Evolution Simulator

Computer Science NEA

Kit Matthewson

Canditate Number: 6373
Centre Number: 58249

February 24, 2023

Abstract

The topic of Evolution is a difficult one to grasp, especially at its most complex when taught for A-Level. The subject requires knowledge of many interacting biological systems and processes along with their underlying causes. I believe that the current teaching method, consisting of presentations and written work, is suitable for considerable improvement through a computer simulation. A visual demonstration of the topic in action, with the user able to quickly see what would take thousands of years to occur in the natural world with access to key statistics as the program runs, would provide great assistance to students regardless of their ability to grasp topics. I analysed the problem using communication with the client and my own research to propose, design, and develop this solution.

Contents

Part 1

Analysis

1.1 The Problem

Evolution is a difficult topic to grasp. It involves many relatively simple processes that combine to form a more complex system. These processes include the role of genes in determining characteristics of an organism; the role these characteristics play in the chance of that organism reproducing; and how random mutations result in organisms gaining an advantage. It would be useful for biology students if they could easily see this take place, as this would allow them to better understand how evolution works. It is hard to do this without a computer simulation, as in nature it takes an extremely long time for any recognisable changes to occur. Another difficulty students have is understanding how parameters of evolution like the mutation rate or reproduction difficulty effect the organisms involved. Small changes in initial conditions can dramatically change the outcome after many generations.

1.2 The Client

My client is Mr. Knowles, a biology teacher. He teaches evolution and genetics to GCSE[1] and A-Level[2] students. The GCSE level is a subset and simplification of the A-Level, so meeting the requirements of the A-Level specification will ensure that it fulfils the requirements needed.

He teaches biology multiple times a day to different year groups. It is important to him that lessons are as easy as possible to plan and deliver without compromising on the quality of teaching he gives.

Currently, it is rare for students to need computers during his lessons. The teacher always has access to a machine, but for students to have their own would require moving to a computer lab, which can cause disruption.

1.3 Client Communication

My initial communication with my client was by email. In this email I outlined my identified problem and gave a rough idea as to what a solution might look like. In my outline of the problem and solution I made sure to use language that would communicate the problem to a biology teacher instead of a computer scientist. This required researching specific terms and definitions so that my objectives and analysis would be clear. I then asked if we could meet in person to discuss more specific details.

In his response Mr. Knowles agreed to a meeting later. During this meeting we discussed a range of topics relating to the problem, current solution, and aims of the new system. We analysed the problem and found that, as I initially proposed, some students find difficulty in understanding a complex and practical topic through only diagrams and explanations like those provided by the current system. This current system involves a PowerPoint of the topic that explains the material but, as is a limitation of this 'theory-based' approach, provides no demonstration of the topic. We discussed how a computer simulation is the only way of providing this demonstration, as the evolution topic cannot be practically shown during a lesson in a school lab. Therefore, we agreed that a computer simulation that aims to match the real biology as closely as possible was a good solution to the problem. We could then begin

[1] https://filestore.aqa.org.uk/resources/biology/specifications/AQA-8461-SP-2016.pdf § 4.6
[2] https://filestore.aqa.org.uk/resources/biology/specifications/AQA-7401-7402-SP-2015.pdf § 3.7

to outline the aims and possibilities of a new system. As basic aims we decided that the solution should at least:

- Provide a way to visually see the organisms in their environment.
- Simulate some number of attributes of the organisms. We determined that the specific attributes that would be modelled would be an important decision and so deciding on them would be a key choice, but I explained that it would be possible to program the simulation in such a way to make it relatively simple to add or modify this list of attributes.
- Present data in a visual and meaningful way such that it is easy to understand.

As further possibilities of the system, we discussed:

- Allowing the population to be separated into distinct environments at some point in time.
- Organisms changing in appearance to reflect their attributes.
- Predators and prey also being simulated.

Kit Matthewson <kit.matthewson@churcherscollege.com>
Thu 24/03/2022 11:19
To: Ian Knowles <iknowles@churcherscollege.com>
Hi Mr Knowles,

I plan to make an 'evolution-simulator' for my computing NEA and I have identified you as a possible client. The problem I am attempting to solve is:

Evolution is a difficult topic to grasp. It involves many relatively simple processes that combine to form a more complex system. These processes include the role of genes in determining characteristics of an organism; the role these characteristics play in the chance of that organism reproducing; and how random mutations result in organisms gaining an advantage.

It would be useful for biology students if they could easily see this take place, as this would allow them to better understand how evolution works. It is hard to do this without a computer simulation, as in nature it takes an extremely long time for any recognisable changes to occur.

Another difficulty students have is understanding how parameters of evolution like the mutation rate or reproduction difficulty effect the organisms involved. Small changes in initial conditions can dramatically change the outcome after many generations.

My general idea is to create a program that will demonstrate evolution by simulating simple organisms that have a basic genome. This will then mutate when the organisms reproduce, hopefully leading to the organisms developing a genome that optimises for reproduction ability.

If you are interested it would be useful to know of any ideas or suggestions you have. We can meet in person if you would find it easier and have the time.

Thanks a lot,
Kit

Figure 1.1: Initial email to client

1.3.1 Communication Analysis

The client stressed the importance of the visual feedback to the user, including both the way the simulation looks, and the way data is presented. To achieve this, I will use the Unity Game Engine as it allows you to easily create and control 3D graphics.

An organism with a set of attributes would be modelled with a `struct` containing a list of `structs` to represent each attribute:

```
public struct Organism:
    public Attribute[] attributes = { ... };

public struct Attribute:
    public const String name;
    public const double weight;
    public double value;
```

This would make it simple to add or modify attributes like the client suggested, for instance if it is decided that a new attribute is needed it can easily be added to the organisms. All areas of the program must be written so that the program is expandable in the future, so that extension objectives can be met.

I will have to research methods of data presentation other than graphs so that the results can be engaging and understandable, although graphs will of course still be needed for some types of data.

1.4 The Current System

Currently, Mr. Knowles teaches using textbooks and PowerPoints (1.2 and 1.3). The only way students interact with the content is by answering questions on it. This can lead to some students not understanding all the content as they may learn the material if it is delivered in a demonstrative way.

Students are taught evolution over a period of one to two weeks. In each lesson a topic is introduced via PowerPoint and handouts. Students then answer questions on worksheets and from the textbook. If a student doesn't understand any of the topic material they can ask their teacher for help.

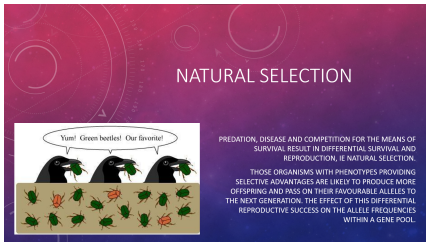

Figure 1.2: Example slide 1

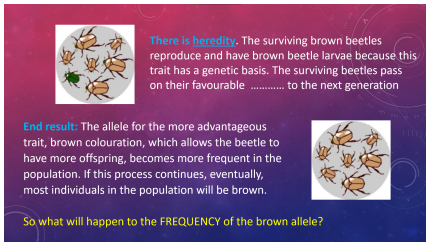

Figure 1.3: Example slide 2

1.5 The User

The user of my program will be my client's students. I cannot assume that they have much computer knowledge and so my program must be designed around this. Through client communication I learnt that the users would want information presented in clear and visual ways, so data should be presented as graphs and visualisations instead of large tables.

The user has access to a Windows 10 machine with average hardware. Most have at least:

- 8 GB of DDR3 memory
- A 2nd generation Intel i5 processor
- A 500 GB hard drive
- Onboard graphics

The client will also need to run the program, and has access to a laptop with:

-

This is suitable for most software used in a school, like word processors, presentation tools, and web browsers.

1.6 User Needs

The user needs the program to be simple to use because they do not have technical knowledge. This means that all interactions with the program should be with a simple graphical user interface.

The output from the program should be displayed in a clear way, including both the running of the simulation and any analysis of it, for instance graphs and statistical tables. For my program to be a

useful teaching tool, it must cover the content in the evolution and genetics sections of the GCSE and A-Level specifications. Including content beyond the A-Level syllabus would confuse the user, and not covering the full breadth would make my program less useful.

The program will need to run efficiently, as the relatively basic computers that the user has access to could become slow quickly if the software is not optimised properly. A slow interface would severely impact the user's experience.

Colours, fonts, and layout should be designed in such a way that it is accessible to all users. The contrast and size of text needs to be suitable for people with poor eyesight. In a school setting, it is essential that all students can use the program. It is also important that the design of the program takes into account cultural differences in views and perspectives, especially on a topic where death, survival, and reproduction will be involved.

It is important for the user that the interface is easy to navigate. For use in the classroom, both the user and client need to be able to quickly work out what the program can do and how to do it. This enables them to be productive whilst using the software and so make efficient use of their time.

The user's productivity and comfort with the software also depends on its stability. The program should crash as little as possible, and if it does the user needs a suitable message explaining the issue. This avoids the feeling that the program crashes 'randomly' and helps them to learn the program. Features of the program must also be consistent and reliable. A colour, symbol, or shape that performs one action in a certain context must consistently perform the same or similar action in all contexts.

Both the client and user need the new solution to integrate with the current solution, to allow gaps in both to be filled and also to allow for future developments in both solutions and the course material.

It should be assumed that the user may make mistakes, so permanent changes like closing the program should come with a confirmation message or undo button. This combines with the user's need for reliability and stability outlined above.

1.7 Hardware Requirements

My final program will be an executable .exe file, so will only run on a Windows machine. Windows 7+ and a modern system would be needed, although the program will not be too intensive. Exact information on hardware requirements is hard to determine, but will be approximately:

- A 1GHz 32-bit or 64-bit processor
- At least 4 GB memory
- A DirectX 10 or higher graphics device
- 1 GB free storage

From: https://docs.unity3d.com/Manual/system-requirements.html.

1.8 Research

1.8.1 Alternative Systems

MinuteLab's Evolution Simulator

This simulator simulates simple organisms with a speed, size, and sense range attributes. The organisms search for food and reproduce or survive based on how efficiently they can do this.[3]

[3] https://labs.minutelabs.io/evolution-simulator/#/s/1/

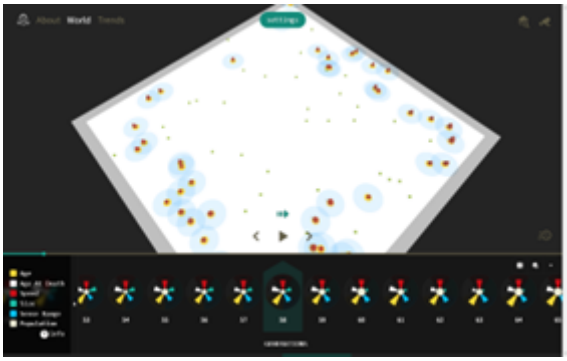

Figure 1.4: Screenshot of the MinuteLab Evolution Simulator

Cary Huang's Evolution Simulator

This simulator evolves creatures made of nodes connected by muscles. This creates a granular approach to demonstrating evolution, where the shape of the creatures changes instead of broad attributes like speed.[4]

Figure 1.5: Screenshot of Cary Huang's Evolution Simulator

PhET Evolution Simulator

This is the most basic of the alternative systems but represents the real world most accurately. Rabbits with variable fur colour, ear size, and teeth size compete for food and avoid predators.[5]

Figure 1.6: Screenshot of the PhET Evolution Simulator

[4] https://openprocessing.org/sketch/205807
[5] https://phet.colorado.edu/en/simulations/legacy/natural-selection/:simulation

1.9 Proposed Solution

My proposed solution is a program that will demonstrate evolution by natural selection. This will be done by simulating the evolution of simple organisms in an environment over a number of generations.

Instead of directly simulating the physical movements of the organisms in their environment I have chosen to approximate this behaviour using algorithms that can be quickly computed. This is a trade-off between the speed that the simulation can be run at and the 'accuracy' of the simulation, but I think that it is worthwhile. The physical movement of the organisms will still be demonstrated visually, just this behaviour will have no effect on their evolution.

Organisms have a number of attributes whose value affects their fitness, which in turn affects their ability to survive and reproduce, driving the evolution process. Data about these attributes will be available to the user, in such a way so that they can easily understand what is going on.

1.9.1 Technology

The program will be written using Unity which uses C# as a scripting language. This allows for easy 3D graphics and user interfaces and is a tool that I am familiar with.

1.9.2 Attributes

An attribute is represented as a struct with a:

- Name: A string
- Benefit Weight: A double from -1 to 1
- Cost Weight: A double from 0 to 1

Each organism has an array of doubles from negative one to one representing the value of each attribute for that organism. This is the value that will be changed through the selection process, progressively becoming closer an optimal value.

The attribute name is only to inform the user what the attribute value aims to represent. The weights affect how the attribute's value influences the organism's fitness. The benefit weight indicates how increasing this value modifies the organism's fitness, and the cost weight indicates the negative impact of increasing this value. These two functions allow for a complex relationship between increasing the attribute value and the organism's fitness, aiming to mimic the real world. The actual functions are described later.

The attributes are hard coded into the program, which will be done in such a way to make adding or modifying the attribute array simple and flexible.

The attributes agreed with the client are:

- Strength
- Intelligence
- Agility
- Endurance
- Disease Susceptibility

Weights

Both of an attribute's weights are predetermined. The exact weight values can only be determined once the program has been written but can be approximated beforehand.

1.9.3 Fitness Function

The attribute-fitness function is the most important part of the program. It aims to answer the question 'Given some attribute value, what is its impact on the organism's chance of survival?'. This is broken down into two parts: the benefit function and the cost function.

Assume the following variables:

v is the attribute value

w_b is the benefit weight

w_c is the cost weight

And the following functions:

$f(x)$ is the attribute-fitness function

$b(x)$ is the benefit function

$c(x)$ is the cost function

Then the functions can be defined as:

$$b(v) = w_b v \tag{1.1}$$
$$c(v) = w_c v^2 \tag{1.2}$$
$$f(v) = \tanh(b(v) - c(v)) \tag{1.3}$$

This ensures that:

- The benefit function will lead to higher or lower values being favoured, depending on the sign of w_b. A larger (in magnitude) w_b increases the strength of this relationship. A value of one makes a high value for this attribute very beneficial, and a value of negative one very damaging to its fitness.
- The cost function quadratically decreases or increases the cost of extreme attribute values depending on the sign of w_c. A larger w_c makes extreme values much more costly. A value of zero shows that there is no cost associated with increasing the value of this attribute.
- The overall attribute-fitness function takes into account both the cost and benefit functions. The tanh acts as an activation function to keep values between negative one and one and ensures that values do not become extreme. tanh is often used as an activation function like this; especially for neural networks, upon which this maths is based.

Taking some values that could represent the weights for the strength attribute and plotting the fitness this produces against the attribute value gives figure 1.7[6]. Here, $f(x)$ is shown in green, and $b(x)$ and $c(x)$ in blue and red respectively. This shows how increasing the value of strength does at first increase the organism's fitness, but after $x = 0.75$ the organism's fitness begins to decrease. This could be explained as the organism's size due to its high strength decreasing its ability to find food and survive.

1.9.4 Initialisation

Population Generation

Before the organisms can be simulated, an initial population must be generated. Each attribute value of each organism will be initialised as random values from the normal distribution:

$$\begin{gathered}\mu = 0,\ \sigma = \frac{1}{\sqrt{2\pi}} \approx 0.4 \\ X \sim \mathcal{N}(\mu, \sigma)\end{gathered} \tag{1.4}$$

This ensures that most (98.8%) of the values generated are in the range negative one to one, with most being close to zero. This represents a relatively homogeneous initial gene pool with only a few outliers to drive significant change.

Organisms are also given an age value that starts at one and decreases by some factor (between zero and one) set by the user each generation.

[6]The graph can be interacted with at https://www.desmos.com/calculator/jugwpdyrus

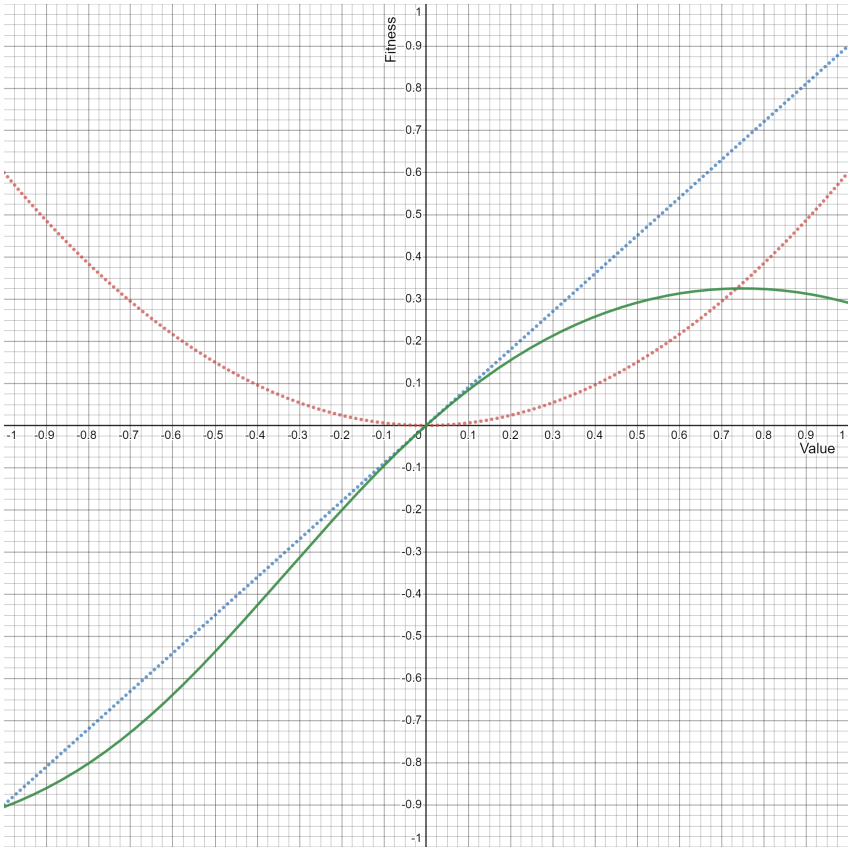

Figure 1.7: Fitness function with $w_b = 0.9$, $w_c = 0.6$.

1.9.5 Generational Evolution

After the initial population of organisms has been created, generations consist of two stages.

Selection

The survival stage is where the organisms live in their environment. They use their given, initially random, attributes to compete for food and evade predators. Although the organisms will be shown exploring their environment and finding food visually based on these attributes, their actual survival chance is computed using the attribute-fitness function described above.

An organism's overall fitness is the average of its attribute fitnesses multiplied by its age value. The fitness of an organism equates to its chance of reproduction using the following algorithm:

1. Every organism in the population calculates its fitness.
2. The fitness of each organism is then mapped from the range of min-fitness to max-fitness, to a range of zero to one in order to produce a normalised fitness value. This means that the fittest organism has a normalised fitness of one, and the least fit a normalised fitness of zero. We can call this value $P(R)$.
3. For each organism, a random (linear) value from zero to one is generated.
4. The random value is compared to the normalised fitness value: If the random value is less than or equal to the normalised value, then the organism proceeds to the next generation.

This ensures that the organism with the highest fitness score will always reproduce ($P(R) = 1$), and the organism with the lowest fitness score will never reproduce ($P(R) = 0$), with the rest of the R values being interpolated between these based on their fitness score.

If an odd number of organisms are selected to reproduce, the organism with the lowest fitness score will be removed from the generation.

Reproduction

Once the group of organisms to reproduce are selected, they can reproduce. Reproduction produces x new organisms, where x is determined by:

$$x = \lceil \frac{f_A + f_B}{2} \times o_{max} \rceil \tag{1.5}$$

Where:

x is the number of new organisms produced

f_A is the fitness of organism A

f_B is the fitness of organism B

o_{max} is the maximum number of organisms that can be produced, configured by the user

Each of the attributes of a new organism is found by:

$$a_{new} = \frac{a_A + a_B}{2} + \text{rand}(X) \tag{1.6}$$

Where:

a_{new} is the new attribute value

a_A is the attribute value of organism A

a_B is the attribute value of organism B

X is the normal distribution $X \sim \mathcal{N}(0, m)$, where m is the mutation strength determined by the user

The new organisms can then be added to the population.

Organisms surviving from the previous generation have their age value multiplied by the ageing factor and new organisms have theirs set to one.

1.9.6 Output

As the program runs, the user will be able to see an individual organism's attributes, as well as graphs showing the average value of the population's attributes and fitness scores over time. The size of the population will also be shown, and this will be the main metric that the user needs as it best represents the real-world evolution process.

By default, generations will advance once every second, giving the user time to see changes as they take place. Initial experimentation with prototypes shows that it generally takes only a few hundred generations for significant changes to be seen.

For more details on individual attributes, the user will have the option to view a graph of an attribute over time, with the attribute being viewed selected from a drop-down list.

The data always on-screen will be:

- The current generation. It would be possible for the generation to be converted with a simple formula to an approximate value of 'years since the start of the simulation', which would be more useful to the user than an arbitrary 'generation' value.
- The current global population. This could be accompanied by a small graph or even just a symbol to show whether the population is increasing, decreasing, or staying roughly the same.
- Data that are associated with the current population, for example all-time max population and a rolling average.

1.10 Objectives

1.10.1 Overview

1. The user is first presented with a series of menus they can navigate.
2. Starting the simulation presents the user with an environment populated by simulated organisms.
3. The organisms can be seen to change over time.
4. The user can see graphs of data about the organisms.

1.10.2 Pages

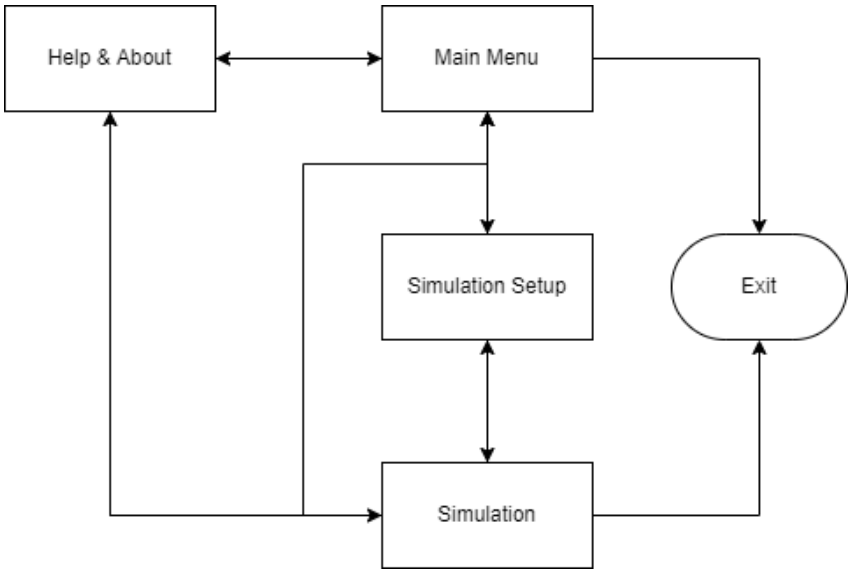

Figure 1.8: Page Flow Diagram

1. Users should be able to navigate the pages of the program as shown in figure 1.8.
2. Main Menu. The main menu is the first page the user sees when they run the program. There should be:
   (a) The name of the program.
   (b) A button to go to the 'Help & About' page.
   (c) A button to go to the 'Simulation Setup' page.
3. Simulation Setup. The simulation setup page allows the user to configure the simulation before it begins.
   (a) There should be a text box for the user to enter a name for the organism 'they' will evolve. This could be defaulted to a randomly generated name.
   (b) Present options to configure the simulation with sliders. There should be a slider for:
      i. Maximum Generations
      ii. Mutation Rate
      iii. Initial Population Size
      iv. Ageing Factor
   (c) Sliders have minimum and maximum values that constrain the input to valid inputs only.
   (d) Sliders allow the user to select only correct values, for instance integers for max generations, but floats for mutation rate.
   (e) Present options to configure the simulation with check boxes. There should be check boxes for:
      i. Enabling attribute costs
      ii. Enabling ageing
   (f) A button to reset all options to defaults.
   (g) A button to begin the simulation.
   (h) Settings should have subtext explaining what the option does.
4. Help & About
   (a) Paragraphs explaining:

i. What the program does.
ii. How to use the program.

(b) A button to go back to the previous page ('Main Menu' or the simulation).

1.10.3 Simulation Page

1. Moving the Camera. As the simulation runs the user will be able to move the camera around the scene to view the organisms as they evolve.
   (a) `WASD` Keys to translate the camera relative to its local rotation.
   (b) Mouse to rotate the camera.
   (c) `Q` and `E` to move the camera up and down respectively.
   (d) `Tab` to move the camera to the next organism.
   (e) Left Click to snap the camera to follow an organism. Using any controls whilst snapped will stop following the organism.
2. Simulation Controls.
   (a) Play/Pause buttons or `Space` to stop or resume the simulation.
   (b) A step button to do one generation.
   (c) A jump button to jump ahead ten generations.
3. Graphs.
   (a) There is a button to view graphs of different data.
   (b) The user can select which data to view on the y-axis. Options are:
      i. Total population.
      ii. Population change.
      iii. The value of attributes across the population. When selected, the user can add to the graph:
         A. Average value across the population.
         B. Q_1, Q_2, and Q_3.
         C. Ten-generation (linear) rolling average.
   (c) The x-axis always shows time in generations.
   (d) The user can select whether to view:
      i. From generation zero onwards.
      ii. The last ten generations.
      iii. The last fifty generations.
   (e) The graph should be a line graph.
   (f) The graph's axes should allow all the selected data points to be shown.
   (g) A logarithmic scale should be used to show population.
   (h) Users should be able to save a graph as an image.

1.10.4 Aesthetics

Accessibility rules are based on the W3's guidelines for web accessibility[7]. In brief, components are considered 'accessible' or 'readable' if:

1. They have a contrast ratio of 4.5:1.
2. Colour is not used as the only distinguishing factor between objects.

These objectives are based on the Material Design guidelines[8].

[7] https://www.w3.org/WAI/tips/designing/
[8] https://material.io/design

1. Fonts.
   (a) All fonts should be either 'Roboto' for text or 'Roboto Mono' for numbers or more technical information.
   (b) Font Sizes should be:
      i. 64pts for level 1 headings
      ii. 48pts for level 2 headings
      iii. 16pts for subtitles
      iv. 14pts for body text
   (c) Button text should be all caps, all other text should be sentence case.
2. Colours.
   (a) The interface should use a colour palette of two colours:
      i. A primary colour to highlight parts of the user interface
      ii. A complementary secondary colour used sparingly to accent key interactions or information, like buttons or progress bars
   (b) The colours selected must contrast against each other and against either black or white text.
   (c) Colour should only be used to draw the user's attention to important areas of the screen.
3. Layout.
   (a) The layout has consistent use of position, size, padding.
   (b) Components must appear predictably.
      i. During setup screens, 'Next' and 'Back' buttons should always appear in the same place
      ii. Data and control windows should be in fixed positions
4. Components. Atomic parts of the user interface that are reused for consistency.
   (a) All components should be:
      i. Familiar. Components must look similar to what the user expects.
      ii. Scannable. The state, type, and position of components should be easily identifiable.
      iii. Clear. The action that a component takes, or purpose it serves, should be obvious to the user.
   (b) Buttons.
      i. Buttons have a fill colour.
      ii. Button text must be easily readable.
      iii. Buttons have a drop shadow.
      iv. Buttons trigger an immediate action.
   (c) Checkboxes.
      i. Checkboxes allow a user to select multiple items in a list.
      ii. The name and category of the checkbox list should be clear.
   (d) Lists.
      i. Lists should be ordered in a logical format.
      ii. Each item in a list must be consistent.
   (e) Progress Indicators.
      i. Progress indicators must be animated.
      ii. Determinate indicators should be used where possible.
      iii. If determinate indicators cannot be used, circular indeterminate indicators should be used.
   (f) Sliders.
      i. Sliders should have immediate effect.
      ii. Sliders should show their numerical value.
      iii. Sliders should present the full range of available values.

1.10.5 Attributes

1. The program stores an array of attributes.
2. Attributes are represented as a struct that has:
   (a) A string name.
   (b) A double benefit weight.
   (c) A double cost weight.
   (d) An attribute fitness function that returns a double and takes no arguments. The algorithm this function must follow is explained later.
3. Attributes coded into the program include at least:
   (a) Strength.
   (b) Intelligence.
   (c) Agility.
   (d) Endurance.
   (e) Disease Susceptibility.
4. Attribute weights are found through testing.
5. Attribute weights produce results similar to what would be expected in the real world.

1.10.6 Organisms

1. The program has a vector of organisms.
2. Organisms are represented as a class.
3. The organism class has:
   (a) A HashMap of attribute names (strings) to attribute values (doubles).
   (b) A double age.
   (c) A fitness function that returns a double and takes no arguments. The algorithm this function must follow is explained later.

1.10.7 Fitness Functions

1. The attribute benefit function $b(v) = w_b v$ is correctly implemented.
2. The attribute cost function $c(v) = w_c v^2$ is correctly implemented.
3. The attribute fitness function $f(v) = \tanh(b(v) - c(v))$ is correctly implemented.
4. If the user disabled attribute costs, then $c(v)$ should not be subtracted (it can be set to zero before the attribute fitness is calculated).
5. The organism fitness function is correctly implemented:
   (a) Compute the attribute fitness of each of an organism's attributes.
   (b) Sum them and find the average.
   (c) Multiply the average by the organism's age modifier if the user chose to enable ageing.
   (d) Return this value.

1.10.8 Evolution Simulation

1. Initialisation happens once at the start of the simulation.
   (a) An organism population of the size set by the user is generated.
   (b) Organism's attributes are set to random values following the normal distribution described in equation 1.4.
   (c) Organism's age value is set to one.
2. Selection occurs at the end of a generation.
   (a) Each organism calculates its fitness.
   (b) The fitness of each organism is normalised by mapping it from the range of fitnesses to the range zero to one.
   (c) The normalised fitness is used as a probability that the organism will survive.
   (d) An even number of organisms are always selected to reproduce.
   (e) The most and least fit organisms always and never, respectively, reproduce (when more than one organism is selected, so as not to violate the previous rule).
3. Reproduction occurs after selection to produce the population for the next generation.
   (a) The number of organisms produced is correctly determined using equation 1.5.
   (b) The attributes of the child organisms are set according to the evolution algorithm; for each attribute:
      i. The average of the parent's attribute values is found.
      ii. A random mutation value is found from a normal distribution based on the user's set mutation strength value.
      iii. The mutation is added to the parent's average and used as the child's value.
   (c) Once organisms have reproduced their age value is multiplied by the user-set ageing factor.
4. Evolutions are computed as fast as possible up to a hundred generations ahead, and the resulting organisms stored in a queue.

1.10.9 Evolution Visualisation

1. An environment is generated for the organisms to inhabit.
   (a) A section of terrain modulated with Perlin noise.
   (b) Details like trees, rocks, and other features are placed procedurally on the terrain.
2. Organisms are shown in the environment.
3. The organism's behaviour reflects their attribute values.

Part 2

Design

2.1 Overall System Design

I will create my program using Unity, which uses C# as a scripting language. This will be run on relatively slow hardware, and so aims to have low minimum requirements of around 4GB of RAM and a CPU from the last decade.

I am building my code with an Object-Orientated approach. This will work well with Unity, which uses an entity component system based around the `MonoBehaviour` class. Classes that inherit from this can be attached to GameObjects in the scene, and override methods like `Start` and `Update` that are run when the program starts and for every frame respectively.

The program is split into three sections:

- The UI, dealing with the main menu, simulation setup, and about pages.
- The Evolution Simulation, dealing with the actual evolution functions. This includes technical skills like complex mathematical models, user defined algorithms, and advanced data types.
- Data processing, dealing with storing data as it is generated, and creating and displaying graphs when they are requested. Merge sort and complex user defined algorithms are used in this area.

From the simulation setup page, users provide inputs that construct an Evolution Configuration object. This object is then passed around the program where needed. As the simulation runs, users can use buttons to view graphs and modify the speed of the generation. Keyboard and mouse controls allow them to move the camera around the environment.

These inputs are processed by various functions, some of which access storage like the function to produce graphs. The processed inputs are used to generate organisms using the evolution algorithms, display data as various graphs, and move organisms around their environment.

2.1.1 Inputs

1. Camera controls:
   (a) `WASD`, `Shift`, and `Space` to translate the camera
   (b) Mouse to rotate the camera
2. Simulation Controls:
   (a) Start or Stop the simulation
   (b) Do one or ten generations quickly
   (c) View graphs
3. Evolution Configuration set by user
4. Preset Simulation Configuration

2.1.2 Processes

1. Camera transformed according to user inputs
2. Evolution simulation
3. Graphs are generated

2.1.3 Storage

1. Organism data is saved to a CSV file so that it can easily be used by both the program and processed by the user themselves, for instance in Excel.
2. Graphs saved as images

2.1.4 Outputs

1. User views simulation from camera perspective
2. Organisms move around their environment with properties based on their attributes
3. Graphs are displayed

2.2 Technical Skills Summary

2.2.1 Complex Mathematical Models

The evolution model is implemented using three key functions:

1. The fitness function (equation 1.1) that calculates the fitness of an organism.
2. The offspring function (equation 1.5) that calculates the number of offspring produced.
3. The reproduction function (equation 1.6) that calculates a new organism's attribute values.

These functions allow for the actual evolution. Fitness is used to decide which organisms will reproduce and survive, and the offspring and reproduction functions determine how many offspring will be produced and what their attributes will be.

This is implemented in the `EvolutionManager` class. Using mathematical functions instead of some kind of physical simulation massively simplifies the process and makes it much more efficient to run.

2.2.2 User-defined Algorithms

In order to simulate evolution, a complex algorithm is used to generate the next generation given the current one. This uses the above mathematical models to calculate whether an organism survives, and if so how many offspring it will produce.

This allows the simulation to be done in advance, as generations can be created before they are needed on a separate thread and added to a queue. Doing this has two key benefits: it decreases the delay when the next generation is needed as the next generation is already being generated as the user is viewing the current generation, and it allows the user to jump forward a number of generations quickly, as up to 100 generations ahead might already be available.

2.2.3 Merge Sort & Recursive Functions

My merge sort implementation uses recursion to efficiently sort a list of organisms. This enables different data to be generated, like histograms of fitness.

Merge sort is used because it is fast, even though the number of organisms that need to be sorted will normally be less than a thousand it is still best for the user for there to be little delay when generating graphs.

2.2.4 Advanced Data Types

All the organisms of a generation are stored in a list, as the number of organisms that will be in each generation cannot be determined. Although a large array could be used in order to avoid the list having to reallocate, this does not pose so much of a cost to performance that would warrant the increase in complexity. The built-in C# dynamic list interface allows you to reserve some number of elements when the array is initialised, so I reserve an estimate of how many organisms will be added in order to reduce the number of reallocations.

A queue is used to allow the generation thread to generate generations asynchronously and push them to the queue where they can then be popped by the simulation thread as needed.

2.3 Data Flow Diagram

See figure 2.1. This diagram shows how the program uses and transfers data.

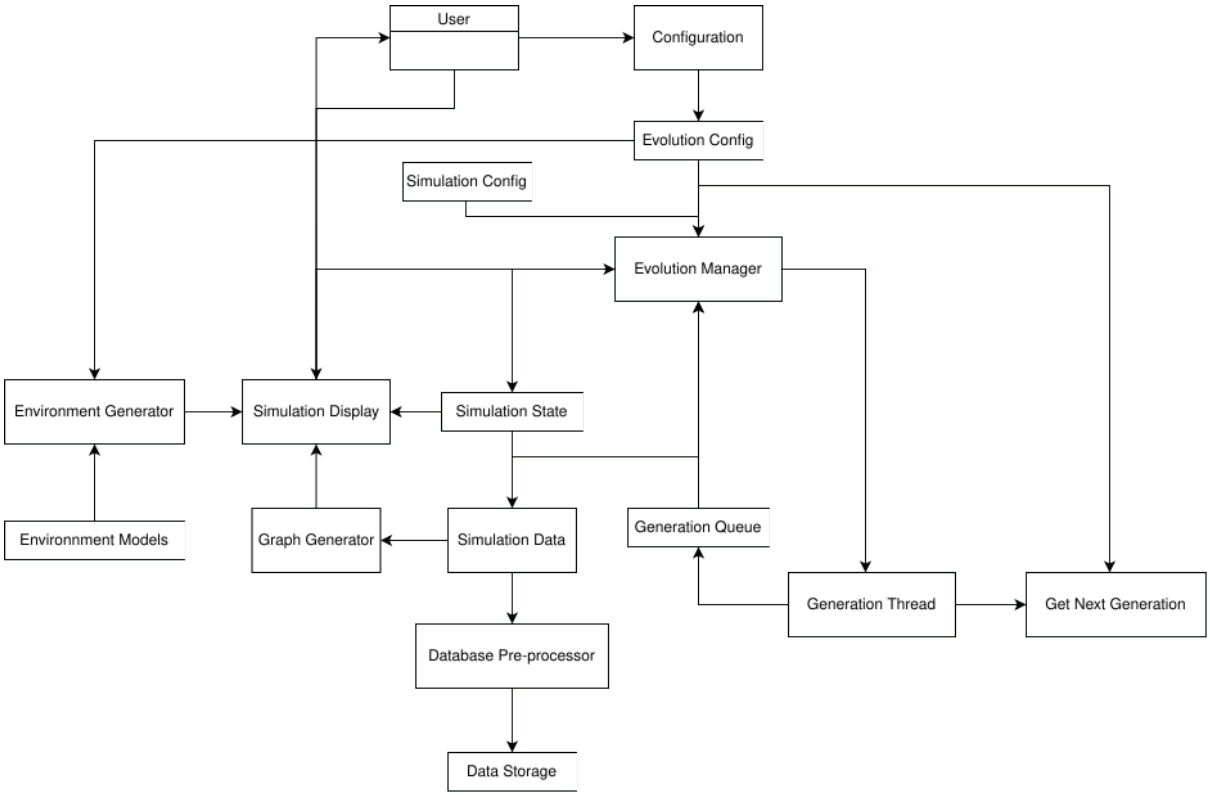

Figure 2.1: Data flow diagram

2.4 UI Flow Diagram

See figure 2.2. This diagram shows how users can move between screens. Note that from the Help & About screen, users will only be able to go back to the previous page, not to either the Main Menu or Simulation page as the diagram could suggest.

2.5 Class Diagram

See figure 2.3. This diagram shows the structure of key classes used in my program and the dependencies between them. Note that `MonoBehaviour`, from which many classes inherit, is a class provided by the Unity API that allows classes to be attatched to a `GameObject` and provides methods like `Start()`, called when the program starts, and `Update()`, called every frame.

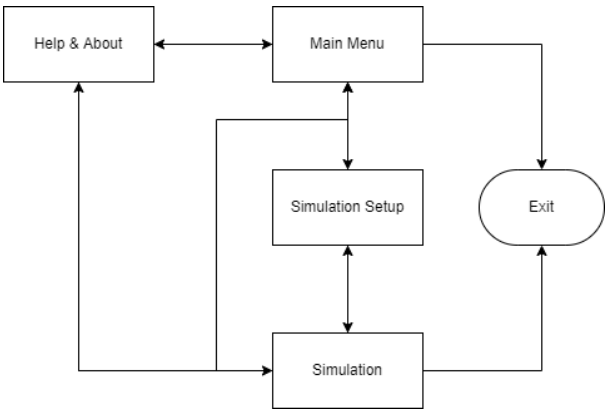

Figure 2.2: UI flow diagram

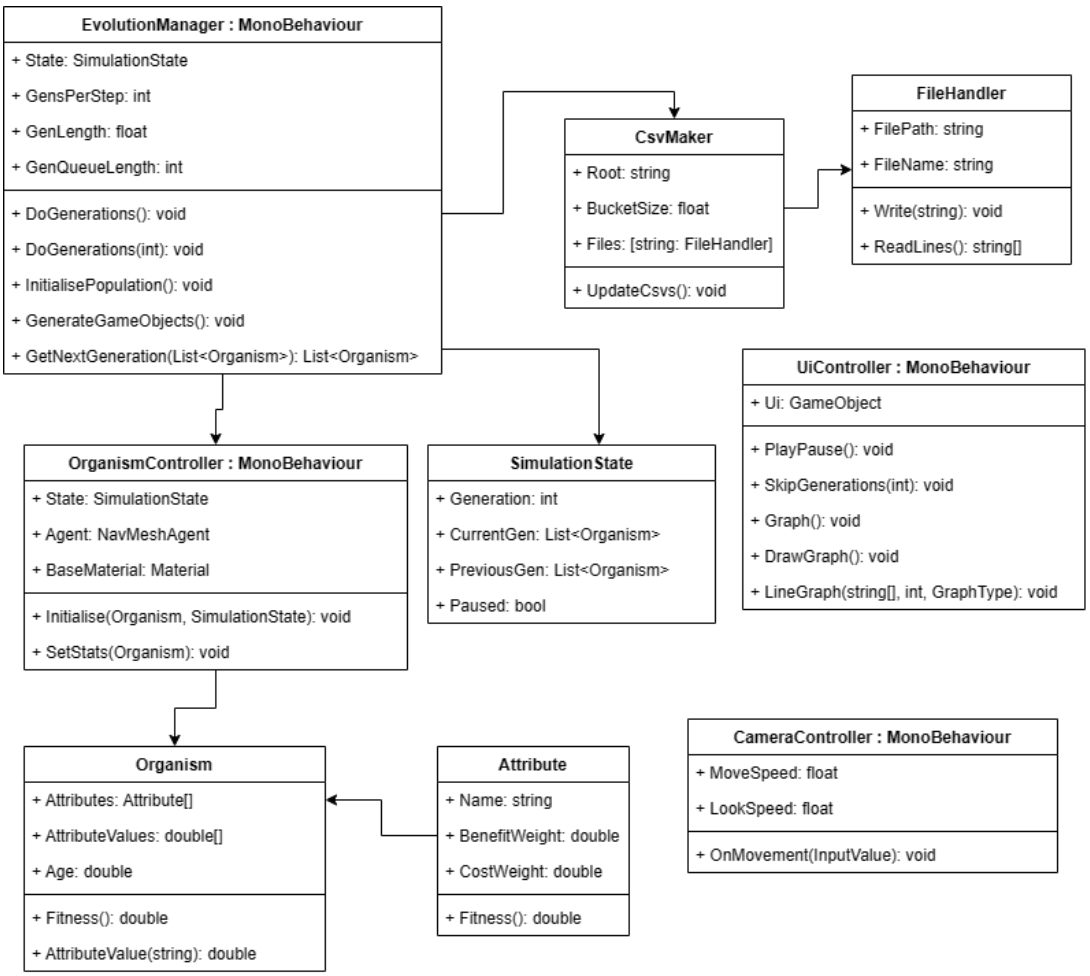

Figure 2.3: Class Diagram

2.6 Class Summaries

2.6.1 Evolution Manager

The evolution manager class performs the actual evolutionary algorithms. It therefore contains the list of the current generation of organisms, as well as a queue of upcoming generations. It displays the generations as needed, creating relevant organisms in the environment with appropriate attributes.

2.6.2 Attribute

The attribute class contains an attribute name, benefit weight, and cost weight. It importantly does not contain a value, as the attribute class represents only an attribute that exists in the simulation. Organisms maintain their own list of attribute values.

2.6.3 Organism

The organism class stores all the data needed to represent an organism on the mathematical side of the simulation. This includes a static list of attributes, a list of their values, and an age. It has a public fitness method that returns the fitness of the organism.

2.6.4 Organism Controller

The organism controller handles the visual representation of some organism. Organism controllers are attached to Unity GameObjects that are displayed to the user. The controller manages the representation of the organism by adjusting data like its movement speed based on attributes of the organism.

2.6.5 Camera Controller

The camera controller allows the user to move the camera around the simulation using a keyboard and mouse. It detects these inputs using Unity's input system and uses vectors and quaternions to transform the camera.

2.6.6 File Handler

The file handler class provides an abstraction to a file pointer so that other areas of the program are simpler. It provides only a simple write method, as reading from files is never needed.

2.6.7 CSV Handler

The CSV Handler maintains the CSV files of statistics. It takes some simulation state as input and writes appropriate data to files using file handlers.

2.6.8 Simulation State

The simulation state class stores various data on the simulation so that it can easily be passed around the program.

2.6.9 Evolution Config

The evolution config class contains the settings set by the user in the configuration screen. It is used by the evolution manager to control various parameters of the simulation.

2.7 Class Property & Method Design

2.7.1 FileHandler.cs

`public class FileHandler`

Provides a simple write-only interface to a file.

`public void Write(string value, bool append = true)`

Writes some text to the file being handled.
Argument 'value': The string to write.
Argument 'append': Whether text should be appended, or the file overwritten with new data.

2.7.2 EvolutionConfig.cs

`[CreateAssetMenu(fileName = "Evolution Config", menuName = "ScriptableObject/EvolutionConfig")]`
Stores configuration data on evolution parameters.

2.7.3 SimulationState.cs

`public class SimulationState : ScriptableObject`
Stores data on the current state of the evolution simulation.

2.7.4 EvolutionManager.cs

`public class EvolutionManager : MonoBehaviour`
Runs the evolution simulation.
`private void DoGenerations(int n)`
Runs 'n' generations and generates GameObjects.
Argument 'n': The number of generations to run
`private void InitialisePopulation()`
Initialises the population based on the 'EvolutionConfig'.
`private void DoGenerations()`
Runs generations whenever the generation queue drops below 'GenQueueLength'
`private void GenerateGameObjects()`
Efficiently creates GameObjects corresponding to the current generation. Deactivates unused objects.
`private List<Organism> GetNextGeneration(List<Organism> organisms)`
Runs the generation algorithms on 'organisms'.
Argument 'organisms': Previous generation
Returns: Next generation
`private T PopRandom<T>(List<T> list)`
Pops a random item from a list.
`private Organism Reproduce(Organism a, Organism b)`
Calculates the offspring of two organisms.
Argument 'a': First organism
Argument 'b': Second organism
Returns: Offspring

2.7.5 Organism.cs

`public class Organism`
Represents an Organism for mathematical functions.
`public Organism()`
Constructs an 'Organism' with random (std = 0.4) attribute values.
`public Organism(double[] attributeValues)`
Constructs an 'Organism' with set attribute values.
Argument 'attributeValues': The attribute values to use.
Throws: 'ArgumentException':
`public void DoAgeing()`
Changes the organism's age by it's ageing factor.
`public double Fitness`
Calculates the Fitness of the organism.
`public double AttributeValue(string name)`
Get an Attribute's value from its name.
Argument 'name': The name of the Attribute to return.
Returns: An Attribute value in the range [-1:1].
Throws: 'ArgumentException':
`public double Fitness(double value)`

Calculate this Attribute's fitness contribution based on some value.
Argument 'value': The value to use in the range [-1:1].
Returns: This Attribute's fitness given 'value' in the range [-1:1].

2.7.6 OrganismController.cs

`public void Initialise(Organism organism, SimulationState state)`
Initialise this 'OrganismController' based on some 'Organism'.
Argument 'organism': The 'Organism' to extract values from.
`public void SetStats(Organism organism)`
Update the attributes of the 'Organism' this controller is based off.
Argument 'organism': The 'Organism' to extract values from.

2.7.7 SimulationSetupController.cs

`public class SimulationSetupController : MonoBehaviour`
Controls the SimulationSetup scene.
`public void Defaults()`
Reset values to defaults (overwrites with a new 'EvolutionConfig').

2.7.8 StaticMenuController.cs

`public class StaticMenuController : MonoBehaviour`
Scene controller that exists across scene changes so that persistent data can be stored.
`public void LoadScene(string name)`
Loads scene 'name'.
Argument 'name': String name of the Scene to load.
`public void Back()`
Returns to the previous scene.

2.7.9 StaticMenuControllerHandle.cs

`public class StaticMenuControllerHandle : MonoBehaviour`
Provides an interface to a 'StaticMenuController', allowing the static controller to exist between scenes.
`public string StaticObjectName = "Static";`
Name of the 'GameObject' with a 'StaticMenuController' that this script should use.
`public void LoadScene(string name)`
Loads scene 'name'.
Argument 'name': String name of the Scene to load.
`public void Back()`
Return to the previous scene.

2.7.10 CameraController.cs

`public class CameraController : MonoBehaviour`
Handles camera movement from keyboard inputs.

2.8 Important Algorithms

2.8.1 Generating Next Generation

This is the most important algorithm in the program. It takes a previous generation and returns the next generation, using the algorithms and equations outlined above (equations 1.9.5, 1.6, and 1.5).

```
sub get_next_generation(prev_generation)
    next_generation = []

    // Selection: eq. 1.9,5
```

```
    max_fitness = maximum_fitness(prev_generation)
    min_fitness = minimum_fitness(prev_generation)

    survivors = []

    for organism in prev_generation
        normalised_fit = (organism.fitness - min_fitness) /
                         (maximum_fitness - minimum_fitness)
        if random(0, 1) <= normalised_fitness then
            survivors.append(organism)
        endif
    endfor

    if survivors.length % 2 != 0
        survivors.remove_at(survivors.count - 1)
    endif

    shuffle(survivors)

    while survivors.length > 0
        a = survivors.pop()
        b = survivors.pop()

        for i = 0 to num_offspring(a, b)
            next_generation.append(offspring(a, b))
        next i

        next_generation.append(a, b)
    endwhile

    return next_generation
endsub

// Returns the maximum fitness from a population
sub maximum_fitness(organisms)

// Returns the minimum fitness from a population
sub minimum_fitness(organisms)

// Randomises the order of an array
sub shuffle(array)

// Returns the number of offspring produced by a and b, as per eq. 1.5
sub num_offspring(a, b)

// Returns an offspring of a and b, as per eq. 1.6
sub offspring(a, b)
```

This is an implementation of the following algorithm:

1. Find the maximum and minimum fitness of the population. This is done so that each organism's fitness can later be normalised.
   (a) Iterate through every organism's fitness.
   (b) Compare the fitness value to a stored maximum and minimum value, initialised to -1 and 1 respectively.
   (c) If higher than the maximum, set the maximum to this value.
   (d) If lower than the minimum, set the minimum to this value.

2. Select which organisms will survive this generation using the algorithm in 1.9.5.
   (a) Normalise the fitness value into the range [0:1].
   (b) Compare the normalised fitness to a random value from the range (0:1).
3. If there are an odd number of organisms selected, remove the last to be added (least fit).
4. Reproduce random organism pairs.
   (a) Shuffle the array of survivors.
   (b) Pop two organisms (a and b).
   (c) Produce the required number of offspring (equation 1.5) using equation 1.6, adding each to the next generation.
   (d) Add organisms a and b to the next generation.
   (e) Repeat steps b to d until no organisms remain.
5. Return the next generation.

2.8.2 Organism Fitness Calculation

The organism and attribute classes each contain respective fitness functions. The attribute fitness function is only ever called by an organism in order to calculate its own fitness. This follows from the fitness equation (equation 1.1).

```
class Organism
    public attributes: Attribute[]
    public age: float

    private attribute_values: float[]

    sub fitness()
        total = 0

        for i = 0 to attributes.len
            total += attributes[i].fitness(attribute_values[i])
        next i

        return (total / attributes.length) * age
    endsub
endclass

class Attribute
    public name: string
    public benefitWeight: float
    public costWeight: float

    sub fitness(value)
        benefit = benefitWeight * value
        cost = costWeight * value * value
        return tanh(benefit - cost)
    endsub
```

2.8.3 Camera Transformation

The camera can be moved with the mouse and keyboard. The CameraController class is attached to the camera and allows this to happen. This allows the camera's movement to be consistent even when the frame rate might vary. The `timeDelta` parameter is provided to the `on_render` function called every frame, and represents the time since the last frame.

```
class CameraController
    public moveSpeed, lookSpeed: float

    // timeDelta is the time since the last render call
    sub on_render(timeDelta)
        translation = get_keyboard_input()
        rotation = get_mouse_delta()

        transform.translate(translation * moveSpeed * timeDelta)
        transform.rotate(rotation * lookSpeed * timeDelta)
    endsub
endclass
```

2.8.4 Generating CSV Files

A CSV file needs to be generated for each attribute, so that the average value can be stored across generations. First it is ensured that the folder that contains the CSVs exists, then the header row that will be used for each file is created. This header row allows for a generation column and a column for each of the distribution buckets (the attribute values are not directly stored, as the will only be displayed as a histogram). A file in the CSV folder can then be created, the header row written, and the file closed.

```
sub generate_csv_files(folder, simulation_state, attributes, bucket_size)
    if !system.folder_exists(folder)
        system.create_folder(folder)
    endif

    header = "gen,"
    for bucket = 0 to 1
        header = header + bucket_size + ","
    next i = i + bucket_size
    header = header + "\n"

    files = FileHandler[attributes.len]
    for i = 0 to attributes.len
        files[i] = system.new_file(folder + "\" + attributes[i].name + ".csv")
        files[i].open()
        files[i].write(header)
        files[i].close()
    next i
endsub
```

2.8.5 Generating Organism GameObjects

Unity uses GameObjects to represent physical objects in the world, as opposed to the data representation used by the simulation equations. A GameObject is therefore needed for each organism in the current generation so that it can be displayed. These GameObjects are given an Organism Controller that deals with their movement and appearance. This Organism Controller is given the data of the organism it represents, so that it can configure itself appropriately.

Because creating and destroying GameObjects is expensive and there are a large number involved (>300 after a few generations), GameObjects are only created if more are needed than have already been made. Otherwise, a GameObject from the previous generation is simply given new data. If the number of organisms has decreased since the last generation GameObjects are disabled, making them invisible without destroying them so they can be reused in future.

```
class EvolutionManager
    private organism_objects: List<GameObject>

    sub generate_gameobjects(organisms)
        for i = 0 to organism_objects.len - 1
```

```
            if i < organisms.len
                organism_objects[i].enable()
                organism_objects[i].organism_controller.set(organisms[i])
            else
                organism_objects[i].disable()
            endif
        next i

        for i = organism_objects.len to organisms.len
            new_object = organism_controller.new_object()
            new_object.set(organisms[i])
            organism_objects.push(new_object)
        next i
    endsub
endclass
```

2.8.6 Merge Sort

Merge sort is used to sort the organisms based on their attributes in order to find statistics like percentiles. It is a recursive algorithm that sorts an array by first splitting the provided array in half until it is of length one, and then merging and returning these arrays.

```
sub merge_sort(list)
    if list.len == 1
        return list
    endif

    left = []
    right = []

    for i = 0 to list.len - 1
        if i < list.len / 2
            left.push(list[i])
        else
            right.push(list[i])
        endif
    endfor

    left = merge_sort(left)
    right = merge_sort(right)

    return left + right
endsub
```

2.9 User Interface

2.9.1 Main Menu

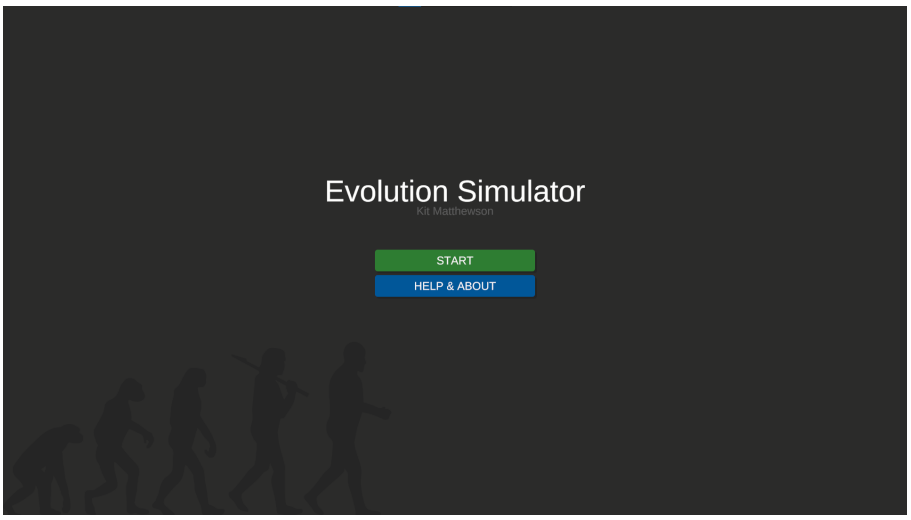

Figure 2.4: Main Menu Screenshot

- Buttons to:
  - Continue to the simulation setup page
  - View some information about the program
  - Exit the program
- The name of the program

2.9.2 Help & About

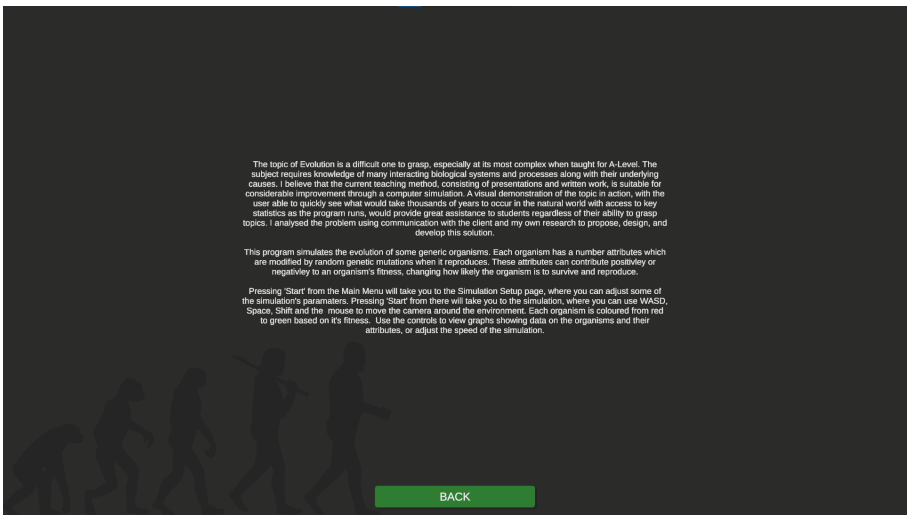

Figure 2.5: Help & About Screenshot

- Information on how to use the program
- Some text explaining what the program does
- A button to return to the main menu

2.9.3 Simulation Setup

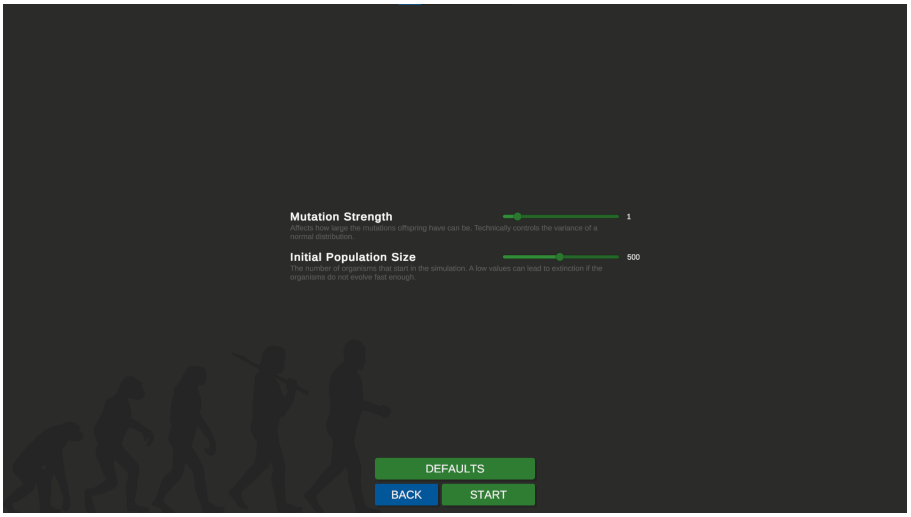

Figure 2.6: Simulation Setup Screenshot

- Sliders, buttons and inputs to configure the simulation
- One line of text explaining what each option does
- Button to:
  - Begin the simulation
  - Reset all the options to their defaults
  - Return to the main menu

2.9.4 Simulation

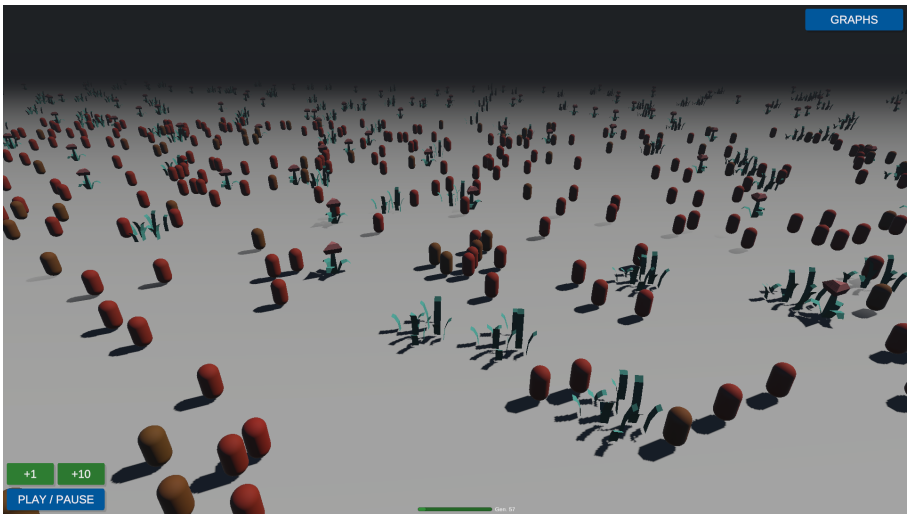

Figure 2.7: Simulation Screenshot

- The visualisation of the simulation
- Generation controls to:
  - Start or stop the simulation
  - Do one generation

  - Do ten generations

- A button to open the graph menu

2.10 Testing

Testing video: https://youtu.be/fQAddzm0Io4

Figure 2.8: QR Code to testing video

These tests are based on the objectives from section 1.10.

No.	Relevant Objectives	Name	Description	Expected Result	Result
Overview Objectives					
1	.1.1	Menus	Navigate the program using on-screen buttons.	The buttons allow you to move through all the various screens.	Pass
2	.1.2	Simulation	Start the simulation.	A number of organisms are shown moving around their environment.	Pass
3	.1.3	Progression	Watch the simulation progress.	The organisms visually change over time.	Pass
4	.1.4	Data	Look at the data presented and use on-screen buttons to view graphs.	Various data can be seen, demonstrating that evolution is taking place.	Pass
Pages					
5	.2.1	Navigation	Press the 'About' button from the main menu screen.	Moves to the about page.	Pass
6			Press the 'Start' button on the main menu screen.	Moves to the simulation setup page.	Pass
7			Press the 'Start' button on the simulation setup screen.	Moves to the simulation screen.	Pass
8			Press the 'Back' button on the simulation setup screen.	Moves to the main menu.	Pass
9			Press the 'About' button from the simulation screen.	Moves to the about page.	Pass
10			Press the 'Back' button on the about page, after previously being on the main menu.	Moves back to the main menu.	Pass
11			Press the 'Back' button on the about page, after previously being on the simulation screen.	Moves back to the simulation screen.	Pass
12	.2.2	Pages	Run the program and look at the main menu screen.	The has the name of the program, and buttons to go to the about page and simulation setup pages.	Pass
13	.2.3		Run the program and navigate to the simulation setup screen.	The page has the controls outlined in .1.3.a, .b, and .e, and has the input validation outlined in .1.3.c and .1.3.d.	Pass

14	.2.4.a		Run the program and navigate to the about page.	The page has text explaining what the program does and how to use it.	Pass
	Simulation Page				
15	.3.1	Camera	Press the WASD keys when on the simulation screen.	The camera translates forwards, backwards, left, and right depending on which key is pressed.	Pass
16			Move the mouse when on the simulation screen.	The camera rotates based on the direction that the mouse moves.	Pass
17			Press Space or Shift when on the simulation screen.	The camera moves up when space is pressed, and down when shift is pressed.	Pass
18	.3.2	Controls	Press the pause button on the simulation screen.	The simulation stops running.	Pass
19			Press the play button on the simulation screen.	The simulation resumes running.	Pass
20			Press the 'One generation' button on the simulation screen.	The simulation quickly executes on generation.	Pass
21			Press the ''Ten generations' button on the simulation screen.	The simulation quickly executes ten generations.	Pass
22	.3.3	Graphs	Press 'View graphs' button	Graph visualisation page opens.	Pass
23			Select to view total population.	Graph of total population over time is shown	Pass
24			Select to view change in population.	Graph of change in population is shown.	Pass
25			Select to view an attribute or fitness value.	Graph of attribute's average value over time is shown.	Pass
26			Select to view an attribute or fitness distribution.	Histogram of attribute's distribution is shown.	Pass
27			With an attribute selected, add quartiles.	Lines for Q1, Q2, and Q3 are shown.	Pass
28			With an attribute selected, add a ten-generation rolling average.	A line for the ten-generation rolling average is shown.	Pass
29			Look at the graph's x-axis.	The x-axis is always shows attribute value for attribute distribution graphs, or time for all other graphs.	Pass

30			Look at the graph's y-axis.		The y-axis is a logarithmic number for population graphs, linear value for attribute value graphs, or frequency density for attribute distribution histograms.	Pass
31			Change the graph's time frame.		The timeframe of the graph can be switched between all time, last ten generations, and last fifty generations.	Pass
32			Look at the type of graph.		The graph is be a histogram for attribute distributions, and a line for all other graphs.	Pass
33	.3.1.h	Graph Saving	Press to save graph image.		Selected graph is saved as an image in a suitable folder.	Pass
34			Press to save graph data.		Selected graph's data is saved as a csv file in a suitable folder.	Pass
			Aesthetics			
35	.4.1	Fonts	Look at the fonts used on each screen.		The only fonts used are 'Roboto' and 'Roboto Mono', and in the sizes listed in .3.1.b. Button text is capitalised.	Pass
36	.4.2	Colours	Look at the colours used in the interface on each screen.		The interface uses a primary and secondary colour that contrast against each other and black and white backgrounds.	Pass
37	.4.3	Layout	Look at the layout of items on each screen.		The layout makes use of consistent size and spacing.	Pass
38	.4.4	Layout Components	Look at the components used as part of the interface on each screen.		The interface reuses standard components for buttons, checkboxes, lists, progress indicators, and sliders.	Pass
			Attributes			
39	.5.1	Static Attributes	Review code		The program stores a static array of attributes.	Pass
40	.5.2	Attribute Struct			Attributes have a name, benefit and cost weight, and an attribute fitness function.	Pass
41	.5.3	Attributes			At least the attributes listed in .4.3 are implemented.	Pass

42	.5.4 and .5.5	Attribute Weights	Review graphs for each attribute's value.	The average value for attributes approaches real-world conditions as the simulation runs.	Pass
		Organisms			
43	.6.1	Organism list	Review code	The program stores all the organisms in a dynamic list.	Pass
44	.6.2 and .6.3	Organism class		Organisms have an array of attribute values and an age.	Pass
		Evolution Simulation			
45	.7.1	Initialisation	Start the program with minimum and maximum initial population sizes.	A population of the appropriate size is generated.	Pass
46			Start the program and view an attribute distribution histogram.	The distribution approximates the normal distribution in 1.4.	Pass
47	.7.2	Selection	Use breakpoints to view an organism's attribute values and calculated fitness, and manually calculate the expected fitness value.	The organism correctly calculates its fitness using equation 1.1.	Pass
48			Use breakpoints to view the normalised fitness values as the simulation runs.	Organism's fitness values are normalised from the range of fitness values in the population to [0:1].	Pass
49			Use breakpoints to view the number of organisms that are selecteed.	An even number of organisms is always selected.	Pass
50			Temporarily decrease the initial population size to ten and then use breakpoints to manually calculate which organisms should be selected in a generation.	Organisms are selected according to the algorithm 1.9.5.	Pass
51	.7.3	Reproduction	Use breakpoints to trace the calculation of the number of offspring.	The number of offspring is determined with equation 1.5.	Pass
52			Use breakpoints to trace `EvolutionManager.Reproduce(a, b)` function.	Offspring attributes are found with equation 1.6	Pass

53			Use breakpoints to view an organism's age change over a number of generations.	Age values are increased at the end of each generation.	Pass
54	.7.4	Async Generation	Use breakpoints and Unity's debugging tools to view how and where generations are created.	Generations are created off of the main thread and added to a queue.	Pass
55	.8.2 & .8.3	Visualisation	View an arbitrary simulation.	Organisms are shown in their environment with properties that reflect their attributes.	Pass

Part 3

Evaluation

3.1 Summary

My program correctly simulates evolution in a way that would help biology students understand the processes of natural selection and genetic mutation. Starting with an initial population of organisms with random characteristics

The program is easy to use, with a simple interface and clear controls. It runs smoothly, even on the basic computers that the user has access to, because of the optimisations that I have made. Crashes do not occur, and all user input is validated and received through sliders that make it impossible to enter invalid information.

The hardest part of the project was generating the graphs, as it required saving large amounts of data which then had to be processed in the correct way when the user requested a graph.

3.2 Objective Evaluation

Tests	**Name**	**Objectives**	**Evaluation**
Overview Objectives			
1	Menus	.1.1	Fully met
2	Simulation	.1.2	Fully met
3	Progression	.1.3	Fully met
4	Data	.1.4	Fully met
Pages			
5-11	Navigation	.2.1	Fully met
12-14	Pages	.2.2, .2.4	Fully met
Simulation Page			
15-17	Camera	.3.1	Fully met
18-21	Controls	.3.2	Fully met
22-32	Graphs	.3.3	Fully met
33-34	Graph Saving	.3.1	Could provide a button to easily open the graph folder
Aesthetics			
35	Fonts	.4.1	Exact font sizes were changed, but are still used consistently
36	Colours	.4.2	Fully met
37	Layout	.4.3	Fully met
38	Layout Components	.4.4	Fully met
Attributes			
39	Static Attributes	.5.1	Fully met
40	Attribute Struct	.5.2	Fully met
41	Attributes	.5.3	Fully met
42	Attribute Weights	.5.4, .5.5	Fully met
Organisms			

43	Organism list	.6.1	Fully met
44	Organism class	.6.2, .6.3	Fully met
Evolution Simulation			
45-46	Initialisation	.7.1	Fully met
47-50	Selection	.7.2	Fully met
51-53	Reproduction	.7.3	Fully met
54	Async Generation	.7.4	There is still some unresponsiveness when a new generation is being displayed, so more optimisations could be done
55	Visualisation	.8.2, .8.3	Fully met

3.3 Client Feedback

In order to get feedback on my completed project I sent it to my client, Mr Knowles, by email and asked him to share his views on things he liked and disliked about it.

In his email he said:

What I like about your program is that it provides a fun and interactive way for students to explore the concepts of evolution. By allowing them to experiment with different scenarios and see the results, students can gain a deeper understanding of how evolution works and how it affects different species. I also appreciate that the program is user-friendly and easy for students to navigate, making it accessible for learners of all abilities. Overall, I think your program is a valuable resource for teaching evolution in the classroom.

One potential criticism of your program is that it may not fully capture the complexity and nuance of real-world evolution. While simulations can be a useful tool for teaching basic concepts, they can also oversimplify the process and fail to represent the full range of factors that influence evolution. As a result, students may come away with a limited understanding of the subject. It would be helpful if the program included additional resources or information to help students explore the more complex aspects of evolution. Additionally, it would be beneficial if the program included more diverse examples and scenarios to illustrate the wide range of ways in which evolution can occur.

3.4 Analysis & Improvements

These improvements are based on further conversation with my client as he best understands what my users, his students, need the most. If I had more time, I could have added:

- More in-depth information and resources to help students explore the complex factors that influence evolution. This could include information on different types of selection, genetic drift, and other processes that impact evolution.
- More diverse examples and scenarios to illustrate the wide range of ways in which evolution can occur. This could include examples from different environments, biomes, and species to show how evolution varies in different contexts.
- The ability for users to create and customize their own scenarios to experiment with different factors and see how they affect the outcome of evolution. This could include options for setting initial traits or environmental conditions and other variables to allow students to explore different scenarios and see how they impact evolution.
- Additional interactive features, such as quizzes, games, or puzzles, to help students practice and reinforce the concepts they have learned. These could include activities that help students apply their knowledge to real-world situations or that challenge them to think critically about the processes of evolution.

Overall my client's feedback was positive, and it seems that he thinks I have met the objectives he laid out at the beginning of the project.

Part 4

Code Listings

4.1 File Structure

Files are arranged as follows:

```
Scripts/
|-- Common/
| |-- FileHandler.cs
| `-- RandomNormal.cs
|-- Data/
| |-- CsvMaker.cs
| |-- EvolutionConfig.cs
| `-- SimulationState.cs
|-- Evolution/
| |-- EvolutionManager.cs
| |-- Organism.cs
| `-- OrganismController.cs
|-- UI/
| |-- SimulationController.cs
| |-- SimulationSetupController.cs
| |-- StaticMenuController.cs
| |-- StaticMenuControllerHandle.cs
| `-- UiController.cs
`-- Visuals/
    |-- CameraController.cs
    |-- ItemPlacer.cs
    `-- NoiseGenerator.cs
```

4.2 CameraController.cs

```
1  using JetBrains.Annotations;
2  using UnityEngine;
3  using UnityEngine.InputSystem;
4
5  /// <summary>
6  /// Handles camera movement from keyboard inputs.
7  /// </summary>
8  public class CameraController : MonoBehaviour {
9      public float MoveSpeed;
10     public float LookSpeed;
11
12     private Vector2 _horizontalMovement;
13     private float _verticalMovement;
14     private Vector2 _mouseDelta;
15
16     [PublicAPI]
```

```
17      private void Start() {
18          Cursor.lockState = CursorLockMode.Locked;
19          Cursor.visible = false;
20      }
21
22      [PublicAPI]
23      private void Update() {
24          Vector3 movement = new(_horizontalMovement.x, 0, _horizontalMovement.y);
25          movement += transform.InverseTransformDirection(Vector3.up * _verticalMovement);
26
27          transform.Translate(MoveSpeed * Time.deltaTime * movement);
28          transform.Rotate(LookSpeed * Time.deltaTime * new Vector3(_mouseDelta.y, _mouseDelta.x,
                ↪ 0));
29          transform.rotation = Quaternion.Euler(transform.rotation.eulerAngles.x, transform.
                ↪ rotation.eulerAngles.y, 0);
30      }
31
32      public void OnFocusToggle() {
33          Cursor.visible = !Cursor.visible;
34
35          Cursor.lockState = Cursor.visible ? CursorLockMode.Confined : CursorLockMode.Locked;
36      }
37
38      public void OnHorizontalMovement(InputValue input) {
39          _horizontalMovement = input.Get<Vector2>();
40      }
41
42      public void OnVerticalMovement(InputValue input) {
43          _verticalMovement = input.Get<float>();
44      }
45
46      public void OnLook(InputValue input) {
47          if (Cursor.lockState == CursorLockMode.Locked) {
48              _mouseDelta = input.Get<Vector2>();
49          }
50      }
51  }
```

4.3 CsvMaker.cs

```
1   using System.Collections.Generic;
2   using System.IO;
3   using System.Text;
4   using JetBrains.Annotations;
5   using UnityEngine;
6
7   [RequireComponent(typeof(EvolutionManager))]
8   public class CsvMaker : MonoBehaviour {
9       public const string Root = "output\\csv";
10      public float BucketSize;
11      public int WriteFrequency;
12
13      public readonly Dictionary<string, FileHandler> Files = new();
14      private SimulationState _state;
15
16      private int _lastWrite = -1;
17
18      [PublicAPI]
19      private void Start() {
20          if (!Directory.Exists(Root)) {
21              Directory.CreateDirectory(Root);
22          }
```

```
23
24          _state = gameObject.GetComponent<EvolutionManager>().State;
25
26          Files["Fitness"] = new FileHandler($"{Root}\\Fitness.csv");
27
28          for (int i = 0; i < Organism.Attributes.Length; i++) {
29              string attributeName = Organism.Attributes[i].Name;
30              Files[attributeName] = new FileHandler($"{Root}\\{attributeName}.csv");
31          }
32
33          StringBuilder headerRow = new("gen,");
34
35          for (int i = 0; i < 2 / BucketSize; i++) {
36              headerRow.Append(i * BucketSize - 1);
37
38              if (i < 2 / BucketSize - 1) {
39                  headerRow.Append(",");
40              }
41          }
42
43          headerRow.Append("\n");
44
45          foreach (string key in Files.Keys) {
46              Files[key].Write(headerRow.ToString());
47          }
48      }
49
50      [PublicAPI]
51      private void Update() {
52          if (_state.Generation < _lastWrite + WriteFrequency) return;
53
54          UpdateCsvs();
55          _lastWrite = _state.Generation;
56      }
57
58      public void UpdateCsvs() {
59          for (int i = -1; i < Organism.Attributes.Length; i++) {
60              int[] buckets = new int[(int)(2 / BucketSize)];
61
62              StringBuilder line = new(buckets.Length * 2);
63
64              line.Append(_state.Generation + ",");
65
66              foreach (Organism organism in _state.CurrentGen) {
67                  double value = i == -1 ? organism.Fitness : organism.AttributeValues[i];
68
69                  int bucket = Mathf.FloorToInt((float)((1 + value) / BucketSize));
70                  bucket = Mathf.Clamp(bucket, 0, buckets.Length - 1);
71
72                  buckets[bucket] += 1;
73              }
74
75              foreach (int frequency in buckets) {
76                  line.Append(frequency + ",");
77              }
78
79              line.Append("\n");
80
81              Files[i == -1 ? "Fitness" : Organism.Attributes[i].Name].Write(line.ToString());
82          }
83      }
84  }
```

4.4 EvolutionConfig.cs

```
using UnityEngine;

/// <summary>
/// Stores configuration data on evolution parameters.
/// </summary>
[CreateAssetMenu(fileName = "Evolution Config", menuName = "ScriptableObject/EvolutionConfig")]
public class EvolutionConfig : ScriptableObject {
    public int InitialPopulationSize = 500;
    public int MinimumOffspring = 2;
    public int MaximumOffspring = 5;
    public double MutationStrength = 0.04;
}
```

4.5 EvolutionManager.cs

```
using JetBrains.Annotations;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using TMPro;
using UnityEngine;
using UnityEngine.UI;
using Random = System.Random;

/// <summary>
/// Runs the evolution simulation.
/// </summary>
public class EvolutionManager : MonoBehaviour {
    public StaticMenuControllerHandle MenuControllerHandle;

    public SimulationState State { get; private set; }

    [Header("UI")]
    public Slider GenProgress;
    public TextMeshProUGUI GenText;

    // Evolution config deals with the actual evolution parameters. Simulation config only
        ↪ affects how this is shown.
    [Header("Simulation Config")]
    public GameObject OrganismObject;

    public int GensPerStep = 10;
    public float GenLength = 5;
    public int GenQueueLength = 10;

    [HideInInspector] public int GenerationGoal = -1;

    private EvolutionConfig _config;

    private readonly List<GameObject> _organismObjects = new();

    private float _lastGeneration;

    private readonly Queue<List<Organism>> _generations = new();
    private Thread _generationThread;

    private readonly Random _rnd = new();

    // Use Awake so CSVs can be made in Start
```

```
44      [PublicAPI]
45      private void Awake() {
46          State = ScriptableObject.CreateInstance<SimulationState>();
47      }
48
49      [PublicAPI]
50      private void Start() {
51          _config = MenuControllerHandle.StaticController.EvolutionConfig;
52
53          InitialisePopulation();
54          _lastGeneration = -GenLength;
55
56          lock (_generations) {
57              _generations.Enqueue(State.CurrentGen);
58          }
59
60          _generationThread = new Thread(DoGenerations);
61          _generationThread.Start();
62      }
63
64      [PublicAPI]
65      private void Update() {
66          GenProgress.value = (Time.time - _lastGeneration) / GenLength;
67
68          GenText.text = $"Gen.␣{State.Generation}";
69
70          lock (_generations) {
71              if (GenerationGoal > State.Generation && _generations.Count == GenQueueLength) {
72                  int n = Mathf.Min(_generations.Count, GenerationGoal - State.Generation);
73
74                  if (n > 1) {
75                      DoGenerations(n);
76                  }
77              }
78          }
79
80          if (State.Paused) {
81              _lastGeneration += Time.deltaTime;
82              return;
83          }
84
85          if (Time.time - _lastGeneration >= GenLength) {
86              DoGenerations(GensPerStep);
87          }
88      }
89
90      /// <summary>
91      /// Runs <c>n</c> generations and generates GameObjects.
92      /// </summary>
93      /// <param name="n">The number of generations to run</param>
94      private void DoGenerations(int n) {
95          if (n < 1) {
96              return;
97          }
98
99          lock (_generations) {
100             for (int i = 0; i < n && _generations.Any(); i++) {
101                 State.PreviousGen = State.CurrentGen; // Is this a reference?
102                 State.CurrentGen = _generations.Dequeue();
103                 State.Generation++;
104             }
105         }
106
```

```
107         _lastGeneration = Time.time;
108         GenerateGameObjects();
109     }
110 
111     /// <summary>
112     /// Initialises the population based on the <c>EvolutionConfig</c>.
113     /// </summary>
114     private void InitialisePopulation() {
115         State.CurrentGen = new List<Organism>(_config.InitialPopulationSize);
116         for (int i = 0; i < _config.InitialPopulationSize; i++) {
117             State.CurrentGen.Add(new Organism());
118         }
119     }
120 
121     // ReSharper disable once FunctionNeverReturns
122     /// <summary>
123     /// Runs generations whenever the generation queue drops below <c>GenQueueLength</c>
124     /// </summary>
125     private void DoGenerations() {
126         while (true) {
127             lock (_generations) {
128                 if (_generations.Count >= GenQueueLength) continue;
129 
130                 var prev = _generations.Count == 0 ? State.PreviousGen : _generations.Last();
131 
132                 var next = GetNextGeneration(prev);
133                 _generations.Enqueue(next);
134             }
135         }
136     }
137 
138     /// <summary>
139     /// Efficiently creates GameObjects corresponding to the current generation. Deactivates
        ↪ unused objects.
140     /// </summary>
141     private void GenerateGameObjects() {
142         for (int i = 0; i < _organismObjects.Count; i++) {
143             if (i < State.CurrentGen.Count) {
144                 _organismObjects[i].SetActive(true);
145                 _organismObjects[i].GetComponent<OrganismController>().Initialise(State.
                        ↪ CurrentGen[i], State);
146             } else {
147                 _organismObjects[i].SetActive(false);
148             }
149         }
150 
151         for (int i = _organismObjects.Count; i < State.CurrentGen.Count; i++) {
152             GameObject newOrg = Instantiate(OrganismObject, gameObject.transform);
153             newOrg.GetComponent<OrganismController>().Initialise(State.CurrentGen[i], State);
154             _organismObjects.Add(newOrg);
155         }
156     }
157 
158     /// <summary>
159     /// Runs the generation algorithms on <c>organisms</c>.
160     /// </summary>
161     /// <param name="organisms">Previous generation</param>
162     /// <returns>Next generation</returns>
163     private List<Organism> GetNextGeneration(List<Organism> organisms) {
164         List<Organism> survivors = new(organisms.Count / 2);
165 
166         double minFit = 1, maxFit = -1;
167 
```

```
168         foreach (double fitness in organisms.Select(organism => organism.Fitness)) {
169             if (fitness > maxFit) {
170                 maxFit = fitness;
171             } else if (fitness < minFit) {
172                 minFit = fitness;
173             }
174         }
175 
176         foreach (Organism organism in organisms) {
177             double normalisedFitness = (organism.Fitness - minFit) / (maxFit - minFit);
178 
179             if (_rnd.NextDouble() <= normalisedFitness) {
180                 survivors.Add(organism);
181             }
182         }
183 
184         if (survivors.Count % 2 != 0) {
185             survivors.RemoveAt(survivors.Count - 1);
186         }
187 
188         List<Organism> nextGeneration = new(organisms.Count);
189 
190         while (survivors.Count > 0 && nextGeneration.Count < 1000) {
191             Organism a = PopRandom(survivors);
192             Organism b = PopRandom(survivors);
193 
194             a.DoAgeing();
195             b.DoAgeing();
196 
197             nextGeneration.Add(a);
198             nextGeneration.Add(b);
199 
200             int offspring = Mathf.CeilToInt((float)((a.Fitness + b.Fitness) * (_config.
                    ↪ MaximumOffspring - _config.MinimumOffspring) / 2) +
201                                             _config.MinimumOffspring);
202 
203             for (int i = 0; i < offspring; i++) {
204                 Organism c = Reproduce(a, b);
205                 nextGeneration.Add(c);
206             }
207         }
208 
209         return nextGeneration;
210     }
211 
212     /// <summary>
213     /// Pops a random item from a list.
214     /// </summary>
215     private T PopRandom<T>(List<T> list) {
216         T item = list[_rnd.Next(0, list.Count)];
217         list.Remove(item);
218 
219         return item;
220     }
221 
222     /// <summary>
223     /// Calculates the offspring of two organisms.
224     /// </summary>
225     /// <param name="a">First organism</param>
226     /// <param name="b">Second organism</param>
227     /// <returns>Offspring</returns>
228     private Organism Reproduce(Organism a, Organism b) {
229         double[] attributeValues = new double[Organism.Attributes.Length];
```

```
230
231         for (int i = 0; i < Organism.Attributes.Length; i++) {
232             double average = (a.AttributeValues[i] + b.AttributeValues[i]) / 2;
233             attributeValues[i] = Mathf.Clamp((float)(average + RandomNormal.Random(_config.
                    ↪ MutationStrength)), -1, 1);
234         }
235
236         return new Organism(attributeValues);
237     }
238 }
```

4.6 FileHandler.cs

```
1  using System.IO;
2  using System.Text;
3
4  /// <summary>
5  /// Provides a simple write-only interface to a file.
6  /// </summary>
7  public class FileHandler {
8      public readonly string FilePath;
9      public readonly string FileName;
10
11     private readonly UTF8Encoding _encoder = new(true);
12
13     public FileHandler(string filePath) {
14         FilePath = filePath;
15         string[] split = filePath.Split(".", 2);
16
17         FileName = split[0];
18         FileName = split[1];
19
20         File.Create(FilePath).Close();
21     }
22
23     /// <summary>
24     /// Writes some text to the file being handled.
25     /// </summary>
26     /// <param name="value">The string to write.</param>
27     /// <param name="append">Whether text should be appended, or the file overwritten with new
           ↪ data.</param>
28     public void Write(string value, bool append = true) {
29         using FileStream fs = File.OpenWrite(FilePath);
30         byte[] data = _encoder.GetBytes(value);
31
32         fs.Seek(0, !append ? SeekOrigin.Begin : SeekOrigin.End);
33
34         fs.Write(data, 0, data.Length);
35         fs.Close();
36     }
37
38     public string[] ReadLines() {
39         using FileStream fs = new(FilePath, FileMode.Open, FileAccess.Read);
40         byte[] bytes = new byte[fs.Length];
41
42         int remaining = (int)fs.Length;
43         int read = 0;
44
45         while (remaining > 0) {
46             int n = fs.Read(bytes, read, remaining);
47
48             read += n;
```

```
            remaining -= n;
        }

        string file = _encoder.GetString(bytes);

        return file.Split("\n");
    }
}
```

4.7 ItemPlacer.cs

```
using JetBrains.Annotations;
using UnityEngine;

[System.Serializable]
public struct ItemType {
    public GameObject Prefab;
    public bool Inverse;

    [Range(0, 1)]
    public float Frequency;
}

[RequireComponent(typeof(NoiseGenerator))]
public class ItemPlacer : MonoBehaviour {
    public ItemType[] Items;
    public Transform Centre;

    [Header("Distribution")]
    public float Density;
    public float Variation;
    public Vector2 Range;

    [PublicAPI]
    private void Start() {
        NoiseGenerator noise = gameObject.GetComponent<NoiseGenerator>();

        for (float y = -Range.y; y < Range.y; y += Density) {
            for (float x = -Range.x; x < Range.x; x += Density) {
                float v = noise.Evaluate(x, y);

                foreach (ItemType item in Items) {
                    switch (item.Inverse) {
                        case true when v < item.Frequency:
                        case false when 1 - v < item.Frequency:
                            continue;
                    }

                    Vector2 pos = new Vector2(x, y) + Random.insideUnitCircle * Variation;
                    GameObject itemGameObject = Instantiate(item.Prefab, Centre);
                    itemGameObject.transform.position = new Vector3(pos.x, 0, pos.y);
                }
            }
        }
    }
}
```

4.8 NoiseGenerator.cs

```
using JetBrains.Annotations;
using UnityEngine;

public class NoiseGenerator : MonoBehaviour {

    public float Frequency;

    private Vector2 _centre;

    [PublicAPI]
    private void Awake() {
        _centre = Random.insideUnitCircle * float.MaxValue;
    }

    public float Evaluate(float x, float y) {
        float v = Mathf.PerlinNoise((x + _centre.x) * Frequency, (y + _centre.y) * Frequency);

        return v;
    }
}
```

4.9 Organism.cs

```
using System;
using UnityEngine;

/// <summary>
/// Represents an Organism for mathematical functions.
/// </summary>
public class Organism {
    public static readonly Attribute[] Attributes = {
        new("Strength", 0.90, 0.70),
        new("Intelligence", 0.95, 0.05),
        new("Agility", 0.85, 0.30),
        new("Endurance", 0.90, 0.15),
        new("Disease Susceptibility", -0.95, 0.00)
    };

    public double[] AttributeValues { get; private set; }

    private const double _ageingFactor = 0.2;
    public double Age { get; private set; } = 1;

    /// <summary>
    /// Constructs an <c>Organism</c> with random (std = 0.4) attribute values.
    /// </summary>
    public Organism() {
        AttributeValues = new double[Attributes.Length];

        for (int i = 0; i < Attributes.Length; i++) {
            AttributeValues[i] = Mathf.Clamp((float)RandomNormal.Random(0.4), -1, 1);
        }
    }

    /// <summary>
    /// Constructs an <c>Organism</c> with set attribute values.
    /// </summary>
    /// <param name="attributeValues">The attribute values to use.</param>
    /// <exception cref="ArgumentException"></exception>
    public Organism(double[] attributeValues) {
        if (attributeValues.Length != Attributes.Length) {
            throw new ArgumentException("attributeValues.Length != attributes.Length");
```

```
40          }
41
42          AttributeValues = attributeValues;
43      }
44
45      /// <summary>
46      /// Changes the organism's age by it's ageing factor.
47      /// </summary>
48      public void DoAgeing() {
49          Age *= _ageingFactor;
50      }
51
52      /// <summary>
53      /// Calculates the Fitness of the organism.
54      /// </summary>
55      public double Fitness {
56          get {
57              double averageFitness = 0;
58
59              for (int i = 0; i < Attributes.Length; i++) {
60                  averageFitness += Attributes[i].Fitness(AttributeValues[i]);
61              }
62
63              averageFitness /= Attributes.Length;
64
65              return averageFitness * Age;
66          }
67      }
68
69      /// <summary>
70      /// Get an Attribute's value from its name.
71      /// </summary>
72      /// <param name="name">The name of the Attribute to return.</param>
73      /// <returns>An Attribute value in the range [-1:1].</returns>
74      /// <exception cref="ArgumentException"></exception>
75      public double AttributeValue(string name) {
76          for (int i = 0; i < Attributes.Length; i++) {
77              if (Attributes[i].Name == name) {
78                  return AttributeValues[i];
79              }
80          }
81
82          throw new ArgumentException("no␣attribute␣named␣{}", name);
83      }
84  }
85
86  /// <summary>
87  /// Represents one of an <c>Organism's</c> Attributes.
88  /// </summary>
89  [Serializable]
90  public struct Attribute {
91      public readonly string Name;
92      public readonly double BenefitWeight;
93      public readonly double CostWeight;
94
95      public Attribute(string name, double benefitWeight, double costWeight) {
96          Name = name;
97          BenefitWeight = benefitWeight;
98          CostWeight = costWeight;
99      }
100
101     /// <summary>
102     /// Calculate this Attribute's fitness contribution based on some value.
```

```
103     /// </summary>
104     /// <param name="value">The value to use in the range [-1:1].</param>
105     /// <returns>This Attribute's fitness given <c>value</c> in the range [-1:1].</returns>
106     public double Fitness(double value) {
107         double benefit = BenefitWeight * value;
108         double cost = CostWeight * value * value;
109         return Math.Tanh(benefit - cost);
110     }
111 }
```

4.10 OrganismController.cs

```
1  using JetBrains.Annotations;
2  using UnityEngine;
3  using UnityEngine.AI;
4
5  [RequireComponent(typeof(NavMeshAgent))]
6  public class OrganismController : MonoBehaviour {
7      public NavMeshAgent Agent;
8      public MeshRenderer Mesh;
9      public Material BaseMaterial;
10     public Gradient Gradient;
11     public SimulationState State;
12
13     /// <summary>
14     /// Initialise this <c>OrganismController</c> based on some <c>Organism</c>.
15     /// </summary>
16     /// <param name="organism">The <c>Organism</c> to extract values from.</param>
17     public void Initialise(Organism organism, SimulationState state) {
18         SetStats(organism);
19
20         State = state;
21
22         gameObject.transform.SetPositionAndRotation(RandPos(), Quaternion.identity);
23         Agent.SetDestination(RandPos());
24     }
25
26     /// <summary>
27     /// Update the attributes of the <c>Organism</c> this controller is based off.
28     /// </summary>
29     /// <param name="organism">The <c>Organism</c> to extract values from.</param>
30     public void SetStats(Organism organism) {
31         float v = (float)((organism.Fitness + 1) / 2);
32
33         Mesh.material = new Material(BaseMaterial) {
34             color = Gradient.Evaluate(v)
35         };
36
37         Agent.speed = (float)(organism.AttributeValue("Agility") * organism.AttributeValue("
               ↪ Endurance") * 20 + 1);
38     }
39
40     [PublicAPI]
41     private void Update() {
42         Agent.isStopped = State.Paused;
43
44         if (Random.value < 0.05) {
45             Agent.SetDestination(RandPos());
46         }
47     }
48
49     public Vector3 RandPos() {
```

```
50          const int width = 100;
51
52          return new Vector3(Random.Range(-width, width), 0, Random.Range(-width, width));
53      }
54  }
```

4.11 RandomNormal.cs

```
1   using System;
2
3   public class RandomNormal {
4       private static readonly Random _rand = new();
5
6       /// <summary>
7       /// Generates a random normal with mu = 0, and standard deviation <c>std</c>
8       /// </summary>
9       /// <param name="std">Standard distribution.</param>
10      public static double Random(double std) {
11          double u1 = 1.0 - _rand.NextDouble();
12          double u2 = 1.0 - _rand.NextDouble();
13          double normal = Math.Sqrt(-2.0 * Math.Log(u1)) * Math.Sin(2.0 * Math.PI * u2);
14
15          return normal * std;
16      }
17  }
```

4.12 SimulationSetupController.cs

```
1   using JetBrains.Annotations;
2   using TMPro;
3   using UnityEngine;
4   using UnityEngine.UI;
5
6   /// <summary>
7   /// Controls the SimulationSetup scene.
8   /// </summary>
9   public class SimulationSetupController : MonoBehaviour {
10      public StaticMenuControllerHandle MenuController;
11      public Slider MutationSlider;
12      public TextMeshProUGUI MutationText;
13      public Slider PopSizeSlider;
14      public TextMeshProUGUI PopSizeText;
15
16      private EvolutionConfig _config;
17
18      [PublicAPI]
19      private void Start() {
20          Defaults();
21      }
22
23      [PublicAPI]
24      private void Update() {
25          _config.MutationStrength = MutationSlider.value;
26          _config.InitialPopulationSize = Mathf.FloorToInt(PopSizeSlider.value);
27
28          // Set text to a value from 0-10 instead of small fractions
29          MutationText.text = (Mathf.Round(Mathf.Lerp(0, 10, MutationSlider.value / MutationSlider
                ↪ .maxValue) * 10) / 10).ToString();
30          PopSizeText.text = PopSizeSlider.value.ToString();
31
```

```
32          MenuController.StaticController.EvolutionConfig = _config;
33      }
34
35      /// <summary>
36      /// Reset values to defaults (overwrites with a new <c>EvolutionConfig</c>).
37      /// </summary>
38      public void Defaults() {
39          _config = ScriptableObject.CreateInstance<EvolutionConfig>();
40
41          MutationSlider.value = (float)_config.MutationStrength;
42          PopSizeSlider.value = _config.InitialPopulationSize;
43      }
44  }
```

4.13 SimulationState.cs

```
1   using System.Collections.Generic;
2   using UnityEngine;
3
4   /// <summary>
5   /// Stores data on the current state of the evolution simulation.
6   /// </summary>
7   public class SimulationState : ScriptableObject {
8       public int Generation;
9       public List<Organism> CurrentGen;
10      public List<Organism> PreviousGen;
11      public bool Paused;
12  }
```

4.14 StaticMenuController.cs

```
1   using JetBrains.Annotations;
2   using UnityEngine;
3   using UnityEngine.SceneManagement;
4
5   /// <summary>
6   /// Scene controller that exists across scene changes so that persistent data can be stored.
7   /// </summary>
8   public class StaticMenuController : MonoBehaviour {
9
10      public EvolutionConfig EvolutionConfig;
11
12      private string _last;
13
14      [PublicAPI]
15      private void Awake() {
16          DontDestroyOnLoad(gameObject);
17          EvolutionConfig = ScriptableObject.CreateInstance<EvolutionConfig>();
18      }
19
20      // ReSharper disable once ParameterHidesMember
21      /// <summary>
22      /// Loads scene <c>name</c>.
23      /// </summary>
24      /// <param name="name">String name of the Scene to load.</param>
25      public void LoadScene(string name) {
26          _last = SceneManager.GetActiveScene().name;
27          SceneManager.LoadScene(name);
28      }
29
```

```
30      /// <summary>
31      /// Returns to the previous scene.
32      /// </summary>
33      public void Back() {
34          SceneManager.LoadScene(_last);
35      }
36  }
```

4.15 StaticMenuControllerHandle.cs

```
1   using JetBrains.Annotations;
2   using UnityEngine;
3
4   /// <summary>
5   /// Provides an interface to a <c>StaticMenuController</c>, allowing the static controller to
        ↪ exist between scenes.
6   /// </summary>
7   public class StaticMenuControllerHandle : MonoBehaviour {
8       /// <summary>
9       /// Name of the <c>GameObject</c> with a <c>StaticMenuController</c> that this script should
            ↪ use.
10      /// </summary>
11      public string StaticObjectName = "Static";
12
13      public StaticMenuController StaticController { get; private set; }
14
15      [PublicAPI]
16      private void Awake() {
17          StaticController = GameObject.Find(StaticObjectName).GetComponent<StaticMenuController
                ↪ >();
18      }
19
20      // ReSharper disable once ParameterHidesMember
21      /// <summary>
22      /// Loads scene <c>name</c>.
23      /// </summary>
24      /// <param name="name">String name of the Scene to load.</param>
25      public void LoadScene(string name) {
26          StaticController.LoadScene(name);
27      }
28
29      /// <summary>
30      /// Return to the previous scene.
31      /// </summary>
32      public void Back() {
33          StaticController.Back();
34      }
35  }
```

4.16 UiController.cs

```
1   using System;
2   using JetBrains.Annotations;
3   using TMPro;
4   using UnityEngine;
5
6   [RequireComponent(typeof(EvolutionManager))]
7   public class UiController : MonoBehaviour {
8       private EvolutionManager _evolutionManager;
9
```

```
10      public CsvMaker CsvMaker;
11
12      public GameObject NonGraphUi;
13      public GameObject GraphUi;
14
15      public Transform GraphParent;
16      public GameObject GraphPoint;
17      public GameObject GraphTick;
18      public Transform GraphMin;
19      public Transform GraphMax;
20
21      public TMP_Dropdown GraphTypeDropdown;
22      public TMP_Dropdown GraphTimeDropdown;
23
24      private bool _graphsShown;
25
26      [PublicAPI]
27      private void Start() {
28          _evolutionManager = GetComponent<EvolutionManager>();
29
30          foreach (Attribute attribute in Organism.Attributes) {
31              GraphTypeDropdown.options.Add(new TMP_Dropdown.OptionData($"{attribute.Name}:␣Average
                    ↪ "));
32              GraphTypeDropdown.options.Add(new TMP_Dropdown.OptionData($"{attribute.Name}:␣
                    ↪ Distribution"));
33          }
34      }
35
36      [PublicAPI]
37      public void SkipGenerations(int n) {
38          _evolutionManager.GenerationGoal = _evolutionManager.State.Generation + n;
39      }
40
41      [PublicAPI]
42      public void PlayPause() {
43          _evolutionManager.State.Paused = !_evolutionManager.State.Paused;
44      }
45
46      [PublicAPI]
47      public void Graph() {
48          _graphsShown = !_graphsShown;
49
50          NonGraphUi.SetActive(!_graphsShown);
51          GraphUi.SetActive(_graphsShown);
52
53          DrawGraph();
54      }
55
56      public void DrawGraph() {
57          string typeDropdown = GraphTypeDropdown.options[GraphTypeDropdown.value].text;
58          string timeDropdown = GraphTimeDropdown.options[GraphTimeDropdown.value].text;
59
60          foreach (Transform child in GraphParent) {
61              Destroy(child.gameObject);
62          }
63
64          var files = CsvMaker.Files;
65
66          FileHandler file;
67          GraphType type = GraphType.Average;
68
69          switch (typeDropdown) {
70              case "Total␣Population":
```

```
71                  type = GraphType.Sum;
72                  file = files["Fitness"];
73                  break;
74              case "Change␣in␣Population":
75                  return;
76              default:
77                  file = typeDropdown == "Fitness" ? files["Fitness"] : files[typeDropdown.Split(":
                        ↪ ")[0]];
78
79                  if (typeDropdown != "Fitness" && typeDropdown.Split(":␣")[1] == "Distribution") {
80                      type = GraphType.Distribution;
81                  }
82
83                  break;
84          }
85
86          string[] contents = file.ReadLines();
87
88          int time = timeDropdown switch {
89              "Last␣10␣Generations" => Mathf.Min(contents.Length - 2, 10),
90              "Last␣50␣Generations" => Mathf.Min(contents.Length - 2, 50),
91              _ => contents.Length - 2
92          };
93
94          if (type == GraphType.Distribution) {
95              Histogram(contents);
96          } else {
97              LineGraph(contents, time, type);
98          }
99      }
100
101     /// <summary>
102     /// Draws a line graph of the sum or average value from the contents file.
103     /// </summary>
104     /// <param name="contents">Lines of csv to graph.</param>
105     /// <param name="time">Will graph the last <c>time</c> rows of <c>contents</c>.</param>
106     /// <param name="type">Statistic to graph (sum or average).</param>
107     private void LineGraph(string[] contents, int time, GraphType type) {
108         float graphHeight = GraphMax.position.y - GraphMin.position.y;
109
110         const int maxYTick = 10;
111         float ySpacing = graphHeight / (maxYTick * 2 + 1);
112
113         for (int i = -10; i <= 10; i++) {
114             float y = i * ySpacing + (graphHeight / 2 + GraphMin.position.y);
115             GameObject tick = Instantiate(GraphTick, new Vector3(GraphMin.position.x, y, 0),
                    ↪ Quaternion.identity, GraphParent);
116
117             tick.transform.localScale = MathF.Abs(i) % 10 == 0 ? new Vector3(15, 2, 1) : new
                    ↪ Vector3(10, 2, 1);
118         }
119
120         string[] header = contents[0].Split(",");
121         for (int i = contents.Length - time - 1; i < contents.Length - 1; i++) {
122             string[] row = contents[i].Split(",");
123
124             float sum = 0;
125             int n = 0;
126
127             for (int j = 1; j < row.Length; j++) {
128                 try {
129                     sum += float.Parse(header[j]) * float.Parse(row[j]);
130                     n += int.Parse(row[j]);
```

```
131                 } catch (FormatException e) {
132                     Console.WriteLine(e);
133                 }
134             }
135 
136             float value = type == GraphType.Average ? sum / n : n;
137             float min = type == GraphType.Average ? -1 : 0;
138             float max = type == GraphType.Average ? 1 : 1_500;
139 
140             float x = Mathf.Lerp(GraphMin.position.x, GraphMax.position.x, (i - contents.Length +
                    ↪  time + 1) / (float)time);
141             float y = Mathf.Lerp(GraphMin.position.y, GraphMax.position.y, Map(value, min, max));
142 
143             Instantiate(GraphPoint, new Vector3(x, y, 0), Quaternion.identity, GraphParent);
144             GameObject tick = Instantiate(GraphTick, new Vector3(x, GraphMin.position.y, 0),
                    ↪ Quaternion.identity, GraphParent);
145             tick.transform.localScale = (i - 1) % 10 == 0 ? new Vector3(2, 15, 1) : new Vector3
                    ↪ (2, 10, 1);
146 
147         }
148     }
149 
150     private void Histogram(string[] contents) {
151 
152     }
153 
154     private static float Map(float v, float min, float max) {
155         return (v - min) / (max - min);
156     }
157 }
158 
159 internal enum GraphType {
160     Average,
161     Sum,
162     Distribution,
163 }
```