Computer Science
A Level OCR NEA H446/03

.

Ecosystem Simulator

CONTENTS

Chapter 1 - ANALYSIS.....5
1.1 The Problem.....5
1.1.1 Defining the Problem.....5
1.1.2 Stakeholders.....5
1.2 How the problem can be solved by computational methods.....7
1.2.1 Thinking Abstractly.....7
1.2.2 Thinking Ahead.....8
1.2.3 Thinking Procedurally and Decomposition.....9
1.2.4 Thinking Logically.....10
1.2.5 Thinking Concurrently.....10
1.3 Research.....11
1.3.1 Looking at Previous Games.....11
1.3.2 User Questionnaire.....17
1.3.3 Client Interview.....23
1.4 Requirements.....27
1.4.1 Features of proposed solution.....27
1.4.2 Limitations of proposed solution.....29
1.4.3 Hardware and Software Requirements.....29
1.4.4 Success Criteria For Each Requirement.....31
Chapter 2 - DESIGN.....33
2.1 Decomposition.....33
2.1.1 Decomposition Overview.....33
2.1.2 Explanation of Each Module.....34
2.2 Menus.....36
2.2.1 Interface Design.....36
2.2.2 Pseudocode.....38
2.2.3 Justification.....39
2.3 Environment User Interface.....40
2.3.1 Interface Design.....40
2.3.2 Flowcharts & Pseudocode.....40
2.3.3 Justification.....42
2.4 Free Camera View.....43
2.4.1 Pseudocode.....43
2.4.2 Justification.....44
2.5 Organism Class Diagrams.....45
2.5.1 Pseudocode.....45
2.5.2 Justification.....46
2.6 Organism Attributes.....47
2.6.1 Interface Design.....47
2.6.2 Pseudocode.....47
2.6.3 Justification.....48
2.7 Organism Behaviours.....49

2.7.1 Pseudocode..... 49
2.7.2 Justification.....51
2.8 Organism Genetics & Mutation..... 52
2.8.1 Pseudocode..... 52
2.8.2 Justification.....53
2.9 Organism Pathfinding.....54
2.9.1 Pseudocode..... 54
2.9.2 Justification.....55
2.10 Player Controlled Movement.....56
2.10.1 Pseudocode..... 56
2.10.2 Justification.....57
2.11 Skill Tree & Evolution.....58
2.11.1 Interface Design.....58
2.11.2 Pseudocode.....58
2.11.3 Justification.....58
2.12 Game Over Screen..... 59
2.12.1 Interface Design..... 59
2.12.2 Flowchart.....60
2.12.3 Justification.....60
2.13 Data Structures..... 61
2.14 Usability Features.....65
2.15 Testing Plan.....66
2.15.1 Iteratively Testing.....66
2.15.2 Post-Development Testing..... 69
Chapter 3 - DEVELOPMENT..... 70
3.1 Development Plan.....70
3.2 Menus - Iteration 1..... 72
3.2.1 Intro/Justification..... 72
3.2.2 Annotated Code..... 72
3.2.3 Iterative Testing..... 75
3.2.4 Fixes Applied.....76
3.2.5 Success from Testing..... 78
3.2.6 Review.....80
3.3 Free Camera - Iteration 2.....81
3.3.1 Intro/Justification..... 81
3.3.2 Annotated Code..... 81
3.3.3 Iterative Testing..... 83
3.3.4 Fixes Applied.....84
3.3.5 Success from Testing..... 85
3.3.6 Review.....85
3.4 Organism & User Interface - Iteration 3.....86
3.4.1 Intro/Justification..... 86
3.4.2 Annotated Code..... 87
3.4.3 Iterative Testing..... 97

3.4.4 Fixes Applied....98
3.4.5 Success from Testing....101
3.4.6 Review....102
3.5 Player Controlled Movement - Iteration 4....103
3.5.1 Intro/Justification....103
3.5.2 Annotated Code....104
3.5.3 Iterative Testing....105
3.5.4 Fixes Applied....106
3.5.5 Success from Testing....106
3.5.6 Review....106
3.6 Organism Behaviours & Pathfinding - Iteration 5....107
3.6.1 Intro/Justification....107
3.6.2 Annotated Code....108
3.6.3 Iterative Testing....119
3.6.4 Fixes Applied....121
3.6.5 Success from Testing....122
3.6.6 Review....123
3.7 Organism Genetics & Mutations - Iteration 6....124
3.7.1 Intro/Justification....124
3.7.2 Annotated Code....124
3.7.3 Iterative Testing....126
3.7.4 Fixes Applied....128
3.7.5 Success from Testing....128
3.7.6 Review....129
3.8 End Screen - Iteration 7....130
3.8.1 Intro/Justification....130
3.8.2 Annotated Code....131
3.8.3 Iterative Testing....132
3.8.4 Fixes Applied....132
3.8.5 Success from Testing....132
3.8.6 Review....132
Chapter 4 - EVALUATION....133
4.1 Post Development Testing for Function & Robustness....134
4.1.1 Test Scenario 1 - Free Cam Simulation....134
4.1.2 Test Scenario 2 - User Inputted Movement Simulation....139
4.2 Usability Testing....143
4.3 Success Criteria Evaluation....145
4.4 Limitations....148
4.5 Maintenance....149
4.6 Future Development....149
References....152
Analysis....152
Design....152
Development....152

Chapter 1 - ANALYSIS

1.1 The Problem

1.1.1 Defining the Problem

My game is a sandbox based game that simulates evolution in a virtual environment. It will contain features such as natural selection, genetic mutation, and adaptation to create dynamic ecosystems populated by evolving organisms. Players will be able to observe and interact with the complex behaviours, traits, and strategies that will occur as populations evolve over generations, such as being able to remove or add different variables into the environment. Users will be able to customise parameters and have real-time data visualisation. They can customise the amount of animals and food the scenario will start with and there will be a live view of the number of these animals and food, displayed by a chart. The simulator will offer a fascinating insight into the mechanisms driving evolutionary processes. Whether for educational purposes or scientific exploration.

The game will appeal to people of all ages who are interested in learning and understanding the concepts of natural selection, genetic mutation and adaptation as well as people who are interested in simulation and sandbox games. The game could be used by biology teachers to show students a visualisation of evolutionary concepts to help them better understand. My game will be played on a PC or laptop using a keyboard and mouse which will restrict the target audience due to not everybody having access to a PC or laptop. However, it would be possible to adapt the game to be played on a mobile phone or tablet, using a touch screen instead of a keyboard and mouse, which would widen the target audience as a lot more people have access to a mobile device than a computer.

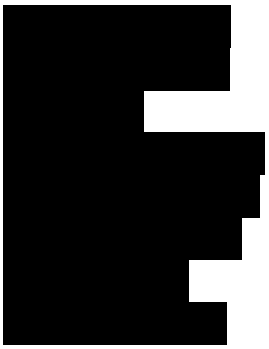
1.1.2 Stakeholders

A stakeholder refers to the people or groups affected by a software development project. For my project, my stakeholders will be people who have an interest in learning more about evolutionary concepts as well as people who have played sandbox and simulation games before, as they will have experience with other similar games to the one I am developing. This will be helpful in the questionnaire and interview process as they will be able to inform me about key features that they enjoyed in previous games. They may also have their own ideas that they have thought of that I could implement into my project.

My main client will be [redacted], who is studying Computer Science BTEC at [redacted]. This means he is knowledgeable about game development and programming. Through the use of interviews and testing I'll be able to gain feedback on the game and make adjustments accordingly to make improvements.

My game's end users will be people interested in learning and developing a deeper understanding about how environments and ecosystems work. My game could be used to teach biology classes about how natural selection and population growth occurs.

Stakeholders :



The feedback from this group of stakeholders will be used during the design and development stages to help ensure the game appeals to a wide range of people by making adjustments with the use of their feedback.

The stakeholders will be using the final solution to learn about how evolution and natural selection occur which is educational for biology students. It could also be entertaining to play the simulation to try and obtain different results from varying population inputs.

1.2 How the problem can be solved by computational methods

My game will be aimed mostly towards students in KS3 and KS4 who are studying biology and wish to develop their understanding of population growth, natural selection and evolution. Algorithms will be used in the game, such as an algorithm to determine which prey a predator will hunt next. For example, I could use an A* algorithm to find the shortest path to a prey with a certain weighted value, such as stamina below a certain value which the predator will hunt.

1.2.1 Thinking Abstractly

Abstraction is defined as a simplified representation of reality. In computer science, abstraction is the process of removing unnecessary details and only including necessary ones. Programmers will commonly use abstraction to make games easier to design and develop since it would be very difficult to develop something identical to reality.

My game will be heavily abstracted since it would be far too complicated to develop an ecosystem simulator that is almost identical to what occurs in real life due to the complexity. Instead, I will be developing a much more simplified version that focuses on the core, fundamental concepts. Here are some examples of abstraction that will be used with the justification.

Visual Graphics: The graphics of my game will be abstracted to a much simpler art style compared to reality. I plan on using a 3D low poly art style, which has blocky, pixelated graphics, similar to games such as “Superhot” and “Muck”. This will save computational time as the computer will have to spend a lot fewer resources rendering graphics, which can instead be used to process algorithms and logical decisions.

Organism Behaviour: Ways in which organisms hunt, eat and reproduce with other organisms will be abstracted to simplify the game for players so that they can more easily understand and be educated about what is happening while having fun at the same time.

Population Growth: My game will contain a feature which allows the user to view real-time data through the use of graphs. They will be able to view data such as the population of a certain organism currently in the environment and how that population has changed over time on a population-time graph. This isn’t shown in real life and therefore is an abstraction.

Time: The time will be abstracted so that players can change the speed at which the simulation is taking place since it would be impractical for the game to occur at the same speed as real life. Players will also have the ability to pause the simulation. This will allow the player to leave the game for as long as they like and then come back to where they were when they left and continue uninterrupted. This is an abstraction since you aren't able to pause or change time in real life.

Abstracting all of these features makes it much more manageable to design, develop and test, allowing me to focus more time on the fundamentals of the game and maintain time efficiency.

1.2.2 Thinking Ahead

Thinking ahead in terms of computational thinking is when programmers consider problems that may occur when developing their software. This then allows them to develop solutions to these problems before they even arise. One way of thinking ahead is by identifying the inputs and outputs of the game, these include:

- Starting population of organisms
- Skill tree selections
- Player controlled organism movements/behaviour
- Population growth over time graph
- Organism statistics and attributes
- Organism positions stored in a database

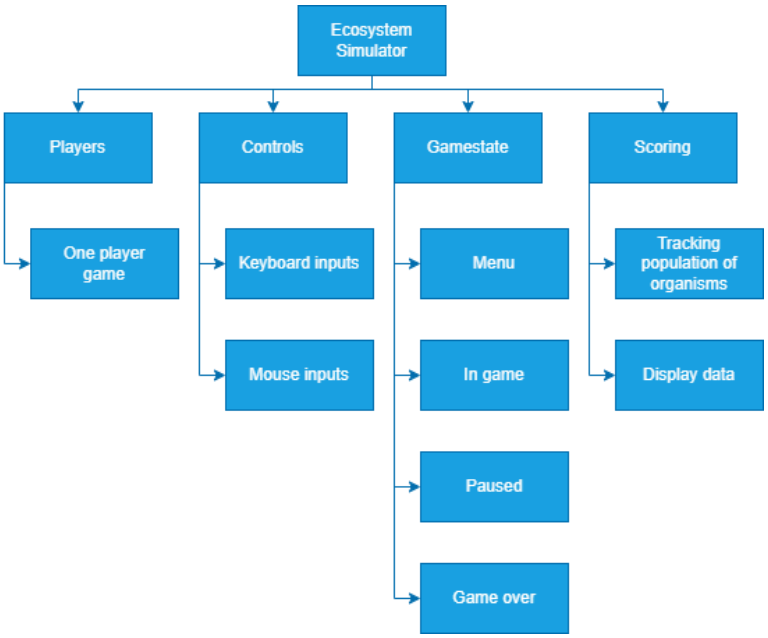
One possible problem that could arise while developing my project is scope creep. This is when developers keep on adding features or functions which are beyond the agreed scope of the project. To circumvent this, I will clearly outline the features I plan on adding in the design phase and will only add additional features if I have spare time at the very end of the project.

Another possible problem that could arise is the optimisation of the game. Video game optimization is how well the game will run on a computer. Factors that come into play with optimisation are the frame rate, resolution and detail of the game. I plan on efficiently optimising my game so that it can be played on various devices. Video games can be optimised by having efficient code and carrying out iterative testing.

1.2.3 Thinking Procedurally and Decomposition

Procedural thinking and decomposition are when you break down a complex problem into smaller, more manageable parts. This is to more effectively approach the problem and solve it part by part rather than tackling it as a singular problem. I'll be utilising subroutines and procedures in my code so that I can make my code more efficient and then I can call them throughout my project to perform certain tasks.

Break down of the problem:



Decomposition is very important in the design and development of my game as it will reduce my time efficiently when my time could be spent working on a fundamental feature rather than a small feature which won't be as impactful and could be removed with decomposition. It is more important focusing on key fundamentals rather than small, unnoticeable details. I will be utilising decomposition heavily in the design process when I'm creating flowcharts. This will also help me create a clearer understanding of features and their designed intentions.

1.2.4 Thinking Logically

One key component of computational thinking is thinking logically. This involves being able to reason and make decisions based on evidence. It's the process of analysing a situation and deciding the most suitable solution for the problem.

An example of thinking logically is deciding which programming language and game engine to use when developing my game. This decision will be based upon logic and reason and I'll pick whichever ones appropriately match the scope and goal of my game. Furthermore, I'll be utilising logical thinking in the research section of my analysis when deciding which features to include in my game. Using evidence from user questionnaires and client interviews, I'll decide which features to include and the importance of each feature. This will give me clarity in the design and development process as I'll know which features to focus more time on.

1.2.5 Thinking Concurrently

Thinking concurrently is when programmers determine which parts of a problem can be solved at the same time. An example of processes occurring concurrently in games is when there are multiple key inputs at the same time. If these inputs could not be processed concurrently then the player would not be able to move forward and jump at the same time for example. Thinking concurrently is what allows us to do this.

- An example of thinking concurrently in my game would be that the different organisms are moving around and interacting with the environment at the same time. It would be extremely impractical if only one organism could move at once and the other had to wait for one to finish what it was doing. Therefore I will make it so all the different organisms can behave at the same time.

- Another implementation of thinking concurrently in my game would be the player interacting with user interfaces, such as viewing an organism's health while the simulation is still occurring, and organisms interacting with the environment at the same time.

1.3 Research

1.3.1 Looking at Previous Games

Importance of Researching Previous Games:

Researching previous games is important since it gives developers an insight into the success and failures of previous games similar to the ones they're planning on developing. This allows them to build off of ideas already established by other games and grow the aspects of the game that work and remove the ones that don't. This will allow them to appeal to people who have played these types of games before and understand what these players want in a game.

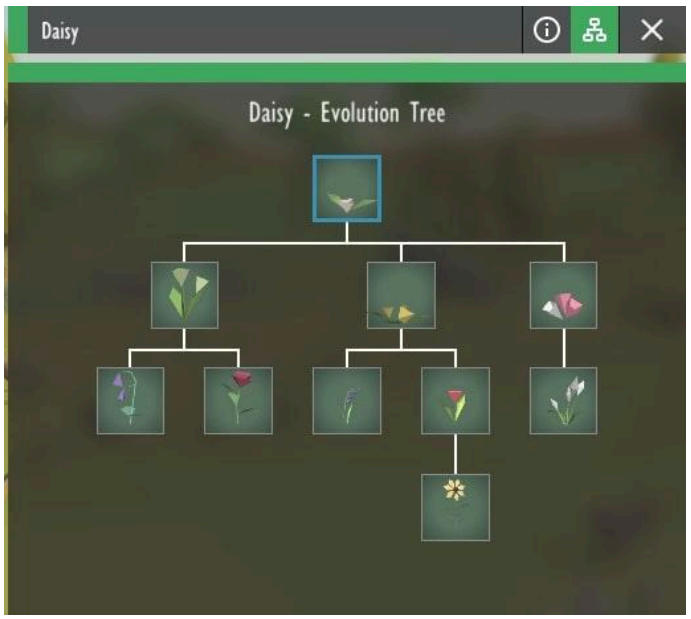
Case Study 1 - Equilinox :
https://www.equilinox.com/

Equilinox is a nature sandbox/simulator game for PC, created by a solo indie developer, Karl Wimble (ThinMatrix), who has documented his development process on his youtube channel. The game aims to create your ecosystem with various animals and plants. Players can manage their ecosystem, breed plants and animals and evolve new species of organisms.

Game Features:

Organism Overview:
In the game, there's a user interface which displays an overview of the organism you select. This allows the player to learn more information about different organisms such as their average life length, their growth time and the category of organism.

Health System:
Players can view the health information of a specific organism. In the screenshot, you can see a user interface box showing the health of a certain apple tree as well as its age, its generation and how far along in growth it is.



Evolution Tree / Game Progression:
Players can evolve and unlock new organisms through a skill tree by spending skill points which can be earned through progressing in the game.

Positive Elements	Negative Elements
• Good progression system with evolution tree • Organism overview gives players key information about different organisms • The health system lets players know the stats of specific organisms in a clear, simple way	• Unlocking some aspects of the game can be time consuming and tedious

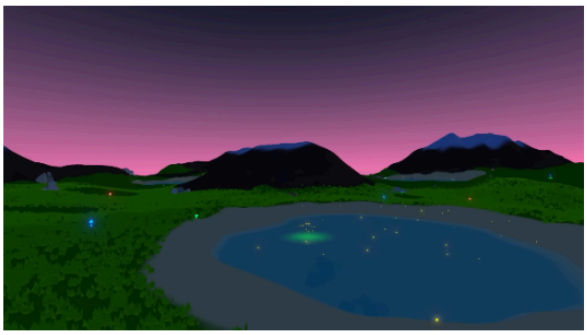
Conclusion:

Equilinox is a calm, relaxing, ecosystem simulator game with a fun progression system. Its low poly graphics allows the player to be immersed in the environment without distracting from the gameplay.

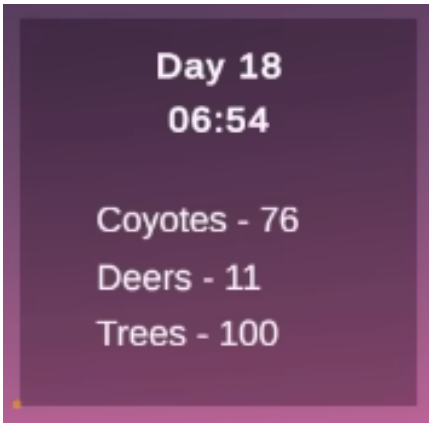
Case Study 2 - VAV Ecosystem Simulation :

I Made an Ecosystem Simulation in Unity. They Evolved. (devlog)

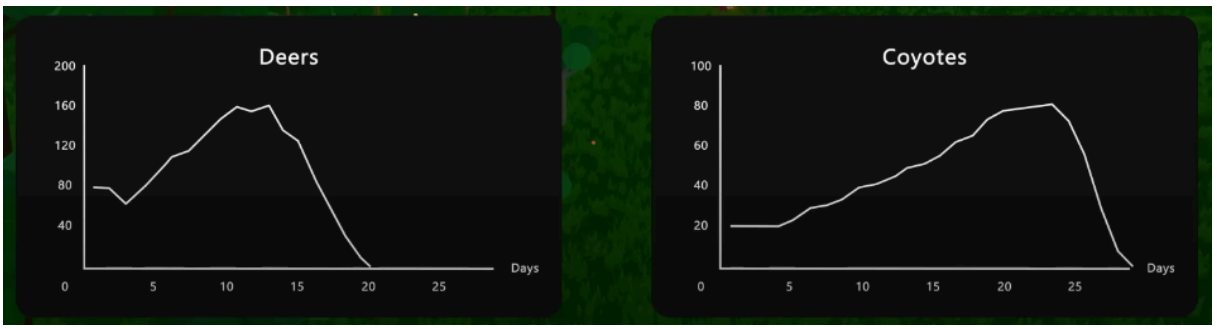
The other game I'll be researching is the ecosystem simulator created by the youtuber VAV with Unity. He created this game for a youtube video, he creates a game where there is an environment with deer, trees and coyotes and their populations are tracked over time. The coyotes hunt the deer and the deer eat the trees.



Organism Behaviour:
Here you can see two deer in the game and above them it shows their current status and their attributes such as energy, water etc. One of the deer is just patrolling the area whereas the other one is walking to reproduce. This lets the player know the status of different organisms.



Time and Population Display:
There's a box in the top right of the screen showing what day the simulation is on as well as the time. It also shows the current population of the coyotes, deer and trees.



Population Data:
This screenshot shows two graphs in the game which are displaying the population of deer and coyotes compared to the number of days which have passed.

Positive Elements	Negative Elements
• Players can easily and quickly see an organism's current behaviour as well as their stats such as hunger and thirst levels • Users can view the current populations of organisms and the time of the simulation easily in the GUI at the top of the screen • Players can see the population of organisms over time and compare the populations of two different organisms to see how they affect each other	• The amount of organisms is limited to only three; coyotes, deers and trees. This makes the gameplay very limiting.

Conclusion:

The ecosystem simulation by VAV is very simplistic, yet efficient in its gameplay to allow the player to understand how ecosystems work. However, the limited amount of organisms within the game restricts how the players can experience the game and limits the replayability.

1.3.2 User Questionnaire

Planning the Questionnaire:

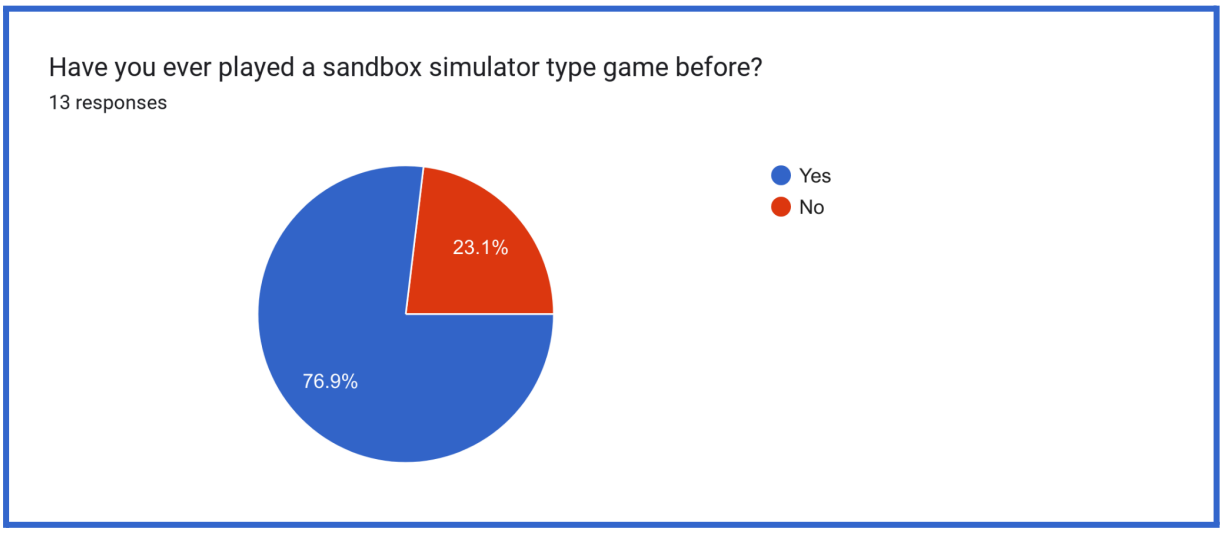
My questionnaire will be a series of yes or no questions sent out to a focus group of 17 year olds. This questionnaire will help me receive general feedback on what most people want in a ecosystem simulator type game and it won't have as much bias as an interview with a single client since the questionnaire will be sent out to a group of people.

Questionnaire Questions:

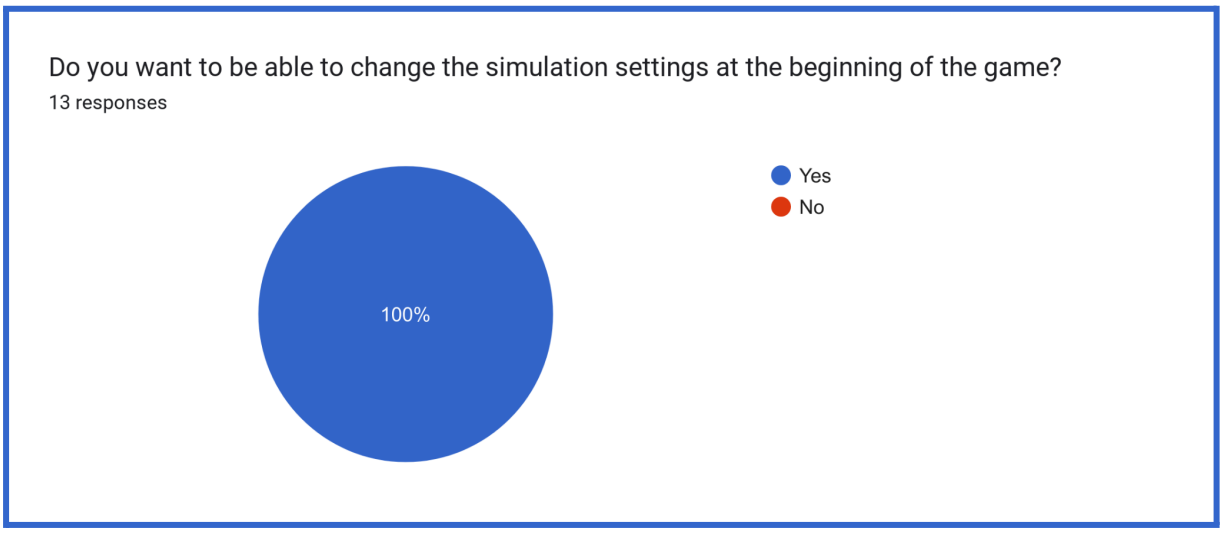
Question	Justification
Have you ever played a sandbox simulator type game before?	Knowing if they have played a sandbox simulator before gives me insight into whether or not they can tell me what they did and did not enjoy from previous ones they've played.
Do you want to be able to change the simulation settings at the beginning of the game?	Lets me know if the focus group wants to be able to customise certain parameters and settings before the simulation starts in the simulation settings menu.
Do you like 3D low poly art style/graphics in games such as "Superhot" and "Muck"?	I had the initial idea to have the graphics of my game to be 3D low poly style so I should ask my focus group if they like these types of graphics before I implement them.
Do you want to be able to view the population of organisms plotted against time throughout the simulation on a graph, once the game has concluded?	Clarifies to me if the focus group wants to be able to view this data at the end of the game.
Do you want a pause/menu button?	Lets me know if the focus group wants the ability to pause the simulation at any given time.
Do you want organisms' status and statistics to be displayed above their head or do you want to have to click on them to view it?	Lets me know whether the focus group wants easy access to the status of the organisms or if they would rather click on them to view it.
Do you want the game to include skill points and a skill tree to evolve organisms, or should the organisms evolve by themselves?	Lets me know whether the focus group would rather have more involvement in the game and have skill points which they spend to evolve organisms or if they want an algorithm which will determine the evolution of organisms.
Do you want to have a game mechanic where you can control the movements and behaviour of a selected organism?	Lets me know if the focus group would want to be able to control a specific organism within the environment or they would rather just observe the environment.

Do you want the game to be multiplayer and online or single-player and offline?	Gives me insight into whether the focus group would rather want the game to just be single player and offline or if they would want to be able to play the game multiplayer as well.
Are there any additional features, that haven't been mentioned, that you would like to see in the game?	The focus group can give any ideas or features they would like to see implemented into the game.

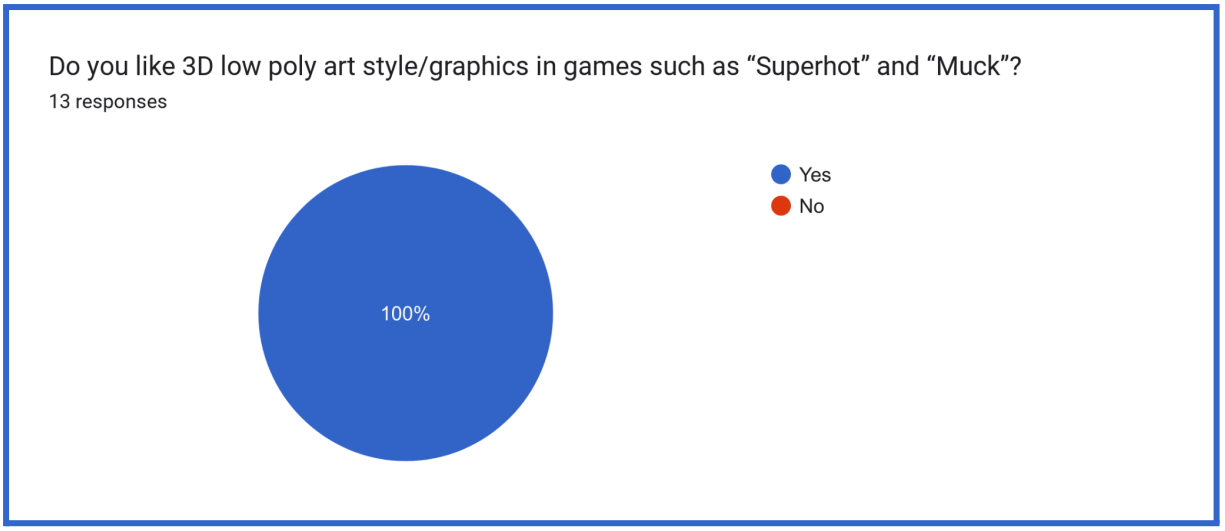
Questionnaire Responses & Review:



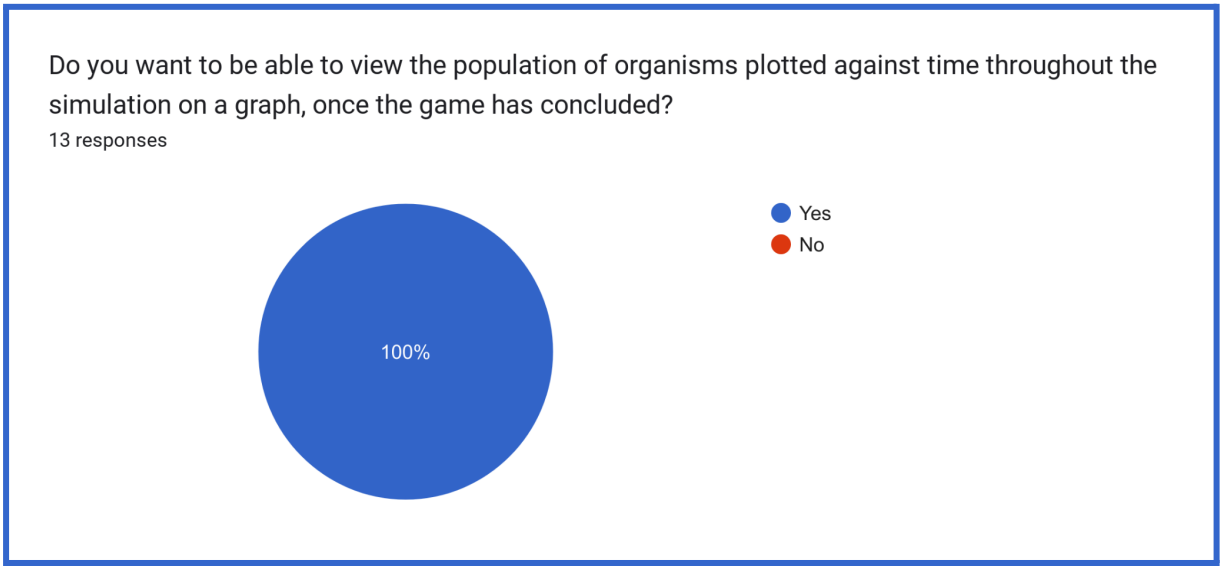
The majority of people have played a sandbox simulator type game before meaning they know what sort of feature and gameplay mechanics it will typically contain. This also means they have formed opinions on whether or not they like these mechanics and gameplay features.



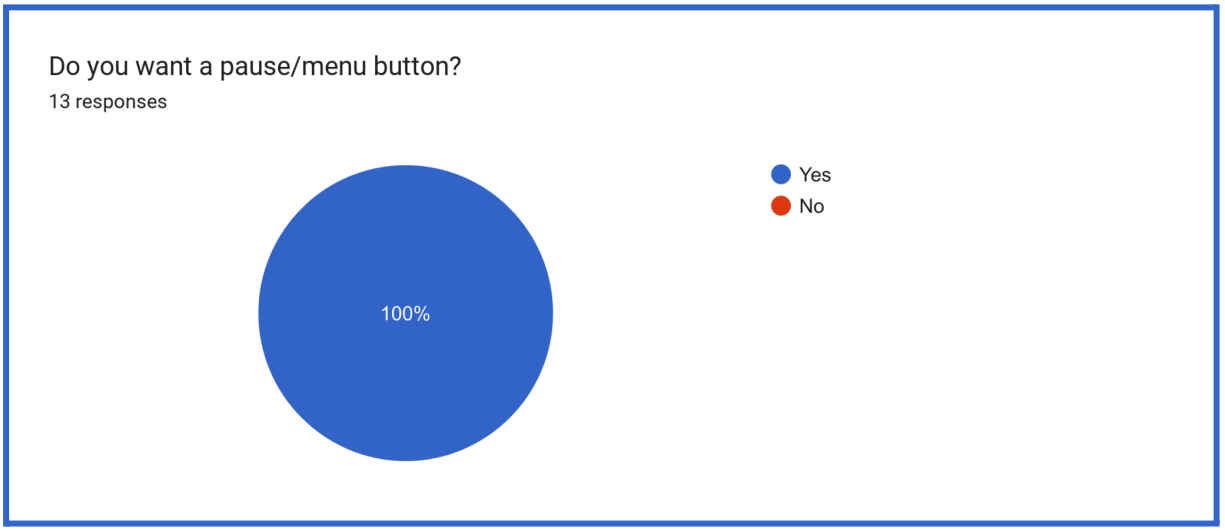
Everybody said that they want to be able to change the simulation settings at the beginning of the game, meaning I will be implementing this as a feature due to the high demand from the focus group.



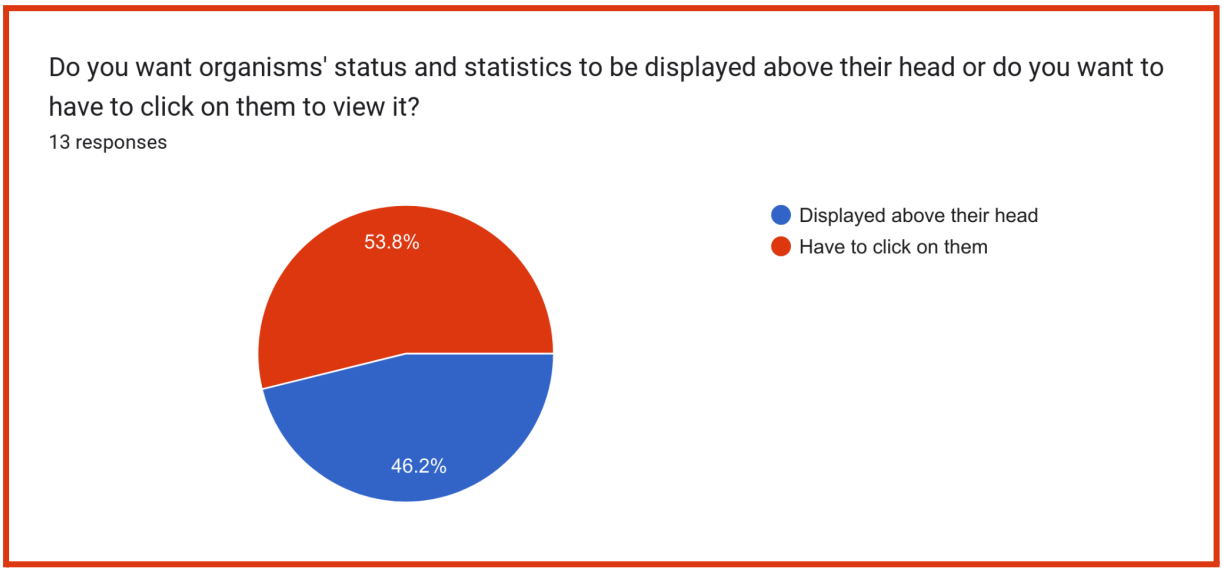
Everybody in the focus group said they like 3D low poly art style/graphics in games such as “Superhot” and “Muck” meaning I will be implementing this into the game since it is in high demand from the focus group and these graphics would be the most enjoyable for the focus group.



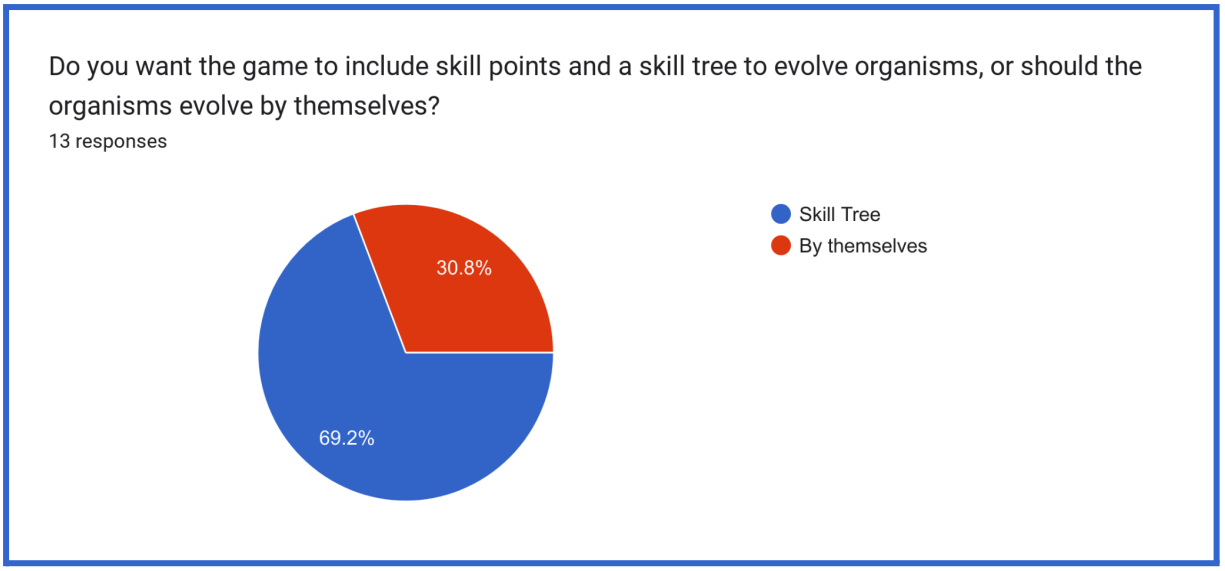
Everybody from the focus group said that they want to be able to view the population of organisms plotted against the simulation on a graph once the game has concluded meaning I will be implementing this since it will allow players to learn how their ecosystem simulation went.



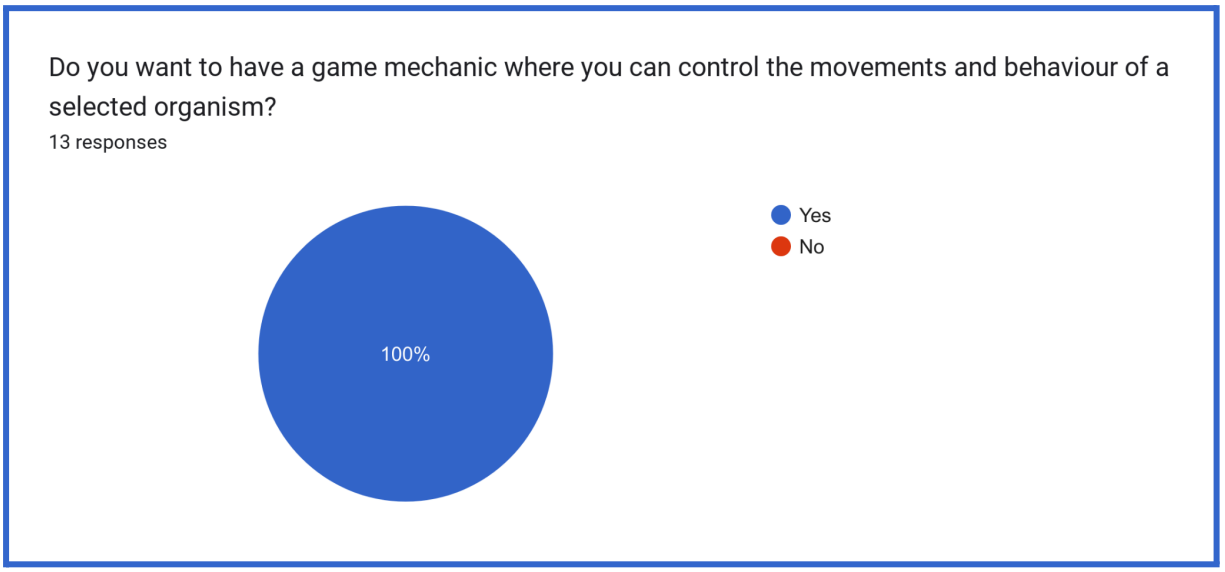
Everybody from the focus group said that they want a pause/menu button. This means I will be implementing a pause/menu button into my game so that players can pause the simulation at any given time and then come back to return where they left off.



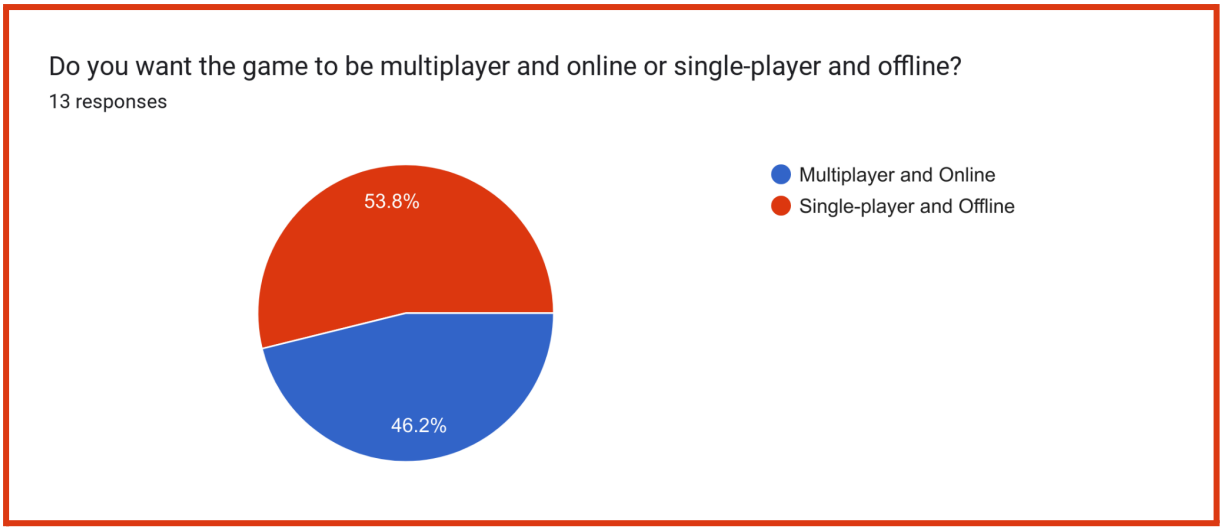
The majority of the focus group said that the way they want to be able to view the organisms' status and statistics is through having to click on them rather than it being displayed above their head. The reason the majority chose to click on them is most likely since if it was displayed above their head it may clutter up the screen.



The majority of the focus group said that they want to be able to gain skill points and evolve the organisms through the use of a skill tree rather than the organism evolving “by themselves” through the use of algorithms. This means that I will be implementing a skill tree and skill points into the game. The reason the majority chose a skill tree is most likely since they want to have more involvement in what happens in the simulation rather than just purely observing it.



Everybody in the focus group said that they want to have a game mechanic where you can control the movement and behaviour of a specific, selected organism. Therefore I will be implementing this feature into my game. The reason everybody from the focus group wants to be able to control an organism is most likely the same reason as the previous question, they want to have input in the simulation rather than just only observing it.



The majority of the focus group decided that they want the game to be single-player and offline rather than it being multiplayer and online therefore I will be making the game a single-player experience which can be played without an internet connection.

Responses To: Are there any additional features, that haven't been mentioned, that you would like to see in the game?

- Animals can form 'tribes' where they stick together and survive together
- Options for customisable map environments - e.g snow, grasslands, jungle
- Customisable organisms
- Day and night cycle that shows how the ecosystem changes at night
- Remainder of the focus group did not answer or answered “no”.

1.3.3 Client Interview

Planning the Interview:

I'll be constructing an interview full of questions where I can find out the basic requirements needed for my game from my client. This will give me a foundation to build upon when designing and developing the game.

Question	Justification
Have you ever played a simulator game before?	This allows me to know the validity of the clients answers as if they have played a simulator before then their input will be more valuable.
If yes, what did you and what did you not enjoy about the game? If no, then why not?	This question follows up from the previous and lets me gain more insight into whether or not the client enjoyed the simulator game(s) they played. If they haven't played one before then it lets me know the reason why.
How important are the graphics?	Allows me to understand how much time I should spend on the graphics since if the graphics are very important to the client then I will spend more time on them. If not, then I'll spend less time.
How important is sound design/effects?	This is similar to the graphics question, understanding how important sound is to the client will allow me to calculate how much time should be spent on them.
Is the replayability of a game important?	This gives me an idea on whether or not I should make my game replayable since if the game is not replayable then the player/client may get bored after the first playthrough.
How realistic should the gameplay be?	Knowing how realistic the gameplay should be will help me in the design and development stage as to how much detail should go into the game and how close it should resemble real life.
What type of environment do you want the game to be set in?	Knowing which environment the client wants the game to be set in may increase their enjoyment while playing if they are more interested in a certain environment than another.
What type of organisms would you like to see in the game?	This is similar to the previous question but is about the organism types instead. I've asked this for the same reason as the one before. It lets me customise the game to be optimised for the clients enjoyment.

What type of behaviours do you want the organisms to have?	Knowing which types of behaviours the client would like to see in the game, possibly from previous environmental simulators they've played, lets me make the game more enjoyable and immersive.
What type of characteristics and stats do you want to see featured in the game, such as hunger levels, thirst levels, etc?	This question will give me insight into which types of characteristics the client wants the organism to have which lets me add more complex features and algorithms into the game which will affect each organism through natural selection, etc. since certain characteristics will be more beneficial than others.
Do you want the game to have a feature where organisms can evolve?	Gives me insight into how complex the client wants the game systems to be.
If the game has a skill tree which is how organisms are evolved, how do you want players to earn the skill points?	This question lets me know in what way the client wants to obtain skill points and progress in that game.
What type of game graphics/style would you want to see?	Tells me if the client wants more simple, decomposed graphics or complex, realistic graphics.
Do you want there to be genetic mutations and if so, what ones do you want there to be in the game?	Give me insight into whether or not the client wants this feature in the game, and if so how it will work.

Interview Script:

Have you ever played a simulator game before?
Yes, some of the simulators I have played before are city skylines, steam engine, and Lawn mower simulator.

What did you and what did you not enjoy about the games?
I enjoyed the simplicity, and freedom to experiment and do as I please in these games.

How important are the graphics?
Graphics in simulation games are important as if it is too detailed it might distract the user from the experience that the game is meant to deliver.

How important is sound design/effects?
Sound design is more important than graphics in my opinion because immersion is very important for the user experience.

Is the replayability of a game important?
The replayability of a game is very important as it will increase the chances of the game having longevity and being fun over a long time.

How realistic should the gameplay be?
The game should be realistic enough to understand what's happening in different situations.

What type of environment do you want the game to be set in?
I want the game to be set in an environment which is grassy or similar to a plains.

What type of organisms would you like to see in the game?
Some of the organisms I would like to see in the game are wolves, deer, sheep and cows.

What type of behaviours do you want the organisms to have?
A type of behaviour the organism could have is hunting where they are searching for prey.

What type of characteristics and stats do you want to see featured in the game, such as hunger levels, thirst levels, etc?
Some of the types of stats I want to see in the game are health, stamina, strength, intellect, thirst and hunger.

Do you want the game to have a feature where organisms can evolve?
Yes the game should have an evolution feature in it.

If the game has a skill tree which is how organisms are evolved, how do you want players to earn the skill points?
The player should earn skill points over time.

What type of game graphics/style would you want to see?
Simplistic graphics similar to games such as lawn mower simulator, so simple graphics.

Do you want there to be genetic mutations and if so, what ones do you want there to be in the game?
Yes, I do want genetic mutations in the game. One of the genetic mutations in the game could be cancer. Another one could be a mutation where this is a chance for one of the stats I listed to be naturally boosted from birth and vice versa .

Review of the Interview:

The purpose of this interview was to help me understand what the client wants from this game. This interview was very insightful as it helped me understand this. From this interview, me and the client established many core fundamentals that are going to be implemented into the game and in what ways.

We established that the simulator should maintain some level of simplicity so that it is not overcomplicated, but it should still contain enough details that it's not oversimplified and still stays true to real life. We decided that the graphics will be an important aspect but should not be too detailed as to not distract the user. Sound design is also important in the game as it will add immersion into the ecosystem.

For the game, we concluded what type of environment the game should be set in, some of the type of organism, types of characteristics and stats that should be in the game, as well as how the evolution and skill point system should be implemented.

Ultimately, this client interview was very successful as it helped me gain knowledge on what the client wants to see in my ecosystem simulation game.

1.4 Requirements

1.4.1 Features of proposed solution

The main proposed features for the Ecosystem Simulator Game:

Feature	Description
Single-player and Offline	The game will be single-player and will be able to be played offline due to the results of the interview and questionnaire.
3D Low Poly Graphics	The game will feature 3D low poly graphics similar to games such as “Muck” and “Superhot” as decided by the results of the interview and questionnaire. Having low poly graphics will also make it so the game will take up less secondary storage and will be easier to run.
Sounds Effects	Add sound effects to the game to make it more immersive and fun to play.
Free Cam View	When the player is not controlling an organism they will be able to observe the simulation in a free cam view where they can fly around the environment.
User Controls	The game will be played using keyboard and mouse inputs.
Simulation Settings Menu	At the beginning of the game, users will be able to configure the simulation settings, as decided by the results of the questionnaire and interview.
Organisms	The game will feature a variety of organisms in the environment.
Organism Attributes	Organisms will have different attributes which will change over time such as hunger, thirst, etc.
Organism Behaviours	Organisms will have different types of behaviours, such as hunting, sleeping, etc.
Organism Genetics	Organisms will carry different genetics such as maximum amount of health, maximum amount of stamina, etc. which will be passed down to their offspring through an algorithm. This will cause and simulate natural selection. Genetics can be dominant or recessive.
Random Organism Mutation	At birth, organisms will have a small chance to obtain a random mutation, this genetic could then affect the natural selection and adaptation of the population. The mutation could be dominant or recessive.
Organism Evolution	Organism will be able to be evolved by the player in a skill tree using skill points earned through playing the game.

Skill Points and Skill Tree	The skill points and skill tree are used by the player to evolve organisms.
Environment	The environment the game will be played in.
Ability to Control Organism	As decided by the results of the questionnaire, players will have the ability to control and play as a specific organism.
Pause Menu	There will be a pause menu so the simulation can be frozen by the user at any given time while playing.
End Screen with Statistics	At the end of the simulation, there will be an end screen displaying the statistics of your playthrough. Such as the population of different organisms against time.

Additional features that may be added if main features are completed:

Organism Tribes	Organisms can form tribes where they will group together and do the same behaviours at the same time. Such as hunting at the same time. They could also share food, etc.
Custom Environments	Ability to change the environment before the game starts to different ones, such as snow, grasslands, jungle.
Customisable Organisms	Players could create their own organisms which they can customise to their own liking and then add into the ecosystem.
Day and Night Cycle	Day and night cycles which will affect the behaviours of animals. For example nocturnal animals will hunt during the night while other organisms are sleeping and vice versa.
Extra Organism Attributes	Add more organism attributes into the game to make it more complex.
Extra Organism Behaviours	Add more organism behaviours into the game to make it more complex.

1.4.2 Limitations of proposed solution

My game will contain limitations due to the nature of the gameplay attempting to emulate real life, it would take extreme complexion to imitate real life on a one to one basis. Therefore limitations will arise while designing and developing the game.

One limitation of my proposed solution is the time constraints in developing the game. Since I have other A level courses and have just under a year to develop the game I will have to manage the scope of the project effectively to make sure it can be completed within the associated time limit. By decomposing problems, I've been able to do just that, as I can make sure the correct time is spent developing each feature of the game. Additionally, I can make sure I don't spend too much time focusing on a singular feature.The justification for this is that my time must be spent efficiently and effectively so that I can complete the game within the scope and time constraints.

Another limitation of my proposed solution is the length a simulation will be. This is because it would be impractical to simulate time in the game identical to real life, as this would mean one day in the game would take as long as a day in real life. This would make playing the game very tedious. Balancing how long each ecosystem environment will last will be very hard to get right. I must make sure that it is not too fast, where the player is not able to process what's occurring, but also not too slow. The justification for this limitation that I've applied is so that the player doesn't get bored while playing.

A final limitation of my proposed solution is the complexity of the game. If it is too complex with too many features it may easily confuse players and discourage them from playing. I should instead focus on a select features and ensure they are as flawless as possible. The justification for this limitation is that it will create a much greater user experience where there are a select few polished features, rather than a large quantity of rushed, unfinished features.

1.4.3 Hardware and Software Requirements

Hardware Requirements:

Requirement	Justification
Keyboard and Mouse	These will be used for the inputs and will let the user interact with the game.
Monitor	Allows the user to graphically view the game on the monitor's screen so that they can interact with it.

Speaker/Headphones	The user will need speakers or headphones to hear the sound in game. The game could be played without sound, however it's not recommended if they want the best experience.
10 GB of Secondary Storage available for developer	If you have one version of unity installed it will take up 7GB of secondary storage so 10GB will be a comfortable amount when developing the game.
1 GB of Secondary Storage available for players	The final version of the game will most likely not take up more than 1GB of secondary storage.
4GB of RAM available	This amount of RAM will be enough to comfortably run and play the game.
CPU	A CPU with x86, x64 architecture with SSE2 instruction set support is required to run Unity.
GPU	A GPU with DX10 (shader model 4.0) capabilities; DirectX 10 capable GPU is required to run Unity.

Software:

Requirement	Justification
Unity Game Engine	The game engine I will be using for my project is Unity. This is a game engine that primarily uses the C# programming language for development.
C# Compiler	The programming language used by unity game engine is C#, therefore a C# compiler is required however this is built into Visual Studio. C# is an object-oriented programming language. This is a programming language that organises software design around data rather than functions. This allows for reusability of code which will be useful in a large project.
Visual Studio	I will be using Visual Studio as my integrated development environment to write my code.
Windows	I will be using the Windows operating system when developing the game as it is the one I am most familiar and comfortable using.

1.4.4 Success Criteria For Each Requirement

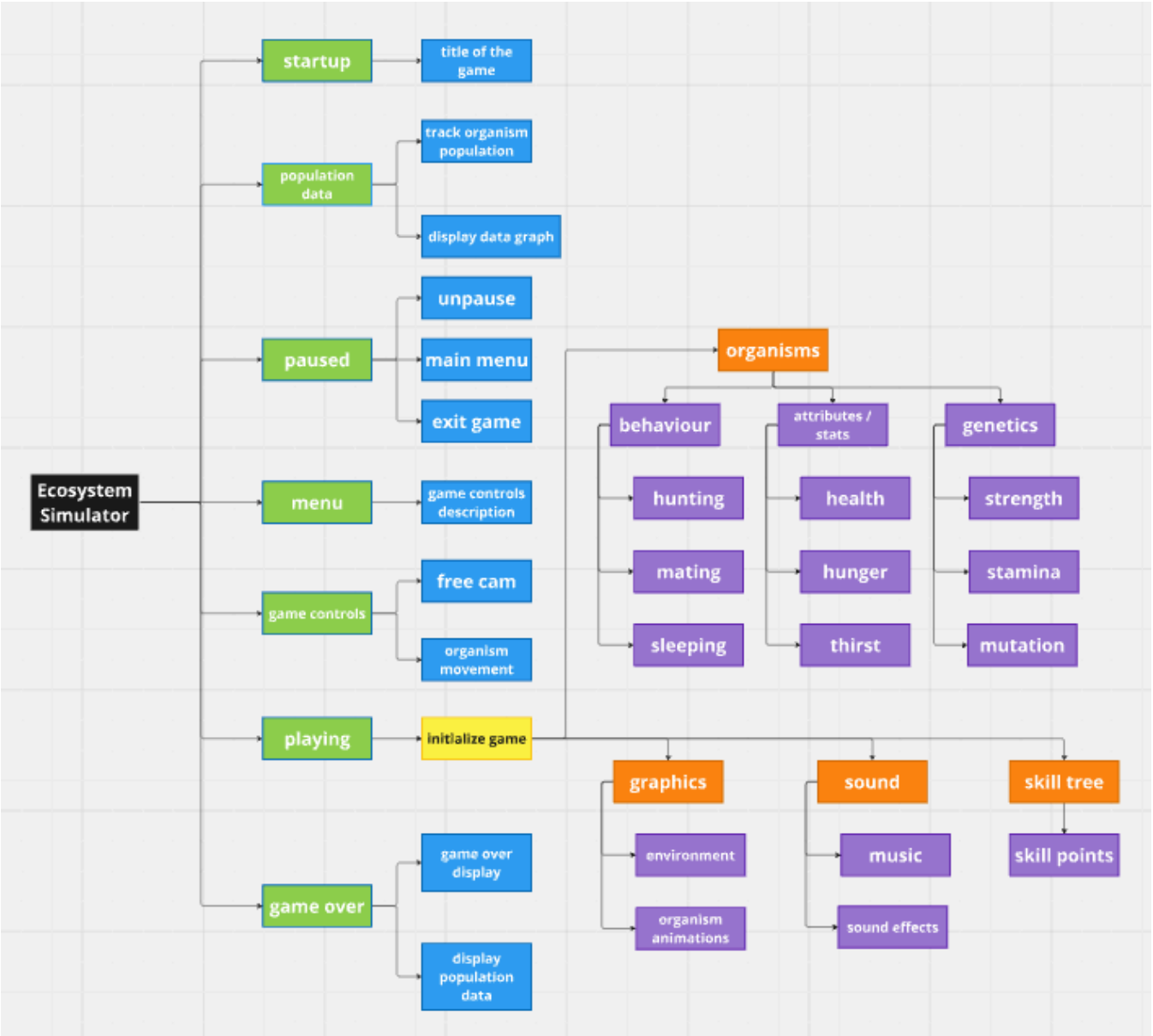
No.	Requirement	Justification	Evidence
1.	Users will have access to a main menu when the game is opened	Allows users to change the simulation settings to their liking before starting the game.	User questionnaire
2.	User will be able to access a free cam to view the environment	The players need a way to be able to view and spectate the ecosystem.	Feature of VAV Ecosystem Simulation
3.	Add the organisms: deer and wolves	Wolves will be the prey who hunt deer, which will eat plants.	N/A
4.	User will be able to select a certain organism and control their movements and behaviours	The user will feel more immersed into the game if they are able to play as one of the organisms.	User questionnaire
5.	Sound effects will be implemented	Adds immersion to the game for the player and makes the game more realistic.	Client Interview
6.	Organism attributes (thirst, hunger and stamina) will be added into the game	Makes each organism unique rather than them all being generic and similar to each other.	Feature of VAV Ecosystem Simulation and results of user questionnaire
7.	Organism behaviours will be added into the game	Adds realism to the game as different organisms will have different tasks and behaviours.	Feature of VAV Ecosystem Simulation and results of user questionnaire
7.1	Organisms will have the behavioural tasks of hunting and eating	Predator organisms will hunt prey as this is what happens in real ecosystems.	Feature of VAV Ecosystem Simulation
7.2	Organisms will have the behavioural tasks of drinking and searching for water	Organisms need water in order to survive, therefore they need to search for water and drink water.	N/A
7.3	Organisms will have the behavioural task of sleeping and being idle	If a day and night cycle is implemented then organisms will sleep as this is realistic.	Questionnaire
7.4	Organisms will have the behavioural tasks of mating and searching for a mate	Organisms will look for mating partners to reproduce with to simulate real life.	Feature of VAV Ecosystem Simulation

8.	Genetic values will be assigned to each individual organisms	Genetics will be used to simulate natural selection and adaptation as the organisms with superior genetics are most likely to survive and pass on those genetics. Genetics to be included are maximum health, hunger, thirst and stamina. Mutations are unique to organism species. Genetics will also either be dominant or recessive.	Feature of VAV Ecosystem Simulation
9.	Organisms will be able to evolve over generations	Organism evolution is a key factor of an ecosystem over a long period of time. Therefore is important to the simulator.	Feature of Equilinox and results of user questionnaire
10.	Offspring will be able to have a chance to inherit a mutational characteristic.	Adds a random variable into the simulation making it more realistic and entertaining as well.	Client interview
11.	Implementations of skill points and a skill tree	Adds a way for players to progress in the game and a way for evolution to work in the simulator.	Feature of Equilinox and results of user questionnaire
12.	Users must be able to access a pause menu while playing the game	Lets the user pause the game to take breaks or if they have to stop and do something in real life and then come back to the game.	User questionnaire
13.	After a simulation has concluded, the end screen will show statistics	The user can see an overview of their simulation and know how different factors and variables affected the populations of different organisms, etc. This will also be useful for users who are using the simulator for educational purposes.	Feature of VAV Ecosystem Simulation and results of user questionnaire
14.	The game will run with stability, optimisations and without major bugs	Game stability and not encountering any major bugs ensures that the game creates a satisfying user experience, since the user will not encounter any optimisation problems or game breaking errors	Computational Methods - Thinking Ahead

Chapter 2 - DESIGN

2.1 Decomposition

2.1.1 Decomposition Overview



Why am I using this approach to help design my problem?

The top down design allows me, the developer, to easily break down each component of the game into easily solvable problems which can all be tackled individually rather than all in one go which would be extremely challenging without any decomposition. It also allows me to incorporate many computational thinking skills such as “thinking ahead” and “thinking logically”.

2.1.2 Explanation of Each Module

Startup

When the game starts a start up menu screen will appear with the game name and/or logo, also possibly a loading bar/wheel.

Population data

Data will be tracked accounting for all organisms which are alive during each time period and this will be categorised by time period with a population count at that time associated with it, you will be able to view/display this data on a graph.

Paused

The game will have the option to be able to pause the game, freezing any environmental interactions until it is unpaused. On this pause menu, you'll be able to return to the main menu or completely exit the game, closing the application.

Menu

On the main menu you will be able to view a list of game controls with the associated keybind.

Game Controls

The game will include controls to be able to operate a free cam where you can "fly" around the map and observe the environment and organisms. You will also be able to control an organism and its movements and environmental interactions.

Playing

Organisms

Behaviour

Organisms will be able to hunt other organisms (prey), mate with the opposite sex to create offspring and sleep.

Attributes / Stats

Individual organisms will have their own attributes/stats such as health, hunger and thirst values.

Genetics

Organisms will have genetic values that they can pass onto their offspring using an algorithm combining and randomising the genetics of the two parents. Examples of genetic values are maximum health and stamina. Offspring can also have a probability to inherit a mutation.

Graphics

Graphics for the environment and organisms and animations will have a 3D low poly art direction

Sound

The game will include background music and sound effects for events.

Skill Tree

The game will include a skill tree with skill points which can be obtained and used to unlock the skills through a tree-like progression system, from the top down.

Game Over

The game will include a game over display when either the total organisms populations are killed off or if the game timer is concluded. At the end of the game a summary of the population data and other statistics will be displayed.

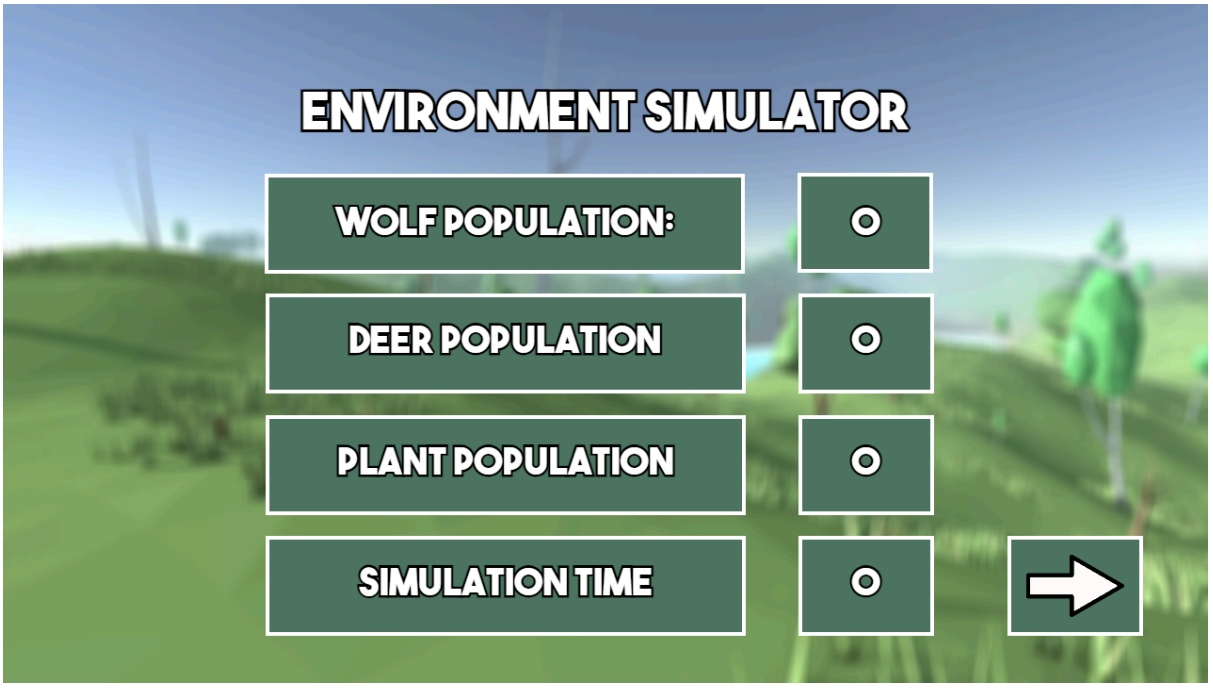
2.2 Menus

2.2.1 Interface Design

Main Menu Selection

Pause Menu Selection

Simulation Initialisation Menu



2.2.2 Pseudocode

Scene State Class

```
// loadScene() is a built-in unity function
class sceneState
     public procedure GameStart()
          loadScene(gamestart)
     endprocedure

     public procedure MainMenu()
          loadScene(mainmenu)
     endprocedure

     public procedure Unpause()
          loadScene(maingame)
     endprocedure

     public procedure Controls()
          loadScene(controls)
     endprocedure

     public procedure Exit()
          quit()
     endprocedure
endclass
```

Main Menu Selection

```
while mainMenuOpen == True
     if clicksStartGame then
          sceneState.GameStart()
          mainMenuOpen = False
     elseif clicksControls then
          sceneState.Controls()
          mainMenuOpen = False
     elseif clicksExit then
          sceneState.Exit()
          mainMenuOpen = False
     endif
endwhile
```

Pause Menu Selection

```
while pauseMenuOpen  == True
     if clicksUnpause then
     7    sceneState.Unpause()
          pauseMenuOpen = False
     elseif clicksControls then
          sceneState.Controls()
          pauseMenuOpen = False
     elseif clicksMainMenu then
          sceneState.MainMenu()
          pauseMenuOpen = False
          mainMenuOpen = True
     endif
endwhile
```

Simulation Initialisation Menu

2.2.3 Justification

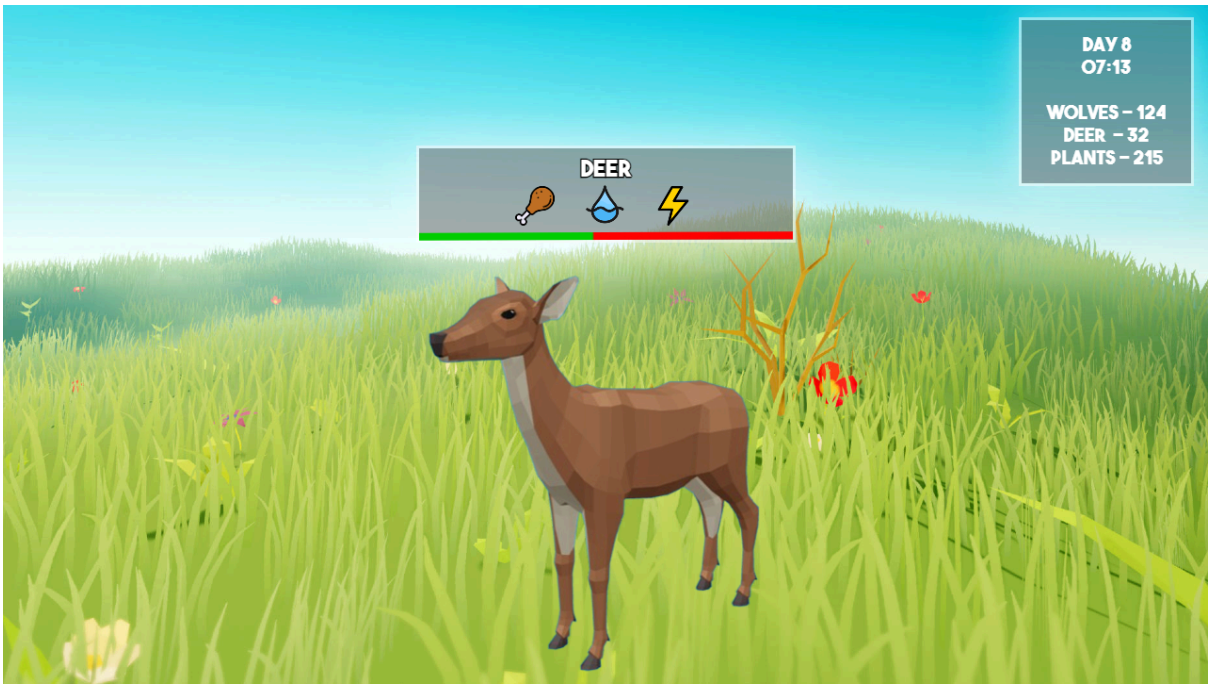
A main menu allows the user to select whether they want to start playing the game, view the game controls or if they would like to exit the game. This is a lot more user friendly than immediately entering the user into the game when they open it.

A pause menu is important since the player should be able to pause the simulation whenever they want and have all the organisms and values not change while the game is paused.

When the user presses “Start Game”, they will be taken to a simulation initiation settings menu. Here they will be able to choose the starting population of the wolves, deer, plants and the simulation time duration. Once all of these fields have been filled, they can press a “Begin Simulation” button which will start the simulation with all their imputed values.

2.3 Environment User Interface

2.3.1 Interface Design



2.3.2 Flowcharts & Pseudocode

Scene States

```
while sceneState == "MainGame"
     if time.end() or organism.population <= 0 then
          sceneState.GameOver()
     endif
endwhile

while sceneState == "Controls"
     if escape_pressed then
          sceneState.MainMenu()
     endif
endwhile
```

Timer

```
import time
timeTotal = 0   // stored as seconds
timerRun = True

while timerRun == True
     if sceneState != “MainGame” then
          timerRun = False
     endif

     time.sleep(1)
     timeTotal += 1
endwhile

timeMinutes = timeTotal DIV 60
timeSeconds = timeTotal MOD 60
timer.Display(timeMinutes, timeSeconds)
```

Day Counter

```
dayNumber == timeTotal DIV 60

// For every minute in real time, one day passes in the game,
gameLength is stored in seconds so 60 is 1 minute
// Will put into the game update function so that it is
constantly updating
```

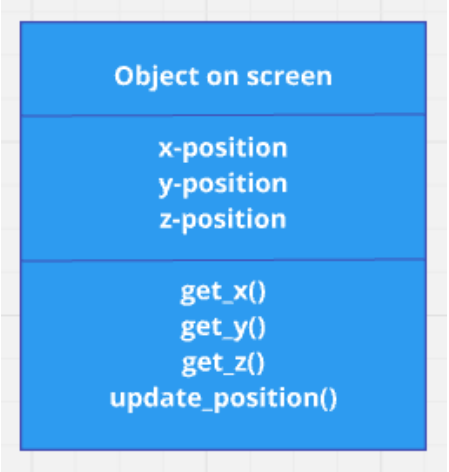
Organism Counter

```
while gameRun = True
     organismA.population == int(input(“Enter starting number
of Organism A: “))

     if organismA.born == True then
          organismA.population += 1
     elseif organismA.dead == True then
          organismA.population -= 1
     endif
endwhile
```

Organism Overhead Display

```
while organism.isAlive == True
     organism_overhead_display = []

     organism_display.append(organism.name, organism.health,
     organism.thirst, organism.hunger, organism.stamina)
endwhile
```

Organism Position Class



2.3.3 Justification

The overhead display on each organism allows the player to easily find out the health, hunger, thirst and stamina of each individual organism. In the top right there will also be a GUI box displaying the day counter and time of the simulation for the user, as well as the population amounts.

2.4 Free Camera View

2.4.1 Pseudocode

Free Cam Movement

How to make a spectator/free camera in Unity! Quick & Easy!

Used the logic from the C# code in this video to create the pseudocode for the Free Cam Movement

```
class freeCam
     public sensitivity       // Mouse sensitivity multiplier
     public slowSpeed         // Camera movement speed when in
slow mode
     public normalSpeed       // Normal camera movement speed
     public sprintSpeed       // Camera movement speed when
sprinting
     public currentSpeed      // Current active movement speed

     // Handle camera rotation based on mouse input
     procedure Rotation():
          // Get mouse input (Y is inverted for proper camera
     control)
          mouseX = get_mouse_x_movement()
          mouseY = -get_mouse_y_movement()

          // Create rotation vector
          mouseInput = Vector3(mouseY, mouseX, 0)

          // Apply rotation based on sensitivity
          rotate_camera(mouseInput * sensitivity)

          // Get current camera angles
          currentRotation = get_camera_rotation_angles()

          // Lock Z rotation to prevent camera roll
          set_camera_rotation(
               x = currentRotation.x,
               y = currentRotation.y,
               z = 0
          )
     endprocedure

     // Handle camera movement based on keyboard input
     procedure Movement():
```

```
        // Get WASD input
        horizontal = get_horizontal_input()  // A/D keys
        vertical = get_vertical_input()      // W/S keys

        // Create movement vector
        movementInput = Vector3(horizontal, 0, vertical)

        // Determine movement speed
        if left_shift_key_pressed then
             currentSpeed = sprintSpeed
        elseif left_alt_key_pressed then
             currentSpeed = slowSpeed
        else
             currentSpeed = normalSpeed
        endif
    endprocedure

    // Move camera based on input and speed
    move_camera(movementInput * currentSpeed * deltaTime)
endclass
```

2.4.2 Justification

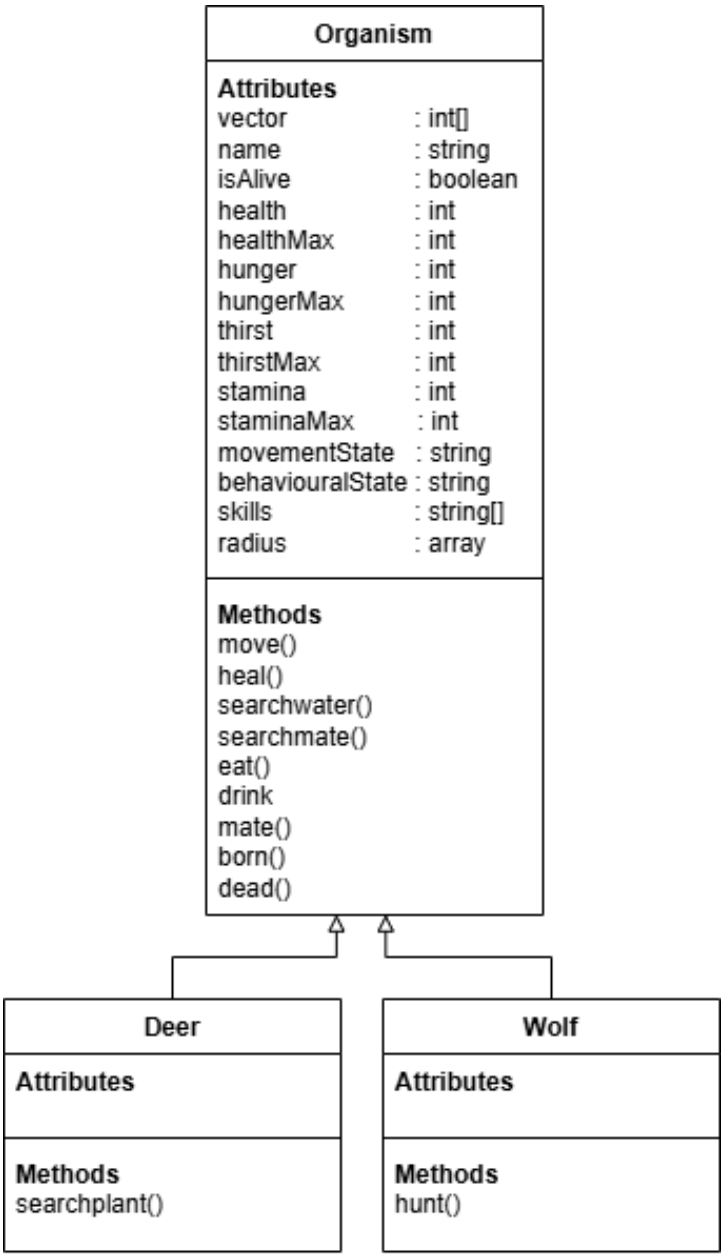
A free cam allows the user to navigate the environment and view the status of the simulation. Such as the behaviour of different organisms and their current status and attributes. This can be viewed by clicking on the animal while in the free cam view and then the organism attributes and details will be displayed. All organisms also have an overhead display view above them while the user is in free cam view.

2.5 Organism Class Diagrams

Class diagrams are a standard way of displaying an object-oriented design. A class contains attributes and methods. Subclasses can be created from a parent class where they inherit the attributes and methods and can add on more. My parent class will be organisms and I will have sub-classes called deer and wolf which inherit from the organism class. An object can be created from a class which in this case would be an individual organism such as a specific deer in the simulation.

2.5.1 Pseudocode

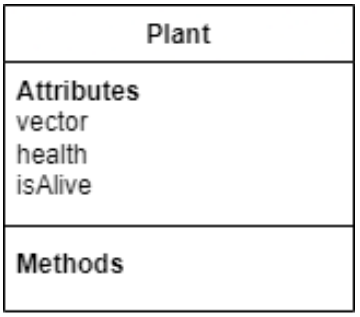
Organism Class OOP



The possible values of the movementState are “walking” or “sprinting”.

The possible values of behavioural state “hunting”, “searchWater”, “eating”, “drinking”, “searchMate”, “mating”, “sleeping” and “idle”.

Plant Class OOP



I will create a separate class for the plants which the deer eat as a plant does not need most of the attributes and methods from the organism parent class so it does not make sense for it to be a subclass.

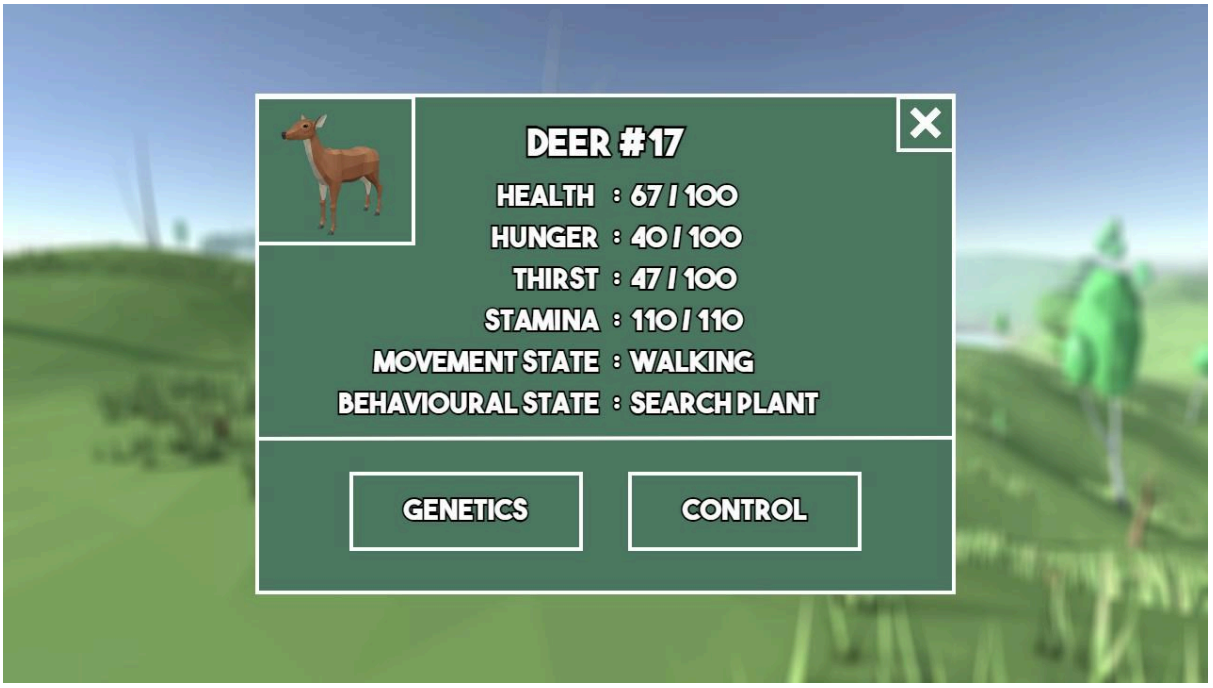
2.5.2 Justification

My game will use classes and OOP to manage the different types of organisms as well as the plants which can be eaten by the deer. The parent class is called an organism which includes all the attributes and methods that all different child classes will utilise. There are two child classes of the organism parents class; these are the deer and the wolf. The deer has the unique method of “searchfood()” which is where the deer will search for the nearest plant. The wolf has the unique method of “hunt()” which is where the wolf will search for the nearest deer to eat.

2.6 Organism Attributes

Each individual organism has their own attribute values. These attributes include health, hunger, thirst, stamina, speed and sight. The maximum value of these attributes is affected by genetics and mutations. This makes each organism unique rather than them all being generic and similar to each other.

2.6.1 Interface Design



When clicking on an individual organism in the simulation it will display the attributes of the organism, the movement state, behavioural state and type of organism. You will also be able to view the genetics of the organisms using the “genetics” button or control the organism by using the “control” button. There is also an exit button represented by an “X” in the top right corner which will return the user to the simulation.

2.6.2 Pseudocode

Hunger

```
import time

while organism.hunger <= 10 and organism.isAlive
     time.sleep(1)
     organism.health -= 3
     organism.stamina -= 3
endwhile
```

Thirst

```
import time

while organism.thrist <= 10
     time.sleep(1)
     organism.health -= 3
     organism.stamina -= 3
endwhile
```

Stamina

```
if organism.stamina <= 0 then
     organism.movementState = "walking"
elseif organism.stamina >= 25 then
     organism.movementState = "running"
```

2.6.3 Justification

These three attributes will affect the movement speed of the organism. So if a deer has a hunger and thirst value of 0 then it will be left vulnerable to predators since it will be losing health and stamina until it can find food and water to regain hunger, thirst and stamina. Eventually its stamina will reach the value 0 and it will be forced to have to be in a walking state meaning predators who are running can easily attack them. This results in population changes within the environment, simulating natural selection. If a deer can maintain its hunger and thirst values, then it'll be able to run away from predators and survive.

2.7 Organism Behaviours

Different organisms have different behaviours depending on whether they are a predator or prey. The two main organisms in my game will be wolves and deer. The wolves will be the predators who hunt the deer and the deer are herbivores so they'll eat plants scattered around the environment.

2.7.1 Pseudocode

WOLF BEHAVIOURS

```
if wolf.hunger <= (wolf.hungerMax / 2) and wolf.hunger <
wolf.thirst then
     wolf.behaviouralState = "hunting"
elseif wolf.thirst <= (wolf.thirstMax / 2) and wolf.thirst <
wolf.hunger then
     wolf.behaviouralState = "searchWater"
elseif wolf.thirst <= (wolf.thirstMax / 2) and wolf.hunger <=
(wolf.hungerMax / 2) and wolf.thirst == wolf.hunger then
     wolf.behaviouralState = "searchWater"
elseif deer.dead() in wolf.radius then
     wolf.behaviouralState = "eating"
elseif water in wolf.radius then
     wolf.behaviouralState = "drinking"
elseif water in wolf.radius and deer.dead in wolf.radius then
     if wolf.thirst <= wolf.hunger then
          wolf.behaviouralState = "drinking"
     else
          wolf.behaviouralState = "eating"
     endif
elseif wolf.hunger > (wolf.hungerMax / 2) and wolf.thirst >
(wolf.thirstMax / 2) and wolf.stamina > (wolf.staminaMax / 2)
then
     wolf.behaviouralState = "searchMate"
elseif mate in wolf.radius and wolf.hunger > (wolf.hungerMax /
2) and wolf.thirst > (wolf.thirstMax / 2) and wolf.stamina >
(wolf.staminaMax / 2) then
     wolf.behaviouralState = "mating"
elseif dayCycleState == "night" then
     wolf.behaviouralState = "sleeping"
else
     wolf.behaviouralState = "idle"
endif
```

DEER BEHAVIOURS

```
if wolf in deer.radius then
     deer.behaviouralState = “runFromWolf”
elseif deer.hunger <= (deer.hungerMax / 2) and deer.hunger <
deer.thirst and wolf not in deer.radius then
     deer.behaviouralState = “searchPlant”
elseif deer.thirst <= (deer.thirstMax / 2) and deer.thirst <
deer.hunger and wolf not in deer.radius then
     deer.behaviouralState = “searchWater”
elseif deer.thirst <= (deer.thirstMax / 2) and deer.hunger <=
(deer.hungerMax / 2) and deer.thirst == deer.hunger and wolf
not in deer.radius then
     deer.behaviouralState = “searchWater”
elseif plant in deer.radius and wolf not in deer.radius then
     deer.behaviouralState = “eating”
elseif water in deer.radius and wolf not in deer.radius then
     deer.behaviouralState = “drinking”
elseif water in deer.radius and plant in deer.radius and wolf
not in deer.radius() then
     if deer.thirst <= deer.hunger then
          deer.behaviouralState = “drinking”
     else
          deer.behaviouralState = “eating”
     endif
elseif deer.hunger > (deer.hungerMax / 2) and deer.thirst >
(deer.thirstMax / 2) and deer.stamina > (deer.staminaMax / 2)
and wolf not in deer.radius then
     deer.behaviouralState = “searchMate”
elseif mate in deer.radius and deer.hunger > (deer.hungerMax /
2) and deer.thirst > (deer.thirstMax / 2) and deer.stamina >
(deer.staminaMax / 2) and wolf not in deer.radius then
     deer.behaviouralState = “mating”
elseif dayCycleState == “night” then
     deer.behaviouralState = “sleeping”
else
     deer.behaviouralState = “idle”
endif
```

2.7.2 Justification

Depending on the attributes of the organism it'll have a different assigned behavioural task. For example, if an organism's thirst is below a certain value the organisms will search for water and once the water is in the organisms sight view it will drink it and its thirst value will then increase. The organisms wolves and deer have slightly different behavioural tasks since a wolf will hunt for deer if its hunger, thirst and stamina are above a certain threshold, but a deer will search for plants to eat instead.

2.8 Organism Genetics & Mutation

List of Genetics:
Health Maximum Value
Hunger Maximum Value
Thirst Maximum Value
Stamina Maximum Value
Speed Maximum Value
Sight Value // Radius range of organism sight

2.8.1 Pseudocode

```
import random

healthMultiplier = float(random.uniform(0.9,1.1))
hungerMultiplier = float(random.uniform(0.9,1.1))
thirstMultiplier = float(random.uniform(0.9,1.1))
staminaMultiplier = float(random.uniform(0.9,1.1))
speedMultiplier = float(random.uniform(0.9,1.1))
sightMultiplier = float(random.uniform(0.9,1.1))

healthMutation = int(random.randint(0, 99))
hungerMutation = int(random.randint(0, 99))
thirstMutation = int(random.randint(0, 99))
staminaMutation = int(random.randint(0, 99))
speedMutation = int(random.randint(0, 99))
sightMutation = int(random.randint(0, 99))

if organism.born() then
     organism.healthMax = ((parentA.healthMax +
     parentB.healthMax) / 2) * healthMultiplier
     organism.hungerMax = ((parentA.hungerMax +
     parentB.hungerMax) / 2) * hungerMultiplier
     organism.thirstMax = ((parentA.thirstMax +
     parentB.thirstMax) / 2) * thirstMultiplier
     organism.staminaMax = ((parentA.staminaMax +
     parentB.staminaMax) / 2) * staminaMultiplier
     organism.speedMax = ((parentA.speedMax +
     parentB.speedMax) / 2) * speedMultiplier
     organism.sightMax = ((parentA.sightMax +
     parentB.sightMax) / 2) * sightMultiplier

     if healthMutation == 50 then
          organism.healthMax = organism.healthMax * 1.25
     elseif hungerMutation == 50 then
```

```
        organism.hungerMax = organism.hungerMax * 1.25
    elseif thirstMutation == 50 then
        organism.thirstMax = organism.thirstMax * 1.25
    elseif staminaMutation == 50 then
        organism.staminaMax = organism.staminaMax * 1.25
    elseif speedMutation == 50 then
        organism.speedMax = organism.speedMax * 1.25
    elseif sightMutation == 50 then
        organism.sightMax = organism.sightMax * 1.25
    endif

    healthMax = round(healthMax)
    hungerMax = round(hungerMax)
    thristMax = round(thristMax)
    staminaMax = round(staminaMax)
    speedMax = round(speedMax)
    sightMax = round(sightMax)
endif
```

2.8.2 Justification

Random multipliers are used to calculate the max values of each attribute for the new organism object when it is born. This is done by using the average of the two parent maxes and then multiplying it by a random decimal between 0.9 and 1.1.

There is a 1% chance for each attribute that it can be mutated. If an attribute is mutated then the max value that has already been calculated will be increased by 25%.

The final max value for each attribute is then rounded to the nearest integer.

2.9 Organism Pathfinding

For the organism pathfinding I will be using the A* algorithm. The A* algorithm is a pathfinding algorithm commonly used in video games. The A* algorithm is used to approximately find the shortest path between two nodes. It utilises two cost functions; the actual cost and an approximate cost. In my game the approximate cost will be the diagonal distance movement cost as organisms in my game can move in any direction as it is 3D. In comparison, grid-based games would use manhattan distance as movement is restricted to just up, down, left and right. This organism pathfinding will be used for the behavioural functions of hunting, searching for water, searching for a mate and searching for plants.

For the pseudocode below I have used this resource to help make it : https://www.geeksforgeeks.org/a-search-algorithm

2.9.1 Pseudocode

```
openList = {}
closedList = {}

startingNode = node that organism.vector is in
openList += startingNode

while openList not empty
     find node with least f in openlist
          node = q
          openList.pop(q)
     get q's 8 successors (the surrounding nodes of q) and set
     their parents to q

     for each successor
          if successor = goal then
               stop search
          else
               successor.g = q.g + distance between successor
               and q
               successor.h = distance from goal to successor
               successor.f = successor.g + successor.h
          endif

          if node with same position as successor in openList
          with lower f than successor then
               skip successor
          elseif node with same position as successor in
          closedList with lower f than successor
```

```
                skip successor
            else
                openList += node
            endif
        next successor

        closedList.push(q)
endwhile
```

2.9.2 Justification

Pathfinding is needed in my game for an organism to make it from its current position to a different, desired position, whether that be hunting an enemy organism to eat it or going to a lake to drink water from it.

2.10 Player Controlled Movement

In my game players will be able to click on a specific organism and control the movement of that organism seen in the organism attributes menu (2.6)

2.10.1 Pseudocode

```
if in attributesMenu and clicksControl then
     playerControlledMovement = True

while playerControlledMovement == True
     class playerController
          public organism.vector
          public velocity
          public normalSpeed
          public sprintSpeed
          public currentSpeed
          public gravity = -9.81
          public grounded = False

          public procedure Update(deltaTime)
               if w_key_pressed then
                    organism.vector(z) += 1
               endif
               if s_key_pressed then
                    organism.vector(z) -= 1
               endif
               if a_key_pressed then
                    organism.vector(x) -= 1
               endif
               if d_key_pressed then
                    organism.vector(x) += 1
               endif

               if left_shift_key_pressed then
                    currentSpeed = sprintSpeed
               else
                    currentSpeed = normalSpeed
               endif

               Movement = moveDirection * currentSpeed *
               deltaTime

               if not ground then
                    velocity.y += gravity * deltaTime
```

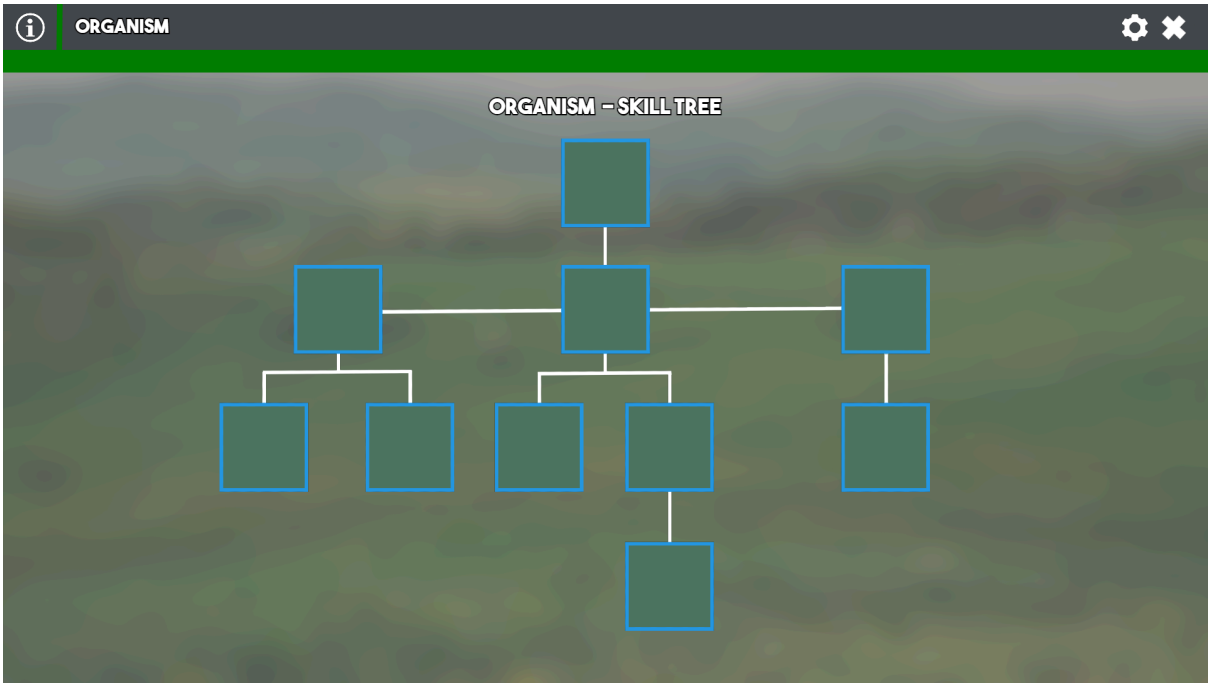
```
                else
                     velocity.y = 0
                endif

                if space_key_pressed and grounded then
                     velocity.y = jumpForce
                     grounded = False
                endif

                position.y += velocity.y * deltaTime

                if position.y <= 0 then
                     position.y = 0
                     grounded = True
                endif

                organism.vector = position
           endprocedure
     endclass
endwhile
```

2.10.2 Justification

Allowing the player to control a specific organism within the organism can make it so if the player is bored of just watching the simulation then they can actually interact with it and affect it through the organisms movements. It can also add more immersion to the game.

2.11 Skill Tree & Evolution

2.11.1 Interface Design



2.11.2 Pseudocode

```
if skill above skill is unlocked then
     skills.unlockable = True
endif

if clicksSkills and skill.unlockable == True then
     skill.unlocked = True
     organism.skills += skill
endif
```

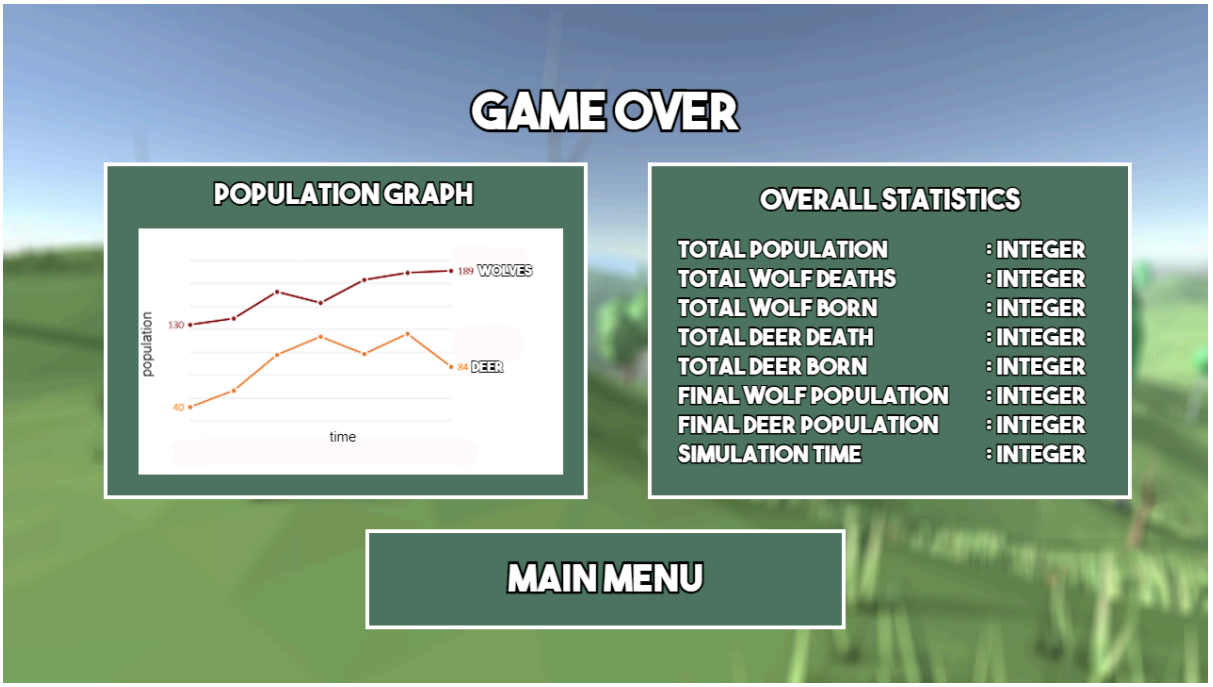
2.11.3 Justification

A skill can only be unlocked if the skill above it in the skill tree is unlocked or there is no skill above it. So the player has to work from top to bottom when unlocking skills.

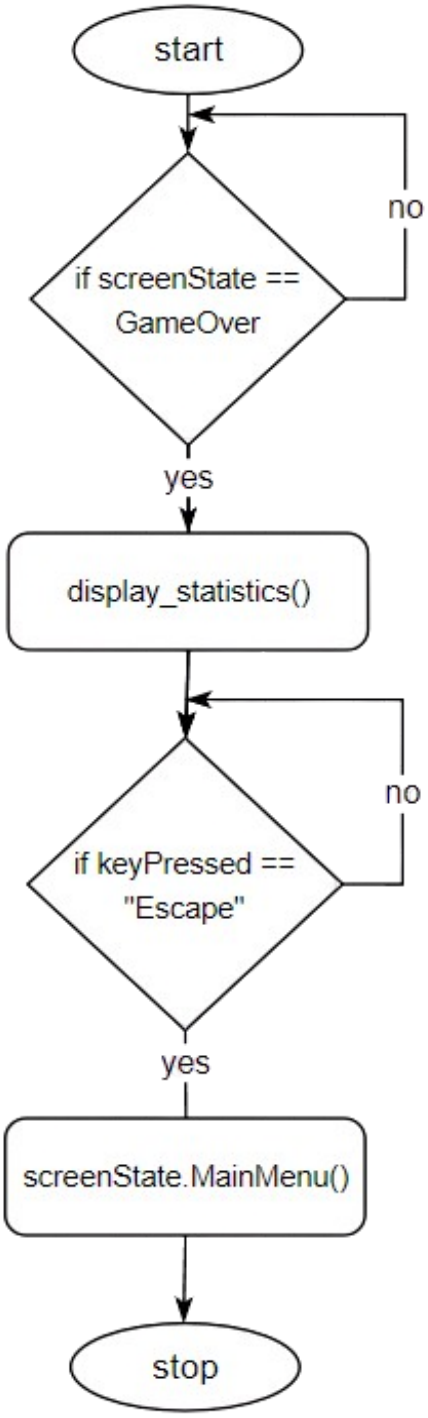
2.12 Game Over Screen

The game over screen will display a graph of the populations of each organism against the simulation time. It will also display the overall statistics such as the total population at the end of the simulation, total death and births of each organism, the final population of each organism and the total time length of the simulation. There will also be a button to return back to the main menu.

2.12.1 Interface Design



2.12.2 Flowchart



2.12.3 Justification

The game over screen with the graph and overall statistics gives the user a clear overview of how their simulation went and gives clear data of the simulation.

2.13 Data Structures

Section	Name	Data Type	Justification
2.2 - Menus	mainMenuOpen	Boolean	If the main menu is open
	pauseMenuOpen	Boolean	If the pause menu is open
2.3 - Environment User Interface	sceneState	String	MainGame or Controls or GameOver
	timeTotal	Integer	Stores running time of current simulation in seconds
	timerRun	Boolean	If the timer is running
	timeMinutes	Integer	Stores the minutes of the total time
	timeSeconds	Integer	Stores the seconds of the total time
	dayNumber	Integer	Stores the total number of days that have occurred in the current simulation
	gameRun	Boolean	If a simulation is current running
	organismA.population	Integer	Stores total population of an organism type (organismA in this example)
	organism_overhead_display	Array	Array containing all the information that will display above an organism (name, health, thirst, hunger, stamina)
2.4 - Free Cam View	sensitivity	Float	Stores the mouse sensitivity of the player
	slowSpeed	Float	Stores the slow movement speed of the free cam
	normalSpeed	Float	Stores the normal movement speed of the free cam
	sprintSpeed	Float	Stores the sprint movement speed of the free cam
	currentSpeed	Float	Stores the current movement speed of the free cam
	mouseX	Float	Stores the mouse X movement
	mouseY	Float	Stores the mouse Y movement
	mouseInput	Array	Array used to create a rotation vector using mouseY and mouseX
	currentRotation	Array	Array storing the current camera angles

	x	Float	Stores current X rotation
	y	Float	Stores current Y rotation
	z	Float	Stores current Z rotation
	horizontal	Boolean	Gets the horizontal input (WASD)
	vertical	Boolean	Gets the Vertical input (WASD)
	movementInput	Array	Vector for movement
2.5 - Organism Class Diagrams	vector	Array	Array containing current vector position of organism
	name	String	Stores organism name (e.g : “Organism #152”)
	isAlive	Boolean	If organism is alive
	health	Integer	Stores current health of organism
	healthMax	Integer	Stores max possible health of organism
	hunger	Integer	Stores current hunger of organism
	hungerMax	Integer	Stores max possible hunger of organism
	thirst	Integer	Stores current thirst of organism
	thirstMax	Integer	Stores max possible thirst of organism
	stamina	Integer	Stores current stamina of organism
	staminaMax	Integer	Stores max possible stamina of organism
	movementState	String	Stores current movement state of organism
	behaviouralState	String	Stores current behavioural state of organism
	skills	2D array	Array containing all the skills that the organism can unlock and the status of if they are unlocked or not
	radius	Array	Array containing the sight range the organism can see using a radius vector
2.7 - Organism Behaviours	dayCycleState	String	Stores the current day night cycle state

2.8 - Organism Genetics & Mutation	healthMultiplier	Float	Stores the health multiplier for genetics that is randomly generated
	hungerMultiplier	Float	Stores the hunger multiplier for genetics that is randomly generated
	thirstMultiplier	Float	Stores the thirst multiplier for genetics that is randomly generated
	staminaMultiplier	Float	Stores the stamina multiplier for genetics that is randomly generated
	speedMultiplier	Float	Stores the speed multiplier for genetics that is randomly generated
	sightMultiplier	Float	Stores the sight multiplier for genetics that is randomly generated
	healthMutation	Integer	Stores the probability that health mutates
	hungerMutation	Integer	Stores the probability that hunger mutates
	thirstMutation	Integer	Stores the probability that thirst mutates
	staminaMutation	Integer	Stores the probability that stamina mutates
	speedMutation	Integer	Stores the probability that speed mutates
	sightMutation	Integer	Stores the probability that sight mutates
2.9 - Organism Pathfinding	openList	2D array	3D array containing open list of vectors of nodes
	closedList	2D array	3D array containing closed list of vectors of nodes
	startingNode	Array	Array storing vector of starting node
	node	Array	Stores vector of current node
	q	Array	Temp array for node vector
	successor	Array	Contains array of successor for node
2.10 - Player Controlled Movement	playerControlledMovement	Boolean	If player controlled movement is being used
	velocity	Float	Stores current velocity value

	normalSpeed	Float	Stores the normal movement speed of the player
	sprintSpeed	Float	Stores the sprint movement speed of the player
	gravity	Constant Float	Stores value of gravity
	grounded	Boolean	If the player is touching the ground
	currentSpeed	Float	Stores the current movement speed of the player
	moveDirection	Float	Stores the value of the current movement direction of the player
	deltaTime	Float	Stores current time
	Movement	Float	Stores movement from calculation of currentSpeed * moveDirection * deltaTime
2.11 - Skill Tree & Evolution	unlockable	Boolean	If a skill is able to be unlocked
	unlocked	Boolean	If a skill is unlocked

2.14 Usability Features

My game will be only available to people with access to computers or laptops, the resolution of the game is going to be 1920x1080. It'll also only be compatible with Windows 10 and 11 since these are the operating systems I will be building and testing for. My game is very suitable to play for a wide range of audiences of different ages due to there being minimal controls or reading required.

A key usability issue with games is colour. Many people have difficulty distinguishing between light and dark colors as well as color blindness. This is why I will be adding colour settings to my game. In the settings menu I'll be adding a color blind mode for deuteranopia (red-green), protanopia (red-green) and tritanopia (yellow-blue). This will make the game more accessible to users with colorblindness who may have difficulty playing the game without this setting. I will also add a brightness setting where the user has a slider from 0 to 1 and can change the brightness level. This will be useful if people are playing the game at different times of the day since at night the game could be too bright and they might need to lower the brightness and vice-versa.

Another potential usability issue is the language of the game. 1.35 billion out of 8 billion people speak english. So if my game is only in English that leaves 6.65 billion people not being able to understand my game. I've consulted my client on this issue and he said that adding other languages to the game will be very time consuming and the translation will most likely be inaccurate so I should not focus my time on adding multiple languages to the game. Therefore I won't be adding support for multiple languages, only English.

The main font used in the game will be Liberation Sans as it is very readable across a variety of different sizes and colours. The color scheme of my game is going to be simple and universal across the game to make sure it has a familiar feel throughout the whole time playing it.

Additionally, the user interfaces are going to be easily accessible and clear to navigate. The name of the buttons is going to be simple so that users can easily understand what they do, such as the "Settings" button will take you to the settings menu.

The controls will be simple and similar to other games so that players don't have a confusing time learning how to play and navigate the game. In the main menu and the settings menu there will be an option to view all the different controls so that the player can easily figure them out at any time during the gameplay or before the game starts.

2.15 Testing Plan

My testing plan will include two types of testing; iterative and post testing. Iterative testing is a form of white-box testing. This means that the tests are based on the code that is written. It can be used for checking if individual algorithms or sections of code are working as intended. Post-development testing is a form of black-box testing. It is only carried out once the entire solution is complete, contrasting iterative which is carried out while the solution is still being developed. Testing is useful as it allows me, the developer, to locate and fix any issues with the program.

2.15.1 Iteratively Testing

Iterative testing focuses on if the system itself is functioning. An example of iterative testing would be if you have a function that adds two numbers together, then you should use iterative testing to check if the correct number is returned as a result of adding these two numbers. Iterative testing is white-box testing and checks to see if specific functions ,procedures and algorithms are working correctly.

Module	Feature	Input	Expected Output
2.2 - Menus	Program Initialisation	Program is clicked on	Program opens within 5 seconds
	“Start Game” button in Main Menu	User clicks on “Start Game” button in main menu	Scene state is changed to the main game and mainMenuOpen variable set to False within 5 seconds
	“Controls” button in Main Menu	User clicks on “Controls” button in main menu	Scene state is changed to the controls and mainMenuOpen variable set to False within 5 seconds
	“Exit Game” button in Main Menu	User clicks on “Exit Game” button in main menu	Game window should close within 5 seconds
	“Main Menu” button in controls menu	Left click on “Main Menu” button in controls menu	Scene state changed to main menu and mainMenuOpen set to True within 5 seconds
	“Unpause” button in Pause Menu	User clicks on “Unpause” button in pause menu	Scene state is changed to the main game and pauseMenuOpen variable set to False within 5 seconds
	“Controls” button in Pause Menu	User clicks on “Controls” button in pause menu	Scene state is changed to the controls and pauseMenuOpen set to False within 5 seconds

	“Main Menu” button in Pause Menu	User clicks on “Main Menu” button in pause menu	Scene state changed to main menu, pauseMenuOpen set to False and mainMenuOpen set to True within 5 seconds
	“Main Menu” button in simulation initialisation menu	Left click on “Main Menu” button in simulation initialisation menu	Scene state changed to main menu and mainMenuOpen set to True within 5 seconds
2.3 - Environment User Interface	Timer	User is in main game and unpaused	Time should be updated and displayed on the screen in the format of minutes and seconds every second
	Day Counter	User is in the main game and unpaused	Day number is the correct value of “timeTotal DIV 60” and should also be displayed on the screen and updated every second
	Organism Counter (Organism Born)	User is in the main game, unpaused and organism.born() method is initiated	Organism population is updated by an addition of 1 within 5 seconds
	Organism Counter (Organism Dead)	User is in the main game, unpaused and organism.dead() method is initiated	Organism population is updated by an subtraction of 1 within 5 seconds
	Organism Overhead Display	User is in the main game and unpaused	An overhead display containing an organisms name, health, thirst, hunger and stamina is display above each organism head and is updated at least every 3 seconds
2.4 - Free Cam View	Free Cam Movement	User is in free cam mode and unpaused	All controls and movement inputs provide appropriate outputs
2.6 - Organism Attributes	Hunger	An organism is alive and its hunger variable is <= 10	Organisms health value should decrease by 3 and stamina decrease by 3 every seconds until the organisms hunger is > 10 or organism dies
	Thirst	An organism is alive and its thirst variable is <= 10	Organisms health value should decrease by 3 and stamina decrease by 3 every seconds until the organisms thirst is > 10 or organism dies
	Stamina	An organism is alive and its stamina variable	Organisms movementState variable is set to “walking”

		is <= 0	
		An organism is alive and its stamina variable is >= 25	Organisms movementState variable is set to “running”
2.7 - Organism Behaviours	Wolf Behaviours	User in main game, unpaused and wolf alive	Wolves’ behaviouralState changes according to their attributes such as their hunger, thirst, stamina, etc.
	Deer Behaviours	User in main game, unpaused and deer alive	Deer’s behaviouralState changes according to their attributes such as their hunger, thirst, stamina, etc.
2.8 - Organism Genetics and Mutation	Attribute Multipliers	User in main game, unpaused and an organism is born	The base attribute max will be the average of the parents one. Then the multiplier to the base of each attribute is in the range of 0.9 - 1.1
	Attribute Mutations	User in main game, unpaused and an organism is born	The probability of an attribute mutation is 1% for each one and if it is fulfilled then that attribute is multiplied by 1.25
2.10 - Player Controlled Movement	Player Controller	User is in player controlled movement mode and unpaused	All controls and movement inputs provide appropriate outputs
2.11 - Skill Tree & Evolution	Unlocked Skill	User clicks on a skill to unlock it	Skill Unlocks
	Unlockable Skill	User unlocks a skill above another skill	Skill is able to be unlocked with required points
2.12 Game Over Screen	Population Graph	Births and deaths of organisms through the simulation	Displays an accurate graph with population as the y-axis and time as the x-axis
	Overall Statistics	Track the total amount of populations for organisms, deaths of organisms and the simulation time.	Displays these values within the overall statistics box.
	“Main Menu” Button in Game Over Screen	User clicks on “Main Menu” Button in game over screen	Scene state is changed to the main menu and mainMenuOpen variable set to True within 5 seconds

2.15.2 Post-Development Testing

Post-development testing will take place once an entire system is complete. This stage of testing does not involve testing individual algorithms,functions or procedures. Instead it focuses on if the solution works as intended and meets the requirements specifications. Stakeholders should be involved in post-development testing.

Test Scenario 1 : Free Cam Simulation

- Open the game
- Press start game
- Input starting conditions
    - Wolf starting population: 10
    - Deer starting population: 10
    - Plant starting population: 10
    - Time: 300 seconds
- Start the simulation
- Use free cam movement to move around
- Click on an organism to view their attributes
- Click on genetics to view that organism genetics
- Exit out of the genetics and attributes menu
- Pause the simulation
- Unpause the simulation
- Wait for simulation to end
- View the data in the game over menu
- Return to main menu
- Exit game

Test Scenario 2 : User Inputted Movement Simulation

- Open the game
- Press start game
- Input starting conditions
    - Wolf starting population: 5
    - Deer starting population: 15
    - Plant starting population: 20
    - Time: 240 seconds
- Start the simulation
- Click on a deer to view their attributes
- Click on control to move around as the organism
- Eat a plant as the deer
- Wait for simulation to end
- View the data in the game over menu
- Return to main menu
- Exit game

Chapter 3 - DEVELOPMENT

3.1 Development Plan

I'll be creating a development plan to organise the projects and make sure that specified deadlines are met so that the completion of the project can stay on track. I've split the project into key iterations with each iteration having a key part of the game completed in it. The deadlines for the development plan includes the testing for each iteration.

Project Iteration	Core Module	Sub Module	Dependency	Due by	Status
1	Menus	Main Menu	-	01.11.2024	Completed
		Controls Menu	Main Menu		Completed
		Pause Menu	Main Menu		Completed
		Initialisation Menu	Main Menu		Completed
2	Free Camera	Mouse Input	-	14.11.2024	Completed
		Keyboard Input	-		Completed
3	Organisms & User Interface	Spawns	Initialisation Menu	28.11.2024	Completed
		Vector Location	-		Completed
		Health	-		Completed
		Stamina	-		Completed
		Hunger	-		Completed
		Thirst	-		Completed
		Speed	-		Completed
		Timer & Day Counter	-		Completed
		Organism Counters	-		Completed
		Overhead Display	-		Completed

4	Player Controlled Movement	Forward	Organisms	12.01.2025	Completed
		Backwards	Organisms		Completed
		Left	Organisms		Completed
		Right	Organisms		Completed
		Mouse	Organisms		Completed
5	Organism Behaviours & Pathfinding	Hunting (Wolf only)	Organisms	26.01.2025	Completed
		Searching for Plant (Deer only)	Organism		Completed
		Eating	Organisms		Completed
		Searching for Water	Organisms		Completed
		Drinking	Organisms		Completed
		Searching for Mate	Organisms		Completed
		Mating	Organisms		Completed
		Sleeping	Organisms		Incomplete
		Idle	Organisms		Incomplete
6	Organism Genetics & Mutation	Genetics	Mating	10.02.2025	Completed
		Mutations	Mating		Completed
7	Skill Tree	Skill Tree Menu	-	24.02.2025	Incomplete
		Unlock Skills	Skill Tree Menu		Incomplete
8	End Screen	Graph	Organisms	01.03.2025	Completed
		Statistics Overview	Organisms		Completed

3.2 Menus - Iteration 1

3.2.1 Intro/Justification

There needs to be menus in the game so that the user can easily navigate throughout the game in a simple and easy to use way. I know users want this due to my User Questionnaire from 1.3.2, where 13 out of 13 people said that they wanted a pause/menu button. The menus I'll be adding are a main menu for when the game is open, a controls menu so the player can view the controls before they start to play the game, a pause menu for when the user is in the game and a simulation initialisation menu for when a player begins a simulation. This achieves Success Criteria no.1 and no.12 from 1.4.4.

{ How to get text from INPUT FIELD in Unity } - User Text Entries
https://discussions.unity.com/t/how-to-freeze-and-unfreeze-my-game/311091

3.2.2 Annotated Code

MainMenu.cs

```
using UnityEngine;

public class MainMenu : MonoBehaviour
{
    // Changes scene to Simulation Initiation Menu
    public void GameStart()
    {
        SceneManager.LoadScene("SimulationInitiation");
    }

    // Changes scene to Controls Menu
    public void Controls()
    {
        SceneManager.LoadScene("ControlsMenu");
    }

    // Changes scene to Main Menu
    public void ReturnToMain()
    {
        SceneManager.LoadScene("MainMenu");
    }

    // Changes scene to Pause Menu
    public void Pause()
    {
        SceneManager.LoadScene("PauseMenu");
    }

    // Changes scene to Environment
    public void Unpause()
    {
        SceneManager.LoadScene("Environment");
    }

    // Changes scene to Environment
    public void gameInitiationEnvironemntPlaceholder()
    {
        SceneManager.LoadScene("Environment");
    }

    // Closes the application
    public void Exit()
    {
        Application.Quit();
    }
}
```

InputHandler.cs

```
using UnityEngine;
using TMPro;
using UnityEditor.Overlays;

Unity Script (1 asset reference) | 0 references
public class InputHandler : MonoBehaviour
{
    // Store the inputed values as strings
    public static string DeerStartPopulation;
    public static string WolfStartPopulation;
    public static string PlantStartPopulation;
    public static string TimeStartLength;

    Unity Message | 0 references
    void Awake()
    {
        // Makes it so the object the script is assigned to isn't destroyed when loading a new scene
        DontDestroyOnLoad(gameObject);
    }

    0 references
    public void OnInputFinished(string value)
    {
        // Remove spaces from the input
        value = value.Trim();

        // Assigns input data to the corresponding variables
        string fieldName = UnityEngine.EventSystems.EventSystem.current.currentSelectedGameObject.name;

        switch (fieldName)
        {
            case "Field1": DeerStartPopulation = value; break;
            case "Field2": WolfStartPopulation = value; break;
            case "Field3": PlantStartPopulation = value; break;
            case "Field4": TimeStartLength = value; break;
        }

        Debug.Log($"{fieldName}: {value}");
    }
}
```

PauseMenuUpdater.cs

```
Unity Script | 1 reference
public class PauseMenuUpdater : MonoBehaviour
{
    public GameObject CanvasPauseMenu;
    public GameObject CanvasControlsMenu;
    public bool PauseMenuStatus;
    public static bool FreeCamControllerUpdater;


    // Start is called once before the first execution of Update after the MonoBehaviour is created
    Unity Message | 0 references
    void Start()
    {
        CanvasPauseMenu.SetActive(false);
        CanvasControlsMenu.SetActive(false);
        PauseMenuStatus = false;
        FreeCamControllerUpdater = false;
        Time.timeScale = 1;
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        // If escape key pressed then show pause menu
        if (Input.GetKey(KeyCode.Escape))
        {
            CanvasPauseMenu.SetActive(true);
            PauseMenuStatus = true;
            FreeCamControllerUpdater = true;
            Time.timeScale = 0;
        }
    }

    // Unpause game
    0 references
    public void Unpause()
    {
        CanvasPauseMenu.SetActive(false);
        CanvasControlsMenu.SetActive(false);
        PauseMenuStatus = false;
        FreeCamControllerUpdater = false;
        Time.timeScale = 1;
    }

    // Changes scene to environment
    0 references
    public void MainMenu()
    {
        SceneManager.LoadScene("MainMenu");
    }

    // Changes menu to pause menu
    0 references
    public void Controls()
    {
        CanvasPauseMenu.SetActive(false);
        PauseMenuStatus = false;
        CanvasControlsMenu.SetActive(true);
    }

    // Returns from controls to pause menu
    0 references
    public void Return()
    {
        CanvasPauseMenu.SetActive(true);
        CanvasControlsMenu.SetActive(false);
        PauseMenuStatus = false;
    }
}
```

3.2.3 Iterative Testing

1 - Main Menu

Test No.	Success Criteria	Test Aim	Input	Expected Output	Actual Output	Pass/Fail	Fixes
1.1	1	Program Initialization	Program is clicked on	Program opens within 5 seconds	No output	Fail	Check 3.2.5 Fixes Applied No.1
1.2	1	“Start Game” button in main menu	Left click on “Start Game” button in main menu	Takes you to Simulation Settings scene within 5 seconds	Takes you to Simulation Settings scene within 5 seconds	Pass	-
1.3	1	“Controls” button in main menu	Left click on “Controls” button in main menu	Takes you to Controls scene within 5 seconds	Takes you to Controls scene within 5 seconds	Pass	-
1.4	1	“Exit Game” button in main menu	Left click on “Exit Game” button in main menu	Closes the application within 5 seconds	Closes the application within 5 seconds	Pass	-

2 - Controls Menu

Test No.	Success Criteria	Test Aim	Input	Expected Output	Actual Output	Pass/Fail	Fixes
2.1	1	“Main Menu” button in controls menu	Left click on “Main Menu” button in controls menu	Takes you to Main Menu scene within 5 seconds	Takes you to Main Menu scene within 5 seconds	Pass	-

3 - Pause Menu

Test No.	Success Criteria	Test Aim	Input	Expected Output	Actual Output	Pass/Fail	Fixes
3.1	12	“Unpause” button in pause menu	Left click on “Unpause” button in pause menu	Takes you to environment scene within 5 seconds	Takes you to environment scene within 5 seconds	Pass	-
3.2	12	“Controls” button in pause menu	Left click on “Pause” button in pause menu	Takes you to Controls scene within 5 seconds	Takes you to Controls scene within 5 seconds	Pass	-
3.3	12	“Main Menu” button in pause menu	Left click on “Main Menu” button in pause menu	Takes you to Main Menu scene within 5 seconds	Takes you to Main Menu scene within 5 seconds	Pass	-

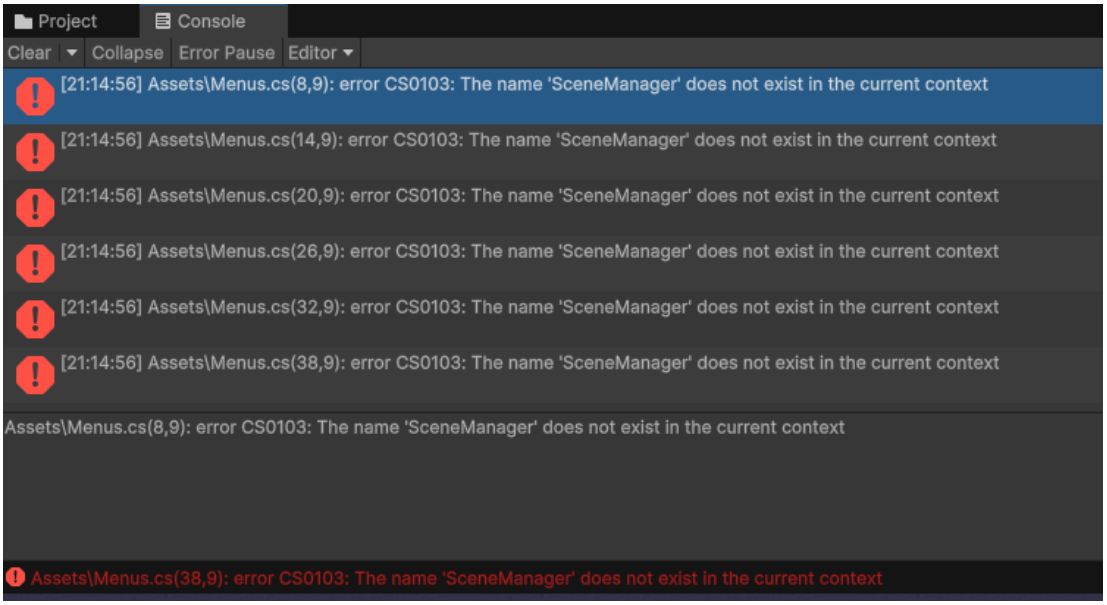
4 - Simulation Initialisation Menu

Test No.	Success Criteria	Test Aim	Input	Expected Output	Actual Output	Pass/Fail	Fixes
4.1	1	“Main Menu” button in simulation initialisation menu	Left click on “Main Menu” button in simulation initialisation menu	Takes you to Main Menu scene within 5 seconds	Takes you to Main Menu scene within 5 seconds	Pass	-
4.2	1	“Deer Population” input field in initialisation menu	Input data into “Deer Population” input field in initialisation menu	Data imputed is stored in “DeerPopulationInput” variable	Data inputted is stored in “DeerPopulationInput” variable	Pass	-
4.3	1	“Wolf Population” input field in initialisation menu	Input data into “Wolf Population” input field in initialisation menu	Data imputed is stored in “WolfPopulationInput” variable	Data inputted is stored in “WolfPopulationInput” variable	Pass	-
4.4	1	“Plant Population” input field in initialisation menu	Input data into “Plant Population” input field in initialisation menu	Data imputed is stored in “PlantPopulationInput” variable	Data inputted is stored in “PlantPopulationInput” variable	Pass	-
4.5	1	“Time Length” input field in initialisation menu	Input data into “Time Length” input field in initialisation menu	Data imputed is stored in “Time Length” variable	Data inputted is stored in “Time Length” variable	Pass	-

3.2.4 Fixes Applied

No.1 - Main Menu - Program Initialization

Evidence



Fix

Added “using UnityEngine.SceneManagement;” to line 2 to fix this error

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour
{
    // Changes scene to Simulation Initiation Menu
    public void GameStart()
    {
        SceneManager.LoadScene("SimulationInitiation");
    }

    // Changes scene to Controls Menu
    public void Controls()
    {
        SceneManager.LoadScene("ControlsMenu");
    }

    // Changes scene to Main Menu
    public void ReturnToMain()
    {
        SceneManager.LoadScene("MainMenu");
    }

    // Changes scene to Pause Menu
    public void Pause()
    {
        SceneManager.LoadScene("PauseMenu");
    }

    // Changes scene to Environment
    public void Unpause()
    {
        SceneManager.LoadScene("Environment");
    }

    // Changes scene to Environment
    public void gameInitiationEnvironemntPlaceholder()
    {
        SceneManager.LoadScene("Environment");
    }

    // Closes the application
    public void Exit()
    {
        Application.Quit();
    }
}
```

Retest

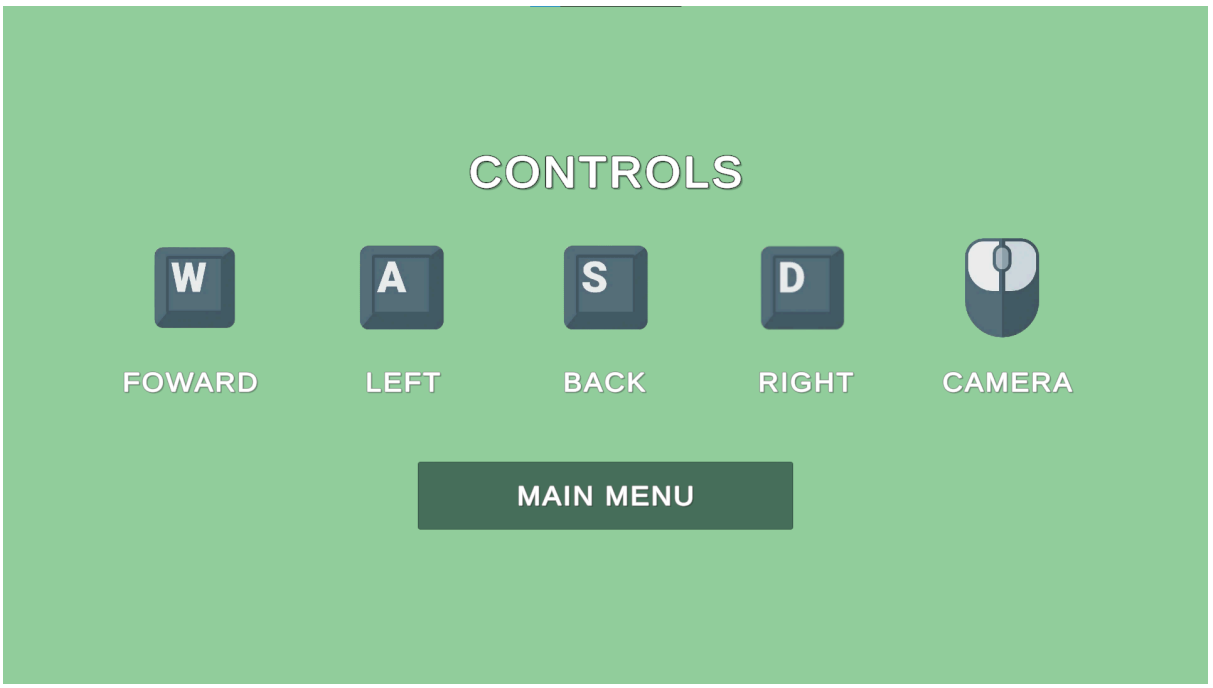
Test No.	Success Criteria	Test Aim	Input	Expected Output	Actual Output	Pass/Fail	Fixes
1.1	1	Program Initialization	Program is clicked on	Program opens within 5 seconds	Program opens within 5 seconds	Pass	-

3.2.5 Success from Testing

6177_menus_Test.mp4

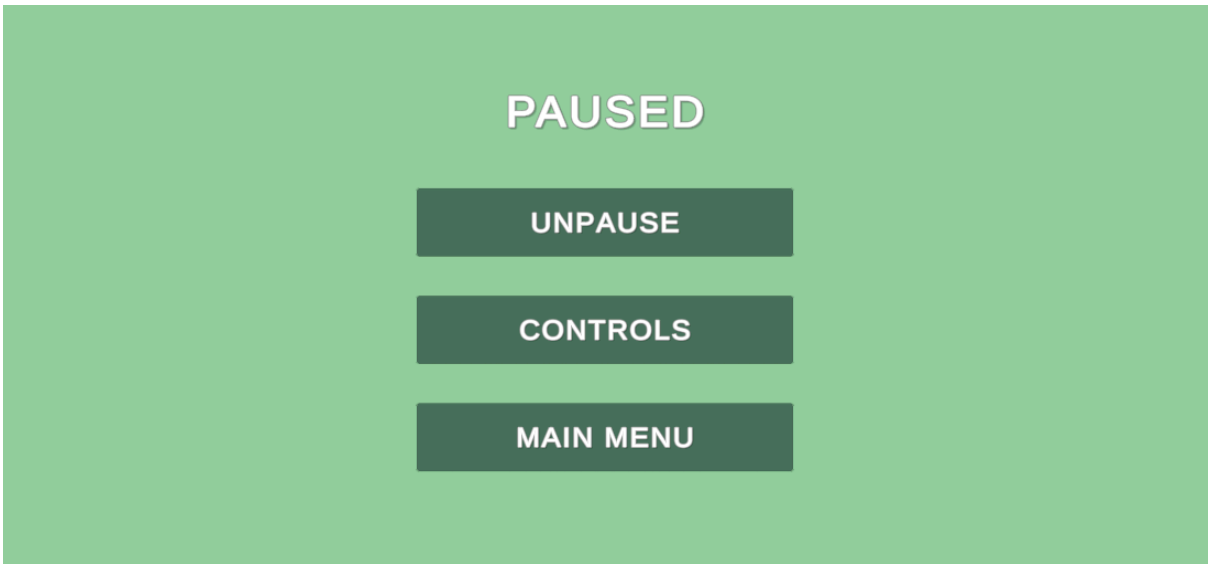
Main Menu

Controls Menu



Initiation Menu

Pause Menu



3.2.6 Review

The layout of the menus is very similar to the mockups in the design section. Some changes have been made such as the controls menu which was not originally made in the design section and adding a back button to the simulation initiation menu as there was not originally one. The client recommended changing the controls menu at a later stage in development to add more controls as they are developed; such as controls needed to interact with organisms and pressing escape to open the pause menu during the simulation. The client also recommends that at a later stage in development, when the environment and organism have been implemented, to change the background of the main menus from solid green to something similar to the original designs where there is an image or possibly a video of the environment being the background. Currently I am on target for my iteration deadline date leaving me with a comfortable amount of time to develop the future iterations.

Client Signature :

3.3 Free Camera - Iteration 2

3.3.1 Intro/Justification

A free cam controller needs to be added into the game so that the player is able to freely move around the environment and spectate the ecosystem simulation in real time. This is a feature from the VAV Ecosystem Simulation which was case study 2 in my research. Adding a free camera will fulfil success criteria no.2 from 1.4.4.

3.3.2 Annotated Code

How to make a spectator/free camera in Unity! Quick & Easy! - Modified this code; originally the code used the left click and mouse direction to move and left shift and left alt to change movement speeds. I changed this to have the controls be space and left control to move up and down the y-axis and keys A and D to move left and right on the x-axis. The keys W and S are used to move forward and backwards on the z-axis. Then the left shift activates sprint speed and normal speed if not pressing it.

FreeCamController.cs

```
Unity Script (1 asset reference) | 0 references
public class FreeCamController : MonoBehaviour
{
    public static Camera FreeCamera;
    public float sensitivity = 5;
    public float normalSpeed = 15;
    public float sprintSpeed = 25;
    float currentSpeed;

    Unity Message | 0 references
    void Start()
    {
        FreeCamera = GameObject.Find("FreeCamPlayer").GetComponent<Camera>();
    }

    Unity Message | 0 references
    void Update()
    {
        // Checks to see if any menus are open and if so suspends free camera movement and enables cursor
        if (DeerOOP.FreeCamControllerUpdater || WolfOOP.FreeCamControllerUpdater || PauseMenuUpdater.FreeCamControllerUpdater)
        {
            Cursor.visible = true;
            Cursor.lockState = CursorLockMode.Confined;
        }
        else
        {
            Cursor.visible = false;
            Cursor.lockState = CursorLockMode.Locked;
            Movement();
            Rotation();
        }
    }

    1 reference
    public void Rotation()
    {
        // Mouse movement input
        Vector3 mouseInput = new Vector3(-Input.GetAxis("Mouse Y"), Input.GetAxis("Mouse X"), 0);
        transform.Rotate(mouseInput * sensitivity);
        Vector3 eulerRotation = transform.rotation.eulerAngles;
        transform.rotation = Quaternion.Euler(eulerRotation.x, eulerRotation.y, 0);
    }
```

```
    1 reference
    public void Movement()
    {
        // Horizontal and vertical input
        Vector3 input = new Vector3(Input.GetAxis("Horizontal"), 0f, Input.GetAxis("Vertical"));

        // Adjust Y-axis movement
        if (Input.GetKey(KeyCode.Space))
        {
            input.y = 1f;
        }
        else if (Input.GetKey(KeyCode.LeftControl))
        {
            input.y = -1f;
        }

        // Adjust Z-axis movement
        if (Input.GetKey(KeyCode.W))
        {
            input.z = 1f;
        }
        else if (Input.GetKey(KeyCode.S))
        {
            input.z = -1f;
        }

        // Adjust X-axis movement
        if (Input.GetKey(KeyCode.A))
        {
            input.x = -1f;
        }
        else if (Input.GetKey(KeyCode.D))
        {
            input.x = 1f;
        }

        // Checks if speed is sprint or normal
        if (Input.GetKey(KeyCode.LeftShift))
        {
            currentSpeed = sprintSpeed;
        }
        else
        {
            currentSpeed = normalSpeed;
        }
        transform.Translate(input * currentSpeed * Time.deltaTime);
    }
}
```

3.3.3 Iterative Testing

1 - Mouse Input

Test No.	Success Criteria	Test Aim	Input	Expected Output	Actual Output	Pass/Fail	Fixes
1.1	2	Mouse movement input of free camera	Move mouse while in free camera mode	Free camera axis will rotate in response to the mouse inputs about a point without tilting/rolling	Program failed to compile	Fail	Check 3.3.4 Fixes Applied No.1

2 - Keyboard Input

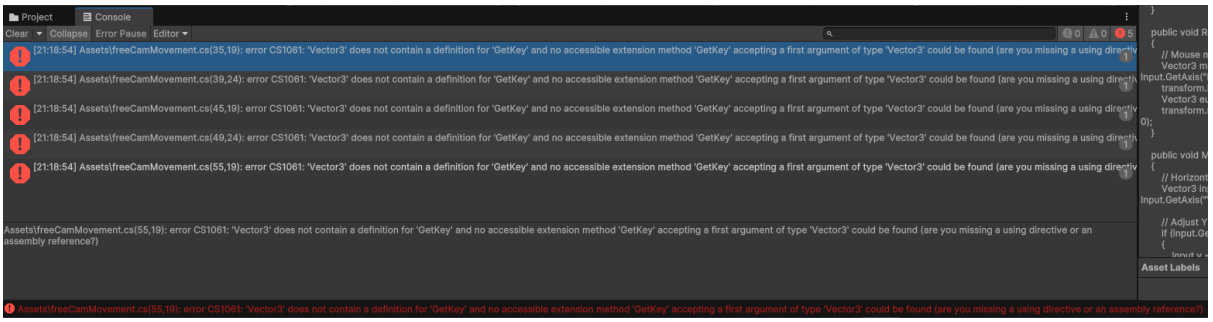
Test No.	Success Criteria	Test Aim	Input	Expected Output	Actual Output	Pass/Fail	Fixes
2.1	2	W key movement in freecam	Press W key while in free camera mode	Free camera position will move forward within 1 second	Free camera position moved forward within 1 second	Pass	-
2.2	2	A key movement in freecam	Press A key while in free camera mode	Free camera position will move to the left within 1 second	Free camera position moved to the left within 1 second	Pass	-
2.3	2	S key movement in freecam	Press S key while in free camera mode	Free camera position will move backwards within 1 second	Free camera position moved backwards within 1 second	Pass	-
2.4	2	D key movement in freecam	Press D key while in free camera mode	Free camera position will move to the left within 1 second	Free camera position moved to the left within 1 second	Pass	-
2.5	2	Space key movement in freecam	Press Space key while in free camera mode	Free camera y-position will increase within 1 second	Free camera y-position increased within 1 second	Pass	-
2.6	2	Left control key movement in freecam	Press Left control key while in free camera mode	Free camera y-position will decrease within 1 second	Free camera y-position decreased within 1 second	Pass	-
2.7	2	Shift key to activate sprint speed	Press shift key while in free camera mode while moving	Free camera speed will increase from the normal speed to the sprint speed	Free camera speed increased from the normal speed to the sprint speed	Pass	-

3.3.4 Fixes Applied

No.1 - Free Camera - Mouse movement input of free camera

Evidence

Program failed to compile



Fix

For statements which checked if a key was being pressed, the variable “input” was used instead of the function “Input”. So the fix was changing all the if statements to having the function “Input” with an uppercase to fix this.

```
// Adjust Y-axis movement
if (Input.GetKey(KeyCode.Space))
{
    input.y = 1f;
}
else if (Input.GetKey(KeyCode.LeftControl))
{
    input.y = -1f;
}

// Adjust Z-axis movement
if (Input.GetKey(KeyCode.W))
{
    input.z = 1f;
}
else if (Input.GetKey(KeyCode.S))
{
    input.z = -1f;
}

// Adjust X-axis movement
if (Input.GetKey(KeyCode.A))
{
    input.x = -1f;
}
else if (Input.GetKey(KeyCode.D))
{
    input.x = 1f;
}

// Checks if speed is sprint or normal
if (Input.GetKey(KeyCode.LeftShift))
{
    currentSpeed = sprintSpeed;
}
else
{
    currentSpeed = normalSpeed;
}
transform.Translate(input * currentSpeed * Time.deltaTime);
```

Retest

Test No.	Success Criteria	Test Aim	Input	Expected Output	Actual Output	Pass/Fail	Fixes
1.1	2	Mouse movement input of free camera	Move mouse while in free camera mode	Free camera axis will rotate in response to the mouse inputs about a point without tilting/rolling	Free camera axis rotates in response to the mouse inputs about a point without tilting/rolling	Pass	-

3.3.5 Success from Testing

6177_freeCamMovement_Test.mp4

3.3.6 Review

The client expressed satisfaction in the movement of the free camera. He said that it felt very smooth and comfortable to move around. He said that the ability to be able to change movement speeds between fast and normal was very useful for precision when flying around the environment. He also said that the ability to move the free camera up and down using the control and space keys was good since it is much easier than having to look down or look up and then moving forward to travel on the y-axis.

Client Signature :

3.4 Organism & User Interface - Iteration 3

3.4.1 Intro/Justification

In this iteration I will be setting up the organisms spawning into the environment on initialisation, using the inputs provided in the menu, along with their OOP and overhead displays. I will also be adding a timer and day counter that will be displayed in a UI along with the current organism population counters. For the spawns I'll use the data from the input fields in the initialisation menu. This data will be used to determine the starting spawn population of the deer, wolf and plants as well as the duration of the simulation. The organism OOP will include their attributes such as health, stamina, etc. along with functions for their different behaviours such as searching for water, searching for mates, etc. The spawn locations for each animal will be randomised and I will make it so animals won't spawn within a certain distance from each other. However, before I set up the organism OOP and spawning I'll have to create the environment that they'll spawn into so I'll do that first.

Note: In-between iterations 2 and 3 I kept on encountering errors opening Unity where it would crash on opening. Eventually I fixed this issue by removing Unity Plastic version control as I had unity plastic and github version control setup at the same time which I believed caused the issue. Removing Unity Plastic fixed the issue and I was able to open the project again without any problems however this caused a slight delay in my deadlines.

https://discussions.unity.com/t/how-to-enable-and-disable-a-canvas-window-by-scripting/117242/2
https://discussions.unity.com/t/how-to-enable-and-disable-components-through-a-script/254116/4

3.4.2 Annotated Code

OrganismInitialization.cs

```
using System;
using System.Linq;
using UnityEngine;
using UnityEngine.AI;

Unity Script (1 asset reference) | 4 references
public class OrganismInitilisation : MonoBehaviour
{
    // Prefabs for each organism
    public GameObject DeerPrefab;
    public GameObject WolfPrefab;
    public GameObject PlantPrefab;

    // Starting populations and time
    public static int DeerStartingPopulation = Int32.Parse(InputHandler.DeerStartPopulation);
    public static int WolfStartingPopulation = Int32.Parse(InputHandler.WolfStartPopulation);
    public static int PlantStartingPopulation = Int32.Parse(InputHandler.PlantStartPopulation);
    public static float TimeStartingLength = Int32.Parse(InputHandler.TimeStartLength);

    public Collider platformCollider;
    private float timeRemaining;
    private bool isSimulationRunning = true;

    // Randomly spawning new plants
    private float spawnTimer = 0f;
    private float spawnInterval = 5;

    Unity Message | 0 references
    void Start()
    {
        // Find the platform object and its collider
        GameObject platform = GameObject.Find("Platform");
        if (platform == null)
        {
            Debug.LogError("Platform object not found");
            return;
        }

        platformCollider = platform.GetComponent<Collider>();
        if (platformCollider == null)
        {
            Debug.LogError("Platform Collider not found");
            return;
        }

        // Initialize the simulation time
        timeRemaining = TimeStartingLength;

        // Spawn the initial organisms
        SpawnObjects(DeerPrefab, DeerStartingPopulation);
        SpawnObjects(WolfPrefab, WolfStartingPopulation);
        SpawnObjects(PlantPrefab, PlantStartingPopulation);
    }
```

```
Unity Message | 0 references
void Update()
{
    if (isSimulationRunning)
    {
        // Decrease time remaining by the time passed since the last frame
        timeRemaining -= Time.deltaTime;

        // Check if the simulation has ended
        if (timeRemaining <= 0)
        {
            timeRemaining = 0;
            isSimulationRunning = false;
            Debug.Log("Simulation Ended");
        }

        // Random chance to spawn a plant every 1 second
        spawnTimer += Time.deltaTime;

        if (spawnTimer >= spawnInterval)
        {
            int randomNum = UnityEngine.Random.Range(0, 5);
            if (randomNum == 0)
            {
                SpawnObjects(PlantPrefab, 1);
                spawnTimer = 0;
            }
        }

    }
}

// Method that spawns in the organisms
4 references
void SpawnObjects(GameObject objectPrefab, int count)
{
    for (int i = 0; i < count; i++)
    {
        Vector3 spawnPosition = GetRandomPositionOnPlatform();
        // Check if position is valid
        if (spawnPosition != Vector3.zero)
        {
            Instantiate(objectPrefab, spawnPosition, Quaternion.identity);
        }
    }
}
```

```
// Function that gets a random position on the platform
1 reference
Vector3 GetRandomPositionOnPlatform()
{
    if (platformCollider == null)
    {
        Debug.LogError("Platform Collider is null");
        return Vector3.zero;
    }

    // Get the platform object's bounds
    Vector3 platformCenter = platformCollider.bounds.center;
    Vector3 platformSize = platformCollider.bounds.size;

    // Generate random positions within the platform's bounds
    float x = UnityEngine.Random.Range(platformCenter.x - platformSize.x / 2, platformCenter.x + platformSize.x / 2);
    float z = UnityEngine.Random.Range(platformCenter.z - platformSize.z / 2, platformCenter.z + platformSize.z / 2);
    float y = 0;

    return new Vector3(x, y, z);
}
}
```

OrganismOOP.cs

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

Unity Script | 2 references
public class OrganismOOP : MonoBehaviour
{
    // Attributes
    public Vector3 position;
    public string organismName;
    public bool isAlive;
    public int health;
    public int healthMax;
    public int hunger;
    public int hungerMax;
    public int thirst;
    public int thirstMax;
    public int stamina;
    public int staminaMax;
    public string movementState;
    public string behaviouralState;
    public string[] skills;
    public float radius;

    // Methods
    4 references
    public virtual void Move()
    {
        // Moving
    }

    // Healing method
    0 references
    public void Heal()
    {
        Debug.Log(organismName + " healed to " + health);
    }

    // Searching for water method
    0 references
    public void SearchWater()
    {
        Debug.Log(organismName + " is searching for water.");
    }
```

```
    // Eating method
    0 references
    public void Eat()
    {
        Debug.Log(organismName + " is eating. Hunger level: " + hunger);
    }

    // Drinking method
    0 references
    public void Drink()
    {
        Debug.Log(organismName + " is drinking. Thirst level: " + thirst);
    }

    // Mating method
    0 references
    public void Mate()
    {
        Debug.Log(organismName + " is mating.");
    }

    // Born method
    0 references
    public void Born()
    {
        Debug.Log(organismName + " was born.");
    }

    // Dead method
    0 references
    public void Dead()
    {
        Debug.Log(organismName + " died.");
    }
}
```

DeerOOP.cs

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

Unity Script (1 asset reference) | 12 references
public class DeerOOP : OrganismOOP
{
    public GameObject DeerCamera;
    public static bool CanvasOrganismDataUI = false;
    public static bool FreeCamControllerUpdater = false;
    private float decreaseTimer = 0f;
    private float decreaseInterval = 1f;
    private float logTimer = 0f;
    private float logInterval = 5f; // Set log frequency to 5 seconds

    public static string selectedDeerName;
    public static float selectedDeerHealth;
    public static float selectedDeerHunger;
    public static float selectedDeerThirst;
    public static float selectedDeerStamina;
    public static string selectedDeerMovementState;
    public static string selectedDeerBehavioralState;

    Unity Message | 0 references
    void Start()
    {
        // Declare variables for the organism
        DeerCamera = GameObject.Find("Camera");
        isAlive = true;
        organismName = "Deer";
        health = 100;
        healthMax = 100;
        hunger = 100;
        hungerMax = 100;
        thirst = 100;
        thirstMax = 100;
        stamina = 100;
        staminaMax = 100;
        movementState = "walking";
        behaviouralState = "idle";
        position = transform.position;
        radius = 5.0f;

        // Set the GameObject's name to the organisms name
        gameObject.name = organismName + "_" + gameObject.GetInstanceID();
        organismName = gameObject.name;

        Debug.Log(organismName + " initialized.");
    }
```

```
Unity Message | 0 references
void Update()
{
    // Checks if organism is alive
    if (isAlive)
    {
        // If organism is not alive then decrease population by 1 and set isAlive to false
        if (health <= 0)
        {
            health = 0;
            EnvironmentData.DeerPopulation -= 1;
            isAlive = false;
        }
        else if (hunger <= 0)
        {
            hunger = 0;
        }
        else if (thirst <= 0)
        {
            thirst = 0;
        }
        else if (stamina <= 0)
        {
            stamina = 0;
        }

        Move();

        // Decrease time by the frames that have passed
        decreaseTimer += Time.deltaTime;
        logTimer += Time.deltaTime;

        if (decreaseTimer >= decreaseInterval)
        {
            if (behaviouralState == "idle")
            {
                hunger -= 1;
                thirst -= 1;
            }

            if (hunger <= 10 || thirst <= 10)
            {
                health -= 3;
                stamina -= 3;
            }

            decreaseTimer = 0f;
        }

        // Log stats at the specified interval
        if (logTimer >= logInterval)
        {
            Debug.Log($"Health: {health}, Stamina: {stamina}, Hunger: {hunger}, Thirst: {thirst}");
            logTimer = 0f;
        }

        if (stamina <= 0)
        {
            movementState = "walking";
        }
        else if (stamina >= 25)
        {
            movementState = "running";
        }
    }
}
```

```
    // Plant search method
    0 references
    public void PlantSearch()
    {
        Debug.Log(organismName + " is searching for plants.");
    }

    // Moving method overrided from parent class
    2 references
    public override void Move()
    {
        // moving
    }

    // Method to detect when FreeCamPlayer left clicks on an organism when in range
    Unity Message | 0 references
    public void OnMouseOver()
    {
        GameObject freecamobject = GameObject.Find("FreeCamPlayer");
        float distance = Vector3.Distance(gameObject.transform.position, freecamobject.transform.position);

        if (distance <= 10 && Input.GetMouseButtonDown(0))
        {
            // Set static variables with current deer's info
            selectedDeerName = organismName;
            selectedDeerHealth = health;
            selectedDeerHunger = hunger;
            selectedDeerThirst = thirst;
            selectedDeerStamina = stamina;
            selectedDeerMovementState = movementState;
            selectedDeerBehavioralState = behaviouralState;

            // Set the CanvasOrganismDataUI to true
            CanvasOrganismDataUI = true;
            FreeCamControllerUpdater = true;
        }
    }
}
```

WolfOOP.cs

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

Unity Script (1 asset reference) | 12 references
public class WolfOOP : OrganismOOP
{
    public GameObject WolfCamera;
    public static bool CanvasOrganismDataUI = false;
    public static bool FreeCamControllerUpdater = false;
    private float decreaseTimer = 0f;
    private float decreaseInterval = 1f;
    private float logTimer = 0f;
    private float logInterval = 5f; // Set log frequency to 5 seconds

    public static string selectedWolfName;
    public static int selectedWolfHealth;
    public static int selectedWolfHunger;
    public static int selectedWolfThirst;
    public static int selectedWolfStamina;
    public static string selectedWolfMovementState;
    public static string selectedWolfBehavioralState;

    Unity Message | 0 references
    void Start()
    {
        // Declare variables for the organism
        WolfCamera = GameObject.Find("Camera");
        isAlive = true;
        organismName = "Wolf";
        health = 100;
        healthMax = 100;
        hunger = 100;
        hungerMax = 100;
        thirst = 100;
        thirstMax = 100;
        stamina = 100;
        staminaMax = 100;
        movementState = "walking";
        behaviouralState = "idle";
        position = transform.position;
        radius = 5.0f;

        // Set the GameObject's name to the organisms name
        gameObject.name = organismName + "_" + gameObject.GetInstanceID();
        organismName = gameObject.name;

        Debug.Log(organismName + " initialized.");
    }
```

```
Unity Message | 0 references
void Update()
{
    // Checks if organism is alive
    if (isAlive)
    {
        // If organism is not alive then decrease population by 1 and set isAlive to false
        if (health <= 0)
        {
            health = 0;
            EnvironmentData.WolfPopulation -= 1;
            isAlive = false;
        }
        else if (hunger <= 0)
        {
            hunger = 0;
        }
        else if (thirst <= 0)
        {
            thirst = 0;
        }
        else if (stamina <= 0)
        {
            stamina = 0;
        }

        Move();

        // Decrease time by the frames that have passed
        decreaseTimer += Time.deltaTime;
        logTimer += Time.deltaTime;

        // Updates variables every second
        if (decreaseTimer >= decreaseInterval)
        {
            if (behaviouralState == "idle")
            {
                hunger -= 1;
                thirst -= 1;
            }

            if (hunger <= 10 || thirst <= 10)
            {
                health -= 3;
                stamina -= 3;
            }

            decreaseTimer = 0f;
        }

        // Log stats at the specified interval (5 seconds)
        if (logTimer >= logInterval)
        {
            Debug.Log($"Health: {health}, Stamina: {stamina}, Hunger: {hunger}, Thirst: {thirst}");
            logTimer = 0f;
        }

        if (stamina <= 0)
        {
            movementState = "walking";
        }
        else if (stamina >= 25)
        {
            movementState = "running";
        }
    }
}
```

```
    // Hunting method
    0 references
    public void Hunt()
    {
        Debug.Log(organismName + " is hunting.");
    }

    // Moving method overrided from parent class
    2 references
    public override void Move()
    {
        // moving
    }

    // Method to detect when FreeCamPlayer left clicks on an orgnism when in range
    Unity Message | 0 references
    public void OnMouseOver()
    {
        GameObject freecamobject = GameObject.Find("FreeCamPlayer");
        float distance = Vector3.Distance(gameObject.transform.position, freecamobject.transform.position);

        if (distance <= 10 && Input.GetMouseButtonDown(0))
        {
            // Set static variables with current wolf's info
            selectedWolfName = organismName;
            selectedWolfHealth = health;
            selectedWolfHunger = hunger;
            selectedWolfThirst = thirst;
            selectedWolfStamina = stamina;
            selectedWolfMovementState = movementState;
            selectedWolfBehavioralState = behaviouralState;

            // Set the CanvasOrganismDataUI to true
            CanvasOrganismDataUI = true;
            FreeCamControllerUpdater = true;
        }
    }
}
```

CanvasSimulationDataUpdater.cs

```
using UnityEngine;
using UnityEngine.UI;

public class CanvasSimulationDataUpdater : MonoBehaviour
{
    // Text components for each variable
    public Text DeerPopulationText;
    public Text WolfPopulationText;
    public Text PlantPopulationText;
    public Text TimeCounterText;

    // Update is called once per frame
    void Update()
    {
        // Update the Text UI to show the current population and time remaining, obtaining data from EnvironmentData script
        DeerPopulationText.text = "Deer - " + EnvironmentData.DeerPopulation.ToString();
        WolfPopulationText.text = "Wolf - " + EnvironmentData.WolfPopulation.ToString();
        PlantPopulationText.text = "Plant - " + EnvironmentData.PlantPopulation.ToString();
        TimeCounterText.text = "Time - " + EnvironmentData.TimeRemaining.ToString("F0") + "s"; // Format float to 2 decimal places
    }
}
```

EnvironmentData.cs

```
using UnityEngine;
using UnityEngine.UI;

Unity Script (1 asset reference) | 6 references
public class EnvironmentData : MonoBehaviour
{
    // Get starting populatations and time from OrganismInitilisation variables
    public static int DeerPopulation = OrganismInitilisation.DeerStartingPopulation;
    public static int WolfPopulation = OrganismInitilisation.WolfStartingPopulation;
    public static int PlantPopulation = OrganismInitilisation.PlantStartingPopulation;
    public static float TimeRemaining = OrganismInitilisation.TimeStartingLength;
    private bool isSimulationRunning = true;


    // Start is called once before the first execution of Update after the MonoBehaviour is created
    Unity Message | 0 references
    void Start()
    {
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        if (isSimulationRunning)
        {
            // Decrease time remaining by the time passed since the last frame
            TimeRemaining -= Time.deltaTime;
            Debug.Log($"Time remaining = {TimeRemaining}");

            // Check if the simulation has ended
            if (TimeRemaining <= 0)
            {
                TimeRemaining = 0;
                isSimulationRunning = false;
                Debug.Log("Simulation Ended");
            }
        }
    }


}
```

CanvasReference.cs

```
using UnityEngine;
using UnityEngine.UI;

Unity Script (1 asset reference) | 0 references
public class reference : MonoBehaviour
{
    public GameObject CanvasOrganismData;

    // References to the UI Text elements
    public Text organismNameText;
    public Text organismHealthText;
    public Text organismHungerText;
    public Text organismThirstText;
    public Text organismStaminaText;
    public Text organismMovementStateText;
    public Text organismBehavioralStateText;

    Unity Message | 0 references
    void Update()
    {
        if (DeerOOP.CanvasOrganismDataUI || WolfOOP.CanvasOrganismDataUI)
        {
            // Activate the canvas UI
            CanvasOrganismData.SetActive(true);

            // Update the UI with the selected organism's data
            UpdateUI();
        }
        else
        {
            // Deactivate canvas if not true
            CanvasOrganismData.SetActive(false);
        }
    }

    // Method to update the UI values once when an organism is selected
    1 reference
    void UpdateUI()
    {
        // If a Deer is selected, update the UI for the deer
        if (DeerOOP.CanvasOrganismDataUI)
        {
            organismNameText.text = DeerOOP.selectedDeerName;
            organismHealthText.text = DeerOOP.selectedDeerHealth.ToString();
            organismHungerText.text = DeerOOP.selectedDeerHunger.ToString();
            organismThirstText.text = DeerOOP.selectedDeerThirst.ToString();
            organismStaminaText.text = DeerOOP.selectedDeerStamina.ToString();
            organismMovementStateText.text = DeerOOP.selectedDeerMovementState;
            organismBehavioralStateText.text = DeerOOP.selectedDeerBehavioralState;
        }
        // If a Wolf is selected, update the UI for the wolf
        else if (WolfOOP.CanvasOrganismDataUI)
        {
            organismNameText.text = WolfOOP.selectedWolfName;
            organismHealthText.text = WolfOOP.selectedWolfHealth.ToString();
            organismHungerText.text = WolfOOP.selectedWolfHunger.ToString();
            organismThirstText.text = WolfOOP.selectedWolfThirst.ToString();
            organismStaminaText.text = WolfOOP.selectedWolfStamina.ToString();
            organismMovementStateText.text = WolfOOP.selectedWolfMovementState;
            organismBehavioralStateText.text = WolfOOP.selectedWolfBehavioralState;
        }
    }

    // Exit method to close the UI and reset cursor
    0 references
    public void Exit()
    {
        DeerOOP.CanvasOrganismDataUI = false;
        DeerOOP.FreeCamControllerUpdater = false;
        WolfOOP.CanvasOrganismDataUI = false;
        WolfOOP.FreeCamControllerUpdater = false;
        CanvasOrganismData.SetActive(false);
        Debug.Log("Organism Data Exitted");
    }

}
```

3.4.3 Iterative Testing

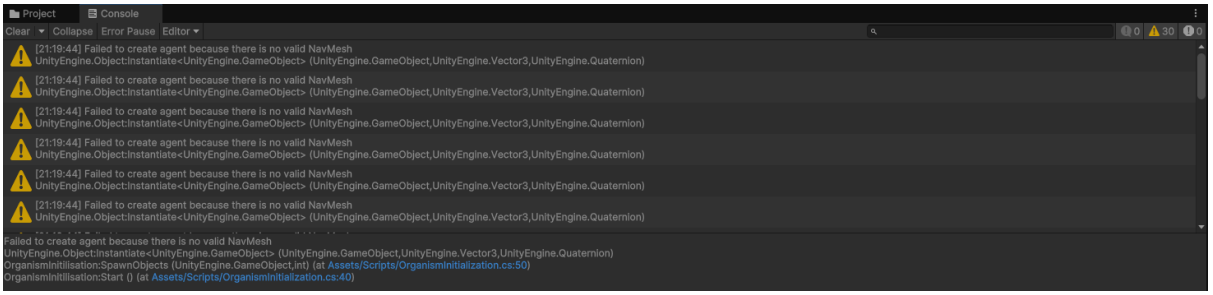
Test No.	Success Criteria	Test Aim	Input	Expected Output	Actual Output	Pass/Fail	Fixes
1.1	3	Organism Spawns	Input values into input fields in the initialisation menu then press the start button	Populations of deer, wolves and plants are set accordingly and are spawned into the environment	Organism all spawned however there was an error message produced relating to the NavMeshes	Fail	Check 3.4.4 Fixes Applied No.1
1.2	6	Organism Health	Track heath values of an organism for 1 minute and check to see if it changes accordingly to conditions in the code	Health attribute value should be accurately represented based on the conditions such as hunger and thirst	Health attribute value is accurately represented based on conditions and updates accordingly	Pass	-
1.3	6	Organism Stamina	Track stamina values of an organism for 1 minute and check to see if it changes accordingly to conditions in the code	Stamina attribute value should be accurately represented based on the condition of the movement state	Stamina attribute value is accurately represented based on the movement state variable	Pass	-
1.4	6	Organism Hunger	Track hunger values of an organism for 1 minute and check to see if it changes accordingly to conditions in the code	Hunger value should naturally decrease over time when the organism is not eating. If it is eating then hunger should increase	Hunger value naturally decrease over time if organism is not eating and increases when the organism is eating	Pass	-
1.5	6	Organism Thirst	Track thirst values of an organism for 1 minute and check to see if it changes accordingly to conditions in the code	Thirst value should naturally decrease over time when the organism is not drinking. If it is drinking then hunger should increase	Thirst value naturally decreases over time when organism is not drinking and increases when organism is drinking	Pass	-
1.6	6	Organism Speed	Track speed of an organism for 1 minute to see if its speed changes accordingly to conditions in the code	If the stamina of an organism is <=0 then the movement state should be set to walking, if the stamina of an organism is >=25 then the movement state should be set to running	Stamina of an organism when <=0 set the movement state to walking, and when the stamina was >=25 the movement state was set to running	Pass	-

1.7	-	Time & Day Counter	Simulation Length Inputted in Initiation Menu	Time and day should change accordingly and be updated every second	Timer decreased and was updated every second	Pass	Check 3.4.4 Fixes Applied No.2
1.8	-	Organism Counters	Input organism population values in the initiation menu	Organism counter should be representative of the population values imputed	The organism population counter value is representative of the inputted value and updates accordingly to death and births of new organisms	Pass	-
1.9	-	Organism Display	Left click while aiming at an organism	Organism display should appear with appropriate information	When an organism is left clicked in free cam mode all their stats show up	Pass	-

3.4.4 Fixes Applied

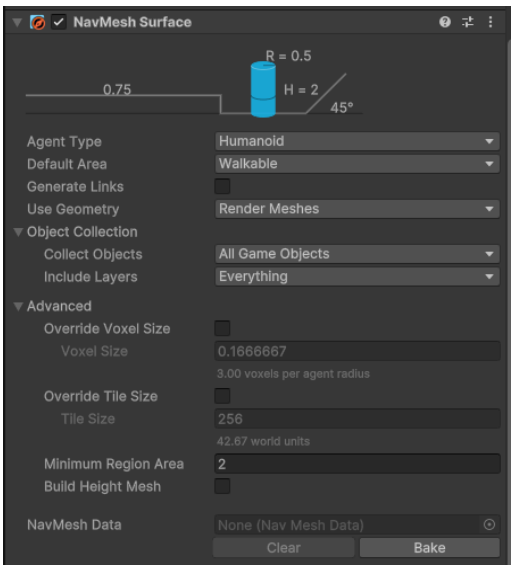
<u>No.1 - Organism & User Interface - Organism Spawns</u>

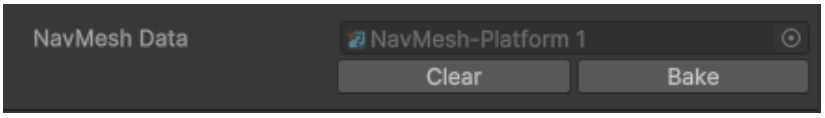
Evidence



Fix

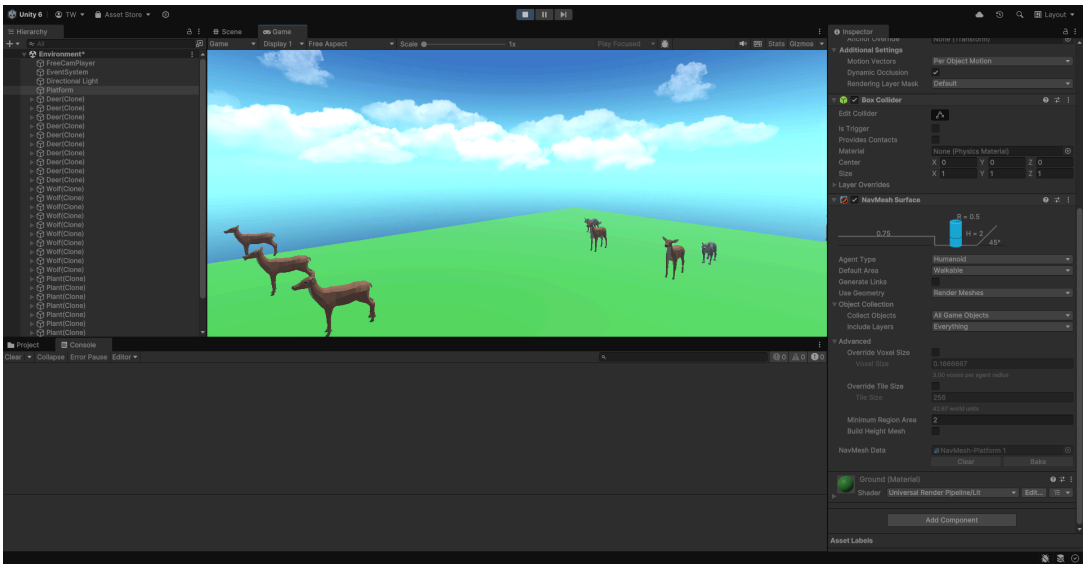
Error produced due to not baking the platform object so I baked the NavMesh Surface of the platform





Retest

Test No.	Success Criteria	Test Aim	Input	Expected Output	Actual Output	Pass/Fail	Fixes
1.1	3	Organism Spawns	Input values into input fields in the initialisation menu then press the start button	Populations of deer, wolves and plants are set accordingly and are spawned into the environment	Organisms all spawned with correct populations into the environment with no issues	Pass	-



No errors produced after baking platform object NavMesh Surface

No.2 - Organism & User Interface - Time & Day Counter

Feature fully worked however the debug logs were flooding the console.

Evidence

Fix

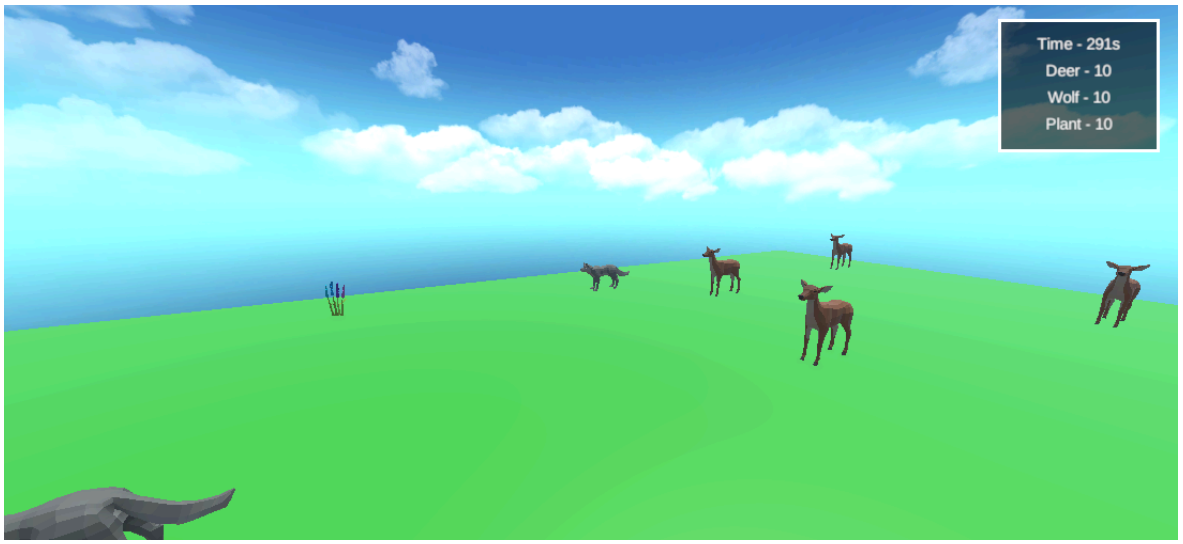
Removed debug log from Update method so it is no longer cluttered the console.

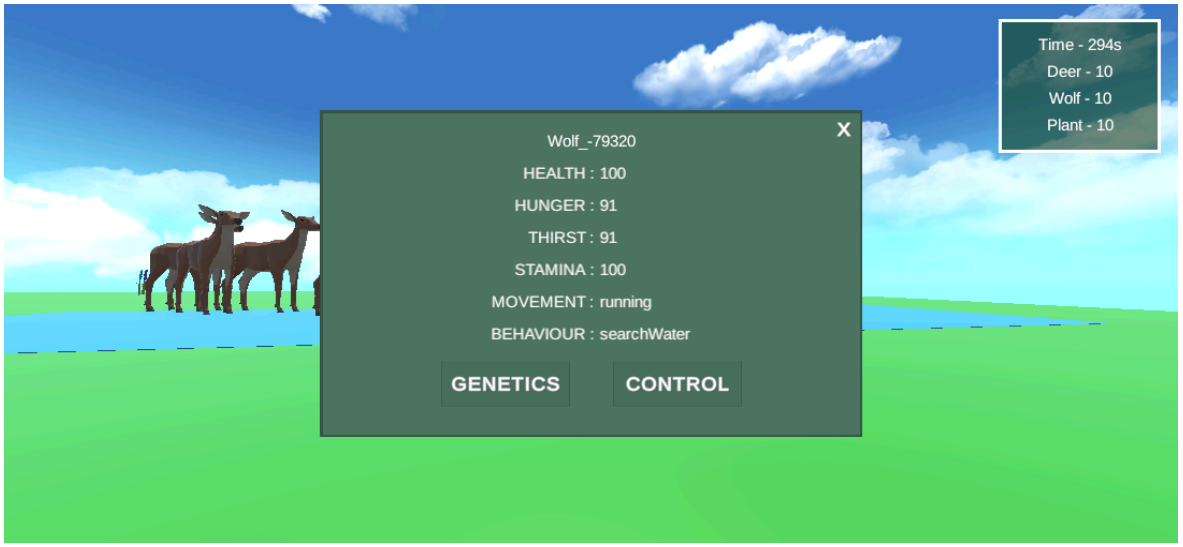
```
// Update is called once per frame
void Update()
{
    if (isSimulationRunning)
    {
        // Decrease time remaining by the time passed since the last frame
        TimeRemaining -= Time.deltaTime;

        // Check if the simulation has ended
        if (TimeRemaining <= 0)
        {
            TimeRemaining = 0;
            isSimulationRunning = false;
            Debug.Log("Simulation Ended");
        }
    }
}
```

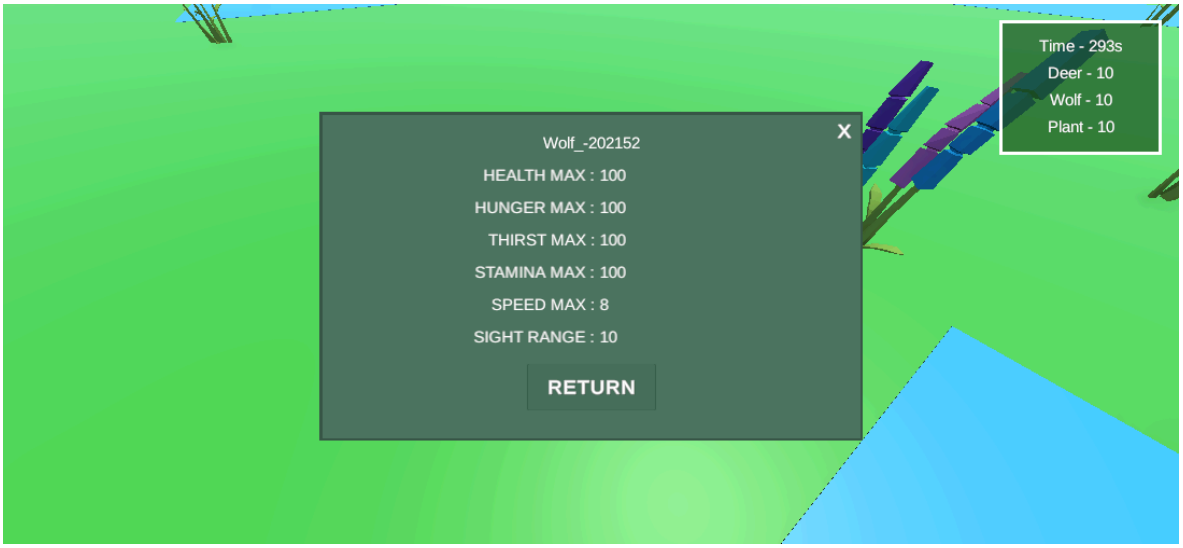
Retest

Test No.	Success Criteria	Test Aim	Input	Expected Output	Actual Output	Pass/Fail	Fixes
1.7	-	Time & Day Counter	Simulation Length Inputted in Initiation Menu	Time and day should change accordingly and be updated every second	Timer decreased and was updated every second	Pass	-

3.4.5 Success from Testing







6177_organismInitiation_Test.mp4
6177_userInterface_Test.mp4

3.4.6 Review

The client was happy with all the features added into this iteration including the organism initiation and user interfaces. However, he said that the user interface design could be updated to look more polished and professional by potentially changing the font or colour scheme at the end of the project if there is time remaining. He also said to potentially set up the lighting to correspond to the time remaining so that it is light or dark at certain times.

Client Signature :

3.5 Player Controlled Movement - Iteration 4

3.5.1 Intro/Justification

In this iteration I'll be setting up the player controlled movement of the organisms. The players will be able to select any organism they wish (except plants since they can't move) and then control their movement and behaviour. When they are in the player controlled movement state they are going to be able to control all the behaviours such as hunting another organism, eating a plant, drinking water or sleeping. This will obviously have an impact on the results of the simulation since instead of the movement of a specific organism being automated it will be user controlled. Player controlled movement is only going to be added for the wolves and deer; not the plants since they cannot move. I've also decided to remove the organism overhead display that is constantly shown above the organisms as it clutters the screen and all this information is already going to be displayed when clicking on an organism.

https://discussions.unity.com/t/distance-between-two-objects/404478/2

3.5.2 Annotated Code

PlayerControlledMovement.cs - Reused some code from FreeCamController.cs for this script since they have similar functionality

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

public class PlayerControlledMovement : MonoBehaviour
{
    public float sensitivity;
    public float normalSpeed;
    public float sprintSpeed;
    float currentSpeed;

    void Update()
    {
        Cursor.visible = false;
        Cursor.lockState = CursorLockMode.Locked;
        Movement();
        Rotation();
    }

    public void Rotation()
    {
        // Mouse movement input
        Vector3 mouseInput = new Vector3(-Input.GetAxis("Mouse Y"), Input.GetAxis("Mouse X"), 0);
        transform.Rotate(mouseInput * sensitivity);
        Vector3 eulerRotation = transform.rotation.eulerAngles;
        transform.rotation = Quaternion.Euler(eulerRotation.x, eulerRotation.y, 0);
    }

    public void Movement()
    {
        // Horizontal and vertical input
        Vector3 input = new Vector3(Input.GetAxis("Horizontal"), 0f, Input.GetAxis("Vertical"));

        // Adjust Z-axis movement
        if (Input.GetKey(KeyCode.W))
        {
            input.z = 1f;
        }
        else if (Input.GetKey(KeyCode.S))
        {
            input.z = -1f;
        }

        // Adjust X-axis movement
        if (Input.GetKey(KeyCode.A))
        {
            input.x = -1f;
        }
        else if (Input.GetKey(KeyCode.D))
        {
            input.x = 1f;
        }

        // Checks if speed is sprint or normal
        if (Input.GetKey(KeyCode.LeftShift)) // and stamina > ...
        {
            currentSpeed = sprintSpeed;
        }
        else
        {
            currentSpeed = normalSpeed;
        }
        transform.Translate(input * currentSpeed * Time.deltaTime);
    }
}
```

3.5.3 Iterative Testing

1 - Mouse Input

Test No.	Success Criteria	Test Aim	Input	Expected Output	Actual Output	Pass/Fail	Fixes
1.1	4	Mouse movement input to look around	Move mouse while in player controlled mode	Camera is fixed to the organisms model and moves accordingly to the mouse inputs	No inputs were working at beginning of simulation start due to multiple cameras	Fail	Check 3.5.4 Fixes Applied No.1

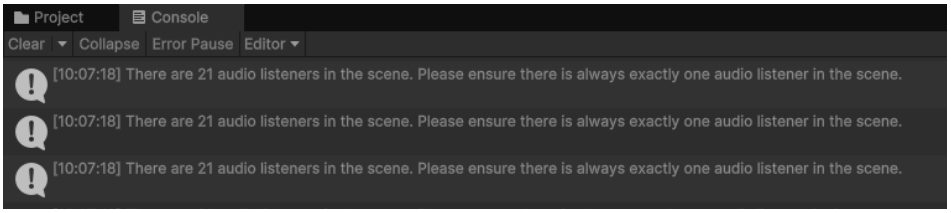
2 - Keyboard Input

Test No.	Success Criteria	Test Aim	Input	Expected Output	Actual Output	Pass/Fail	Fixes
2.1	4	W key movement in player controlled mode	Press W key while in player controlled mode	Organism should move forward and the camera is fixed to its position	Camera was not switching to organism camera and remained as free camera	Pass	-
2.2	4	A key movement in player controlled mode	Press A key while in player controlled mode	Organism should move to the left and the camera is fixed to its position	Organism moves to the left and the camera is fixed to its position	Pass	-
2.3	4	S key movement in player controlled mode	Press S key while in player controlled mode	Organism should move backwards and the camera is fixed to its position	Organism moves backwards and the camera is fixed to its position	Pass	-
2.4	4	D key movement in player controlled mode	Press D key while in player controlled mode	Organism should move right and the camera is fixed to its position	Organism moves right and the camera is fixed to its position	Pass	-
2.5	4	Shift key to activate sprint speed while in player controlled mode	Press shift key while in player controlled mode	Organism movement speed should turn to sprinting	Organism movement speed turns to sprinting when the shift key is pressed	Pass	-

3.5.4 Fixes Applied

No.1 - Player Controlled Movement - Mouse Input

Evidence



Fix

Disabled camera and audio listeners for organisms when they are initiated so at the beginning of the simulation the error does not occur.

Retest

Test No.	Success Criteria	Test Aim	Input	Expected Output	Actual Output	Pass/Fail	Fixes
1.1	4	Mouse movement input to look around	Move mouse while in player controlled mode	Camera is fixed to the organisms model and moves accordingly to the mouse inputs	All input worked and the model moved according while fixed to the model camera	Pass	-

3.5.5 Success from Testing

6177_playerControlledMovement_Test.mp4

3.5.6 Review

The client was happy with all the features added in this iteration which included player controlled movement where the user is able to select any organism they want and then be able to control that organism's movements. However, since I have decided to develop the player controlled movement before implementing the functionality for the organism behaviours I will have to add the ability to control organism behaviours while in the player controlled movement state after developing the next iteration. In hindsight I should have developed the organism behaviours first and then the player controlled movement afterwards.

Client Signature :

3.6 Organism Behaviours & Pathfinding - Iteration 5

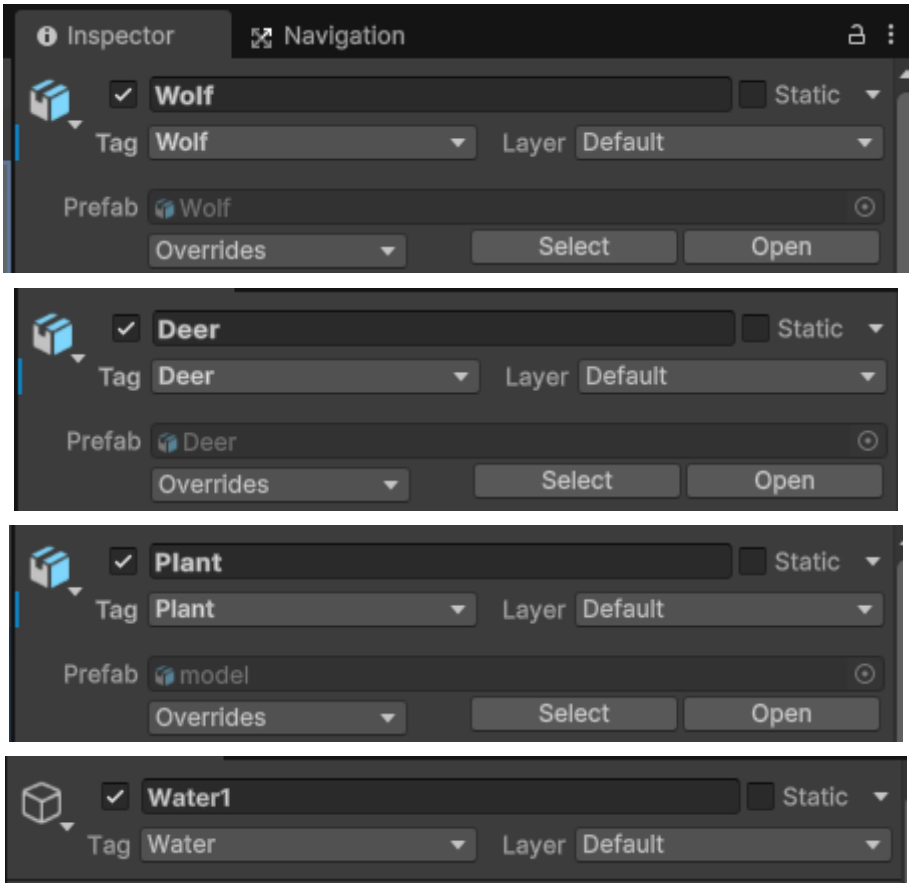
3.6.1 Intro/Justification

In this iteration I'll be setting up all the organism behaviours logic along with the corresponding animations and sound effects. The organism behaviours of eating, searching for water, drinking, searching for mate, mating, sleeping and idle will be added for the organisms deer and wolves. Searching for plants will be added only for deer and hunting will be added only for wolves. For the pathfinding I will be using Unitys built in 3D pathfinding rather than coding my own A* pathfinding algorithm using nodes. Unity 3D pathfinding uses NavMeshAgents and then I will be incorporating scripting to determine which objects should move to which other objects depending on the current behaviour status. This is a feature from the VAV Ecosystem Simulation and fulfils success criteria 7. I know stakeholders want this due to the results of the user questionnaire; behaviours will make the game feel more immersive and add realism. I've also decided to not include the day or night cycle along with the sleeping behaviour even though this is in the success criteria due to complexity of setting up the day and night cycle and having the organisms sleep.

Setup 3D Pathfinding Agents In Minutes Using Unity
https://discussions.unity.com/t/destroy-gameobject/861745
https://discussions.unity.com/t/move-object-in-opposite-direction-of-another/212974

3.6.2 Annotated Code

Added Deer, Wolf, Plant and Water tags to the prefabs for detection



DeerOOP.cs

```
// Pathfinding variables
public LayerMask detectionLayer;
public float distanceToWolf;
public float distanceToPlant;
public float distanceToMate;
public float distanceToWater;
public int matingCooldown = 0;
```

```
// Update function
Unity Message | 0 references
void Update()
{
    // Checks if organism is alive
    if (isAlive)
    {
        Detection();
        BehaviourUpdating();
        Pathfinding();

        // If organism is not alive then decrease population by 1 and set isAlive to false
        if (health <= 0)
        {
            organismDead();
        }
        // Makes sure hunger, thirst and stamina cannot go below 0
        else if (hunger <= 0) hunger = 0;
        else if (thirst <= 0) thirst = 0;
        else if (stamina <= 0) stamina = 0;

        // Updates movement state depending on stamina
        if (stamina <= 25)
        {
            movementState = "walking";
        }
        else if (stamina > 25)
        {
            movementState = "running";
        }

        // Decrease time by the frames that have passed
        decreaseTimer += Time.deltaTime;
        logTimer += Time.deltaTime;

        // Updates variables every second
        if (decreaseTimer >= decreaseInterval)
        {
            // Naturally  decrease hunger and thrist over time
            if (health >= 0)
            {
                hunger -= 3;
                thirst -= 3;
            }
            // If hunger or thirst is <= 50 then decrease health and stamina
            if (hunger <= 50 || thirst <= 50)
            {
                health -= 3;
                stamina -= 3;
            }
            // Decrease mating cooldown by 1 every second if it is greater than zero
            if (matingCooldown > 0)
            {
                matingCooldown -= 1;
            }

            decreaseTimer = 0f;
        }

        // Log stats at the specified interval
        if (logTimer >= logInterval)
        {
            Debug.Log($"Health: {health}, Stamina: {stamina}, Hunger: {hunger}, Thirst: {thirst}");
            logTimer = 0f;
        }
    }
}
```

```
// Detection logic used to find distances to other objects
2 references
public void Detection()
{
    GameObject closestWolf = null;
    float closestDistanceToWolf = Mathf.Infinity;
    GameObject closestPlant = null;
    float closestDistanceToPlant = Mathf.Infinity;
    GameObject closestWater = null;
    float closestDistanceToWater = Mathf.Infinity;
    GameObject closestDeer = null;
    float closestDistanceToMate = Mathf.Infinity;
```

```
    // Detect the closest wolf
    foreach (GameObject wolf in GameObject.FindGameObjectsWithTag("Wolf"))
    {
        float distance = Vector3.Distance(transform.position, wolf.transform.position);
        if (distance < closestDistanceToWolf && distance <= 100f)
        {
            closestDistanceToWolf = distance;
            distanceToWolf = closestDistanceToWolf;
            closestWolf = wolf;
        }

        // Destroy if colliding with a Wolf
        if (distance <= 1f)
        {
            organismDead();
            return;
        }
    }

    // Detect the closest plant
    foreach (GameObject plant in GameObject.FindGameObjectsWithTag("Plant"))
    {
        float distance = Vector3.Distance(transform.position, plant.transform.position);
        if (distance < closestDistanceToPlant && distance <= 100f)
        {
            closestDistanceToPlant = distance;
            distanceToPlant = closestDistanceToPlant;
            closestPlant = plant;
        }
    }

    // Detect the closest water
    foreach (GameObject water in GameObject.FindGameObjectsWithTag("Water"))
    {
        float distance = Vector3.Distance(transform.position, water.transform.position);
        if (distance < closestDistanceToWater && distance <= 100f)
        {
            closestDistanceToWater = distance;
            distanceToWater = closestDistanceToWater;
            closestWater = water;
        }
    }

    // Detect the closest deer for mating
    foreach (GameObject deer in GameObject.FindGameObjectsWithTag("Deer"))
    {
        if (deer == gameObject || !deer.activeInHierarchy)
        {
            continue; // Skip self or inactive deer
        }

        DeerOOP potentialMate = deer.GetComponent<DeerOOP>();
        if (potentialMate != null && potentialMate.behaviouralState != "mating")
        {
            // If potential mate is found and it's not already mating
            float distance = Vector3.Distance(transform.position, deer.transform.position);
            if (distance < closestDistanceToMate && distance <= 100f)
            {
                closestDistanceToMate = distance;
                distanceToMate = closestDistanceToMate;
                closestDeer = deer;
            }
        }
    }
}
```

```
// Updates the Behavioural State depending on certain conditions
2 references
void BehaviourUpdating()
{
    // If a wolf is near the deer then run from it and take highest priority
    if (distanceToWolf <= radius)
    {
        behaviouralState = "runFromWolf";
        return;
    }

    // Define useful variables
    bool isHungry = hunger <= (hungerMax / 2);
    bool isThirsty = thirst <= (thirstMax / 2);
    bool needsFood = isHungry && hunger < thirst && distanceToPlant > 3f;
    bool needsWater = isThirsty && thirst < hunger && distanceToWater > 3f;
    bool nearPlant = distanceToPlant <= 3f;
    bool nearWater = distanceToWater <= 3f;
    bool nearMate = distanceToMate <= 3f;
```

```
    // Behavioural conditions with the priority order: searching for mate, then water, then food
    if (!isHungry && !isThirsty && (stamina > (staminaMax / 2)) && (matingCooldown == 0))
    {
        if (!nearMate)
        {
            behaviouralState = "searchMate";
        }
        else
        {
            behaviouralState = "mating";
        }
    }
    else if (needsFood)
    {
        behaviouralState = "searchPlant";
    }
    else if (needsWater)
    {
        behaviouralState = "searchWater";
    }
    else if (isHungry && isThirsty && hunger == thirst && distanceToWater > 1f)
    {
        behaviouralState = "searchWater";
    }
    else if (nearWater && nearPlant)
    {
        if (thirst <= hunger)
        {
            behaviouralState = "drinking";
        }
        else
        {
            behaviouralState = "eating";
        }
    }
    else if (nearPlant)
    {
        behaviouralState = "eating";
    }
    else if (nearWater)
    {
        behaviouralState = "drinking";
    }
    else
    {
        if (thirst <= hunger)
        {
            behaviouralState = "searchWater";
        }
        else
        {
            behaviouralState = "searchPlant";
        }
    }
}
```

```
// Function for the behavioural status of the wolf that determines the pathfinding
2 references
void Pathfinding()
{
    switch (behaviouralState)
    {
        case "runFromWolf":
            RunFromWolf();
            break;

        case "searchPlant":
            SearchForPlant();
            break;

        case "searchWater":
            SearchForWater();
            break;

        case "searchMate":
            SearchForMate();
            break;

        case "eating":
            // Eating behavior
            hunger = hungerMax;
            break;

        case "drinking":
            //Drinking();
            thirst = thirstMax;
            break;

        case "mating":
            // Mating behavior
            matingCooldown = 20;
            break;

        default:
            Debug.LogError("Error: " + organismName + " has an unrecognized behaviouralState.");
            break;
    }
}
```

```
// Pathfinding to move deer the opposite direction from the wolf
1 reference
void RunFromWolf()
{
    GameObject closestWolf = null;
    float closestDistanceToWolf = Mathf.Infinity;

    // Find the closest wolf
    foreach (GameObject wolf in GameObject.FindGameObjectsWithTag("Wolf"))
    {
        float distance = Vector3.Distance(transform.position, wolf.transform.position);
        if (distance < closestDistanceToWolf && distance <= 100f)
        {
            closestDistanceToWolf = distance;
            closestWolf = wolf;
        }
    }

    // If a wolf is detected, run away
    if (closestWolf != null)
    {
        agent.isStopped = false;
        if (movementState == "running")
        {
            agent.speed = 8f;
        }
        else
        {
            agent.speed = 4f;
        }
        Vector3 directionAwayFromWolf = transform.position - closestWolf.transform.position;
        agent.SetDestination(transform.position + directionAwayFromWolf);
    }
    else
    {
        agent.isStopped = true;
    }
}
```

```
// Pathfinding to move deer towards the closest plant
1 reference
void SearchForPlant()
{
    GameObject closestPlant = null;
    float closestDistanceToPlant = Mathf.Infinity;

    // Find the closest plant
    foreach (GameObject plant in GameObject.FindGameObjectsWithTag("Plant"))
    {
        float distance = Vector3.Distance(transform.position, plant.transform.position);
        if (distance < closestDistanceToPlant && distance <= 100f)
        {
            closestDistanceToPlant = distance;
            closestPlant = plant;
        }
    }

    if (closestPlant != null)
    {
        if (closestDistanceToPlant <= agent.stoppingDistance * 1.5f)
        {
            agent.isStopped = true;
            return;
        }

        agent.isStopped = false;
        if (movementState == "running")
        {
            agent.speed = 8f;
        }
        else
        {
            agent.speed = 4f;
        }
        agent.SetDestination(closestPlant.transform.position);
    }
    else
    {
        agent.isStopped = true;
    }
}
```

```
// Pathfinding to move deer towards the closest water
1 reference
void SearchForWater()
{
    GameObject closestWater = null;
    float closestDistanceToWater = Mathf.Infinity;

    // Find the closest water source
    foreach (GameObject water in GameObject.FindGameObjectsWithTag("Water"))
    {
        float distance = Vector3.Distance(transform.position, water.transform.position);
        if (distance < closestDistanceToWater && distance <= 100f)
        {
            closestDistanceToWater = distance;
            closestWater = water;
        }
    }

    if (closestWater != null)
    {
        if (closestDistanceToWater <= agent.stoppingDistance * 1.5f)
        {
            agent.isStopped = true;
            return;
        }

        agent.isStopped = false;
        if (movementState == "running")
        {
            agent.speed = 8f;
        }
        else
        {
            agent.speed = 4f;
        }
        agent.SetDestination(closestWater.transform.position);
    }
    else
    {
        agent.isStopped = true;
    }
}
```

```
// Pathfinding to move deer towards the closest mate
1 reference
void SearchForMate()
{
    GameObject closestDeer = null;
    float closestDistanceToMate = Mathf.Infinity;

    // Find the closest potential mate
    foreach (GameObject deer in GameObject.FindGameObjectsWithTag("Deer"))
    {
        if (deer == gameObject || !deer.activeInHierarchy)
        {
            continue;
        }

        DeerOOP potentialMate = deer.GetComponent<DeerOOP>();
        if (potentialMate != null && Vector3.Distance(transform.position, deer.transform.position) < closestDistanceToMate)
        {
            closestDistanceToMate = Vector3.Distance(transform.position, deer.transform.position);
            closestDeer = deer;
        }
    }

    if (closestDeer != null)
    {
        agent.isStopped = false;
        if (movementState == "running")
        {
            agent.speed = 8f;
        }
        else
        {
            agent.speed = 4f;
        }
        agent.SetDestination(closestDeer.transform.position);
    }
    else
    {
        agent.isStopped = true;
    }
}
```

```
2 references
void organismDead()
{
    health = 0;
    EnvironmentData.DeerPopulation -= 1;
    isAlive = false;
    // If organism is dead then destory gameObject
    Object.Destroy(gameObject);
}
```

WolfOOP.cs

```
// Detection logic used to find distances to other objects
1 reference
public void Detection()
{

    GameObject closestDeer = null;
    float closestDistanceToDeer = Mathf.Infinity;
    GameObject closestWater = null;
    float closestDistanceToWater = Mathf.Infinity;
    GameObject closestWolf = null;
    float closestDistanceToMate = Mathf.Infinity;


    // Detect the closest deer
    foreach (GameObject deer in GameObject.FindGameObjectsWithTag("Deer"))
    {
        float distance = Vector3.Distance(transform.position, deer.transform.position);
        if (distance < closestDistanceToDeer && distance <= 100f)
        {
            closestDistanceToDeer = distance;
            distanceToDeer = closestDistanceToDeer;
            closestDeer = deer;
        }
    }

    // Detect the closest water
    foreach (GameObject water in GameObject.FindGameObjectsWithTag("Water"))
    {
        float distance = Vector3.Distance(transform.position, water.transform.position);
        if (distance < closestDistanceToWater && distance <= 100f)
        {
            closestDistanceToWater = distance;
            distanceToWater = closestDistanceToWater;
            closestWater = water;
        }
    }

    // Detect the closest wolf for mating
    foreach (GameObject wolf in GameObject.FindGameObjectsWithTag("Wolf"))
    {
        if (wolf == gameObject || !wolf.activeInHierarchy)
        {
            continue; // Skip self or inactive wolf
        }

        WolfOOP potentialMate = wolf.GetComponent<WolfOOP>();
        if (potentialMate != null && potentialMate.behaviouralState != "mating")
        {
            // If potential mate is found and it's not already mating
            float distance = Vector3.Distance(transform.position, wolf.transform.position);
            if (distance < closestDistanceToMate && distance <= 100f)
            {
                closestDistanceToMate = distance;
                distanceToMate = closestDistanceToMate;
                closestWolf = wolf;
            }
        }
    }
}
```

```
// Updates the Behavioural State depending on certain conditions
1 reference
void BehaviourUpdating()
{
    // Define useful variables
    bool isHungry = hunger <= (hungerMax / 2);
    bool isThirsty = thirst <= (thirstMax / 2);
    bool needsFood = isHungry && hunger < thirst && distanceToDeer > 3f;
    bool needsWater = isThirsty && thirst < hunger && distanceToWater > 3f;
    bool nearDeer = distanceToDeer <= 3f;
    bool nearWater = distanceToWater <= 3f;
    bool nearMate = distanceToMate <= 3f;

    // Behavioural conditions with the priority order: searching for mate, then water, then food
    if (!isHungry && !isThirsty && (stamina > (staminaMax / 2)) && (matingCooldown == 0))
    {
        if (!nearMate)
        {
            behaviouralState = "searchMate";
        }
        else
        {
            behaviouralState = "mating";
        }
    }
    else if (needsFood)
    {
        behaviouralState = "searchDeer";
    }
    else if (needsWater)
    {
        behaviouralState = "searchWater";
    }
    else if (isHungry && isThirsty && hunger == thirst && distanceToWater > 1f)
    {
        behaviouralState = "searchWater";
    }
    else if (nearWater && nearDeer)
    {
        if (thirst <= hunger)
        {
            behaviouralState = "drinking";
        }
        else
        {
            behaviouralState = "eating";
        }
    }
    else if (nearDeer)
    {
        behaviouralState = "eating";
    }
    else if (nearWater)
    {
        behaviouralState = "drinking";
    }
    else
    {
        if (thirst <= hunger)
        {
            behaviouralState = "searchWater";
        }
        else
        {
            behaviouralState = "searchDeer";
        }
    }
}
```

```
// Function for the behavioural status of the wolf that determines the pathfinding
1 reference
void Pathfinding()
{
    switch (behaviouralState)
    {
        case "searchDeer":
            SearchForDeer();
            break;

        case "searchWater":
            SearchForWater();
            break;

        case "searchMate":
            SearchForMate();
            break;

        case "eating":
            // Eating behavior
            hunger = hungerMax;
            break;

        case "drinking":
            // Drinking behavior
            thirst = thirstMax;
            break;

        case "mating":
            // Mating behavior
            matingCooldown = 20;
            break;

        default:
            Debug.LogError("Error: " + organismName + " has an unrecognized behaviouralState.");
            break;
    }
}
```

OrganismInitialization.cs

Added functionality to randomly spawn in plants using

```
// Randomly spawning new plants
private float spawnTimer = 0f;
private float spawnInterval = 1f;
```

```
Unity Message | 0 references
void Update()
{
    if (isSimulationRunning)
    {
        // Decrease time remaining by the time passed since the last frame
        timeRemaining -= Time.deltaTime;

        // Check if the simulation has ended
        if (timeRemaining <= 0)
        {
            timeRemaining = 0;
            isSimulationRunning = false;
            Debug.Log("Simulation Ended");
        }

        // Random chance to spawn a plant every 1 second
        spawnTimer += Time.deltaTime;

        if (spawnTimer >= spawnInterval)
        {
            int randomNum = Random.Range(0, 5);
            if (randomNum == 0)
            {
                SpawnObjects(PlantPrefab, 1);
            }
        }

    }
}
```

PlantOOP.cs

Create PlantOOP script to delete plant if a deer eats it

```
// Update is called once per frame
Unity Message | 0 references
void Update()
{
    Detection();
}

// Function to delete plant if deer is eating it
1 reference
public void Detection()
{
    // Detect the closest deer
    foreach (GameObject deer in GameObject.FindGameObjectsWithTag("Deer"))
    {
        float distance = Vector3.Distance(transform.position, deer.transform.position);

        // Destroy if colliding with a deer
        if (distance <= 1f)
        {
            EnvironmentData.PlantPopulation -= 1;
            Object.Destroy(gameObject);
            return;
        }
    }
}
```

3.6.3 Iterative Testing

1 - Deer

Test No.	Success Criteria	Test Aim	Input	Expected Output	Actual Output	Pass/Fail	Fixes
1.1	7.1	Searching for Plant Behaviour	Hunger is <= half of max hunger and hunger is less than thirst and a wolf isn't in the deers radius	Pathfinding will take place to find the nearest plant	Stats were decreasing by 3 times less speed then they should be when hunger is less than or equal to half the max hunger and hunger is less than thirst and wolf isn't in radius	Fail	Check 3.6.4 Fixes Applied No.1
1.2	7.1	Eating Behaviour	Plant in deers radius and wolf isn't	Eating animation and hunger and stamina will increase	Eating animation playa and hunger and stamina increase when plants are in radius and wolf isn't	Pass	-
1.3	7.2	Searching for Water Behaviour	Thirst is less than half of max thirst and thirst is less than hunger	Pathfinding will take place to find water	Pathfinding takes place to find the nearest water when thirst is less than half of the max and thirst is less than hunger	Pass	-
1.4	7.2	Drinking Behaviour	Water in radius	Drinking animation and thirst increases	Drinking animation plays and thirst increases when water is in radius	Pass	-
1.5	7.4	Searching for Mate Behaviour	Hunger, thirst and stamina is more than half the maxes	Pathfinding will take place to find the nearest mate	Pathfinding takes place to find the nearest mate that isn't current mating or in danger when hunger, thirst and stamina are greater than half the maxes	Pass	-
1.6	7.4	Mating Behaviour	Mate in radius	Mating animation and stamina decreases	Mating animation plays and stamina decrease when mate is in radius	Pass	-

2 - Wolves

Test No.	Success Criteria	Test Aim	Input	Expected Output	Actual Output	Pass/Fail	Fixes
2.1	7.1	Hunting Behaviour	Hunger is <= half of max hunger and hunger is less than thirst	Pathfinding will take place to find the nearest deer	Stats of organisms weren't decreasing when hunger is less than or equal to half the max hunger and hunger is less than thirst	Fail	Check 3.6.4 Fixes Applied No.2
2.2	7.1	Eating Behaviour	Deer dead and in wolf radius	Eating animation and hunger and stamina will increase	Eating animation plays and hunger and stamina increases when a dead deer is in radius	Pass	-
2.3	7.2	Searching for Water Behaviour	Thirst is less than half of max thirst and thirst is less than hunger	Pathfinding will take place to find water	Pathfinding takes place to find the nearest water when thirst is less than half of the max and thirst is less than hunger	Pass	-
2.4	7.2	Drinking Behaviour	Water in radius	Drinking animation and thirst increases	Drinking animation plays and thirst increases when water is in radius	Pass	-
2.5	7.4	Searching for Mate Behaviour	Hunger, thirst and stamina is more than half the maxes	Pathfinding will take place to find the nearest mate	Pathfinding takes place to find the nearest mate that isn't current mating or in danger when hunger, thirst and stamina are greater than half the maxes	Pass	-
2.6	7.4	Mating Behaviour	Mate in radius	Mating animation and stamina decreases	Mating animation plays and stamina decrease when mate is in radius	Pass	-

3.6.4 Fixes Applied

No.1 - Organism Behaviours Deer - Searching for Plant Behaviour

Evidence

```
[00:38:23] Health: 100, Stamina: 100, Hunger: 91, Thirst: 91
UnityEngine.Debug:Log (object)
[00:38:23] Health: 100, Stamina: 100, Hunger: 91, Thirst: 91
UnityEngine.Debug:Log (object)
[00:39:37] Health: 100, Stamina: 100, Hunger: 86, Thirst: 86
UnityEngine.Debug:Log (object)
[00:39:37] Health: 100, Stamina: 100, Hunger: 86, Thirst: 86
UnityEngine.Debug:Log (object)
```

Fix

Changed decreaseInterval and logInterval to 1f

```
private float decreaseTimer = 0f;
private float decreaseInterval = 1f;
private float logTimer = 0f;
private float logInterval = 1f;
```

```
[00:45:00] Health: 100, Stamina: 100, Hunger: 86, Thirst: 86
UnityEngine.Debug:Log (object)
[00:45:01] Health: 100, Stamina: 100, Hunger: 85, Thirst: 85
UnityEngine.Debug:Log (object)
```

Retest

Test No.	Success Criteria	Test Aim	Input	Expected Output	Actual Output	Pass/Fail	Fixes
1.1	7.1	Searching for Plant Behaviour	Hunger is <= half of max hunger and hunger is less than thirst and a wolf isn't in the deers radius	Pathfinding will take place to find the nearest plant	Pathfinding took place to find and move to the nearest plant	Pass	-

No.2 - Organism Behaviours Wolves - Hunting Behaviour

Evidence

```
[00:37:17] Health: 100, Stamina: 100, Hunger: 100, Thirst: 100
UnityEngine.Debug:Log (object)
[00:37:17] Health: 100, Stamina: 100, Hunger: 100, Thirst: 100
UnityEngine.Debug:Log (object)
[00:37:17] Health: 100, Stamina: 100, Hunger: 100, Thirst: 100
UnityEngine.Debug:Log (object)
[00:37:17] Health: 100, Stamina: 100, Hunger: 100, Thirst: 100
UnityEngine.Debug:Log (object)
[00:37:17] Health: 100, Stamina: 100, Hunger: 100, Thirst: 100
UnityEngine.Debug:Log (object)
[00:37:17] Health: 100, Stamina: 100, Hunger: 100, Thirst: 100
UnityEngine.Debug:Log (object)
```

Fix

Made sure that hunger and thirst decrease using a timer

```
// Updates variables every second
if (decreaseTimer >= decreaseInterval)
{
    // Naturally  decrease hunger and thrist over time
    if (health >= 0)
    {
        hunger -= 3;
        thirst -= 3;
    }
```

Retest

Test No.	Success Criteria	Test Aim	Input	Expected Output	Actual Output	Pass/Fail	Fixes
2.1	7.1	Hunting Behaviour	Hunger is <= half of max hunger and hunger is less than thirst	Pathfinding will take place to find the nearest deer	Pathfinding took place with wolf moving to closest deer	Pass	-

3.6.5 Success from Testing

6177_organismBehaviorsAndPathfinding_Test.mp4

3.6.6 Review

In this iteration I've changed some values from my design pseudocode when creating the actual code so that the simulation can operate smoother and at a faster rate; such as the amount the hunger and thirst naturally decrease every second. I've changed this from decreasing by 1 every second to instead decreasing by 3 every second just to speed up the simulation and make it more engaging to the user. I've also decided to remove the idle and sleeping behaviours. This is since they also slowed down the simulation, for example the idle just made the simulation come to a complete halt. The user would have to wait until the organisms have met conditions to start searching for water or food which means their hunger or thirst would have to go below 50. I've also added a 20% chance for a plant to spawn in every second so that the plant population doesn't deplete from deers eating them. Furthermore, I've made a change to the player controlled movement; the user will no longer be able to control the behaviours of the organism through manual input, rather they will just manually occur based on the behavioural conditions in the deerOOP and wolfOOP scripts.

Overall, the client expressed satisfaction with all features added in this iteration which included the organism behaviours along with the corresponding pathfinding, along with the slight changes made from the initial design.

Client Signature :

3.7 Organism Genetics & Mutations - Iteration 6

3.7.1 Intro/Justification

In this iteration I will be adding the organism's genetics along with the probability of a mutation occurring. The maximum value of the organism's health, hunger, thirst, stamina, speed and sigh will be calculated using an equation which gets the average of the two parents max values and then multiplies them by a random number from 0.9 to 1.1. Moving the average up or down 10%. Then there is a 1% for each value that a random mutation takes place where the maximum value is multiplied by 1.25. This feature is from the VAV Ecosystem Simulation and fulfils success criteria 8.

3.7.2 Annotated Code

DeerOOP.cs

```
// Updates stats to calculated ones
1 reference
public void InitialiseStats(int health, int hunger, int thirst, int stamina, int speed, int sight)
{
    healthMax = health;
    hungerMax = hunger;
    thirstMax = thirst;
    staminaMax = stamina;
    speedMax = speed;
    radius = sight;
}

// Applies mutations and genetics and instatiates offspring
0 references
public static DeerOOP Reproduce(DeerOOP parentA, DeerOOP parentB, GameObject deerPrefab)
{
    // Base multipliers
    float healthMultiplier = Random.Range(0.9f, 1.1f);
    float hungerMultiplier = Random.Range(0.9f, 1.1f);
    float thirstMultiplier = Random.Range(0.9f, 1.1f);
    float staminaMultiplier = Random.Range(0.9f, 1.1f);
    float speedMultiplier = Random.Range(0.9f, 1.1f);
    float sightMultiplier = Random.Range(0.9f, 1.1f);

    // Mutation probabilities
    int healthMutation = Random.Range(0, 100);
    int hungerMutation = Random.Range(0, 100);
    int thirstMutation = Random.Range(0, 100);
    int staminaMutation = Random.Range(0, 100);
    int speedMutation = Random.Range(0, 100);
    int sightMutation = Random.Range(0, 100);

    // Creates attributes and rounds them using parent values
    int health = Mathf.RoundToInt(((parentA.healthMax + parentB.healthMax) / 2f) * healthMultiplier);
    int hunger = Mathf.RoundToInt(((parentA.hungerMax + parentB.hungerMax) / 2f) * hungerMultiplier);
    int thirst = Mathf.RoundToInt(((parentA.thirstMax + parentB.thirstMax) / 2f) * thirstMultiplier);
    int stamina = Mathf.RoundToInt(((parentA.staminaMax + parentB.staminaMax) / 2f) * staminaMultiplier);
    int speed = Mathf.RoundToInt(((parentA.speedMax + parentB.speedMax) / 2f) * speedMultiplier);
    int sight = Mathf.RoundToInt(((parentA.radius + parentB.radius) / 2f) * sightMultiplier);

    // Mutation checks
    if (healthMutation == 50) health = Mathf.RoundToInt(health * 1.25f);
    else if (hungerMutation == 50) hunger = Mathf.RoundToInt(hunger * 1.25f);
    else if (thirstMutation == 50) thirst = Mathf.RoundToInt(thirst * 1.25f);
    else if (staminaMutation == 50) stamina = Mathf.RoundToInt(stamina * 1.25f);
    else if (speedMutation == 50) speed = Mathf.RoundToInt(speed * 1.25f);
    else if (sightMutation == 50) sight = Mathf.RoundToInt(sight * 1.25f);

    // Spawn new deer
    Vector3 spawnPos = parentA.transform.position + new Vector3(Random.Range(-2f, 2f), 0, Random.Range(-2f, 2f));
    GameObject baby = Instantiate(deerPrefab, spawnPos, Quaternion.identity);
    DeerOOP babyDeer = baby.GetComponent<DeerOOP>();
    babyDeer.InitialiseStats(health, hunger, thirst, stamina, speed, sight);

    return babyDeer;
}
```

WolfOOP.cs

```
// Updates stats to calculated ones
1 reference
public void InitialiseStats(int health, int hunger, int thirst, int stamina, int speed, int sight)
{
    healthMax = health;
    hungerMax = hunger;
    thirstMax = thirst;
    staminaMax = stamina;
    speedMax = speed;
    radius = sight;
}

// Applies mutations and genetics and instatiates offspring
0 references
public static WolfOOP Reproduce(WolfOOP parentA, WolfOOP parentB, GameObject wolfPrefab)
{
    // Base multipliers
    float healthMultiplier = Random.Range(0.9f, 1.1f);
    float hungerMultiplier = Random.Range(0.9f, 1.1f);
    float thirstMultiplier = Random.Range(0.9f, 1.1f);
    float staminaMultiplier = Random.Range(0.9f, 1.1f);
    float speedMultiplier = Random.Range(0.9f, 1.1f);
    float sightMultiplier = Random.Range(0.9f, 1.1f);

    // Mutation probabilities
    int healthMutation = Random.Range(0, 100);
    int hungerMutation = Random.Range(0, 100);
    int thirstMutation = Random.Range(0, 100);
    int staminaMutation = Random.Range(0, 100);
    int speedMutation = Random.Range(0, 100);
    int sightMutation = Random.Range(0, 100);

    // Creates attributes and rounds them using parent values
    int health = Mathf.RoundToInt(((parentA.healthMax + parentB.healthMax) / 2f) * healthMultiplier);
    int hunger = Mathf.RoundToInt(((parentA.hungerMax + parentB.hungerMax) / 2f) * hungerMultiplier);
    int thirst = Mathf.RoundToInt(((parentA.thirstMax + parentB.thirstMax) / 2f) * thirstMultiplier);
    int stamina = Mathf.RoundToInt(((parentA.staminaMax + parentB.staminaMax) / 2f) * staminaMultiplier);
    int speed = Mathf.RoundToInt(((parentA.speedMax + parentB.speedMax) / 2f) * speedMultiplier);
    int sight = Mathf.RoundToInt(((parentA.radius + parentB.radius) / 2f) * sightMultiplier);

    // Mutation checks
    if (healthMutation == 50) health = Mathf.RoundToInt(health * 1.25f);
    else if (hungerMutation == 50) hunger = Mathf.RoundToInt(hunger * 1.25f);
    else if (thirstMutation == 50) thirst = Mathf.RoundToInt(thirst * 1.25f);
    else if (staminaMutation == 50) stamina = Mathf.RoundToInt(stamina * 1.25f);
    else if (speedMutation == 50) speed = Mathf.RoundToInt(speed * 1.25f);
    else if (sightMutation == 50) sight = Mathf.RoundToInt(sight * 1.25f);

    // Spawn new wolf
    Vector3 spawnPos = parentA.transform.position + new Vector3(Random.Range(-2f, 2f), 0, Random.Range(-2f, 2f));
    GameObject baby = Instantiate(wolfPrefab, spawnPos, Quaternion.identity);
    WolfOOP babyWolf = baby.GetComponent<WolfOOP>();
    babyWolf.InitialiseStats(health, hunger, thirst, stamina, speed, sight);

    return babyWolf;
}
```

3.7.3 Iterative Testing

1 - Genetics

Test No.	Success Criteria	Test Aim	Input	Expected Output	Actual Output	Pass/Fail	Fixes
1.1	8	Health Max Value	Parent A and B Health Max Value and Random Multiplier from 0.9 to 1.1	Child healthMax = ((parentA.healthMax + parentB.healthMax) / 2) * healthMultiplier	Child healthMax = ((parentA.healthMax + parentB.healthMax) / 2) * healthMultiplier	Fail	Check 3.7.4 Fixes Applied No.1
1.2	8	Hunger Max Value	Parent A and B Hunger Max Value and Random Multiplier from 0.9 to 1.1	Child hungerMax = ((parentA.hungerMax + parentB.hungerMax) / 2) * hungerMultiplier	Child hungerMax = ((parentA.hungerMax + parentB.hungerMax) / 2) * hungerMultiplier	Pass	-
1.3	8	Thirst Max Value	Parent A and B Thirst Max Value and Random Multiplier from 0.9 to 1.1	Child thirstMax = ((parentA.thirstMax + parentB.thirstMax) / 2) * thirstMultiplier	Child thirstMax = ((parentA.thirstMax + parentB.thirstMax) / 2) * thirstMultiplier	Pass	-
1.4	8	Stamina Max Value	Parent A and B Stamina Max Value and Random Multiplier from 0.9 to 1.1	Child staminaMax = ((parentA.staminaMax + parentB.staminaMax) / 2) * staminaMultiplier	Child staminaMax = ((parentA.staminaMax + parentB.staminaMax) / 2) * staminaMultiplier	Pass	-
1.5	8	Speed Max Value	Parent A and B Speed Max Value and Random Multiplier from 0.9 to 1.1	Child speedMax = ((parentA.speedMax + parentB.speedMax) / 2) * speedMultiplier	Child speedMax = ((parentA.speedMax + parentB.speedMax) / 2) * speedMultiplier	Pass	-
1.6	8	Sight Max Value	Parent A and B Sight Max Value and Random Multiplier from 0.9 to 1.1	Child sightMax = ((parentA.sightMax + parentB.sightMax) / 2) * sightMultiplier	Child sightMax = ((parentA.sightMax + parentB.sightMax) / 2) * sightMultiplier	Pass	-

2- Mutation

Test No.	Success Criteria	Test Aim	Input	Expected Output	Actual Output	Pass/Fail	Fixes
2.1	10	Health Mutation	Random generated number from 0 to 99	1% chance of healthMax being multiplied by 1.25	1% chance of healthMax being multiplied by 1.25	Pass	-
2.2	10	Hunger Mutation	Random generated number from 0 to 99	1% chance of hungerMax being multiplied by 1.25	1% chance of hungerMax being multiplied by 1.25	Pass	-
2.3	10	Thirst Mutation	Random generated number from 0 to 99	1% chance of thirstMax being multiplied by 1.25	1% chance of thirstMax being multiplied by 1.25	Pass	-
2.4	10	Stamina Mutation	Random generated number from 0 to 99	1% chance of staminaMax being multiplied by 1.25	1% chance of staminaMax being multiplied by 1.25	Pass	-
2.5	10	Speed Mutation	Random generated number from 0 to 99	1% chance of speedMax being multiplied by 1.25	1% chance of speedMax being multiplied by 1.25	Pass	-
2.6	10	Sight Mutation	Random generated number from 0 to 99	1% chance of sightMax being multiplied by 1.25	1% chance of sightMax being multiplied by 1.25	Pass	-

3.7.4 Fixes Applied

No.2 - Organism Behaviours Wolves - Hunting Behaviour

Evidence

Organisms weren't instantiating when mating occurred

Fix

Forgot to add the call to the function into the mating case within the pathfinding function.

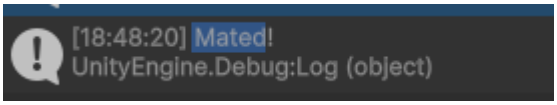
```
1 reference
private DeerOOP FindSecondClosestDeer()
{
    var deerList = GameObject.FindGameObjectsWithTag("Deer")
        .Where(d => d != gameObject && d.activeInHierarchy)
        .Select(d => new { obj = d, dist = Vector3.Distance(transform.position, d.transform.position) })
        .OrderBy(d => d.dist)
        .ToList();

    return deerList.Count >= 2
        ? deerList[1].obj.GetComponent<DeerOOP>()
        : (deerList.Count == 1 ? deerList[0].obj.GetComponent<DeerOOP>() : null);
}
```

```
case "mating":
    if (matingCooldown == 0)
    {
        matingCooldown = 20;
        DeerOOP parentA = GetComponent<DeerOOP>();
        DeerOOP parentB = FindSecondClosestDeer();
        Reproduce(parentA, parentB, DeerPrefab);
        Debug.Log("Mated!");
    }
    break;
```

Retest

Test No.	Success Criteria	Test Aim	Input	Expected Output	Actual Output	Pass/Fail	Fixes
1.1	8	Health Max Value	Parent A and B Health Max Value and Random Multiplier from 0.9 to 1.1	Child healthMax = ((parentA.healthMax + parentB.healthMax) / 2) * healthMultiplier	Child healthMax = ((parentA.healthMax + parentB.healthMax) / 2) * healthMultiplier	Pass	-

3.7.5 Success from Testing



6117_mating_Test.mp4

3.7.6 Review

Overall the client was satisfied with this iteration. Both the genetics of the organisms and the possible mutations were working fully as intended and new organisms of both the deer and wolves were being instantiated when mating occurred with its new attributes which were derived from the calculations that took place using the parents attributes; allowing natural selection to now occur in my simulation. One problem that occurred is that the client noticed that sometimes there was a bug where two organisms would sometimes be created instead of a singular one. However, the client was overall pleased with this iteration and signed off on it.

Client Signature :

3.8 End Screen - Iteration 7

3.8.1 Intro/Justification

In this iteration I will be adding a final end screen which will be displayed when the simulation has concluded, once the timer has reached zero. On this screen it will display an overview of the simulation including the final populations of each organism: deer, wolves and plants. There will also be a graph displayed with the x-axis as the simulation time and the y-axis as the population. There will be three different lines for each different organism displaying how the organism populations changed over time and how each population affected the others. This feature is from the VAV Ecosystem Simulation and results of the user questionnaire. It fulfils success criteria 13.

3.8.2 Annotated Code

```
Unity Script (1 asset reference) | 7 references
public class EnvironmentData : MonoBehaviour
{
    // Get starting populatations and time from OrganismInitilisation variables
    public static int DeerPopulation = OrganismInitilisation.DeerStartingPopulation;
    public static int WolfPopulation = OrganismInitilisation.WolfStartingPopulation;
    public static int PlantPopulation = OrganismInitilisation.PlantStartingPopulation;
    public static float TimeRemaining = OrganismInitilisation.TimeStartingLength;
    private bool isSimulationRunning = true;

    // End Screen UI Values
    public Text totalPopulationText;
    public Text finalDeerPopulationText;
    public Text finalWolfPopulationText;
    public Text finalSimulationTimeText;

    // Get end screen menu canvas
    public GameObject CanvasEndScreen;


    // Start is called once before the first execution of Update after the MonoBehaviour is created
    Unity Message | 0 references
    void Start()
    {
        CanvasEndScreen.SetActive(false);
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        if (isSimulationRunning)
        {
            // Decrease time remaining by the time passed since the last frame
            TimeRemaining -= Time.deltaTime;

            // Check if the simulation has ended
            if (TimeRemaining <= 0 || DeerPopulation == 0 || WolfPopulation == 0)
            {
                TimeRemaining = 0;
                isSimulationRunning = false;
                Debug.Log("Simulation Ended");
                // Changes UI to end screen
                CanvasEndScreen.SetActive(true);

                // Set UI text values to current variable values
                totalPopulationText.text = DeerPopulation.ToString();
                finalDeerPopulationText.text = WolfPopulation.ToString();
                finalWolfPopulationText.text = WolfPopulation.ToString();
                finalSimulationTimeText.text = TimeRemaining.ToString();

            }
        }
    }
}
```

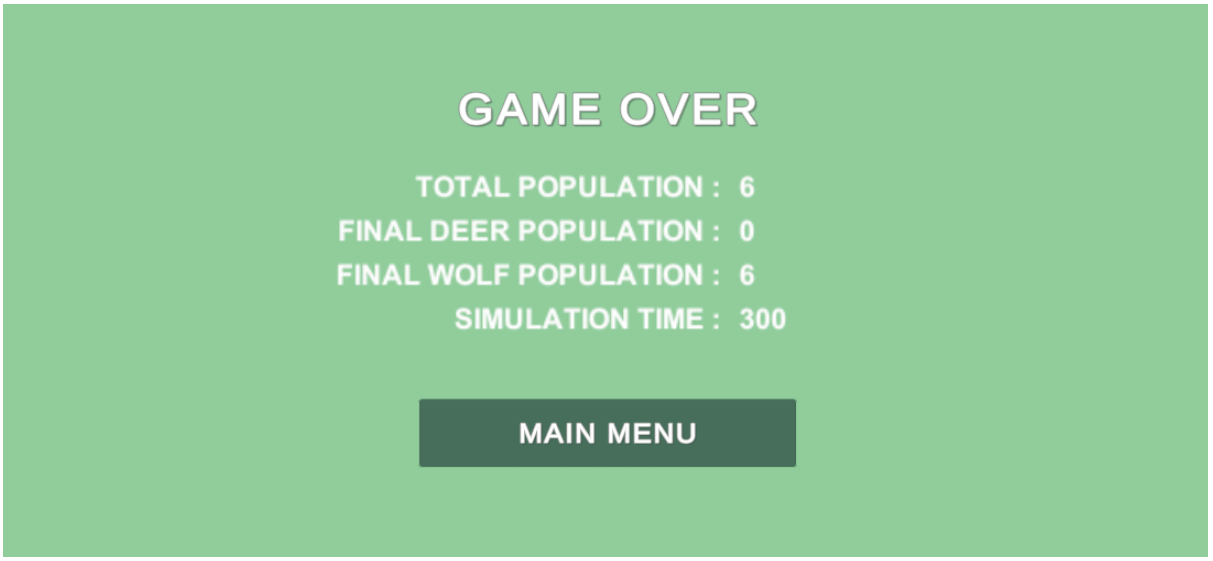
3.8.3 Iterative Testing

Test No.	Success Criteria	Test Aim	Input	Expected Output	Actual Output	Pass/Fail	Fixes
1.1	13	Graph Display	Population data gathered throughout the simulation	Population data is displayed on a graph plotted against time	Didn't include the graph in end screen	Fail	-
1.2	13	Statistics Overview	Population data at the end of the simulation	Final population data is displayed in text format	Final population and time was displayed in end screen	Pass	-

3.8.4 Fixes Applied

There were no errors or bugs produced however I decided to not develop the graph display and just the statistics overview.

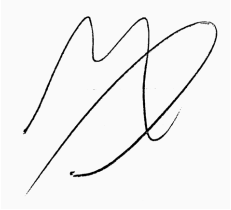
3.8.5 Success from Testing



3.8.6 Review

I've decided not to add the graph into the end screen due to time constraints and complications. Adding a graph would mean having to track what the population of each organism is at certain intervals of the simulation. However, overall for this iteration the client was satisfied since the end screen fulfilled its purpose of showing the results of the simulation. My client also suggested in the future to possibly add the starting populations along with the final populations so the user can compare the two at the end.

Client Signature :



Chapter 4 - EVALUATION

4.1 Post Development Testing for Function & Robustness

4.1.1 Test Scenario 1 - Free Cam Simulation

- Open the game
- Press start game
- Input starting conditions
    - Wolf starting population: 10
    - Deer starting population: 10
    - Plant starting population: 10
    - Time: 300 seconds
- Start the simulation
- Use free cam movement to move around
- Click on an organism to view their attributes
- Click on genetics to view that organism genetics
- Exit out of the genetics and attributes menu
- Pause the simulation
- Unpause the simulation
- Wait for simulation to end
- View the data in the game over menu
- Return to main menu
- Exit game

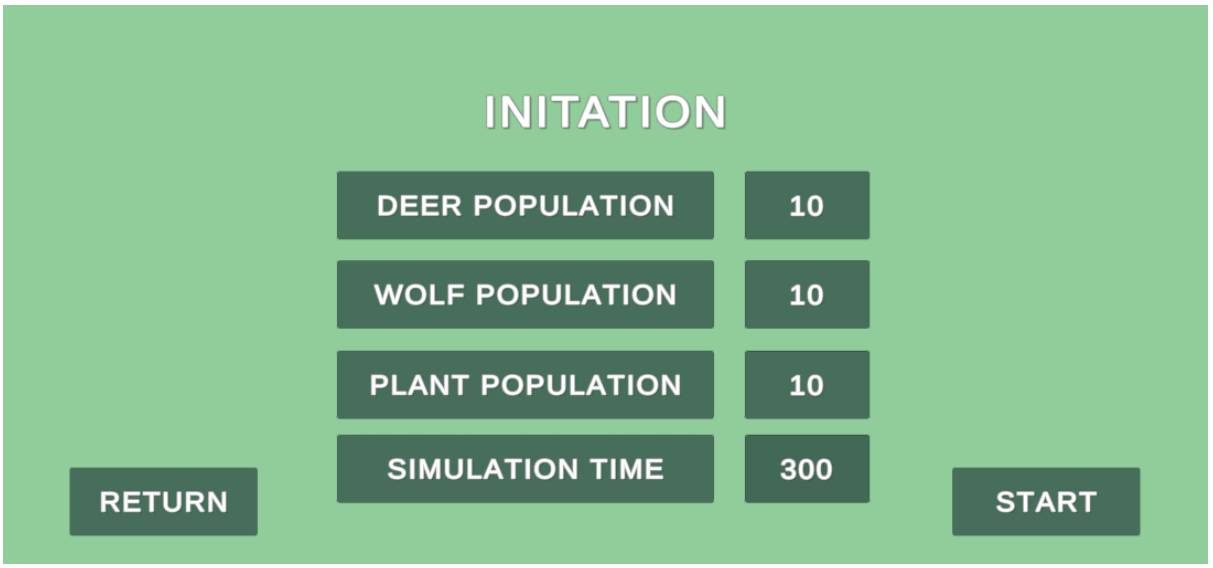
Scenario Testing Feedback

1) **Main Menu (Success Criteria 1):** The main menu was fully working as intended. There was no lag between swapping menus and pressing buttons. However I could have added animations and sound effects to make the transitions between menus more seamless.

Evidence

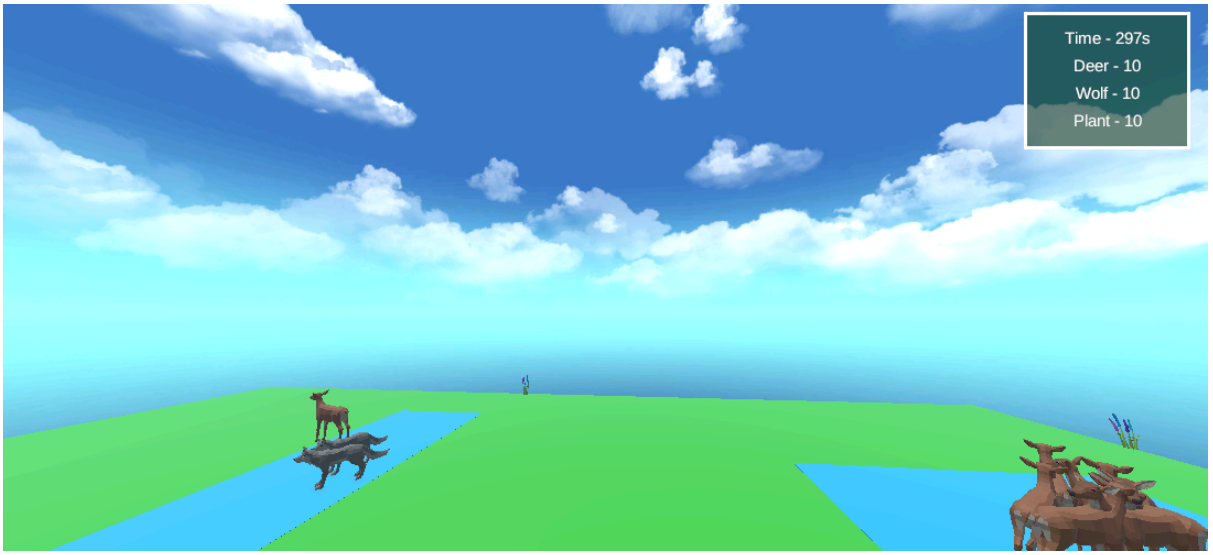
2) **Starting Game with Initiation Menu (Success Criteria 3):** Inputted values for simulation initiation for deer population, wolf population, plant population and time simulation length.

Evidence



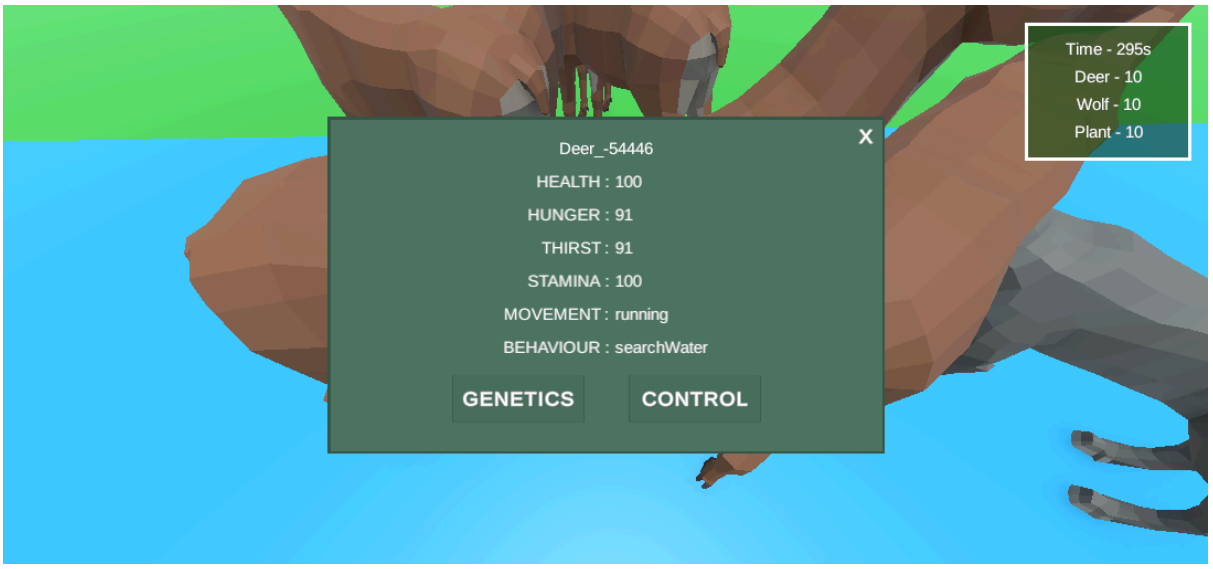
3) **Free Camera to Move Around (Success Criteria 2):** The free camera movement worked as intended. You could look in all directions and activate faster movement by pressing the shift key.

Evidence
6177_freeCam_PostDev1Test.mp4



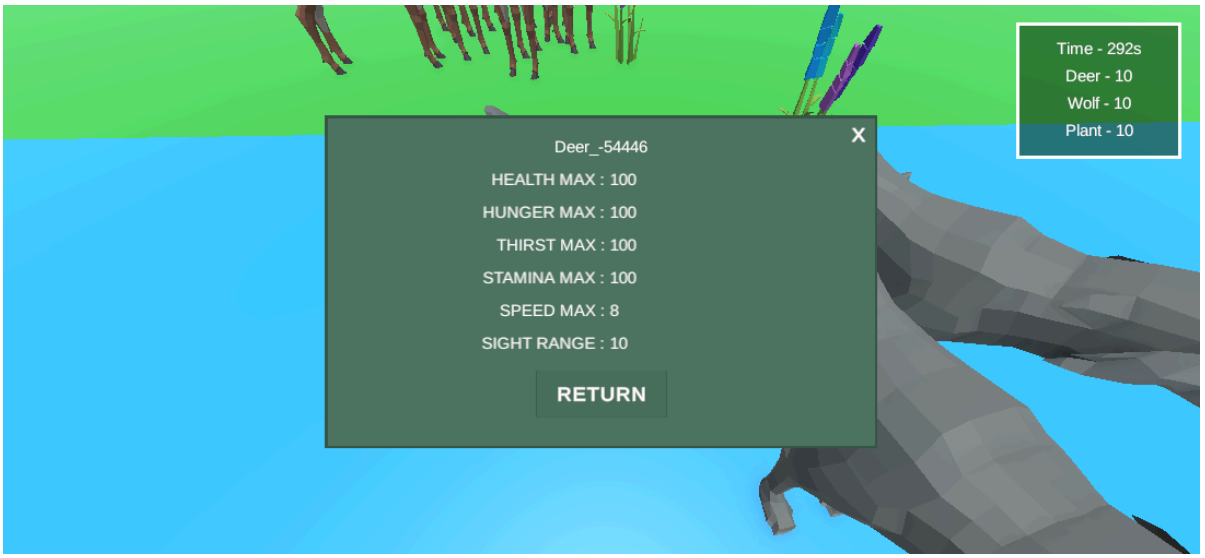
4) **Organism Attributes UI (Success Criteria 6):** The organism attributes UI worked fully as intended. When you clicked on an organism in range the UI would show and display all the appropriate information. The free cam movement would also be displayed so that you can use the cursor to navigate the menu.

Evidence



5) **Organism Genetics UI (Success Criteria 8):** The organism genetics UI works, when you click on the genetics button in the organism attributes UI it takes you to the genetics UI

Evidence



6) **Exit Out UI (Success Criteria 14):** Exiting out of the UI worked efficiently. When pressing the X button in both attributes and genetics UI.

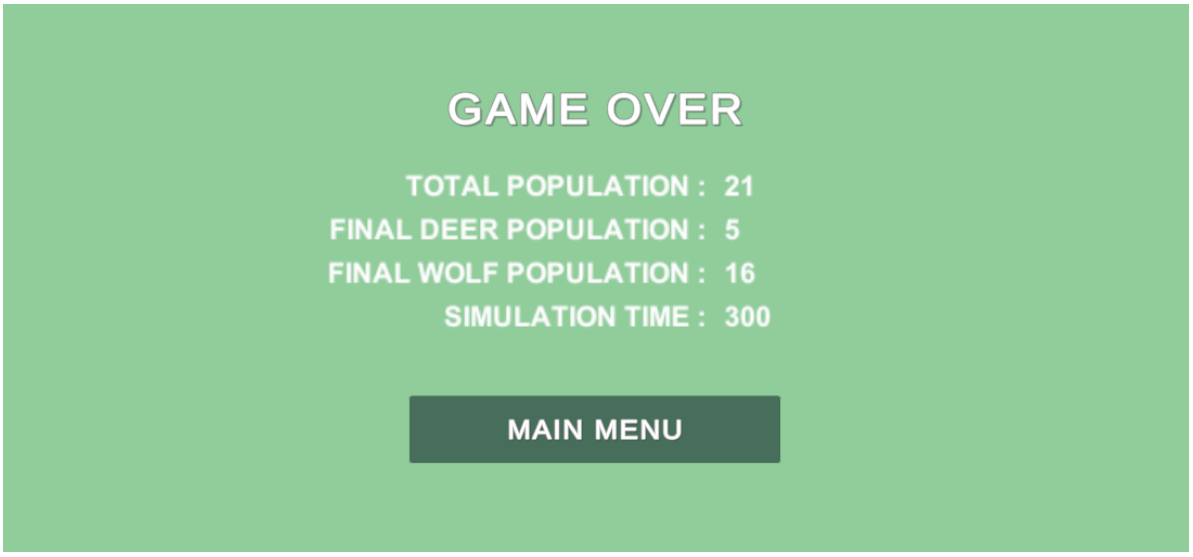
Evidence
6177_exitUI_postDev1Test.mp4

7) **Pausing Simulation (Success Criteria 12):** Pausing the simulation worked correctly, when pressing the escape button the pause menu would show with an unpause, controls and main menu button. When unpausing the simulation would carry on from where you left off when you paused it.

Evidence
6177_pauseMenu_postDev1Test.mp4

8) **Game Over Data (Success Criteria 13):** The game over screen works as intended and displays the appropriate data however it does not show the data graph.

Evidence



Overall Assessment

Overall the program has had most of the initial intended core features added however some features have not been added and not all of the success criteria has been fulfilled. Such as being able to use a skill tree and skill points was not added to the game despite being in the success criteria. The reason for this being after I had finished my design section and began working on the development I released that some of my intended functionality would be too complex or ambitious to implement within my time frame. Instead I decided it would be better to not implement these features and rather spend more time and development focus on other core functions of the game so that they can be more well rounded. If I was given more resources and time to develop my project I could have most likely implemented these features, however with the completed functionality that I did add still work very well.

The design of the game is very similar to my initial concepts in the design section such as the menus and user interfaces. The style of the simulation is all matching and suitable with a green theme for the menus with the same hexadecimal code colours being used.

Suggested Improvements

- One improvement that could have been added is adding some sort of mountains or surroundings around the walkable area for the organisms. This is since currently it is just a flat green platform that stops and has a void-like area beyond it. Adding some mountains or surroundings and a limit to the area the camera can move to will implement more immersion and realism into the simulation.
- Additionally, in hindsight I should have added a limit for the input variables since currently the user can enter a very high number such as 1000 which would most likely cause the program to crash due to a large amount of game objects being rendered concurrently.

4.1.2 Test Scenario 2 - User Inputted Movement Simulation

- Open the game
- Press start game
- Input starting conditions
    - Wolf starting population: 5
    - Deer starting population: 15
    - Plant starting population: 20
    - Time: 240 seconds
- Start the simulation
- Click on a deer to view their attributes
- Click on control to move around as the organism
- Eat a plant as the deer
- Wait for simulation to end
- View the data in the game over menu
- Return to main menu
- Exit game

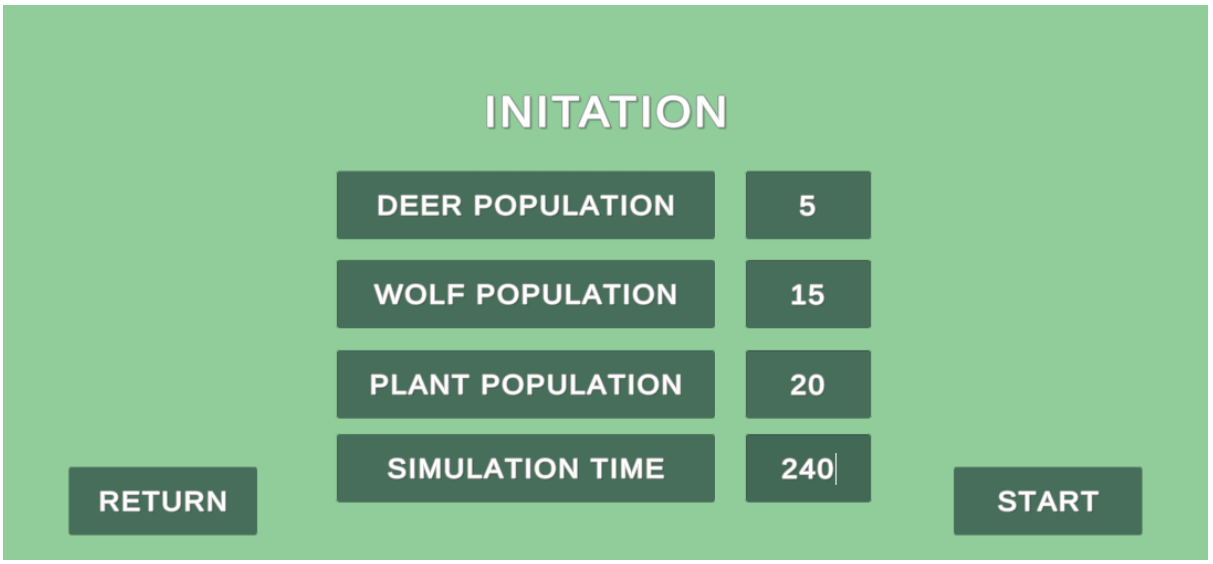
Scenario Testing Feedback

1) **Main Menu (Success Criteria 1):** The main menu was fully working as intended. There was no lag between swapping menus and pressing buttons. However I could have added animations and sound effects to make the transitions between menus more seamless.

Evidence

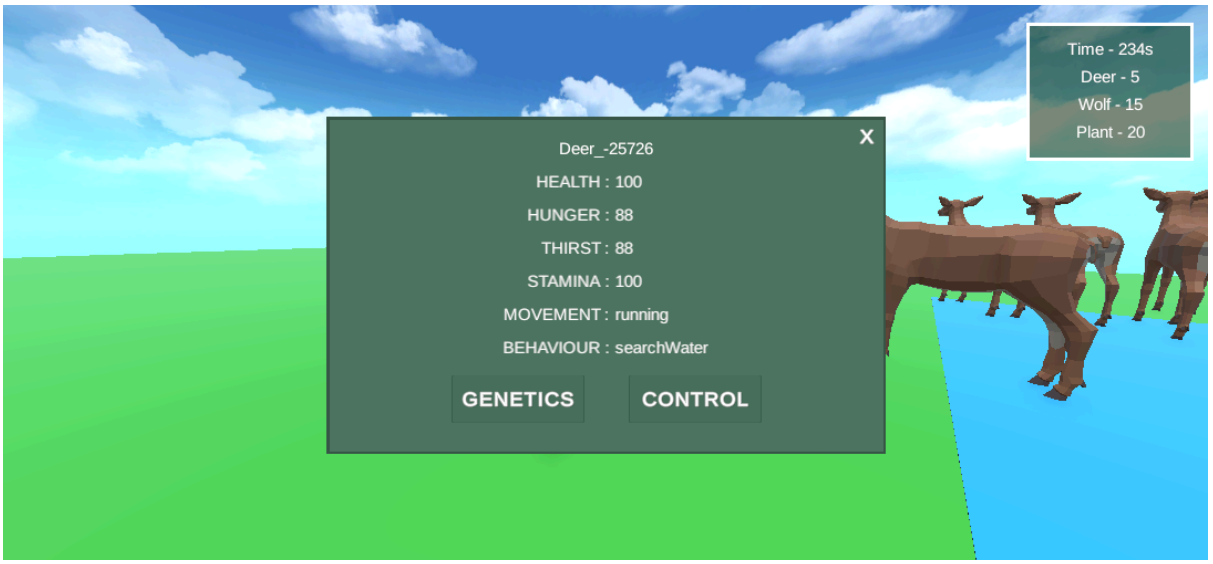
2) **Starting Game with Initiation Menu (Success Criteria 3):** Inputted values for simulation initiation for deer population, wolf population, plant population and time simulation length.

Evidence



3) **Organism Attributes UI (Success Criteria 6):** The organism attributes UI worked fully as intended. When you clicked on an organism in range the UI would show and display all the appropriate information. The free cam movement would also be displayed so that you can use the cursor to navigate the menu.

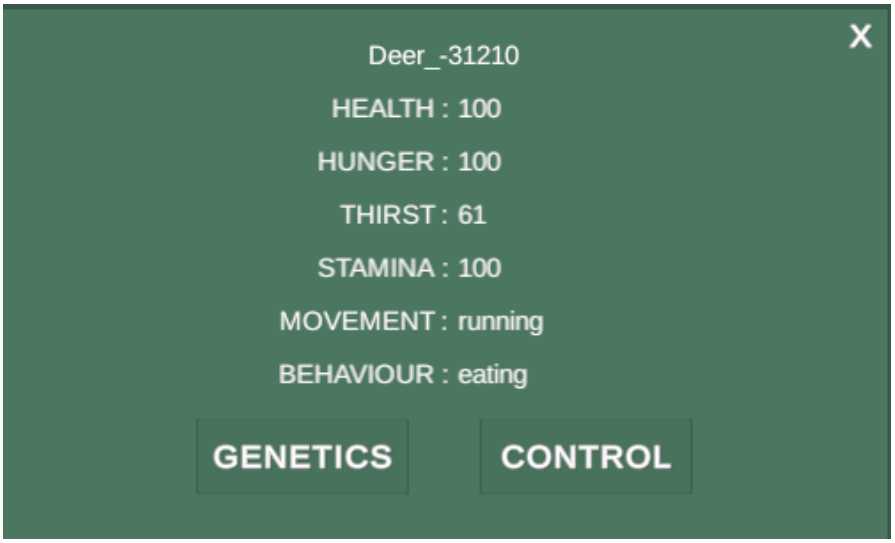
Evidence



4) **Controlling Organism (Success Criteria 4):** When clicking the control button it works as intended and changes the camera to the selected organism. You are then able to control the movement of this organism.

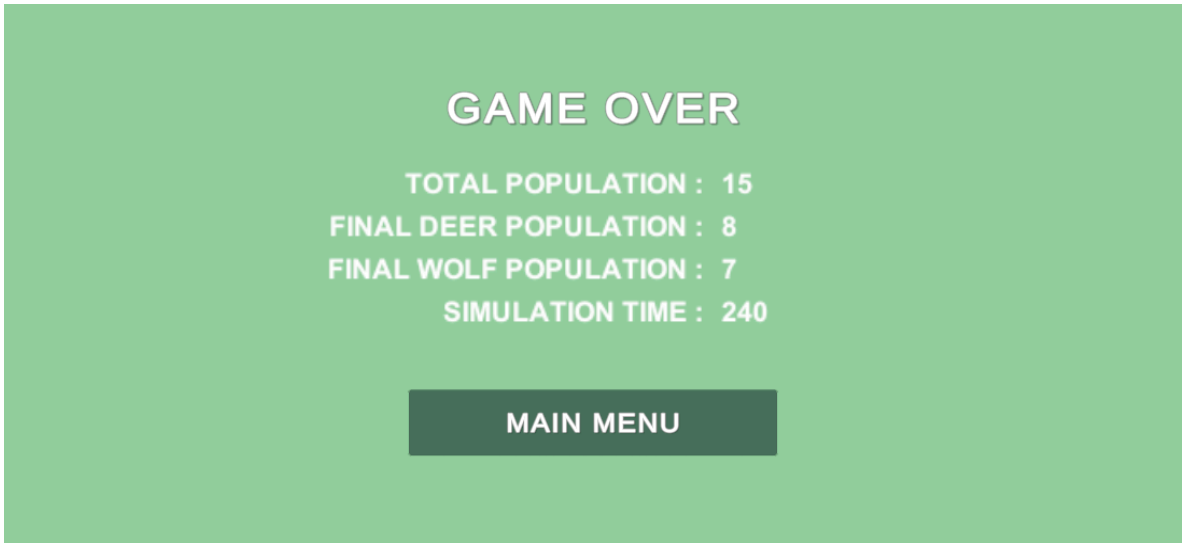
5) **Eat a plant as the deer (Success Criteria 7.1):** I have not added manually controlling the organism's behaviours. Instead the behaviours are automatically done through the behaviour conditional states.

Evidence



6) **Game Over Data (Success Criteria 13):** The game over screen works as intended and displays the appropriate data however it does not show the data graph.

Evidence



Overall Assessment

Overall the majority of the core functionality is functioning in this scenario test however there are some functions that are only partially completed. Such as when controlling an organism, not being able to actually control its behaviour with your own user inputs. Instead it just automatically changes based on conditional statements in the organism's OOP code. Another feature that is missing is the behaviour of an organism sleeping. This is since I decided early in my development to not implement a day night cycle and since this would be required for sleeping and they complimented each other, sleeping was not added.

Despite some features not being added or only partially completed, my client was still satisfied with my end product and felt as though it had completed the intended task of being able to teach people how natural selection and ecosystems work while also being fun and enjoyable to use at the same time. However they also felt as though adding all of the original features would have amplified the simulation and made it more complete. Such as adding evolution and skill trees.

Suggested Improvements

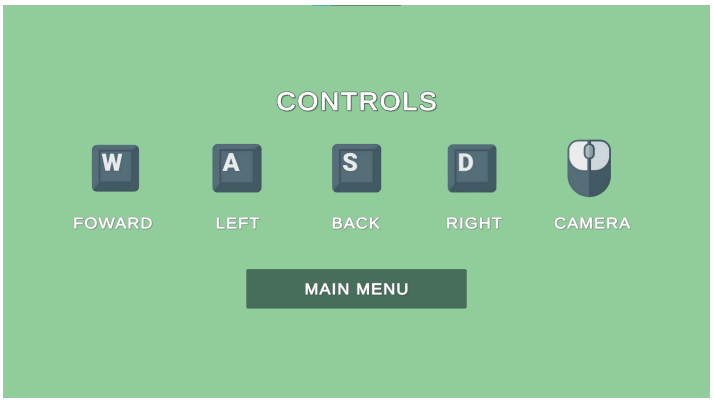
- Adding all my intended features from the design section would also make the game more complete if I had additional time and resources.
- An improvement that could have been added is allowing the user to control the behaviours of the organisms when they are in the control state. Currently they can only control the movement of the organism they are controlling and not the actual behaviours. Rather the organism is just automatically changed based on the behavioural state conditions and not user inputs.
- Update the controls menu to show how to control an organism and what button is used to open the interface since currently the controls menu is only displaying how to use the free camera.

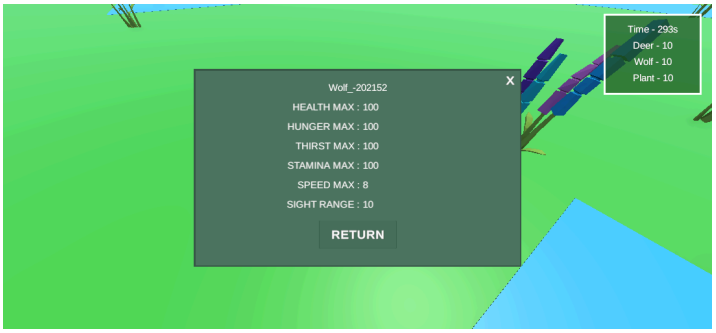
4.2 Usability Testing

User Interfaces & Menus

The user interfaces are easily accessible and clear to navigate. The name of the buttons are simple so that users can easily understand what they do, such as the “Controls” that takes you to the available controls. However, I have not updated the controls menu to show every single available input such as the controls that you can use in the settings menu.

Evidence





Resolution
The resolution of my game was fixed to 1920x1080 so if you were to play it on a different monitor or device with a different resolution you may encounter errors. In future development I could modify the game so you can change the resolution or so that it dynamically adjusts to whatever resolution you are playing on. This would allow a wider range of audience to play.

Colour Settings
I failed to add colour blindness settings for usability into my game. This is an issue since it will restrict certain users who may have difficulty playing especially since there are many green colours shown. Another issue is the brightness settings that were not added since if people are playing at different times of the day since at night the game could be too bright and they might need to lower the brightness and vice-versa.

Language Settings
I could have added different language settings that would allow a wider range of users to play. However this is not as large an issue since the user interfaces in the game are still simple and people could possibly still play without understanding english as there is not a large quantity of text used.

4.3 Success Criteria Evaluation

Index	Success Criteria / Requirement	Status	Justification	Testing Evidence
1.	Users will have access to a main menu when the game is opened	Completed	This requirement was fulfilled as when the game is opened the main menu is instantly shown and available	3.2.3 - Test No's 1.1, 1.2, 1.3, 1.4
2.	User will be able to access a free cam to view the environment	Completed	This requirement was fulfilled since when an environment is initiated the user is spawned in as a free camera controller which has the ability to fly around the environment and click on any organism to views that organisms statistics	3.3.3 - Test No's 1.1, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7
3.	Add the organisms: deer and wolves	Completed	This requirement was fulfilled since the organisms deer and wolves were both added since in the initiation menu you can select how many you wish to spawn and then they will be instantiated into the environment.	3.2.3 - Test No's 4.1, 4.2, 4.3, 4.4, 4.5 3.4.3 - Test No 1.1
4.	User will be able to select a certain organism and control their movements and behaviours	Partially completed	This requirement was partially completed since players are able to control organisms and their movements by clicking on an organism to view its statistics and then pressing the "Control" button; however the player is not able to manually control the behaviours, instead they are automatically completed based on the behaviours conditions in the organism scripts depending on the location of the organism	3.5.3 - Test No's 1.1, 2.1, 2.2, 2.3, 2.4, 2.5
5.	Sound effects will be implemented	Completed	This requirement was completed since I implemented sound effect for button presses in the main menu along with	3.2.3 - Test No's 1.1, 1.2, 1.3, 1.4
6.	Organism attributes (thirst, hunger and stamina) will be added into the game	Completed	This requirement was completed since the organisms, wolves and deer, both contain the attributes health, thirst, hunger and stamina	3.4.3 - Test No's 1.2, 1.3, 1.4, 1.5, 1.6
7.	Organism behaviours will be added into the game	Completed	This requirement was completed as organisms behaviours were added into the game along with conditions for the organisms behaviours to change	3.6.3 - Test No's 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6

7.1	Organisms will have the behavioural tasks of hunting and eating	Completed	This requirement was completed since the behavioural task of hunting and eating were both added including animations for them. Although deer did not have a behavioural task called hunt, instead it searched for plants since the deer cannot hunt wolves, only wolves can hunt deer.	3.6.3 - Test No's 1.1, 1.2, 2.1, 2.2
7.2	Organisms will have the behavioural tasks of drinking and searching for water	Completed	This requirement was completed as the organism deer and wolves both had the behavioural tasks of searching for water and drinking water along with the corresponding animations.	3.6.3 - Test No's 1.3, 1.4, 2.3, 2.4
7.3	Organisms will have the behavioural task of sleeping and being idle	Partially completed	This requirement is partially completed since the behavioural task of being idle was added where an organism will randomly move around with no destination but sleeping was not added due to complexity of creating a day and night cycle	-
7.4	Organisms will have the behavioural tasks of mating and searching for a mate	Completed	This requirement was completed since the organisms had the behavioural tasks of searching for a mate and mating along with the corresponding animations. When two organisms mated this instantiated a new organism which is created using genetic values and possible random mutation	3.6.3 - Test No's 1.5, 1.6, 2.5, 2.6
8.	Genetic values will be assigned to each individual organisms	Completed	This requirement was complete since when a new organisms is born it is instantiated with genetic values calculated from its two parents genetic values	3.7.3 - Test No's 1.1, 1.2, 1.3, 1.4, 1.5, 1.6
9.	Organisms will be able to evolve over generations	Incomplete	This requirement is incomplete as I decided to remove typically evolution from the game and rather have the organisms contain genetic values and mutations	-
10.	Offspring will be able to have a chance to inherit a mutational characteristic.	Completed	This requirement is completed since when a new organism is born there is a random chance it can have a mutational characteristic which will multiply one of its genetic values	3.7.3 - Test No's 2.1, 2.2, 2.3, 2.4, 2.5, 2.6
11.	Implementations of skill points and a skill tree	Incomplete	This requirement is incomplete as I decided to not to implement skill points and a skill tree since it would make the simulation too complex and would also affect the realism of the game since in real life organisms do not have skill points which can be used to upgrade them	-

12.	Users must be able to access a pause menu while playing the game	Completed	This requirement was fulfilled as I added a pause menu that can be used whenever in the environment state simply by pressing the “Escape” button	3.2.3 - Test No’s 3.1, 3.2, 3.3
13.	After a simulation has concluded, the end screen will show statistics	Completed	This requirement was fulfilled since when the simulation timer reached zero	3.8.3 - Test No 1.2
14.	The game will run with stability, optimisations and without major bugs	Completed	This requirement is completed as the game runs without crashing and no major game breaking bugs as seen in the post-development scenario testing	Test Scenarios 1 and 2

Changes to initial design

- **Player Controlled Movements (Success Criteria 4) -** This requirement was partially completed since players are able to control organisms and their movements by clicking on an organism to view its statistics and then pressing the “Control” button; however the player is not able to manually control the behaviours, instead they are automatically completed based on the behaviours conditions in the organism scripts depending on the attributes and location of the organism
- **Organism Behaviours (Success Criteria 7.3) -** I have completed all other success criteria under 7 including the behavioural tasks of hunting for wolves, searching for plants for deer, eating, drinking, searching for water, mating and searching for a mate. However I did not include the addition of sleeping and being idle. This is since these behavioural states both slowed down the pace of the simulation when I initially had them included. At times the simulation came to a complete stop when organisms were idle and when they were sleeping all the organisms would sleep at the same time, essentially pausing the game which was disengaging for the user.
- **Evolution (Success Criteria 9) -** I did not complete success criteria 9 since having organisms evolving over generations would include having to add new types of organisms and creating an organism's evolution path, neither of which were included in my initial design meaning it would take extensive time to implement and could have possibly disrupted my development schedule and deadlines.
- **Skill Tree (Success Criteria 11) -** This requirement is incomplete as I decided to not to implement skill points and a skill tree due to time constraints and since it would make the simulation too complex. It would also possibly affect the realism of the game since in real life organisms do not have skill points which can be used to upgrade them

4.4 Limitations

Currently my game isn't designed to be used on a very large scale and is instead intended to be an introduction for students or people interested in learning more about natural selection and how ecosystems work while also having fun at the same time. If I were to develop my game to be used on a larger scale and be more practical for research intensive purposes I would add more organisms and variables within the simulation. This could include things such as obstacles in the environment, implementing the intended feature of day and night cycles, having the attributes of the organisms be more realistic and similar to real life, etc.

Another limitation of my game is that the movement of the organisms could be too simple. Currently they are just using simple pathfinding which finds the nearest specific type of object that it is looking for then moves in a direct straight line to that object. The pathfinding doesn't anticipate objects or possible predators such as wolves that could be in the way. Instead it will just walk straight and then only avoid a wolf when it is close enough to activate the "runFromWolf" behaviour.

The end screen statistics menu is also a limitation in my program. I've only added the basic information of the final population data along with the simulation time length, fulfilling success criteria 13. This is limiting as it does not provide a detailed insight into how the populations varied over time throughout the simulation and what possible factors affected the populations.

Furthermore, another limitation could be not implementing evolution into the game. This is since evolution and organisms being able to evolve from one species to another would add more realism and complexity to the simulation; possibly making it more suited for research intensive purposes. Adding evolution would require there to be more species/organisms in the game along with them to have their own evolutionary path and conditions for evolution to take place. This would then fulfil success criteria 9.

Additionally, a limitation to my program is that I have not added a limitation to the number that can be entered into the organisms initiation data. This could cause issues as if the user were to add very large numbers it would instantiate a large number of game objects which could harm performance or even crash the game. This would then result in success criteria 14 failing.

A final limitation in my simulation would be that the control menu is too simple. Currently it is only displaying the controls needed to use the free camera to move around in the simulation but it has not included how to click on an organism to open the attributes and genetics menu or the controls of how to use player controlled movement on an organism.

4.5 Maintenance

One area of maintenance is the maintainability of the code. My program has been developed so that it is rather easily maintainable if someone other than me were to develop it further from this point on. This is due to the fact that I've created code that can be maintained through the use of comments and functions just to name a few. The use of comments allows someone who has never seen my code before to read the script and have an understanding of what the code's function and intention is without having to think too in depth. The use of functions allows code to be changed easily since rather than having to change code every single time through a script, instead you only have to modify it inside the function and then when it is called this modified function is run. I've also utilised well named variables and functions which allows the program to be more easily maintained as it can be understood easily.

Some changes I would make would be possible to split up some of the larger scripts into smaller, separate scripts. Such as the organism OOP scripts could have been split up so that the pathfinding functionalities are in a separate script.

One concern with maintainability is bugs and errors. Due to my iteration testing I've been able to identify, fix and then retest most of the bugs so that they can be resolved. However there are still some that I have not identified. One way to combat this could be to do a large scale beta test. This is where you would have a large sample of users who will all play the game on different devices, operating systems etc. They will then report any issues of bugs they encounter along with what inputs they had which resulted in this issue. This would then allow the developer to resolve any niche bugs or issues that certain users may encounter while using the program.

Something that could be added to the game is more sound effects since currently they are only very minimal. I could add background music as well as specific sound effects for different behaviours however I should be cautious to make sure the user is not overwhelmed by sound if there are for example, too many behaviours happening at a single time.

Furthermore, If I am to develop my simulation further it would include adding more functionality and complexity into it. For this to occur I should make sure that my code is as optimised as possible to ensure there is no lag or errors when playing. This will enhance the user experience and fulfil success criteria 14. One example of optimising the code would be to make it so that certain variables are updated less often, as well as adding a total organisms cap so that there are not too many objects. Additionally a render distance for game objects could be implemented so game objects at a further distance are rendered with less quality than ones which are closer.

4.6 Future Development

If I had more time or just decided to continue developing my program I could improve my current, existing features and also implement new features, either from my initial design or just brand new features. There is still a lot of further development and implementation that can be added into my simulation if I had more time and resources to spend working on it. However since my project has limited time to be completed I have had to have strict deadlines and time management of when certain sections and aspects of the simulation are to be completed by. This has resulted in some of my success criterias not being met since I have either not fully completed or implemented them.

One feature that I would improve with further development would be the organism pathfinding. I previously talked about this in the limitations section about how the organism movement is too simple. This is since currently the organism is just travelling in a fixed line towards its destination. Instead I could improve this by making it so that the organism avoids certain objects when travelling to its destination. Such as pre-anticipating a wolf rather than just waiting until you are in range to run away from one. Another example of smart pathfinding would be for an organism who is searching for a mate to not go towards a mate who is in closer proximity to another organism who is searching for a mate since it is likely that those two will mate and have a mating cooldown before the organism travels to them.

Secondly, another current feature that I could improve with further development is the player controlled organism system. I could make it so that the user's inputs can decide the organisms behaviours rather than them being decided automatically using behavioural conditions. I could have also improved the end screen by adding a graph of the statistics throughout the time of the simulation.

A feature that I have not designed already is a replay system for the simulation that would track all the stats of the organisms and their movements throughout. The user would then be able to watch an exact replay of the simulation.

Additional accessibility features would be added to make the simulation appeal to a larger audience. These features could include ones I have previously mentioned in the usability testing section such as support for different languages, support for colour blind modes and support for different resolutions to be rendered. Also,

If I were to further develop the simulation program, I would improve the environment by adding models for the surroundings such as possibly mountains, trees, grass etc. This will create a more atmospheric environment which will enhance the user experience.

Furthermore, In future development I would also implement my success criterias that failed and went uncompleted. This includes the implementation of organism evolution, success criteria 9, so that when a new organism is born there is a chance that it will be able to evolve into a new species if certain conditions are met at the time of its birth. In order to implement evolution I could set up certain pathways that organisms can evolve through.

The other success criteria that failed which I could implement into future development is the skill tree and skill points, success criteria 11. This could be used to upgrade and change specific organisms' statistics and attributes. Finally, I should add statistics of the organisms populations throughout the simulation time since this will provide a more accurate and detailed representation of how natural selection and ecosystem work in real time.

If further development of my program was to occur I could possibly publicly release it for the public to use. I could release it as either a paid or free version, proprietary or even open source. Releasing it to the public would help bring exposure to it and possibly even help more people learn about how ecosystems and natural selection work.

In conclusion, the stakeholders, my client and I are all very happy and content with the completion of this project. The majority of success criteria were met with only some being partially completed or not completed at all however this did not drastically affect the result of the problem that the simulation was attempting to solve.That being an introduction into educating people into how natural selection and ecosystems work. I believe I have solved this problem and that my program is a suitable introduction into educated people on natural selection and ecosystems.

References

Analysis

1.3 Research

Images from Equilinox game research obtained from here

- https://www.equilinox.com/

VAV Ecosystem Simulator is from this video

- https://www.youtube.com/watch?v=Bfcg4tS8hpw

Design

2.2 - Menus

Font used in user interface designs

- https://www.dafont.com/the-bold-font.font

2.4 - Free Camera View

How to make setup a spectator camera

- https://www.youtube.com/watch?v=rxa4N4z65pg

2.9 - Organism Pathfinding

A* Pathfinding Algorithm

- https://www.geeksforgeeks.org/a-search-algorithm

Development

3.2 - Menus

How to get text from input fields

- https://www.youtube.com/watch?v=guelZvubWFY

How to Freeze Game

- https://discussions.unity.com/t/how-to-freeze-and-unfreeze-my-game/311091

Computer key images for control menu

- https://www.techonthenet.com/clipart/keyboard/index.php

3.3 - Free Camera

Setup camera in unity

- https://www.youtube.com/watch?v=rxa4N4z65pg

3.4 - Organism & User Interface

Animal models used

- https://poly.pizza/bundle/Animated-Animal-Pack-ILAPXeUYiS

Environment models used

- https://poly.pizza/bundle/Ultimate-Stylized-Nature-Pack-zylyYd9yGr

How to enable and disable game objects

- https://discussions.unity.com/t/how-to-enable-and-disable-a-canvas-window-by-scripting/117242/2

How to enable and disable components using a script

- https://discussions.unity.com/t/how-to-enable-and-disable-components-through-a-script/254116/4

3.5 - Player Controlled Movement

Find Distance between two objects

- https://discussions.unity.com/t/distance-between-two-objects/404478/2

3.6 - Organism Behaviours & Pathfinding

How to setup 3D pathfinding

- https://www.youtube.com/watch?v=mJu-zdZ9dyE

How to destroy an object

- https://discussions.unity.com/t/destroy-gameobject/861745

How to make an object move in the opposite direction of another object

- https://discussions.unity.com/t/move-object-in-opposite-direction-of-another/212974

Final Client Sign Off :