of natural evolution. At this point, I had no idea of the extent and reach of evolutionary algorithms and just found the subject interesting. Around this same time, I was still struggling to come up with a topic for my third-year dissertation. This is when the idea of creating an evolution simulation was spawned. I then had the idea of making it interactive in order to showcase what a genetic algorithm does and make the program enjoyable to use for others who are interested in genetic algorithms. I know that the project is possible due to similar concepts being achieved by others such as Nathan Rooy [X] who managed to achieve a similar result, with the use of neural networks. Here is a screenshot from Rooy’s simulation.

Figure ‑ Nathan Rooy’s Solution [X]

### The Problem

When choosing to study biologically inspired computing, I was expecting the topic to be more based around nature itself, and simulating nature within the scope of computing. Mainly due to my own poor research, I found this was not the case, so I decided to relate it more to nature myself, learning about evolution and genetic algorithms in the process. The genetic algorithms I have used so far involve simply changing parameters and watching numbers pop up in a console. I did not find this very engaging and thought it would be more interesting if I could actively see what the algorithm was doing over several generations. This could make the subject more interesting for those who want to learn about genetic algorithms, those with an interest in biologically inspired computing or people who find the idea of evolution and tiny virtual organisms interesting.

### My Approach

Similar projects, such as one by Nathan Rooy [4], tend to ignore negative factors in an organism’s environment such as predators or poison. This project will, at the very least, include poison in the environment to add another factor for the organism to adapt to. My project will also be interactive, allowing the user to alter the genetic algorithm to produce vastly different results. This could be used in teaching to provide a more exciting approach to genetic algorithm optimisation. I will be using C++’s graphics library with the help of a manual from Stanford University WRONG [4] to display the entities on screen along with information pertaining to the algorithm. I will be using a collection of external resources [5] as well as University lecture material [6] in order to detect and handle collisions between organisms and food/poison, affecting the organism’s health accordingly. Each organism will have a chromosome, comprising of several genes. These genes will contain important information on the organisms, such as size, speed, food/poison perception radius. There will also be random variables used in movement calculations that gage the general intelligence of an organism. Each generation, the healthiest organisms will be carried over to the next generation, along with a selection of their offspring, generated from the genetic algorithm. The end result should be a self-sustaining colony of organisms.

## Aims & Objectives

### Aim

Explore how a genetic algorithm can be manufactured to simulate evolution amongst artificial organisms, providing an interactive interface where parameters can be altered to adjust the effectiveness of evolution.

### Objectives

1. Theorize and create a genetic algorithm that increases the calculated fitness of an organism, effectively evolving it.

The algorithm will need to be extensive, containing numerous methods which can be changed by a user to alter the effectiveness of the algorithm.

1. Implement a GUI which displays organisms, other entities, and the current statistics of the genetic algorithm.

The interface is needed to display the organisms moving in the environment, finding food, and avoiding poison. The program would work without it but is needed for the user to see the evolution in action.

1. Create a movement system whereby organisms choose where to move based on their size, quality of their vision, and nearby food and poison positions.

The movement system will require the creation of a physics engine that can accurately determine the position of organisms based on numerous factors. This will take a lot of research and way in which each organism moves will be partly based on their genes.

1. Create a collision detection system that allows organisms to interact with food/poison entities and have their health affected accordingly. This will be visible in the program.

Without a collision system, there is no way to keep organisms within the confines of the environment, able to consume food, and able to avoid poison. The organisms would also be able to travel through each other without it.

1. Program a death and reproduction cycle which will remove the previous organisms, reproduce them using the genetic algorithm and create a new set of organisms with new chromosome values.

This is key for the actual evolution to occur as there will need to be many generations for the organisms to evolve and become more suited to the environment. The genetic algorithm will manage the reproduction.

1. Introduce an interactive component to the GUI that would allow a user to modify the algorithm's specific parameters to change how fast or effective the evolution of organisms is.

This is another crucial objective as it makes up the entire interactive portion of the project. This will most likely consist of sliders for mutation/crossover chance and the option to change the selection method the algorithm uses.

# Background

This section of the **FINISH THIS FINISH THIS FINISH THIS**

## Genetic Algorithms

### Overview

To understand genetic algorithms, you need to take a step back and first look at evolutionary computing. Evolutionary computing is a research area within computer science, inspired by the process of natural evolution. Although you probably have a solid grasp on the concept of evolution, the Introduction to Evolutionary Computing describes it as follows:

A given environment is filled with a population of individuals that strive for survival and reproduction. The fitness of these individuals is determined by the environment, and relates to how well they succeed in achieving their goals. In other words, it represents their chances of survival and of multiplying. [X]

This style of problem solving is known as trial-and-error or generate-and-test, where you have a collection of possible solutions of varying qualities that establish how close they are to the ideal solution of a particular problem. Depending on their quality, they are then used as a basis for creating new solutions. This step is repeated until, ideally, a perfect solution is found. Most of the time it is more likely to be a very good solution, rather than perfect. There is a clear link between evolution and this method of problem solving.

Environment 🡨🡪 Problem Individual 🡨🡪 Possible Solution Fitness 🡨🡪 Quality

Genetic algorithms in particular follow a very specific structure. The algorithm starts by generating the initial population of individuals. These individuals each possess a chromosome, which in turn contains a selection of genes. As these terms will be used frequently throughout this paper, here is a diagram to aid in understanding. In this context, A1 to A5 are the individuals/organisms.



Figure ‑ Population, Chromosomes and Genes [X]

Next, each individual has their fitness calculated which ranks them depending on their ability to solve the given problem. The selection phase is next which selects the fittest individuals to have their genes passed on to the next generation. This process places the selected individuals into pairs, which then become parents.

The next two stages in the algorithm are the reproduction stages, the first being crossover. Crossover takes each pair of parents and creates offspring by combining their genes to create new chromosomes. It is called crossover as this switching of genes normally takes place at different places, dependent on one or many crossover points. This concept and its different variations will be discussed later. Mutation is the next step in the algorithm. For each new offspring, there is a low probability for each of its genes to be mutated. This could be as simple as flipping a bit or inverting entire subsets of chromosomes. Once again, this will be expanded on later.

This new population of individuals will then have their fitness values calculated and the evolution cycle continues. The algorithm will have a termination criterion that is activated when either a suitable solution is converged upon, the population has not improved for X generations or a specified generation limit has been reached.

### Algorithm Selection

##### Population Management

There are two key population management models for genetic algorithms, the generational model, and the steady-state model. In the generational model, the entire population is replaced by offspring at the end of each generation. Most simple genetic algorithms use this model, keeping the population size, mating pool size and offspring size the same, to make replacing the entire population simple.

With the steady-state model, only part of the population is replaced by its offspring, each generation. The proportion of the population that gets replaced is known as the generational gap. As these models are performed based on a population’s fitness, either can be used independent of what the chosen problem is.

#### Parent Selection

There are also two types of parent selection, fitness proportional selection (FPS) and ranking selection. FPS is where the probability of an individual being selected for mating depends on its fitness compared to that of the rest of the population. This is the most common method used but has some drawbacks. Individuals with particularly high fitness values, especially early in the algorithm tend to take over the population as they are overly selected. This reduces the chance that the algorithm will search through a large space of possible solutions and is known as premature convergence. There also exists another issue that if fitness values are very close throughout the population, selection pressure is lacking, resulting in almost random selection. This has been shown to cause very slow fitness increasement in later generations of the algorithm. One solution to this issue is called windowing, where the fitness of the least-fit individual is subtracted from the fitness of the rest of the population.

Ranking selection was created due to the drawbacks of FPS. The idea is that it keeps a constant selection pressure by ranking the population based on fitness and then assigning selection probability based on rank, rather than the fitness value itself. This fixes both key concerns with the FPS approach.

#### Selection Methods

There are many selection methods for genetic algorithms which carry out the actual selecting process. One of these and perhaps the simplest method is roulette wheel sampling. The idea is you spin a wheel with the population on, the size of each individual’s section dependent on their fitness or rank. Whatever individual the wheel lands on, will become a parent. Below is a diagram demonstrating this.

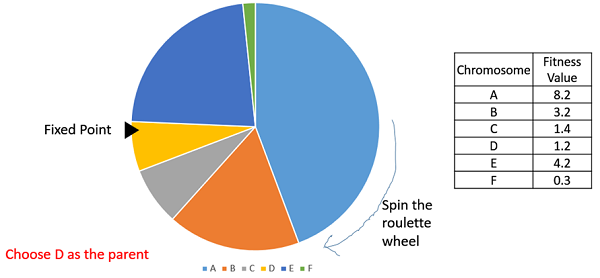


Figure ‑ Roulette Wheel Selection [X]

Similar to roulette wheel sampling is stochastic universal sampling (SUS), which is also a wheel but has multiple fixed points, allowing multiple parents to be selected at once and increasing the chance of an individual with high fitness being selected. SUS is demonstrated below.

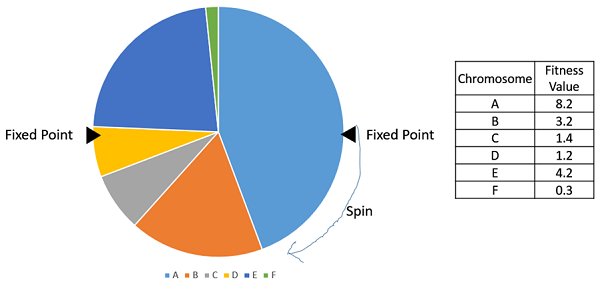


Figure ‑ Stochastic Universal Sampling [X]

Another commonly used selection method is tournament selection. N number of individuals are chosen from the population randomly. These N individuals are then compared against each other, based on fitness, and the fittest individual is selected to become a parent. This is then repeated for each parent. In an alternative version, there can be multiple tournament survivors.

There is also random selection where parents are selected from the population completely randomly. As there is no selection pressure this method is mostly avoided but still available as an option.

### Algorithm Crossover

Crossover is directly related to biological reproduction where two parents are selected and offspring are produced using the genetic material of the parents. Crossover is usually applied with a very high probability and in many cases the probability is one. There are many crossover methods, with some being more suited to this project than others. Two simple operators are single point crossover and multiple point crossover.

One point crossover involves taking a random index within the chromosome and creating a splitting point at that index. From there both parents are split, and the children are created by swapping the tails of the two parents from that index onwards. Multiple point crossover is very similar but this time the parents are split at multiple points along the chromosome. The children are then creating by alternating the segments of the parent chromosomes in each child. The following diagram explains this quite well.

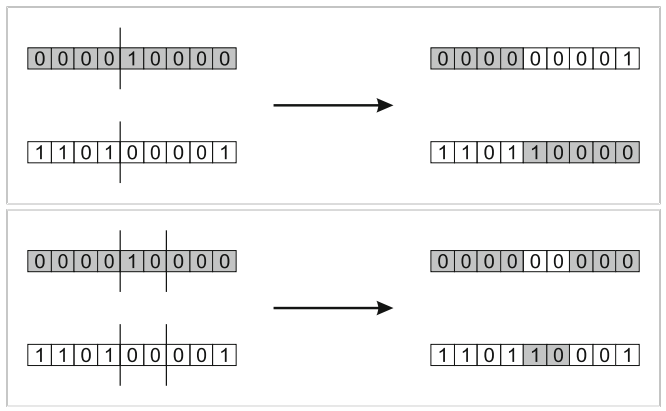


Figure ‑ Point Crossovers [X]

Uniform crossover is another method commonly used. For this method, each gene in the first offspring has an equal chance of being inherited from either parent. This happens for each gene in the first offspring and then the second offspring is created from the inverse of the first. Once again here is a diagram.

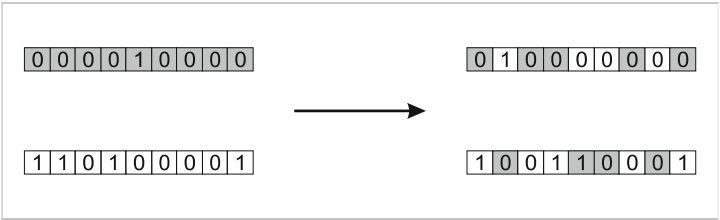


Figure ‑ Uniform Crossover [X]

### Algorithm Mutation

Mutation in genetic algorithms is used to keep diversity in the population and is essential for convergence. Mutation is also applied with a low probability to ensure that the algorithm does not just become a random search. Once again there are many mutation methods that can be used.

Bit flip mutation is the simplest form of mutation and is used for chromosomes encoded in binary. It is as simple as it is named, with genes being flipped between one and zero. This would not be applicable for this project as the chromosomes are encoded with real values. Random resetting is similar but meant for other encoding methods. In this method, the gene is swapped with a random value created from a set of possible, defined values.

Creep mutation works by adding a small positive or negative value to a gene. The size of the step is dependent on the size of the fitness values as you only want to make small steps. The distribution should be symmetric, centred around zero, to ensure the value can move the same amount positively or negatively.

Scramble mutation and inversion mutation are very similar mutation in the sense that they would be implemented similarly. Scramble mutation takes a subset of the chromosome and shuffles the genes within the subset. Inversion mutation also takes place on a random subset of the chromosome, but instead reverses the order of the subset. Swap mutation involves swapping two genes in the chromosome randomly. The problem with these methods is that they don’t introduce new values into the chromosome and if used exclusively, the entire population will keep recycling the same genes. It could be effective to combine one of these methods with random resetting or creep mutation.

### Algorithm Survivors

Genetic Algorithm: A Tutorial Review [6]. This short paper outlines the features of a genetic algorithm as well as the basic pseudocode for a functioning algorithm. Each component of a GA is described in detail, such as the selection methods and crossover/mutation methods for reproduction. It also outlines the different GA expressions and explains each one. I used this resource for my early research but still come back to it occasionally as a reminder of certain aspects.

Evolving Simple Organisms using Deep Learning from Scratch [2]. This blog post which I found quite recently tries to solve a similar problem in scratch using a neural network. This was helpful as it showed, at the very least, that the problem is solvable. It also led me to an interesting question. How am I going to make an organism better or worse at navigating to the food? The author used a neural network and used a normalized value representing the direction to the nearest food as an input. The direction the organism turns is then calculated within the neural network. Reading his blog, I ended learning about neural networks and their usefulness in a situation like this. Although I do not plan on using neural networks, I believe I can create a similar result for determining the angle an organism rotates. I can use a simple function to determine the angle from where the organism is facing to the food/poison, then calculate a turning angle based on the values stored in the organism’s chromosomes.

The next section will look at the simulation of physics within the program to create organisms that are capable of moving within an environment accurately.

## Physics Simulation

As this project is supposed to simulate organisms in an environment, I felt it was important to incorporate real physics into the simulation. This will make the organisms more realistic and make the movement of each one look more natural. One way of achieving this is by implementing, and forcing organism to abide by, Newtonian physics. Newton’s three laws of motion are as follows.

### First Law

“When viewed in an inertial reference frame, an object either remains at rest or continues to move at a constant velocity, unless acted upon by a force.” [X]

This first law introduces the idea that an object moves after a force is applied to it. A force has a direction and a magnitude, which is measured in Newtons. Newton’s first law is also known as the law of inertia, inertia being resistance of any physical object to any change in its velocity. A still object resists any force trying to move it and a moving object resists any force trying to change its velocity. Any object in a simulation displays inertia based in its mass, the higher the mass, the harder it is to move with a force.

### Second Law

“The vector sum of the forces *F* on an object is equal to the mass *m* of that object multiplied by the acceleration vector *a* of the object: *F = ma*.” [X]

It makes more sense in this simulation to rearrange the equation as (. This is because each organism will have a known weight value as well as a force to be applied based on its decisions and genes. This means the change in velocity for each organism can be calculated from the force applied divided by the mass of itself.

### Third Law

“When one body exerts a force on a second body, the second body simultaneously exerts a force equal in magnitude and opposite in direction on the first body.” [X]

Essentially and well known as, for every action, there is an equal and opposite reaction. In the context of this project, it basically means that the objects within the simulation pass force between one another, if two organisms hit each other, they will not just stop but act according to the momentum each one is carrying.

Momentum is represented as . Due to the law of conservation of momentum, the total momentum of a system of objects remains constant. Despite the objects exerting equal and opposite forces onto each other, the velocities do not change by the same amount but instead change proportionally to their mass, conserving their momentum.

### Calculating Movement

From here onwards, I will refer to aspects of the simulation in the context of this project. For example, objects are now organisms. The SUVAT equations of motion helped me to understand how motion over time is calculated and how the positions of organisms within the simulation could be manipulated. SUVAT stands for the five variables of linear acceleration:

* S = Displacement (how much an organism’s position has changed by).
* U = Initial velocity.
* V = Final velocity.
* A = Acceleration.
* T = Time.

The SUVAT equations allow you to work out any of the five variables based on the values of the other variables. To demonstrate this, the first equation is as follows, . This means that if you know the constant acceleration (m/s/s) and time then you can work out how much the acceleration affects the overall velocity of an organism. There is a problem with SUVAT equations in that they are dependent on there being constant acceleration or velocity. In this simulation, organisms move in different ways relative to the current conditions of their environment. This means that these factors are never constant. Calculus is needed to work out how the positions of the organisms should change, starting with derivatives.

* Position is noted as .
* Velocity is the first derivative of position with respect to time, noted as .
* Acceleration is the second derivative of position with respect to time, noted as .

Using these derivates and the equation , you can work out the velocity of an organism if you know its positions at two points in time. Here is an example:

Organism position at 0.5 seconds = [5,0]

Organism position at 3.5 seconds = [35,0]

This equation does not account for whether the velocity varied between 0.5 seconds and 3.5 seconds, only giving the average velocity. This is solved by repeating the equation many times a second, referred to as the hertz (Hz). Many game physics engines update at anywhere from 60Hz to 240Hz. Even those running at 240Hz may not be perfectly accurate but are enough to simulate physics to a respectable degree. Once again there is a problem with this approach in respect to this project. We need to calculate an organism’s new position based on its changing velocity, whereas this approach calculates velocity based on positions. The next step to solving the problem is integration.

Integration is used to determine how changes in an organism position’s derivatives effect its position over time. The integrals of the position’s derivatives are ( and ). in this context is the time between each sample (timestep). In a simulation dt is the time between each frame, or . There are several integration methods that can be used starting with Euler’s Integration.

These values are calculated each update cycle, taking the values from the current cycle (n) and adding the derivatives, multiplied by the timestep in order to calculate the next cycle’s values (n+1). The problem with this method is that velocity is treated as a constant value during a timestep whereas the velocity should be constantly changing due to the integration of acceleration. This ultimately results in a loss of accuracy over time. There is an alternate method, known as Backward Euler, which calculates the derivative at the next timestep rather than the current timestep. This is shown below.

This method has a glaring problem for a real-time simulation in that it needs to know the velocity and acceleration from the next timestep, which is information you do not have. Fortunately, there is a third option which provides more accuracy over time while not forcing you to predict the future. This is known as ‘Symplectic Euler’. Here, acceleration is integrated using the current cycle values and velocity can then be integrated more accurately relative to the current state of the organism’s motion.

Having a small timestep is very important when using any of these integration methods. The result of the calculation will never be perfect and having a small timestep can help to minimize the inaccuracy of the simulation over longer periods of time. The following example, provided by Richard Davison’s game physics tutorial [X], demonstrates how different timesteps used in Symplectic integration can affect how positions are calculated:

We know from the SUVAT calculations that after 2.4 seconds, a constant acceleration of 2 m/s should result in an end position of:

Using Semi-Implicit Euler (Symplectic) integration results in the following positions:

* Position using timestep 0.01 = 5.784
* Position using timestep 0.1 = 6
* Position using timestep 0.2 = 6.24
* Position using timestep 0.4 = 6.72
* Position using timestep 0.8 = 7.68

Richard then goes on to describe how a 60Hz or 60 frames per second simulation, which equates to a timestep of 0.0166 seconds would be “fairly accurate” but would still lead to the position being inaccurate after a decent amount of time. This suggests that physics engines should aim to run at as many hertz as possible while still maintaining adequate performance.

This content was covered due to the importance of a physics engine as a part of the model, and this is the best simulation technique I found for the context of the project. The next section I will be covering is collision detection and resolution.

## Collision Detection and Resolution

### Overview

Collision detection is extremely important in many computer systems, being a “fundamental problem in computer animation, computer graphics, physically-based modelling, and robotics.” [[X](http://gamma.cs.unc.edu/COLLISION_PON/collision.html#:~:text=Collision%20detection%20is%20a%20fundamental,interactions%20in%20the%20simulated%20environments.)] It is easy to think that if you have two circles on the screen, the computer should know whether or not they are touching or not but instead a series of equations is needed, as well as different types and phases of collision detection for the purposes of accuracy and efficiency. Anytime the mouse is moved in a program or webpage there is most likely a function in the background tracking exactly where the mouse is and whether it is within the bounds of another element.

Just as important is collision resolution. As an example, if you watched a simulation of two cars colliding at high speed you could easily tell at what point the cars collided. However, there is a good chance that you would not be able to tell exactly how the cars would act afterwards. Many factors need to be considered. How fast was each car going? What direction was each car travelling? What material are they driving on? How much does each car weigh? The job of the simulation in this case, would be to take all these factors into consideration and calculate the trajectory of each car after the collision.

There are three types of collision detection needed for the project, rectangle to rectangle, circle to circle and rectangle to circle. There are multiple methods for collision detection and for each I would need to also get the penetration distance between the two objects and the direction of least penetration.

Jeffrey Thompson’s collision detection website [4] provided me with an extensive resource for collision detection between different basic shapes. Each example is well explained, readable and easy to understand. He explains that there are “more efficient ways to detect these collisions” [X] and that the principles are taught with minimal math. This was a great starting point and provided interactive samples with each section.

### Circle/Circle

To calculate whether two circles are colliding you need to first get the distance between the centres of both circles. This is achieved by the following calculation, with circle A and circle B.

* DistanceX = A Centre X – B Centre X.
* DistanceY = A Centre Y – B Centre Y.
* Distance = Square Root (DistanceX^2 + DistanceY^2).

You then check whether ‘Distance’ is less than or equal to the sum of the radii of the circles. If this is true, then they are colliding.

A picture containing wall

Description automatically generated

Figure ‑ Circle/Circle Collision [[X](https://happycoding.io/tutorials/processing/collision-detection)]

To resolve a collision you need some more information, chiefly, the penetration distance of the collision and the direction of least penetration (collision normal vector). For circle-to-circle collisions finding the penetration distance is easy. It is the radii of the two circles, minus the distance between the two centres at the time of collision. To get collision normal you need the direction vector between the two circles by subtracting the position of the second circle from the first. You then normalise this value.

### Rectangle/Rectangle

If you have two rectangles A and B, here are the checks you must do to detect whether they are colliding.

1. Is the right side of A to the right of the left side of B?
2. Is the left side of A to the left of the right side of B?
3. Is the bottom side of A below the top side of B?
4. Is the top side of A above the bottom side of B?

If all of these are true, then the two shapes are colliding. Although not easy to follow by reading, it would be easy to implement in C++.

### Rectangle/Circle

For this collision method, you need to first work out which side of the rectangle is closest to the circle and then you carry out the following tests.

1. If the circle is to the right, check against the right edge of the rectangle.
2. If the circle is to the left, check against the left edge of the rectangle.
3. If the circle is above, check against the top edge of the rectangle.
4. If the circle is below, check against the bottom edge of the rectangle.

Simple enough. Once you know which side you need to check you use the Pythagorean Theorem to calculate the distance between the centre of the circle and the edge of the rectangle closest to the edge. Here is how this value is found:

* DistanceX = Circle Centre X – Left/Right Edge (dependent on previous tests).
* DistanceY = Circle Centre Y – Top/Bottom Edge (dependent on previous tests).
* Distance = Square Root (DistanceX^2 + DistanceY^2).

You then check whether ‘Distance’ is less than or equal to the radius of the circle. If this is true, then they are colliding.

### Collision Resolution

A collision where two objects overlap would be an error in the simulation, however it is inevitable. The physics engine could move an object after finding no collisions in that frame, right into the position of another. On the next frame this collision would be spotted and only then can it be resolved. There are two stages to collision resolution. The first step is moving the objects out of each other so that they are no longer colliding. The second step is accurately simulating the velocity of the two objects after that have separated, as they would no longer carry on in the same direction or at the same magnitude as they did before the collision.

For the first step you would need to use the penetration and collision normal values calculated in the detection stage, as well as the inverse mass of each object. The mass is needed to ensure that an organism would not be able to move a wall, or a smaller organism would move more than a larger organism. Everything should be moved proportionally to its mass. Inverse mass is used to create objects with infinite mass, such as the walls of the environment. You would set the inverse mass to 0 and when the position change is calculated, it will be multiplied by 0. This is how the positions would be resolved, where the two objects are and :

)

For the second stage of collision resolution, the velocity of the colliding objects will be directly manipulated, instead of using integration, to cause an instant change in momentum as bodies bounce off each other. To ensure that the objects move in the correct (and opposite) direction when colliding and momentum is conserved by the engine, an impulse is calculated. The equation for an impulse between two colliding objects is as follows, provided by Rich Davison [X]:

The total mass would be calculated already from separating the two objects which leaves the top half of the equation to understand, starting with . This is the relative velocity between the two objects. To work out how much velocity is travelling in the direction of the collision between the objects, the dot product of the relative velocity and the collision normal, denoted, is calculated. This is .

Although not relevant in the context of this project, is mainly used to calculate how objects of different material types behave when colliding with other objects. The coefficient of restitution, denoted , however is important. This is a value typically between 0 and 1 where a value of 1 would mean that an object loses no velocity when colliding with another object. Conversely, a value of 0 means that an object would completely halt when a collision occurred. These would be known respectively as a perfectly elastic collision and a perfectly inelastic collision.

These collision detection and resolution methods were covered due to the importance of accuracy when it comes to the simulation of the organisms’ environment. This concludes the background and research portion of this dissertation. The next section of this document will cover the design and implementation of the system.

# Design & Implementation

## Project Plan

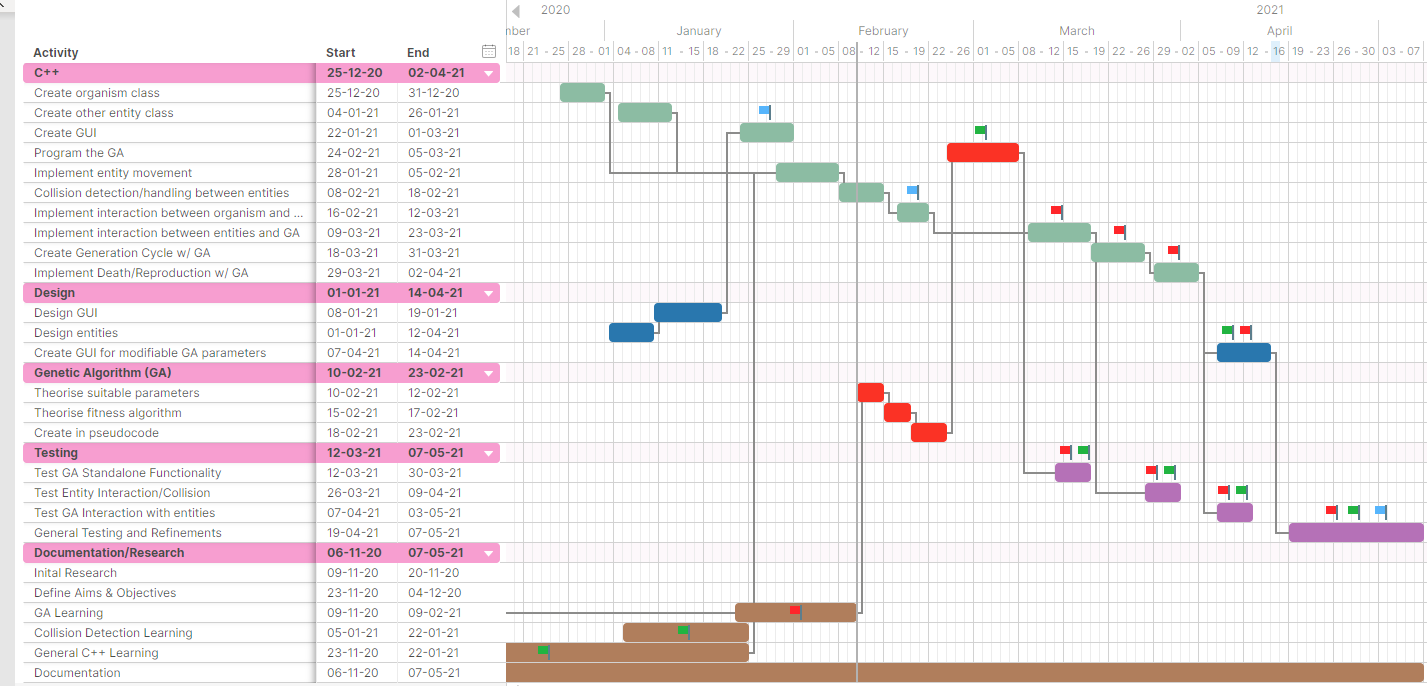


Figure ‑ Project Plan Diagram

### Plan Explanation

The diagram above details the project plan that I have used throughout the course of the year, beginning in November 2020, and ending in May 2021. The diagram was created using tomsplanner.com [X] in the early stages of the project. It is split into 5 sections:

1. Programming

Any task that involves using Visual Studio and creating the actual simulation in which this project is aiming to produce. This will often have overlap with the genetic algorithm and design section, indicated by a flag of the respective colour of these sections.

1. Genetic Algorithm

This section is for any task related to the genetic algorithm used on the project. The algorithm needed to be built from the ground up, tailored specifically to work hand in hand with the C++ program. This meant careful planning was needed, as well as achieved in a suitable time for the rest of the project to run according to plan.

1. Design

These tasks relate to general designing of aspects of the project. This ended up being a very small part of the project in the end as I kept the visual front of the program very simple to spend more time on the complicated backend.

1. Testing

Any task related to ensuring correct functionality of the program and algorithm. This was a long and important part of the project as every single algorithm method and aspect of the physics engine had to be tested. Although a lot of testing was done at the tail end of the project, testing was continuous throughout due to adopting an Agile development methodology.

1. Research/Documentation

Constant research and documentation have taken place throughout the lifespan of the project. Despite there being start and end dates for each of the learning tasks, I have been continually learning new things since beginning this dissertation. Documentation has been frequent, with me writing down how I am implementing features as I create them and keeping the aims and objectives noted down ready for the final writeup.

As you can see from the dependency arrows in Figure 3.1, nearly every piece of the project had to be completed in a set order, or else continuing would become impossible. Later I will discuss how this affected the rate in which the project was completed and whether I would change my approach in the future.

## System Design

### System Requirements

#### Functional Requirements

1. At program launch, the system must provide the user with a list of options regarding altering the GA and number of elements in the environment.
2. The system must display all organisms, food elements, poison elements and the current generation number in a window, all in the correct positions according to the physics engine.
3. The console should output the current statistics of the genetic algorithm during, and after each generation.
4. At the end of each generation, all elements in the scene should be replaced with the new elements created by the GA.
5. The system should detect collisions between elements and pass this information to the physics engine for resolution.
6. When a collision is detected, the velocity and position of affected elements should be handled accurately by the physics engine and displayed to the user.
7. The GA should be able to carry out multiple operations for each of the following: population selection, chromosome crossover, and gene mutation.

#### Non-Functional Requirements

1. The GA should receive the old population and return the next generation in less than 1 second.
2. The system should run on the following operating systems:
   1. Windows 10 64-bit/32-bit: Home, Professional, Education, Enterprise.
   2. Windows 7 SP1: Home Premium, Professional, Enterprise, Ultimate.
3. The system should support the following hardware:
   1. A video card that supports a minimum resolution of 1280x720. Ideally a resolution of (1366x768) should be in use.
   2. At least 50MB free disk space.
   3. 4GB of RAM, ideally 8GB.

### Simple UML Diagram

The project has a lot of classes that need to work together for physics, collisions, and algorithm to function accurately. I have created a simple UML-esque system diagram to give an idea of how these classes work together.

Diagram

Description automatically generated

Figure ‑ Simplified System Diagram

### Object Oriented Approach

I went with an object-oriented approach to designing the system for several reasons. Due to the different systems at play in the program, the modularity of OOP made debugging and troubleshooting much easier as I could pinpoint exactly which system was causing issues. The modularity aspect also meant that I had could go back and alter different systems within the project without fear of breaking others. Inheritance made managing the different objects in the simulation much simpler as they could all be referred to as the parent BaseObject class, meaning less redundant code. Polymorphism made updating these objects each frame in different ways while using the same update() method possible.

### Completed System UML Diagram

A picture containing text, receipt, screenshot

Description automatically generated

Figure ‑ System UML Diagram

## GUI Design

I made a deliberate choice to keep the UI and its elements simple to focus more on the backend algorithm and physics engine. For this reason, I used a simple console input/output system for the purpose of user input. Trying to include all parameters in the simulation window seemed unnecessary and convoluted and would only serve to make the project more difficult. Below is a figure showing the console at program launch.

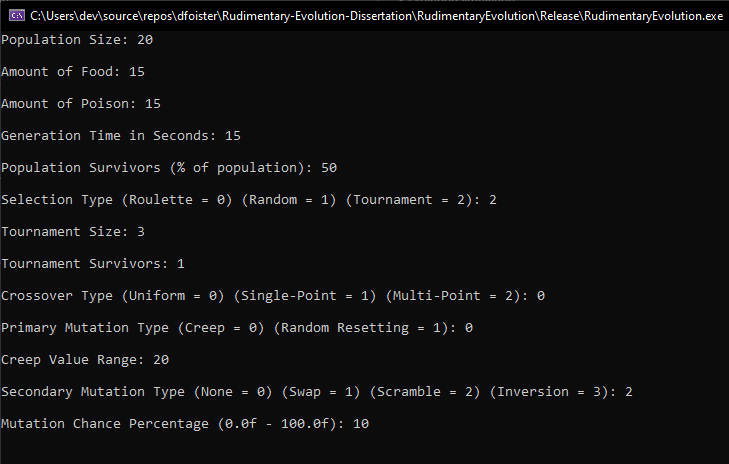


Figure ‑ Console at Program Launch

The simulation window was kept simple, with a plain grey background and black walls to display the organisms’ environment. The organisms themselves I displayed as hollow purple circles, with small green and red dots to show food and poison, respectively. I also display the generation number in the top left for the user’s convenience. This is displayed below.

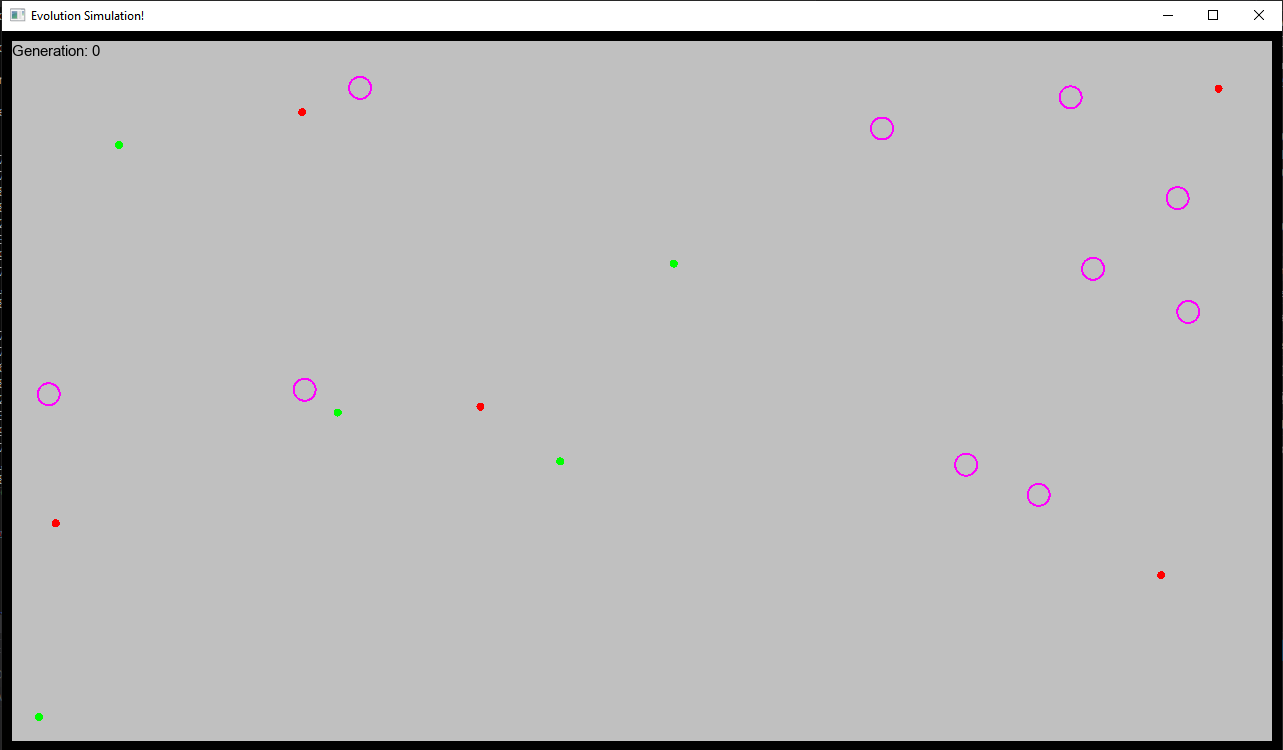


Figure ‑ Program GUI

## Tools & Technologies

### C++17

Before I began this project, I was not sure which language I was going to use. Java was my preferred and most used language, but I had recently started using C++, and there were a few things I liked compared to Java.

C++ is a compiled language, meaning it is much faster than other languages and offers much greater control due to being a strongly-typed unsafe language. As cplusplus.com forum user, Albatross states:

C++ is a language that expects the programmer to know what he or she is doing, but allows for incredible amounts of control as a result. [x]

The language also supports dynamic type checking, which has helped create the collision classes.

Java's garbage collection system constantly checks whether memory is still in use, whereas C++ trusts the programmer only to allocate and use what they need to. In a simulation where there are many calculations taking place every frame, this is rather important.

SFML, which provides the GUI for the project, was also primarily created for the C and .NET languages, which helped in the decision-making process.

### Visual Studio 2019

Visual Studio (VS) was my IDE of choice as I have used it many times for previous projects I have undertaken. The inbuilt IntelliSense feature has saved hours that would have been spent searching through class documentation by providing a robust code-completion aid. The debugger is also the best I have used and has been very useful to me throughout the year. Git integration out of the box has also made source control and backing up the project a swift process.

### Eigen

The Eigen header library was used for Vector2 representation and calculations. This was needed for representing positions and the forces used on elements of the simulation. I was tempted to create my own vector class; however, it seemed more sensible to use a simple header library and fill in the blanks where needed as the time would be better spent elsewhere.

### SFML

Simple and Fast Multimedia Library provides the ability to apply a simple interface to Windows applications to help with the development of games. There are five modules within the library, of which I used three. These three were the system, window, and graphic modules. The alternative option I had was Simple DirectMedia Layer (SDL) which is a library that provides low level access to graphics using Direct3D and OpenGL. SDL had a couple of problems which made me go with SFML. SDL is not very beginner friendly and learning to use it effectively would take more time than what would be worth for the project. SDL can also require extra external libraries depending on what you want to do, whereas SFML is completely self-contained for the purpose I required it for.

## Application Development Methodology

I had the choice of a couple methodologies I could have used for the project, chiefly agile or waterfall. I ended up choosing waterfall for a couple of reasons. I had a good idea of what I wanted to develop from the very beginning of the project. Since then, the aim and objectives for the project have stayed the same and I am using the same requirements I created around that time. I have also stayed true to my original plan, created back in November. This resulted in me falling into a waterfall-styled development methodology quite naturally as the project progressed. The implementation stage of the project was structured and followed the project plan, rarely needing me to go back to alter past code. Although I did investigate agile and its benefits, I felt that a more structured overall plan would help me to complete the project on time and to the specification I had created.

## Algorithm Development

### Planning

Although I wanted the algorithm to be customisable by the user, there were a couple of factors that either could not be changeable or I felt were not necessary for the project. One of these factors was the population initialisation. The two main methods for this were random initialisation and heuristic initialisation.

The problem with the heuristic approach was that there is not a known heuristic for the problem I am trying to solve. I also did not want a fully random approach as I soon found out while experimenting with the early stages of the simulation, extreme values can ruin the entire simulation. For example, if one organism has an incredibly high speed and size it will hit other organisms, in turn, pushing them towards or away from food and poison they otherwise would have found. Although realistically this could happen, this population is specifically designed to ignore each other. The approach I went with was to use a mixture of both. I experimented with a reasonable range of values that a gene could have to begin with and then the population was initialized with a range of these semi-random values.

The other factor I addressed before the project took place was the population model for the algorithm. The two most popular approaches are the steady state model whereby just a couple of organisms are replaced in each generation with new offspring, and the generation model where the entire population is replaced each generation. I found that the difference between the choices was too drastic, with steady state being too slow for a problem where generations can last over a minute long. Meanwhile, generational runs the risk of deviating away from the best solution by removing potentially perfect organisms. Once again, I compromised to a solution, I believed would be better. By default, each generation would keep 50% of its population through the selection process and new offspring will be created from this 50% through crossover and mutation.

I had to also decide on a fitness function suitable for this problem. Initially it would just take the populations current health values when the generation ends and use that, however I ran into an issue. What if the entire population died before the generation ended? Slightly less drastically, what if any 10% of the population died before the generation ended. It would not make sense that two organisms that died at different times would have the same fitness value. So, I decided to track each organism’s lifetime and current health, combining these to create the fitness value. This means that dead organisms will be ranked by the time they stayed alive and living organisms will be ranked by their current health.

### Algorithm Overview

The algorithm is called at the end of a generation within the main evolution simulation class. These are the steps it takes to produce a new population of organisms:

* Fitness values for each organism calculated.
* Selection process is carried out.
* Crossover is performed to create offspring.
* Mutation alters the populations’ chromosomes.
* The new population is prepared for insertion into the simulation and passed back into the main simulation class.

The parameters for the algorithm are selected at runtime by the user which can then be directly accessed by the genetic algorithm class. Below is the implementation for the general methods listed above.

Sort Population by Fitness

This method simply sorts the population vector by their fitness values, which are calculated by adding together the lifetime of the organism and its current health when the generation ended.

#### Selection Process -> Crossover Process -> Mutation Process

Each of these methods check which parameters the user has chosen at runtime and picks the correct function. Below is a code snippet from the mutation process showing how this is achieved.



Figure ‑ Mutation Process Code

The crossover process method also randomises the selected population and pairs them, in preparation for mating via crossover.

#### Create New Population

This method goes through the list of organisms and places their chromosome into a new organism. This is to ensure any previous data held on an organism from the last generation is removed and the simulation can have a completely fresh set of organisms. This list of organisms is then returned to the simulation class, ready to be placed into the simulation.

### Selection Implementation

#### Roulette Wheel Selection & Stochastic Universal Sampling (SUS)

Roulette wheel selection keeps a variable that contains the total of all fitness values from the population. A random value between zero and the total fitness is then created. Next, a loop goes through the population, adding their fitness value to a temporary variable and comparing this temporary variable against the random value. If the random value is less than or equal to the temporary variable, this organism is added to the list of organisms to be mated and placed in the next generation.



Figure ‑ Roulette Wheel Selection Code

SUS is implemented in a similar fashion to roulette wheel selection, except two random values are pulled from the ‘wheel’ instead of just one. The loop searches for the first value first and then it is repeated for the second value.

#### Random Selection

This is quite a simple method. It calculates 10 random, unique numbers between 0 and the population size. These are used as indexes for the population vector. N number of organisms from the population are then added to the new list of organisms, based on the random indexes. N in this case would be the pre-determined size of organisms to be selected for crossover and mutation.

#### Tournament Selection

This function first loops through the number of participants in a tournament (the tournament size) and adds that number of random organisms from the population to a temporary vector. This vector is then sorted by the fitness values of the organisms within. Finally, the N fittest organisms from that vector are added to the list of organisms to be mated and placed in the next generation, where N is the number of survivors. Both the tournament size and survivor amount can be modified.



Figure ‑ Tournament Selection Code

### Crossover Implementation

#### Uniform Crossover

The chromosomes from both organisms in a mating pair are looped through. For each gene, in both chromosomes, there is a 50% chance they are swapped in the offspring. A diagram detailing from tutorialspoint.com is below. [x] These offspring are then placed, with the parents, in a vector ready to be passed to the mutation function. This process is repeated for each mating pair.



Figure ‑ Uniform Crossover [X]

#### Single-Point Crossover

For single point crossover, a random point along the chromosomes of the parents is chosen using the program’s random engine. The tail end of each chromosome is then swapped in the offspring. Both new organisms and the parents are then moved to a vector to be passed to the mutation function. This process is repeated for each mating pair.



Figure ‑ Single Point Crossover [X]

#### Multi-Point Crossover

The multiple point crossover function is very similar to the one above, except this time there are two dividing points on a chromosome. To get these separate points, I created a vector containing all indexes in a chromosome. The vector is then shuffled, and all but the first two values are removed. This remaining vector would now contain the two dividing points.



Figure ‑ Multiple Point Crossover [X]

### Mutation Implementation

I have split mutation into two sections. Section one is optional and contains position swapping mutation methods such as scramble, inversion, and swap mutation. These methods do alter the chromosomes but do not introduce new gene values into the population. This limits the extent at which the population’s overall fitness can improve, but does add variation between generations, especially when two or more of the same organisms have made it into the new population. The second section of mutation is compulsory and consists of creep mutation and random resetting. These methods introduce new genes into the gene pool and are key to evolving the population beyond the first few generations.

#### Scramble & Inversion Mutation

The mutation methods start off by doing a probability check to see if the mutation chance condition is met. For scramble mutation, a vector containing all indexes in a chromosome is created. This vector is shuffled, and the first two values are sorted and placed into a separate vector. These two values act as a subset of the chromosome. The genes within the subset are also randomly shuffled and replace the values previously in the subset of the chromosome. This is then repeated for each organism in the population. Inversion mutation is the same as scramble in the sense that a random subset of each chromosome is altered. The difference is, instead of scrambling the subset, it is instead inverted. Below I have included a code snippet demonstrating how scramble mutation is carried out.



Figure ‑ Scramble Mutation Code

#### Swap Mutation

Swap mutation is quite simple. If the mutation probability check is successful, then a gene in the chromosome is swapped with another gene. This is then repeated as if it is only done once, the other mutation methods are more likely and more capable of having a larger impact on the population.



Figure ‑ Swap Mutation Code

#### Creep Mutation

Creep mutation occurs per each gene rather than against the whole chromosome. For each gene there is a chance that a value between -X and +X is added onto that gene’s current value. The X is chosen by the user when the program begins if the user selects creep mutation.



Figure ‑ Creep Mutation Code

#### Random Resetting Mutation

Random resetting is also computed per gene, like creep mutation above. The only difference between the two is that creep mutation adds a value onto the current gene whereas random resetting completely replaces this gene’s value with another value. The bounds for this value are chosen by the user when the program begins. This is the code that sits within the for loop for each organism.



Figure ‑ Random Resetting Code

### Algorithm Output

At the start of the algorithm where the fitness values are calculated the algorithm outputs each organism’s chromosome, as well as its calculated fitness. This also includes the best fitness within the population as well as the average fitness for that generation.



Figure ‑ Algorithm Output

## Organism Development

### The Chromosome

Despite being a crucial aspect of the project, an organism’s chromosome is represented simply as a vector of floating-point values, each corresponding to a gene. The default values for these genes fall between 80 and 120.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Position | 0 | 1 | 2 | 3 | 4 | 5 |
| Gene | Normal  Speed | Food/Poison  Impulse | Body Size  (Mass) | Base Health | Food  Radius | Poison Radius |

Figure ‑ Chromosome Representation

Normal Speed – Controls the speed at which the organisms travel with no other influences.

Food/Poison Impulse – The impulse speed applied to an organism when in range of food/poison.

Body Size (Mass) – The size of the organism, which also correlates to its mass. Heavier organisms also move slower due to their mass. This means an organism with a high normal speed could be slower than a lighter organism with a lower normal speed.

Base Health – The health which the organisms are ‘born’ with. Each frame this decreases by 0.10.

Food Radius & Poison Radius – The visual radius in which organisms can detect food or poison.

### Normal Movement

As shown below, the standard movement for organisms is quite simple. A random direction is chosen each frame and a force is added to the organism in that direction, the strength correlating to its normal speed gene.



Figure ‑ Organism Movement Code

### Food and Poison

Each frame, the organism loses 0.10 from its current health. The only way for the organism to gain health is through consuming food in the environment, restoring 25 health. However, if the organism consumes poison, it loses 40 health. Considering base health in generation 1 starts between 80 and 120, there is a great incentive for organisms to learn to avoid poison.

Each organism has a visual radius in which they can see food and poison. If they see food, then they will start to move towards to it at a speed relative to their food/poison impulse gene. Alternatively, if they are in range of poison, they will have a chance each frame to move away from the poison. Both their chance to move away and the speed in which they do so is relative to the food/poison impulse gene.

## Physics Engine Development

### High Level Explanation

1. The main simulation loop calls the physics engine update method, passing the timestep.
2. The physics engine calculates the result of the accumulated forces on each body.

(Acceleration Integration)

1. Forces calculated from any collisions are accounted for by the physics engine.

(Collision Detection & Collision Resolution)

1. New positions for each body in the simulation are integrated for the next frame.

(Velocity Integration)

### Rigid Bodies

Rigid bodies are the objects in which the simulation is made up of. As none of the elements in the simulation change shape over time, soft bodies are not needed so that leaves us with rigid bodies. Each of these bodies holds a position, a mass, and a volume in which the mass is spread across.

The mass for these bodies is stored as inverse mass, or . This is done for a couple of reasons. Instead of dividing, the CPU can multiply instead, which is slightly faster. Although this is does not have a huge impact, it can be noticeable in larger systems with several thousand bodies. A more important reason is that it allows the simulation to have bodies with infinite mass. If a body has an inverse mass of zero, it is infinitely heavy. This is useful for walls as I do not want organisms to be able to push the boundaries of the environment away from the screen.

### Integrating Acceleration

Each frame, each body holds all the accumulative forces that have been applied to it. When the physics engine updates, it takes all the forces and calculates the resulting acceleration. The formula used for this is . A temporary velocity variable is then calculated using this acceleration and the timestep passed from the main loop.



Figure ‑ Acceleration Integration Code

This velocity cannot be used yet as it has not considered collisions and how they might have affected the organism’s movement. You can see in the figure above that the bodies store a variable named ‘damp\_’. This variable stores a value just below 1 (0.999) and ensures that objects slow down over time. This is simulation air resistance and friction without the need to calculate these values. Although this slightly disobeys Newton’s first law, it is adequate for the simulation I am creating.

### Collision Detection/Resolution Within the Physics Engine

It was difficult to decide where to talk about the following implementation of collision detection and resolution. I decided that because the velocity integration is first dependent on collisions being resolved I would talk about resolution here and the information the physics engine needs to resolve collisions from the collision detection class. Below you can see code from within the SimPhysics class which loops through each pair of colliders in the system and checks whether they are colliding. Specifics on how collisions are detected will be covered in chapter 3.9. The result from this method returns three values; true or false depending on whether they are colliding, the penetration distance of one collider into another, and the normal vector which in this case is the direction of least penetration. If they are colliding, both colliders are passed into a collision pair object, which is then added to a list of collisions.



Figure ‑ (Physics Engine) Collision Detection Code

I have split the resolution into two halves, penetration resolution and velocity resolution. Penetration resolution is moving the two objects out of each other. This may sound as simple as just moving each object an equal distance away from each other but then I would run into the problem of a wall moving whenever an organism collides with it. To solve this, I needed to make each object move proportionally to their mass. The bottom two lines show how this is achieved. If the object has an infinite mass, as the wall would, their position change is multiplied by ( ). If the inverse mass is 0, this means the wall would not move at all, while the other collider will move all the way out of the wall.



Figure ‑ Collision Penetration Resolution

The second half of the resolution needs to calculate the movement change over time caused by the collision as organisms bounces off each other or off a wall. This is achieved by directly manipulating the velocity through an impulse.



The impulse is a vector as it is a derivative of force. The complicated part is now working out the impulse vector, referred to as ‘j’ in the code below. The impulse needs to both get the bodies travelling in the correct direction (which is opposite to each other due to Newton’s Third Law) as well as conserving the momentum of the bodies. This is achieved in a few steps.

As seen above in the research portion of the project, the equation for the impulse is as follows:

The bottom half of the equation is simple the total mass which I have already saved as a variable from above. This leaves me with , which is the coefficient of restitution and , which is the dot product of the relative velocity, and the collision normal. You can see the dot product of relative velocity and the normal calculated below, but the coefficient of restitution is simply 0.8. This value was chosen as a somewhat middle ground between 1, where the objects would lose no magnitude on collision and 0, where the objects would come to a complete halt.



Figure ‑ Collision Velocity Resolution

### Integrating Velocity

Once all collisions have been resolved and all previously applied forces have been accumulated and integrated, the new position of each body a be determined by integrating the velocity. This marks the end of the current frame and will set the positions for the next frame. This is the physics engine loop in its entirety.



Figure ‑ Velocity Integration Code

## Collision Detection Development

Each simulation object stores its own collider object. Colliders have several properties and methods which aid in detecting collisions between objects. The properties include:

* Collider area.
* Position of collider.
* Radius / Width / Height.
* Colour.
* Collider Type (Organism, food, wall, etc...).
* Collider Shape (Rectangle/Circle).
* minExtentX & maxExtentX (used for calculations)

### Circle/Circle Collision

Perhaps the simplest of the collision detection methods, circle to circle collision simply checks whether the distance between the centres of two circles is less than the sum of their radii. If this is true, the circles must be colliding. This leaves the penetration distance and collision normal to be calculated.

The penetration distance is just the radii of the circles, minus the distance between their centres. The collision normal is the line vector (variable ‘delta’), normalized. This leaves you with a unit vector, giving the direction between the two circles.



Figure ‑ Circle/Circle Collision Code

### Rectangle/Circle

Collisions between circles and rectangles can be detected by clamping the sphere’s position to the minimum and maximum extent of the rectangle on both axes. This gives you the closest point on the rectangle to the circle. If the closest point is less than the radius of the circle away from the centre of the circle, then the two objects are colliding. Luckily, C++ has a built-in clamp function as of recently.

Finding the normal vector and penetration distance is now an easy task. The normal vector is found by normalizing the line vector between the circle’s centre and the closest point on the rectangle. This gives you the direction vector between the two points. The penetration distance is just the difference between the radius and the distance between the closest point and the circle’s centre.



Figure ‑ Rectangle/Circle Collision Code

### Rectangle/Rectangle

Although I had already implemented it by the time I realised, I did not need to check for collisions between rectangle objects in the simulation. As the moving elements within the simulations all use circle colliders, only the walls of the environment are rectangular and do not move throughout the course of the program’s runtime. It is a shame that this was the case as working out the penetration distance and normal vector for rectangle-to-rectangle collisions was the most complicated and took more time than the others. I will include the code for this collision detection method in Appendix 4.

## Renderer Development

The rendering on the project is mostly managed by the SFML graphics library, this class just takes advantage of the capabilities of the library and provides a way to draw the elements I need in an efficient manner. The class contains all the drawing functions needed for the program as well as holding the window object itself. The four drawing functions included are:

* DrawCircle: used for drawing food and poison elements.
* DrawBox: used for drawing the outer walls of the environment.
* DrawOrganism: used for drawing the population.
* DrawHealth: used for drawing the organism’s current health above it.

Main.cpp is where the program begins, creating a new renderer object. Using the SFML library, a window is then created at a resolution of 1280 by 720 and a framerate limit is set. I chose 720p as 16:9 is the most common aspect ratio for modern computer monitors and laptop. Also, even cheaper laptops tend to have a resolution no smaller than 1366 by 768. In the worst case where the resolution is smaller than this, the program can be resized, only altering the simulation for a few frames.

At this point a new EvolutionSimulation object is created, with the window object passed into its constructor, allowing elements to be drawn outside of main.cpp. Next the main game loop begins.

## Global Header

This file holds all global variables used throughout the program. Throughout the development of the project, it would store variables that I had not yet decided how to use or where to use them. Some of the variables it holds include:

* Population size.
* Number of food pieces.
* Number of poison pieces.
* The window resolution.
* The framerate limit.
* Default mutation chance.
* Default crossover chance.
* Default tournament size.
* Default survivor amount.
* Default generation time (in seconds).

When the user is prompted for the properties of the program, the answers are stored in the global header for use throughout runtime.

# Testing

## Overview

A large portion of the program is non-deterministic and there is no exact expected outcome for a lot of functions. Couple with the stochasticity of the genetic algorithm mean that visual testing is going to an important factor in testing the solution as well as some manual white box testing. Collision detection and resolution is an aspect of the program that is deterministic and for any two shapes in any two positions, the program should know whether they have collided, how far they have collided and where they have collided.

## Testing Solution

### Element Creation

Each time the program is ran, the user chooses the number of organisms, food pieces and poison pieces present in the environment. The UI should show the correct number of these elements in different positions across the environment.

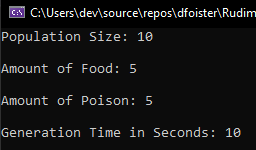
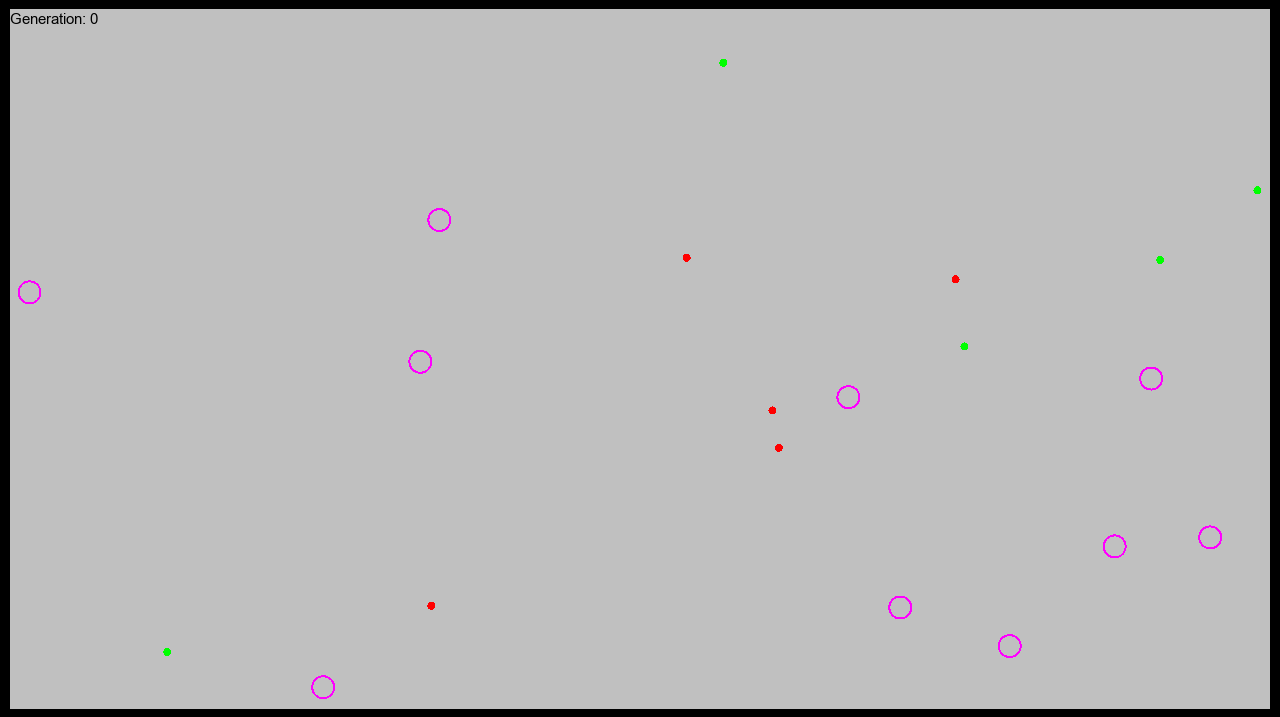


Figure ‑ Program Start

Figure ‑ Program Running with Numbered Elements



1

2

4

5

6

8

7

9

10

1

2

3

4

5

1

2

3

4

5

### Selection Methods

#### Roulette Wheel Selection

#### Stochastic Universal Sampling

3

#### Random Selection

### Crossover Methods

For all crossover testing the same parameters have been used:

* Population: 20.
* Food: 1.
* Poison: 1.
* Generation Time: 1 second.
* Selection Method: Roulette Wheel.
* Mutation: Disabled
* Survivors: 10

As crossover does not take fitness or mutation into account, I was able to disable mutation completely and reduce other factors down to nearly nothing. This mean all chromosomes were selected nearly randomly but will not affect the outcome of the crossover procedure.

#### Uniform Crossover

The figure below is output from the uniform crossover method. Chromosomes 1 and 2 represent the parent chromosomes, whereas three and four represent the children created via uniform crossover. Here you can see that each gene has remained in the same position as it did in the parent chromosome but each child alternates randomly which gene it receives.

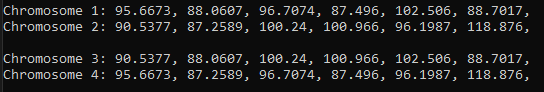


Figure ‑ Uniform Crossover Test

#### Single Point Crossover

Like the test above, chromosomes 1 and 2 are the parent chromosomes and the bottom two are the offspring. The output of the function shows that the algorithm is successfully splitting the parent chromosomes at a single point and assigning each split to the correct child chromosomes.

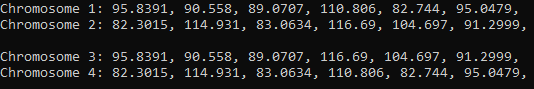


Figure ‑ Single Point Crossover Test

#### Multiple Point Crossover

Multiple-point crossover is almost identical to single point except, as the name suggests, the chromosomes are split at multiple points. In this GA, they are split exactly twice. Although in this test, they are split evenly into three slices, containing two genes each, the two splits can happen anywhere on the chromosomes.

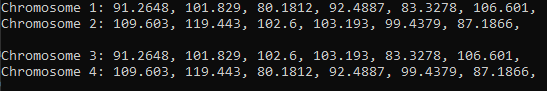


Figure ‑ Multiple Point Crossover Test

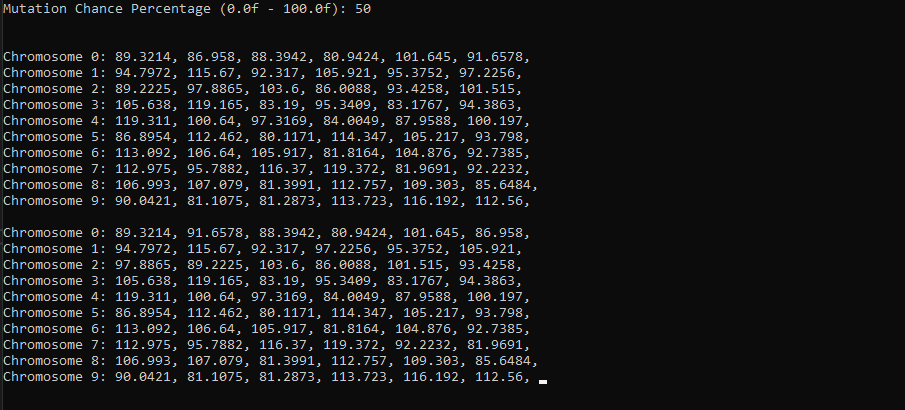
### Mutation Methods

For the mutation tests, I printed 10 random chromosomes to the console and then passed them into the function, along with a mutation chance. I then outputted the same chromosomes after the function had completed to see whether the chromosomes had been mutated correctly.

#### Swap Mutation

As discussed in section 3, if the swap mutator is called, it will swap one or two random genes on a chromosome with two other random genes on that same chromosome. With a mutation chance of 50%, we should expect around half of the 10 chromosomes to be mutated, with one or two genes swapped. I will highlight the mutated chromosomes in green on the figure below. Once again, the top set of chromosomes are the originals, and the bottom set are the chromosomes after the function has ran.

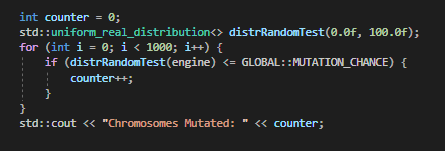
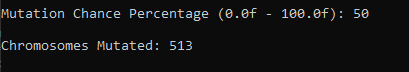
Figure ‑ Swap Mutation Test



As you can see, I have highlighted six out of the ten chromosomes. With a mutation chance of 50% and only 10 chromosomes, 60% is an acceptable result. To show that the random engine I am using is reliable I also tested whether the mutator would have been activated if called 1000 times with a mutation chance of 50%. The results of this are below.

Essentially, a random number is generated between 0 and 100. If this number is 50 or less, then the mutator would have been called and therefore a counter is incremented. Out of 1000 calls, the counter is incremented 513 times, or 51.3% of the time, which is more than acceptable for the simulation.

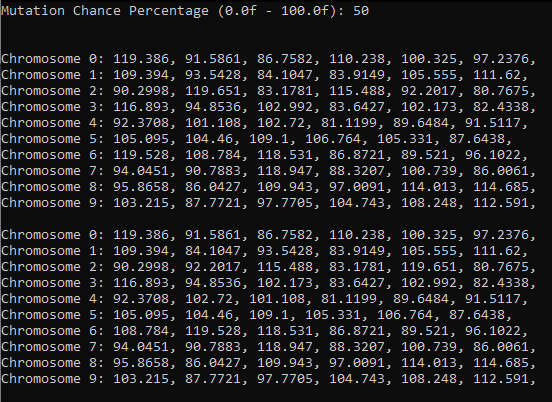
Figure ‑ Swap Mutation & Random Engine Test



#### Scramble/Shuffle Mutation

If the scramble mutator is called, a random subset of that chromosome has its genes shuffled. Below is output from this function when 10 random organisms have been passed to it. A mutation chance of 50% has been used once again. On the left I will highlight which chromosomes have been mutated and then on the right I will highlight the subsets which has been scrambled, with colours corresponding to the original chromosome.

Figure ‑ Scramble Mutation Test



#### Inversion Mutation

This test is almost identical to the last test except instead of scrambling the random subset, it is instead inverted. Here you can see exactly 50% of the chromosomes are mutated and each has a randomly inverted subset.

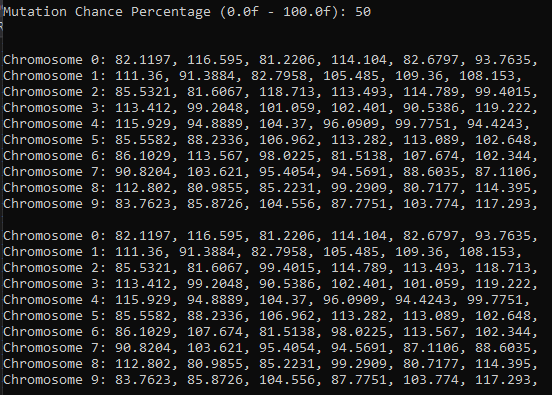


Figure ‑ Inversion Mutation Test

#### Creep Mutation

Creep mutation carries out mutation on single genes rather than the entire chromosome. For this reason, I lowered the mutation chance to 10% for the test to avoid cluttering the figure below. Once again, I have highlighted the mutated genes and colour coordinated them with the gene from the original chromosome. For this test, the creep value was set between -15 and +15 and as you can see, all mutated genes fall into this range. The percentage of mutated genes in this example was 11.6%, landing very close to the chosen 10% mutation chance.

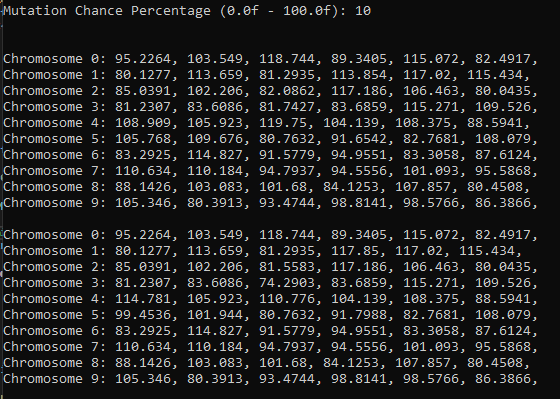


Figure ‑ Creep Mutation Test

#### Random Resetting Mutation

As random mutation also performs against single genes rather than the entire chromosome, I kept the mutation chance to 10 for this test. I set the lower bounds of the random value to 1000, and the upper bound to 2000. I did this to make it clearer where mutation had occurred. When attempting to use the program for the purpose of evolution simulation I would not use bounds as high as these. Once again, the rate of mutation was 11.6% in this iteration, falling close to the 10% target.

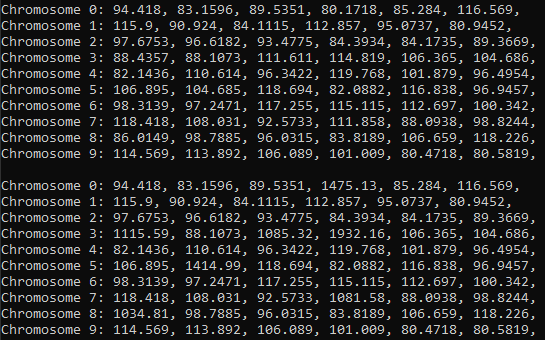


Figure ‑ Random Resetting Test

### Collision Detection

The best way to test collision detection and resolution would be to show the simulation in action, however that will not be possible within this document. The next best thing I could do was output the collisions as they were being resolved, detailing the types of objects, their positions, the collision normal, and the penetration distance. These collisions were outputted within the collision resolution method, as there is a loop which loops through each pair of colliders that had collided that frame. Within the console output you can see collisions between an organism and a piece of food, two organisms colliding with each other, as well as the same stubborn organism repeatedly headbutting a wall.

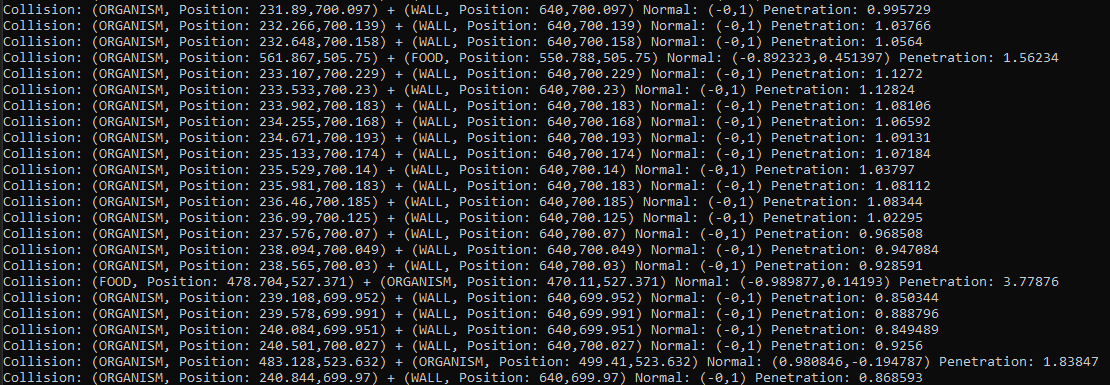


Figure ‑ Collision Detection Test

# Evaluation

## Results

I figured that an important part of this dissertation would be demonstrating that the genetic algorithm can allow the organisms’ fitness to increase over time through evolution of their genes. This section of the evaluation will then look at certain parameters that can be used to achieve this result. This also gives a chance to demonstrate all functions in the GA working together to create new populations.

### Example One

Full output from program can be found at the end of the document, Appendix 1.

#### Parameters

1. Population Size: 20
2. Amount of Food: 20
3. Amount of Poison: 15
4. Generation Time (seconds): 25
5. Population Survivors: 50% (10)
6. Selection: Roulette Wheel
7. Crossover: Multiple-Point
8. Primary Mutation: Creep
9. Creep Value: 25
10. Secondary Mutation: Scramble
11. Mutation Chance: 15%

#### Results

|  |  |  |
| --- | --- | --- |
| Generation | Average Fitness | Best Fitness |
| 1 | 135.3 | 197.496 |
| 2 | 136.316 | 200.08 |
| 3 | 130.65 | 212.531 |
| 4 | 157.733 | 223.023 |
| 5 | 155.392 | 252.843 |
| 6 | 163.34 | 244.631 |
| 7 | 164.979 | 262.588 |
| 8 | 158.092 | 309.657 |
| 9 | 163.951 | 313.738 |
| 10 | 164.345 | 332.006 |
| 11 | 173.789 | 311.109 |
| 12 | 175.209 | 325.14 |
| 13 | 173.018 | 319.547 |
| 14 | 171.242 | 324.109 |
| 15 | 177.129 | 357.613 |

Figure ‑ Result One Table

Figure ‑ Result One Average Fitness

Figure ‑ Result One Best Fitness

#### Evaluation

### Example Two

Full output from program can be found at the end of the document, Appendix 2.

#### Parameters

1. Population Size: 20
2. Amount of Food: 15
3. Amount of Poison: 20
4. Generation Time (seconds): 25
5. Population Survivors: 50% (10)
6. Selection: Tournament (Size: 2, Survivors: 1)
7. Crossover: Uniform
8. Primary Mutation: Creep
9. Creep Value: 25
10. Secondary Mutation: Swap
11. Mutation Chance: 15%

#### Results

|  |  |  |
| --- | --- | --- |
| Generation | Average Fitness | Best Fitness |
| 1 | 103.398 | 169.747 |
| 2 | 105.177 | 179.358 |
| 3 | 107.448 | 197.841 |
| 4 | 116.691 | 200.594 |
| 5 | 118.482 | 200.17 |
| 6 | 106.396 | 219.849 |
| 7 | 110.786 | 205.577 |
| 8 | 117.95 | 206.773 |
| 9 | 130.619 | 212.15 |
| 10 | 103.806 | 225.187 |
| 11 | 118.345 | 225.571 |
| 12 | 116.21 | 239.245 |
| 13 | 124.619 | 242.864 |
| 14 | 129.155 | 256.686 |
| 15 | 132.72 | 275.183 |

Figure ‑ Result Two Table

Figure ‑ Result Two Average Fitness

Figure ‑ Result Two Best Fitness

#### Evaluation

### Example Three

Full output from program can be found at the end of the document, Appendix 3.

#### Parameters

1. Population Size: 16
2. Amount of Food: 15
3. Amount of Poison: 15
4. Generation Time (seconds): 30
5. Population Survivors: 50% (8)
6. Selection: Random
7. Crossover: Single-Point
8. Primary Mutation: Creep
9. Creep Value: 20
10. Secondary Mutation: None
11. Mutation Chance: 10%

#### Results

|  |  |  |
| --- | --- | --- |
| Generation | Average Fitness | Best Fitness |
| 1 | 121.997 | 208.129 |
| 2 | 76.2252 | 152.227 |
| 3 | 90.9262 | 226.323 |
| 4 | 113.822 | 159.69 |
| 5 | 113.931 | 214.493 |
| 6 | 97.7446 | 203.057 |
| 7 | 75.1091 | 173.03 |
| 8 | 82.2169 | 163.003 |
| 9 | 81.285 | 127.108 |
| 10 | 95.028 | 162.917 |
| 11 | 97.3042 | 209.287 |
| 12 | 78.382 | 123.886 |
| 13 | 88.0719 | 134.109 |
| 14 | 106.922 | 218.997 |
| 15 | 114.133 | 214.193 |

Figure ‑ Result Three Best Fitness

Figure ‑ Result Three Average Fitness

Figure ‑ Result Three Best Fitness

#### Evaluation

I kept these results in to show that the population is not guaranteed to improve over the course of 15 generations. This can be due to several factors, such as the parameters of the GA not being suitable for evolution to occur. In this instance, random selection is used. There is no selection pressure, meaning that the weakest organisms in the population have the same chance of producing offspring and continuing to the next generation as the fittest organisms. The smaller population size leaves less room for outstanding organisms to develop by chance and continue to improve the population. Also, the combination of single-point crossover and no secondary mutation decreases the genetic diversity of the population, meaning fewer gene combinations can be searched by the solution each generation. Another factor in the poor performance is simply luck. Genetic algorithms are not guaranteed to come to a suitable solution due to the random factors involved in the selection and reproduction process.

## Requirements Evaluation

### Functional Requirement Evaluation

#### At program launch, the system must provide the user with a list of options regarding altering the GA and number of elements in the environment.

I feel confident in saying that this requirement has been met as the user is prompted with a console window on launch of the program. You can refer to [Figure 4-1](#_GUI_Design) to see how this prompt is displayed and the options the user is presented with. Although this requirement has been fulfilled, it deviates from the original plan of a GUI with sliders, present in the simulation window.

#### The system must display all organisms, food elements, poison elements and the current generation number in a window, all in the correct positions according to the physics engine.

#### The console should output the current statistics of the genetic algorithm during, and after each generation.

The console outputs each organism, its chromosome, each fitness value as well as the best and average fitness for each generation. In that sense the requirement has been met and is sufficient for the project. However, I do think more statistics could be present, such as which organisms were selected, where crossover occurred, and which organisms were mutated. It would definitely clutter the console, but I could have found another method of presenting the information.

#### At the end of each generation, all elements in the scene should be replaced with the new elements created by the GA.

The results posted in the previous section show that this requirement has been met. This requirement was either achieved or not achieved, leaving little room for possible improvements or alteration, making it a definite success.

#### The system should detect collisions between elements and pass this information to the physics engine for resolution.

Figure 4-12

#### When a collision is detected, the velocity and position of affected elements should be handled accurately by the physics engine and displayed to the user.

#### The GA should be able to carry out multiple operations for each of the following: population selection, chromosome crossover, and gene mutation.

### Non-Functional Requirement Evaluation

#### The system should run on Windows 10 64-bit/32-bit and Windows 7 SP1

I have tested the system, in its current state on a 64-bit version of Windows 10 Pro, running on the latest version. The program runs without needing any additional software. I have also tried a slightly older build on a 32-bit Windows 7 laptop with Service Pack 1 installed. Once again, the program ran with no issues. There is no reason the program should not also run on Windows 8 but unfortunately I do not have the means to test this, therefore I could not add it as a requirement.

#### The system should support the following hardware: A video card that supports a minimum resolution of 1280x720. Ideally a resolution of (1366x768) should be in use.

The program window itself runs at a resolution of 1280x720 so any computer running at this resolution should have no issue running the program. I found on a laptop running at 1366x768 that you could run the program while maintaining a view of the console which I believe to be the ideal way to operate the program.

#### The system should support the following hardware: At least 50MB free disk space.

I chose 50MB as I did not believe the program would ever require that much storage space and figured it would be a suitable target to stay under. The release build of the program currently sits at 33.8MB, which meets the requirement handily, although I did not believe it would end up this large.

#### The system should support the following hardware: 4GB of RAM.

I have only been able to test the system with a minimum of 6GB of RAM. Therefore, I cannot say that this requirement has been met fully. However, even with 50 organisms, 50 food objects, and 50 food objects the program never surpassed 80MB memory usage during the testing process.

# Conclusion

## Satisfaction of Aim & Objectives

### Aim

The aim of this project was to explore how a genetic algorithm can be manufactured to simulate evolution amongst artificial organisms, providing an interactive interface where parameters can be altered to adjust the effectiveness of evolution.

### Objectives

#### Theorize and create a genetic algorithm that increases the calculated fitness of an organism, effectively evolving it.

The algorithm will need to be extensive, containing numerous methods which can be changed by a user to alter the effectiveness of the algorithm.

#### Implement a GUI which displays organisms, other entities, and the current statistics of the genetic algorithm.

The interface is needed to display the organisms moving in the environment, finding food, and avoiding poison. The program would work without it but is needed for the user to see the evolution in action.

#### Create a movement system whereby organisms choose where to move based on their size, quality of their vision, and nearby food and poison positions.

The movement system will require the creation of a physics engine that can accurately determine the position of organisms based on numerous factors. This will take a lot of research and way in which each organism moves will be partly based on their genes.

1. Create a collision detection system that allows organisms to interact with food/poison entities and have their health affected accordingly. This will be visible in the program.

Without a collision system, there is no way to keep organisms within the confines of the environment, able to consume food, and able to avoid poison. The organisms would also be able to travel through each other without it.

#### Program a death and reproduction cycle which will remove the previous organisms, reproduce them using the genetic algorithm and create a new set of organisms with new chromosome values.

This is key for the actual evolution to occur as there will need to be many generations for the organisms to evolve and become more suited to the environment. The genetic algorithm will manage the reproduction.

#### Introduce an interactive component to the GUI that would allow a user to modify the algorithm's specific parameters to change how fast or effective the evolution of organisms is.

This is another crucial objective as it makes up the entire interactive portion of the project. This will most likely consist of sliders for mutation/crossover chance and the option to change the selection method the algorithm uses.

## Summary

## Future Work

# References

[1] – Bacardit, J 2020, *L02 – Genetic Algorithms,* lecture notes, Biologically-Inspired Computing CSC3423, Newcastle University, delivered 18th October 2020.

[2] – Rooy, N 2017, *Evolving Simple Organisms using a Genetic Algorithm and Deep Learning from Scratch with Python,* Nathan Rooy, viewed 20th January 2021, <<https://nathanrooy.github.io/posts/2017-11-30/evolving-simple-organisms-using-a-genetic-algorithm-and-deep-learning/>>

[3] – Stanford University 2020, *graphics.h,* Stanford University, viewed 12th January 2021, <<https://web.stanford.edu/class/archive/cs/cs106b/cs106b.1126/materials/cppdoc/graphics.html>>

[4] – Thompson, J 2020, *Collision Detection,* Jeffrey Thompson, viewed 20th December 2020, <<http://www.jeffreythompson.org/collision-detection>>

[5] – Speirs, N 2020, *C++ Lecture Material*, lecture notes, Programming for Games CSC3221, Newcastle University, delivered 29th September 2020.

[6] – Mukhopadhyay, D, Balitanas, M, Farkhod, A, Jeon, S & Bhattacharyya, D 2009, ‘Genetic Algorithm: A Tutorial Review’, *International Journal of Grid and Distributed Computing,* vol. 2, no. 3, pp. 25-32.

[X] - TomsPlanner. 2009. *Tom's Planner - Gantt charts for the rest of us.* [online] Available at: <https://www.tomsplanner.com/>.

[X] - Eiben, A. and Smith, J., 2015. *Introduction to Evolutionary Computing (Natural Computing Series)*. 2nd ed. Heidelberg: Springer, pp.13-14.

[X] - Kumar, U., 2020. Genetic Algorithm. [Blog] *Analytics Vidhya*, Available at: <<https://medium.com/analytics-vidhya/genetic-algorithm-5aba4aac48f7>>.

[X] - Tutorials Point. n.d. *Genetic Algorithms - Crossover*. [online] Available at: <https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_parent_selection.htm>.

[X] - Eiben, A. and Smith, J., 2015. *Introduction to Evolutionary Computing (Natural Computing Series)*. 2nd ed. Heidelberg: Springer, pp.53-54.

[X] - Nave, R., 2017. *Newton's Laws*. [online] Hyperphysics.phy-astr.gsu.edu. Available at: <http://hyperphysics.phy-astr.gsu.edu/hbase/Newt.html>. [Accessed 13 April 2020]

[X] - Davison, R 2020, *Game Physics Part 2*, lecture notes, Computer Science CSC3222, Newcastle University, delivered January 2021.

[X] - Ponamgi, M., Cohen, J. and Lin, M., 1995. *Collision Detection for Virtual Environments and Simulations Using Incremental Computation*. [online] Gamma.cs.unc.edu. Available at: <http://gamma.cs.unc.edu/COLLISION_PON/collision.html#:~:text=Collision%20detection%20is%20a%20fundamental,interactions%20in%20the%20simulated%20environments.>. [Accessed 21 April 2021].

[X] - Workman, K., n.d. *Processing Collision Detection*. [online] Happy Coding. Available at: <https://happycoding.io/tutorials/processing/collision-detection>.

[X] - Cplusplus.com. n.d. *A Brief Description - C++ Information*. [online] Available at: <https://www.cplusplus.com/info/description/>

# Appendices

## Appendix 1

Population Size: 20

Amount of Food: 20

Amount of Poison: 15

Generation Time in Seconds: 25

Population Survivors (% of population): 50

Selection Type (Roulette = 0) (Random = 1) (Tournament = 2): 0

Crossover Type (Uniform = 0) (Single-Point = 1) (Multi-Point = 2): 2

Primary Mutation Type (Creep = 0) (Random Resetting = 1): 0

Creep Value Range: 25

Secondary Mutation Type (None = 0) (Swap = 1) (Scramble = 2) (Inversion = 3): 2

Mutation Chance Percentage (0.0f - 100.0f): 15

================================================================================

Organism 1: 113.313 85.2372 119.808 118.461 92.749 97.848 Fitness: 141.291

Organism 2: 106.152 92.13 95.468 102.885 95.2008 102.327 Fitness: 150.715

Organism 3: 107.462 101.312 95.8434 99.6068 99.3216 114.48 Fitness: 172.436

Organism 4: 98.4222 118.765 96.6203 114.695 88.9969 81.2363 Fitness: 147.526

Organism 5: 118.147 88.9171 96.4577 92.4511 92.5249 86.3127 Fitness: 115.283

Organism 6: 107.643 94.6827 91.508 97.2956 107.588 116.238 Fitness: 120.128

Organism 7: 86.7976 118.731 86.9917 103.335 95.1426 89.5373 Fitness: 176.165

Organism 8: 97.7984 89.888 115.991 109.126 99.654 92.0229 Fitness: 131.957

Organism 9: 84.4283 100.238 89.1774 83.321 97.0124 95.6693 Fitness: 81.1535

Organism 10: 95.4121 113.699 105.819 114.946 115.897 105.143 Fitness: 137.778

Organism 11: 100.202 110.069 112.949 117.696 80.5813 88.2403 Fitness: 140.527

Organism 12: 95.9409 95.7797 103.164 100.014 116.926 85.7798 Fitness: 172.843

Organism 13: 111.374 90.3081 113.908 90.6791 84.117 96.3705 Fitness: 188.509

Organism 14: 82.827 86.4151 119.231 84.476 106.802 88.0455 Fitness: 132.309

Organism 15: 90.8035 102.549 80.337 81.0589 115.579 97.7948 Fitness: 103.891

Organism 16: 95.6585 93.9073 119.44 80.1541 90.6262 107.213 Fitness: 102.987

Organism 17: 115.421 118.258 97.7698 106.368 84.3325 108.213 Fitness: 64.2009

Organism 18: 111.663 92.0951 82.2935 82.7765 92.4118 86.3396 Fitness: 130.609

Organism 19: 108.37 95.9617 99.8204 100.369 110.887 95.3489 Fitness: 98.2016

Organism 20: 84.7573 80.2138 109.667 99.6639 80.673 102.446 Fitness: 197.496

Best Fitness: 197.496

Avg Fitness: 135.3

================================================================================

Organism 1: 118.147 88.9171 96.4577 92.4511 92.5249 86.3127 Fitness: 164.837

Organism 2: 95.4121 121.596 105.819 114.946 115.897 105.143 Fitness: 162.33

Organism 3: 100.202 110.069 112.949 117.696 80.5813 88.2403 Fitness: 165.079

Organism 4: 95.4121 113.699 105.819 114.946 115.897 105.143 Fitness: 162.333

Organism 5: 100.202 110.069 112.949 117.696 78.609 88.2403 Fitness: 115.083

Organism 6: 111.663 92.0951 82.2935 82.7765 92.4118 86.3396 Fitness: 105.163

Organism 7: 113.313 85.2372 118.461 119.808 92.749 97.848 Fitness: 165.845

Organism 8: 107.643 94.6827 91.508 102.357 107.588 116.238 Fitness: 119.682

Organism 9: 98.4222 118.765 96.6203 114.695 88.9969 81.2363 Fitness: 137.08

Organism 10: 86.7976 118.731 86.9917 86.3261 95.1426 89.5373 Fitness: 100.722

Organism 11: 97.4234 101.093 105.819 114.946 92.5249 86.3127 Fitness: 122.332

Organism 12: 95.4121 113.699 96.4577 113.907 115.897 105.143 Fitness: 164.836

Organism 13: 100.202 110.069 108.283 117.696 88.2403 115.897 Fitness: 200.08

Organism 14: 95.4121 113.699 105.819 114.946 80.5813 105.143 Fitness: 137.331

Organism 15: 100.202 110.069 112.949 82.7765 86.3537 88.2403 Fitness: 115.163

Organism 16: 110.676 92.0951 82.2935 117.696 80.5813 86.3396 Fitness: 90.0832

Organism 17: 113.313 85.2372 115.791 97.2956 92.749 97.848 Fitness: 94.6826

Organism 18: 107.643 94.6827 119.808 118.461 107.588 116.238 Fitness: 165.848

Organism 19: 98.4222 118.765 108.432 103.335 88.9969 81.2363 Fitness: 100.722

Organism 20: 86.7976 118.731 96.6203 114.695 95.1426 95.5835 Fitness: 137.08

Best Fitness: 200.08

Avg Fitness: 136.316

================================================================================

Organism 1: 85.4963 118.731 86.9917 86.3261 112.996 95.943 Fitness: 83.9149

Organism 2: 118.147 88.9171 96.4577 70.6363 92.5249 85.5428 Fitness: 50.0393

Organism 3: 98.4222 118.765 108.432 120.281 88.9969 81.2363 Fitness: 175.92

Organism 4: 100.202 110.069 108.283 115.897 88.2403 117.696 Fitness: 190.283

Organism 5: 95.4121 113.699 105.819 114.946 80.5813 105.143 Fitness: 137.532

Organism 6: 121.693 118.765 108.432 103.335 88.9969 81.2363 Fitness: 100.924

Organism 7: 100.202 110.069 108.283 133.668 88.2403 115.897 Fitness: 90.285

Organism 8: 86.7976 118.731 96.6203 114.695 95.9339 117.189 Fitness: 87.2841

Organism 9: 95.4121 121.596 105.819 114.946 115.897 105.143 Fitness: 212.531

Organism 10: 77.5396 110.069 112.949 117.696 80.5813 111.507 Fitness: 115.284

Organism 11: 86.7976 89.5373 65.7197 86.3261 110.069 117.696 Fitness: 58.9149

Organism 12: 118.147 88.9171 96.4577 92.4511 99.1877 86.3127 Fitness: 65.0399

Organism 13: 98.4222 118.765 116.91 103.335 88.2403 81.2363 Fitness: 175.92

Organism 14: 100.202 110.069 108.283 117.696 88.9969 115.897 Fitness: 165.282

Organism 15: 95.4121 89.7178 115.76 103.335 80.5813 105.143 Fitness: 125.924

Organism 16: 98.4222 98.263 105.819 114.946 88.9969 86.9917 Fitness: 162.532

Organism 17: 100.202 110.069 96.6203 114.695 86.8282 115.897 Fitness: 97.2841

Organism 18: 86.7976 138.06 108.283 117.696 88.2403 95.5835 Fitness: 115.285

Organism 19: 95.4121 110.069 112.949 117.696 115.897 105.143 Fitness: 115.284

Organism 20: 100.202 121.596 105.819 114.946 80.5813 88.2403 Fitness: 87.5349

Best Fitness: 212.531

Avg Fitness: 130.65

================================================================================

Organism 1: 86.7976 118.731 121.414 114.695 95.9339 103.459 Fitness: 82.7746

Organism 2: 100.202 110.069 98.2808 117.696 88.9969 115.897 Fitness: 90.7757

Organism 3: 95.4121 121.596 105.819 114.946 115.897 105.143 Fitness: 138.026

Organism 4: 100.202 110.069 120.451 114.695 86.8282 115.897 Fitness: 137.772

Organism 5: 100.202 110.069 108.283 117.696 88.9969 110.429 Fitness: 90.7757

Organism 6: 96.9229 107.042 83.492 103.335 88.9969 81.2363 Fitness: 101.415

Organism 7: 100.202 110.069 108.283 115.897 88.2403 114.244 Fitness: 223.973

Organism 8: 95.4121 113.699 105.819 114.946 88.9969 80.5813 Fitness: 163.023

Organism 9: 77.5396 110.069 92.7676 128.273 102.364 111.507 Fitness: 90.7757

Organism 10: 100.202 121.596 105.819 114.946 56.6347 88.2403 Fitness: 263.022

Organism 11: 86.7976 118.731 96.6203 117.696 88.9969 117.189 Fitness: 190.772

Organism 12: 100.202 110.069 85.7649 114.695 95.9339 115.897 Fitness: 172.771

Organism 13: 95.4121 138.776 96.6203 114.695 124.573 105.143 Fitness: 147.772

Organism 14: 100.202 110.069 105.819 98.5248 86.8282 115.897 Fitness: 188.025

Organism 15: 103.335 110.069 81.2363 115.897 108.283 105.143 Fitness: 140.775

Organism 16: 121.693 118.765 102.402 103.335 88.9969 81.2363 Fitness: 126.415

Organism 17: 100.202 119.077 83.8686 114.946 80.5813 117.696 Fitness: 223.023

Organism 18: 95.4121 113.699 108.283 115.897 88.2403 105.143 Fitness: 188.976

Organism 19: 77.5396 110.069 112.949 114.946 80.5813 111.507 Fitness: 163.026

Organism 20: 100.202 116.835 82.9908 117.696 80.5813 76.2126 Fitness: 190.772

Best Fitness: 223.023

Avg Fitness: 157.733

================================================================================

Organism 1: 86.7976 118.731 96.6203 117.696 88.9969 117.189 Fitness: 190.593

Organism 2: 86.7976 118.731 98.8883 117.696 88.9969 117.189 Fitness: 215.593

Organism 3: 100.202 85.2976 99.4764 114.695 86.8282 115.897 Fitness: 137.593

Organism 4: 95.4121 97.3853 105.819 121.547 115.897 117.293 Fitness: 252.843

Organism 5: 100.202 110.069 105.819 98.5248 102.841 115.897 Fitness: 71.4249

Organism 6: 95.4121 113.699 108.283 115.897 88.2403 105.143 Fitness: 113.797

Organism 7: 95.4121 138.776 96.6203 114.695 124.573 105.143 Fitness: 162.592

Organism 8: 77.5396 110.069 112.949 114.946 80.5813 111.507 Fitness: 187.843

Organism 9: 95.4121 121.596 105.819 114.946 115.897 105.143 Fitness: 162.844

Organism 10: 95.4121 124.23 108.283 115.897 88.2403 105.143 Fitness: 113.797

Organism 11: 86.7976 118.731 110.498 117.696 88.9969 117.189 Fitness: 165.593

Organism 12: 86.7976 118.731 80.3427 117.696 88.9969 117.189 Fitness: 115.596

Organism 13: 100.202 110.069 101.76 114.946 86.8282 115.897 Fitness: 137.846

Organism 14: 95.4121 121.596 105.819 114.695 115.897 105.143 Fitness: 162.594

Organism 15: 100.202 113.699 108.283 98.5248 86.8282 115.897 Fitness: 146.424

Organism 16: 106.269 110.069 105.819 115.897 88.2403 105.143 Fitness: 138.797

Organism 17: 95.4121 110.069 96.6203 114.695 124.573 105.143 Fitness: 212.592

Organism 18: 77.5396 138.776 112.949 114.946 80.5813 111.507 Fitness: 162.846

Organism 19: 95.4121 113.699 108.283 114.946 115.897 105.143 Fitness: 157.845

Organism 20: 103.178 121.596 105.819 115.897 88.2403 105.143 Fitness: 88.7975

Best Fitness: 252.843

Avg Fitness: 155.392

================================================================================

Organism 1: 95.4121 114.949 105.819 121.547 115.897 117.293 Fitness: 194.63

Organism 2: 95.4121 138.776 96.6203 114.695 124.573 105.143 Fitness: 187.781

Organism 3: 77.5396 138.776 112.949 114.946 80.5813 111.507 Fitness: 213.029

Organism 4: 105.819 124.052 121.596 114.695 109.169 105.143 Fitness: 112.783

Organism 5: 107.764 97.3853 105.819 121.547 115.897 117.293 Fitness: 244.631

Organism 6: 95.4121 133.835 96.6203 114.695 124.573 105.143 Fitness: 162.78

Organism 7: 100.202 98.5248 86.8282 86.2687 115.897 113.699 Fitness: 146.609

Organism 8: 95.4121 110.069 96.6203 114.695 124.573 105.143 Fitness: 112.783

Organism 9: 106.269 110.069 105.819 115.897 105.525 105.143 Fitness: 163.981

Organism 10: 86.7976 99.9263 80.3427 117.696 88.9969 117.189 Fitness: 215.78

Organism 11: 95.4121 138.776 96.6203 114.695 115.897 117.293 Fitness: 172.78

Organism 12: 95.4121 97.3853 105.819 121.547 124.573 105.143 Fitness: 94.6341

Organism 13: 77.5396 138.776 112.949 114.695 80.5813 111.507 Fitness: 187.78

Organism 14: 95.4121 121.596 105.819 114.946 115.897 105.143 Fitness: 123.033

Organism 15: 95.4121 110.069 96.6203 121.547 115.897 117.293 Fitness: 119.634

Organism 16: 95.4121 81.8841 105.819 91.6123 124.573 105.143 Fitness: 197.779

Organism 17: 100.202 113.699 109.398 98.5248 86.8282 115.897 Fitness: 121.612

Organism 18: 113.479 110.069 108.283 114.695 124.573 105.143 Fitness: 222.779

Organism 19: 106.269 118.731 90.5336 115.897 88.2403 105.143 Fitness: 213.981

Organism 20: 86.7976 110.069 105.819 117.696 88.9969 117.189 Fitness: 90.7835

Best Fitness: 244.631

Avg Fitness: 163.34

================================================================================

Organism 1: 106.269 110.069 105.819 115.897 122.366 105.143 Fitness: 188.79

Organism 2: 95.4121 121.596 122.414 114.946 115.897 105.143 Fitness: 172.84

Organism 3: 105.819 124.052 121.596 114.695 105.143 109.169 Fitness: 137.591

Organism 4: 107.764 97.3853 105.819 121.547 115.897 117.293 Fitness: 194.44

Organism 5: 95.4121 105.819 97.3853 97.3853 107.992 105.143 Fitness: 194.439

Organism 6: 95.4121 138.776 96.6203 114.695 124.573 105.143 Fitness: 162.588

Organism 7: 95.4121 138.776 96.6203 114.695 115.897 117.293 Fitness: 262.588

Organism 8: 107.764 97.3853 105.819 121.547 115.897 117.293 Fitness: 119.443

Organism 9: 95.4121 116.843 96.6203 114.695 115.897 117.293 Fitness: 112.591

Organism 10: 86.7976 110.069 105.819 117.696 88.9969 117.189 Fitness: 165.588

Organism 11: 106.269 121.596 101.218 114.946 105.525 105.143 Fitness: 187.838

Organism 12: 93.4545 129.052 105.819 115.897 115.897 105.143 Fitness: 163.79

Organism 13: 123.437 124.052 105.819 121.547 113.247 105.143 Fitness: 179.441

Organism 14: 107.764 97.3853 121.596 114.695 115.897 117.293 Fitness: 162.588

Organism 15: 95.4121 138.776 96.6203 114.695 117.058 105.143 Fitness: 197.588

Organism 16: 95.4121 97.3853 105.819 121.547 124.573 105.143 Fitness: 144.44

Organism 17: 108.294 97.3853 96.6203 100.217 115.897 117.293 Fitness: 72.5913

Organism 18: 107.764 138.776 105.819 121.547 115.897 117.293 Fitness: 194.44

Organism 19: 95.4121 138.776 96.6203 114.695 88.9969 122.044 Fitness: 187.589

Organism 20: 86.7976 110.069 105.819 117.696 115.897 117.189 Fitness: 165.589

Best Fitness: 262.588

Avg Fitness: 164.979

================================================================================

Organism 1: 95.4121 138.776 96.6203 117.696 107.992 88.2685 Fitness: 125.285

Organism 2: 86.7976 110.069 122.248 114.695 88.9969 117.189 Fitness: 212.28

Organism 3: 86.7976 110.069 105.819 117.696 88.9969 117.189 Fitness: 115.284

Organism 4: 86.7976 110.069 99.8288 117.696 115.897 117.189 Fitness: 115.285

Organism 5: 95.4121 138.776 96.6203 114.695 117.058 105.143 Fitness: 162.282

Organism 6: 86.7976 110.069 105.819 134.554 88.9969 135.701 Fitness: 169.657

Organism 7: 107.764 97.3853 105.819 126.563 115.897 117.293 Fitness: 99.1515

Organism 8: 107.764 98.7368 121.596 114.695 115.897 117.293 Fitness: 112.284

Organism 9: 86.7976 107.034 105.819 117.696 93.5112 117.189 Fitness: 115.284

Organism 10: 123.437 119.44 105.819 121.547 132.059 105.143 Fitness: 204.133

Organism 11: 95.4121 111.637 122.118 114.695 88.9969 88.2685 Fitness: 212.281

Organism 12: 86.7976 138.776 96.6203 132.872 117.058 112.23 Fitness: 190.281

Organism 13: 86.7976 110.069 99.8288 117.696 88.9969 117.189 Fitness: 75.2852

Organism 14: 86.7976 110.069 105.819 117.696 115.897 97.2111 Fitness: 190.284

Organism 15: 95.4121 138.776 96.6203 114.695 88.9969 105.143 Fitness: 97.2843

Organism 16: 86.7976 110.069 105.819 127.574 117.058 117.189 Fitness: 309.657

Organism 17: 107.764 97.3853 121.596 114.695 108.299 117.293 Fitness: 137.284

Organism 18: 118.699 98.7368 105.819 126.563 115.897 117.293 Fitness: 149.148

Organism 19: 86.7976 110.069 105.819 117.696 88.9969 117.189 Fitness: 190.281

Organism 20: 123.437 124.052 105.819 121.547 113.247 105.143 Fitness: 119.136

Best Fitness: 309.657

Avg Fitness: 158.092

================================================================================

Organism 1: 86.7976 110.069 105.819 134.554 88.9969 135.701 Fitness: 206.738

Organism 2: 135.097 119.44 105.819 121.547 132.059 105.143 Fitness: 178.734

Organism 3: 86.7976 125.371 105.819 127.574 117.058 106.587 Fitness: 124.76

Organism 4: 95.4121 138.776 96.6203 98.634 116.997 88.074 Fitness: 124.884

Organism 5: 95.4121 111.637 122.118 114.695 88.9969 88.2685 Fitness: 186.88

Organism 6: 62.658 119.06 105.819 117.696 93.5112 117.189 Fitness: 184.884

Organism 7: 86.7976 124.185 105.819 134.554 88.9969 133.973 Fitness: 131.74

Organism 8: 104.59 110.069 105.819 127.574 117.058 134.311 Fitness: 184.759

Organism 9: 86.7976 110.069 105.819 117.696 88.9969 117.189 Fitness: 149.884

Organism 10: 107.764 92.6095 121.125 109.212 115.897 95.4948 Fitness: 158.749

Organism 11: 86.7976 106.115 105.819 121.547 88.9969 135.701 Fitness: 93.735

Organism 12: 111.282 119.44 105.819 134.554 132.059 93.1491 Fitness: 206.738

Organism 13: 86.7976 95.5333 105.819 117.696 107.992 117.189 Fitness: 114.883

Organism 14: 95.4121 138.776 96.6203 127.574 117.058 88.2685 Fitness: 174.758

Organism 15: 95.4121 107.034 105.819 117.696 93.5112 88.2685 Fitness: 174.884

Organism 16: 86.7976 111.637 118.739 114.695 88.9969 106.553 Fitness: 71.8828

Organism 17: 86.7976 110.069 105.819 127.574 140.606 135.701 Fitness: 134.76

Organism 18: 86.7976 110.069 105.819 134.554 88.9969 110.608 Fitness: 313.738

Organism 19: 86.7976 110.069 105.819 126.563 88.9969 117.189 Fitness: 123.751

Organism 20: 107.764 97.3853 105.819 117.696 115.897 117.293 Fitness: 114.883

Best Fitness: 313.738

Avg Fitness: 163.951

================================================================================

Organism 1: 86.7976 110.069 105.819 127.574 140.606 135.701 Fitness: 175.027

Organism 2: 111.282 119.44 105.819 134.554 153.566 93.1491 Fitness: 207.006

Organism 3: 85.126 107.034 105.819 117.696 110.068 88.2685 Fitness: 165.148

Organism 4: 130.219 119.44 105.819 134.554 132.115 93.1491 Fitness: 332.006

Organism 5: 86.7976 111.637 118.739 114.695 88.9969 101.385 Fitness: 112.151

Organism 6: 111.282 119.44 111.314 134.554 132.059 93.1491 Fitness: 207.006

Organism 7: 95.4121 122.799 96.6203 127.574 117.058 88.2685 Fitness: 110.029

Organism 8: 104.59 110.069 105.005 127.574 117.058 134.311 Fitness: 250.026

Organism 9: 86.7976 121.462 105.819 127.574 117.058 106.587 Fitness: 150.028

Organism 10: 95.4121 119.578 96.6203 113.471 128.218 88.2685 Fitness: 225.025

Organism 11: 86.7976 110.069 105.819 134.554 140.606 135.701 Fitness: 157.007

Organism 12: 111.282 119.44 105.819 127.574 132.059 93.1491 Fitness: 150.026

Organism 13: 95.4121 107.034 105.819 117.696 93.5112 88.2685 Fitness: 215.148

Organism 14: 125.218 119.44 87.1632 134.554 132.059 93.1491 Fitness: 132.008

Organism 15: 86.7976 105.964 118.739 114.695 88.9969 106.553 Fitness: 137.149

Organism 16: 111.282 111.637 105.819 134.554 132.059 93.1491 Fitness: 182.006

Organism 17: 95.4121 138.776 105.819 123.903 117.058 89.6248 Fitness: 175.026

Organism 18: 104.59 110.069 96.6203 127.574 104.546 134.311 Fitness: 150.026

Organism 19: 86.7976 138.776 96.6203 127.574 117.058 106.587 Fitness: 175.029

Organism 20: 95.4121 125.371 105.819 127.574 117.058 88.2685 Fitness: 120.025

Best Fitness: 332.006

Avg Fitness: 164.345

================================================================================

Organism 1: 86.7976 105.964 118.739 114.695 88.9969 106.553 Fitness: 237.251

Organism 2: 133.231 111.637 105.819 134.554 132.059 93.1491 Fitness: 142.111

Organism 3: 86.7976 138.776 96.6203 127.574 117.058 106.587 Fitness: 150.131

Organism 4: 95.4121 107.034 105.819 117.696 93.5112 88.2685 Fitness: 100.252

Organism 5: 86.7976 121.462 105.819 152.262 117.058 106.587 Fitness: 135.131

Organism 6: 111.282 119.44 105.819 131.598 153.566 93.1491 Fitness: 157.113

Organism 7: 93.3342 88.2685 102.038 127.574 122.799 95.4121 Fitness: 120.132

Organism 8: 95.4121 119.578 96.6203 113.471 113.419 88.2685 Fitness: 96.0289

Organism 9: 111.282 119.44 101.141 134.554 132.059 93.1491 Fitness: 242.11

Organism 10: 104.59 110.069 120.528 127.574 104.546 134.311 Fitness: 150.129

Organism 11: 86.7976 126.475 105.819 114.695 86.0046 106.553 Fitness: 162.25

Organism 12: 111.282 105.964 118.739 134.554 132.059 93.1491 Fitness: 232.109

Organism 13: 76.2413 107.034 105.819 140.256 93.5112 106.587 Fitness: 140.255

Organism 14: 95.4121 138.776 96.6203 127.574 117.058 88.2685 Fitness: 235.129

Organism 15: 86.7976 121.462 105.819 127.574 117.058 106.587 Fitness: 150.132

Organism 16: 111.282 119.44 105.819 134.554 153.566 93.1491 Fitness: 207.109

Organism 17: 78.9977 138.548 96.6203 113.471 128.218 88.2685 Fitness: 111.03

Organism 18: 95.4121 122.799 96.6203 127.574 117.058 88.2685 Fitness: 200.129

Organism 19: 93.6308 119.44 111.314 127.574 104.546 93.1491 Fitness: 200.129

Organism 20: 104.59 110.069 108.561 134.554 132.059 134.311 Fitness: 311.109

Best Fitness: 311.109

Avg Fitness: 173.789

================================================================================

Organism 1: 95.4121 119.578 96.6203 116.787 113.419 88.2685 Fitness: 136.041

Organism 2: 93.6308 119.44 111.314 127.574 108.415 93.1491 Fitness: 150.14

Organism 3: 76.2413 107.034 83.8201 137.406 98.4254 106.587 Fitness: 162.822

Organism 4: 95.4121 138.776 96.6203 127.574 117.058 88.2685 Fitness: 225.14

Organism 5: 104.59 110.069 120.528 127.574 104.546 134.311 Fitness: 325.14

Organism 6: 133.231 111.637 105.819 127.556 132.059 93.1491 Fitness: 227.121

Organism 7: 101.469 122.799 96.6203 127.574 117.058 88.2685 Fitness: 120.143

Organism 8: 86.7976 138.776 96.6203 127.574 117.058 106.804 Fitness: 250.142

Organism 9: 93.6308 109.125 111.314 127.574 104.546 93.1491 Fitness: 295.14

Organism 10: 95.4121 107.034 98.7484 135.441 93.5112 88.2685 Fitness: 175.264

Organism 11: 95.4121 119.578 111.314 113.471 113.419 88.2685 Fitness: 286.037

Organism 12: 106.587 127.574 117.058 104.546 105.819 93.1491 Fitness: 175.142

Organism 13: 76.2413 107.034 96.6203 140.256 93.5112 106.587 Fitness: 162.822

Organism 14: 95.4121 138.776 105.819 127.574 117.058 88.2685 Fitness: 175.14

Organism 15: 104.59 111.637 128.154 115.122 132.059 134.311 Fitness: 232.12

Organism 16: 133.231 112.283 120.528 127.574 104.546 93.1491 Fitness: 150.143

Organism 17: 95.4121 122.799 96.6203 127.574 117.058 88.2685 Fitness: 200.14

Organism 18: 86.7976 138.776 96.6203 111.637 127.574 134.311 Fitness: 200.14

Organism 19: 93.6308 119.44 111.314 117.696 104.546 93.1491 Fitness: 190.265

Organism 20: 95.4121 117.058 88.2685 105.819 95.2173 127.574 Fitness: 125.144

Best Fitness: 325.14

Avg Fitness: 175.209

================================================================================

Organism 1: 95.4121 138.776 105.819 127.574 117.058 88.2685 Fitness: 135.551

Organism 2: 113.222 111.637 128.154 115.122 132.059 134.311 Fitness: 163.095

Organism 3: 133.231 112.283 120.528 127.574 104.546 93.1491 Fitness: 175.547

Organism 4: 97.011 107.034 96.6203 140.256 93.5112 106.587 Fitness: 163.229

Organism 5: 85.0827 138.776 105.819 119.081 117.058 88.2685 Fitness: 185.547

Organism 6: 93.6308 109.125 111.314 127.574 104.546 93.1491 Fitness: 150.548

Organism 7: 106.587 127.574 117.058 104.546 105.819 93.1491 Fitness: 172.52

Organism 8: 95.4121 138.776 105.819 127.574 117.058 90.1459 Fitness: 225.547

Organism 9: 94.8356 138.776 105.819 127.574 108.222 88.2685 Fitness: 175.548

Organism 10: 93.6308 119.44 111.314 127.574 106.035 93.1491 Fitness: 150.547

Organism 11: 91.7386 138.776 105.819 115.122 132.059 88.2685 Fitness: 163.095

Organism 12: 104.59 111.637 128.154 126.309 117.058 134.311 Fitness: 145.547

Organism 13: 133.231 112.283 120.528 140.256 93.5112 93.1491 Fitness: 163.23

Organism 14: 76.2413 107.034 96.6203 127.574 104.546 106.587 Fitness: 150.549

Organism 15: 95.4121 138.776 111.314 127.574 104.546 88.2685 Fitness: 200.547

Organism 16: 93.6308 109.125 105.819 127.574 117.058 89.683 Fitness: 175.547

Organism 17: 106.587 127.574 127.567 98.3286 117.058 93.1491 Fitness: 152.519

Organism 18: 88.6865 138.776 105.819 138.416 105.819 88.2685 Fitness: 135.548

Organism 19: 95.4121 136.739 111.314 127.574 117.058 88.2685 Fitness: 319.547

Organism 20: 104.946 138.776 105.819 127.574 108.415 93.1491 Fitness: 175.547

Best Fitness: 319.547

Avg Fitness: 173.018

================================================================================

Organism 1: 106.587 127.574 127.567 98.3286 92.1813 93.1491 Fitness: 120.792

Organism 2: 88.6865 138.776 105.819 140.799 105.819 88.2685 Fitness: 155.875

Organism 3: 91.7386 138.776 105.819 115.122 132.059 88.2685 Fitness: 107.585

Organism 4: 97.011 101.847 96.6203 140.256 111.659 106.587 Fitness: 172.716

Organism 5: 94.8356 138.776 105.819 127.574 108.222 88.2685 Fitness: 150.034

Organism 6: 133.231 112.283 120.528 127.574 93.8632 93.1491 Fitness: 225.033

Organism 7: 93.6308 109.125 108.68 127.574 117.058 70.605 Fitness: 175.035

Organism 8: 104.59 111.637 141.505 126.309 117.058 134.311 Fitness: 173.768

Organism 9: 106.587 127.574 127.567 98.3286 117.058 93.1491 Fitness: 245.789

Organism 10: 106.587 127.574 93.6308 107.034 117.058 93.1491 Fitness: 324.109

Organism 11: 123.211 138.776 105.819 138.416 105.819 93.1491 Fitness: 145.876

Organism 12: 88.6865 127.574 127.567 98.3286 117.058 88.2685 Fitness: 170.788

Organism 13: 114.893 107.034 76.7319 115.122 132.059 107.018 Fitness: 237.582

Organism 14: 97.011 138.776 105.819 140.368 93.5112 106.587 Fitness: 222.716

Organism 15: 94.8356 112.283 120.528 127.574 108.222 88.2685 Fitness: 175.035

Organism 16: 133.231 138.776 105.819 127.574 104.546 95.0179 Fitness: 150.034

Organism 17: 93.6308 109.125 122.812 126.309 117.058 89.683 Fitness: 173.769

Organism 18: 104.59 111.637 139.059 141.258 117.058 134.311 Fitness: 175.035

Organism 19: 122.012 127.574 127.567 92.6845 117.058 93.1491 Fitness: 95.7918

Organism 20: 106.587 127.574 127.567 98.3286 117.058 93.1491 Fitness: 170.789

Best Fitness: 324.109

Avg Fitness: 171.242

================================================================================

Organism 1: 122.012 127.574 127.567 92.6845 117.058 93.1491 Fitness: 166.438

Organism 2: 93.6308 109.125 108.68 127.574 117.058 70.605 Fitness: 201.326

Organism 3: 105.819 115.122 138.776 91.7386 132.059 88.2685 Fitness: 313.875

Organism 4: 106.587 127.574 127.567 98.3286 117.058 85.6845 Fitness: 122.084

Organism 5: 104.59 111.637 139.059 141.258 117.058 134.311 Fitness: 240.01

Organism 6: 97.011 138.776 105.819 140.368 93.5112 105.544 Fitness: 174.12

Organism 7: 91.7386 138.776 105.819 115.122 132.059 88.2685 Fitness: 138.878

Organism 8: 94.8356 138.776 105.819 127.574 108.222 86.9175 Fitness: 211.326

Organism 9: 114.893 85.2662 76.7319 111.424 132.059 107.018 Fitness: 138.877

Organism 10: 97.011 138.776 105.819 140.368 98.5742 106.587 Fitness: 59.1226

Organism 11: 122.012 127.574 108.68 127.574 117.058 102.309 Fitness: 136.329

Organism 12: 76.6256 109.125 111.916 92.6845 117.058 70.605 Fitness: 151.437

Organism 13: 91.7386 138.776 119.375 115.122 117.058 88.2685 Fitness: 263.874

Organism 14: 106.587 127.574 127.567 98.3286 125.749 97.8815 Fitness: 357

Organism 15: 104.59 111.637 115.873 140.368 93.5112 134.311 Fitness: 139.121

Organism 16: 97.011 138.776 153.324 141.258 117.058 106.587 Fitness: 165.01

Organism 17: 83.8892 138.776 105.819 108.222 127.574 88.2685 Fitness: 101.33

Organism 18: 94.8356 138.776 105.819 115.122 132.059 87.6964 Fitness: 138.877

Organism 19: 114.893 107.034 105.819 140.368 132.059 107.018 Fitness: 239.12

Organism 20: 97.011 138.776 76.7319 115.122 93.5112 106.587 Fitness: 163.874

Best Fitness: 357

Avg Fitness: 177.129

================================================================================

## Appendix 2

Population Size: 20

Amount of Food: 15

Amount of Poison: 20

Generation Time in Seconds: 25

Population Survivors (% of population): 50

Selection Type (Roulette = 0) (Random = 1) (Tournament = 2): 0

Crossover Type (Uniform = 0) (Single-Point = 1) (Multi-Point = 2): 0

Primary Mutation Type (Creep = 0) (Random Resetting = 1): 0

Creep Value Range: 25

Secondary Mutation Type (None = 0) (Swap = 1) (Scramble = 2) (Inversion = 3): 1

Mutation Chance Percentage (0.0f - 100.0f): 15

================================================================================

Organism 1: 117.721 104.548 106.678 119.182 83.0233 104.995 Fitness: 144.746

Organism 2: 115.644 85.7378 111.803 118.239 90.4426 106.099 Fitness: 118.805

Organism 3: 117.721 104.548 130.323 119.182 83.0233 108.958 Fitness: 169.747

Organism 4: 107.865 99.8599 96.7555 88.6912 116.665 86.0808 Fitness: 64.2603

Organism 5: 117.721 104.548 113.976 119.182 88.2718 104.995 Fitness: 119.75

Organism 6: 68.9261 105.399 109.614 124.901 103.285 114.541 Fitness: 157.068

Organism 7: 87.2434 97.8027 97.0218 114.515 88.277 115.517 Fitness: 90.0841

Organism 8: 117.721 104.548 113.976 119.182 83.0233 104.995 Fitness: 94.7508

Organism 9: 91.4287 97.1735 95.3994 80.1547 115.945 97.6623 Fitness: 80.7239

Organism 10: 113.326 114.691 81.8269 93.4693 102.136 111.657 Fitness: 69.0384

Organism 11: 115.644 85.7378 113.976 119.182 90.4426 104.995 Fitness: 119.75

Organism 12: 117.721 104.548 111.803 118.239 83.0233 110.479 Fitness: 68.8083

Organism 13: 117.721 104.548 90.8834 119.182 116.665 86.0808 Fitness: 94.7508

Organism 14: 96.7555 107.865 99.8599 88.6912 83.0233 122.369 Fitness: 89.2603

Organism 15: 84.3393 105.399 103.285 106.501 104.995 109.614 Fitness: 107.07

Organism 16: 83.0233 104.548 119.182 113.976 117.721 114.541 Fitness: 119.75

Organism 17: 107.079 97.8027 94.0144 114.515 88.277 86.1145 Fitness: 65.0843

Organism 18: 87.2434 104.548 95.4995 138.319 83.0233 131.203 Fitness: 94.7508

Organism 19: 91.4287 120.303 81.8269 94.2883 115.945 111.657 Fitness: 105.724

Organism 20: 97.2962 96.6421 95.3994 75.3315 102.136 97.6623 Fitness: 94.0384

Best Fitness: 169.747

Avg Fitness: 103.398

================================================================================

Organism 1: 107.865 99.8599 96.7555 88.6912 116.665 86.0808 Fitness: 89.3629

Organism 2: 117.721 83.5046 93.2121 119.182 88.2718 104.995 Fitness: 69.8533

Organism 3: 113.249 104.548 130.323 119.182 83.0233 108.958 Fitness: 69.8533

Organism 4: 91.4287 120.303 81.8269 94.2883 115.945 111.657 Fitness: 129.958

Organism 5: 83.0233 104.548 119.182 113.976 117.721 114.541 Fitness: 64.6472

Organism 6: 115.644 85.7378 111.803 118.239 90.4426 106.099 Fitness: 118.911

Organism 7: 119.182 104.548 113.976 117.721 83.0233 104.995 Fitness: 119.853

Organism 8: 115.644 85.7378 111.803 118.239 90.4426 106.099 Fitness: 68.9109

Organism 9: 59.074 105.399 109.614 124.901 114.928 114.541 Fitness: 100.572

Organism 10: 117.721 104.548 113.976 119.182 88.2718 104.995 Fitness: 94.8533

Organism 11: 117.721 99.8599 113.976 88.6912 88.2718 86.0808 Fitness: 179.358

Organism 12: 107.865 116.68 96.7555 97.1743 128.387 104.995 Fitness: 119.85

Organism 13: 91.4287 120.303 81.8269 119.182 114.096 111.657 Fitness: 144.85

Organism 14: 117.721 104.548 130.323 94.2883 97.0158 108.958 Fitness: 94.96

Organism 15: 83.0233 85.7378 91.9845 113.976 90.4426 106.099 Fitness: 139.646

Organism 16: 115.644 104.548 119.182 118.239 129.892 114.541 Fitness: 93.9109

Organism 17: 115.644 98.6785 111.803 126.537 83.0233 106.099 Fitness: 144.848

Organism 18: 117.721 85.7378 113.976 118.239 90.4426 104.995 Fitness: 93.9109

Organism 19: 141.37 104.548 90.9385 119.182 88.2718 114.541 Fitness: 169.851

Organism 20: 68.9261 105.399 123.231 124.901 103.285 104.995 Fitness: 125.572

Best Fitness: 179.358

Avg Fitness: 105.177

================================================================================

Organism 1: 68.9261 105.399 123.231 124.901 103.285 104.995 Fitness: 120.571

Organism 2: 107.865 116.68 96.7555 97.1743 128.387 104.995 Fitness: 197.841

Organism 3: 83.0233 85.7378 91.9845 115.07 90.4426 106.099 Fitness: 89.647

Organism 4: 115.355 99.8599 113.976 88.6912 77.124 86.0808 Fitness: 89.3627

Organism 5: 91.4287 120.303 81.8269 119.182 114.096 99.4053 Fitness: 69.8531

Organism 6: 107.865 116.68 96.7555 97.1743 128.387 104.995 Fitness: 122.846

Organism 7: 117.721 85.7378 113.976 118.239 90.4426 104.995 Fitness: 93.9107

Organism 8: 127.82 98.6785 111.803 126.537 86.6072 106.099 Fitness: 77.2088

Organism 9: 119.182 104.548 113.976 117.721 67.7896 104.995 Fitness: 118.392

Organism 10: 113.249 104.548 130.323 119.182 75.1027 89.3556 Fitness: 54.8531

Organism 11: 68.9261 120.747 123.231 124.901 103.285 104.995 Fitness: 100.572

Organism 12: 107.631 105.399 96.7555 108.762 128.387 104.995 Fitness: 97.8458

Organism 13: 83.0233 85.9042 91.9845 113.976 90.4426 86.0808 Fitness: 114.645

Organism 14: 112.738 99.8599 113.976 88.6912 88.2718 96.0662 Fitness: 39.3623

Organism 15: 91.4287 116.68 81.8269 119.182 114.096 104.995 Fitness: 154.848

Organism 16: 107.865 120.303 96.7555 97.1743 128.387 111.657 Fitness: 122.846

Organism 17: 115.644 126.537 111.803 90.4426 85.7378 104.995 Fitness: 152.206

Organism 18: 117.721 111.746 119.494 118.239 83.0233 106.099 Fitness: 143.907

Organism 19: 119.182 104.548 113.976 117.721 83.0233 104.995 Fitness: 93.3929

Organism 20: 113.249 104.548 130.323 119.182 83.0233 108.958 Fitness: 94.8523

Best Fitness: 197.841

Avg Fitness: 107.448

================================================================================

Organism 1: 107.865 101.628 77.1982 83.8101 128.387 104.995 Fitness: 147.872

Organism 2: 91.4287 116.68 82.9546 119.182 114.096 104.995 Fitness: 169.878

Organism 3: 68.9261 105.399 123.231 124.901 103.285 104.995 Fitness: 75.5988

Organism 4: 127.82 98.6785 111.803 133.563 86.6072 106.099 Fitness: 152.233

Organism 5: 107.865 104.897 104.995 86.7084 133.09 97.1743 Fitness: 97.872

Organism 6: 115.644 126.537 127.01 90.4426 85.7378 118.61 Fitness: 191.136

Organism 7: 107.631 105.399 96.7555 108.762 114.118 104.995 Fitness: 104.459

Organism 8: 69.5393 105.399 123.231 124.901 107.158 103.89 Fitness: 100.599

Organism 9: 107.865 122.49 96.7555 97.1743 128.387 116.195 Fitness: 122.872

Organism 10: 112.738 99.8599 113.976 88.6912 88.2718 96.0662 Fitness: 139.389

Organism 11: 107.865 116.68 107.593 119.182 114.096 104.995 Fitness: 194.874

Organism 12: 91.4287 116.68 79.6818 97.1743 128.387 104.995 Fitness: 122.872

Organism 13: 68.9261 105.399 111.803 106.099 111.174 149.634 Fitness: 177.234

Organism 14: 127.82 76.1824 123.231 124.901 103.285 104.995 Fitness: 200.594

Organism 15: 115.644 116.68 96.7555 97.1743 85.7378 104.995 Fitness: 47.8718

Organism 16: 107.865 126.537 111.803 88.9986 128.387 104.995 Fitness: 91.1402

Organism 17: 68.9261 105.399 147.148 124.901 115.61 104.995 Fitness: 100.599

Organism 18: 107.631 105.399 96.7555 108.762 91.5116 104.995 Fitness: 84.4594

Organism 19: 107.865 99.8599 113.976 88.6912 88.2718 104.995 Fitness: 89.3888

Organism 20: 112.738 120.602 96.7555 97.1743 128.387 96.0662 Fitness: 122.872

Best Fitness: 200.594

Avg Fitness: 116.691

================================================================================

Organism 1: 68.9261 129.775 147.148 124.901 115.61 104.995 Fitness: 125.492

Organism 2: 127.82 98.6785 111.803 133.563 86.6072 106.099 Fitness: 109.155

Organism 3: 106.099 105.399 145.896 68.9261 111.174 111.803 Fitness: 141.691

Organism 4: 91.4287 116.68 69.1195 119.182 114.096 104.995 Fitness: 94.7725

Organism 5: 68.9261 105.399 147.148 124.901 115.61 104.995 Fitness: 175.488

Organism 6: 107.631 105.399 96.7555 108.762 114.118 104.995 Fitness: 119.354

Organism 7: 104.995 51.5026 123.231 124.901 103.285 127.82 Fitness: 100.492

Organism 8: 115.644 116.68 96.7555 97.1743 85.7378 104.995 Fitness: 72.7663

Organism 9: 107.631 103.654 96.7555 108.762 91.5116 104.995 Fitness: 84.354

Organism 10: 68.9261 105.399 147.148 124.901 115.61 104.995 Fitness: 125.493

Organism 11: 68.9261 105.399 111.803 124.901 86.6072 106.099 Fitness: 150.489

Organism 12: 115.61 98.6785 147.148 133.563 127.82 104.995 Fitness: 119.151

Organism 13: 91.4287 105.399 82.9546 106.099 111.174 149.634 Fitness: 200.17

Organism 14: 68.9261 116.68 111.803 119.182 114.096 94.5617 Fitness: 169.771

Organism 15: 107.631 96.9081 147.148 104.995 114.118 124.901 Fitness: 110.492

Organism 16: 68.9261 105.399 96.7555 108.762 115.61 104.995 Fitness: 44.3533

Organism 17: 115.644 116.68 96.7555 97.1743 85.7378 104.995 Fitness: 172.765

Organism 18: 127.82 76.1824 123.231 124.901 103.285 104.995 Fitness: 150.491

Organism 19: 68.9261 105.399 147.148 108.762 115.61 104.995 Fitness: 134.353

Organism 20: 107.631 105.399 79.9829 124.901 91.5116 104.995 Fitness: 125.492

Best Fitness: 200.17

Avg Fitness: 118.482

================================================================================

Organism 1: 106.099 105.399 145.896 68.9261 111.174 111.803 Fitness: 9.71629

Organism 2: 68.9261 105.399 147.148 124.901 115.61 104.995 Fitness: 175.566

Organism 3: 115.644 125.329 96.7555 97.1743 85.7378 104.995 Fitness: 47.8446

Organism 4: 78.7476 115.63 147.148 124.901 115.61 104.995 Fitness: 85.5715

Organism 5: 107.631 105.399 96.7555 123.15 114.118 104.995 Fitness: 84.4323

Organism 6: 84.9611 88.2726 82.9546 106.099 111.174 149.634 Fitness: 81.7694

Organism 7: 68.9261 116.68 88.2667 119.182 114.096 94.5617 Fitness: 219.849

Organism 8: 82.9546 105.399 91.4287 106.099 111.174 149.634 Fitness: 106.769

Organism 9: 65.5847 105.399 147.148 108.762 115.61 104.995 Fitness: 109.432

Organism 10: 68.9261 105.399 92.2663 124.901 86.6072 106.099 Fitness: 75.5716

Organism 11: 68.9261 104.995 131.972 85.7378 124.901 105.399 Fitness: 100.57

Organism 12: 115.644 116.68 96.7555 97.1743 96.2927 104.995 Fitness: 172.844

Organism 13: 68.9261 105.399 147.148 108.762 115.61 104.995 Fitness: 69.4324

Organism 14: 107.631 105.399 96.7555 124.901 114.118 104.995 Fitness: 125.571

Organism 15: 91.4287 116.68 82.9546 119.182 114.096 94.5617 Fitness: 219.849

Organism 16: 68.9261 105.399 111.803 106.099 111.174 149.634 Fitness: 181.765

Organism 17: 91.4287 88.0003 145.896 106.099 111.174 149.634 Fitness: 81.7694

Organism 18: 106.099 105.399 82.9546 68.9261 111.174 111.803 Fitness: 44.5961

Organism 19: 68.9261 105.399 147.148 124.901 123.034 104.995 Fitness: 75.5716

Organism 20: 68.9261 105.399 111.803 108.762 86.6072 106.099 Fitness: 84.4324

Best Fitness: 219.849

Avg Fitness: 106.396

================================================================================

Organism 1: 105.399 82.9546 149.634 106.099 111.174 91.4287 Fitness: 81.7794

Organism 2: 78.7476 115.63 147.148 124.901 137.139 104.995 Fitness: 75.5816

Organism 3: 96.2927 97.1743 96.7555 116.68 115.644 104.995 Fitness: 122.855

Organism 4: 91.4287 116.68 82.9546 119.182 114.096 106.895 Fitness: 119.859

Organism 5: 91.4287 116.68 82.9546 119.182 107.585 94.5617 Fitness: 79.8613

Organism 6: 91.4287 116.68 90.3636 119.182 114.096 94.5617 Fitness: 104.862

Organism 7: 115.644 116.68 96.7555 97.1743 96.2927 104.995 Fitness: 97.8547

Organism 8: 68.9261 105.399 111.803 125.654 111.174 149.634 Fitness: 106.778

Organism 9: 115.644 125.329 96.7555 97.1743 85.7378 104.995 Fitness: 72.8547

Organism 10: 78.7476 104.525 147.148 124.901 115.61 104.995 Fitness: 205.577

Organism 11: 78.7476 105.399 147.148 124.901 111.174 169.913 Fitness: 125.58

Organism 12: 82.9546 91.5757 91.4287 106.099 115.61 104.995 Fitness: 106.778

Organism 13: 132.268 115.644 82.9546 116.68 96.2927 94.5617 Fitness: 144.858

Organism 14: 91.4287 116.68 96.7555 107.914 114.096 108.364 Fitness: 82.8547

Organism 15: 91.4287 116.68 82.9546 119.182 127.118 94.5617 Fitness: 144.858

Organism 16: 91.4287 116.68 82.9546 119.182 114.096 94.5617 Fitness: 119.862

Organism 17: 68.9261 126.342 111.803 97.1743 111.174 149.634 Fitness: 122.855

Organism 18: 115.644 105.399 96.7555 106.099 96.2927 104.995 Fitness: 106.779

Organism 19: 112.491 115.63 147.148 118.621 85.7378 104.995 Fitness: 47.8545

Organism 20: 78.7476 125.329 84.9161 118.573 115.61 104.995 Fitness: 125.58

Best Fitness: 205.577

Avg Fitness: 110.786

================================================================================

Organism 1: 91.4287 116.68 82.9546 119.182 107.585 94.5617 Fitness: 69.8572

Organism 2: 91.4287 116.68 82.9546 119.182 94.611 94.5617 Fitness: 94.8561

Organism 3: 91.4287 116.68 96.7555 107.914 114.096 108.364 Fitness: 133.59

Organism 4: 78.7476 104.525 147.148 124.901 117.241 104.995 Fitness: 75.5768

Organism 5: 78.7476 104.525 147.148 124.901 115.61 104.995 Fitness: 100.577

Organism 6: 115.644 116.68 96.7555 97.1743 87.6332 104.995 Fitness: 122.85

Organism 7: 115.644 105.399 96.7555 90.616 96.2927 104.995 Fitness: 156.774

Organism 8: 82.9546 91.5757 91.4287 106.099 113.634 104.995 Fitness: 81.7746

Organism 9: 78.7476 105.399 147.148 135.934 111.174 169.913 Fitness: 125.577

Organism 10: 78.7476 104.525 147.148 124.901 115.61 98.6151 Fitness: 200.572

Organism 11: 91.4287 140.329 82.9546 119.182 107.585 94.5617 Fitness: 79.857

Organism 12: 91.4287 116.68 82.9546 119.182 107.585 94.5617 Fitness: 119.856

Organism 13: 78.7476 100.688 147.148 124.901 115.61 104.995 Fitness: 150.575

Organism 14: 85.7125 130.239 90.6403 107.914 114.096 108.364 Fitness: 58.5896

Organism 15: 78.7476 116.68 147.148 124.901 115.61 104.995 Fitness: 125.577

Organism 16: 115.644 91.1748 104.525 104.995 96.2927 97.1743 Fitness: 97.8499

Organism 17: 130.199 105.399 96.7555 106.099 115.61 104.995 Fitness: 206.773

Organism 18: 82.9546 91.5757 91.4287 108.278 96.2927 104.995 Fitness: 81.7745

Organism 19: 89.9866 104.525 124.901 169.913 111.174 147.148 Fitness: 125.576

Organism 20: 78.7476 105.399 146.3 124.901 115.61 104.995 Fitness: 150.576

Best Fitness: 206.773

Avg Fitness: 117.95

================================================================================

Organism 1: 82.9546 104.995 91.5757 106.099 113.634 91.4287 Fitness: 131.675

Organism 2: 78.281 91.5757 91.4287 106.099 113.634 104.995 Fitness: 131.674

Organism 3: 78.7476 100.688 147.148 124.901 115.61 104.995 Fitness: 85.4752

Organism 4: 101.417 116.68 104.995 97.1743 87.6332 96.7555 Fitness: 72.75

Organism 5: 78.7476 105.399 146.3 124.901 115.61 104.995 Fitness: 175.472

Organism 6: 91.4287 116.68 82.9546 119.182 107.585 94.5617 Fitness: 144.756

Organism 7: 78.7476 100.094 116.68 147.148 115.61 104.995 Fitness: 160.472

Organism 8: 78.7476 104.995 162.947 135.521 115.61 100.688 Fitness: 85.4768

Organism 9: 78.7476 100.688 147.148 124.901 115.61 104.995 Fitness: 100.476

Organism 10: 85.5683 78.6347 147.148 124.901 115.61 104.995 Fitness: 125.475

Organism 11: 82.9546 91.5757 91.4287 106.099 113.634 123.658 Fitness: 212.15

Organism 12: 82.9546 91.5757 91.4287 106.099 113.634 104.995 Fitness: 56.6746

Organism 13: 115.644 100.688 160.598 97.1743 115.61 112.398 Fitness: 122.747

Organism 14: 96.7555 114.014 78.7476 85.9454 124.901 104.995 Fitness: 100.477

Organism 15: 92.2269 116.68 146.3 124.901 115.61 104.995 Fitness: 175.471

Organism 16: 91.4287 105.399 82.9546 119.182 107.585 94.5617 Fitness: 154.755

Organism 17: 147.148 100.688 124.901 78.7476 115.61 104.995 Fitness: 125.477

Organism 18: 78.7476 116.68 147.148 124.901 115.61 104.995 Fitness: 150.476

Organism 19: 78.7476 100.688 147.148 124.901 126.672 104.995 Fitness: 125.475

Organism 20: 78.7476 100.688 147.148 124.901 115.61 104.995 Fitness: 85.4749

Best Fitness: 212.15

Avg Fitness: 130.619

================================================================================

Organism 1: 82.9546 91.5757 91.4287 106.099 113.634 140.768 Fitness: 22.0146

Organism 2: 78.281 91.5757 91.4287 106.099 113.634 104.995 Fitness: 81.1974

Organism 3: 71.9846 78.6347 147.148 124.901 115.61 104.995 Fitness: 124.999

Organism 4: 113.634 129.24 91.5757 104.995 82.9546 91.4287 Fitness: 131.196

Organism 5: 78.7476 104.995 147.148 100.688 115.61 136.121 Fitness: 149.999

Organism 6: 78.7476 100.688 104.995 124.901 126.672 147.148 Fitness: 99.9981

Organism 7: 101.417 116.68 126.853 97.1743 87.6332 96.7555 Fitness: 72.2726

Organism 8: 78.7476 116.68 147.148 107.97 115.61 104.995 Fitness: 99.9996

Organism 9: 115.644 100.688 160.598 97.9604 115.61 112.398 Fitness: 122.272

Organism 10: 78.281 91.5757 77.8385 106.099 113.634 104.995 Fitness: 81.1974

Organism 11: 89.2728 91.5757 91.4287 106.099 113.634 123.658 Fitness: 81.1974

Organism 12: 82.9546 73.9282 91.4287 106.099 113.634 104.995 Fitness: 56.1974

Organism 13: 104.995 78.6347 113.634 105.801 91.5757 85.5683 Fitness: 225.187

Organism 14: 82.9546 104.995 147.148 124.901 115.61 91.4287 Fitness: 99.9988

Organism 15: 102.701 75.9892 147.148 124.901 125.971 104.995 Fitness: 194.996

Organism 16: 126.672 78.7476 147.148 124.901 100.688 104.995 Fitness: 124.996

Organism 17: 96.1649 104.14 104.995 111.012 115.61 104.995 Fitness: 57.2727

Organism 18: 78.7476 116.68 147.148 124.901 87.6332 96.7555 Fitness: 149.995

Organism 19: 137.785 100.688 91.4287 97.1743 113.634 107.683 Fitness: 122.272

Organism 20: 78.281 91.5757 160.598 114.75 115.61 112.398 Fitness: 106.197

Best Fitness: 225.187

Avg Fitness: 103.806

================================================================================

Organism 1: 78.7476 116.68 147.148 107.97 115.61 104.995 Fitness: 118.645

Organism 2: 100.545 129.24 91.5757 104.995 94.7146 91.4287 Fitness: 115.67

Organism 3: 102.701 75.9892 147.148 124.901 125.971 104.995 Fitness: 125.573

Organism 4: 96.4426 73.9282 91.4287 106.099 113.634 104.995 Fitness: 56.7738

Organism 5: 78.7476 116.68 147.148 107.97 115.61 104.995 Fitness: 68.6453

Organism 6: 78.7476 104.995 147.148 100.688 115.61 136.121 Fitness: 76.3628

Organism 7: 126.672 78.7476 147.148 104.67 100.688 104.995 Fitness: 75.576

Organism 8: 78.281 70.5318 91.4287 88.7861 113.634 104.995 Fitness: 106.773

Organism 9: 78.7476 82.6167 147.148 100.688 115.61 136.121 Fitness: 176.362

Organism 10: 78.7476 100.688 104.995 124.901 126.672 147.148 Fitness: 100.575

Organism 11: 78.7476 106.344 147.148 107.97 82.9546 91.4287 Fitness: 158.642

Organism 12: 113.634 116.68 91.6456 104.995 115.61 104.995 Fitness: 155.669

Organism 13: 102.701 73.9282 75.5749 124.901 113.634 104.995 Fitness: 225.571

Organism 14: 82.9546 75.9892 147.148 106.099 125.971 104.995 Fitness: 26.7733

Organism 15: 91.0348 104.995 147.148 100.688 115.61 104.995 Fitness: 151.358

Organism 16: 78.7476 116.68 147.148 107.97 115.61 135.105 Fitness: 133.643

Organism 17: 78.281 91.5757 91.4287 106.099 100.688 104.995 Fitness: 81.7738

Organism 18: 128.408 78.7476 147.148 124.901 113.634 104.995 Fitness: 150.575

Organism 19: 78.7476 91.1021 104.995 78.868 115.61 147.148 Fitness: 176.359

Organism 20: 78.7476 104.995 122.152 124.901 126.672 139.179 Fitness: 85.5746

Best Fitness: 225.571

Avg Fitness: 118.345

================================================================================

Organism 1: 104.995 58.2942 99.9256 113.634 116.087 102.701 Fitness: 75.1918

Organism 2: 78.7476 82.6167 147.148 100.38 115.61 136.121 Fitness: 75.9786

Organism 3: 126.672 78.7476 147.148 104.67 100.688 104.995 Fitness: 54.9608

Organism 4: 78.7476 116.68 147.148 107.97 115.61 104.995 Fitness: 133.257

Organism 5: 78.7476 116.68 147.148 107.97 118.365 135.105 Fitness: 83.2611

Organism 6: 78.7476 116.68 147.148 107.97 115.61 135.105 Fitness: 58.2611

Organism 7: 102.701 96.6255 75.5749 124.901 113.634 91.485 Fitness: 110.189

Organism 8: 102.701 75.9892 147.148 124.901 125.971 104.995 Fitness: 125.188

Organism 9: 103.036 100.688 104.995 124.901 126.672 144.851 Fitness: 75.1918

Organism 10: 104.995 70.5318 91.4287 88.7861 78.281 113.634 Fitness: 114.077

Organism 11: 81.6167 73.9282 147.148 124.901 113.634 136.121 Fitness: 100.192

Organism 12: 78.7476 82.6167 91.0864 85.0475 127.995 104.995 Fitness: 150.977

Organism 13: 126.672 116.68 166.696 107.97 132.143 104.995 Fitness: 108.26

Organism 14: 78.7476 63.8134 147.148 104.67 100.688 104.995 Fitness: 54.9608

Organism 15: 78.7476 126.044 147.148 107.97 115.61 146.114 Fitness: 83.2611

Organism 16: 78.7476 94.8118 142.27 107.97 104.91 135.105 Fitness: 58.2611

Organism 17: 134.139 73.9282 147.148 102.701 113.634 104.995 Fitness: 100.192

Organism 18: 102.701 75.9892 75.5749 124.901 125.971 104.995 Fitness: 239.245

Organism 19: 78.281 124.901 113.798 100.688 113.634 104.995 Fitness: 150.187

Organism 20: 78.7476 70.5318 91.4287 88.7861 126.672 147.148 Fitness: 39.0764

Best Fitness: 239.245

Avg Fitness: 116.21

================================================================================

Organism 1: 102.701 96.6255 75.5749 124.901 113.634 91.485 Fitness: 125.174

Organism 2: 78.7476 82.6167 147.148 100.38 115.61 136.121 Fitness: 175.65

Organism 3: 78.7476 94.8118 142.27 107.97 104.91 135.105 Fitness: 108.244

Organism 4: 78.7476 94.8118 142.27 107.97 104.91 135.105 Fitness: 108.244

Organism 5: 78.7476 82.6167 75.5126 98.0221 127.995 104.995 Fitness: 45.3207

Organism 6: 104.995 64.6199 91.4287 88.7861 78.281 113.634 Fitness: 139.059

Organism 7: 126.672 116.68 166.696 107.97 132.143 104.995 Fitness: 108.244

Organism 8: 102.701 75.9892 147.148 124.901 125.971 86.0224 Fitness: 242.864

Organism 9: 104.995 70.5318 91.4287 88.7861 78.281 113.634 Fitness: 114.06

Organism 10: 78.7476 82.6167 147.148 100.38 115.61 136.121 Fitness: 100.654

Organism 11: 113.634 96.6255 147.148 115.709 78.7476 124.901 Fitness: 175.17

Organism 12: 102.701 72.6416 75.5749 100.38 115.61 91.485 Fitness: 135.653

Organism 13: 78.7476 71.0472 142.27 107.97 104.91 135.105 Fitness: 183.24

Organism 14: 78.7476 135.105 94.8118 107.97 104.91 142.27 Fitness: 83.2444

Organism 15: 104.995 70.5318 101.975 88.7861 127.995 104.995 Fitness: 114.06

Organism 16: 78.7476 99.991 91.4287 85.0475 78.281 113.634 Fitness: 60.3214

Organism 17: 126.672 116.68 147.148 124.901 151.823 104.995 Fitness: 95.1748

Organism 18: 75.9892 102.701 115.422 107.97 125.971 166.696 Fitness: 158.242

Organism 19: 111.312 99.5844 147.148 88.7861 78.281 113.634 Fitness: 89.06

Organism 20: 78.7476 70.5318 91.4287 100.38 115.61 136.121 Fitness: 50.6541

Best Fitness: 242.864

Avg Fitness: 124.619

================================================================================

Organism 1: 104.995 70.5318 91.4287 88.7861 78.281 113.634 Fitness: 138.971

Organism 2: 78.7476 70.5318 91.4287 100.38 115.61 136.121 Fitness: 125.565

Organism 3: 104.995 64.6199 91.4287 88.7861 78.281 113.634 Fitness: 113.971

Organism 4: 85.3049 75.9892 147.148 124.901 125.971 86.0224 Fitness: 135.083

Organism 5: 124.901 96.6255 75.5749 102.701 116.602 91.485 Fitness: 100.085

Organism 6: 115.61 82.6167 147.148 100.38 78.7476 152.463 Fitness: 75.565

Organism 7: 78.7476 94.8118 142.27 107.97 104.91 135.105 Fitness: 83.1552

Organism 8: 78.7476 63.2029 147.148 100.38 115.61 136.121 Fitness: 100.565

Organism 9: 78.7476 94.8118 142.27 107.97 104.91 135.105 Fitness: 118.155

Organism 10: 102.701 96.6255 75.5749 124.901 113.634 91.485 Fitness: 150.082

Organism 11: 104.995 58.9513 91.4287 100.38 115.61 113.634 Fitness: 75.565

Organism 12: 78.7476 70.5318 91.4287 70.1858 78.281 136.121 Fitness: 138.971

Organism 13: 102.701 91.4287 64.7143 123.802 125.971 113.634 Fitness: 125.086

Organism 14: 104.995 64.6199 147.148 88.7861 78.281 86.0224 Fitness: 113.971

Organism 15: 98.3665 120.213 75.5749 100.38 68.6764 113.634 Fitness: 150.562

Organism 16: 102.701 82.6167 147.148 124.901 115.61 136.121 Fitness: 150.084

Organism 17: 78.7476 82.6167 142.27 100.38 115.61 136.121 Fitness: 175.561

Organism 18: 56.7188 94.8118 147.148 97.4184 104.91 135.105 Fitness: 256.686

Organism 19: 102.701 94.8118 75.5749 107.97 113.634 91.485 Fitness: 183.154

Organism 20: 142.27 117.634 80.95 124.901 91.6639 96.6255 Fitness: 150.086

Best Fitness: 256.686

Avg Fitness: 129.155

================================================================================

Organism 1: 78.7476 70.5318 91.4287 70.1858 78.281 136.121 Fitness: 18.4314

Organism 2: 104.995 82.098 91.4287 100.38 115.61 135.072 Fitness: 145.6658

Organism 3: 102.701 96.6255 75.5749 124.901 93.5728 91.485 Fitness: 275.183

Organism 4: 78.7476 70.5318 91.4287 70.1858 77.8654 136.121 Fitness: 45.4706

Organism 5: 104.995 58.9513 91.4287 100.38 115.61 113.634 Fitness: 100.666

Organism 6: 104.995 58.9513 99.6506 100.38 115.61 113.634 Fitness: 50.6657

Organism 7: 56.7188 117.953 147.148 97.4184 104.91 135.105 Fitness: 47.7037

Organism 8: 124.901 96.6255 102.701 75.5749 113.634 91.485 Fitness: 125.184

Organism 9: 142.27 136.121 78.7476 100.38 115.61 82.6167 Fitness: 50.6657

Organism 10: 56.7188 94.8118 147.148 86.3775 104.91 135.105 Fitness: 47.7037

Organism 11: 64.6199 97.0238 91.4287 104.995 63.519 113.634 Fitness: 99.0711

Organism 12: 102.701 58.9513 75.5749 124.901 113.634 91.485 Fitness: 100.187

Organism 13: 104.995 90.7478 91.4287 100.38 115.61 113.634 Fitness: 85.6653

Organism 14: 104.995 76.948 66.8686 100.38 115.61 113.634 Fitness: 75.6658

Organism 15: 56.7188 58.9513 147.148 100.38 134.021 113.634 Fitness: 150.661

Organism 16: 104.995 94.8118 91.4287 97.4184 104.91 135.105 Fitness: 47.7037

Organism 17: 78.7476 82.6167 142.27 124.901 113.634 136.121 Fitness: 175.181

Organism 18: 102.701 75.5749 91.485 83.4891 115.61 96.6255 Fitness: 125.666

Organism 19: 104.995 72.5455 147.148 97.4184 78.281 135.105 Fitness: 72.7039

Organism 20: 56.7188 113.634 91.4287 78.8001 104.91 64.6199 Fitness: 69.0713

Best Fitness: 275.183

Avg Fitness: 137.72

## Appendix 3

Population Size: 16

Amount of Food: 15

Amount of Poison: 15

Generation Time in Seconds: 30

Population Survivors (% of population): 50

Selection Type (Roulette = 0) (Random = 1) (Tournament = 2): 1

Crossover Type (Uniform = 0) (Single-Point = 1) (Multi-Point = 2): 1

Primary Mutation Type (Creep = 0) (Random Resetting = 1): 0

Creep Value Range: 20

Secondary Mutation Type (None = 0) (Swap = 1) (Scramble = 2) (Inversion = 3): 0

Mutation Chance Percentage (0.0f - 100.0f): 10

Organism 1: 118.755 100.052 116.332 111.498 85.7218 94.5316 Fitness: 76.7281

Organism 2: 111.662 115.109 111.835 91.9597 117.922 101.144 Fitness: 182.187

Organism 3: 119.606 88.2356 118.732 118.851 101.697 86.0328 Fitness: 109.081

Organism 4: 91.2118 93.665 98.0752 98.712 111.683 112.3 Fitness: 163.94

Organism 5: 87.857 88.6036 84.1864 108.049 89.9685 83.931 Fitness: 148.278

Organism 6: 88.8062 80.0751 94.9389 88.0938 93.9657 118.046 Fitness: 103.322

Organism 7: 102.59 116.296 83.3136 90.509 95.7231 102.289 Fitness: 55.7385

Organism 8: 93.6558 108.543 114.564 117.905 109.827 93.6154 Fitness: 208.129

Organism 9: 92.0055 113.863 112.433 92.9871 101.445 99.8905 Fitness: 133.217

Organism 10: 110.909 115.209 106.018 107.89 107.379 89.6726 Fitness: 173.116

Organism 11: 112.961 103.227 102.159 82.4149 106.575 108.976 Fitness: 82.6445

Organism 12: 97.9942 97.2912 108.153 85.7762 116.258 116.78 Fitness: 101.006

Organism 13: 111.042 89.6432 96.8167 107.122 112.099 83.2429 Fitness: 97.3512

Organism 14: 81.5334 97.6248 96.2762 118.915 93.9311 114.085 Fitness: 134.142

Organism 15: 114.029 89.4261 98.7649 111.534 84.5259 98.8115 Fitness: 101.763

Organism 16: 96.0443 87.7239 88.148 106.074 84.158 84.0092 Fitness: 81.304

Best Fitness: 208.129

Avg Fitness: 121.997

================================================================================

Organism 1: 112.961 103.227 102.159 82.4149 93.9311 114.085 Fitness: 27.6804

Organism 2: 91.2118 83.805 98.0752 98.712 85.7218 94.5316 Fitness: 28.1144

Organism 3: 88.8062 80.0751 94.9389 88.0938 93.9657 118.046 Fitness: 29.5607

Organism 4: 87.857 88.6036 84.1864 108.049 108.533 83.931 Fitness: 98.746

Organism 5: 92.0055 113.863 112.433 92.9871 101.445 99.8905 Fitness: 133.685

Organism 6: 118.755 100.052 116.332 101.786 85.7218 94.5316 Fitness: 77.1959

Organism 7: 91.2118 93.665 98.0752 107.454 111.683 112.3 Fitness: 64.4095

Organism 8: 114.029 89.4261 98.7649 111.534 84.5259 98.8115 Fitness: 77.231

Organism 9: 81.5334 97.6248 96.2762 118.915 97.2972 114.085 Fitness: 84.6112

Organism 10: 112.961 103.227 102.159 82.4149 117.467 108.976 Fitness: 73.1123

Organism 11: 87.857 88.6036 84.1864 108.049 101.445 99.8905 Fitness: 98.7457

Organism 12: 92.0055 113.863 112.433 92.9871 107.791 83.931 Fitness: 58.6845

Organism 13: 118.755 100.052 116.332 93.3178 111.683 107.392 Fitness: 77.1959

Organism 14: 88.8062 80.0751 94.9389 88.0938 84.5259 98.8115 Fitness: 78.7913

Organism 15: 114.029 89.4261 112.439 111.534 93.9657 118.046 Fitness: 152.227

Organism 16: 81.5334 97.6248 96.2762 118.915 106.575 108.976 Fitness: 59.6121

Best Fitness: 152.227

Avg Fitness: 76.2252

================================================================================

Organism 1: 92.0055 113.863 112.433 92.9871 107.791 83.931 Fitness: 36.2642

Organism 2: 81.5334 97.6248 96.2762 118.915 106.575 108.976 Fitness: 112.192

Organism 3: 114.029 89.4261 98.7649 116.569 84.5259 98.8115 Fitness: 54.811

Organism 4: 87.857 88.6036 84.1864 108.049 108.533 83.931 Fitness: 101.326

Organism 5: 81.5334 97.6248 90.6992 118.915 98.2756 114.085 Fitness: 137.192

Organism 6: 114.029 89.4261 112.439 111.534 93.9657 118.046 Fitness: 54.811

Organism 7: 91.2118 83.805 98.0752 98.712 85.7218 94.5316 Fitness: 91.989

Organism 8: 87.857 88.6036 84.1864 108.049 101.445 99.8905 Fitness: 51.3259

Organism 9: 79.4616 126.74 112.433 92.9871 126.748 108.976 Fitness: 61.2644

Organism 10: 81.5334 97.6248 96.2762 118.915 106.575 83.931 Fitness: 87.1916

Organism 11: 114.029 89.4261 98.7649 111.534 84.5259 83.931 Fitness: 104.81

Organism 12: 87.857 88.6036 84.1864 108.049 108.533 98.8115 Fitness: 126.326

Organism 13: 81.5334 111.324 96.2762 118.915 97.2972 118.046 Fitness: 62.1922

Organism 14: 114.029 89.4261 112.439 111.534 93.9657 115.286 Fitness: 79.811

Organism 15: 91.2118 83.805 110.669 98.712 85.7218 99.8905 Fitness: 66.9896

Organism 16: 87.857 88.6036 84.1864 108.049 101.445 94.5316 Fitness: 226.323

Best Fitness: 226.323

Avg Fitness: 90.9262

================================================================================

Organism 1: 87.857 88.6036 84.1864 112.278 101.445 99.8905 Fitness: 148.826

Organism 2: 114.029 89.4261 98.7649 121.791 84.5259 83.931 Fitness: 77.3118

Organism 3: 81.5334 97.6248 96.2762 118.915 121.174 70.9489 Fitness: 159.69

Organism 4: 114.029 89.4261 112.439 111.534 93.9657 104.968 Fitness: 102.31

Organism 5: 91.2118 83.805 95.7145 98.712 85.7218 99.8905 Fitness: 139.486

Organism 6: 79.4616 126.74 112.433 92.9871 126.748 108.976 Fitness: 83.7654

Organism 7: 91.2118 83.805 98.0752 98.712 85.7218 94.5316 Fitness: 139.49

Organism 8: 81.5334 111.324 96.2762 118.915 97.2972 118.046 Fitness: 134.693

Organism 9: 87.3696 88.6036 84.1864 111.534 84.5259 83.931 Fitness: 102.312

Organism 10: 114.029 89.4261 98.7649 108.049 101.445 99.8905 Fitness: 148.827

Organism 11: 95.1999 97.6248 96.2762 111.534 93.9657 118.046 Fitness: 102.312

Organism 12: 114.029 89.4261 112.439 120.248 106.575 83.931 Fitness: 109.692

Organism 13: 91.2118 83.805 110.669 92.9871 126.748 108.976 Fitness: 133.765

Organism 14: 79.4616 126.74 112.433 98.712 85.7218 99.8905 Fitness: 64.4903

Organism 15: 91.2118 83.805 98.0752 118.915 97.2972 118.046 Fitness: 109.693

Organism 16: 81.5334 111.324 96.2762 98.712 85.7218 77.2542 Fitness: 64.4903

Best Fitness: 159.69

Avg Fitness: 113.822

================================================================================

Organism 1: 114.029 104.158 98.7649 121.791 84.5259 83.931 Fitness: 162.57

Organism 2: 79.4616 126.74 112.433 92.9871 126.748 108.976 Fitness: 158.768

Organism 3: 95.1999 97.6248 96.2762 111.534 93.9657 118.046 Fitness: 52.319

Organism 4: 91.2118 83.805 98.0752 98.712 85.7218 94.5316 Fitness: 164.496

Organism 5: 114.029 89.4261 112.439 120.248 106.575 83.931 Fitness: 96.0332

Organism 6: 87.857 88.6036 84.1864 112.278 101.445 99.8905 Fitness: 78.0632

Organism 7: 81.5334 111.324 96.2762 98.712 84.5872 77.2542 Fitness: 214.493

Organism 8: 81.5334 111.324 96.2762 118.915 97.2972 118.046 Fitness: 84.7003

Organism 9: 114.029 89.4261 98.7649 121.791 84.5259 108.976 Fitness: 87.5751

Organism 10: 79.4616 126.74 112.433 98.5381 126.748 83.931 Fitness: 68.7727

Organism 11: 95.1999 97.6248 96.2762 122.338 93.9657 94.5316 Fitness: 77.3191

Organism 12: 91.2118 83.805 98.0752 98.712 85.7218 118.046 Fitness: 114.498

Organism 13: 114.029 89.4261 112.439 120.248 106.575 99.8905 Fitness: 161.028

Organism 14: 87.857 88.6036 79.1661 112.278 83.8909 83.931 Fitness: 103.062

Organism 15: 81.5334 111.324 96.2762 98.712 85.7218 118.046 Fitness: 89.4976

Organism 16: 81.5334 111.324 96.2762 118.915 97.2972 95.9576 Fitness: 109.699

Best Fitness: 214.493

Avg Fitness: 113.931

================================================================================

Organism 1: 95.1999 97.6248 96.2762 111.534 99.5665 118.046 Fitness: 77.3173

Organism 2: 122.136 89.4261 98.7649 121.791 84.5259 108.976 Fitness: 112.573

Organism 3: 109.488 84.1977 96.2762 122.338 93.9657 94.5316 Fitness: 173.119

Organism 4: 91.2118 83.805 98.0752 98.712 85.7218 94.5316 Fitness: 89.4958

Organism 5: 81.5334 111.324 96.2762 118.915 97.2972 95.9576 Fitness: 109.698

Organism 6: 91.2118 83.805 98.0752 98.712 85.7218 118.046 Fitness: 64.4958

Organism 7: 81.5334 111.324 96.2762 118.915 97.2972 118.046 Fitness: 119.697

Organism 8: 87.857 88.6036 84.1864 112.278 101.445 99.8905 Fitness: 203.057

Organism 9: 94.8528 97.6248 96.2762 111.534 87.9053 108.976 Fitness: 127.317

Organism 10: 114.029 89.4261 100.127 121.791 84.5259 118.046 Fitness: 112.573

Organism 11: 95.1999 97.6248 96.2762 122.338 93.9657 94.5316 Fitness: 98.1213

Organism 12: 103.096 83.805 98.0752 98.712 85.7218 94.5316 Fitness: 39.4954

Organism 13: 81.5334 111.324 96.2762 118.915 97.2972 118.046 Fitness: 59.6984

Organism 14: 91.2118 83.805 98.0752 98.712 85.7218 95.9576 Fitness: 64.4956

Organism 15: 81.5334 111.324 96.2762 118.915 97.2972 99.8905 Fitness: 59.6984

Organism 16: 87.857 88.6036 84.1864 112.278 101.445 118.046 Fitness: 53.0615

Best Fitness: 203.057

Avg Fitness: 97.7446

================================================================================

Organism 1: 81.5334 111.324 96.2762 118.915 97.2972 95.9576 Fitness: 59.6125

Organism 2: 95.1999 97.6248 96.2762 111.534 99.5665 118.046 Fitness: 127.231

Organism 3: 94.8528 97.6248 96.2762 93.2983 87.9053 108.976 Fitness: 52.2313

Organism 4: 109.488 84.1977 96.2762 122.338 93.9657 94.5316 Fitness: 63.0354

Organism 5: 103.096 83.805 98.0752 98.712 85.7218 94.5316 Fitness: 39.4095

Organism 6: 91.2118 83.805 98.0752 98.712 85.7218 118.046 Fitness: 64.4099

Organism 7: 91.2118 83.805 98.0752 98.712 85.7218 94.5316 Fitness: 64.4097

Organism 8: 95.1999 97.6248 96.2762 122.338 93.9657 94.5316 Fitness: 63.0354

Organism 9: 91.8629 111.324 96.2762 118.915 97.2972 118.046 Fitness: 59.6125

Organism 10: 95.1999 97.6248 96.2762 111.534 99.5665 95.9576 Fitness: 52.2313

Organism 11: 94.8528 97.6248 96.2762 111.534 87.9053 94.5316 Fitness: 77.2314

Organism 12: 109.488 78.7018 96.2762 122.338 93.9657 108.976 Fitness: 63.0354

Organism 13: 103.096 83.805 98.0752 98.712 85.7218 118.046 Fitness: 64.4099

Organism 14: 91.2118 83.805 98.0752 98.712 85.7218 80.9364 Fitness: 64.4099

Organism 15: 91.2118 83.805 98.0752 98.712 85.7218 94.5316 Fitness: 114.41

Organism 16: 95.1999 97.6248 86.4699 122.338 93.9657 94.5316 Fitness: 173.03

Best Fitness: 173.03

Avg Fitness: 75.1091

================================================================================

Organism 1: 91.2118 83.805 98.0752 98.712 85.7218 80.9364 Fitness: 64.3818

Organism 2: 95.1999 97.6248 96.2762 111.534 99.5665 95.9576 Fitness: 102.204

Organism 3: 94.8528 97.6248 96.2762 111.534 101.125 94.5316 Fitness: 77.2037

Organism 4: 81.5334 111.324 96.2762 118.915 97.2972 95.9576 Fitness: 109.584

Organism 5: 95.1999 97.6248 86.4699 122.338 93.9657 94.5316 Fitness: 163.003

Organism 6: 91.2118 83.805 98.0752 98.712 85.7218 94.5316 Fitness: 39.3818

Organism 7: 94.8528 97.6248 114.15 93.2983 87.9053 108.976 Fitness: 83.9685

Organism 8: 109.488 78.7018 96.2762 122.338 93.9657 108.976 Fitness: 63.0077

Organism 9: 91.2118 83.805 98.0752 98.712 99.5665 95.9576 Fitness: 39.3818

Organism 10: 95.1999 97.6248 96.2762 111.534 85.7218 80.9364 Fitness: 52.2036

Organism 11: 94.8528 97.6248 96.2762 111.534 97.2972 95.9576 Fitness: 137.202

Organism 12: 81.5334 111.324 96.2762 118.915 87.9053 94.5316 Fitness: 84.5848

Organism 13: 95.1999 97.6248 86.4699 122.338 85.7218 94.5316 Fitness: 113.007

Organism 14: 75.1194 83.805 98.0752 98.712 93.9657 94.5316 Fitness: 39.3818

Organism 15: 94.8528 97.6248 96.2762 93.2983 93.9657 108.976 Fitness: 58.968

Organism 16: 109.488 78.7018 96.2762 122.338 87.9053 108.976 Fitness: 88.0077

Best Fitness: 163.003

Avg Fitness: 82.2169

================================================================================

Organism 1: 109.488 78.7018 96.2762 122.338 93.9657 108.976 Fitness: 62.9123

Organism 2: 75.1194 83.805 98.0752 98.712 93.9657 94.5316 Fitness: 64.2868

Organism 3: 81.9416 97.6248 96.2762 111.534 85.7218 80.9364 Fitness: 77.1083

Organism 4: 94.8528 97.6248 96.2762 93.2983 93.9657 108.976 Fitness: 33.8726

Organism 5: 109.488 78.7018 96.2762 117.243 103.888 108.976 Fitness: 112.912

Organism 6: 91.2118 95.0714 98.0752 81.4833 85.7218 88.7234 Fitness: 89.2868

Organism 7: 95.1999 97.6248 86.4699 122.338 93.9657 94.5316 Fitness: 62.9123

Organism 8: 81.5334 111.324 96.2762 118.915 97.2972 106.528 Fitness: 59.4894

Organism 9: 109.488 78.7018 98.0752 98.712 93.9657 94.5316 Fitness: 89.2868

Organism 10: 75.1194 83.805 96.2762 122.338 93.9657 122.296 Fitness: 87.9123

Organism 11: 95.1999 89.201 96.2762 93.2983 93.9657 108.976 Fitness: 83.873

Organism 12: 94.8528 97.6248 96.2762 112.192 85.7218 80.9364 Fitness: 127.108

Organism 13: 109.488 78.7018 98.0752 98.712 85.7218 74.3669 Fitness: 64.2866

Organism 14: 91.2118 83.805 96.2762 122.338 87.9053 120.721 Fitness: 112.911

Organism 15: 104.346 97.6248 96.2762 118.915 97.2972 84.3072 Fitness: 109.489

Organism 16: 81.5334 111.324 86.4699 122.338 93.9657 94.5316 Fitness: 62.9123

Best Fitness: 127.108

Avg Fitness: 81.285

================================================================================

Organism 1: 91.2118 95.0714 98.0752 81.4833 85.7218 88.7234 Fitness: 27.2998

Organism 2: 89.7581 78.7018 98.0752 98.712 85.7218 74.3669 Fitness: 64.296

Organism 3: 109.488 78.7018 96.2762 122.338 93.9657 108.976 Fitness: 87.9204

Organism 4: 75.1194 83.805 98.0752 98.712 93.9657 94.5316 Fitness: 64.296

Organism 5: 104.346 97.6248 96.2762 118.915 97.2972 84.3072 Fitness: 59.4986

Organism 6: 81.9416 97.6248 96.2762 111.534 85.7218 80.9364 Fitness: 77.1176

Organism 7: 81.5334 111.324 86.4699 122.338 113.1 85.6013 Fitness: 87.9216

Organism 8: 81.5334 111.324 96.2762 118.915 97.2972 106.528 Fitness: 84.4987

Organism 9: 91.2118 91.2966 116.23 98.712 70.7139 74.3669 Fitness: 74.296

Organism 10: 109.488 78.7018 97.0314 81.4833 85.7218 88.7234 Fitness: 122.067

Organism 11: 109.488 78.7018 98.0752 98.712 80.3803 94.5316 Fitness: 89.296

Organism 12: 75.1194 66.9369 96.2762 134.432 93.9657 108.976 Fitness: 162.917

Organism 13: 104.346 97.6248 96.2762 111.534 85.7218 80.9364 Fitness: 112.116

Organism 14: 81.9416 97.6248 96.2762 118.915 97.2972 84.3072 Fitness: 134.495

Organism 15: 81.5334 111.324 96.2762 118.915 97.2972 106.528 Fitness: 134.496

Organism 16: 81.5334 111.324 86.4699 122.338 93.9657 94.5316 Fitness: 137.916

Best Fitness: 162.917

Avg Fitness: 95.028

================================================================================

Organism 1: 91.2118 95.0714 98.0752 81.4833 85.7218 95.8632 Fitness: 27.2308

Organism 2: 104.346 97.6248 110.014 81.4833 85.7218 88.7234 Fitness: 27.2308

Organism 3: 81.5334 92.3116 71.9146 98.712 85.7218 74.3669 Fitness: 27.9985

Organism 4: 81.5334 111.324 89.3086 108.502 97.2972 106.528 Fitness: 59.2932

Organism 5: 81.5334 111.324 96.2762 118.915 97.2972 106.528 Fitness: 59.2932

Organism 6: 121.606 78.7018 97.0314 81.4833 85.7218 90.9289 Fitness: 46.8616

Organism 7: 109.488 78.7018 96.2762 122.338 93.9657 108.976 Fitness: 112.715

Organism 8: 81.5334 111.324 86.4699 122.338 113.1 85.6013 Fitness: 112.713

Organism 9: 89.7581 78.7018 105.922 91.4181 85.7218 74.3669 Fitness: 64.0903

Organism 10: 95.0104 97.6248 96.2762 118.915 97.2972 84.3072 Fitness: 109.292

Organism 11: 81.5334 111.324 96.2762 118.915 97.2972 106.528 Fitness: 159.288

Organism 12: 81.5334 111.324 96.2762 118.915 97.2972 106.528 Fitness: 209.287

Organism 13: 109.488 78.7018 97.0314 122.338 93.9657 106.872 Fitness: 112.713

Organism 14: 109.488 78.7018 96.2762 81.4833 104.883 88.7234 Fitness: 121.862

Organism 15: 89.7581 78.7018 98.0752 122.338 113.1 85.6013 Fitness: 147.711

Organism 16: 91.2118 95.0714 98.0752 118.915 97.2972 84.3072 Fitness: 159.287

Best Fitness: 209.287

Avg Fitness: 97.3042

================================================================================

Organism 1: 109.488 78.7018 96.2762 81.4833 104.883 88.7234 Fitness: 27.2428

Organism 2: 81.5334 111.324 110.014 81.4833 85.7218 88.7234 Fitness: 27.2428

Organism 3: 81.5334 111.324 96.2762 118.915 97.2972 106.528 Fitness: 109.298

Organism 4: 104.346 97.6248 93.6613 81.4833 85.7218 88.7234 Fitness: 71.867

Organism 5: 89.7581 78.7018 105.922 91.4181 85.7218 74.3669 Fitness: 31.8013

Organism 6: 81.5334 111.324 89.3086 108.502 97.2972 117.364 Fitness: 73.8858

Organism 7: 95.0104 97.6248 96.2762 118.915 97.2972 84.3072 Fitness: 69.2983

Organism 8: 81.5334 111.324 96.2762 118.915 97.2972 106.528 Fitness: 94.2964

Organism 9: 121.606 78.7018 78.3394 81.4833 85.7218 90.9289 Fitness: 71.867

Organism 10: 104.346 97.6248 95.15 118.915 97.2972 106.528 Fitness: 59.2983

Organism 11: 89.7581 78.7018 89.3086 101.635 97.2972 106.528 Fitness: 123.886

Organism 12: 81.5334 111.324 105.922 91.4181 85.7218 74.3669 Fitness: 81.8018

Organism 13: 109.488 78.7018 96.2762 118.915 97.2972 84.3072 Fitness: 84.2984

Organism 14: 95.0104 97.6248 96.2762 81.4833 104.883 88.7234 Fitness: 121.865

Organism 15: 81.5334 111.324 97.0314 81.4833 85.7218 90.9289 Fitness: 96.867

Organism 16: 130.687 78.7018 96.2762 118.915 97.2972 106.528 Fitness: 109.297

Best Fitness: 123.886

Avg Fitness: 78.382

================================================================================

Organism 1: 81.5334 111.324 105.922 91.4181 85.7218 74.3669 Fitness: 25.5188

Organism 2: 81.5334 111.324 110.014 81.4833 85.7218 88.7234 Fitness: 96.6808

Organism 3: 109.488 78.7018 96.2762 86.1228 104.883 88.7234 Fitness: 46.6805

Organism 4: 89.7581 78.7018 105.922 91.4181 85.7218 73.599 Fitness: 56.6155

Organism 5: 81.5334 111.324 105.922 91.4181 85.7218 74.3669 Fitness: 56.6155

Organism 6: 81.5334 111.324 96.2762 118.915 97.2972 106.528 Fitness: 109.11

Organism 7: 103.448 78.7018 106.331 118.915 97.2972 84.3072 Fitness: 134.109

Organism 8: 81.5334 111.324 89.3086 99.867 87.0421 117.364 Fitness: 98.6995

Organism 9: 130.687 78.7018 96.2762 133.364 79.1241 106.528 Fitness: 109.112

Organism 10: 81.5334 111.324 124.834 81.4833 85.7218 103.08 Fitness: 96.6808

Organism 11: 127.027 66.0281 96.2762 79.325 104.883 88.7234 Fitness: 96.6807

Organism 12: 89.7581 78.7018 105.922 91.4181 85.7218 74.3669 Fitness: 106.616

Organism 13: 81.5334 111.324 96.2762 118.915 97.2972 84.3072 Fitness: 84.1114

Organism 14: 109.488 78.7018 96.2762 118.915 97.2972 106.528 Fitness: 59.1121

Organism 15: 81.5334 111.324 89.3086 92.413 97.2972 106.528 Fitness: 98.6991

Organism 16: 130.687 78.7018 96.2762 118.915 97.2972 117.364 Fitness: 134.109

Best Fitness: 134.109

Avg Fitness: 88.0719

================================================================================

Organism 1: 127.027 66.0281 94.4778 79.325 104.883 88.7234 Fitness: 44.4134

Organism 2: 87.1919 111.324 96.2762 118.915 97.2972 84.3072 Fitness: 84.0035

Organism 3: 109.488 78.7018 96.2762 110.469 97.2972 106.528 Fitness: 109.004

Organism 4: 103.448 78.7018 106.331 118.915 97.2972 88.2039 Fitness: 84.0025

Organism 5: 130.687 78.7018 96.2762 133.364 79.1241 106.528 Fitness: 148.449

Organism 6: 72.5911 111.324 89.3086 92.413 97.2972 106.528 Fitness: 107.502

Organism 7: 130.687 78.7018 96.2762 118.915 117.176 117.364 Fitness: 84.0035

Organism 8: 81.5334 111.324 96.2762 118.915 117.206 106.528 Fitness: 218.997

Organism 9: 138.627 66.0281 96.2762 118.915 97.2972 84.3072 Fitness: 109.003

Organism 10: 81.5334 111.324 96.2762 79.325 104.883 88.7234 Fitness: 69.4134

Organism 11: 117.168 78.7018 96.2762 118.915 97.2972 84.3072 Fitness: 84.0034

Organism 12: 103.448 89.9526 106.331 118.915 97.2972 106.528 Fitness: 109.004

Organism 13: 130.687 78.7018 96.2762 92.413 97.2972 106.528 Fitness: 182.501

Organism 14: 81.5334 111.324 89.3086 133.364 79.1241 106.528 Fitness: 123.452

Organism 15: 130.687 84.9404 96.2762 118.915 97.2972 106.528 Fitness: 84.0025

Organism 16: 81.5334 111.324 96.2762 118.915 97.2972 117.364 Fitness: 69.0033

Best Fitness: 218.997

Avg Fitness: 106.922

================================================================================

Organism 1: 109.488 78.7018 96.2762 110.469 97.2972 106.528 Fitness: 125.751

Organism 2: 103.448 89.9526 112.973 118.915 97.2972 106.528 Fitness: 159.194

Organism 3: 81.5334 126.5 89.3086 133.364 79.1241 106.528 Fitness: 98.6457

Organism 4: 87.1919 111.324 96.2762 118.915 83.5787 84.3072 Fitness: 134.193

Organism 5: 64.117 111.324 96.2762 118.915 117.206 106.528 Fitness: 109.197

Organism 6: 130.687 84.9404 96.2762 118.915 97.2972 106.528 Fitness: 119.195

Organism 7: 130.687 78.7018 96.2762 118.915 117.176 117.364 Fitness: 59.1968

Organism 8: 130.687 78.7018 96.2762 92.413 97.2972 106.528 Fitness: 57.6951

Organism 9: 109.488 78.7018 96.2762 118.915 97.2972 106.528 Fitness: 184.192

Organism 10: 103.448 89.9526 106.331 110.469 97.2972 99.9745 Fitness: 100.75

Organism 11: 81.5334 111.324 89.3086 118.915 97.2972 84.3072 Fitness: 84.1963

Organism 12: 87.1919 111.324 96.2762 133.364 79.1241 106.528 Fitness: 73.6457

Organism 13: 81.5334 111.324 96.2762 118.915 97.2972 106.528 Fitness: 109.194

Organism 14: 130.687 84.9404 96.2762 118.915 117.206 106.528 Fitness: 214.193

Organism 15: 130.687 65.8804 96.2762 92.413 97.2972 106.528 Fitness: 82.6948

Organism 16: 130.687 78.7018 96.2762 118.915 117.176 117.364 Fitness: 94.1969

Best Fitness: 214.193

Avg Fitness: 114.133

## Appendix 4

