NATURAL SELECTION SIMULATOR

Contents

Analysis Phase

THE PROBLEM

In this NEA, I will attempt to code a program that simulates natural selection. Natural selection can be a difficult topic for many biology students (including myself) to comprehend. Due to the fact that the actual process of natural selection occurs over several generations, it is often overwhelming when students try to visualise it. Therefore, the topic is taught mainly in theory, with students having to accept the facts given to them – without being able to question or witness how the process would work. Graphs are also used to demonstrate areas such as carrying capacity or the predator-prey relationship, which can become confusing when there is nothing to refer it to.

WHAT IS NATURAL SELECTION?

The definition of natural selection is: the process whereby organisms better adapted to their environment tend to survive and produce more offspring. The theory of natural selection was first proven by Charles Darwin when he visited the Galápagos Islands. There, he noticed that the beaks of the finches on each island were slightly different. Later, he proved that this was due to the availability of food on each island (e.g. more fruits would lead to a different beak shape to more worms available). The birds acquired the beak shape initially due to mutation (a beneficial trait). These birds were more likely to survive and reproduce, as they were better adapted to the environment. The beneficial trait of the beak shape was passed onto offspring. Eventually, over many generations, most of the birds had this unique beak shape. Natural selection is also seen in many other organisms (e.g. giraffes evolving to have long necks to reach leaves on trees). An example in the UK would be the peppered moth scenario. Before the Industrial Revolution, peppered moths were white in order to camouflage against tree trunks to avoid being hunted by birds. However, during the Industrial Revolution, there was a lot of pollution, and as a result, tree trunks became blackened due to the soot. The white moths now stood out when placed on the darker trees, meaning they were more likely to be hunted. Through natural selection, phenotypically black peppered moths appeared. These are now prevalent in urban areas.

POTENTIAL USER(S)

My target audience would be mainly teachers – due to the fact that they may find the resource useful enough to show to their classes. However, I would aim to appeal to students as well, in order to aid their revision of the topic. Specifically, I aim to target the project to biology students and teachers. Although some people who do not have the fundamental knowledge of natural selection may want to experiment with the simulation themselves, I believe that an explanation of every part of the program would cause it to become wordy and uninteresting. The user must be kept engaged. I expect the majority of the people who would use the program would be biology students or teachers. As a result, I will create this simulation with the assumption that the user has at least some knowledge on the process of natural selection. The user interface must be clear and easy to use, as (most likely) the user will not have a significant understanding of how to use simulation software. Additionally, if it was to be used in classes, teachers would expect to use it for less than an hour. Therefore, a complicated user interface would discourage them from using it.

USER REQUIREMENTS

First, a start screen must be created with basic information on natural selection, along with basic control instructions. There should be an option for the user to input the initial population of the community, or use a default option. Next, a plane must be created in order to place the ‘organisms’ that will be representing animal populations. The organisms should be able to spawn randomly in order to replicate reproduction. However, this must happen in a methodical way – fulfil the conditions required to breed (e.g. find a partner/reach breeding age). There should be a suitable

system set, so that the organisms have designated energy, and a way to die. And, most importantly, there should be food available. I also aim to have a page where this information on population is graphed for the user to see, along with some information on why the graph is the shape it is. To actually witness how beneficial traits are passed on to the offspring, there need to be mutations – such as speed, fertility etc. There needs to be a process on how these should be passed on to the next generation (dominant or recessive), and the chance for mutation in the first place. There should also be an option to activate predators so that users can see the relationship between predator and prey.

RESEARCH

Although school attempts to prepare us for the real life world, the information we learn is differs to experiencing it in real life. The best way to practice real life genuine situations is simulations. They give the learner a chance to experiment and discover responses themselves, and they are the closest thing to reality. Through simulations we can accurately illustrate real events - it's a faster and cheaper effective way to improve the learner's skills and competences. Simulations are increasingly often used in higher education settings. In STEM (science, technology, engineering, and mathematics) education, they are used to facilitate a deeper understanding of concepts and relationships between them, advance inquiry, problem solving, and decision making. There has been a lot of research done in the medical field – on how simulations are used to advance motor and technical skills of prospective doctors, nurses, and emergency teams. Simulation-based learning also occurs in other fields, such as teaching, engineering, and management. There are many topics in which a simulation would increase the understanding of the learner. For example, in cases where it would be risky or dangerous for the learner to practice the task (e.g. if someone learning to drive again after a serious injury can benefit greatly from driving in a simulator with the supervision of professionals before driving an actual vehicle). Another example, would be in the medical field; students could use simulations to practice procedures they have learnt about in theory – e.g. performing surgery in VR. This way, there would be harm to the patient if anything went wrong, and the student would be more confident in performing surgery in real life. Simulations would also be useful in cases where the theory of the learning takes place over a long time – simulations would allow the learner to visualise something that may not happen in their lifetime (e.g. natural selection, which happens over many generations).

GRAPHS AND DEFINITIONS

Variation is important to understand, as it plays a big part in natural selection. Without variation, organisms would be very susceptible to changes in the environment and would not be able to adapt and change to survive. There are many different ways genetic variation is caused. Summarising them:

1) Genes have different alleles (alternative forms of a gene that code for different characteristics – e.g. there would be an allele coding for green eyes and a separate allele coding for blue eyes. However, these both would be the same gene coding for eye colour). Different combinations of alleles causes variation.
2) Mutation – changes in the DNA sequence causes the proteins that are coded for by the DNA to be different to usual. This protein change can cause both visible/non-visible characteristics. If the mutation occurs in the gametes, it can be passed onto offspring.
3) Gametes (sex cells) are produced in a way that each is slightly different (the genetic information of the parents is 'mixed up') leading to each gamete being genetically different. This is why siblings do not have the same DNA.

4) Adding on from the previous point, chance also plays a big part in variation of an individual. It is up to chance which two of the gametes combine, and each gamete is different from another.

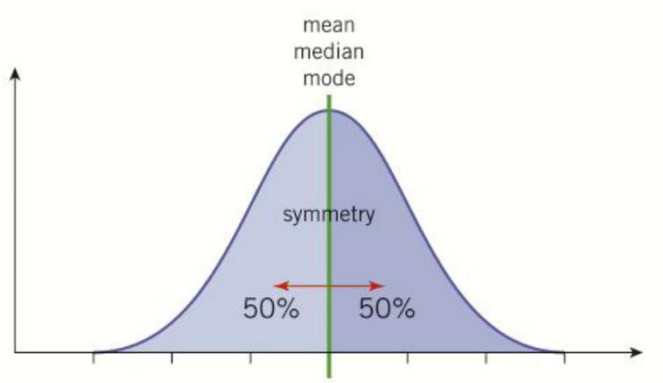


Figure 1 – Normal Distribution Curve

Often, what we see with continuous variation (variation that can be measured, such as height or weight), is a bell curve. The majority of the population will be in the middle, while few individuals are placed on either extreme. Due to the amount of ways variation can occur, it is impossible to simulate them all. Therefore, I will be focusing on mutation in the project.

If the growth of a new population is plotted on a graph over time, regardless of the organism, most populations will show the same trends. This is called the population growth curve:

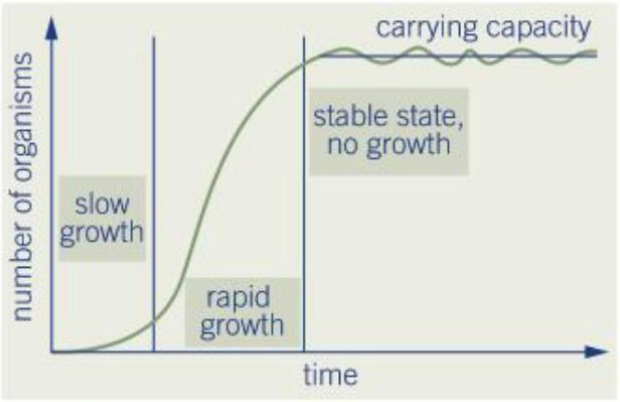


Figure 2 – Growth curve of most natural populations. This is referred to as a sigmoid population growth curve

First, the initially small population reproduces. The birth rate is higher than the death rate at this point, so the population increases. However, the growth is fairly slow. Over time, there are more individuals at breeding age, causing the population to increase exponentially. There are no constraints to limit the population growth at this point. Finally, further population growth is limited by external constraints (such as food availability). The population is in a fairly stable state, with some fluctuations due to factors such as predators. The birth and death rate are approximately equal. The maximum population size an environment can support is known as the carrying capacity.

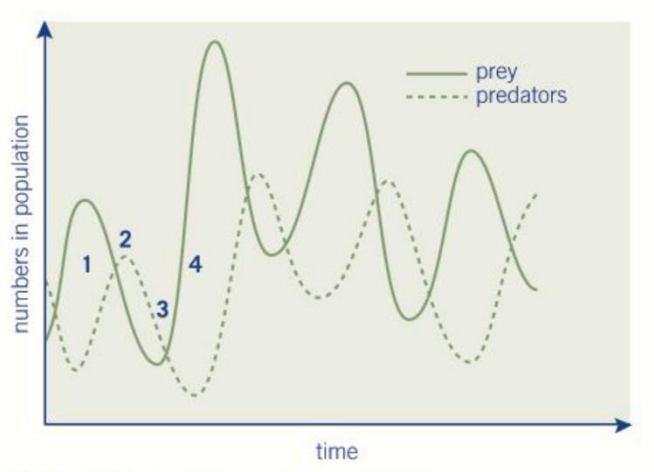


Figure 3 *– General predator-prey graph*

This is a general predator-prey graph. The sizes of the predator and prey populations are interlinked. As the population of one changes, it causes a change in the size of the other population. This results in fluctuations. In general, all predator-prey relationships follow the same pattern. 1 – an increase in the prey population provides more food for the predators, allowing more to survive and reproduce. This, in turn, increases the predator population. 2 – the increases predator population eats more prey organisms, causing a decline in the prey population. The death rate of the prey is greater than birth rate. 3 – the reduced prey population can no longer sustain the larger predator population. Competition for food increases, resulting in a decrease of the predator population. 4 – the reduce in predators mean that less of the prey are being killed. More prey survives and reproduces, increasing the prey population. The cycle begins again. In the wild, it isn't this simple, as other factors also affect the population of both species. However, for the simulation, I will assume there are no other external factors.

SOURCES

https://journals.sagepub.com/doi/full/10.3102/0034654320933544
Chernikova, O., Heitzmann, N., Stadler, M., Holzberger, D., Seidel, T., & Fischer, F. (2020). Simulation-Based Learning in Higher Education: A Meta-Analysis. *Review of Educational Research*, *90*(4), 499–541. https://doi.org/10.3102/0034654320933544

https://centrical.com/top-4-benefits-of-simulations-in-learning-practices/
The 4 Top Benefits Of Simulations In Learning Practices. (2020, October 21). Centrical. https://centrical.com/top-4-benefits-of-simulations-in-learning-practices/

https://www.usa.edu/blog/simulation-in-education/
Benefits of Simulation in Education | USAHS. (2021, September 1). University of St. Augustine for Health Sciences. https://www.usa.edu/blog/simulation-in-education/

https://gamedevacademy.org/complete-guide-to-procedural-level-generation-in-unity-part-1/
Renan Oliveira. (2019, July 23). *Complete Guide to Procedural Level Generation in Unity – Part 1*. GameDev Academy. https://gamedevacademy.org/complete-guide-to-procedural-level-generation-in-unity-part-1/

CLIENT INTERVIEW

Interview with Ms Dhillon (Biology teacher):

1) What is the biggest problem you face when teaching about natural selection?

   A: Most examples of natural selection happen over millions of years and we have incomplete evidence which makes it hard to explain. Also, the fact that a mutation in one particular habitat which is a disadvantage and would die out, could be an advantage in a completely different habitat and end up being passed on.

2) Do students find it difficult to visualise the topics you teach about (e.g. carrying capacity/predator-prey relationships)?

   A: No not really, the graphs especially for predator prey relationships are relatively easy to interpret.

3) Do you think students would benefit from a visual aid such as a simulation? (Why/why not?)

   A: Visual aids are always useful to be able to demonstrate concepts that might be harder to explain and understand if you only hear about them.

4) If you have already looked at existing simulations, what would you say was the main point of improvement for them?

   A: I haven't seen actual simulations, I have seen very simple models on PowerPoint and made one myself but don't have the technical expertise to demonstrate all the concepts or make it look particularly exciting

5) If you were to use a natural selection simulator in your classes, which interactive features would you want it to have and why?

   A: I would want a bank of features which can be randomly allocated to different individuals in a population. I would want there to be an option to change the habitat so that the same random feature which comes out in a mutation might be an advantage in one location and a disadvantage in another, to be possible to show what happens after multiple generations so it doesn't look as if the changes that are happening happen straight away

6) What would you expect the simulation to offer visually? (For example, in terms of user interface/the design of the predators or prey)

   A: Completely up to you. You can use examples from real life, perhaps as an introduction? Or you can choose to use characters which are completely fictional but it's clear that the concept would be translate able to real life examples.

Interview with Catherine Oo (biology student):

1) What is the biggest problem you face when learning about natural selection?

   A: It can get slightly confusing when imagining how the relationships between predator and prey occur in the real world (compared to just looking at the graph). And, how mutations can be passed down so that the entire population could have the same mutation at one point.

2) Do you find it difficult to visualise the topics you learn about (e.g. carrying capacity/predator-prey relationships)?

A: Yes, as stated before, it is a lot to take in without something to look at. It can be hard to imagine it on a larger scale too.

3) Do you think you would benefit from a visual aid such as a simulation? (Why/why not?)

A: I think I would benefit a lot from a visual aid, because my mind could recall the aid when recalling the topic.

4) If you have already looked at existing simulations, what would you say was the main point of improvement for them?

A: I have not looked in depth at any existing simulations – the ones I have looked quickly at seem quite complicated to use.

5) If you were to use a natural selection simulator for your revision, which interactive features would you want it to have and why?

A: I would want it to have a graph to display both populations of predators and prey – and have them together so I can compare and see the relationship. Also, I would like to have an option to manipulate certain aspects of the animals so they can become faster for example.

6) What would you expect the simulation to offer visually? (For example, in terms of user interface/the design of the predators or prey)

A: Something easy to understand and simple to use, because otherwise it may put people off.

PROBLEMS WITH EXISTING SYSTEMS

When looking online, I found a few existing natural selection simulations. Firstly, I discovered a quite in depth simulation: https://phet.colorado.edu/en/simulations/natural-selection.

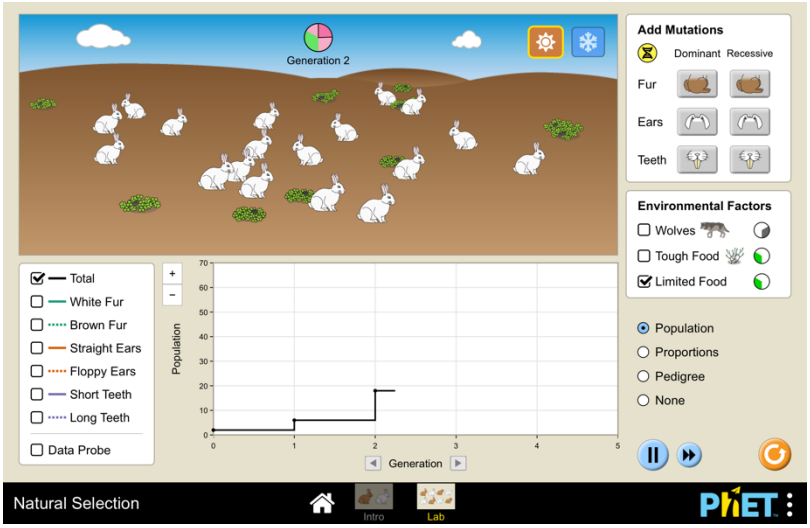


Figure 4 – Natural selection simulator by PHET

Advantages: there were options for mutations and it seemed to be specific to one species (bunnies). This made it easier to imagine and understand. There was a graph which showed the population of the bunnies and there were available options for the graph to show the populations of the bunnies with different traits. This is a very good way to see which traits are beneficial and which get passed on. There was also an option to add predators (wolves), which made the simulation more realistic.

Disadvantages: However, in my opinion, the user interface was too cluttered and hard to understand. It was easy to become overwhelmed, especially when the bunny population was growing at a very quick pace. There was only one graph shown, and it had no explanation as to why anything was happening. Overall, it was a good simulation, but many features could be improved upon to better the understanding of the learner.

The second simulation I looked at was from: https://www.biologysimulations.com/natural-selection.

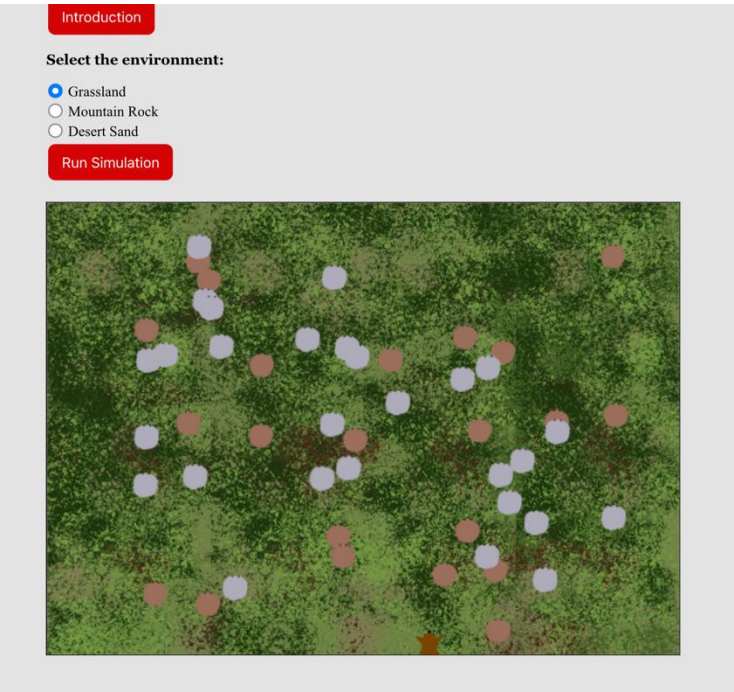


Figure 5 – Natural selection simulation by biology simulations

Advantages: There was an option to change the terrain of the simulation, and there was an introduction explaining how the simulation worked. In this simulation, there was an interactive feature as the person is the predator. The goal is to 'eat' as many individuals as possible, and the program gives the ending population of each coloured individual.

Disadvantages: this does not really simulate natural selection. It seems to be more of a game and doesn't serve to educate. The user is not able to add mutations, or see how the individuals reproduce. As far as I can see, the changing of the environment does not affect the way the simulation is run. Overall, there are many features to be improved in this simulation, as it needs to serve more of an educational purpose.

Finally, I looked at another program that had larvae as the species in the simulation: https://cloud.anylogic.com/model/26c923cb-6735-4534-af33-ac5a406825ff?mode=SETTINGS.

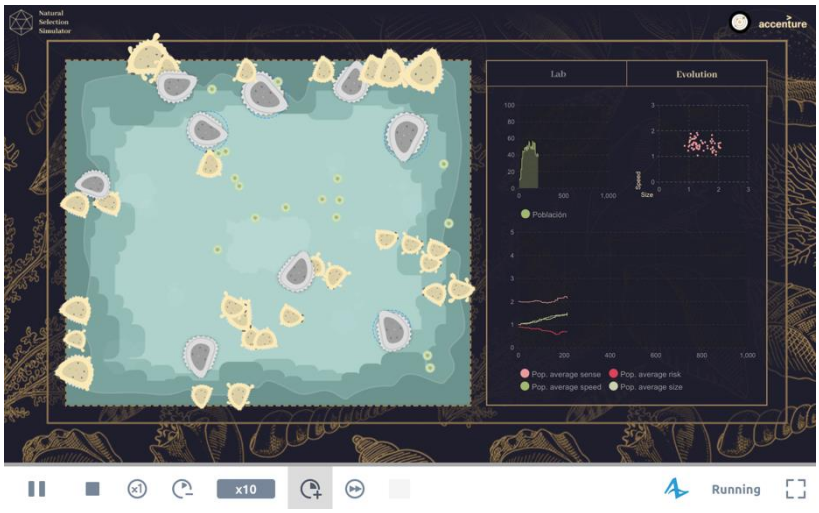


Figure 6 – Natural selection simulator by anylogic

Advantages: this simulation was probably the best out of all the ones I looked at. There were options to enable mutations, and change the food availability. There was also a way to change the speed of the simulation, which meant you didn't have to wait long to see the graphs start to develop. Another feature I liked, was the amount of graphs the program had. This really gave an in depth view of how over time the population evolves to have the best characteristics suited to the environment.

Disadvantages: some parts of the program were translated poorly, and some parts weren't translated at all, leading to some difficulty in understanding the content of the simulation. There was also no way to add predators to see the relationship between predator and prey. Overall, this was the best of the 3, but there were still some things that could've been improved to make the experience better.

In my simulation, I hope to add to and improve some of the features that these systems implemented. The user interface should not be cluttered, and should be easy to understand and navigate for a beginner. There should also be mutations that can be experimented with (perhaps with the chance of passing it down). Finally, there should be an option to add predators, with a way to see the relationship between predator and prey.

OBJECTIVES

Here are my main objectives (things I would like to implement in my program):

1) Implement system that allows the organism to have set amounts of energy (e.g. each organism has a certain amount of energy points which get used up as it walks)

2) Ensure that the chickens, predators and food spawn in a random arrangement each time, and not on top each other so that they cannot be seen

3) Ensure that the movement of the predators/prey is random so that the simulation can be accurately represented.

4) Create a 3D terrain around the plane via 3D procedural generation and Perlin Noise.

5) Create a simulation that accurately represents the reproduction that occurs during natural selection – a greater amount of food allows the prey to reproduce.

6) Create a simulation that accurately represents the death that occurs during natural selection – this could be linked to how much energy the individual has (e.g. running out of energy means death or running into a predator)

7) Implement the ability of organisms to have mutations – this relates back to natural selection itself, but there should be a set chance for mutations.

8) Allow the user to have the ability to change features in the simulation
    8.1) Allow the user to change the amount of food that spawns at the beginning of each generation through a slider
    8.2) Allow the user to change the amount of predators on the plane at one time – in this way, the deaths of the chickens could be controlled through a slider

9) Create a graph system that displays the population of the organisms - this would allow the user to understand/comprehend the theory behind what they are seeing and witness the figures behind the process.
    9.1) Create an x and y axis with appropriate scales so that if the number of generations that are cycled through is large, the axis will be able to fit each of the points
    9.2) Create a line graph with clear points marked and connections with the next point
    9.3) Create 2 separate lines – one for the population of chickens and one for the population of foxes

10) Implement a settings page where the user is able to manipulate settings such as speed of the program – this would enable the user to make the whole process faster (so that they wouldn't have to sit through the entirety of the generation living/dying.

MODELLING

Essentially, within this simulation, I hope to implement a clear and easy to understand user interface, and a simulation with both predators and prey. The prey should be able to move around randomly and collect food. Once it reaches the red area (surrounding the plane) it will stop and depending on the amount of food it has collected, it will either live, die or reproduce. The predators will be able to be spawned, and multiple mutations will be implemented for the chickens (e.g. speed, energy). A graph will show the populations of predators and prey so that the relationship between them can be seen.

Design Stage

TERRAIN DESIGN

In this simulation I procedurally generated levels using pseudorandom noise to generate height maps. In order to procedurally generate levels, noise functions must be implemented in the code. A noise function is a function that generates pseudorandom values based on some input arguments. By pseudorandom it means the values look random, despite being algorithmically generated. In

practice, you can see this noise as a variable, which can be used as the height, temperature, or moisture values of a level region. In this case, I used it as the height of different coordinates of our level. There are different noise functions, but I used one called Perlin Noise.

The noise map will receive as parameters the map height, map width and a scale. Then, it will generate a matrix representing a noise map, with the noise in each coordinate of the level. For each coordinate, the noise value is generated using the Mathf.PerlinNoise function. This function receives two parameters and generate a value between 0 and 1, representing the noise.

A method will be responsible for changing the plane mesh vertices according to the noise map. It will iterate through all the tile coordinates and change the corresponding vertex y coordinate to be the noise value multiplied by a height multiplier.

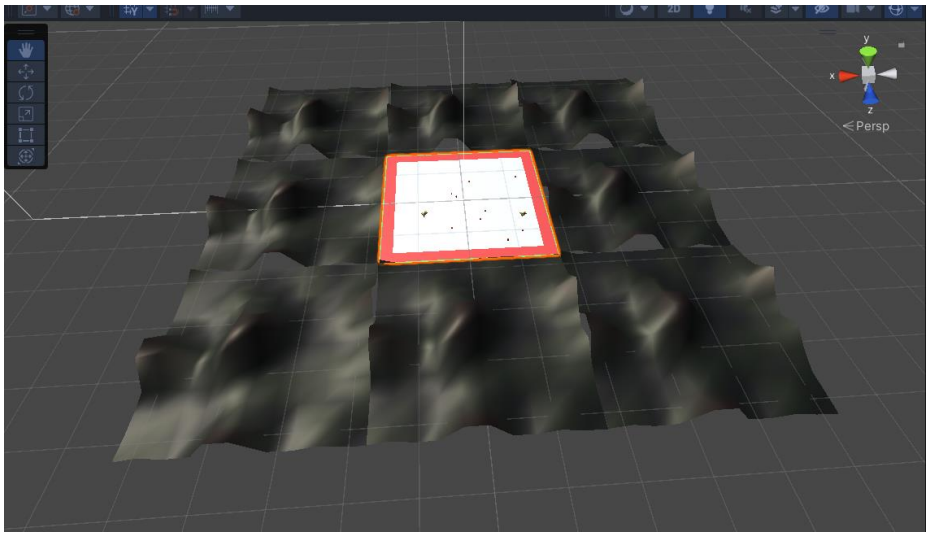


Figure 7 – Screenshot showing 3D terrain design

PREY DESIGN

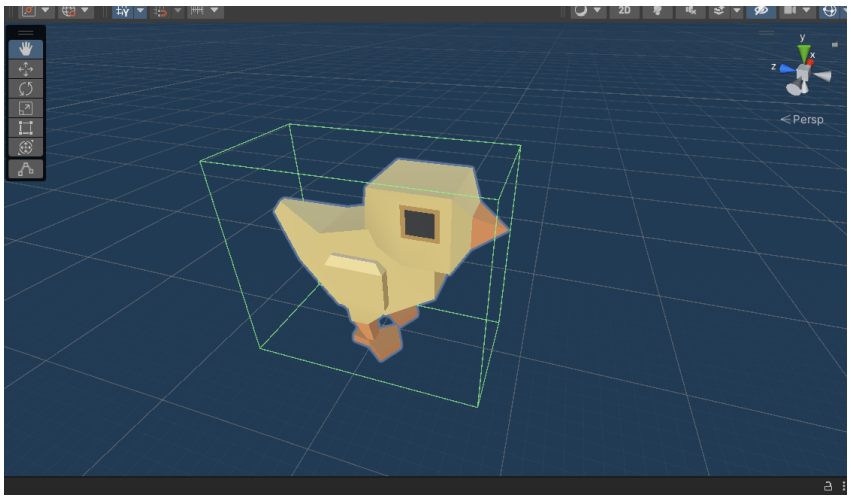


Figure 8 – Screenshot showing prey design

The prey I decided to use were chickens, as they have many natural predators and they seemed to be a good model for the simulation. The imported model has 3 different animations that I utilised – idle, eating and walking. I mapped the animations, linking them together:

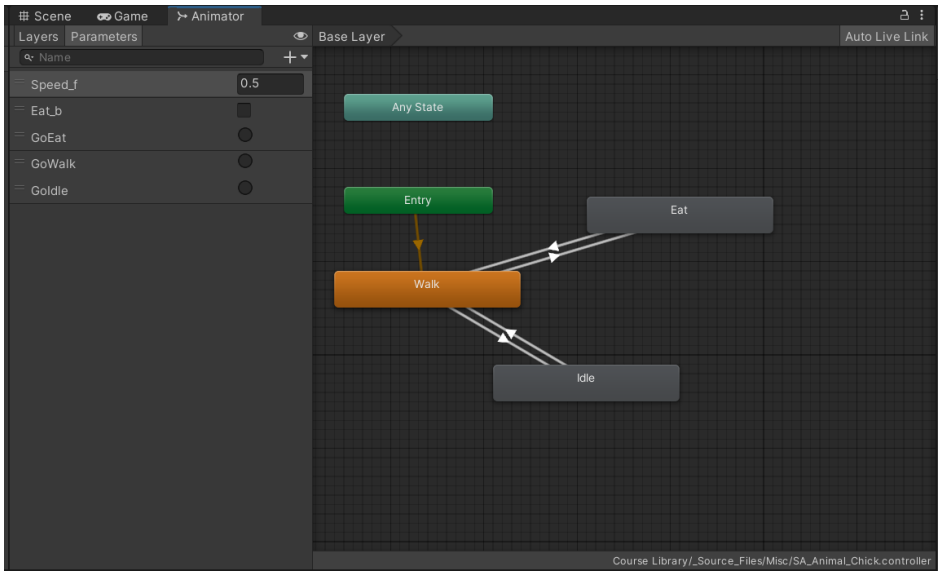
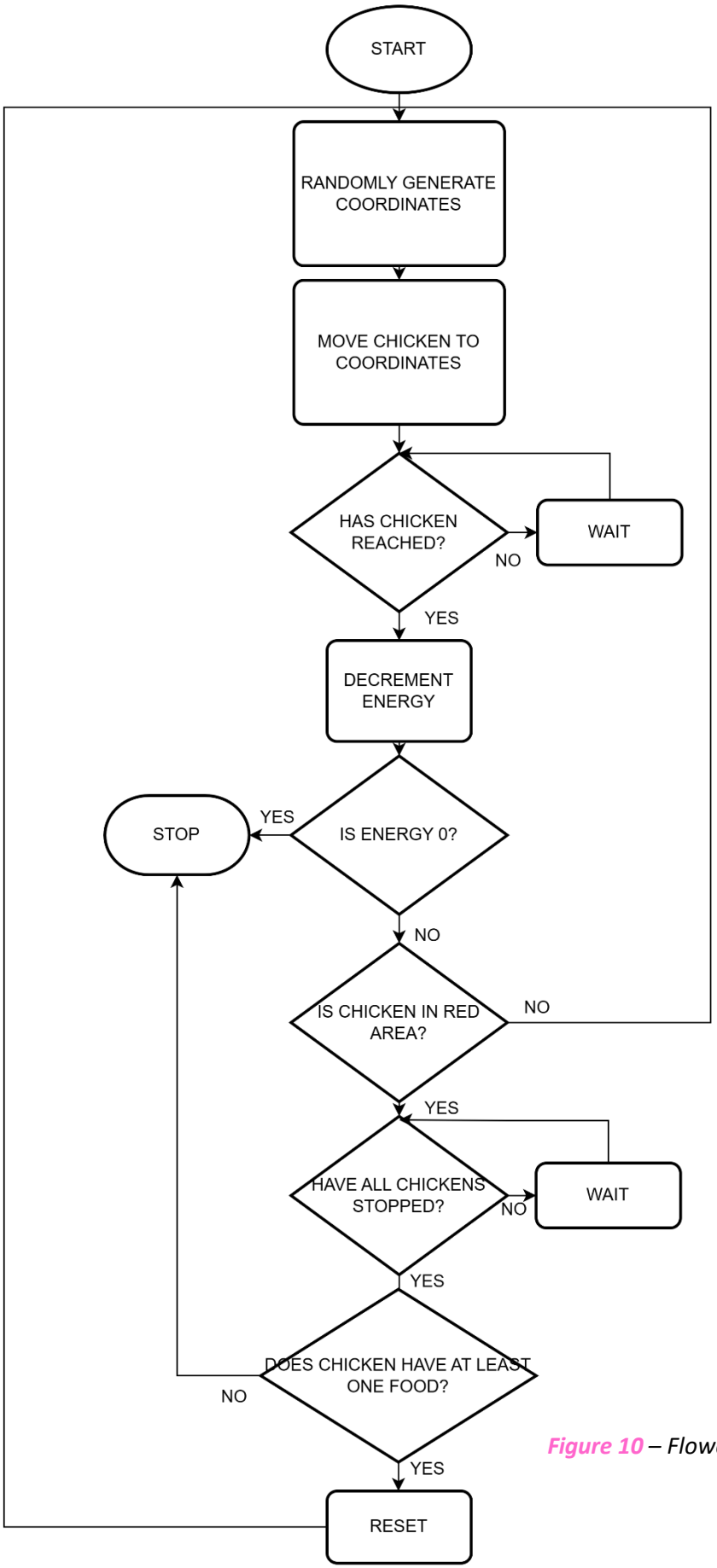


Figure 9 – Screenshot showing animation mapping of prey

→ MoveChicken

The chicken moves to random coordinates every time. Every time this happens, energy is lost. This serves to simulate real life. If the chicken is in a red area, it stops and waits for all chickens to stop moving too. After, the amount of food is checked per chicken. Depending on this, the chicken either dies, survives or reproduces.

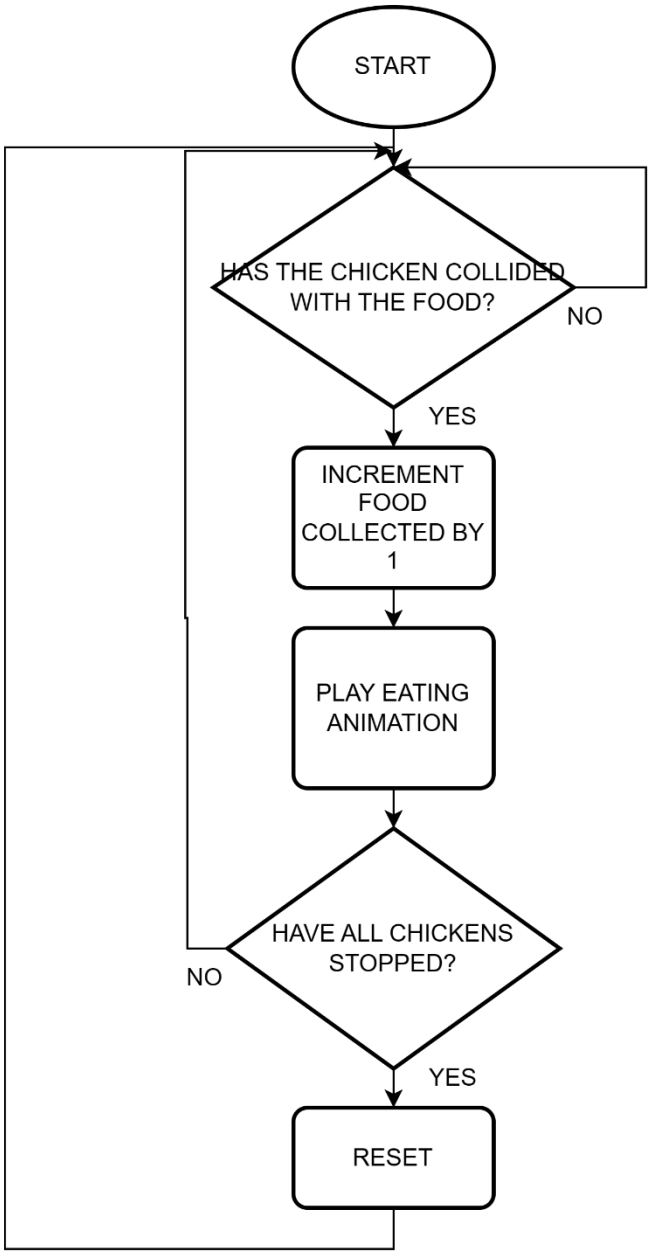


<- This relates to objectives **1**, **3** and **6**

Figure 10 – Flowchart of MoveChicken

→ FoodCollect

The chicken collects food as it moves across the plane, by colliding with the food object. This is stored separately for each chicken. After all chickens have stopped moving, the food for each is checked: 0 – dies, 1 – survives, 2+ - reproduces. The chickens that live on to the next generation have their amounts of food reset to 0 again.

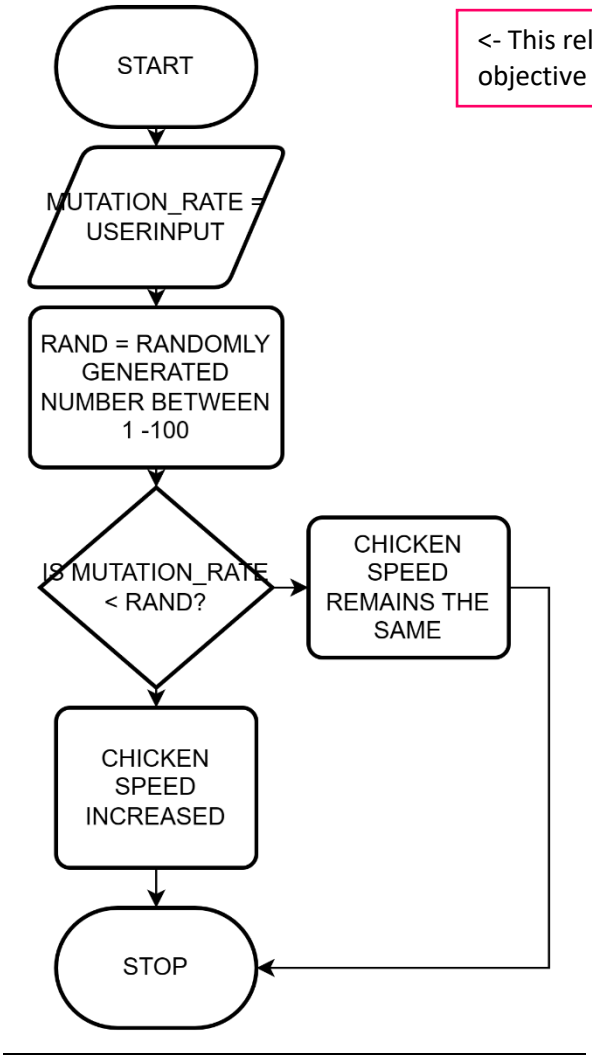


<- This relates to objective **2**

Figure 11 – Flowchart of FoodCollect

→ Mutations

The mutations will occur randomly, decided as the chickens spawn. The user will have the option to enter the mutation rate (between 0-100). The mutation then occurs randomly among the population. There is a speed mutation and an energy mutation (which follows the same process as below).



<- This relates to objective **7**

Figure 12 *– Flowchart showing how mutations occur*

PLANE DESIGN

The design of the plane is relatively simple, with 2 sections consisting of a plain white centre and a red border around the outside. If the chickens reach the red area, they stop. I wanted to make the plane a single colour so that the simulation wouldn't be confusing and hard to see – it would take the focus away from the actual simulation.

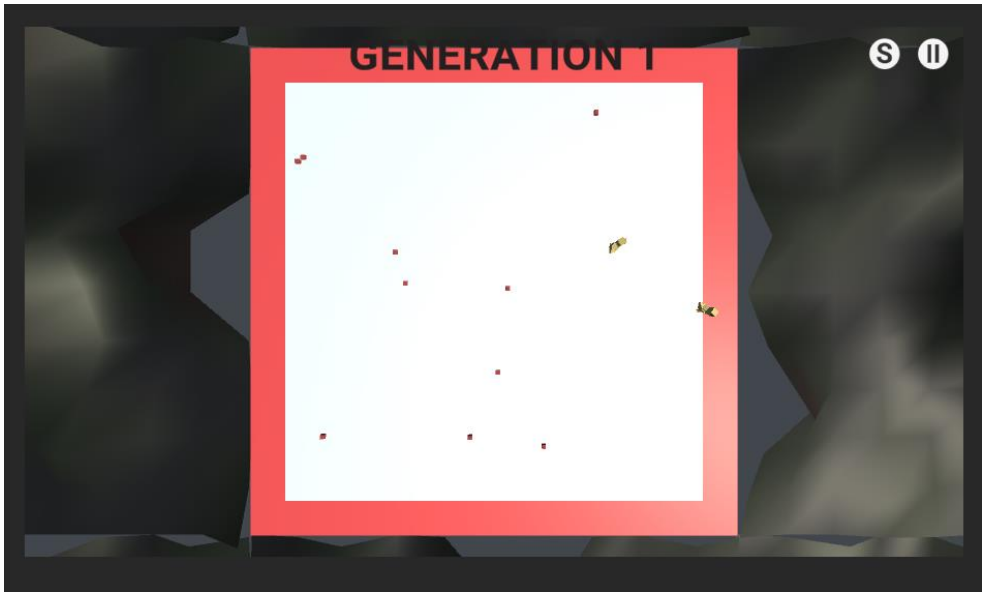
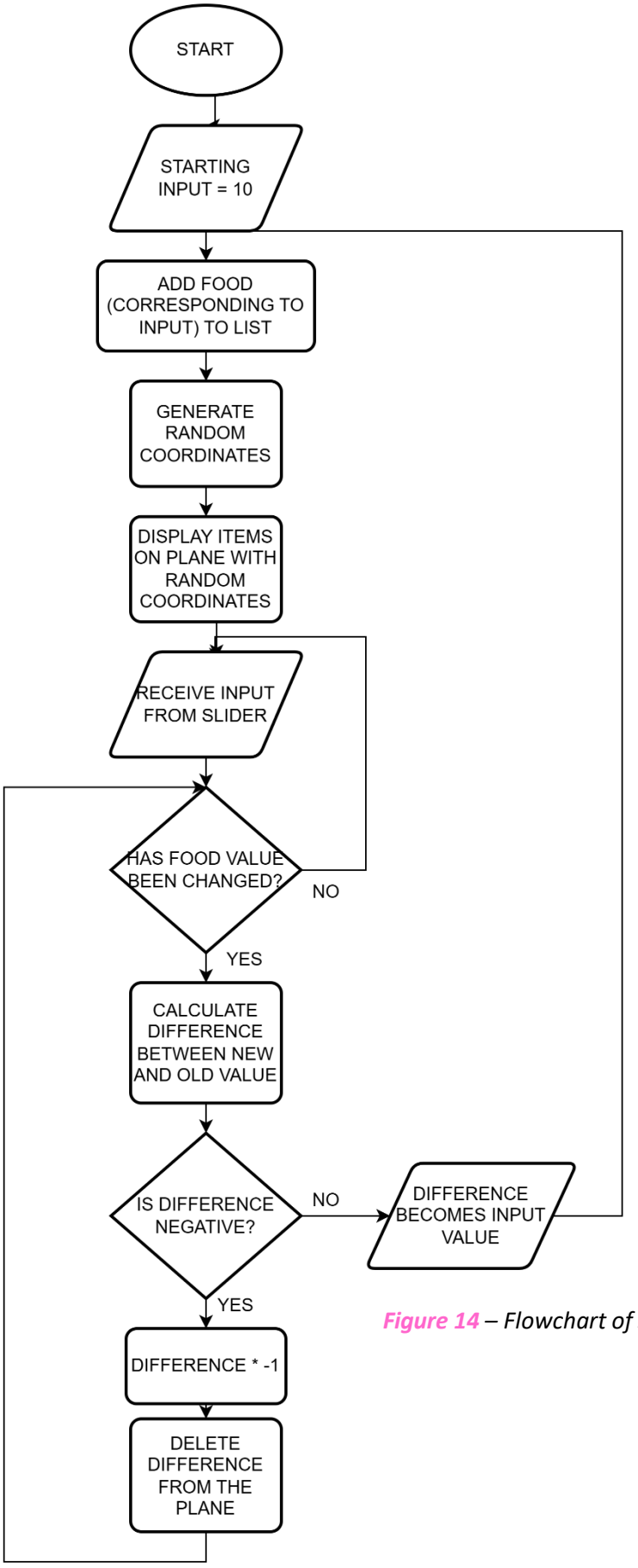


Figure 13 – Screenshot showing design of plane

→ SpawnFood

At the start of the simulation, the starting amount of food is displayed on the plane. Then, the slider determines the change in the food on the plane. If the slider is decreased, food is deleted, and if it is increased, the food is added. The food (like the chickens) is also added at random coordinates to avoid food being spawned on top of each other.



<- This relates to objective **2**

Figure 14 – Flowchart of SpawnFood

→ SpawnChicken

The starting input is the input that the user types in at the beginning of the simulation. This amount of chickens is added to the list and displayed on the plane. The chickens will die/get deleted if they run out of energy or don't have enough food at the end of a generation. If all the chickens are deleted, this will be reflected in the list, so the end screen will appear.

<- This relates to objectives **2**, **5** and **6**

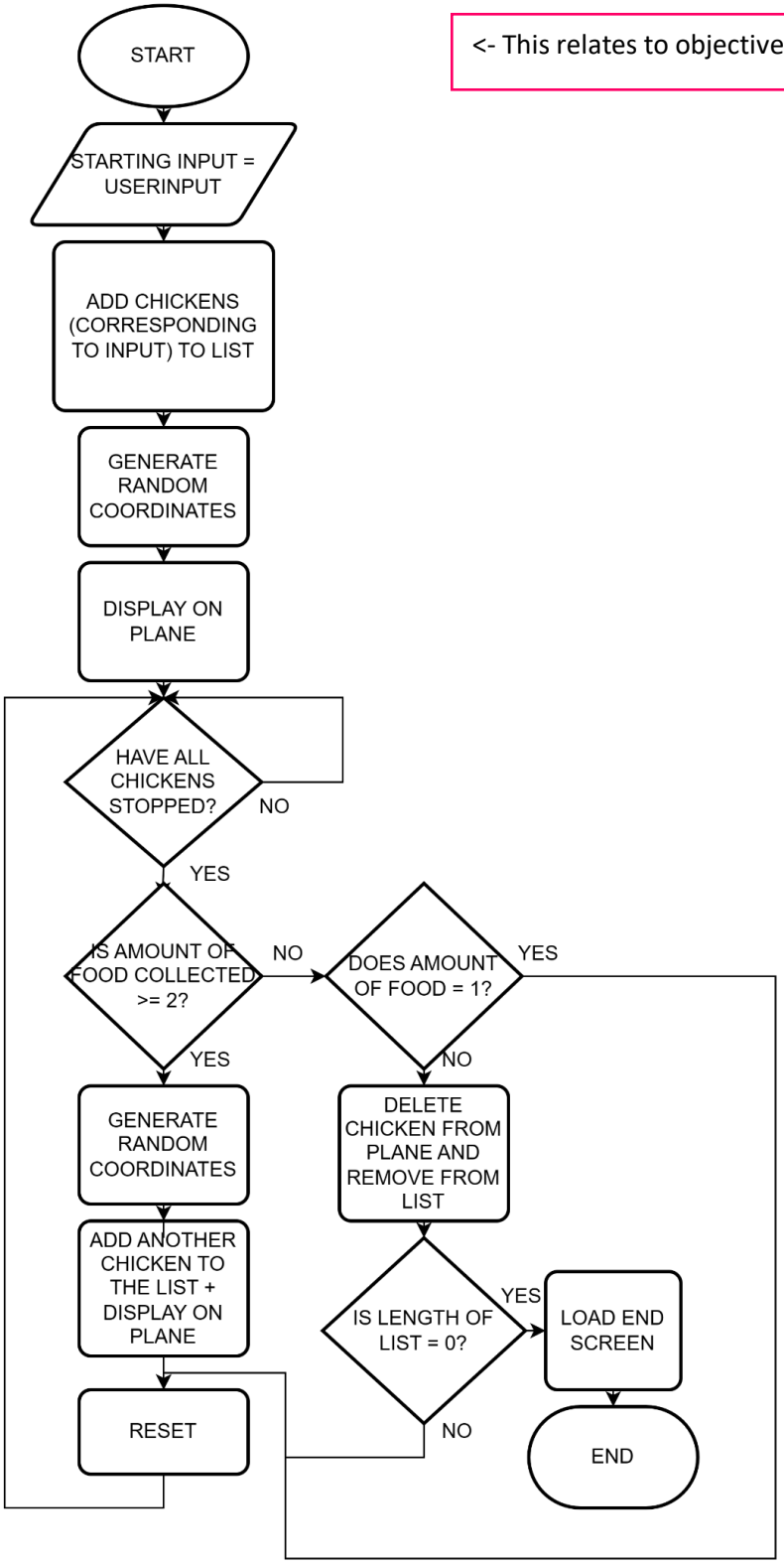
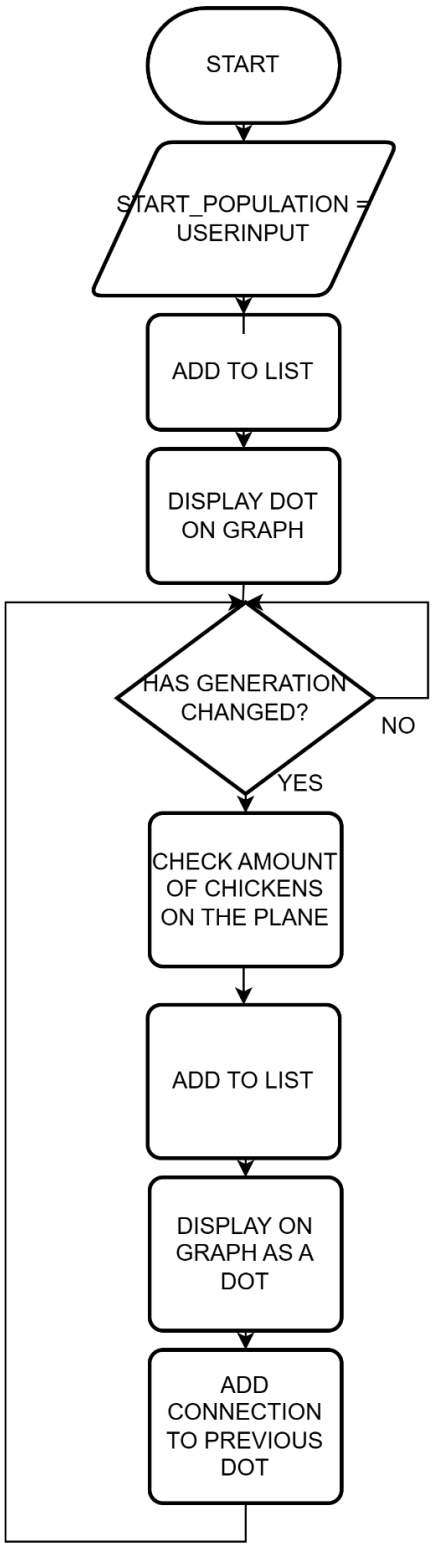


Figure 15 – Flowchart of SpawnChicken

→ WindowGraph

The graph is displayed in a separate area, where the points will be the population of prey at the beginning of each generation. The points will be connected via a line (line graph). The x axis will be dynamic – so that all of the generations will be displayed, and the trend can be seen.



<- This relates to objective **9**

Figure 16 – Flowchart showing how the graph is displayed

PREDATOR DESIGN

The predator I decided to use was a fox. This is because foxes are natural predators to chickens. Like the chicken, the predator is imported with animations that need to be mapped with triggers between states.

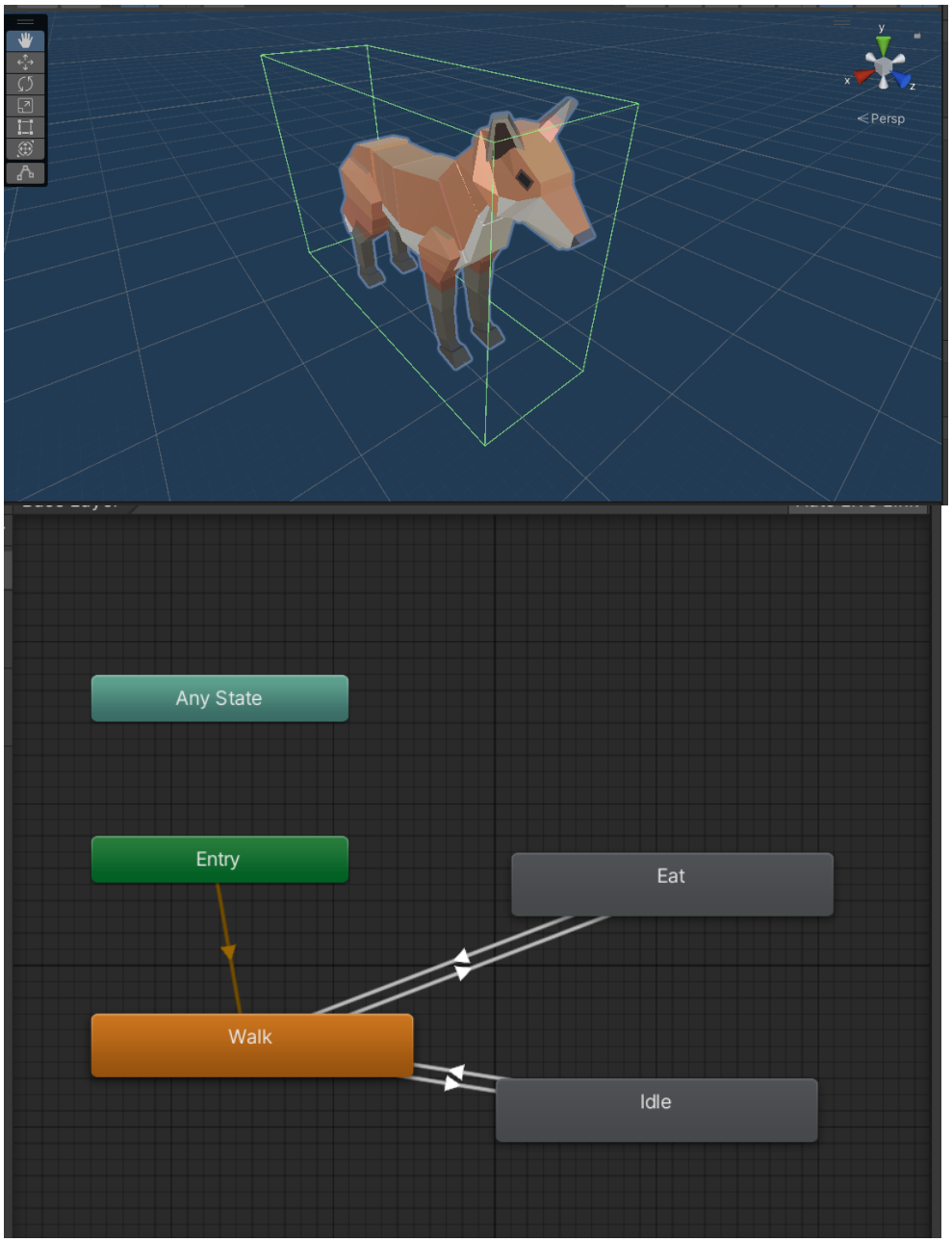


***Figure 17** – Screenshots to show the design and animation mapping of the fox predator*

→ SpawnPredator

The number of predators on the plane can be controlled via the settings page. Once the value is increased, more predators are spawned, if decreased, predators are deleted. The predators (like the chickens and food) are also added at random coordinates to avoid being spawned on top of each other.

<- This relates to objectives **2** and **8.2**

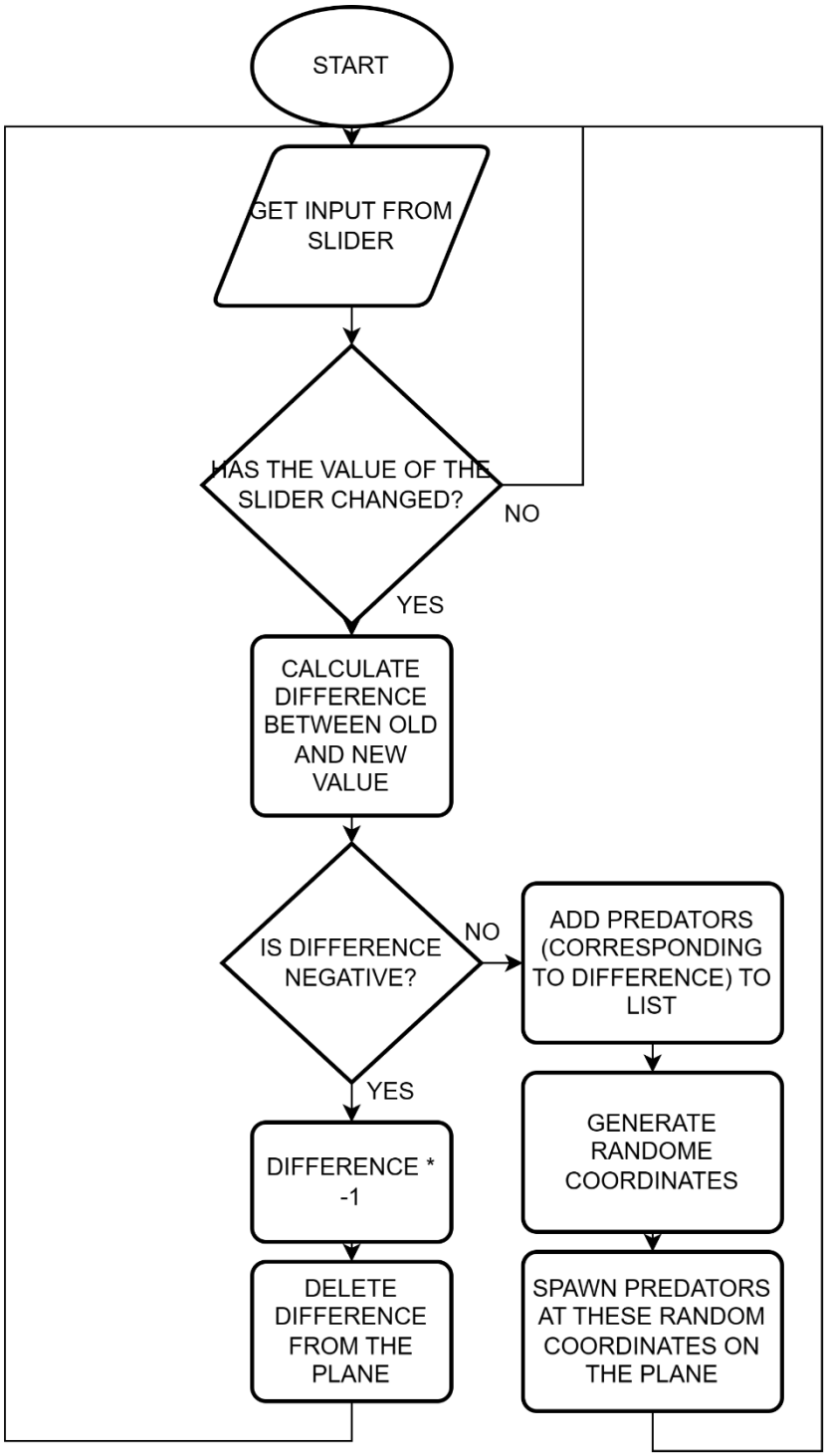
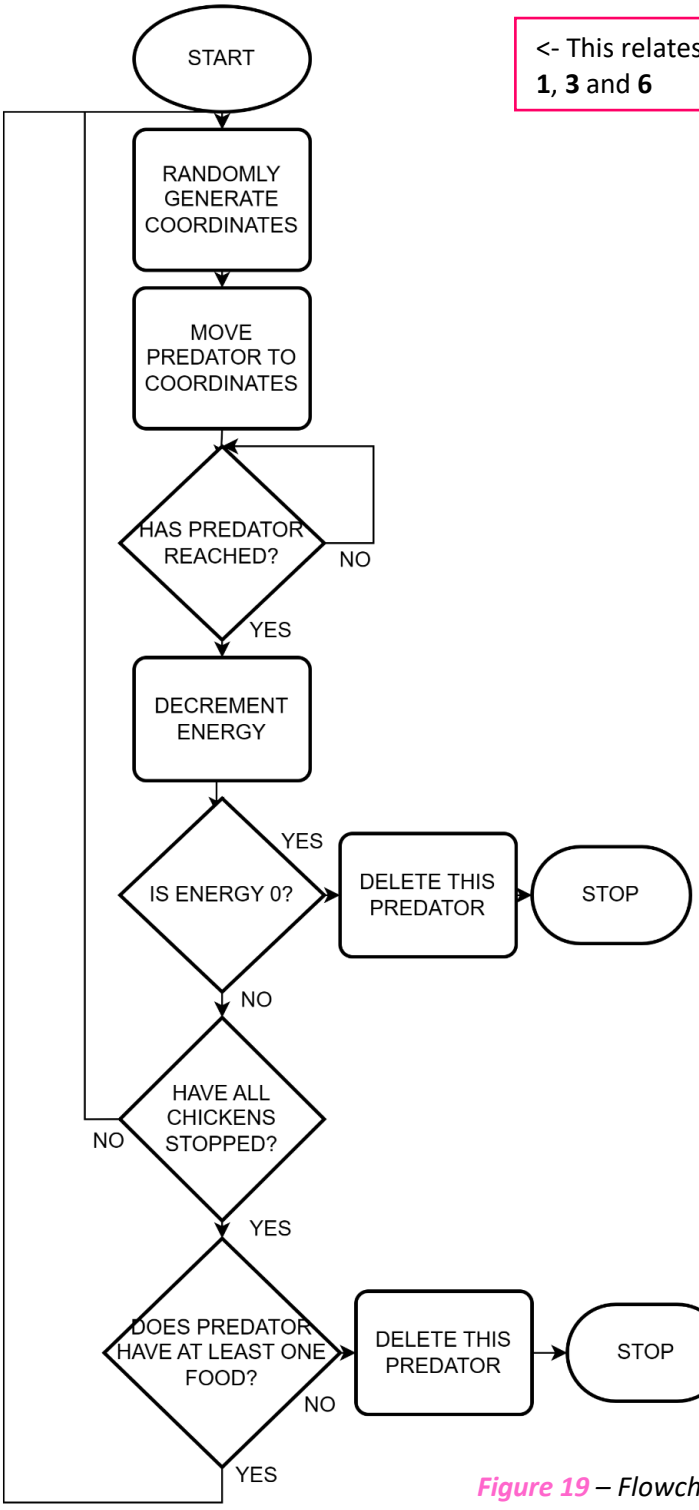


Figure 18 – Flowchart of SpawnPredator

→ MovePredator

The predator moves to random coordinates every time. Every time this happens, energy is lost, like the chickens. The predator cannot reach the red area, and if it collides with a chicken, it 'eats' it. The predators will not be able to reproduce, as the population of chickens is small, so it may die out too easily.

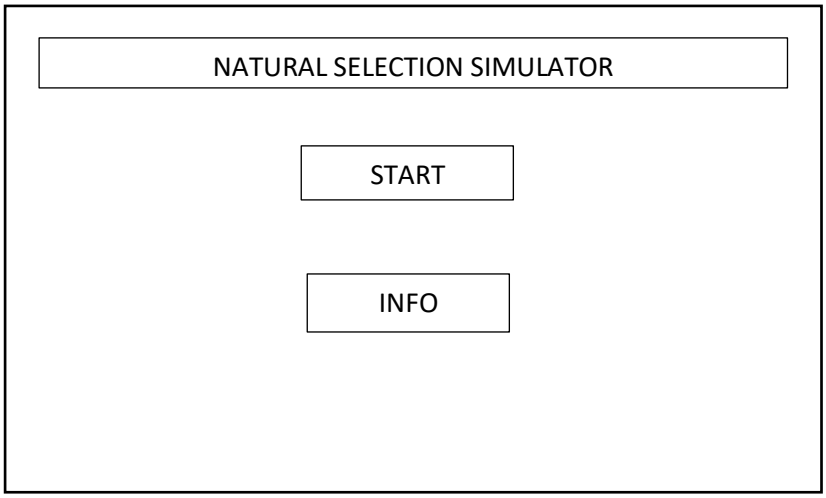


<- This relates to objectives **1**, **3** and **6**

Figure 19 – Flowchart of MovePredator

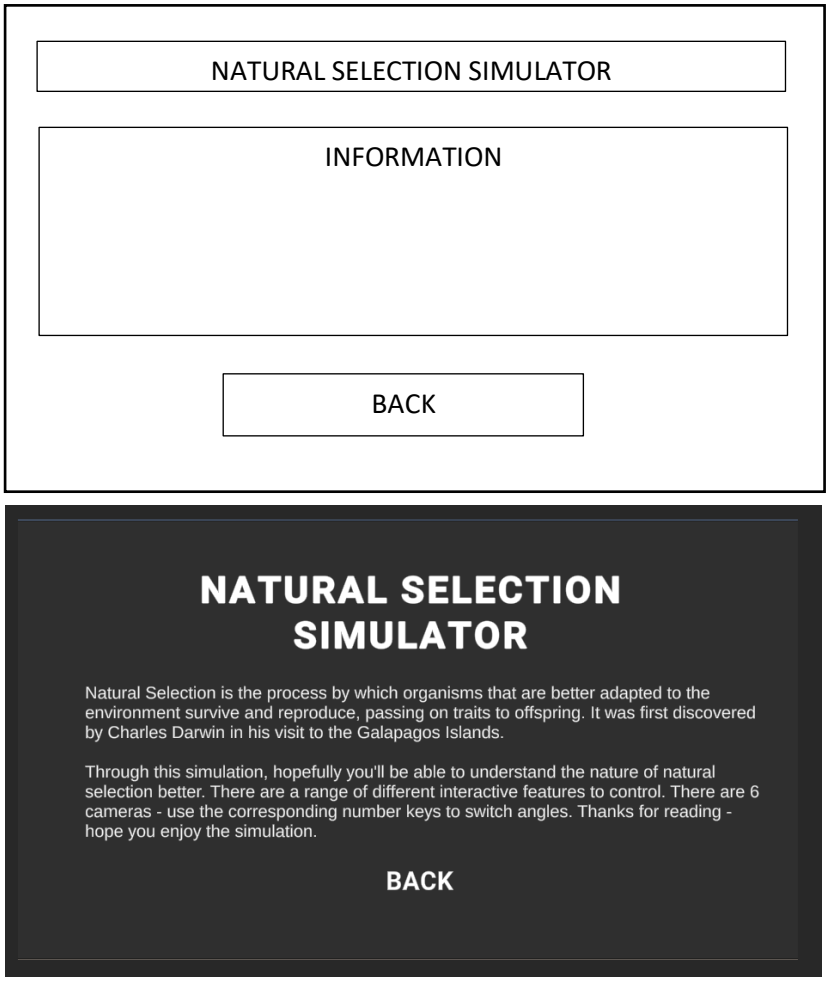
USER INTERFACE DESIGN

With the design of the user interface, it is necessary to have a clean, simple, and therefore easy to understand design. I chose the dark grey theme so that the white text would contrast and draw more attention. The home screen has 2 buttons. One to start the simulation and one to go to the information screen. I added an information screen, to clarify some information about natural selection in case it is needed. There are icons in the corner when the actual simulation plays – to avoid text covering any content.



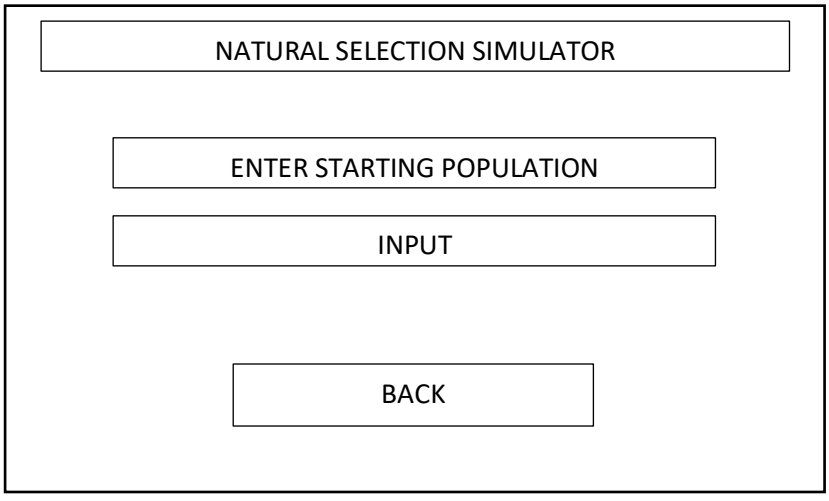
Home Screen – with the title of the program, the start button and the icon to go to the information screen.

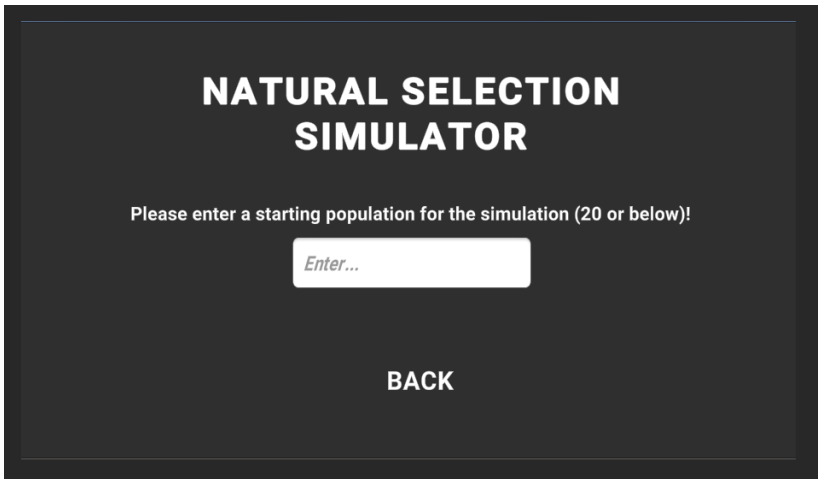
Figure 20 *– Screenshot and design showing the home screen*



Information Screen – when the icon is clicked, this screen appears. It contains information about the simulation and natural selection.

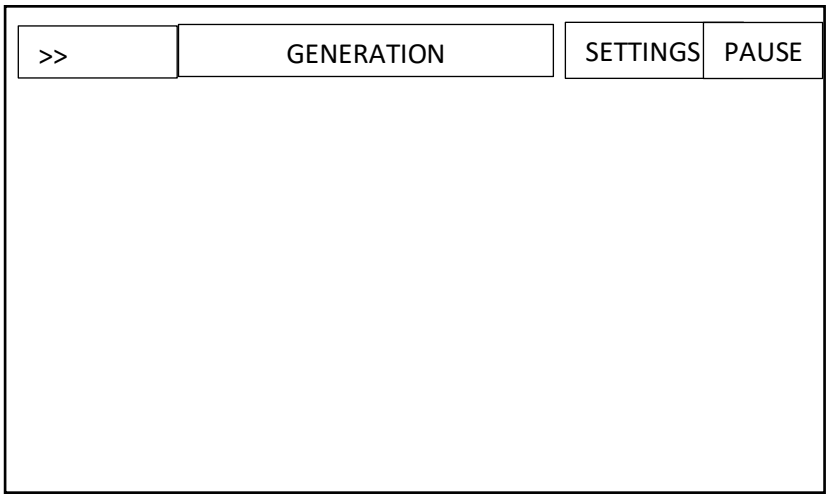
Figure 21 *– Screenshot and design showing the information screen*

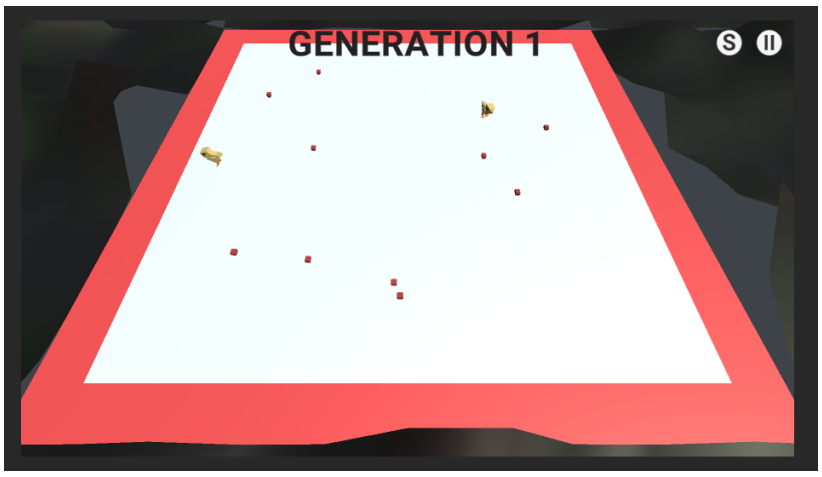




Play Screen – allows you to enter a starting population.

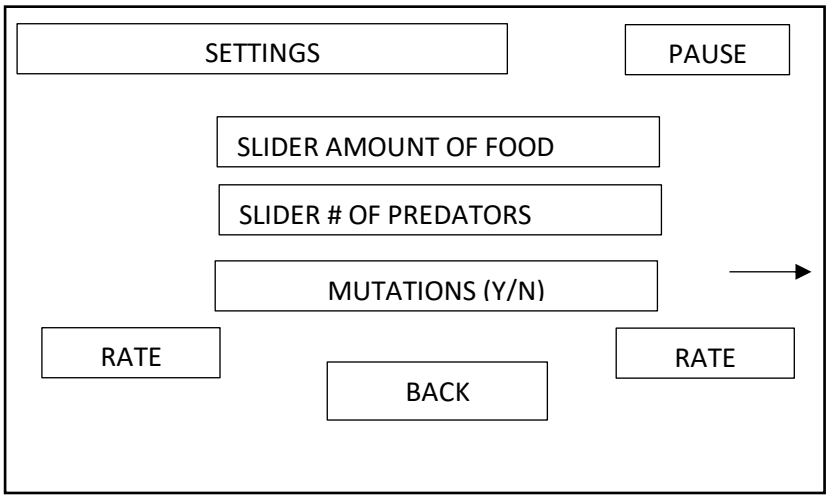
Figure 22 – Screenshot and design showing the play screen





Shows the interface when the simulation is played. The small icons in the corner represent settings, pause and fast forward buttons.

Figure 23 – Screenshot and design showing the interface of the simulation



<- This relates to objectives **8** and **10**

Settings Page – sliders can control the variables that have been implemented. The arrow leads to the graph page.

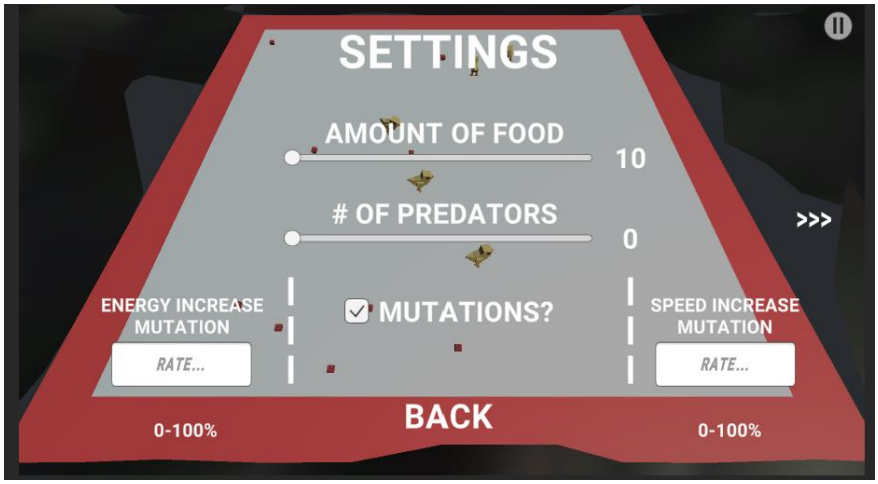
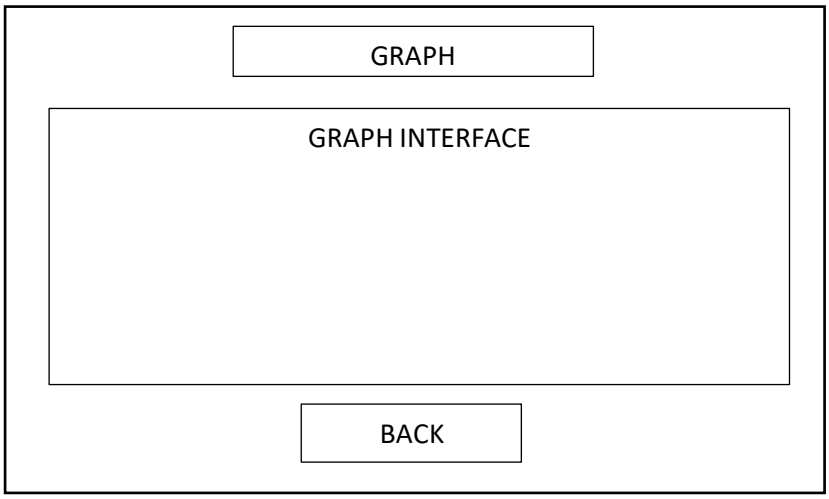


Figure 24 *– Screenshot and design showing the settings page*



<- This relates to objective **9**

Graph Page – shows a line graph of the population of the chickens. This updates as a new generation starts.

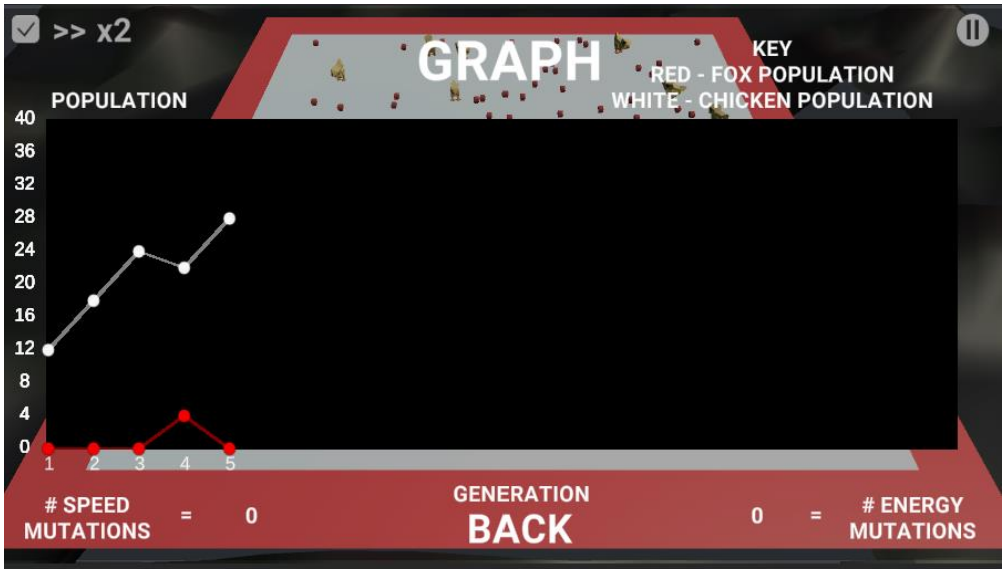
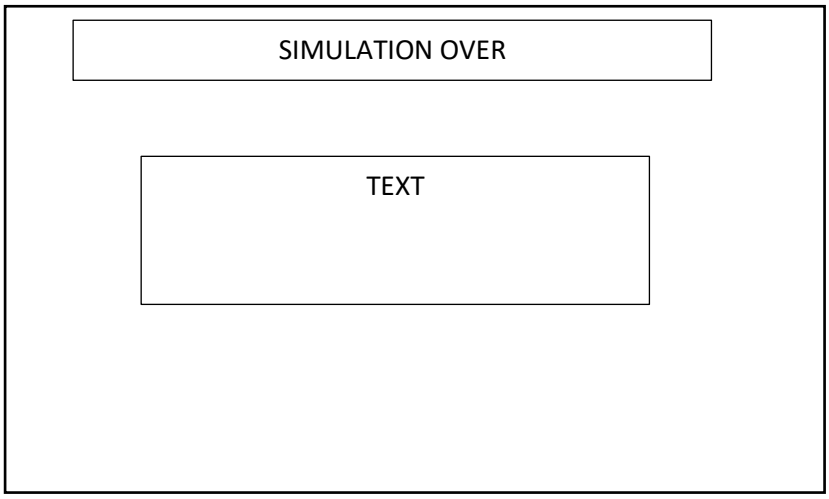


Figure 25 – Screenshot and design of graph screen (with values till generation 5)



End Screen – when all chickens die, this screen is shown.

Figure 26 - Screenshot and design showing the end screen

Technical Solution

FOODCOLLECT.CS

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FoodCollect : MonoBehaviour
{
   public int NumOfFood;

   public void FoodCollected()
   {
      NumOfFood++;
   }
}
```

FOODCOLLISION.CS

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FoodCollision : MonoBehaviour //this is attached to food prefab
{
   public static int collision = 0;


   private void OnTriggerEnter(Collider other) //if collision occurs
   {
      if ((other.tag == "Chicken") || (other.tag == "Speed") || (other.tag == "Energy") || (other.tag == "SE"))
      {
         FoodCollect foodCollect = other.GetComponent<FoodCollect>(); //foodcollected incremented and object destroyed
         if (foodCollect != null)
         {
            foodCollect.FoodCollected();
            Destroy(gameObject);
         }
      }

   }

}
```

MOVECHICKEN.CS

```
using System.Collections;
using System.Collections.Generic;
```

```
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
using TMPro;

public class MoveChicken : MonoBehaviour
{
   public float movementduration = 2.0f; //the time it takes for the chicken to move to the
coordinates generated
   private bool hasarrived = false; //if the chicken has arrived to its new coordinates
   private Animator anim;
   private float waitTime;
   public static int collision;
   public int FoodCollected;
   float randX;
   float randZ;
   private Rigidbody ChickenRigidbody;
   bool stop = false;
   bool redstart = false;
   bool allStopped = false;
   public static bool reproduce = false;
   public static bool done = false;
   private int energychicken = 20; //energy that the chicken has (change in reset too)
   public static float energymutationRate = 0;
   private int amount_of_energy = 20; //store for the amount of energy of the chicken
   public static bool reset = false;
   public static float mutationRate = 0;
   public static bool FastForward = false; //checks if fast forward toggle is on
   public static int ToggleOn = 0;

   // Start is called before the first frame update
   void Start()
   {
      anim = GetComponent<Animator>();
      this.ChickenRigidbody = gameObject.GetComponent<Rigidbody>();
      mutationRate = PauseResume.mutationRate; //mutations occur as a new chicken is spawned
      if (mutationRate != 0) //speed mutation
      {
         int temp = Random.Range(1, 101); //random number between 1-100
         if (temp <= mutationRate) //if the random number is less than/equal to the mutation rate
         {
            movementduration = 1.0f; //essentially, randomising to see if mutation occurs based on the
rate
            this.gameObject.tag = "Speed";
         }
      }
      energymutationRate = PauseResume.energymutationRate;
      if (energymutationRate != 0) //energy mutation
      {
         int temp = Random.Range(1, 101); //same as the speed mutation
         if (temp <= energymutationRate)
```

```
        {
            amount_of_energy = 30;
            this.energychicken = amount_of_energy;
            this.gameObject.tag = "Energy";
        }
    }
    if ((movementduration == 1.0f) && (amount_of_energy == 30)) //checks whether both
mutations have been applied
    {
        this.gameObject.tag = "SE";
    }
    FastForward = PauseResume.FastForward;
    if (FastForward == true) //if fast forward toggle is on
    {
        movementduration = movementduration * 0.5f;
    }
  }

  public IEnumerator RedStart()
  {
    yield return new WaitForSeconds(15.0f); //amount of time before chickens can go into red area
    redstart = true;
  }

  public void ChangeSpeed()
  {

    ToggleOn = PauseResume.ToggleOn; //checks fast forward toggle (has 3 states - 0 = off, 1 = on, 2
= off (after it was on))
    if (ToggleOn == 1) //when on
    {
        if (this.gameObject.tag != "Speed") //checks if speed mutation applies - if yes then movement
duration will be lower than 1
        {
            if (movementduration > 1.0f) //increases the speed
            {
                movementduration = movementduration * 0.5f;
                ToggleOn = PauseResume.ToggleOn;
            }
        }
        else
        {
            movementduration = movementduration * 0.5f;
            ToggleOn = PauseResume.ToggleOn;
        }


    }
    else if (ToggleOn == 2) //when off
    {
        if (this.gameObject.tag != "Speed")
```

```
        {
            if (movementduration < 2.0f) //slows down the speed again
            {
                movementduration = movementduration * 2.0f;
                ToggleOn = PauseResume.ToggleOn;
            }
        }
        else
        {
            movementduration = movementduration * 2.0f;
            ToggleOn = PauseResume.ToggleOn;
        }

    }
    ToggleOn = PauseResume.ToggleOn;
}

public IEnumerator Wait()
{
    yield return new WaitForSeconds(1.0f); //check if this time is correct!!
    this.anim.SetTrigger("GoWalk"); //walk animation

}

// Update is called once per frame
void Update()
{
    if (redstart == false)
    {
        StartCoroutine(RedStart());
    }

    HasArrived();
    AllStopped();
    ToggleOn = PauseResume.ToggleOn;
    if (ToggleOn != 0) //if the fast forward toggle is changed
    {
        ChangeSpeed();
        ToggleOn = PauseResume.ToggleOn;
    }

}
private IEnumerator MoveToPoint(Vector3 targetPos)
{
    CheckRed();
    float timer = 0.0f; //timer initialised
    Vector3 startPos = transform.position; //startPos is the initial coordinates
    while (timer < movementduration) //while the timer is less than the movement duration
    {
        timer += Time.deltaTime; //timer incremented by change in time
        float t = timer / movementduration; //timer divided by the movement duration
```

```
        t = t * t * t * (t * (6f * t - 15f) + 10f);
        Vector3 directionpos = targetPos - startPos; //to work out position vector for the rotation distance
        transform.position = Vector3.Lerp(startPos, targetPos, t); //makes chicken move to new coordinates
        transform.rotation = Quaternion.Slerp(transform.rotation, Quaternion.LookRotation(directionpos), Time.deltaTime * 5f); //makes chicken rotate in direction of movement
        yield return null;


    }
    waitTime = Random.Range(0.0f, 2.0f); //idle for a random amount of time
    anim.SetTrigger("GoIdle");
    yield return new WaitForSeconds(waitTime);
    hasarrived = false;
    this.energychicken = this.energychicken - 1; //decrements energy
    if (this.energychicken <= 0)
    {
      Debug.Log("run out of energy");
      Destroy(this.gameObject); //when energy reaches 0, the chicken is destroyed
      StartCoroutine(Waiting());

    }
  }


  private void HasArrived()
  {

    if (done == true) //when reproduction has finished
    {
      hasarrived = false;
      stop = true;
    }

    if (stop == false) //hasn't stopped
    {
      if (!hasarrived)
      {

        hasarrived = true; //checks every frame whether the chicken has arrived
        if (redstart == false)
        {
          //Debug.Log("false");
          randX = Random.Range(-140.0f, 140.0f); //coordinates generated
          randZ = Random.Range(-140.0f, 140.0f); //140 w/o red area
          while (Vector3.Distance(transform.position, new Vector3(randX, 0.0f, randZ)) >= 50.0f || (Vector3.Distance(transform.position, new Vector3(randX, 0.0f, randZ)) <= 10.0f)) //makes sure the distance is within 10-50
          {
```

```
                randX = Random.Range(-140.0f, 140.0f); //if not, new coordinates are generated
                randZ = Random.Range(-140.0f, 140.0f);
            }
            //anim.speed = 1;
            anim.SetTrigger("GoWalk");
            StartCoroutine(MoveToPoint(new Vector3(randX, 0.0f, randZ))); //with the new
coordinates

        }
        else if (redstart == true)
        {

            randX = Random.Range(-170.0f, 170.0f); //coordinates generated
            randZ = Random.Range(-170.0f, 170.0f); //140 w/o red area
            while (Vector3.Distance(transform.position, new Vector3(randX, 0.0f, randZ)) >= 50.0f ||
(Vector3.Distance(transform.position, new Vector3(randX, 0.0f, randZ)) <= 10.0f)) //makes sure the
distance is within 10-50
            {
                randX = Random.Range(-170.0f, 170.0f); //if not, new coordinates are generated
                randZ = Random.Range(-170.0f, 170.0f);
            }

            anim.SetTrigger("GoWalk");
            StartCoroutine(MoveToPoint(new Vector3(randX, 0.0f, randZ))); //with the new
coordinates

        }


      }
    }
  }
  private void OnTriggerEnter(Collider other) //if it collides with food...
  {
    if (other.tag == "Food")
    {
      this.anim.SetTrigger("GoEat"); //eat animation
      this.FoodCollected++; //specific to each chicken
      this.energychicken++;
      StartCoroutine(Wait());
    }
    else if (other.tag == "Predator") //if predator collides with the chicken, it is destroyed
    {
      Destroy(this.gameObject);
      StartCoroutine(Waiting());
    }

  }
```

```
  private void CheckRed()
  {
    if ((((randZ >= 150) && (randZ <= 170)) || ((randZ <= -150) && (randZ >= -170))) || ((randX >=
150) && (randX <= 170)) || ((randX <= -150) && (randX >= -170))) //coordinates of the red area
(150/153)
    {

      this.ChickenRigidbody.velocity = Vector3.zero; //stops
      stop = true;
      this.anim.SetTrigger("GoIdle");
    }
  }



  public void AllStopped() //when all the chickens have stopped, this is called
  {
    allStopped = SpawnChicken.allStopped;
    if (this.gameObject != null)
    {
      if (allStopped == true)
      {
        stop = true;
        hasarrived = true;
        if (this.FoodCollected == 0) //destroyed if no food collected
        {
          Destroy(this.gameObject);
          StartCoroutine(Waiting());
        }
        else if (this.FoodCollected >= 2) //reproduces if 2 or more are collected
        {
          reproduce = true;
          done = SpawnChicken.done;
          if (done == false) //done - true when chicken has reproduced
          {
            done = SpawnChicken.done;
          }
          if (done == true)
          {
            reproduce = false;
          }
          reset = true; //the reset variable is changed

        }
        reset = true;
        StartCoroutine(Reset());
      }

    }
```

```
    }

    public IEnumerator Reset()
    {
        if (this.gameObject != null) //resets all the variables
        {
            yield return new WaitForSeconds(0.001f);
            Debug.Log("resetting");
            reset = false;
            allStopped = SpawnChicken.allStopped;
            hasarrived = false;
            stop = false;
            reproduce = false;
            redstart = false;
            done = SpawnChicken.done;
            this.FoodCollected = 0;
            energychicken = amount_of_energy;
            yield return new WaitForSeconds(0.001f);
            HasArrived();
        }


    }
    private IEnumerator Waiting() //time delay
    {
        yield return new WaitForSeconds(0.001f);
    }
}
```

MOVEPREDATOR.CS

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MovePredator : MonoBehaviour
{
    private Animator anim;
    private Rigidbody PredatorRigidbody;
    private bool hasarrived = false;
    float randX;
    float randZ;
    private float movementduration = 2.0f;
    private float waitTime;
    private int energypredator = 15; //30 for chickens (15)
    public int FoodCollected = 0;
    public static int ToggleOn = 0;

    // Start is called before the first frame update
    void Start()
    {
        anim = GetComponent<Animator>();
```

```
        this.PredatorRigidbody = gameObject.GetComponent<Rigidbody>();
    }

    private IEnumerator MoveToPoint(Vector3 targetPos)
    {
        float timer = 0.0f; //timer initialised
        Vector3 startPos = transform.position; //startPos is the initial coordinates
        while (timer < movementduration) //while the timer is less than the movement duration
        {
            timer += Time.deltaTime; //timer incremented by change in time
            float t = timer / movementduration; //timer divided by the movement duration
            t = t * t * t * (t * (6f * t - 15f) + 10f);
            Vector3 directionpos = targetPos - startPos; //to work out position vector for the rotation
distance
            transform.position = Vector3.Lerp(startPos, targetPos, t); //makes chicken move to new
coordinates
            transform.rotation = Quaternion.Slerp(transform.rotation,
Quaternion.LookRotation(directionpos), Time.deltaTime * 5f); //makes chicken rotate in direction of
movement
            yield return null;

        }
        waitTime = Random.Range(0.0f, 2.0f); //idle for a random amount of time
        anim.SetTrigger("GoIdle");
        yield return new WaitForSeconds(waitTime);
        hasarrived = false;
    }
    public IEnumerator Wait()
    {
        yield return new WaitForSeconds(1.0f);
        this.anim.SetTrigger("GoWalk");
    }

    private IEnumerator Waiting()
    {
        yield return new WaitForSeconds(0.001f);
    }

    public void ChangeSpeed() //checks fast forward toggle (has 3 states - 0 = off, 1 = on, 2 = off (after
it was on))
    {
        ToggleOn = PauseResume.ToggleOn;
        if (ToggleOn == 1) //when on
        {
            if (movementduration > 1.0f) //checks if speed mutation applies - if yes then movement
duration will be lower than 1
            {
                movementduration = movementduration * 0.5f;  //increases the speed
                ToggleOn = PauseResume.ToggleOn;
            }
        }
```

```
    else if (ToggleOn == 2) //when off
    {
        if (movementduration < 2.0f)
        {
            movementduration = movementduration * 2.0f; //increases speed again
            ToggleOn = PauseResume.ToggleOn;
        }

    }
    ToggleOn = PauseResume.ToggleOn;
}

// Update is called once per frame
void Update()
{
    HasArrived();
    ToggleOn = PauseResume.ToggleOn;
    if (ToggleOn != 0)
    {
        ChangeSpeed();
    }
}

private void HasArrived()
{
    if (!hasarrived)
    {

        hasarrived = true; //checks every frame whether the chicken has arrived
        randX = Random.Range(-140.0f, 140.0f); //coordinates generated
        randZ = Random.Range(-140.0f, 140.0f); //140 w/o red area
        while (Vector3.Distance(transform.position, new Vector3(randX, 0.0f, randZ)) >= 50.0f ||
(Vector3.Distance(transform.position, new Vector3(randX, 0.0f, randZ)) <= 10.0f)) //makes sure the
distance is within 10-50
        {
            randX = Random.Range(-140.0f, 140.0f); //if not, new coordinates are generated
            randZ = Random.Range(-140.0f, 140.0f);
        }
        anim.SetTrigger("GoWalk");
        StartCoroutine(MoveToPoint(new Vector3(randX, 0.0f, randZ))); //with the new coordinates
        this.energypredator = this.energypredator - 1;
        if (this.energypredator <= 0)
        {
            Debug.Log("run out of energy");
            Destroy(this.gameObject);
            StartCoroutine(Waiting());

        }
    }
}
private void OnTriggerEnter(Collider other) //if it collides
```

```
    {
        if ((other.tag == "Chicken") || (other.tag == "Speed") || (other.tag == "Energy") ||(other.tag ==
"SE"))
        {
            this.anim.SetTrigger("GoEat");
            this.FoodCollected++;
            energypredator = energypredator + 5; //adds 5 to energy every time a chicken is eaten
            StartCoroutine(Wait());
        }

    }
}
```

NOISEMAPGENERATION.CS

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class NoiseMapGeneration : MonoBehaviour
{
    public float[,] GenerateNoiseMap(int mapDepth, int mapWidth, float scale, float offsetX, float
offsetZ)
    {
        // create an empty noise map with the mapDepth and mapWidth coordinates
        float[,] noiseMap = new float[mapDepth, mapWidth];
        for (int zIndex = 0; zIndex < mapDepth; zIndex++)
        {
            for (int xIndex = 0; xIndex < mapWidth; xIndex++)
            {
                // calculate sample indices based on the coordinates and the scale
                float sampleX = (xIndex + offsetX) / scale;
                float sampleZ = (zIndex + offsetZ) / scale;
                // generate noise value using PerlinNoise
                float noise = Mathf.PerlinNoise(sampleX, sampleZ);
                noiseMap[zIndex, xIndex] = noise;
            }
        }
        return noiseMap;
    }
}
```

PAUSERESUME.CS

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class PauseResume : MonoBehaviour
{
    public GameObject PauseScreen;
```

```
public GameObject PauseButton;
public GameObject SettingsButton;
public GameObject SettingsScreen;
public GameObject GraphScreen;
public Toggle MutationToggle;
public GameObject SpeedMutation; //inputfield
public GameObject EnergyMutation;//inputfield
public Slider slider;
public Slider predatorslider;
public float amountFood;
public static float PredNumber;
public static bool GamePaused;
public static bool reset;
public static int gennumber = 1;
public int predatornumber;
public static bool sliderchange = false;
public static float mutationRate = 0;
public static float energymutationRate = 0;
private int num_speedmutation = 0;
private int num_energymutation = 0;
private int num_bothmutation = 0;

[SerializeField]
private TextMeshProUGUI textgeneration;

[SerializeField]
private TextMeshProUGUI text; //text to show value of slider

[SerializeField]
private TextMeshProUGUI predatortext; //text to show value of slider

[SerializeField]
private TextMeshProUGUI inputfielddisplay_text;

[SerializeField]
private TMP_InputField input_field;

[SerializeField]
private TextMeshProUGUI inputfielddisplay2_text;

[SerializeField]
private TMP_InputField input_field2;

[SerializeField]
private TextMeshProUGUI speed_num;

[SerializeField]
private TextMeshProUGUI energy_num;

public Toggle SpeedToggle;
[SerializeField]
```

```
    private TextMeshProUGUI speed_text;
    public static int ToggleOn = 0;
    public static bool FastForward = false;

    // Start is called before the first frame update
    void Start()
    {
        PauseScreen.SetActive(false); //initially you can see the buttons but not their corresponsding
screens
        PauseButton.SetActive(true);
        SettingsButton.SetActive(true);
        SettingsScreen.SetActive(false);
        GraphScreen.SetActive(false);
        SpeedMutation.SetActive(false);
        EnergyMutation.SetActive(false);
        GamePaused = false; //the game is not paused
        slider.minValue = 10;
        slider.maxValue = 100; //food slider
        predatorslider.minValue = 0;
        predatorslider.maxValue = 10;
    }

    // Update is called once per frame
    void Update()
    {
        if (GamePaused) //controls the pausing of the game
        {
            Time.timeScale = 0;
        }
        else
        {
            Time.timeScale = 1;
        }
        reset = MoveChicken.reset;

        if (reset == true)
        {
            reset = false;
            StartCoroutine(Waiting());
        }

       if (SpeedToggle.isOn) //if the fast forward toggle is on
       {
           FastForward = true;
       }
       else
       {
           FastForward = false;
       }
```

```
      predatornumber = GameObject.FindGameObjectsWithTag("Predator").Length; //checks amount
of predators on the plane
      sliderchange = SpawnPredator.ChangeSlider;
      if (sliderchange == true) //checks against value of slider and changes it accordingly
      {
         predatorslider.value = predatornumber;
         predatortext.text = predatornumber.ToString();
         sliderchange = false;
      }
      sliderchange = SpawnPredator.ChangeSlider;

      num_speedmutation = GameObject.FindGameObjectsWithTag("Speed").Length; //checks how
many speed and energy mutations have occured
      num_energymutation = GameObject.FindGameObjectsWithTag("Energy").Length;
      num_bothmutation = GameObject.FindGameObjectsWithTag("SE").Length;
      speed_num.text = (num_speedmutation + num_bothmutation).ToString(); //displays them
      energy_num.text = (num_energymutation + num_bothmutation).ToString();
   }
   public IEnumerator Waiting()
   {
      yield return new WaitForSeconds(0.001f);
      reset = MoveChicken.reset;
      gennumber++;
      textgeneration.text = "GENERATION " + gennumber; //increments the gen number in the
simulation

   }

   public void PauseGame() //if the pause button is clicked...
   {
      GamePaused = true;
      PauseScreen.SetActive(true);
      PauseButton.SetActive(false);
      SettingsScreen.SetActive(false);
      SettingsButton.SetActive(true);
      textgeneration.enabled = false;
      GraphScreen.SetActive(false);
   }

   public void ResumeGame() //if the resume button is clicked...
   {
      GamePaused = false;
      PauseScreen.SetActive(false);
      PauseButton.SetActive(true);
      SettingsScreen.SetActive(false);
      SettingsButton.SetActive(true);
      textgeneration.enabled = true;
      GraphScreen.SetActive(false);
   }

   public void SettingsPage() //if the settings button is clicked...
```

```
{
    GamePaused = true;
    PauseScreen.SetActive(false);
    PauseButton.SetActive(true);
    SettingsButton.SetActive(false);
    SettingsScreen.SetActive(true);
    textgeneration.enabled = false;
    GraphScreen.SetActive(false);
}

public void MainScreen() //if the back button is clicked (on settings page)...
{
    GamePaused = false;
    PauseScreen.SetActive(false);
    PauseButton.SetActive(true);
    SettingsScreen.SetActive(false);
    SettingsButton.SetActive(true);
    textgeneration.enabled = true;
    GraphScreen.SetActive(false);
}

public void GraphPage() //if graph button is pressed
{
    GamePaused = true;
    PauseScreen.SetActive(false);
    PauseButton.SetActive(true);
    SettingsButton.SetActive(false);
    SettingsScreen.SetActive(false);
    GraphScreen.SetActive(true);
    textgeneration.enabled = false;
}

public void ChangeToggle() //if mutation toggle is pressed
{

    if (MutationToggle.isOn) //displays input fields
    {
        SpeedMutation.SetActive(true);
        EnergyMutation.SetActive(true);
    }
    else
    {
        SpeedMutation.SetActive(false);
        EnergyMutation.SetActive(false);
    }

}

public IEnumerator ChangeSpeed() //affects the toggle (0 = off, 1 = on, 2 = off after it has been on)
{
    if (SpeedToggle.isOn)
```

```
    {
        ToggleOn = 1;
        speed_text.text = "x2";
        yield return new WaitForSeconds(0.0001f);
        ToggleOn = 0;
    }
    else if (!SpeedToggle.isOn)
    {
        ToggleOn = 2;
        speed_text.text = "x1";
        yield return new WaitForSeconds(0.0001f);
        ToggleOn = 0;
    }
    ToggleOn = 0;
}

public void ChangeSpeed1()
{
    StartCoroutine(ChangeSpeed());
}

public void GetMutationRate() //gets input of speed mutation
{
    inputfielddisplay_text.text = "0-100%";
    Debug.Log(input_field.text);
    try
    {
        int temp = int.Parse(input_field.text);
        if ((temp < 0) || (temp > 100))
        {
            inputfielddisplay_text.text = "try again!"; //error handling when putting in mutation rates
            input_field.text = "";
        }
        else if ((temp >= 0) && (temp <= 100))
        {
            inputfielddisplay_text.text = "accepted - change will start in next gen";
            mutationRate = temp;
        }
    }
    catch
    {
        inputfielddisplay_text.text = "whole numbers only!";
        input_field.text = "";
    }

}

public void GetEnergyMutationRate() //gets input of energy mutation
{
    inputfielddisplay2_text.text = "0-100%";
    Debug.Log(input_field2.text);
```

```
        try
        {
            int temp = int.Parse(input_field2.text);
            if ((temp < 0) || (temp > 100))
            {
                inputfielddisplay2_text.text = "try again!"; //error handling when putting in mutation rates
                input_field2.text = "";
            }
            else if ((temp >= 0) && (temp <= 100))
            {
                inputfielddisplay2_text.text = "accepted - change will start in next gen";
                energymutationRate = temp;
            }
        }
        catch
        {
            inputfielddisplay2_text.text = "whole numbers only!";
            input_field2.text = "";
        }

    }

    public void ChangeFood(float newFood) //for the slider : changes the amount of food
    {
        amountFood = newFood;
        text.text = newFood.ToString();
    }

    public void Predator(float PredatorNumber) //for the slider : changes the amount of predators
    {
        PredNumber = PredatorNumber;
        predatortext.text = PredNumber.ToString();
    }
}
```

SPAWNCHICKEN.CS

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class SpawnChicken : MonoBehaviour
{
    public GameObject PreyPrefab; //sets the chicken as the prefab (thing that will be cloned)
    public float xPos;
    public float zPos;
    public Vector3 pos;
    List<GameObject> chickenclones = new List<GameObject>(); // list for the chicken prefabs
    public int populationnumber;
    public static bool allStopped = false; //checks if all chickens have stopped
```

```
  int counter = 0;
  Vector3 prev_velocity = new Vector3(0, 0, 0);
  Vector3 cur_velocity = new Vector3(0, 0, 0);
  int i = 0;
  int j = 0;
  int k = 0;
  int foodcount = 0;
  public static bool reproduce = false;
  public static bool done = false;
  private Rigidbody PrefabRB; //rigidbody component of chicken clone
  private Animator chickenanim; //animation component of chicken clone
  int foodcollected = 0; //amount of food collected
  public static bool reset = false;


  private void Start()
  {
    populationnumber = (StateNameController.population); //from the start screen, population is
passed
    RandomFood(populationnumber);
    StartCoroutine(Velocity()); //checks if chickens have stopped
  }
  public IEnumerator Delay()
  {
    yield return new WaitForSeconds(2.1f);
    StartCoroutine(Velocity());
  }

  void Update()
  {

    reproduce = MoveChicken.reproduce; //checks if there are any chickens reproducing
    if (reproduce == true) //if all reproduction has finished
    {
      k = 0;
      while (k < chickenclones.Count)
      {
        if (chickenclones[k] != null)
        {
          foodcollected = chickenclones[k].GetComponent<MoveChicken>().FoodCollected; //for
every chicken - checks food collected
          if (foodcollected >= 2)
          {
            foodcount++;
          }
        }
        k++;

      }
      Reproduce(foodcount); //spawns chickens in the next generation (born)
      reproduce = false;
```

```
            j = 0;
            k = 0;
            foodcount = 0;
        }
        reset = MoveChicken.reset;
        if (reset == true)
        {
            allStopped = false;
            done = false;
            reset = MoveChicken.reset;
            StartCoroutine(Delay());

        }
        CheckActive();
        CheckEnding();
    }

    void RandomFood(int populationnumber) //spawns chickens with initial population
    {

        for (int i = 0; i < populationnumber; i++)
        {
            xPos = Random.Range(-140.0f, 140.0f); //sets random coordinates for the chicken to spawn at
(y coordinate will always be 0)
            zPos = Random.Range(-140.0f, 140.0f);
            pos = new Vector3(xPos, 0.0f, zPos);
            if (Vector3.Distance(transform.position, pos) <= 3.0f) //makes sure space between chickens is
more than 0.5
            {
                xPos = Random.Range(-140.0f, 140.0f);
                zPos = Random.Range(-140.0f, 140.0f);
                pos = new Vector3(xPos, 0.0f, zPos);
            }
            GameObject ChickenClone = Instantiate(PreyPrefab, pos, Quaternion.Euler(new Vector3(0,
Random.Range(0, 360), 0))); //spawns in the chicken at the random coordinates generated
            chickenclones.Add(ChickenClone); //adds to list
        }



    }
    public IEnumerator Velocity()
    {
        while (allStopped == false) //while all chickens have not stopped
        {
            i = 0;
            counter = 0;
            while (i < chickenclones.Count) //for all chickens, checks their position now and after some
time - if it is the same it has stopped
            {
                if (chickenclones[i] != null)
```

```
            {
                bool passed = false;
                prev_velocity = chickenclones[i].transform.position;
                do
                {
                    prev_velocity = chickenclones[i].transform.position;
                    yield return new WaitForSeconds(2.1f); //2.1f - max time chicken is idle is 2, so 2.1 to
avoid false positive
                    try
                    {
                        cur_velocity = chickenclones[i].transform.position;
                        if ((Vector3.Distance(cur_velocity, prev_velocity) >= 0.00) &&
(Vector3.Distance(cur_velocity, prev_velocity) < 0.1f)) //bit risky? - 0.01 instead??
                        {
                            counter++; //counter keeps track of the number of chickens that have stopped
                            passed = true;
                        }
                    }
                    catch
                    {

                        CheckActive();
                        i = 0;
                        counter = 0;
                        allStopped = false;

                    }

                }
                while (passed == false);
            }
            i++;
            CheckActive();

        }
        if (counter == chickenclones.Count) //if the number of chickens stopped is the same as the
total number of chickens (i.e. when all have stopped)
        {
            allStopped = true;
            Debug.Log("all have stopped");
        }
        else
        {
            counter = 0;
            allStopped = false; //sends this to move chicken script
            i = 0;
        }

    }
```

```
    }

    void Reproduce(int spawn) //reproduction of next generation chickens (if food >= 2)
    {

        while (j < spawn)
        {
            int random = Random.Range(0, 4); //random number 0 - 3 (determines where the chicken
spawns)
            if (random == 0)
            {
                float randomz = Random.Range(150.0f, 170.0f);
                float randomx = Random.Range(-170.0f, 170.0f);
                Vector3 position = new Vector3(randomx, 0.0f, randomz);
                GameObject ChickenClone = Instantiate(PreyPrefab, position, Quaternion.Euler(new
Vector3(0, Random.Range(0, 360), 0))); //spawns in the chicken at the random coordinates
generated
                chickenclones.Add(ChickenClone); //adds to list
                PrefabRB = ChickenClone.GetComponent<Rigidbody>();
                chickenanim = ChickenClone.GetComponent<Animator>();
                PrefabRB.velocity = Vector3.zero;
                chickenanim.SetTrigger("GoIdle");
                j++;
            }
            else if (random == 1)
            {
                float randomz = Random.Range(-150.0f, -170.0f);
                float randomx = Random.Range(-170.0f, 170.0f);
                Vector3 position = new Vector3(randomx, 0.0f, randomz);
                GameObject ChickenClone = Instantiate(PreyPrefab, position, Quaternion.Euler(new
Vector3(0, Random.Range(0, 360), 0))); //spawns in the chicken at the random coordinates
generated
                chickenclones.Add(ChickenClone); //adds to list
                PrefabRB = ChickenClone.GetComponent<Rigidbody>();
                chickenanim = ChickenClone.GetComponent<Animator>();
                PrefabRB.velocity = Vector3.zero;
                chickenanim.SetTrigger("GoIdle");
                j++;
            }
            else if (random == 2)
            {
                float randomz = Random.Range(-170.0f, 170.0f);
                float randomx = Random.Range(-150.0f, -170.0f);
                Vector3 position = new Vector3(randomx, 0.0f, randomz);
                GameObject ChickenClone = Instantiate(PreyPrefab, position, Quaternion.Euler(new
Vector3(0, Random.Range(0, 360), 0))); //spawns in the chicken at the random coordinates
generated
                chickenclones.Add(ChickenClone); //adds to list
                PrefabRB = ChickenClone.GetComponent<Rigidbody>();
                chickenanim = ChickenClone.GetComponent<Animator>();
                PrefabRB.velocity = Vector3.zero;
```

```
            chickenanim.SetTrigger("GoIdle");
            j++;
        }
        else if (random == 3)
        {
            float randomz = Random.Range(-170.0f, 170.0f);
            float randomx = Random.Range(150.0f, 170.0f);
            Vector3 position = new Vector3(randomx, 0.0f, randomz);
            GameObject ChickenClone = Instantiate(PreyPrefab, position, Quaternion.Euler(new
Vector3(0, Random.Range(0, 360), 0))); //spawns in the chicken at the random coordinates
generated
            chickenclones.Add(ChickenClone); //adds to list
            PrefabRB = ChickenClone.GetComponent<Rigidbody>();
            chickenanim = ChickenClone.GetComponent<Animator>();
            PrefabRB.velocity = Vector3.zero;
            chickenanim.SetTrigger("GoIdle");
            j++;
        }
        done = true; //indicates that reproduction has finished

    }


  }

  void CheckActive() //checks if any chickens that have been destroyed are still null in the list
  {
    int a = 0;
    while (a < chickenclones.Count)
    {
      if (chickenclones[a] == null)
      {
        chickenclones.RemoveAt(a);
      }
      a++;
    }

  }

  public void CheckEnding() //if all chickens have died
  {
    if (chickenclones.Count == 0)
    {
      SceneManager.LoadScene("EndScreen");
    }
  }
}
```

SPAWNFOOD.CS

```
using System.Collections;
using System.Collections.Generic;
```

```
using UnityEngine;

public class SpawnFood : MonoBehaviour
{
   public GameObject FoodPrefab; //sets the apple as the prefab (thing that will be cloned)
   public float xPos;
   public float zPos;
   public Vector3 pos;
   public float timer = 5.0f; //timer to space out random spawning
   public float amountFood = 10;
   int i = 0;
   int j = 0;
   List<GameObject> foodclones = new List<GameObject>(); //list to store the food prefabs
   private GameObject newGo;
   private float newFood;
   private Vector3 prevPos = new Vector3(0, 0, 0);
   public static bool gamePaused;
   public static bool reset = false;

   IEnumerator Delay(float amountFood)
   {
      if (reset == true)
      {
         reset = MoveChicken.reset;
      }

      while (i < amountFood)
      {
         xPos = Random.Range(-140.0f, 140.0f); //sets random coordinates for the food to spawn at (y
coordinate will always be 0)
         zPos = Random.Range(-140.0f, 140.0f);
         pos = new Vector3(xPos, 0.0f, zPos);

         while (Vector3.Distance(prevPos, pos) <= 5.0f) //makes sure space between food is more than
5
         {
            xPos = Random.Range(-140.0f, 140.0f);
            zPos = Random.Range(-140.0f, 140.0f);
            pos = new Vector3(xPos, 0.0f, zPos);
         }
         yield return new WaitForSeconds(0.001f); //adds a time delay - 0.01f
         newGo = Instantiate(FoodPrefab, pos, Quaternion.identity); //creates new food in the
variable newGo
         foodclones.Add(newGo); //this is then added to the list
         prevPos = pos;
         i++;
      }
      int k = 0;
      while (k < foodclones.Count)
      {
         if (foodclones[k] == null)
```

```
        {
            foodclones.RemoveAt(k);
        }
        k++;
    }
    k = 0;
}

void Start()
{
    StartCoroutine(Delay(amountFood));
}

void Update()
{
    reset = MoveChicken.reset;
    newFood = GameObject.Find("Canvas").GetComponent<PauseResume>().amountFood; //gets
value of slider from PauseResume
    gamePaused = PauseResume.GamePaused;
    if (gamePaused == false)
    {
        if (newFood != amountFood) //if the new value differs from the old value...
        {
            if (newFood > amountFood) //to spawn more food...
            {
                i = 0;
                amountFood = newFood - amountFood; //calculates the remainder
                StartCoroutine(Delay(amountFood)); //instantiates the remainder
                amountFood = newFood; //makes the values the same again
                i = 0;
            }
            if (newFood < amountFood) //to delete existing food...
            {
                j = 0;
                float difference = amountFood - newFood; //calculates the difference between the old
and new values
                while (j < difference)
                {
                    UnityEngine.Object.Destroy(foodclones[j], 1.0f); //destroys food prefabs
                    foodclones.RemoveAt(j); //removes from the list
                    Debug.Log(foodclones.Count);
                    j++;
                }
                amountFood = newFood; //makes values the same again
                j = 0;
            }
            amountFood = newFood;
        }
        if (reset == true)
        {
```

```
            float resetfood = foodclones.Count -
GameObject.FindGameObjectsWithTag("Food").Length;
            i = 0;
            StartCoroutine(Delay(resetfood));

        }
    }
  }
  }
```

SPAWNPREDATOR.CS

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SpawnPredator : MonoBehaviour
{
   public GameObject PredatorPrefab;
   public float xPos;
   public float zPos;
   public Vector3 pos;
   List<GameObject> predatorclones = new List<GameObject>();
   float pred_num = 0;
   float new_pred_num = 0;
   public static bool gamePaused;
   int j = 0;
   public static bool ChangeSlider = false;

   public void PredatorSpawner(float populationnumber)
   {
      for (int i = 0; i < populationnumber; i++)
      {
         xPos = Random.Range(-140.0f, 140.0f); //sets random coordinates for the chicken to spawn at
(y coordinate will always be 0)
         zPos = Random.Range(-140.0f, 140.0f);
         pos = new Vector3(xPos, 0.0f, zPos);
         if (Vector3.Distance(transform.position, pos) <= 3.0f) //makes sure space between chickens is
more than 0.5
         {
            xPos = Random.Range(-140.0f, 140.0f);
            zPos = Random.Range(-140.0f, 140.0f);
            pos = new Vector3(xPos, 0.0f, zPos);
         }
         GameObject ChickenClone = Instantiate(PredatorPrefab, pos, Quaternion.Euler(new
Vector3(0, Random.Range(0, 360), 0))); //spawns in the chicken at the random coordinates
generated
         predatorclones.Add(ChickenClone); //adds to list
      }

   }
```

```
public IEnumerator Delay()
{
    yield return new WaitForSeconds(0.01f);
    ChangeSlider = false;
}

void Update()
{
    CheckActive();
    new_pred_num = PauseResume.PredNumber; //gets value of slider from PauseResume
    gamePaused = PauseResume.GamePaused;
    if (gamePaused == false)
    {
        if (new_pred_num != pred_num) //if the new value differs from the old value...
        {
            if (new_pred_num > pred_num) //to spawn more food...
            {
                pred_num = new_pred_num - pred_num; //calculates the remainder
                PredatorSpawner(pred_num); //instantiates the remainder
                pred_num = new_pred_num; //makes the values the same again

            }
            else if (new_pred_num < pred_num) //to delete existing food...
            {
                j = 0;
                float difference = pred_num - new_pred_num; //calculates the difference between the
old and new values
                while ((j < difference) && (predatorclones.Count != 0))
                {
                    if (predatorclones[j] != null)
                    {
                        UnityEngine.Object.Destroy(predatorclones[j], 0.1f); //destroys food prefabs

                    }
                    CheckActive();
                    j++;
                }
                pred_num = new_pred_num; //makes values the same again
                pred_num = GameObject.FindGameObjectsWithTag("Predator").Length;
                j = 0;
            }
            pred_num = new_pred_num;

        }
        int k = 0;
        while (k < predatorclones.Count) //checks if null objects in list
        {
            if (predatorclones[k] == null)
            {
                predatorclones.RemoveAt(k);
                ChangeSlider = true;
```

```
            }
            k++;
        }
        k = 0;
        StartCoroutine(Delay());
    }
  }

  void CheckActive()
  {
    int k = 0;
    while (k < predatorclones.Count)
    {
      if (predatorclones[k] == null)
      {
        predatorclones.RemoveAt(k);
        ChangeSlider = true;
      }
      k++;
    }
    k = 0;
  }
}
```

STARTPOPINPUT.CS

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using UnityEngine.SceneManagement;

public class StartPopInput : MonoBehaviour
{
  private int userpopulation;

  [SerializeField]
  private TMP_InputField input; //what the user inputs


  [SerializeField]
  private TextMeshProUGUI text; //the text above the input field

  public void GetInput(string population)
  {
    CheckInput(int.Parse(population)); //changes input string to integer
    StateNameController.population = int.Parse(population);
  }
```

```
    public void CheckInput(int population)
    {
        if ((population <= 20) && (population >= 1)) //checking whether the number is valid
        {
            text.text = "Valid response! Loading simulation..."; //changes text
            input.text = ""; //input text cleared
            StartCoroutine(SceneWithLoadDelay(1)); //delays the loading of the scene (linked to
IEnumerator)
        }
        else if (population > 20) //data out of range
        {
            text.text = "Please enter a number that is 20 or below!";
            input.text = "";
        }
        else if (population < 1)
        {
            text.text = "Please enter a number above 1!";
        }
        else
        {
            text.text = "Invalid response. Please try again!";
            input.text = "";
        }

    }
    IEnumerator SceneWithLoadDelay(int numsecs) //creates a delay when the right input is entered
    {
        yield return new WaitForSeconds(numsecs);
        PlaySim();
    }
    public void PlaySim()
    {

        SceneManager.LoadScene("Simulation"); //loads the scene with the name "simulation"

    }
}
```

STATENAMECONTROLLER.CS

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class StateNameController : MonoBehaviour //for the static variables needed between scenes
{
    public static int population; //static variable allows for access in different scripts (add this
somewhere and this script can be deleted)

    public void Restart()
```

```
    {
        SceneManager.LoadScene("Menu");
    }
}
```

SWITCHCAMERA.CS

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SwitchCamera : MonoBehaviour
{
    public GameObject cam1;
    public GameObject cam2;
    public GameObject cam3;
    public GameObject cam4;
    public GameObject cam5;
    public GameObject cam6;
    public Transform parentObject;
    public float zoomamount;
    public float maxClamp = 10;
    public float ROTspeed = 10;


    // Start is called before the first frame update
    void Start()
    {
        cam1.SetActive(true);
        cam2.SetActive(false);
        cam3.SetActive(false);
        cam4.SetActive(false);
        cam5.SetActive(false);
        cam6.SetActive(false);

    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetButtonDown("Switch1")) //is the a key
        {
            cam1.SetActive(true);
            cam2.SetActive(false);
            cam3.SetActive(false);
            cam4.SetActive(false);
            cam5.SetActive(false);
            cam6.SetActive(false);
        }
        if (Input.GetButtonDown("Switch2")) //is the s key
        {
            cam1.SetActive(false);
```

```
            cam2.SetActive(true);
            cam3.SetActive(false);
            cam4.SetActive(false);
            cam5.SetActive(false);
            cam6.SetActive(false);
        }
        if (Input.GetButtonDown("Switch3")) //is the d key
        {
            cam1.SetActive(false);
            cam2.SetActive(false);
            cam3.SetActive(true);
            cam4.SetActive(false);
            cam5.SetActive(false);
            cam6.SetActive(false);
        }
        if (Input.GetButtonDown("Switch4")) //is the f key
        {
            cam1.SetActive(false);
            cam2.SetActive(false);
            cam3.SetActive(false);
            cam4.SetActive(true);
            cam5.SetActive(false);
            cam6.SetActive(false);
        }
        if (Input.GetButtonDown("Switch5")) //is the g key
        {
            cam1.SetActive(false);
            cam2.SetActive(false);
            cam3.SetActive(false);
            cam4.SetActive(false);
            cam5.SetActive(true);
            cam6.SetActive(false);
        }
        if (Input.GetButtonDown("Switch6")) //is the h key
        {
            cam1.SetActive(false);
            cam2.SetActive(false);
            cam3.SetActive(false);
            cam4.SetActive(false);
            cam5.SetActive(false);
            cam6.SetActive(true);
        }

    }
}
```

TILEGENERATION.CS

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
using UnityEngine.UI;

[System.Serializable]
public class TerrainType
{
   public string name;
   public float height;
   public Color color;
}

public class TileGeneration : MonoBehaviour
{
   [SerializeField]
   private TerrainType[] terrainTypes;

   [SerializeField]
   private float heightMultiplier;

   [SerializeField]
   private AnimationCurve heightCurve;

   [SerializeField]
   NoiseMapGeneration noiseMapGeneration;

   [SerializeField]
   private MeshRenderer tileRenderer;

   [SerializeField]
   private MeshFilter meshFilter;

   [SerializeField]
   private MeshCollider meshCollider;

   [SerializeField]
   private float mapScale;

   void Start()
   {
      GenerateTile();
   }
   void GenerateTile()
   {
      // calculate tile depth and width based on the mesh vertices
      Vector3[] meshVertices = this.meshFilter.mesh.vertices;
      int tileDepth = (int)Mathf.Sqrt(meshVertices.Length);
      int tileWidth = tileDepth;
      // calculate the offsets based on the tile position
      float offsetX = -this.gameObject.transform.position.x;
      float offsetZ = -this.gameObject.transform.position.z;
      float[,] heightMap = this.noiseMapGeneration.GenerateNoiseMap(tileDepth, tileWidth,
this.mapScale, offsetX, offsetZ);
```

```
    // generate a heightMap using noise
    Texture2D tileTexture = BuildTexture(heightMap);
    this.tileRenderer.material.mainTexture = tileTexture;
    UpdateMeshVertices(heightMap);
  }
  private Texture2D BuildTexture(float[,] heightMap)
  {
    int tileDepth = heightMap.GetLength(0);
    int tileWidth = heightMap.GetLength(1);
    Color[] colorMap = new Color[tileDepth * tileWidth];
    for (int zIndex = 0; zIndex < tileDepth; zIndex++)
    {
      for (int xIndex = 0; xIndex < tileWidth; xIndex++)
      {
        // transform the 2D map index is an Array index
        int colorIndex = zIndex * tileWidth + xIndex;
        float height = heightMap[zIndex, xIndex];
        // choose a terrain type according to the height value
        TerrainType terrainType = ChooseTerrainType(height);
        // assign the color according to the terrain type
        colorMap[colorIndex] = terrainType.color;
      }
    }
    // create a new texture and set its pixel colors
    Texture2D tileTexture = new Texture2D(tileWidth, tileDepth);
    tileTexture.wrapMode = TextureWrapMode.Clamp;
    tileTexture.SetPixels(colorMap);
    tileTexture.Apply();
    return tileTexture;
  }
  private void UpdateMeshVertices(float[,] heightMap)
  {
    int tileDepth = heightMap.GetLength(0);
    int tileWidth = heightMap.GetLength(1);
    Vector3[] meshVertices = this.meshFilter.mesh.vertices;
    // iterate through all the heightMap coordinates, updating the vertex index
    int vertexIndex = 0;
    for (int zIndex = 0; zIndex < tileDepth; zIndex++)
    {
      for (int xIndex = 0; xIndex < tileWidth; xIndex++)
      {
        float height = heightMap[zIndex, xIndex];
        Vector3 vertex = meshVertices[vertexIndex];
        // change the vertex Y coordinate, proportional to the height value. The height value is
evaluated by the heightCurve function, in order to correct it.
        meshVertices[vertexIndex] = new Vector3(vertex.x, this.heightCurve.Evaluate(height) *
this.heightMultiplier, vertex.z);
        vertexIndex++;
      }
    }
    // update the vertices in the mesh and update its properties
```

```
        this.meshFilter.mesh.vertices = meshVertices;
        this.meshFilter.mesh.RecalculateBounds();
        this.meshFilter.mesh.RecalculateNormals();
        // update the mesh collider
        this.meshCollider.sharedMesh = this.meshFilter.mesh;
    }

    TerrainType ChooseTerrainType(float height)
    {
        // for each terrain type, check if the height is lower than the one for the terrain type
        foreach (TerrainType terrainType in terrainTypes)
        {
            // return the first terrain type whose height is higher than the generated one
            if (height < terrainType.height)
            {
                return terrainType;
            }
        }
        return terrainTypes[terrainTypes.Length - 1];
    }

}
```

WINDOWGRAPH.CS

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;


public class Window_Graph : MonoBehaviour
{
    [SerializeField] private Sprite circleSprite;
    private RectTransform graphContainer;
    public GameObject GraphScreen;
    private RectTransform labelTemplateX;
    private RectTransform labelTemplateY;
    private static int initialpopulation;
    private static int gen_number;
    GameObject lastCircleGameObject = null;
    GameObject lastCircleGameObjectP = null;
    List<int> valuelist = new List<int>();
    List<int> valuelistP = new List<int>();

    void Start()
    {
        graphContainer =
GraphScreen.transform.Find("graphContainer").GetComponent<RectTransform>();
        labelTemplateX = graphContainer.Find("labelTemplateX").GetComponent<RectTransform>();
        labelTemplateX.gameObject.SetActive(false);
```

```
        labelTemplateY = graphContainer.Find("labelTemplateY").GetComponent<RectTransform>();
        labelTemplateY.gameObject.SetActive(false);
        initialpopulation = (StateNameController.population);
        gen_number = PauseResume.gennumber;
        valuelist.Add(initialpopulation);
        valuelistP.Add(0);
        ShowGraph(valuelist, 0);
        ShowGraphPredator(valuelistP, 0);
    }

    void Update()
    {
        int gen = PauseResume.gennumber;
        if (gen_number != gen)
        {
            Debug.Log("in here");
            gen_number = PauseResume.gennumber;
            int temp = (GameObject.FindGameObjectsWithTag("Chicken").Length) +
(GameObject.FindGameObjectsWithTag("Speed").Length) +
(GameObject.FindGameObjectsWithTag("Energy").Length) +
(GameObject.FindGameObjectsWithTag("Chicken").Length); //finds how many chickens on the plane
at the start of each gen
            valuelist.Add(temp); //adds to a list
            ShowGraph(valuelist, gen_number - 1);
            int temp2 = (GameObject.FindGameObjectsWithTag("Predator").Length);
            valuelistP.Add(temp2); //adds to a list
            ShowGraphPredator(valuelistP, gen_number - 1);
        }

    }

    private GameObject CreateCircle(Vector2 anchoredPosition)
    {
        GameObject gameObject = new GameObject("circle", typeof(Image));
        gameObject.transform.localScale = new Vector3(0.1397466f, 0.4096875f, 1.0f); //scale of the
dot
        gameObject.transform.SetParent(graphContainer, false);
        gameObject.GetComponent<Image>().sprite = circleSprite;
        RectTransform rectTransform = gameObject.GetComponent<RectTransform>();
        rectTransform.anchoredPosition = anchoredPosition;
        rectTransform.sizeDelta = new Vector2(11, 11);
        rectTransform.anchorMin = new Vector2(0, 0);
        rectTransform.anchorMax = new Vector2(0, 0);
        return gameObject;
    }

    private void ShowGraph(List<int> valuelist, int j)
    {
        float graphHeight = graphContainer.sizeDelta.y;
        float graphWidth = graphContainer.sizeDelta.x;
        float yMaximum = 40f; //max y value
```

```
        float xSize = graphWidth / 20;//controls how spaced out the dots are
        {
            float xPosition = j * xSize;
            float yPosition = (valuelist[j] / yMaximum) * graphHeight;
            GameObject circleGameObject = CreateCircle(new Vector2(xPosition, yPosition));
            if (lastCircleGameObject != null)
            {

CreateDotConnection(lastCircleGameObject.GetComponent<RectTransform>().anchoredPosition,
circleGameObject.GetComponent<RectTransform>().anchoredPosition);
            }
            lastCircleGameObject = circleGameObject;

            RectTransform labelX = Instantiate(labelTemplateX);
            labelX.SetParent(graphContainer, false);
            labelX.gameObject.SetActive(true);
            labelX.anchoredPosition = new Vector2(xPosition + 3, -5.2f); //formatting the x axis labels
            labelX.GetComponent<TextMeshProUGUI>().text = (j+1).ToString();
            labelX.localScale = new Vector3(0.05593378f, 0.1593675f, 1f);
        }
        int separatorcount = 10; //how many y axis labels there are
        for (int i = 0; i <= separatorcount; i++)
        {
            RectTransform labelY = Instantiate(labelTemplateY);
            labelY.SetParent(graphContainer, false);
            labelY.gameObject.SetActive(true);
            float normalisedvalue = i * 1f / separatorcount;
            labelY.anchoredPosition = new Vector2(0.3f, normalisedvalue * graphHeight);
            labelY.GetComponent<TextMeshProUGUI>().text = Mathf.RoundToInt(normalisedvalue *
yMaximum).ToString();
            labelY.localScale = new Vector3(0.05593378f, 0.1593675f, 1f);
        }

    }

    private void CreateDotConnection (Vector2 dotPositionA, Vector2 dotPositionB) //creates
connection between 2 dots on the graph
    {
        GameObject gameObject = new GameObject("dotConnection", typeof(Image));
        gameObject.transform.SetParent(graphContainer, false);
        gameObject.GetComponent<Image>().color = new Color(1, 1, 1, .5f);
        RectTransform rectTransform = gameObject.GetComponent<RectTransform>();
        Vector2 dir = (dotPositionB - dotPositionA).normalized;
        float distance = Vector2.Distance(dotPositionA, dotPositionB);
        rectTransform.anchorMin = new Vector2(0, 0);
        rectTransform.anchorMax = new Vector2(0, 0);
        rectTransform.sizeDelta = new Vector2(distance, 3f);
        rectTransform.anchoredPosition = dotPositionA;
        rectTransform.anchoredPosition = dotPositionA + dir * distance * .5f;
        rectTransform.localEulerAngles = new Vector3(0, 0, GetAngleFromVectorFloat(dir));
```

```
    gameObject.transform.localScale = new Vector3(1.0f, 0.2f, 1.0f); //thickness of the line (y value
change)
  }

  float GetAngleFromVectorFloat(Vector2 dir)
  {
    dir = dir.normalized;
    float n = Mathf.Atan2(dir.y, dir.x) * Mathf.Rad2Deg;
    if (n < 0) n += 360;
    return n;
  }

  private void ShowGraphPredator(List<int> valuelist, int j)
  {
    float graphHeight = graphContainer.sizeDelta.y;
    float graphWidth = graphContainer.sizeDelta.x;
    float yMaximum = 40f; //max y value
    float xSize = graphWidth / 20;//controls how spaced out the dots are
    {
      float xPosition = j * xSize;
      float yPosition = (valuelist[j] / yMaximum) * graphHeight;
      GameObject circleGameObject = CreateCircle(new Vector2(xPosition, yPosition));
      circleGameObject.GetComponent<Image>().color = new Color (255, 0, 0);
      if (lastCircleGameObjectP != null)
      {

CreateDotConnectionPredator(lastCircleGameObjectP.GetComponent<RectTransform>().anchoredP
osition, circleGameObject.GetComponent<RectTransform>().anchoredPosition);
      }
      lastCircleGameObjectP = circleGameObject;
    }

  }
  private void CreateDotConnectionPredator(Vector2 dotPositionA, Vector2 dotPositionB) //creates
connection between 2 dots on the graph
  {
    GameObject gameObject = new GameObject("dotConnection", typeof(Image));
    gameObject.transform.SetParent(graphContainer, false);
    gameObject.GetComponent<Image>().color = new Color(255,0,0, 0.5f);
    RectTransform rectTransform = gameObject.GetComponent<RectTransform>();
    Vector2 dir = (dotPositionB - dotPositionA).normalized;
    float distance = Vector2.Distance(dotPositionA, dotPositionB);
    rectTransform.anchorMin = new Vector2(0, 0);
    rectTransform.anchorMax = new Vector2(0, 0);
    rectTransform.sizeDelta = new Vector2(distance, 3f);
    rectTransform.anchoredPosition = dotPositionA;
    rectTransform.anchoredPosition = dotPositionA + dir * distance * .5f;
    rectTransform.localEulerAngles = new Vector3(0, 0, GetAngleFromVectorFloat(dir));
    gameObject.transform.localScale = new Vector3(1.0f, 0.2f, 1.0f); //thickness of the line (y value
change)
  }
```

}

TECHNIQUES USED

TECHNIQUE	PAGE NUMBER
OOP	Throughout
Classes	Throughout
Enumerators	Throughout (mainly 31-39)
Procedures and Functions	Throughout
Unity 2D	Throughout (mainly 40-46)
Unity 3D	Throughout
Lists	46 & 54
Vectors	33-35 & 38-39
Perlin Noise	40
Arrays (2D)	40

Testing Phase

TESTING PLAN

Test Number	Purpose	Input Data	Expected Result	Pass/Fail
1 – Main Menu				
1.1	To see whether the info button makes the info screen appear	Click info button	The info screen should appear	In Testing Video
1.2	To check if the start button makes the population input screen appear	Click the start button	The population screen should appear (where you can input a population number)	In Testing Video
2 – Population Input				
2.1	Check if input can handle values outside of the range	Type in a value outside of the range (e.g. 100)	Error message appears	In Testing Video
2.2	Check if input can handle negative values	Type in a negative value (e.g. -20)	Error message appears	In Testing Video
2.3	Check if input can handle non-integer values	Type in non-integer value (e.g. a string)	Does not accept	In Testing Video
2.4	Check if the right value switches scene to simulation	Type in a value within the given range	Acceptance message + scene switches to simulation	In Testing Video

2.5	If correct population number spawns (User input) (objective 2)	Type in a value within the given range	Number that was input becomes population number (chickens spawn)	In Testing Video
3 – Settings Page				
3.1	Check if increasing the food slider increases the number of food on the field (objective 2, objective 8.1)	Increase slider (using mouse)	More food spawns on the field	In Testing Video
3.2	Check if decreasing food slider decreases the number of food on the field (objective 8.1)	Decrease slider (using mouse)	Food gets deleted from the field	In Testing Video
3.3	Check if increasing the predator slider increases the number of predators on the field (objective 2, objective 8.2)	Increase slider (using mouse)	More predators spawn on the field	In Testing Video
3.4	Check if decreasing the predator slider decreases the number of predators on the field (objective 8.2)	Decrease slider (using mouse)	Predators are deleted from the field	In Testing Video
3.5	Check if predators running out of energy and dying updates the slider too (objective 1, objective 6, objective 8.2)	Wait for predator to run out of energy	Slider number represents the new number of predators	In Testing Video
3.6	Check if clicking/unclicking mutation box makes mutation	Click mutation box (using mouse)	Clicking makes mutation inputs appear/disappear	In Testing Video

	inputs appear/disappear (objective 7)			
3.7	If negative/non-integer/outside of range value makes error appear (for energy mutation) (objective 7)	Type in negative/non integer/outside of range value	Error message displayed	In Testing Video
3.8	If value in range is accepted (for energy mutation) (objective 7)	Type in value in the range	Acceptance message displayed (mutation rate implemented)	In Testing Video
3.9	If negative/non-integer/outside of range value makes error appear (for speed mutation) (objective 7)	Type in negative/non integer/outside of range value	Error message displayed	In Testing Video
3.10	If value in range is accepted (for speed mutation) (objective 7)	Type in value in the range	Acceptance message displayed (mutation rate implemented)	In Testing Video
4 – Pause Screen				
4.1	Check if clicking the pause button makes all movement stop and pause screen displayed	Click pause button (using the mouse)	All movement stops and the pause screen is displayed	In Testing Video
4.2	If clicking resume button makes movement resume on the plane	Click the resume button (with the mouse)	Movement resumes and the simulation screen appears again	In Testing Video
5 – Graph Screen				
5.1	If clicking the arrow on the settings screen makes graph screen appear (objective 9)	Click arrow (with mouse)	Graph screen should appear	In Testing Video
5.2	If starting population is displayed	Choose starting population and	Node should be displaying the	In Testing Video

	correctly on graph (objective 9)	check if circle appears in the right spot on graph	correct starting population at x=1	
5.3	Check if every new generation adds a node to the graph (objective 9)	Every generation, check if graph has been updated	Should be multiple nodes that display the population at the start of every new generation	In Testing Video
5.4	Check if there are line connections between nodes (objective 9)	After multiple generations, check if nodes have been connected	Should be line connection between nodes that connect them	In Testing Video
5.5	Check if increasing the number of predators displays the new population (objective 9)	Every generation, check if graph has been updated	Should be multiple nodes that display the population at the start of every new generation	In Testing Video
6 – Simulation				
6.1	Check if chickens move in a random direction with the walk animation (objective 3)	Debug.Log message displaying direction vector	The chickens should move in different directions with different wait times after they reach	In Testing Video
6.2	Check if predators move in a random direction with the walk animation (objective 3)	Debug.Log message displaying direction vector	The predators should move in different directions with different wait times after they reach	In Testing Video
6.3	Check if eat animation occurs when chickens collide with food	Move chicken to collide with the food and check if animation plays	Chicken should do the eating animation	In Testing Video
6.4	Check if eat animation occurs when predators collide with food	Move predator to collide with the food and check if animation plays	Predator should do the eating animation	In Testing Video

6.5	If chickens reach the red area, they should stop	Move chicken to red area and check if it stops	Chicken should stop and do an idle animation	In Testing Video
6.6	If chickens or predators run out of energy, they should delete (objective 1, objective 6)	Set energy to a low number to ensure it runs out	Debug message and deletion of predator/chicken	In Testing Video
6.7	Check if when all the chickens have stopped, the chickens with 2 or more food reproduces (objective 5)	Move chicken to positions so that it collects 2+ food and then move it to red area	Chicken should clone – with the clone spawning somewhere random in the red area	In Testing Video
6.8	Check if when all the chickens have stopped, the chickens with 1 food does not die (objective 5, objective 6)	Move chicken to positions so that it collects 1 food and then move it to red area	The chicken should stay on the plane for the next generation	In Testing Video
6.9	Check if when all the chickens have stopped, the chickens with 0 die (objective 5, objective 6)	Move chicken to red area (without collecting food)	Chicken should die	In Testing Video
6.10	Check if when all chickens have stopped, the generation number increases	Allow all chickens to move to the red area and stop	Generation number should increment	In Testing Video
6.11	If all chickens die, the simulation should stop (end screen displayed)	Change energy settings to 1 for chickens and allow them to die	The end screen should be displayed, showing an ending message	In Testing Video
6.12	Check if fast forward button increases rate of movement for both chickens and predators (objective 10)	Click the Fast Forward toggle	The movement speed of chickens and predators should increases by x2	In Testing Video
6.13	Check whether generated terrain blocks view of the plane	Play the simulation and change camera angles to see if	The cameras should not be blocked and	In Testing Video

	(objective 4)	anything is blocked	everything should be visible	

Evaluation Phase

OBJECTIVES MET?

1) Implement system that allows the organism to have set amounts of energy (e.g. each organism has a certain amount of energy points which get used up as it walks)

This was achieved through assigning energy to both the chickens and the predators. Every time they move, one energy is used up until the energy = 0 (when they die).

2) Ensure that the chickens, predators and food spawn in a random arrangement each time, and not on top each other so that they cannot be seen

Every time a chicken/predator is spawned, random coordinates are generated, with a minimum space between them, so that they are fairly spaced out.

3) Ensure that the movement of the predators/prey is random so that the simulation can be accurately represented.

This was achieved by generating random coordinates for the chickens/predators to move to. The direction vector was calculated so that they looked in the direction they were moving.

4) Create a 3D terrain around the plane via 3D procedural generation and Perlin Noise.

This was done by creating planes around the main simulation plane and generating unique terrain for each of them, using Perlin Noise.

5) Create a simulation that accurately represents the reproduction that occurs during natural selection – a greater amount of food allows the prey to reproduce.

If the chicken collects no food, it dies, if the chicken collects one food, it survives onto the next generation, if the chicken collects 2 or more food, it reproduces (one clone).

6) Create a simulation that accurately represents the death that occurs during natural selection – this could be linked to how much energy the individual has (e.g. running out of energy means death or running into a predator)

If the chicken has no food at the end of a generation or it runs out of energy, it dies. It also dies if it gets 'eaten' by a predator. Energy is used up as the chicken moves.

7) Implement the ability of organisms to have mutations – this relates back to natural selection itself, but there should be a set chance for mutations.

The user is able to input the rate of energy and speed mutations (as a percentage 0-100). This is then implemented in the next generation.

8) Allow the user to have the ability to change features in the simulation

8.1) Allow the user to change the amount of food that spawns at the beginning of each generation through a slider

I implemented a slider on the settings page, and the user is able to move the slider to correspond to the amount of food spawned on the plane – every generation, more food is added to the plane to make up the difference.

8.2) Allow the user to change the amount of predators on the plane at one time – in this way, the deaths of the chickens could be controlled through a slider

The predator slider updates in real time – since the predators may run out of energy and die, the slider must be updated with the new population so more can be added if necessary.

9) Create a graph system that displays the population of the organisms - this would allow the user to understand/comprehend the theory behind what they are seeing and witness the figures behind the process.

9.1) Create an x and y axis with appropriate scales so that if the number of generations that are cycled through is large, the axis will be able to fit each of the points

The space between points on the graph is quite small, so the x axis is able to fit a fairly large amount of generations. The y-axis has a maximum population number of 40, which it is unlikely to extend beyond.

9.2) Create a line graph with clear points marked and connections with the next point

The points are clear and the connections are in the correct direction and visible. The points are accurate and the value is able to be read clearly.

9.3) Create 2 separate lines – one for the population of chickens and one for the population of foxes

I implemented 2 different lines on the same graph, with different colours – the predator population is red, the prey population is white. This is so the user is able to differentiate between the 2.

10) Implement a settings page where the user is able to manipulate settings such as speed of the program – this would enable the user to make the whole process faster (so that they wouldn't have to sit through the entirety of the generation living/dying.

This was achieved by making the speed of the chickens and predators increase when a toggle was switched on, and decrease back to normal when switched off.

CHALLENGES

The main challenges I faced are listed below:

1) Generating the 3D terrain using Perlin Noise – the terrain wasn't varied enough at first (looked very monotone and lots of repeating areas). However, by adjusting a few variables such as the height and the scale, I was able to make it look random. Also, parts of the terrain were colliding with the plane and affecting the simulation, so I comprised the terrain of several different planes, rather than just one.
2) Checking whether all chickens have stopped – the first time I implemented this, I used the coordinates of the red area where the chickens stop. But, after the chickens reset, their

movement was very jerky. To fix this, I had to compare their velocity to see whether they had stopped.

3) Implementing the graph – this was a challenge because putting the connections between dots was quite complicated. I had to calculate the direction that the line had to be to perfectly connect the 2 points. Also, I had to get real time information from the plane on the populations of chickens and foxes to be able to plot it while the simulation was running.
4) Updating the predator slider – because of the nature of the simulation, I felt it was necessary to update the predator slider as the foxes died. This is so that the user could increase/decrease the population of predators as they ran out of energy. However, this was a challenge, as I had to set it up so that the simulation would detect a change in the population of foxes and act accordingly.

CLIENT FEEDBACK

From Ms Dhillon (Biology teacher):

Your simulation looks very well put together and simple to use. I can see that you used chickens and foxes, which is accurate to real life – since they have a predator and prey relationship in real life. The graph looks very clear and the relationship between the chickens and foxes can be identified. It would be nice if you could add different habitats, where different mutations are beneficial, just to get a worldwide view of how natural selection works. There will be some disadvantageous mutations too, while you only have advantageous ones.

Feedback from Catherine Oo (Biology Student):

I think it's really good! I like how the graph is displayed and the colour isn't too harsh and it's easy to understand. I thought the user interface was very intuitive and the actual simulation is quite accurate to how natural selection works. The simulation was kept very simple, but still has the main features so that anyone could understand it – you didn't overcomplicate things by adding unnecessary variables.

FUTURE IMPROVEMENTS

1) Add more mutations – If I had more time, I would have added more mutations along with the 2 I already have (energy & speed). For example, a mutation in the fertility rate – leading to more chickens being spawned as offspring.
2) Make a larger plane + increase starting population size (i.e. implement the simulation on a larger scale) – The scale I used was quite small, so that changes in the variables make a big impact. But, this can cause the population to die out quite quickly. With a larger population size, the trend in the graph would be clearer to see.
3) Enable the predators to reproduce like the chickens – This adds onto the previous point, as with a small population, the predator population would increase and kill the prey population fairly quickly. With a larger population, I could enable the predators to reproduce and the graph would show the relationship between the predators and prey.
4) Add something to identify chickens with a mutation – For example, perhaps changing the colour or adding an icon above their head.
5) Enabling the chickens to have a field of vision in order to identify where to move to collect food – this would make them more efficient in collecting food and surviving, since at the moment, they move randomly to collect food.
6) Making the x and y axis dynamic – so that if the generation number or the population ever exceeded the max value, the graph would still be able to display this.