Evolution Simulation

Contents:

Analysis:

Program Overview:

I plan to write a program to help teach evolution in schools to biology students from year 7 to GCSE level. Evolution is a hard and complex topic to understand. Therefore, I am going to create an evolution simulator that will model how organisms reproduce in order to allow the user to observe how organisms adapt to their surrounding environment through random mutation. These mutations will vary the features of the creature, which will increase the creature's probability of reproducing and so passing it's genes on. I also plan to implement a graphing system in order to plot the data collected from the simulation and to be able to show the user details on the simulation. Users will also be able to log back into their account so they can view their saved simulations that they can enter and pause at any time.

What is the problem I am solving?

I am going to create an evolution simulator to help teach evolution in schools for pre-GCSE Biology. Currently, teachers rely on drawings and sketches in textbooks and powerpoints to teach evolution, however this makes the lessons very conceptual and uninteresting for young pupils. Instead students prefer more visually stimulating and engaging ways of learning.

Existing Solutions:

Teachers currently must draw diagrams and explanations as well as note taking from the textbook, and even then, many students are left not understanding many of the basic concepts of evolution. Outside of education there are various evolution simulators that allow users to track populations and other features through graph plotting, which is essential as it provides users with a deeper understanding of the progressions of characteristics over time.

Existing solutions have evolution of many features including food types consumed, size, speed, sensory ranges for food detection. However, they do apply the genetic approach to the way that the organisms reproduce, which is a key component of my program.

One major downfall of other simulators is the lack of control that they give the user with the environment and therefore are rather limited in their usage for teaching. However this could easily be changed and in conjunction with graphing the populations and their characteristics over time, would permit users to observe how changes in the environment that the organisms inhabit would cause changes to the characteristics observed in the organisms due to evolution. Existing solutions also do not allow users to save their simulations to return to them later, something which would be crucial in a teaching environment.

End User:

My target end user is biology teachers and people interested in learning about evolution. I plan to make my simulation visually appealing in order to capture the interest of younger students while at the same time maintain a very educational focus for higher level users with the graph plotting. My program must show the progression within organisms as time goes on and slight mutations become more prevalent in a species over many generations, while at the same time providing a fun interactive task for students and non-education focused users. The program must also be user friendly and visually stimulating as otherwise it will seem boring and students will quickly lose interest. If students find it fun to play around with, they will also be more likely to use the simulation to answer their own questions outside of the classroom.

Interview:

In order to ensure that my program is useful for my target end user I went to the school's Biology department and asked one of the teachers, Mrs Cuss, the following questions (in bold) and got the following responses (not in bold).

- **Would graph plotting help to show how the populations of organisms change and adapt over time help to demonstrate how the organisms evolve?**

- That would be useful, especially for slightly older students and to show how evolution does not just happen within a single organism and how it is a process that happens over time and many generations.

- **Would you like to be able to save the simulation in order to return to it later?**

- Yes, this would be great.

- **Would you like to see various environments and if so, which do you believe would be the most interesting or useful in order to show species differentiation?**

- That would be ideal if possible. This would particularly to show species evolve to and adapt over multiple generations to their surroundings. We would like to have a single environment per map but allow the students to change certain parameters, such as food availability, number of organisms, etc, as it would make them more intrigued by the simulation.

- **Would you like to see organisms learn about new food sources and go to the best sources?**

- That would add an extra level of realism to the simulation, however this wouldn't really add to the teaching part of the projects as much as it goes beyond the pre-GCSE and GCSE syllabus.

- **Would you like to be able to set the number of starting organisms and quantity of food to make the simulation competitive and challenging?**

- Yes, this again would allow the students to interact with the simulation more. This would also help students to understand how species adapt to their surroundings according to the environmental pressures in the system.

- **What other functionalities would you find useful for showing the process of evolution?**

- I think it would be especially useful if teachers were able to save simulation settings and then students could access and use those same settings and then run their own simulations with those same settings.

User Requirements:

Using the answers from my interview, I believe I will create a single map with a single environment on it, however I will allow the users to change the settings to the environment in order to give teachers and users more control over simulation and in the teachers case, showcase particular results such as increase in fat for colder climates over time.

Broad Objectives:

1	Have multiple different environments to show how through evolution the creatures will tend to specialise for different environments.	The user must be able to see the difference between the multiple areas and my program must be able to show how the different areas have creatures specialised to them.
2	Creatures will have to collect food and water to survive.	Creatures must move around and try to collect the things that they require to survive.
3	For creatures to reproduce they must find a mate (a compatible creature).	Creatures will not be able to pass on their "genes" and reproduce unless they have contact with a fellow member of their species.
4	Users will be able to have an account.	Users must be able to sign up and log in.
5	Allow users to pause and save one of their simulations and to continue them at will.	Users must be able to reload their most recently saved simulation.
6	2D graphics to allow users to view the simulation.	The simulations must show how the creatures move around and behave.
7	Graphing of data to see how the simulations attributes change over time (e.g. a family tree).	The program will create graphs to show the relationship between organisms and their properties over time. This must be easy to use.

Programming Language:

I have decided to write my program using python and the extensions pygame and pygame menu due to their ease of use and the way that they create good looking and user-friendly user interfaces. Pygame menu will allow me to create menus that would be easy to integrate in python. Python also allows for easy and simple object oriented programming which will help me to structure my code into smaller blocks and objectives.

Possible Extensions:

There are various extensions I could implement in my program if I have extra time at the end of my project. The following are some possible extensions:

My interviewee also suggested that I implement a mode where teachers can save simulation settings and their students can run their own simulation off the settings given to them by their teachers. This would likely be time consuming so could potentially be implemented as an extension.

I also think it is essential for the users to be able to save their simulation as my interviewee seemed particularly keen on this part of my project. They also felt that organisms being able to memorize locations and returning to them would make my simulation more representative of real life organisms and make the simulation more realistic however that it would not add to the teaching side of the simulation. Due to the difficulty of this task I plan to do this as an extension if I have time to do so.

I could also use hash and salt functions for additional security in users' personal data storage (passwords).

Design:

What is the aim of my project?

The aim of my project is to create a program that simulates the evolution of organisms for the school biology department as well as people outside of education who are interested in the process. Throughout the simulation my program will collect data on the simulation being run and will allow the user to plot graphs of the data to see how the properties of the organisms within the simulation vary over many generations. The simulation will also provide an interface for the simulation where users will be able to observe the simulation as it happens, with all of the organisms on the map moving around and going about their tasks (e.g. collecting of food and water, reproduction, etc.).

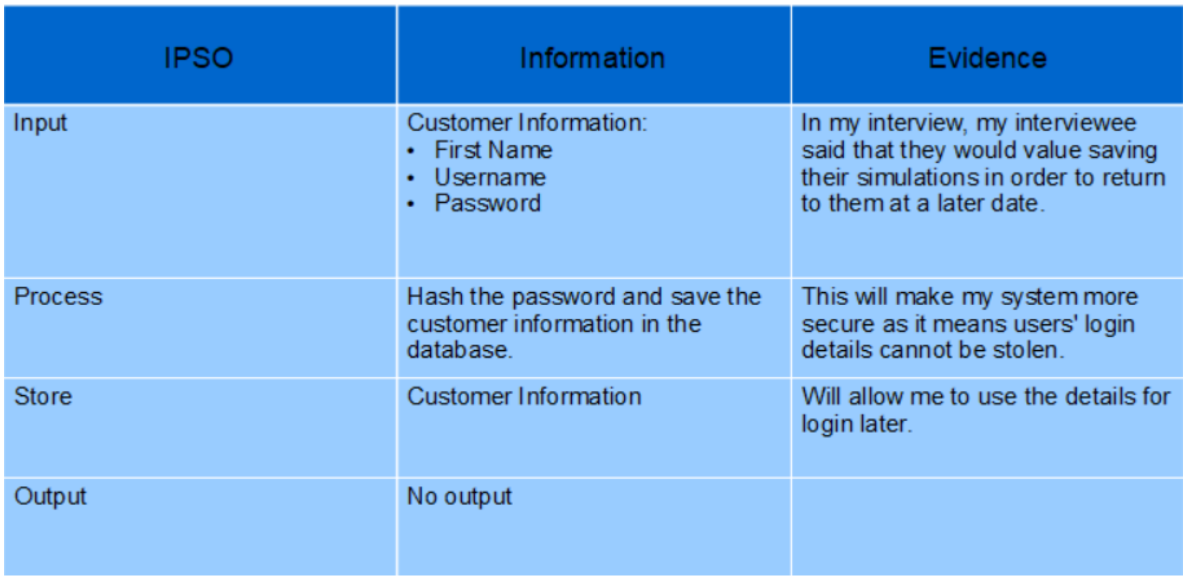
Users will be able to generate their own map with either a preset values or with their own inputted values, that will determine the properties of the simulation. This will include the number of starting organisms, the number of food sources, the number of water sources and map size. These properties will cause changes within how the organisms interact with each other and the system as a whole and therefore, the users will be able to observe how varying these starting values will affect the populations over time.

The simulation will also at a deeper level show the ways in which the organisms adapt through the colour of the organisms themselves, which I will use to represent the chromosome of each organism. As time goes on users will be able to note the specialisation as organisms start to distinguish themselves into colour groups. (As certain types of behaviours and genes become more common).

In this project I will be taking inspiration from the book Hidden Order by John.H.Holland which describes the structure of the chromosome I will be using and how it could be used for building complex adaptive systems.

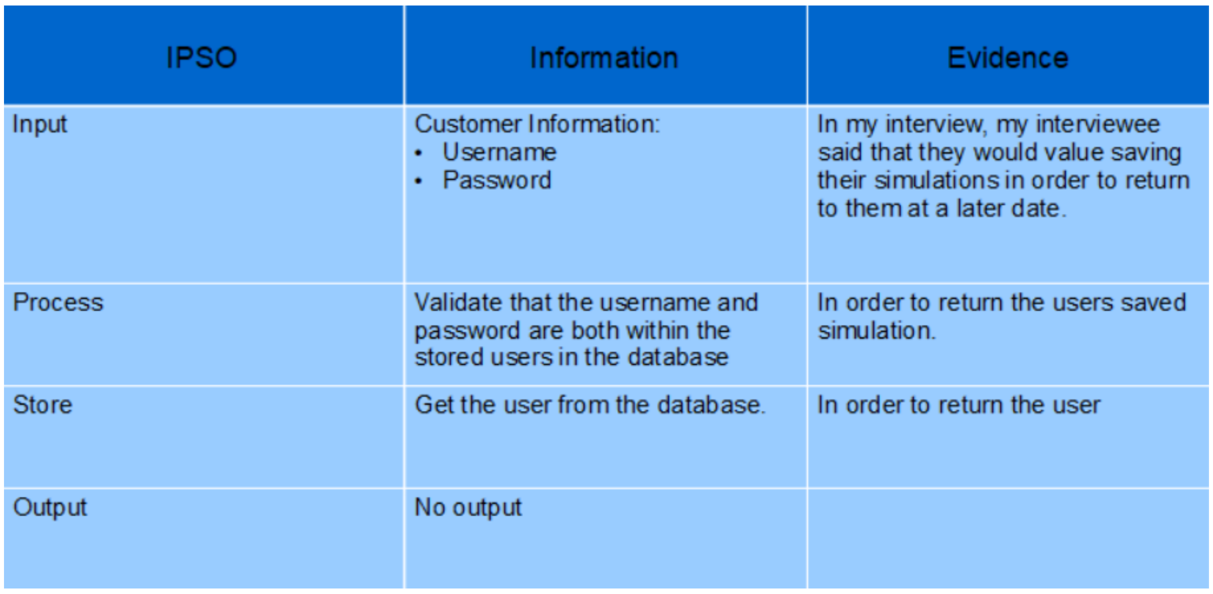
IPSO Tables:

The following IPSO tables show an overview of the requirements for each of the separate processes within my program.



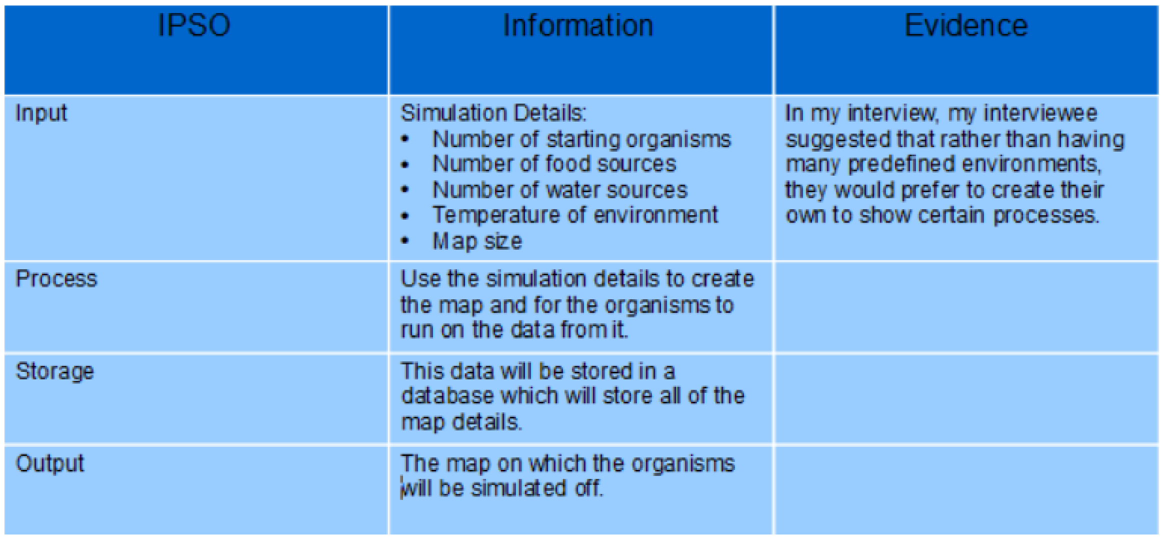
IPSO	Information	Evidence
Input	Customer Information: • First Name • Username • Password	In my interview, my interviewee said that they would value saving their simulations in order to return to them at a later date.
Process	Hash the password and save the customer information in the database.	This will make my system more secure as it means users' login details cannot be stolen.
Store	Customer Information	Will allow me to use the details for login later.
Output	No output	

IPSO table for user sign-up



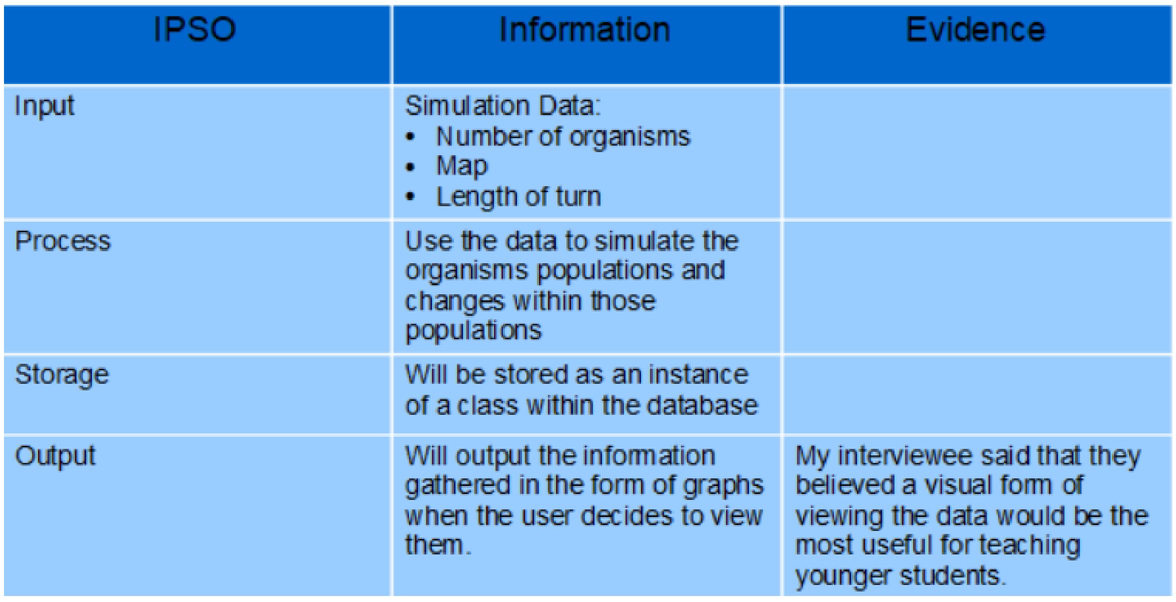
IPSO	Information	Evidence
Input	Customer Information: • Username • Password	In my interview, my interviewee said that they would value saving their simulations in order to return to them at a later date.
Process	Validate that the username and password are both within the stored users in the database	In order to return the users saved simulation.
Store	Get the user from the database.	In order to return the user
Output	No output	

IPSO table for user login



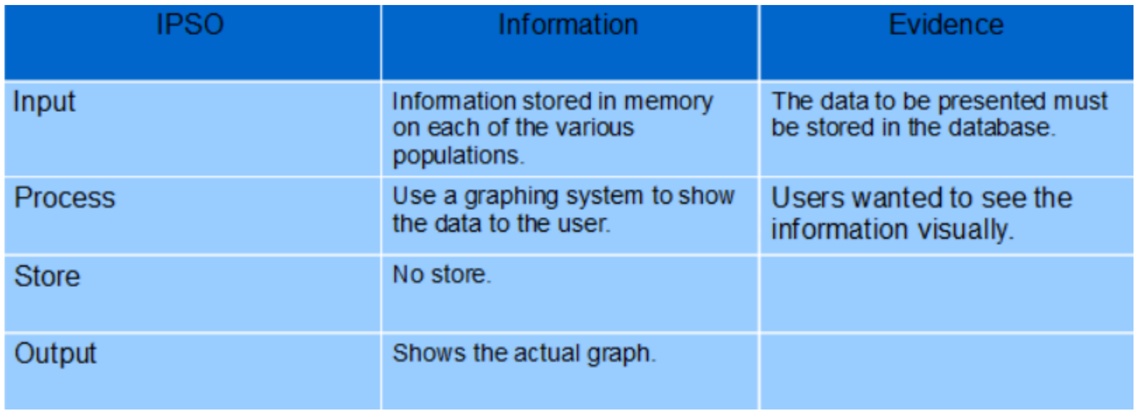
IPSO	Information	Evidence
Input	Simulation Details: • Number of starting organisms • Number of food sources • Number of water sources • Temperature of environment • Map size	In my interview, my interviewee suggested that rather than having many predefined environments, they would prefer to create their own to show certain processes.
Process	Use the simulation details to create the map and for the organisms to run on the data from it.	
Storage	This data will be stored in a database which will store all of the map details.	
Output	The map on which the organisms will be simulated off.	

IPSO table for map generation



IPSO	Information	Evidence
Input	Simulation Data: • Number of organisms • Map • Length of turn	
Process	Use the data to simulate the organisms populations and changes within those populations	
Storage	Will be stored as an instance of a class within the database	
Output	Will output the information gathered in the form of graphs when the user decides to view them.	My interviewee said that they believed a visual form of viewing the data would be the most useful for teaching younger students.

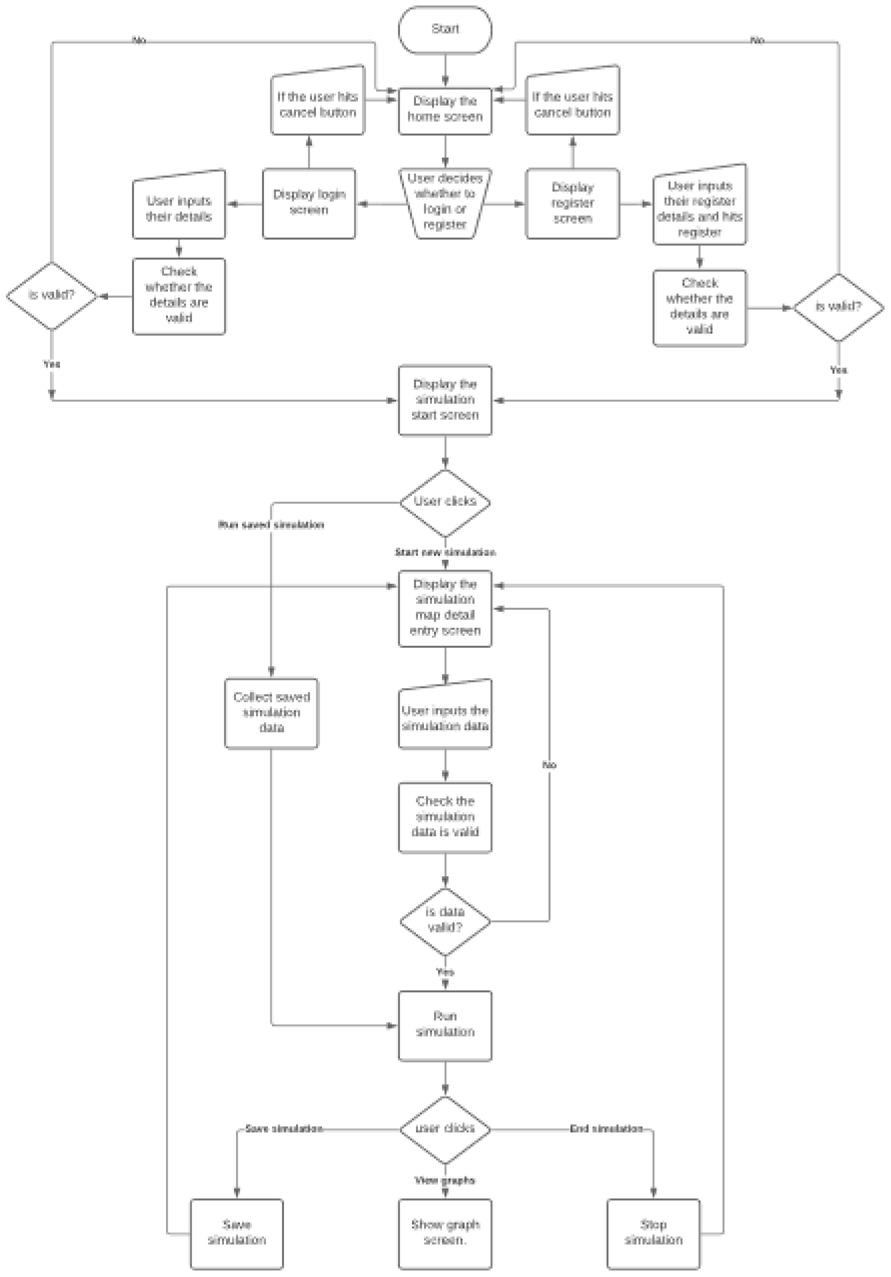
IPSO table for simulating populations and collecting simulation data



IPSO	Information	Evidence
Input	Information stored in memory on each of the various populations.	The data to be presented must be stored in the database.
Process	Use a graphing system to show the data to the user.	Users wanted to see the information visually.
Store	No store.	
Output	Shows the actual graph.	

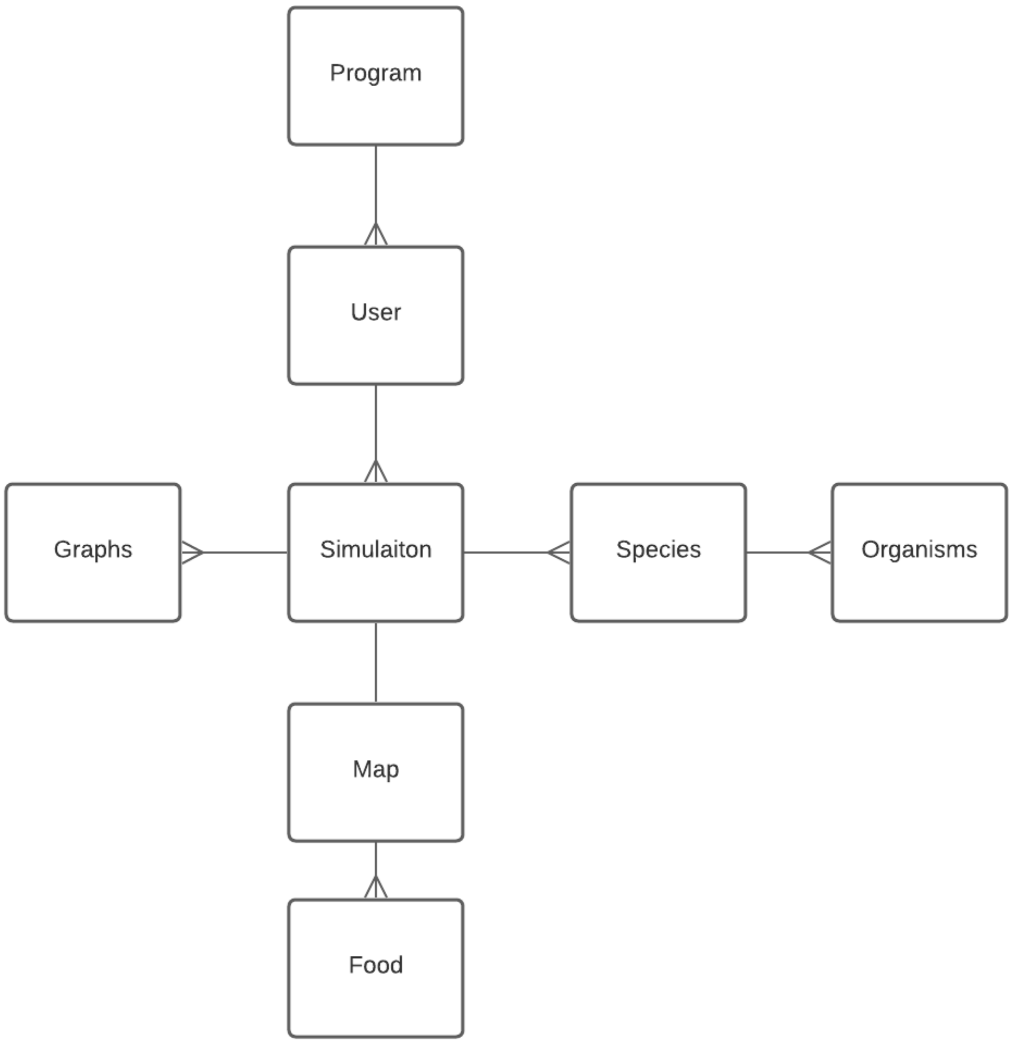
IPSO table for the graphing of simulation data

System Flowchart:


Start
Display the home screen
If the user hits cancel button
If the user hits cancel button
User decides whether to login or register
Display login screen
Display register screen
User inputs their details
User inputs their register details and hits register
Check whether the details are valid
Check whether the details are valid
is valid?
is valid?
No
No
Yes
Yes
Display the simulation start screen
User clicks
Run saved simulation
Start new simulation
Display the simulation map detail entry screen
Collect saved simulation data
User inputs the simulation data
Check the simulation data is valid
is data valid?
No
Yes
Run simulation
user clicks
Save simulation
View graphs
End simulation
Save simulation
Show graph screen.
Stop simulation

This system flowchart provides an insight into the way that the various pages are linked to each other and how users would navigate from one section of my program to the next. The flowchart also shows the conditions for moving between these screens (not all due to restraints with the number of objects I could have on a single flowchart).

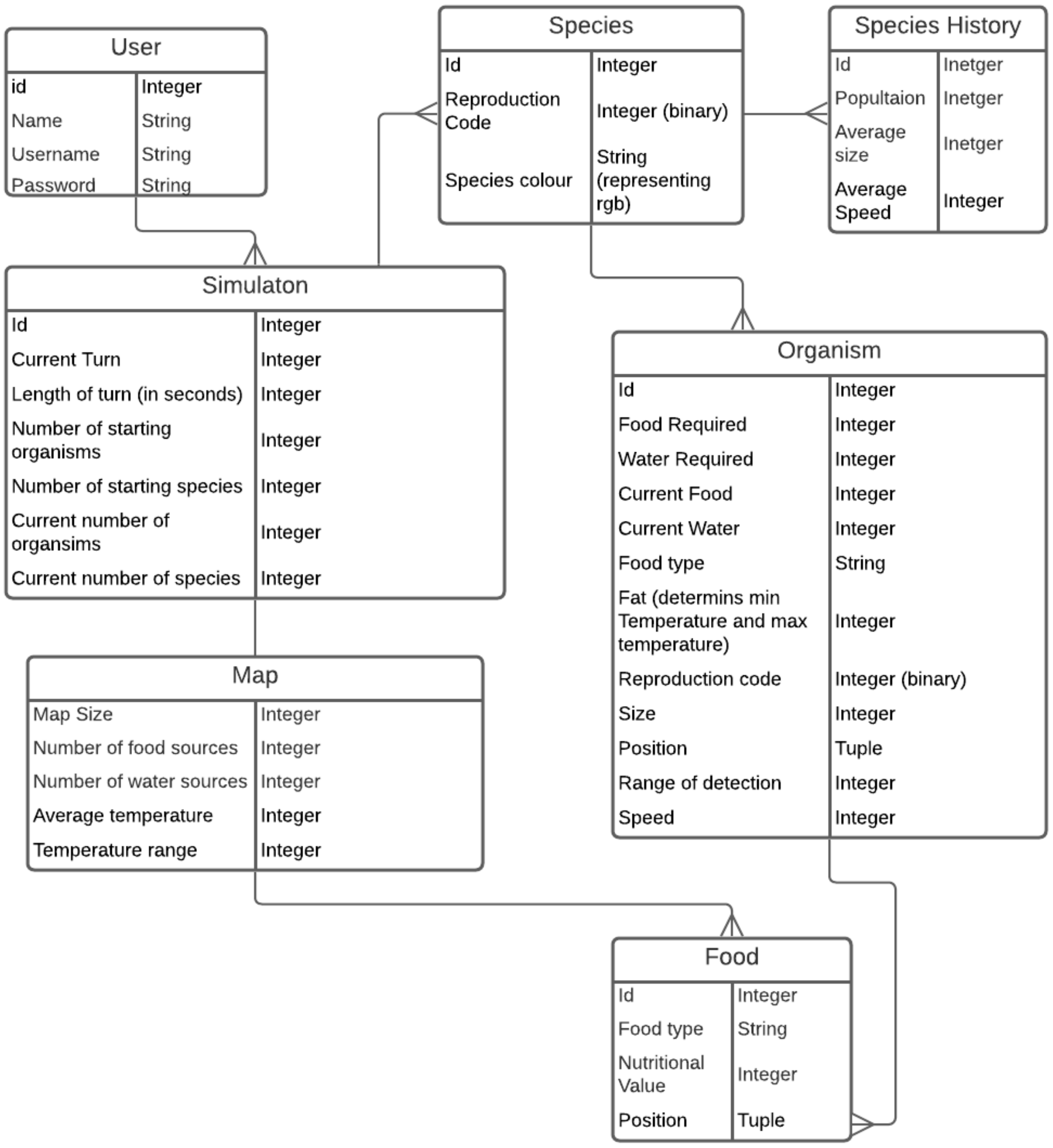
UML Diagram:



The entity relationship diagram above shows the relationships between the various objects within the program. The program will have many users. Each user will in turn have up to three saved simulations which belong to that user but may not be run simultaneously. Each of these simulations will in turn each have a single map which is either randomised by the simulation or specified by the user. These maps will also each have their own food sources. Each simulation will also have various graphs which will be plotted using the data collected by the simulation. Each simulation will also have various species which will vary between themselves.

New species may be created or destroyed according to the simulation (a species will be deleted if their population becomes 0 and a new species will be created when an organism can no longer reproduce with any other species). Each species will then in turn have one or more organisms that belong to that species. Organisms within a species may have slight changes in their characteristics between organisms.

Database Structure:


User
id | Integer
Name | String
Username | String
Password | String
Species
Id | Integer
Reproduction Code | Integer (binary)
Species colour | String (representing rgb)
Species History
Id | Inetger
Popultaion | Inetger
Average size | Inetger
Average Speed | Integer
Simulaton
Id | Integer
Current Turn | Integer
Length of turn (in seconds) | Integer
Number of starting organisms | Integer
Number of starting species | Integer
Current number of organsims | Integer
Current number of species | Integer
Map
Map Size | Integer
Number of food sources | Integer
Number of water sources | Integer
Average temperature | Integer
Temperature range | Integer
Organism
Id | Integer
Food Required | Integer
Water Required | Integer
Current Food | Integer
Current Water | Integer
Food type | String
Fat (determins min Temperature and max temperature) | Integer
Reproduction code | Integer (binary)
Size | Integer
Position | Tuple
Range of detection | Integer
Speed | Integer
Food
Id | Integer
Food type | String
Nutritional Value | Integer
Position | Tuple

The entity relationship diagram above represents the database structure of my program. Each user will have a set of login details within the database so that they are able to log into their account. Each user will then have up to three simulations which must each have an id and some program details specified by the user such as the turn length. The simulation table will also store the details for graph plotting so that when a user logs back into that simulation they are able to view all the previous data collected by that simulation.

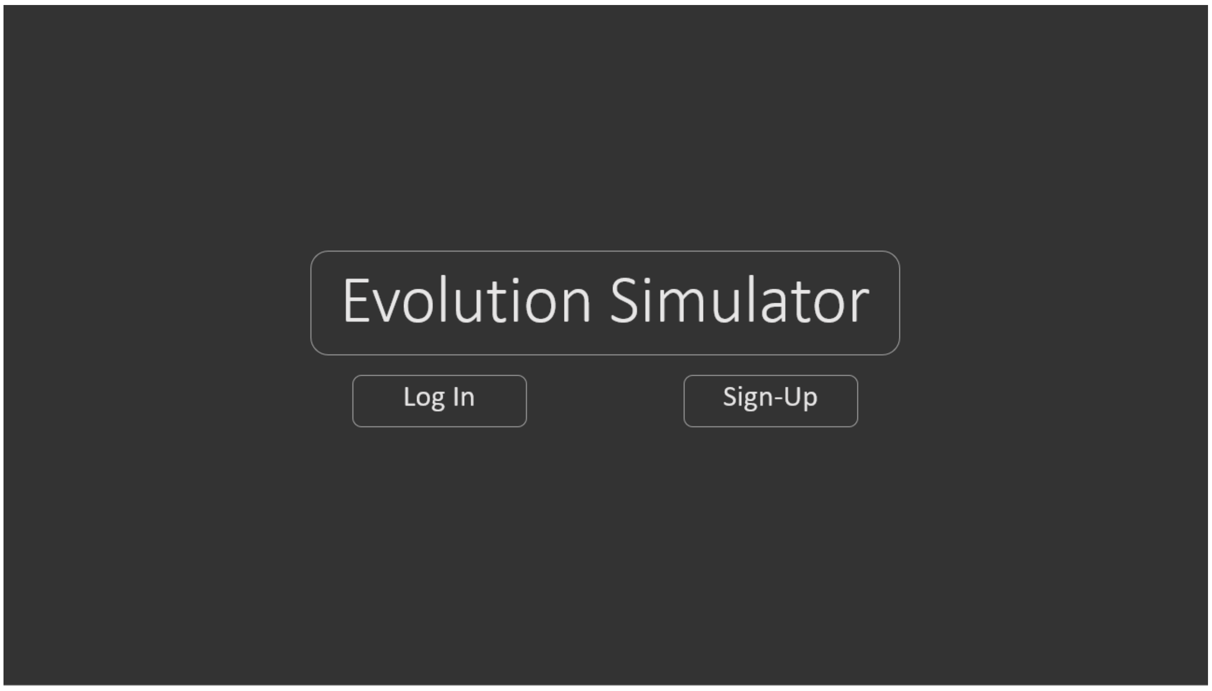
The simulation will also have a map which will store all the map's details such as the size of the map etc. This will be specific to each simulation. Each map will in turn also have various specific food sources which will provide more or less food when consumed. These will be defined when the user inputs the number of food sources in the simulation and the size of each type will be either randomized or user defined.

The simulation will also have many species, which will hold the common species details such as the reproduction code that must be matched (this will be explained later on) in order for an organism from the species to reproduce with another organism from that species. Each species will in turn have many organisms (unless they go extinct or are almost extinct). The species will each also have a species history table outlining their population and details over time.

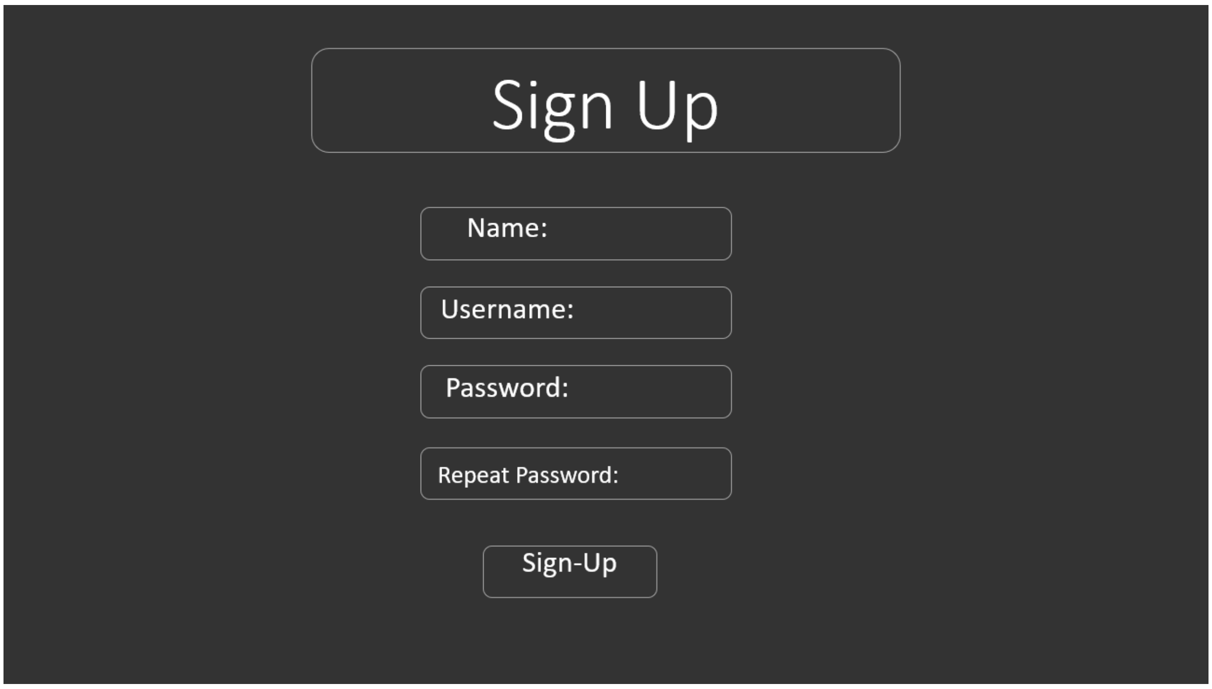
These organisms will then each have their own storage of their current survival details. These will include the amount of food they have collected and how much they require in order to survive the turn. They will also have other details such as the temperatures that that specific organism can survive etc.

User Interface Design:

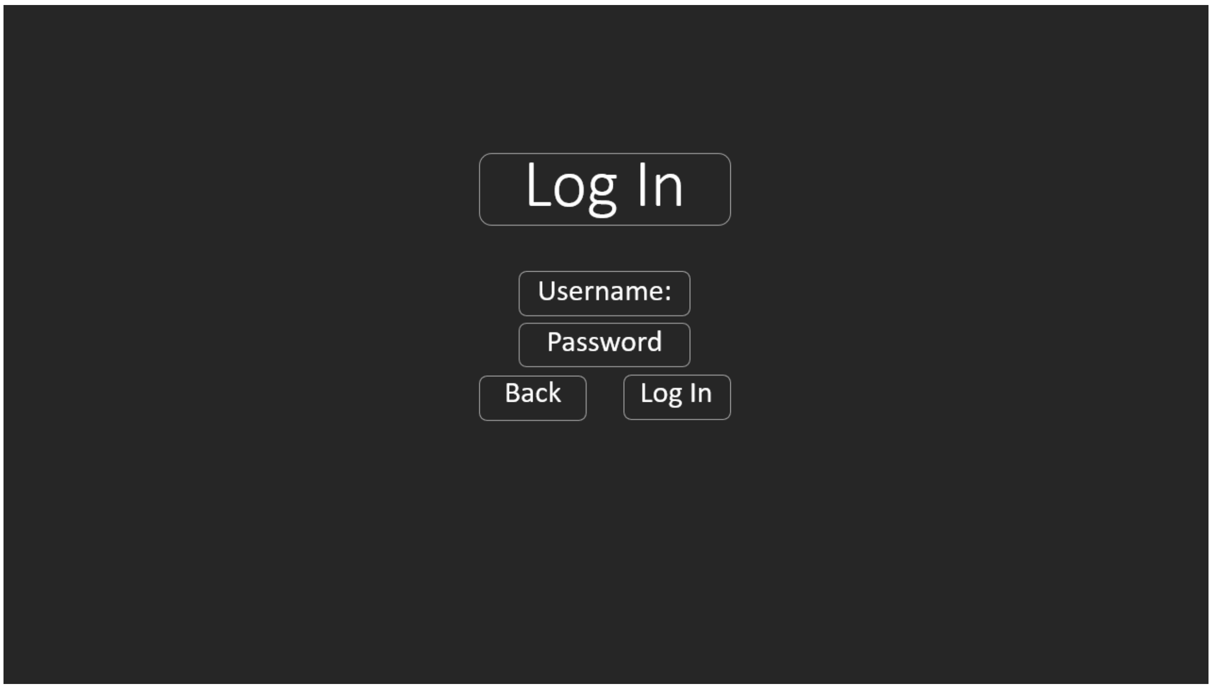
Welcome page:



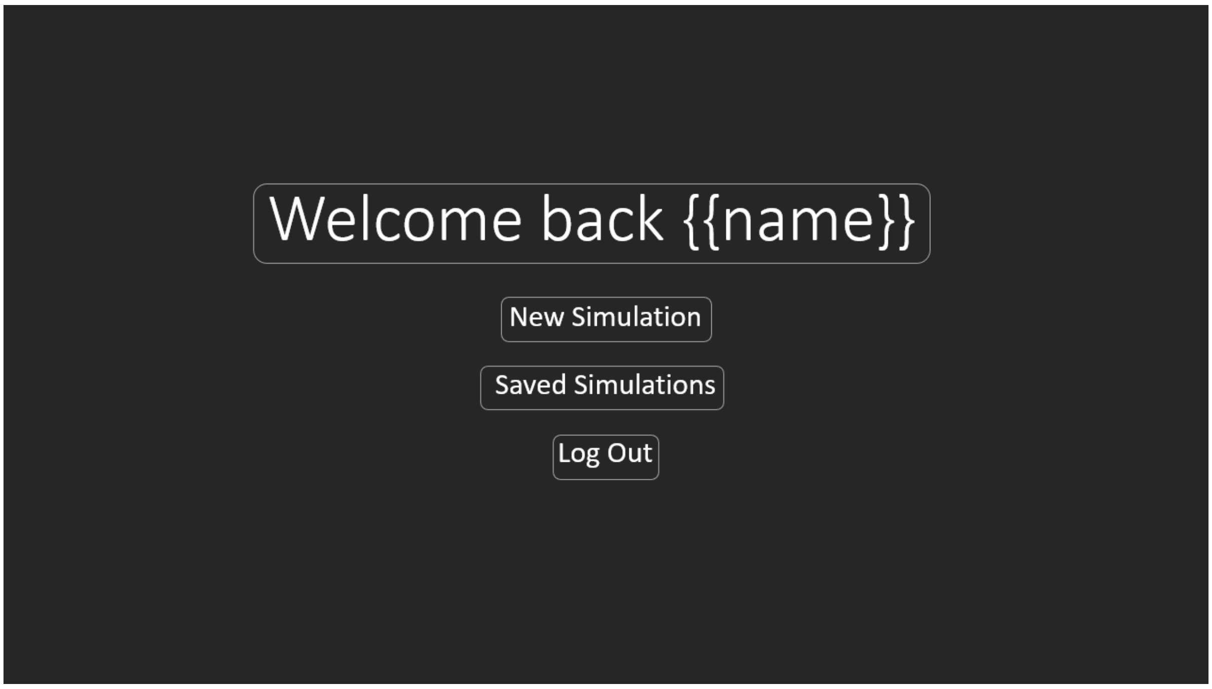
Sign-Up page:



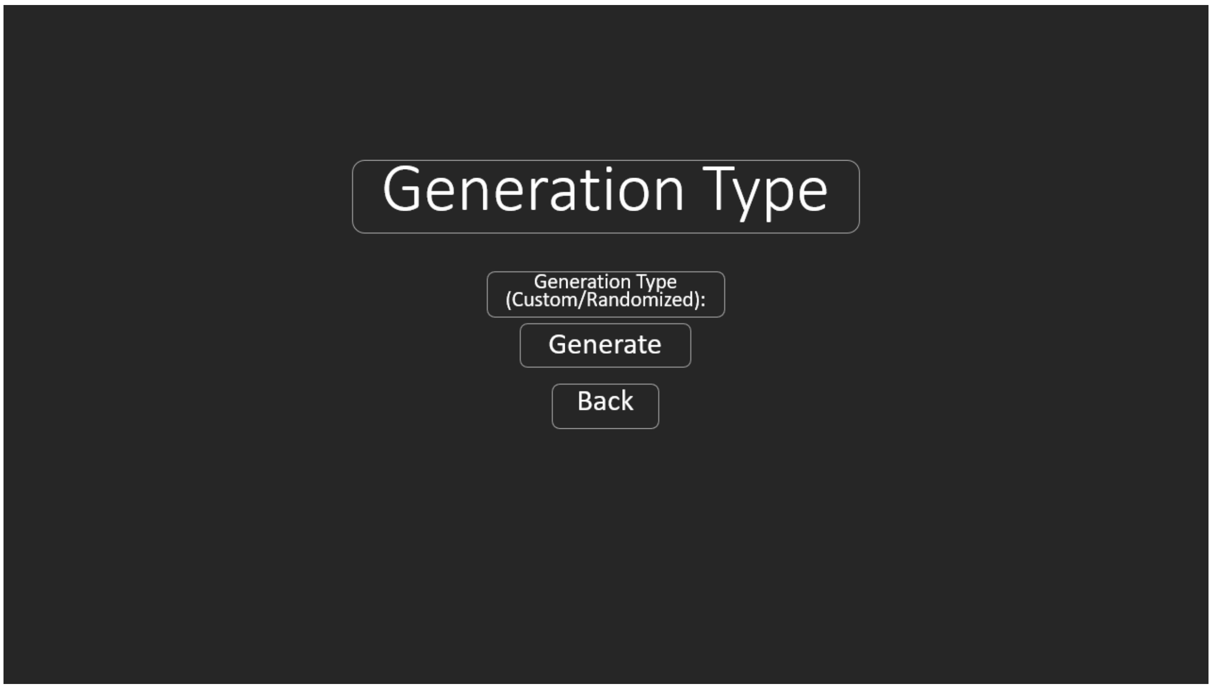
Login page:



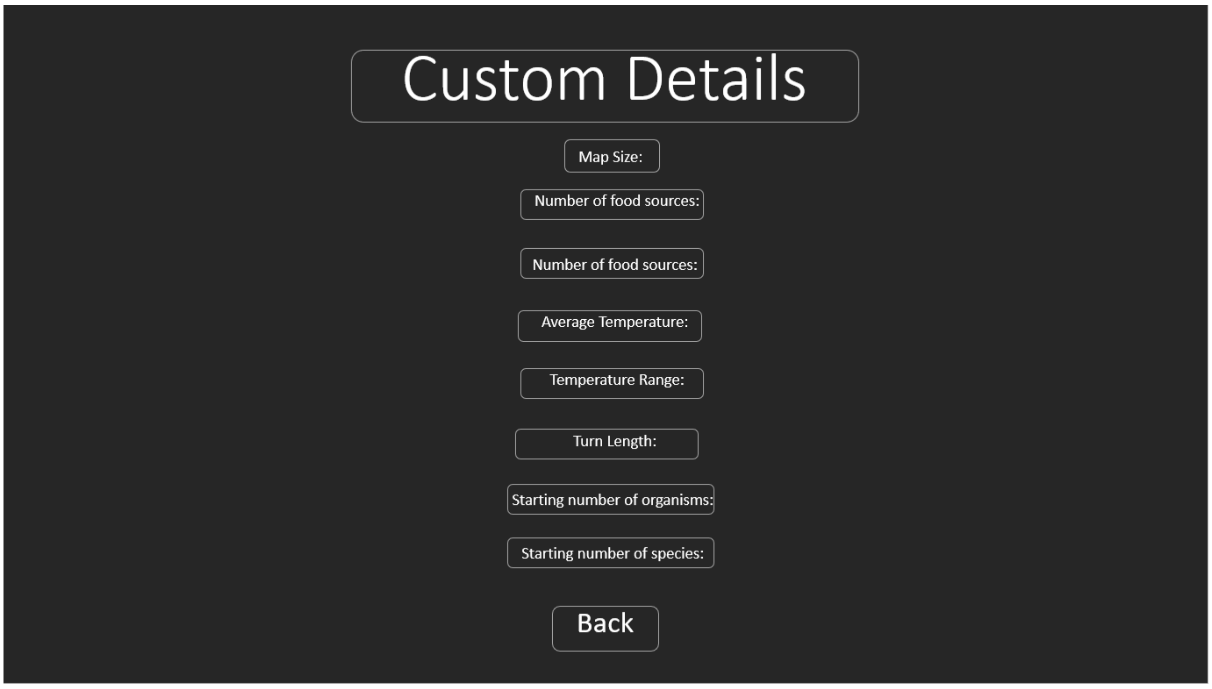
Home page:



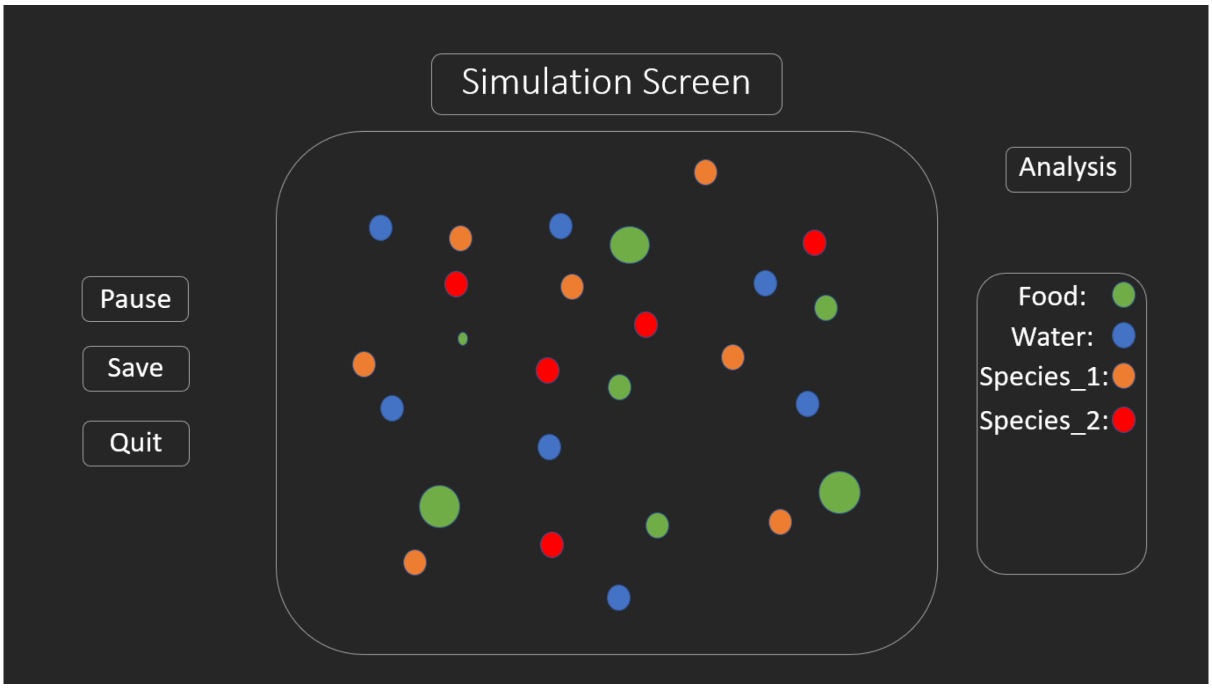
Generation Selection Screen:



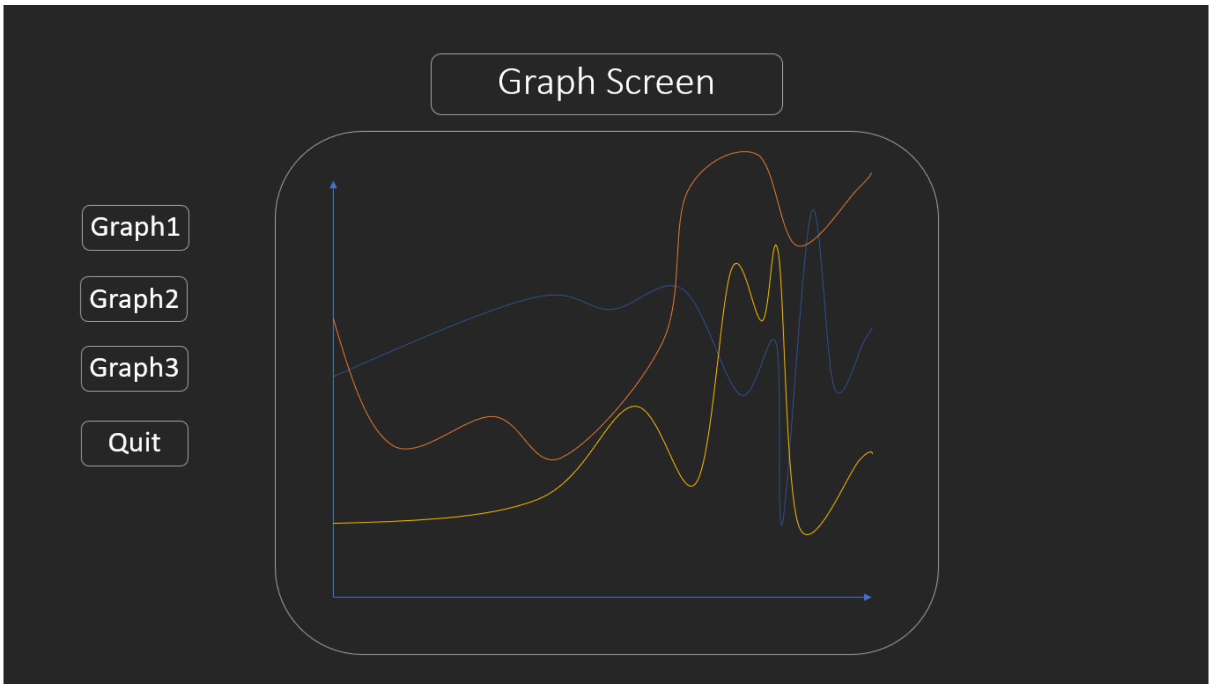
Detail Entry Screen:



Simulation Screen:



Graph Screen:



Program Overview:

0.1	Create a database according to plan	Will be the basis for the program and store all the programs data.
0.2	Create a welcome screen	Create a welcome screen
0.3	Create a user sign-up	Must have a screen that allows data entry and add the data to the database.
0.4	Create a user log-in	Must have a screen that allows data entry and then validate the data collected with the database.

0.5	Create a homepage	This will be the screen that the user arrives at when they have logged in: 1. Option to create a new simulation – button leading to simulation generation screen. 2. Option to load a saved simulation – button leading to saved simulation screen. 3. Option to log back out – button leading back to the log-in screen.
0.6	Create a new simulation detail entry screen.	This will be the entry of the map details and will allow the user to input their own or randomly generate a map. 1. Option for user data entry – which displays a data entry screen that links to the database and updates the least recently used simulation slot for the user. 2. Option for random map – a button that will randomly select variables. 3. Generate button that runs the map generation.

0.7	Map and simulation screen generation	Will generate the map according to the data inputted or randomized in the simulation detail entry. 1. Generate a screen for the simulation. 2. Collect data to show a map on the screen with the properties defined in the database – this will include the size of the map (the number of squares), the number and size of the food sources, the number of water sources.

0.8	Populate the map	1. Create x number of species objects, where x is the number defined in the database – these will have randomly generated traits which will be different for all the species. r 2. Generate y number of organism objects, where y is the number defined in the database – these will be equally split amongst all the species. 3. Randomly place all the organisms around the map and represent them with a circle of the organisms' colour.
0.9	Create simulation	Create a simulation as described in the simulation overview table.
1.0	Generating graphs	1. Using the data gathered by the simulation, display the graph screen. 2. On this graph screen allow the user to select which details they'd like to plot against the number of turns. 3. Use the selected response to gather the correct data from the database and plot the graph. 4. Display the graph on the screen for the user to see.

Simulation Overview:

0.9.1	Implement the organism movement.	1. Randomly select a number – 1,2,3 or 4 x times every second where x is the organism's speed.

		2. Make the object on the screen move in that direction (1: move up, 2: move, right, etc.).
0.9.2	Check the area around.	1. Collect the data for the range of detection of the organism from the database for that organism. 2. Next check all the squares which are within that range.
0.9.3	Reacting to the objects around them.	1. If there is an organism – check its species and if same, try and reproduce. Otherwise if it eats food type meat, next time you move away from it. 2. If there is food (or water) and the organisms have not had enough to eat (or drink) then approach the food square next turn.
0.9.4	Consuming food	1. If you are next to a food or water square consume it and add the nutrition to your current food in the database. 2. Delete the object that you consumed.
0.9.5	Fighting	1. Compare the two organisms' attack and defence tags. Determine who is the net winner of resources. 2. Winner adds resources to their reservoir. 3. If one of the organisms dies then it becomes a meat object that can be consumed.

0.9.6	Reproducing	1. If there is an organism of the same species, compare their reproduction codes to that of their species. Add the number of bits that match together and divide by the total number of reproduction code bits for that species. Then add together and divide be two to find their average percentage match. This is then used as the probability of successful reproduction. 2. Then create an offspring with a mixture of their properties and some random changes.
0.9.7	End of turn	1. At the end of every turn the data will be collected by the system and stored as the data for that turn for graph plotting.
0.9.8	Loss of Resources	1. Every time an organism moves, it must lose resources. This will be equal to the number of turns that the organism has been alive for.
0.9.9	Tweaking	1. In this section, I will add new chromosomes, and make any necessary adjustments to my code.

Technical Solution:

Creating the Database (Version 0.1):

```
import sqlalchemy
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, relationship
from sqlalchemy import Column, Integer, String, ForeignKey, DateTime

Base = declarative_base()

engine = create_engine('sqlite:///evolution_simulator.db', echo=True)
Session = sessionmaker(bind=engine)
Base.metadata.create_all(engine)
session = Session()

class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    name = Column(String(20))
    username = Column(String(20))
    password = Column(String(20))

    simulations_realtionship = relationship("Simulation", back_populates='user_relationship')

    def __repr__(self):
        return "<User(id: {0}, name: {1}, username: {2}, password: {3})".format(self.id, self.name,
self.username, self.password)

class Simulations(Base):
    __tablename__ = "simulations"
    id = Column(Integer, primary_key=True)
    current_turn = Column(Integer)
    length_of_turn = Column(Integer)
    number_of_starting_organisms = Column(Integer)
    number_of_starting_species = Column(Integer)
    current_number_of_organisms = Column(Integer)
    current_number_of_species = Column(Integer)

    user_id = Column(ForeignKey("users.id"))
    user_relationship = relationship("User", back_populates='simulations_relationship')

    species_relaionship = relationship("Species", back_populates='simulation_relationship')

    map_relationship = relationship("Map", back_populates='simulation_realationship')

    def __repr__(self):
        return "<Simulation(id: {0}, current turn: {1}, length of turn: {2}, current organisms: {3}, current
species: {4}, user: {5})>".format(self.id, self.current_turn, self.length_of_turn,
self.current_number_of_organisms, self.current_number_of_species, self.user_id)

class Species(Base):
    __tablename__ = "species"
    id = Column(Integer, primary_key=True)
    reproduction_code = Column(String(50))
    species_colour = Column(String(10))
    simulation_id = Column(ForeignKey("simulations.id"))
    simulation_relationship = relationship("Simulation", back_populates='species_relationship')

    species_history_relaionship = relationship("SpeciesHistory",
back_populates='species_relationship')

    organism_relationship = relationship("Organism", back_populates='species_relationship')

    def __repr__(self):
        return "<Species(species id: {0}, reproduction code: {1}, scpecies colour: {2}, simulation:
{3})>".format(self.id, self.reproduction_code, self.species_colour, self.simulation_id)

class SpeciesHistory(Base):
    __tablename__ = "species_history"
    id = Column(Integer, primary_key=True)
    population = Column(Integer)
    average_size = Column(Integer)
    average_speed = Column(Integer)

    species_id = Column(ForeignKey("species.id"))
    species_relationship = relationship("Species", back_populates='species_history_relationship')

    def __repr__(self):
        return "<SpeciesHistory(species history turn: {0}, population: {1}, average size: {2}, average
speed: {3}, species: {4})>".format(self.id, self.population, self.average_size, self.average_speed,
self.species_id)

class Organism(Base):
    __tablename__ = "organisms"
    id = Column(Integer, primary_key=True)
    food_required = Column(Integer)
    current_food = Column(Integer)
    water_required = Column(Integer)
    current_water = Column(Integer)
    fat_content = Column(Integer)
    reproduction_code = Column(Integer)
    size = Column(Integer)
    x_position = Column(Integer)
    y_position = Column(Integer)
    range_of_detection = Column(Integer)
    speed = Column(Integer)

    species_id = Column(ForeignKey("species.id"))
    species_relationship = relationship("Species", back_populates='organism_relationship')

    food_relationship = relationship("Food", back_populates='organism_relationship')

    def __repr__(self):
        return "<Organism(organism id: {0}, speces{1}, food: {2}/{3}, water: {4}/{5}, fat content: {6},
reproduction code: {7}, size: {8}, position: ({9}:{10}), range_of_detection: {11}, speed:
{12})>".format(self.id, self.species_id, self.current_food, self.food_required, self.current_water,
self.water_required, self.fat_content, self.reproduction_code, self.size, self.x_position, self.y_position,
self.range_of_detection, self.speed)

class Map(Base):
    __tablename__ = "maps"
    id = Column(Integer, primary_key=True)
    map_size = Column(Integer)
    number_of_food_sources = Column(Integer)
    number_of_water_sources = Column(Integer)
    average_temperature = Column(Integer)
    temperature_range = Column(Integer)

    simulation_id = Column(ForeignKey("simulations.id"))
    simulation_relationship = relationship("Simulation", back_populates='map_relationship')

    food_relationship = relationship("Food", back_populates="map_relationship")

    def __repr__(self):
        return "<Map(map_id: {0}, map size: {1}, number of food sources: {2}, number of water sources: {3}, averatge temperature: {4}, temperature range: {5}, simulation id: {6})>".format(self.id, self.map_size, self.number_of_food_sources, self.number_of_water_sources, self.average_temperature, self.temperature_range, self.simulation_id)

class Food(Base):
    __tablename__ = "foods"
    id = Column(Integer, primary_key=True)
    food_type = Column(Integer)
    nutritional_value = Column(Integer)
    position_x = Column(Integer)
    position_y = Column(Integer)

    map_id = Column(ForeignKey("simulations.id"))
    map_relationship = relationship("Map", back_populates='food_realtionship')

    organism_id = Column(ForeignKey("organisms.id"))
    organism_relationship = relationship("Organism", back_populates='food_relationship')

    def __repr__(self):
        return "<Food(food id: {0}, nutritional value: {1}, position: ({2},{3}), map id: {4}, organism id: {5})>".format(self.id, self.food_type, self.nutritional_value, self.position_x, self.position_y, self.map_id, self.organism_id)

session.commit()
```

Testing the database with mock data:

Filling the database with data:

In order to test my database, I must fill it with data so that I can then perform a series of queries to test that all sections of the code work. I filled the database with a series of .csv files that I generated using Mockaroo (https://mockaroo.com/) a website that generates test data files that I then read and used to fill my database. The following code shows how I read these files and added the data to the relevant tables in my database.

```
objects_to_add = []

if session.query(User).count() == 0:

    with open("users.csv", "r") as users_file:

        lines = users_file.readlines()
        for line in lines:
            name, username, password = line.rstrip().split(",")
            new_user = User(name=name, username=username, password=password)
            objects_to_add.append(new_user)
    session.add_all(objects_to_add)
    session.commit()
```

The file mentioned in the code above (users.csv) was the following (this is a cropped example):

```
id,name,username,password
1,Otto,otaplin0,VcY9owv
2,Skylar,ssheed1,06cHAKuiey
3,Ginger,gdowne2,WCDYOCso
4,Eugenius,estokoe3,wMDVkWGM
5,Angelica,aburgiss4,DVJ3xCN
```

I repeated the highlighted section of my code for each table within the database, modifying it accordingly. Each of these repeats also had its own .csv file similar to the one above (again modified according to the table's attributes).

Debugging the database:

While trying to write the section of code to add the mock data to the database I came across the following errors.

Variable name error:

```
species_relaionship = relationship("Species", back_populates='simulation_relationship')
```

In the line above I misspelt the variable name simulation_relationship causing an error in the relationships between the two objects.

I fixed this by changing the line to the following line:

```
species_relaionship = relationship("Species", back_populates='simulation_relationship')
```

Forgetting variables within the populating of the database:

```
current_turn, length_of_turn, starting_number_of_organisms, starting_number_of_species, current_number

_of_organisms, current_number_of_species, user_id = line.rstrip().split(",")

new_simulation = Simulation(current_turn=current_turn, length_of_turn=length_of_turn,
starting_number_of_organisms=starting_number_of_organisms, starting_number_of_species=starting_number_of_species,
current_number_of_organisms=current_number_of_organisms, current_number_of_species=current_number_of_species,
user_id=user_id)
```

In the line above I forgot to read and add the variables starting_number_of_organisms and starting_number_of_species

I fixed this by changing the line to the following line:

```
current_turn, length_of_turn, starting_number_of_organisms, starting_number_of

_species, current_number_of_organisms, current_number_of_species, user_id = li

ne.rstrip().split(",")

new_simulation = Simulation(current_turn=current_turn, length_of_turn=length_of_turn,
starting_number_of_organisms=starting_number_of_organisms, starting_number_of_species=starting_number_of_species,
current_number_of_organisms=current_number_of_organisms, current_number_of_species=current_number_of_species,
user_id=user_id)
```

Testing the database:

In order to test my database, I wrote a new python file that imported the database and all other relevant data and proceeded to write a series of test queries for each of the components of my database.

Testing the “users” table:

Test_1:

I wrote the following test to test the value of the first row of the users table.

```
user = session.query(User).first()

print(user)
```

Expected output – The first user in the database and all its data displayed according to user’s __repr__ command:

<User(id: 1, name: Buiron, username: bloades0, password: 03zXeye)>

Output I got:

<User(id: 1, name: name, username: username, password: password)>

This error was due to me having forgotten to remove the first line from the data files which I used as mock data.

In order to fix this, I removed the first line (which contains the table name from each of my mock data files). I then recreated my database. In my program this will not be an issue as I will only add the data and not the table names.

Output after the change:

<User(id: 1, name: Buiron, username: bloades0, password: 03zXeye)>

Test_2:

I wrote the following test to test the last value in the users table.

```
user = session.query(User).order_by(User.id.desc()).first()

print(user)
```

Expected output – The last user in the database and all its data displayed according to user’s __repr__ command:

<User(id: 100, name: Raven, username: rlazarus2r, password: 7jwPje)>

Output I got:

<User(id: 100, name: Raven, username: rlazarus2r, password: 7jwPje)>

Test_3:

I wrote the following test to test the attributes of each user in the database.

```
user = session.query(User).filter (User.name == "Rosamond").one()

print(user)


user = session.query(User).filter (User.username == "rzumfelde1").one()

print(user)
```

```
user = session.query(User).filter (User.password == "RdVvqyy").one()

print(user)
```

Expected output – I expect the same output for all of them as the test data to find all belongs to the same user. The result should be:

```
<User(id: 2, name: Rosamond, username: rzumfelde1, password: RdVvqyy)>
```

Output I got:

```
<User(id: 2, name: Rosamond, username: rzumfelde1, password: RdVvqyy)>
```

Test_4:

I wrote the following test to test the relationship between the table users and the simulations table.

```
user = session.query(User).first()

simulations = session.query(Simulation).filter(Simulation.user_relationship == user).all()

print(simulations)
```

Expected output – an array of simulation items displayed according to simulation's __repr__ command all with id of the first user (which should be 1).

```
[<Simulation(id: 77, current turn: 64, length of turn: 22, current organisms: 54, current species: 45, user: 1)>, <Simulation(id: 98, current turn: 86, length of turn: 23, current organisms: 59, current species: 18, user: 1)>]
```

Output I got:

```
[<Simulation(id: 77, current turn: 64, length of turn: 22, current organisms: 54, current species: 45, user: 1)>, <Simulation(id: 98, current turn: 86, length of turn: 23, current organisms: 59, current species: 18, user: 1)>]
```

Test_6:

I wrote the following test to test the map relationship of the simulations table in the database.

```
user = session.query(User).first()

simulation = user.simulations_relationship

print(simulation)
```

Expected output – The map belonging to simulation with simulation id 1, with its data displayed according to Map's __repr__ command:

```
[<Simulation(id: 77, current turn: 64, length of turn: 22, current organisms: 54, current species: 45, user: 1)>, <Simulation(id: 98, current turn: 86, length of turn: 23, current organisms: 59, current species: 18, user: 1)>]
```

Output I got:

```
[<Simulation(id: 77, current turn: 64, length of turn: 22, current organisms: 54, current species: 45, user: 1)>, <Simulation(id: 98, current turn: 86, length of turn: 23, current organisms: 59, current species: 18, user: 1)>]
```

Testing the "simulations" table:

Test_1:

I wrote the following test to test the value of the first row of the simulations table.

```
simulation = session.query(Simulation).first()

print(simulation)
```

Expected output – The first simulation in the database and all its data displayed according to simulation's __repr__ command:

```
<Simulation(id: 1, current turn: 22, length of turn: 75, current organisms: 7, current species: 49, user: 35)>
```

This does not include the starting number of species or starting number of organisms as the data is not relevant to each simulation past the initial generation phase.

Output I got:

```
<Simulation(id: 1, current turn: 22, length of turn: 75, current organisms: 7, current species: 49, user: 35)>
```

Test_2:

I wrote the following test to test the value of the last row of the simulations table.

```
simulation = session.query(Simulation).order_by(Simulation.id.desc()).first()
```

```
print(simulation)
```

Expected output – The last simulation in the database and all its data displayed according to simulation’s __repr__ command:

> <Simulation(id: 100, current turn: 86, length of turn: 39, current organisms: 9, current species: 91, user: 23)>

Output I got:

> <Simulation(id: 100, current turn: 86, length of turn: 39, current organisms: 9, current species: 91, user: 23)>

Test_3:

I wrote the following test to test each of the attributes of items in the simulation table in the database.

```
simulation = session.query(Simulation).filter(Simulation.current_turn == 54).first()

print(simulation)


simulation = session.query(Simulation).filter(Simulation.length_of_turn == 49).first()

print(simulation)


simulation = session.query(Simulation).filter(Simulation.starting_number_of_organisms == 58).first()

print(simulation)


simulation = session.query(Simulation).filter(Simulation.starting_number_of_species == 95).first()

print(simulation)


simulation = session.query(Simulation).filter(Simulation.current_number_of_organisms == 8).first()

print(simulation)


simulation = session.query(Simulation).filter(Simulation.current_number_of_species == 67).first()
```

```
print(simulation)
```

Expected output – The first simulation in the database which have the attributes given (which all belong to the same simulation for ease in testing) and all its data displayed according to simulation's __repr__ command. They should all give the same output:

```
<Simulation(id: 2, current turn: 54, length of turn: 49, current organisms: 8, current species: 67, user: 64)>
```

Output I got:

```
<Simulation(id: 2, current turn: 54, length of turn: 49, current organisms: 8, current species: 67, user: 64)>
```

Test_4:

I wrote the following test to test the species relationship of the simulations table in the database.

```
simulation = session.query(Simulation).first()

species = session.query(Species).filter(Species.simulation_relationship == simulation).all()

print(species)
```

Expected output – An array with all the species that belong to simulation with simulation id of 1:

```
[<Species(species id: 2, reproduction code: 4, species colour: 0, simulation: 1)>,
<Species(species id: 7, reproduction code: 0, species colour: 2, simulation: 1)>,
<Species(species id: 57, reproduction code: 0, species colour: 12, simulation: 1)>]
```

Output I got:

```
[<Species(species id: 2, reproduction code: 4, species colour: 0, simulation: 1)>,
<Species(species id: 7, reproduction code: 0, species colour: 2, simulation: 1)>,
<Species(species id: 57, reproduction code: 0, species colour: 12, simulation: 1)>]
```

Test_5:

I used the following code to check the backpopulate function of the simulation table.

```
simulation = session.query(Simulation).first()

species = simulation.species_relationship

print(species)
```

Expected output – An array with all the species that belong to simulation with simulation id of 1:

```
[<Species(species id: 2, reproduction code: 4, species colour: 0, simulation: 1)>,
<Species(species id: 7, reproduction code: 0, species colour: 2, simulation: 1)>,
<Species(species id: 57, reproduction code: 0, species colour: 12, simulation: 1)>]
```

Output I got:

```
[<Species(species id: 2, reproduction code: 4, species colour: 0, simulation: 1)>,
<Species(species id: 7, reproduction code: 0, species colour: 2, simulation: 1)>,
<Species(species id: 57, reproduction code: 0, species colour: 12, simulation: 1)>]
```

Test_6:

I wrote the following test to test the map relationship of the simulations table in the database.

```
simulation = session.query(Simulation).first()

test_map = session.query(Map).filter(Map.simulation_relationship == simulation).one()

print(test_map)
```

Expected output – The map belonging to simulation with simulation id 1, with its data displayed according to Map's __repr__ command:

```
<Map(map id: 2, map size: 60, number of food sources: 9, number of water sources: 16, average temperature: 32, temperature range: 5, simulation id: 1)>
```

Output I got:

```
sqlalchemy.orm.exc.MultipleResultsFound: Multiple rows were found for one()
```

This error was due to the simulation having more than one map. This was due to a repetition in the test data that I randomly generated. This will not be a problem in my program as I will be careful to not have repeated simulation ids. I fixed the repetition in my test data and re-ran the program.

Output I got:

```
<Map(map id: 2, map size: 60, number of food sources: 9, number of water sources: 16, average temperature: 32, temperature range: 5, simulation id: 1)>
```

Test_7:

I wrote the following test backpopulate function of the simulation table in the database.

```
simulation = session.query(Simulation).first()

map_test = simulation.map_relationship

print(map_test)
```

Expected output – The map belonging to simulation with simulation id 1, with its data displayed according to Map’s __repr__ command:

[<Map(map id: 2, map size: 60, number of food sources: 9, number of water sources: 16, average temperature: 32, temperature range: 5, simulation id: 1)>]

Output I got:

[<Map(map id: 2, map size: 60, number of food sources: 9, number of water sources: 16, average temperature: 32, temperature range: 5, simulation id: 1)>]

Testing the “species” table:

Test_1:

I wrote the following test to test the value of the first row of the species table.

```
species = session.query(Species).first()

print(species)
```

Expected output – The first simulation in the database and all its data displayed according to simulation’s __repr__ command:

<Species(species id: 1, reproduction code: 4, species colour: 9, simulation: 70)>

Output I got:

<Species(species id: 1, reproduction code: 4, species colour: 9, simulation: 70)>

Test_2:

I wrote the following test to test the last value in the species table.

```
species = session.query(Species).order_by(Species.id.desc()).first()

print(species)
```

Expected output – The last user in the database and all its data displayed according to species’ __repr__ command:

<Species(species id: 100, reproduction code: 0, species colour: 13, simulation: 31)>

Output I got:

<Species(species id: 100, reproduction code: 0, species colour: 13, simulation: 31)>

Test_3:

I wrote the following test to test each of the attributes of items in the species table in the database.

```
species = session.query(Species).filter(Species.reproduction_code == 3).first()

print(species)



species = session.query(Species).filter(Species.species_colour == 13).first()

print(species)
```

Expected output – The first species in the database which have the attributes given (which all belong to the same species for ease in testing) and all its data displayed according to species' __repr__ command. They should all give the same output:

<Species(species id: 3, reproduction code: 3, species colour: 13, simulation: 28)>

Output I got:

<Species(species id: 3, reproduction code: 3, species colour: 13, simulation: 28)>

Test_4:

I wrote the following test to test the species_history relationship of the species table in the database.

```
species = session.query(Species).first()

species_history = session.query(SpeciesHistory).filter(SpeciesHistory.species_relationship == species).all()

print(species_history)
```

Expected output – An array with all the species_history that belong to species with species id of 1 (in this case I only expect one result due me having few species_history in my test data however in my actual simulation, each species will have many species_history):

[<SpeciesHistory(species history turn: 75, population: 75, average size: 47, average speed: 9, species: 1)>]

Output I got:

[<SpeciesHistory(species history turn: 75, population: 75, average size: 47, average speed: 9, species: 1)>]

Test_5:

I wrote the following test backpopulate function of the organism table in the database.

```
species = session.query(Species).first()

species_history = species.species_history_relationship

print(species_history)
```

Expected output – An array with all the species_history that belong to species with species id of 1 (in this case I only expect one result due me having few species_history in my test data however in my actual simulation, each species will have many species_history):

```
[<SpeciesHistory(species history turn: 75, population: 75, average size: 47, average speed: 9, species: 1)>]
```

Output I got:

```
[<SpeciesHistory(species history turn: 75, population: 75, average size: 47, average speed: 9, species: 1)>]
```

Test_6:

I wrote the following test to test the species_history relationship of the species table in the database.

```
species = session.query(Species).first()

organism = session.query(Organism).filter(Organism.species_relationship == species).all()

print(organism)
```

Expected output – An array with all the organisms that belong to species with species id of 1 (some of the variables such as food do not make sense as the data was randomly generated. In my simulation the food an organism has will not be able to exceed the amount of food they require. It will cap at that amount):

```
[<Organism(organism id: 41, species: 1, food: 78/76, water: 50/72, fat content: 42, reproduction code: 3, size: 7, position: (6:36), range_of_detection: 4, speed: 8)>]
```

Output I got:

```
[<Organism(organism id: 41, species: 1, food: 78/76, water: 50/72, fat content: 42, reproduction code: 3, size: 7, position: (6:36), range_of_detection: 4, speed: 8)>]
```

Test_7:

I wrote the following test backpopulate function of the organism table in the database.

```
species = session.query(Species).first()

organism = species.organism_relationship

print(organism)
```

Expected output – An array with all the organisms that belong to species with species id of 1 (some of the variables such as food do not make sense as the data was randomly generated. In my simulation the food an organism has will not be able to exceed the amount of food they require. It will cap at that amount):

[<Organism(organism id: 41, species: 1, food: 78/76, water: 50/72, fat content: 42, reproduction code: 3, size: 7, position: (6:36), range_of_detection: 4, speed: 8)>]

Output I got:

[<Organism(organism id: 41, species: 1, food: 78/76, water: 50/72, fat content: 42, reproduction code: 3, size: 7, position: (6:36), range_of_detection: 4, speed: 8)>]

Testing the “species history” table:

Test_1:

I wrote the following test to test the value of the first row of the species history table.

```
species_history = session.query(SpeciesHistory).first()

print(species_history)
```

Expected output – The first species history in the database and all its data displayed according to species history’s __repr__ command:

<SpeciesHistory(species history turn: 1, population: 4, average size: 22, average speed: 84, species: 33)>

Output I got:

<SpeciesHistory(species history turn: 1, population: 4, average size: 22, average speed: 84, species: 33)>

Test_2:

I wrote the following test to test the value of the last row of the species history table.

```
species_history = session.query(SpeciesHistory).order_by(SpeciesHistory.id.desc()).first()

print(species_history)
```

Expected output – The last species history in the database and all its data displayed according to species history’s __repr__ command:

<SpeciesHistory(species history turn: 100, population: 4, average size: 36, average speed: 48, species: 2)>

Output I got:

<SpeciesHistory(species history turn: 100, population: 4, average size: 36, average speed: 48, species: 2)>

Test_3:

I wrote the following test to test each of the attributes of items in the species history table in the database.

```
species_history = session.query(SpeciesHistory).filter(SpeciesHistory.population == 74).first()

print(species_history )


species_history  = session.query(SpeciesHistory).filter(SpeciesHistory.average_size == 25).first()

print(species_history)


species_history  = session.query(SpeciesHistory).filter(SpeciesHistory.average_speed == 52).first()

print(species_history)
```

Expected output – The first species history in the database which have the attributes given (which all belong to the same species history for ease in testing) and all its data displayed according to species history's __repr__ command. They should all give the same output:

<SpeciesHistory(species history turn: 2, population: 74, average size: 25, average speed: 52, species: 82)>

Output I got:

<SpeciesHistory(species history turn: 2, population: 74, average size: 25, average speed: 52, species: 82)>

Testing the “organism” table:

Test_1:

I wrote the following test to test the value of the first row of the organism table.

```
organism= session.query(Organism).first()

print(organism)
```

Expected output – The first organism in the database and all its data displayed according to organism’s __repr__ command:

```
<Organism(organism id: 1, species: 51, food: 93/26, water: 56/68, fat content: 5,
reproduction code: 4, size: 38, position: (49:31), range_of_detection: 9, speed: 6)>
```

Output I got:

```
<Organism(organism id: 1, species: 51, food: 93/26, water: 56/68, fat content: 5,
reproduction code: 4, size: 38, position: (49:31), range_of_detection: 9, speed: 6)>
```

Test_2

I wrote the following test to test the last value in the species table.

```
organism = session.query(Organism).order_by(Organism.id.desc()).first()

print(organism)
```

Expected output – The last user in the database and all its data displayed according to species’ __repr__ command:

```
<Organism(organism id: 100, species: 5, food: 21/25, water: 47/76, fat content: 6,
reproduction code: 3, size: 25, position: (82:98), range_of_detection: 5, speed: 8)>
```

Output I got:

```
<Organism(organism id: 100, species: 5, food: 21/25, water: 47/76, fat content: 6,
reproduction code: 3, size: 25, position: (82:98), range_of_detection: 5, speed: 8)>
```

Test_3:

I wrote the following test to test each of the attributes of items in the organism table in the database.

```
organism = session.query(Organism).filter(Organism.current_food == 14).first()

print(organism)


organism = session.query(Organism).filter(Organism.food_required == 88).first()

print(organism)
```

```
organism = session.query(Organism).filter(Organism.current_water == 13).first()

print(organism)


organism = session.query(Organism).filter(Organism.water_required == 93).first()

print(organism)



organism = session.query(Organism).filter(Organism.fat_content == 32).first()

print(organism)


organism = session.query(Organism).filter(Organism.reproduction_code == 1).first()

print(organism)


organism = session.query(Organism).filter(Organism.size == 19).first()

print(organism)


organism = session.query(Organism).filter(Organism.x_position == 44).first()

print(organism)


organism = session.query(Organism).filter(Organism.y_position == 18).first()

print(organism)


organism = session.query(Organism).filter(Organism.range_of_detection == 3).first()

print(organism)


organism = session.query(Organism).filter(Organism.speed == 7).first()

print(organism)
```

Expected output – The first organisms in the database which have the attributes given (which all belong to the same organism for ease in testing) and all its data displayed according to organism's __repr__ command. They should all give the same output:

```
<Organism(organism id: 2, species: 68, food: 14/88, water: 13/93, fat content: 32,
reproduction code: 1, size: 19, position: (44:18), range_of_detection: 3, speed: 7)>
```

Output I got:

```
<Organism(organism id: 2, species: 68, food: 14/88, water: 13/93, fat content: 32, reproduction code: 1, size: 19, position: (44:18), range_of_detection: 3, speed: 7)>
```

Test_4:

I wrote the following test to test the food relationship of the organism table in the database.

```
organism = session.query(Organism).first()

food = session.query(Food).filter(Food.organism_relationship == organism).all()

print(food)
```

Expected output – An array with all the food that the organism with organism id of 1 can consume:

```
[<Food(food id: 1, food type: veg, nutritional value: 60, position: (16:98), map id: 57, organism id: 1)>, <Food(food id: 4, food type: meat, nutritional value: 61, position: (73:3), map id: 21, organism id: 1)>]
```

Output I got:

```
[<Food(food id: 1, nutritional value: veg, position: (60,16), map id: 98, organism id: 57)>, <Food(food id: 4, nutritional value: meat, position: (61,73), map id: 3, organism id: 21)>]
```

This error was due to the __repr__ command for class Food being erroneous. I forgot to display the food type of the food object which led to all the other attributes being shifted up and so all the data being displayed for the wrong attributes.

This was the __repr__ command before the correction:

```
def __repr__(self):
    return "<Food(food id: {0}, nutritional value: {1}, position: ({2},{3}), map id: {4}, organism id: {5})>".format(self.id, self.food_type, self.nutritional_value, self.position_x, self.position_y, self.map_id, self.organism_id)
```

This was the __repr__ command after the correction:

```
    def __repr__(self):
        return "<Food(food id: {0}, food type: {1}, nutritional value: {2}, position: ({3}:{4}), map id: {5}, organism id: {6})>".format(self.id, self.food_type, self.nutritional_value, self.position_x, self.position_y, self.map_id, self.organism_id)
```

Output I got:

```
[<Food(food id: 1, food type: veg, nutritional value: 60, position: (16:98), map id: 57, organism id: 1)>, <Food(food id: 4, food type: meat, nutritional value: 61, position: (73:3), map id: 21, organism id: 1)>]
```

Test_5:

I wrote the following test backpopulate function of the organism table in the database.

```
organism = session.query(Organism).first()
food = organism.food_relationship
print(food)
```

Expected output – An array with all the food that the organism with organism id of 1:

```
[<Food(food id: 1, food type: veg, nutritional value: 60, position: (16:98), map id: 57, organism id: 1)>, <Food(food id: 4, food type: meat, nutritional value: 61, position: (73:3), map id: 21, organism id: 1)>]
```

Output I got:

```
[<Food(food id: 1, food type: veg, nutritional value: 60, position: (16:98), map id: 57, organism id: 1)>, <Food(food id: 4, food type: meat, nutritional value: 61, position: (73:3), map id: 21, organism id: 1)>]
```

Testing the “map” table:

Test_1:

I wrote the following test to test the value of the first row of the organism table.

```
map_test = session.query(Map).first()

print(map_test)
```

Expected output – The first map in the database and all its data displayed according to map's __repr__ command:

<Map(map id: 1, map size: 89, number of food sources: 66, number of water sources: 71, average temperature: 9, temperature range: 9, simulation id: 17)>

Output I got:

<Map(map id: 1, map size: 89, number of food sources: 66, number of water sources: 71, average temperature: 9, temperature range: 9, simulation id: 17)>

Test_2:

I wrote the following test to test the last value in the maps table.

```
map_test = session.query(Map).order_by(Map.id.desc()).first()

print(map_test)
```

Expected output – The last user in the database and all its data displayed according to species' __repr__ command:

<Map(map id: 100, map size: 12, number of food sources: 35, number of water sources: 37, average temperature: 12, temperature range: 8, simulation id: 74)>

Output I got:

<Map(map id: 100, map size: 12, number of food sources: 35, number of water sources: 37, average temperature: 12, temperature range: 8, simulation id: 74)>

Test_3:

I wrote the following test to test each of the attributes of items in the map table in the database.

```
map_test = session.query(Map).filter(Map.map_size == 60).first()

print(map_test)
```

```
map_test = session.query(Map).filter(Map.number_of_food_sources == 9).first()

print(map_test)


map_test = session.query(Map).filter(Map.number_of_water_sources == 16).first()

print(map_test)


map_test = session.query(Map).filter(Map.average_temperature == 32).first()

print(map_test)


map_test = session.query(Map).filter(Map.temperature_range == 5).first()

print(map_test)
```

Expected output – The first map in the database which have the attributes given (which all belong to the same map for ease in testing) and all its data displayed according to map's __repr__ command. They should all give the same output:

> <Map(map id: 2, map size: 60, number of food sources: 9, number of water sources: 16, average temperature: 32, temperature range: 5, simulation id: 1)>

Output I got:

> <Map(map id: 2, map size: 60, number of food sources: 9, number of water sources: 16, average temperature: 32, temperature range: 5, simulation id: 1)>

Test_4:

I wrote the following test to test the food relationship of the map table in the database.

```
Map_test = session.query(Map).first()

food = session.query(Food).filter(Food.map_relationship == map_test).all()

print(food)
```

Expected output – An array with all the food that belongs to the map with map id of 1:

> [<Food(food id: 9, food type: meat, nutritional value: 76, position: (70:65), map id: 1, organism id: 85)>]

Output I got:

[<Food(food id: 9, food type: meat, nutritional value: 76, position: (70:65), map id: 1, organism id: 85)>]

Test_5:

I wrote the following test backpopulate function of the map table in the database.

```
map_test = session.query(Map).first()

food = map_test.food_relationship

print(food)
```

Expected output – An array with all the food that belongs to map with map id of 1:

[<Food(food id: 9, food type: meat, nutritional value: 76, position: (70:65), map id: 1, organism id: 85)>]

Output I got:

[<Food(food id: 9, food type: meat, nutritional value: 76, position: (70:65), map id: 1, organism id: 85)>]

Testing the “food” table:

Test_1:

I wrote the following test to test the value of the first row of the food table.

```
food = session.query(Food).first()

print(food)
```

Expected output – The first food in the database and all its data displayed according to food’s __repr__ command:

<Food(food id: 1, food type: veg, nutritional value: 60, position: (16:98), map id: 57, organism id: 1)>

Output I got:

<Food(food id: 1, food type: veg, nutritional value: 60, position: (16:98), map id: 57, organism id: 1)>

Test_2:

I wrote the following test to test the value of the last row of the food table.

```
food = session.query(Food).order_by(Food.id.desc()).first()

print(food)
```

Expected output – The last food in the database and all its data displayed according to food's __repr__ command:

<Food(food id: 10, food type: nuts, nutritional value: 22, position: (21:8), map id: 34, organism id: 77)>

Output I got:

<Food(food id: 10, food type: nuts, nutritional value: 22, position: (21:8), map id: 34, organism id: 77)>

Test_3:

I wrote the following test to test each of the attributes of items in the food table in the database.

```
food = session.query(Food).filter(Food.food_type == "veg").first()

print(food)


food = session.query(Food).filter(Food.nutritional_value == 60).first()

print(food)


food = session.query(Food).filter(Food.position_x == 16).first()

print(food)


food = session.query(Food).filter(Food.position_y == 98).first()

print(food)
```

Expected output – The first food items in the database which have the attributes given (which all belong to the same food item for ease in testing) and all its data displayed according to food's __repr__ command. They should all give the same output:

<Food(food id: 1, food type: veg, nutritional value: 60, position: (16:98), map id: 57, organism id: 1)>

Output I got:

```
<Food(food id: 1, food type: veg, nutritional value: 60, position: (16:98), map id: 57, organism id: 1)>
```

Finalising Version 0.1:

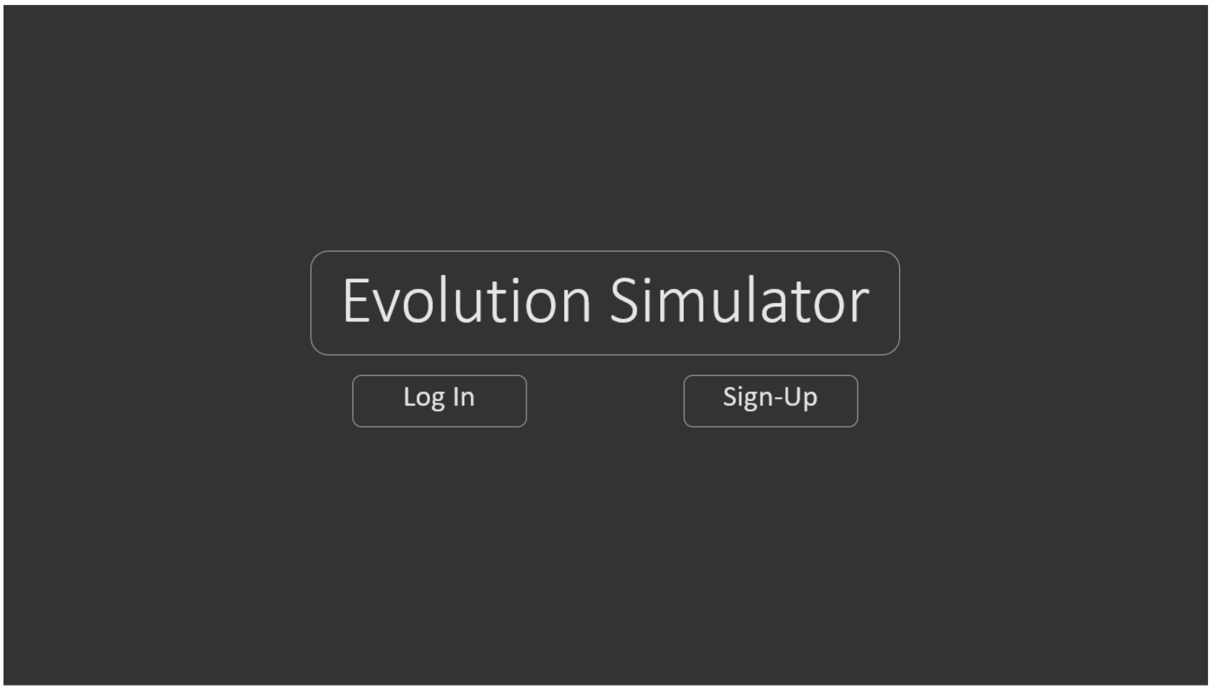
After debugging, the following is the code I have developed to create my database.

```
import sqlalchemy
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, relationship
from sqlalchemy import Column, Integer, String, ForeignKey, DateTime

Base = declarative_base()
engine = create_engine('sqlite:///evolution_simulator.db', echo=True)
Session = sessionmaker(bind=engine)
Base.metadata.create_all(engine)
session = Session()

class User(Base):
   __tablename__ = 'users'
   id = Column(Integer, primary_key=True)
   name = Column(String(20))
   username = Column(String(20))
   password = Column(String(20))

   simulations_relationship = relationship("Simulation", back_populates='user_relationship')

   def __repr__(self):
      return """<User(id: {0}, name: {1}, username: {2}, password: {3})>""".format(self.id, self.name,
self.username, self.password)

class Simulation(Base):
   __tablename__ = "simulations"
   id = Column(Integer, primary_key=True)
   current_turn = Column(Integer)
   length_of_turn = Column(Integer)
   starting_number_of_organisms = Column(Integer)
   starting_number_of_species = Column(Integer)
   current_number_of_organisms = Column(Integer)
   current_number_of_species = Column(Integer)

   user_id = Column(ForeignKey("users.id"))
   user_relationship = relationship("User", back_populates='simulations_relationship')
   species_relationship = relationship("Species", back_populates='simulation_relationship')
   map_relationship = relationship("Map", back_populates='simulation_relationship')

def __repr__(self):
      return "<Simulation(id: {0}, current turn: {1}, length of turn: {2}, current organisms: {3}, current
species: {4}, user: {5})>".format(self.id, self.current_turn, self.length_of_turn,
self.current_number_of_organisms, self.current_number_of_species, self.user_id)

class Species(Base):
   __tablename__ = "species"
   id = Column(Integer, primary_key=True)
   reproduction_code = Column(String(50))
   species_colour = Column(String(10))
```

```
    simulation_id = Column(ForeignKey("simulations.id"))
    simulation_relationship = relationship("Simulation", back_populates='species_relationship')
    species_history_relationship = relationship("SpeciesHistory",
back_populates='species_relationship')
    organism_relationship = relationship("Organism", back_populates='species_relationship')

    def __repr__(self):
        return "<Species(species id: {0}, reproduction code: {1}, species colour: {2}, simulation:
{3})>".format(self.id, self.reproduction_code, self.species_colour, self.simulation_id)

class SpeciesHistory(Base):
    __tablename__ = "species_history"
    id = Column(Integer, primary_key=True)
    population = Column(Integer)
    average_size = Column(Integer)
    average_speed = Column(Integer)
    species_id = Column(ForeignKey("species.id"))
    species_relationship = relationship("Species", back_populates='species_history_relationship')

    def __repr__(self):
        return "<SpeciesHistory(species history turn: {0}, population: {1}, average size: {2}, average
speed: {3}, species: {4})>".format(self.id, self.population, self.average_size, self.average_speed,
self.species_id)

class Organism(Base):
    __tablename__ = "organisms"
    id = Column(Integer, primary_key=True)
    food_required = Column(Integer)
    current_food = Column(Integer)
    water_required = Column(Integer)
    current_water = Column(Integer)
    fat_content = Column(Integer)
    reproduction_code = Column(Integer)
    size = Column(Integer)
    x_position = Column(Integer)
    y_position = Column(Integer)
    range_of_detection = Column(Integer)
    speed = Column(Integer)

    species_id = Column(ForeignKey("species.id"))
    species_relationship = relationship("Species", back_populates='organism_relationship')
    food_relationship = relationship("Food", back_populates='organism_relationship')
    def __repr__(self):
        return "<Organism(organism id: {0}, species: {1}, food: {2}/{3}, water: {4}/{5}, fat content: {6},
reproduction code: {7}, size: {8}, position: ({9}:{10}), range_of_detection: {11}, speed:
{12})>".format(self.id, self.species_id, self.current_food, self.food_required, self.current_water,
self.water_required, self.fat_content, self.reproduction_code, self.size, self.x_position, self.y_position,
self.range_of_detection, self.speed)

class Map(Base):
    __tablename__ = "maps"
    id = Column(Integer, primary_key=True)
    map_size = Column(Integer)
    number_of_food_sources = Column(Integer)
    number_of_water_sources = Column(Integer)
    average_temperature = Column(Integer)
    temperature_range = Column(Integer)

    simulation_id = Column(ForeignKey("simulations.id"))
    simulation_relationship = relationship("Simulation", back_populates='map_relationship')
    food_relationship = relationship("Food", back_populates="map_relationship")

    def __repr__(self):
        return "<Map(map id: {0}, map size: {1}, number of food sources: {2}, number of water sources:
{3}, average temperature: {4}, temperature range: {5}, simulation id: {6})>".format(self.id,
self.map_size, self.number_of_food_sources, self.number_of_water_sources,
self.average_temperature, self.temperature_range, self.simulation_id)

class Food(Base):
    __tablename__ = "foods"
    id = Column(Integer, primary_key=True)
    food_type = Column(Integer)
    nutritional_value = Column(Integer)
    position_x = Column(Integer)
    position_y = Column(Integer)

    map_id = Column(ForeignKey("maps.id"))
    map_relationship = relationship("Map", back_populates='food_relationship')
    organism_id = Column(ForeignKey("organisms.id"))
    organism_relationship = relationship("Organism", back_populates='food_relationship')

    def __repr__(self):
        return "<Food(food id: {0}, food type: {1}, nutritional value: {2}, position: ({3}:{4}), map id: {5},
organism id: {6})>".format(self.id, self.food_type, self.nutritional_value, self.position_x, self.position_y,
self.map_id, self.organism_id)

session.commit()
```

Creating the Welcome Screen (Version 0.2):

Proposed Welcome Screen Design:



Overview:

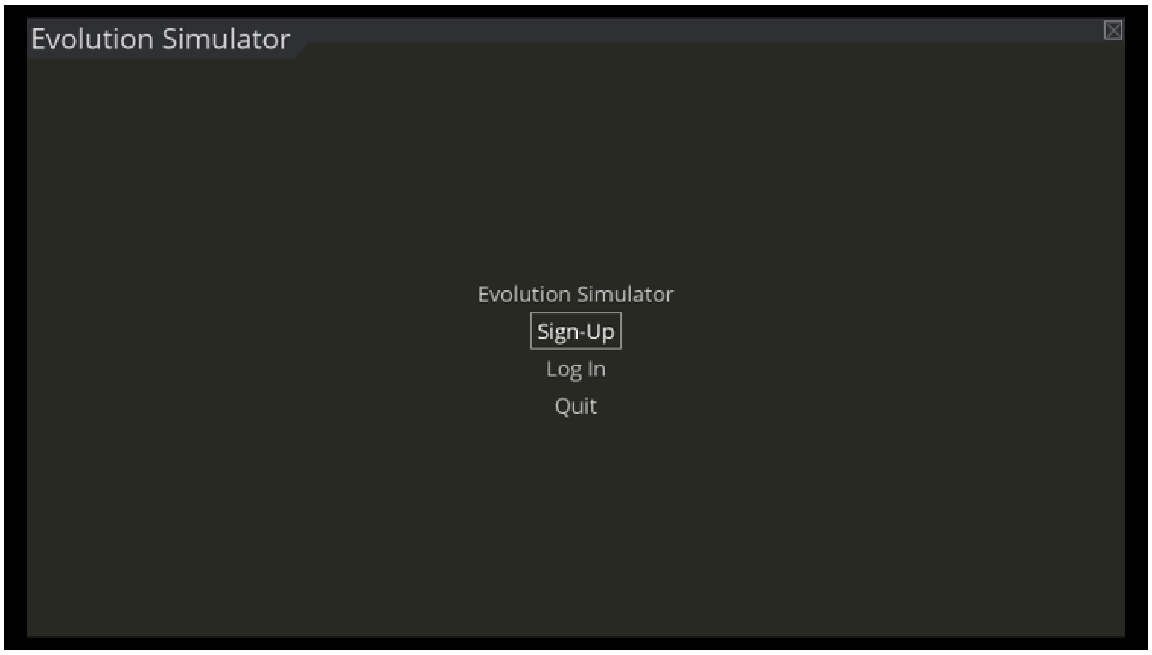
The welcome screen will provide a welcome for the user. It will allow the user to choose which screen they wish to move on to.

Version 0.2 code:

```
import pygame
import pygame_menu
from pygame.locals import *
from tkinter import *
window = Tk()
pygame.init()
SCREEN_WIDTH = window.winfo_screenwidth()
SCREEN_HEIGHT = window.winfo_screenheight()

class welcome_screen():
    def __init__(self):
        SCREEN_WIDTH = window.winfo_screenwidth()
        SCREEN_HEIGHT = window.winfo_screenheight()
        surface = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT), FULLSCREEN)
        self.menu = pygame_menu.Menu(SCREEN_HEIGHT, SCREEN_WIDTH, 'Evolution Simulator',
theme=pygame_menu.themes.THEME_DARK)
        self.menu.add_label('Evolution Simulator')
        self.menu.add_button('Sign-Up', self.sign_in)
        self.menu.add_button('Log In', self.log_in)
        self.menu.add_button('Quit', pygame_menu.events.EXIT)
        self.menu.mainloop(surface)
    def sign_in(self):
        print("Sign Up")
    def log_in(self):
        print("Log In")
welcome_screen()
```

In order to create the welcome page I use the pygame menu to create a simple easy to use welcome screen. In subsequent versions I will implement the sign in and log in functions to move the user onto the signup and log in pages

Welcome Screen I created:



Quick Test of the Welcome Screen Version:

When I used my mouse to hit the sign-up button to test that the menu calls the correct function when the sign-up button is clicked:

“Sign In” which was the expected output from print(“Sign In”)

When I used my mouse to hit the login button to test that the menu calls the correct function when the login button is clicked:

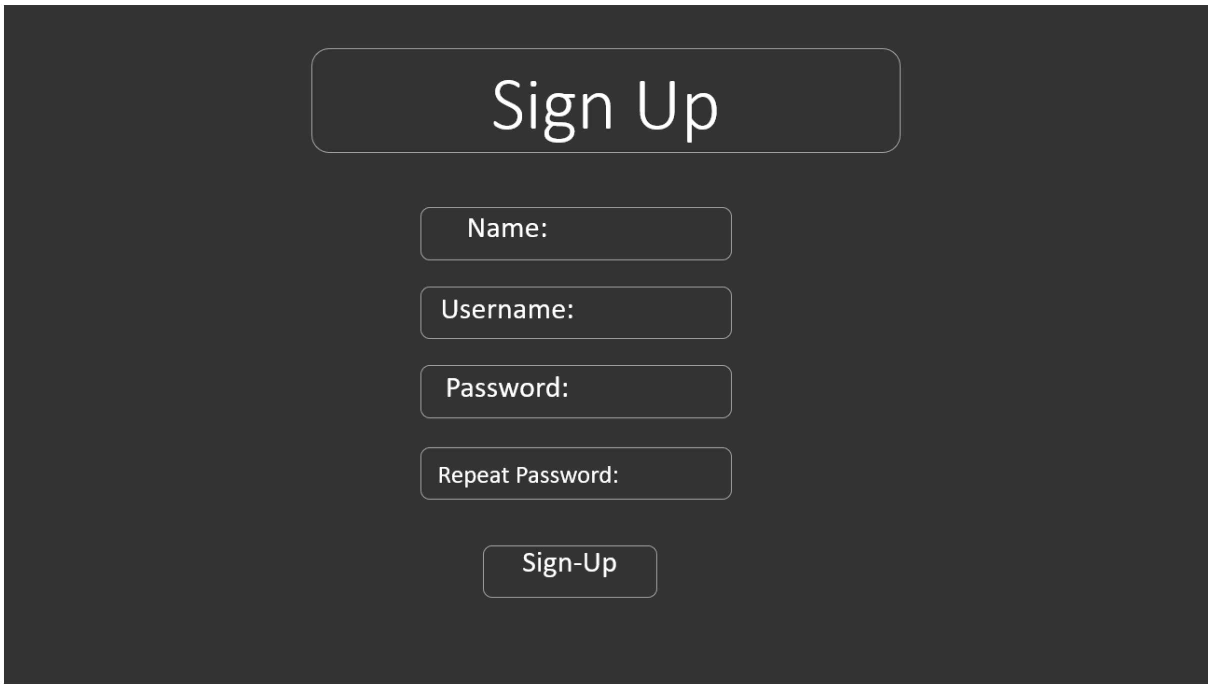
“Log In” which was the expected output from print(“Log In”)

When I used my mouse to hit the quit button to test that the program will close when asked to :

The pygame closed as and the program stopped running both as expected

Creating the Sign Up (Version 0.3):

Proposed Sign-Up Screen Design:



The sign up screen should allow the user to input their details for the account they wish to create. If the details are invalid it should return them back to the sing up screen to try again.

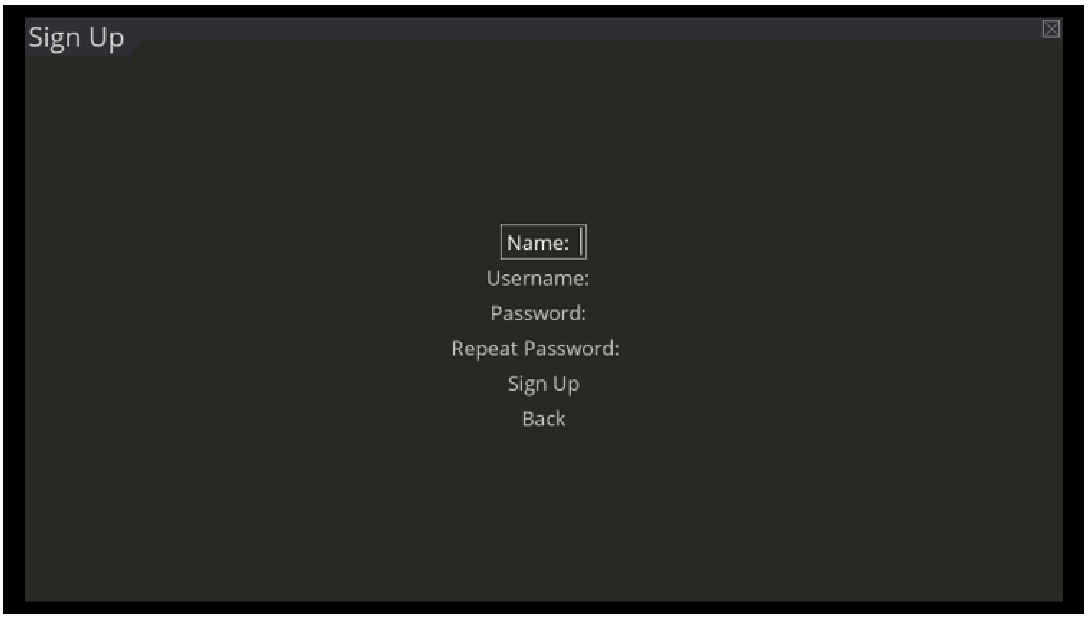
Version 0.3 code:

```
import pygame
import pygame_menu
from pygame.locals import *
from tkinter import *
window = Tk()
pygame.init()
SCREEN_WIDTH = window.winfo_screenwidth()
SCREEN_HEIGHT = window.winfo_screenheight()

from Evolution_simulator_schema_v2 import User, Simulation, Species, SpeciesHistory, Organism,
Map, Food
from sqlalchemy.orm import sessionmaker
from sqlalchemy import create_engine
from datetime import datetime
engine = create_engine('sqlite:///evolution_simulator.db', echo=False)
session = sessionmaker(bind=engine)()
class Program():
    def __init__(self):
        self.user_logged_in = None
        Welcome_screen()
```

```
class Welcome_screen():
    def __init__(self):
        SCREEN_WIDTH = window.winfo_screenwidth()
        SCREEN_HEIGHT = window.winfo_screenheight()
        surface = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT), FULLSCREEN)
        self.menu = pygame_menu.Menu(SCREEN_HEIGHT, SCREEN_WIDTH, 'Evolution Simulator',
theme=pygame_menu.themes.THEME_DARK)
        self.menu.add_label('Evolution Simulator')
        self.menu.add_button('Sign-Up', self.sign_in)
        self.menu.add_button('Log In', self.log_in)
        self.menu.add_button('Quit', pygame_menu.events.EXIT)
        self.menu.mainloop(surface)

    def sign_in(self):
        Signup_screen()

    def log_in(self):
        print("Log In")

class Signup_screen():
    def __init__(self):
        SCREEN_WIDTH = window.winfo_screenwidth()
        SCREEN_HEIGHT = window.winfo_screenheight()
        surface = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT), FULLSCREEN)
        self.menu = pygame_menu.Menu(SCREEN_HEIGHT, SCREEN_WIDTH, "Sign Up",
theme=pygame_menu.themes.THEME_DARK)
        self.input_name = self.menu.add_text_input("Name:  ", default="")
        self.input_username = self.menu.add_text_input("Username:  ", default="")
        self.input_password = self.menu.add_text_input("Password:  ", default="", password=True)
        self.input_repeat_password = self.menu.add_text_input("Repeat Password:   ", default="",
password=True)
        self.menu.add_button("Sign Up", self.sign_up)
        self.menu.add_button("Back", self.welcome_screen)
        self.menu.mainloop(surface)

    def welcome_screen(self):
        Welcome_screen()

    def home_screen(self):
        print("Home Screen")

    def sign_up(self):
        data = self.menu.get_input_data(recursive=False)
        input_name = data[self.input_name.get_id()]
        input_username = data[self.input_username.get_id()]
        input_password = data[self.input_password.get_id()]
        input_repeat_password = data[self.input_repeat_password.get_id()]

        self.validate_data(input_name, input_username, input_password, input_repeat_password)

    def validate_data(self, name, username, password, repeat_password):
        usernames = session.query(User).filter(User.username == username).all()
        if len(usernames) != 0:
            Signup_screen()
        if password != repeat_password:
            Signup_screen()
        self.add_data_to_database(name, username, password, repeat_password)

    def add_data_to_database(self, name, username, password, repeat_password):
        user = User(name=name, username=username, password=password)
        session.add(user)
        session.commit()
        Program.user_logged_in = user
        self.home_screen()

Program()
```

In version 0.3 I implemented some changes to the initialisation of the program and created the program class to store the user that is currently logged in.

Sign-Up Screen I created:



The solution I created accepts user inputs for name, username and password. It then asks for the user to re-enter the password to ensure no errors were made in the password entry. It then sends the user onto their home screen.

Quick Test of the Sign Up Screen Version:

When I use my mouse to hit the sign-up button to test that the menu calls the correct function when the sign-up button is clicked, the program displays Home Page which is the expected output.

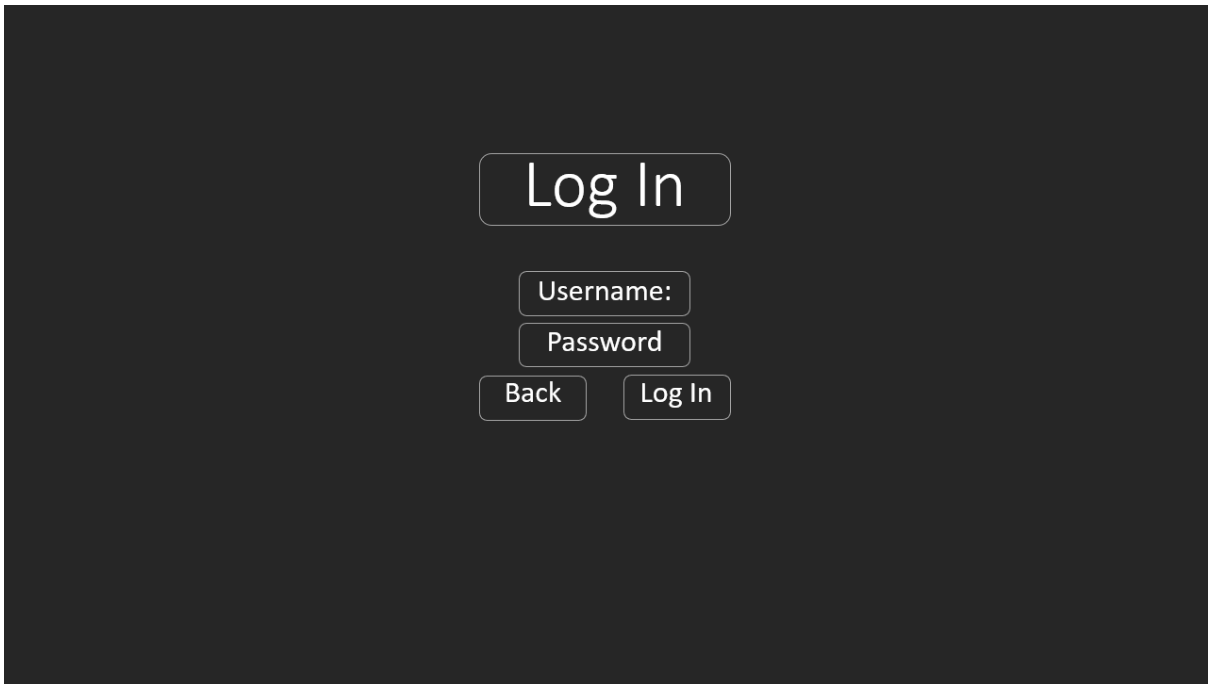
When I try to sign up a new user with the same username as an already existing user in the database, the user is not added and the program sends the user to an empty signup screen to try again.

When I use my mouse to hit the back button to test that the program will return to the welcome page, I am sent back to the home page.

I will test whether the data has been successfully added to by database through version 0.4 where I will use the data to log in.

Creating the Log In (Version 0.4):

Proposed Login Screen Design:



Overview:

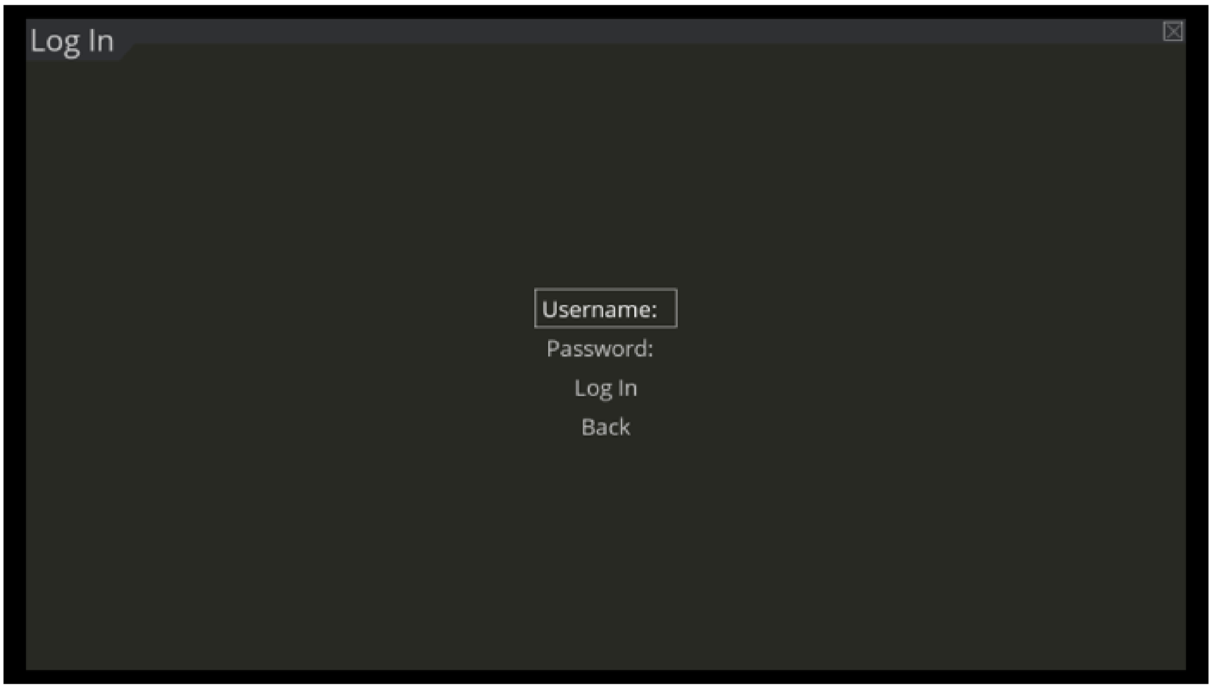
The user should be able to enter their details. If they are in the simulation database then the user should be signed in and move to their home screen.

Version 0.4 code:

```
class Login_screen():
    def __init__(self):
        SCREEN_WIDTH = window.winfo_screenwidth()
        SCREEN_HEIGHT = window.winfo_screenheight()
        surface = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT), FULLSCREEN)
        self.menu = pygame_menu.Menu(SCREEN_HEIGHT, SCREEN_WIDTH, "Log In",
theme=pygame_menu.themes.THEME_DARK)
        self.username = self.menu.add_text_input("Username:  ", default="")
        self.password = self.menu.add_text_input("Password:  ", default="", password=True)
        self.menu.add_button("Log In", self.log_in)
        self.menu.add_button("Back", self.welcome_screen)
        self.menu.mainloop(surface)

    def welcome_screen(self):
        Welcome_screen()

    def home_screen(self):
        Home_screen()

    def log_in(self):
        data = self.menu.get_input_data(recursive=False)
```

```
username = data[self.username.get_id()]
password = data[self.password.get_id()]
user = session.query(User).filter(User.username == username).one_or_none()
if user == None:
    Login_screen()
if user.password  == password:
    Program.user_logged_in = user
    self.home_screen()
else:
    Login_screen()
```

Log In Screen I created:



The solution I created accepts user inputs for username and password. It then finds the user with the given username and checks that the password is correct. It then passes the user onto their home screen.

Quick Test of the Log In Screen Version:

When I enter the login details of an existing use to test that the users were added in version 0.3 and that the log in works, the program displays Home Page, which is the expected output.

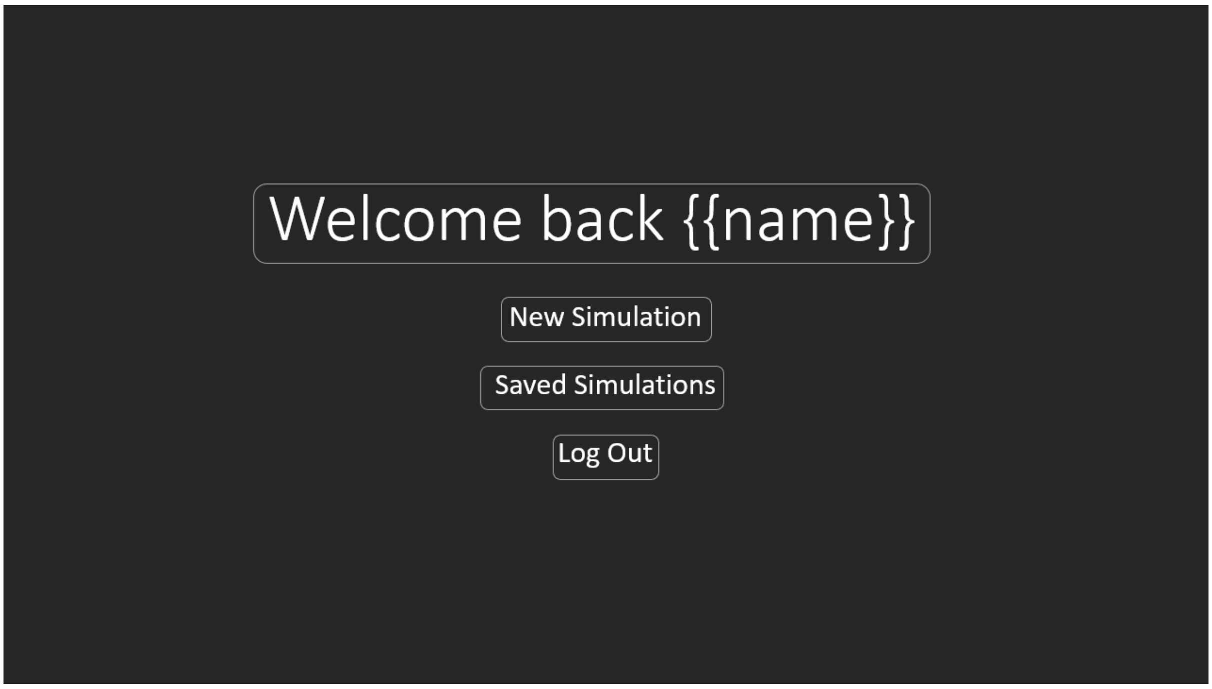
When I enter the details of a user that doesn’t exist, I am sent to an empty version of the login screen, as expected.

When I enter the username of a user that does exist with an incorrect password, I am sent to an empty version of the login screen, as expected.

When I hit the back button with my mouse I am returned to the Welcome Screen as expected

Creating the Home Screen(Version 0.5):

Proposed Home Screen Design:



Overview:

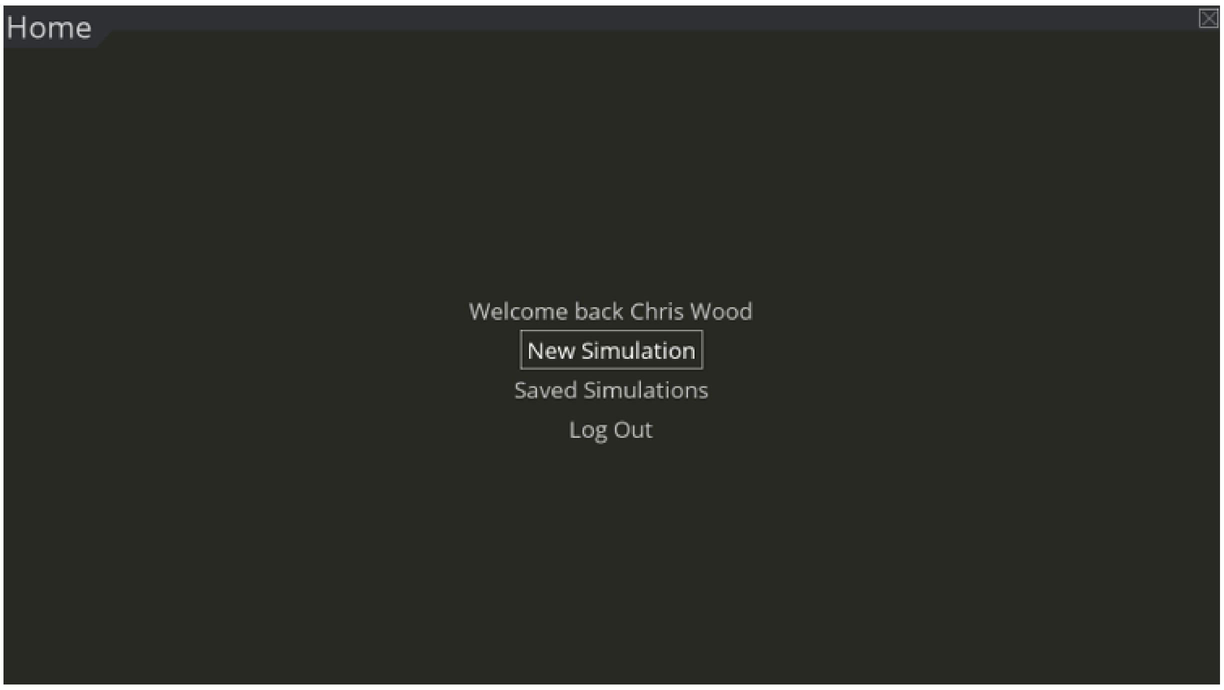
The home screen should display the user signed in name and 3 buttons as shown above. They should link to those screens.

Version 0.5 code:

```
class Home_screen():
    def __init__(self):
        SCREEN_WIDTH = window.winfo_screenwidth()
        SCREEN_HEIGHT = window.winfo_screenheight()
        surface = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT), FULLSCREEN)
        self.menu = pygame_menu.Menu(SCREEN_HEIGHT, SCREEN_WIDTH, 'Home',
theme=pygame_menu.themes.THEME_DARK)
        self.menu.add_label("Welcome back " + Program.user_logged_in.name)
        self.menu.add_button('New Simulation', self.new_simulation_screen)
        self.menu.add_button('Saved Simulations', self.saved_simulations_screen)
        self.menu.add_button('Log Out', self.log_out)
        self.menu.mainloop(surface)

    def welcome_screen(self):
        Welcome_screen()

    def new_simulation_screen(self):
        print("new simulation screen")

    def saved_simulations_screen(self):
        print("saved simulations screen")
```

```
def log_out(self):
    self.welcome_screen()
```

Home Screen I created:



Quick Test of the Home Screen Version:

The name displayed is the name of the user currently logged in so the simulation must have logged into the correct user.

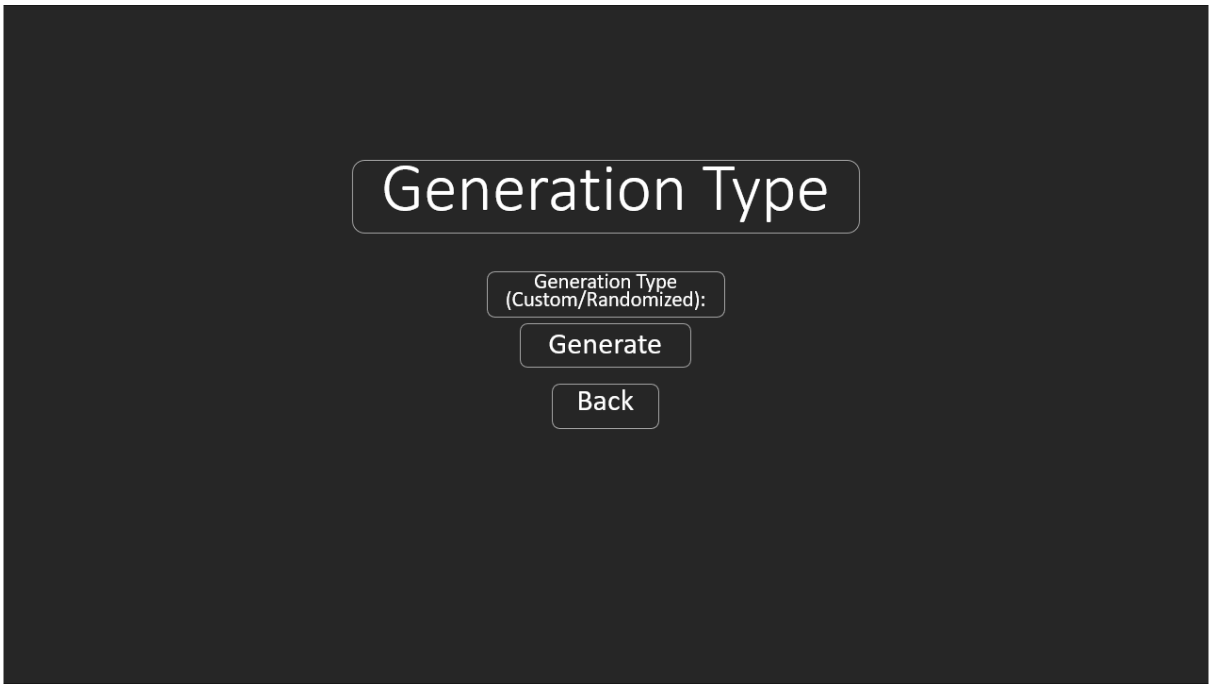
When I click the New Simulation to test that the button works, the program outputs “new simulation screen” as expected.

When I click the Saved Simulations button to test that the button works, the program outputs “saved simulations screen” as expected.

When I click the Log Out button the program logs me out and displays the welcome screen as expected.

Creating the Generation Type Selection (Version 0.6.1):

Proposed Generation Type Selector Screen Design:



Overview:

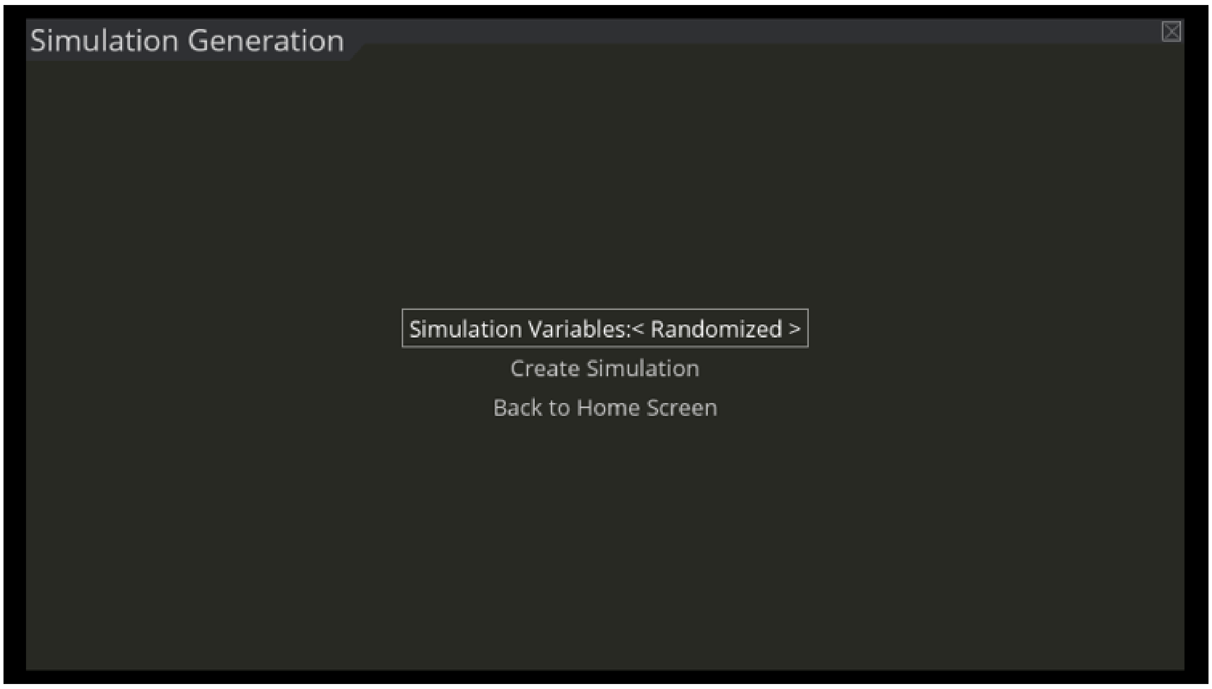
The generation screen should show a switchable button which allows the user to select the simulation creation type they wish.

Version 0.6.1 code:

```
class New_simulation_screen():
    def __init__(self):
        self.current_generation_type = "Random"
        SCREEN_WIDTH = window.winfo_screenwidth()
        SCREEN_HEIGHT = window.winfo_screenheight()
        surface = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT), FULLSCREEN)
        self.menu = pygame_menu.Menu(SCREEN_HEIGHT, SCREEN_WIDTH, 'Simulation
Generation', theme=pygame_menu.themes.THEME_DARK)
        self.menu.add_selector('Simulation Variables:', [('Randomized', 1), ('    Custom    ', 2)],
onchange=self.change_input)
        self.menu.add_button("Create Simulation", self.create_simulation)
        self.menu.add_button("Back to Home Screen", self.home_screen)
        self.menu.mainloop(surface)

    def home_screen(self):
        Home_screen()

    def change_input(self, val_1, val_2):
        if val_2 == 1:
            self.current_generation_type = "Random"
        if val_2 == 2:
            self.current_generation_type = "Custom"

    def create_simulation(self):
        if self.current_generation_type == "Custom":
            print("Custom Generate")
        elif self.current_generation_type == "Random":
            self.randomly_generate

    def randomly_generate(self):
        print("Randomly Generate")
```

This code creates the screen and also handles the selector, storing the current value of the selector as an attribute of the object. This is then used to decide whether to send the user on to a generation detail selection screen or straight to their simulation.

Generation Selector I created:



The currently selected button allows the user to change between random and custom simulation generation by clicking the Simulation Variables button.

Quick Test of the Simulation Generation Type Version:

When I click on selector with my mouse it changes from randomized to customised.

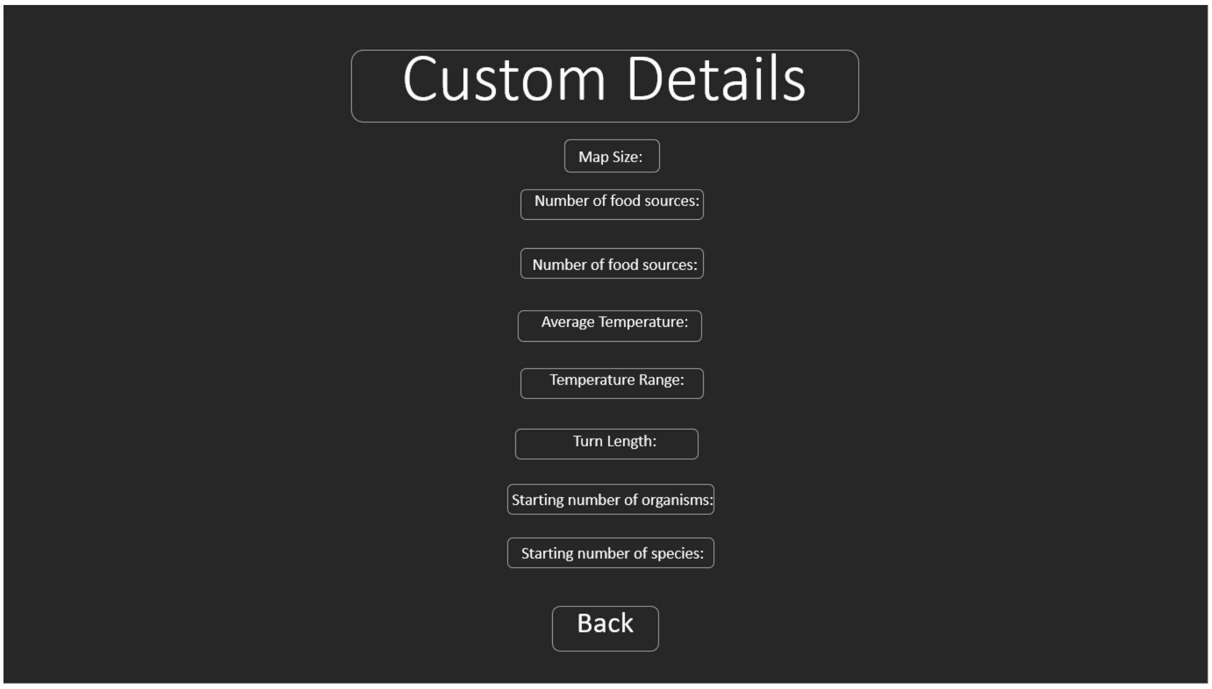
When I click generate and the selector is on the option randomized, the program outputs “Randomly Generate” as expected.

When I click generate and the selector is on the option custom, the program outputs “Custom Generate” as expected.

When I click the back button the program returns me to the home page as expected.

Creating the Custom Data Entry Screen (Version 0.6.2):

Proposed Custom Data Entry Design:



Overview:

Should allow the user to enter the details that they wish to use to for their simulation.

Version 0.6.2 code:

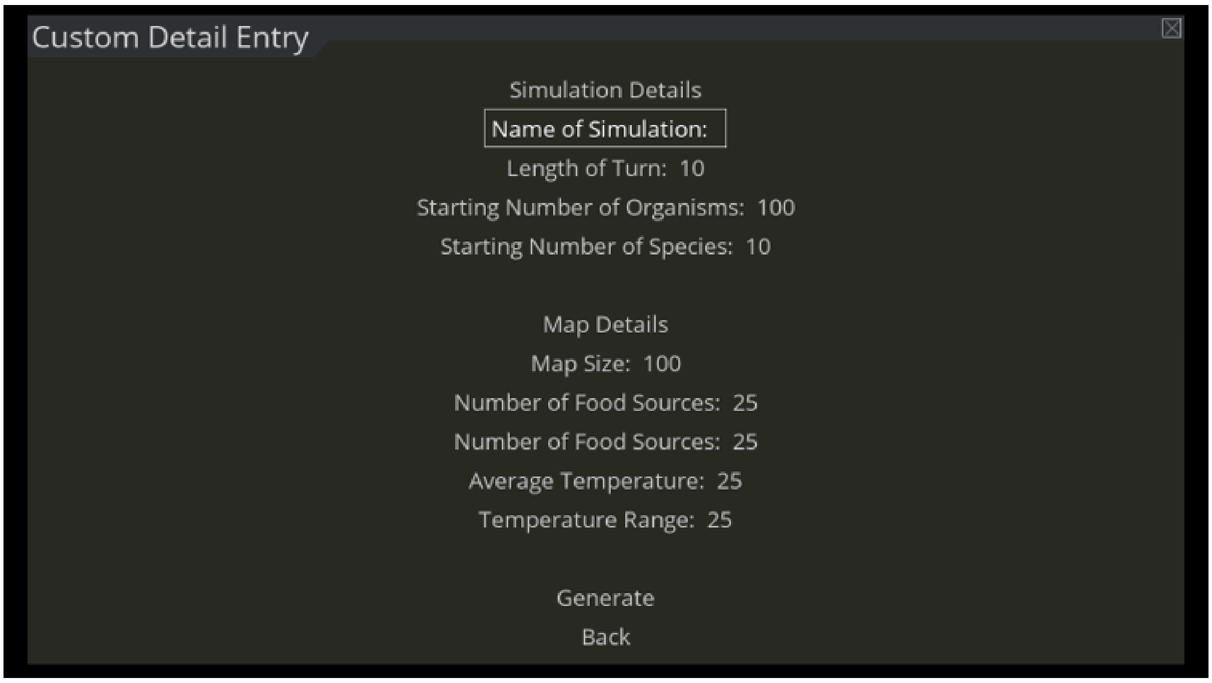
```
class Custom_Detail_Entry_screen():
    def __init__(self):
        SCREEN_WIDTH = window.winfo_screenwidth()
        SCREEN_HEIGHT = window.winfo_screenheight()
        surface = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT), FULLSCREEN)
        self.menu = pygame_menu.Menu(SCREEN_HEIGHT, SCREEN_WIDTH, "Custom Detail Entry",
theme=pygame_menu.themes.THEME_DARK)
        self.menu.add_label("Simulation Details")
        self.simulation_name = self.menu.add_text_input("Name of Simulation:  ", default="Simulation_")
        self.length_of_turn = self.menu.add_text_input("Length of Turn:  ", default="10")
        self.starting_number_of_organisms = self.menu.add_text_input("Starting Number of Organisms:
", default="100")
        self.starting_number_of_species = self.menu.add_text_input("Starting Number of Species:  ",
default="10")
        self.menu.add_label("")
        self.menu.add_label("Map Details")
        self.map_size = self.menu.add_text_input("Map Size:  ", default="10")
        self.number_of_food_sources = self.menu.add_text_input("Number of Food Sources:  ",
default="5")
        self.meat_to_veg_ratio = self.menu.add_text_input("Meat to Veg Ratio:  ", default="1:3")
        self.number_of_water_sources = self.menu.add_text_input("Number of Water Sources:  ",
default="5")
        self.menu.add_label("")
        self.menu.add_button("Generate", self.collect_input_data)
        self.menu.add_button("Back", self.home_screen)
        self.menu.mainloop(surface)

    def home_screen(self):
        Home_screen()

    def generating_screen(self):
        Simulation_Screen()

    def collect_input_data(self):
        data = self.menu.get_input_data(recursive=False)
        simulation_name = data[self.simulation_name.get_id()]
        date_created = datetime.now()
        length_of_turn = data[self.length_of_turn.get_id()]
        starting_number_of_organisms = data[self.starting_number_of_organisms.get_id()]
        starting_number_of_species = data[self.starting_number_of_species.get_id()]

        map_size = data[self.map_size.get_id()]
        number_of_food_sources = data[self.number_of_food_sources.get_id()]
        meat_to_veg_ratio = data[self.meat_to_veg_ratio.get_id()]
        number_of_water_sources = data[self.number_of_water_sources.get_id()]

        self.generate(simulation_name, date_created, length_of_turn, starting_number_of_organisms,
starting_number_of_species, map_size, number_of_food_sources, meat_to_veg_ratio,
number_of_water_sources)

    def generate(self, simulation_name, date_created, lenth_of_turn, starting_number_of_organisms,
starting_number_of_species, map_size, number_of_food_sources, meat_to_veg_ratio,
number_of_water_sources):
        current_simulations = session.query(Simulation).filter(Simulation.user_relationship ==
Program.user_logged_in).order_by(Simulation.date_last_used.desc()).all()
        if len(current_simulations) >= 3:
            session.delete(current_simulations[-1])
        simulation = Simulation(simulation_name=simulation_name, date_last_used=date_created,
current_turn=0, length_of_turn=lenth_of_turn,
starting_number_of_organisms=starting_number_of_organisms,
starting_number_of_species=starting_number_of_species,
current_number_of_organisms=starting_number_of_organisms,
current_number_of_species=starting_number_of_species, user_id=Program.user_logged_in.id,
user_relationship=Program.user_logged_in)
        map_to_add = Map(map_size=map_size, number_of_food_sources=number_of_food_sources,
meat_to_veg_ratio=meat_to_veg_ratio, number_of_water_sources=number_of_water_sources,
simulation_id=simulation.id, simulation_relationship=simulation)
        session.add(simulation)
        session.commit()

        Program.current_simulation = simulation
        self.generating_screen()
```

The code above creates the detail entry screen, collects the data inputted and adds it to the database. This creates both the simulation and the map belonging to the simulation. It also handles the deleting of the least recently used simulation (There are only ever three simulations per user).

I also made some changes to my simulation schema in the database by adding the attributes name and date_created to my simulations table.

```
class Simulation(Base):
    __tablename__ = "simulations"
    id = Column(Integer, primary_key=True)
    simulation_name = Column(String(50))
    date_last_used = Column(DateTime)
    current_turn = Column(Integer)
    length_of_turn = Column(Integer)
    starting_number_of_organisms = Column(Integer)
    starting_number_of_species = Column(Integer)
    current_number_of_organisms = Column(Integer)
    current_number_of_species = Column(Integer)

    user_id = Column(ForeignKey("users.id"))
    user_relationship = relationship("User", back_populates='simulations_relationship')

    species_relationship = relationship("Species", back_populates='simulation_relationship')

    map_relationship = relationship("Map", back_populates='simulation_relationship')

    def __repr__(self):
        return "<Simulation(id: {0}, simulation name: {1}, date last used: {2}, current turn: {3}, length of
turn: {4}, current organisms: {5}, current species: {6}, user: {7})>".format(self.id, self.simualtion_name,
self.date_created, self.current_turn, self.length_of_turn, self.current_number_of_organisms,
self.current_number_of_species, self.user_id)
```

Generation Selector I created:


Custom Detail Entry

Simulation Details

Name of Simulation:

Length of Turn: 10

Starting Number of Organisms: 100

Starting Number of Species: 10

Map Details

Map Size: 100

Number of Food Sources: 25

Number of Food Sources: 25

Average Temperature: 25

Temperature Range: 25

Generate

Back

The image above shows the detail entry screen for the user. The values already in the input space are recommended values that are displayed with the screen which can also be changed to suit the users desired environment. The recommended values will likely change once my simulation is fully written and I can find what a sensible value for each attribute is.

Quick Test of the Detail Entry Version:

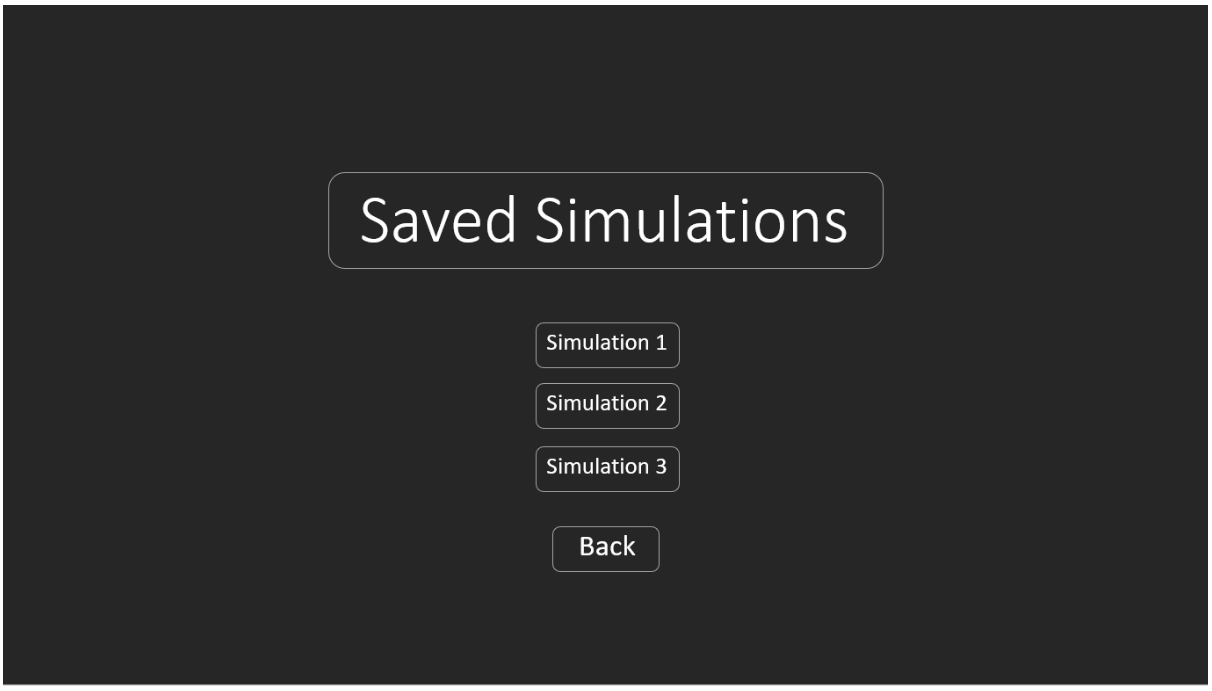
When I click generate and the selector is on the option randomized, the program outputs “Generating” as expected.

When I click the back button the program returns me to the home page as expected.

I will test whether the data has been correctly added and deleted to and from the database in version 0.6.3

Creating the Saved Simulations Screen (Version 0.6.3):

Proposed Saved Simulations Design:



Overview:

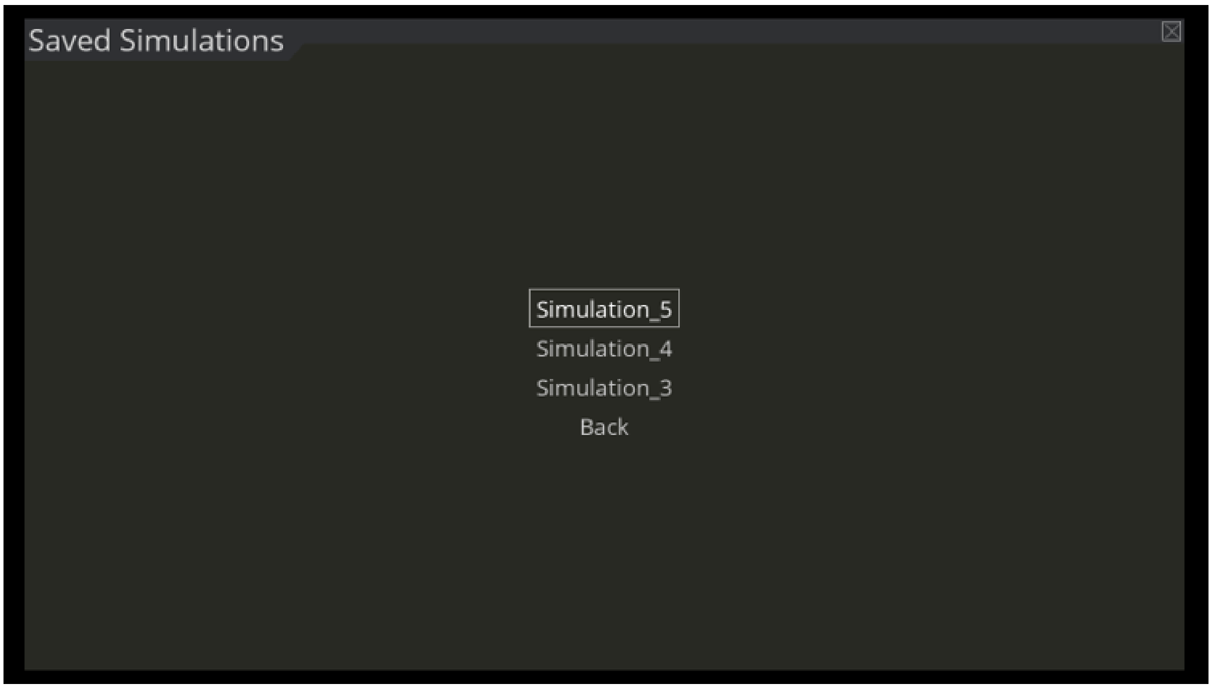
The screen should show three buttons with the names of the three most recently accessed simulations that belong to that user. When clicked it should load that simulation for running.

Version 0.6.3 code:

```
class Saved_Simulations_screen():
    def __init__(self):
        SCREEN_WIDTH = window.winfo_screenwidth()
        SCREEN_HEIGHT = window.winfo_screenheight()
        surface = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT), FULLSCREEN)
        self.menu = pygame_menu.Menu(SCREEN_HEIGHT, SCREEN_WIDTH, "Saved Simulations",
theme=pygame_menu.themes.THEME_DARK)
        user_simulations = session.query(Simulation).filter(Simulation.user_relationship ==
Program.user_logged_in).order_by(Simulation.date_last_used.desc()).all()
        if len(user_simulations) == 0:
            self.menu.add_label("You have no saved simulations")
        else:
            self.menu.add_button(user_simulations[0].simulation_name, self.load_simulation_1)
            try:
                self.menu.add_button(user_simulations[1].simulation_name, self.load_simulation_2)
            except:
                pass
            try:
                self.menu.add_button(user_simulations[2].simulation_name, self.load_simulation_3)
            except:
                pass
        self.menu.add_button("Back", self.home_screen)
        self.menu.mainloop(surface)

    def home_screen(self):
        Home_screen()

    def load_simulation_1(self):
        simulation = session.query(Simulation).filter(Simulation.user_relationship ==
Program.user_logged_in).order_by(Simulation.date_last_used.desc()).first()
        print(simulation)
        simulation.date_last_used = datetime.now()
        session.commit()

    def load_simulation_2(self):
        simulations = session.query(Simulation).filter(Simulation.user_relationship ==
Program.user_logged_in).order_by(Simulation.date_last_used.desc()).all()
        simulation = simulations[1]
        print(simulation)
        simulation.date_last_used = datetime.now()
        session.commit()

    def load_simulation_3(self):
        simulations = session.query(Simulation).filter(Simulation.user_relationship ==
Program.user_logged_in).order_by(Simulation.date_last_used.desc()).all()
        simulation = simulations[2]
        print(simulation)
        simulation.date_last_used = datetime.now()
        session.commit()
```

This code creates the saved simulations screen. It displays the three most recently used simulations and will eventually send them onto the simulation generation screen for processing and displaying for the user. It also updates the date_last_used when a simulation is reloaded by the user.

Saved Simulations Screen I created:



The image above shows the saved simulation screen for the user. The screen consists of 3 simulations which are the three stored in the database (my program will only store the last three accessed simulations). They are buttons that will in subsequent versions load the simulation of the simulation button that was clicked and will generate it for the user to resume.

Quick Test of the Saved Simulations Screen Version:

When I have no simulations, the screen shows you have no simulations and similarly when I only have one or two it will only display the one or two simulations I have. This shows that the code from version 0.6.2 correctly creates a simulation item under the name of the user that is currently logged in.

When I have three simulations, three simulation buttons are displayed.

If I have added more than three simulations then the program will only display the three most recent simulations as the other simulation has been deleted.

When I click on one of the simulation buttons, the simulation with that name is displayed as expected.

When I click on the earliest used simulation and go back to the home screen and then back into the saved simulations screen again, the simulation has moved from the bottom of the display to the top which proves that the date_last_used is correctly changed when I click the button.

When I click the back button the program returns me to the home page as expected.

Randomly Generating of a Simulation (Version 0.6.4):

Overview:

This section should randomly select the details for the simulation and create a new simulation with these details. The values for the typical user inputs are replaced by randomised values using random.randint. The value for the randomisation will likely change in order to make the simulation more streamlined and better.

Version 0.6.4 code:

```
    def randomly_generate(self):
        simulation_name = "Simulation_" + str(date.today())
        date_created = datetime.now()
        length_of_turn = random.randint(5,20)
        starting_number_of_organisms = random.randint(10,20)
        starting_number_of_species = random.randint(5,20)

        map_size = random.randint(5,20)
        number_of_food_sources = random.randint(2,10)
        meat_to_veg_ratio = str(random.randint(1,5))+":"+str(random.randint(1,5))
        number_of_water_sources = random.randint(2,10)

        simulation = Simulation(simulation_name=simulation_name, date_last_used=date_created,
current_turn=0, length_of_turn=length_of_turn,
starting_number_of_organisms=starting_number_of_organisms,
starting_number_of_species=starting_number_of_species,
current_number_of_organisms=starting_number_of_organisms,
current_number_of_species=starting_number_of_species, user_id=Program.user_logged_in.id,
user_relationship=Program.user_logged_in)
        session.add(simulation)
        session.commit()

        map_to_add = Map(map_size=map_size, number_of_food_sources=number_of_food_sources,
meat_to_veg_ratio=meat_to_veg_ratio, number_of_water_sources=number_of_water_sources,
simulation_id=simulation.id, simulation_relationship=simulation)
        session.add(map_to_add)
        session.commit()
        Program.current_simulation = simulation
        self.generate_simulation()
```

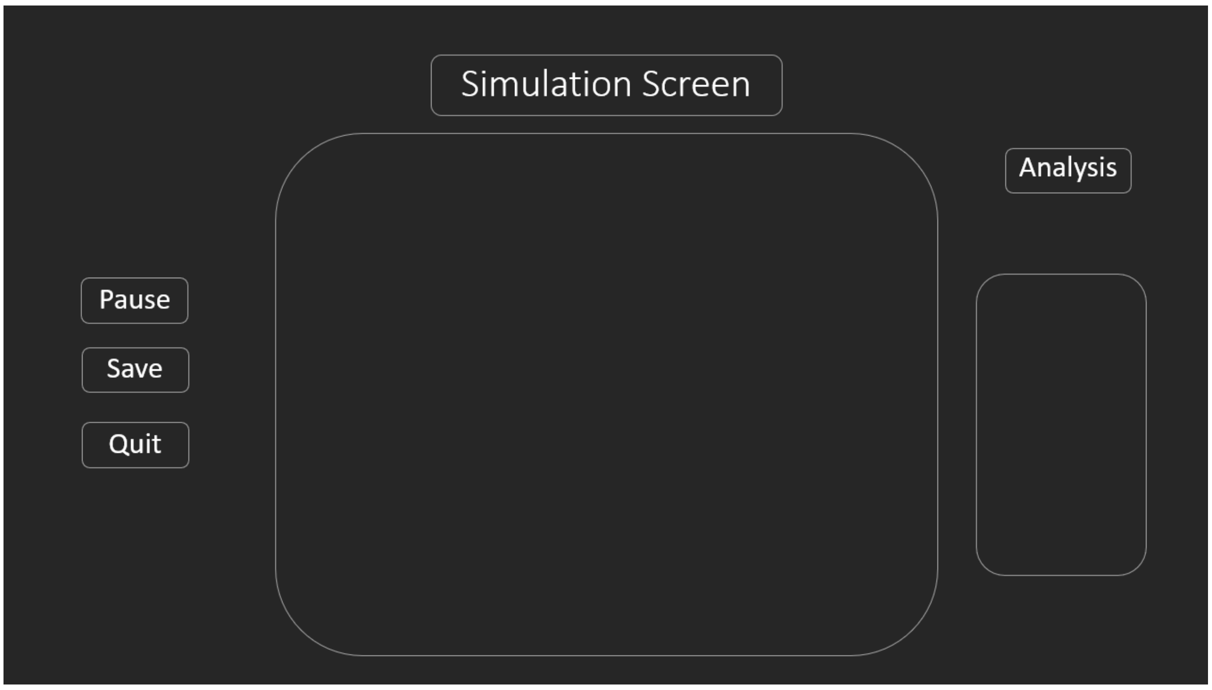
In this version of the program, I replaced the function randomly_generate in the class new simulation screen. The code randomizes the simulation details (this includes both simulation and map). The current values were picked out of what I believe would be reasonable however these are very likely to change once the program is finished in order for them to create reasonable simulation. It creates a new simulation and map and calls the function generate_simulation which I also had to add. This function currently only prints “generating simulation”, however in subsequent versions this will be implemented to display the simulation.

Quick Test of the Saved Simulations Screen Version:

When I have randomly generate selected and hit the generate button, if I go to saved simulations, I can see that a new simulation has been created with the simulation_name simulation_ followed by the date of the day that it was created.

Creating the Simulations Screen (Version 0.7.1):

Proposed Simulation Screen Design:



Overview:

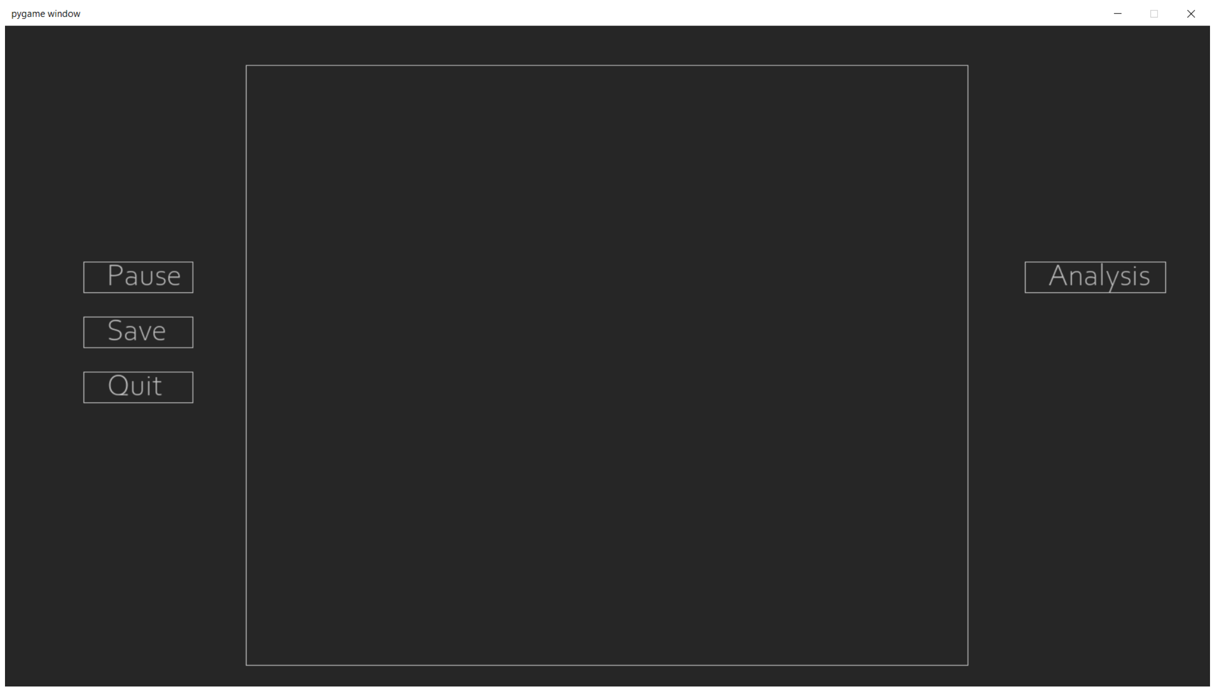
This should display the backdrop of the simulation. The buttons should all be linked to methods.

Version 0.7.1 code:

```
class Simulation_Screen():
    def __init__(self):
        self.simulation = Program.current_simulation
        self.simulation_map = self.simulation.map_relationship[0]
        self.SCREEN_WIDTH = window.winfo_screenwidth()
        self.SCREEN_HEIGHT = window.winfo_screenheight()
        self.surface = pygame.display.set_mode((self.SCREEN_WIDTH, self.SCREEN_HEIGHT))
        self.surface.fill(BLACK)

        self.SURFACE_WIDTH = int(self.SCREEN_WIDTH*(3/5))
        self.SURFACE_HEIGHT = int(self.SCREEN_HEIGHT-100)
        surface=self.surface
        self.simulation_surface = pygame.surface.Surface((self.SURFACE_WIDTH,
self.SURFACE_HEIGHT))
        self.simulation_surface.fill(BLACK)

        self.block_size = int(self.SURFACE_HEIGHT/self.simulation_map.map_size)

        self.pause_button_position = [100,300]
        self.save_button_position = [100, 370]
        self.quit_button_position = [100,440]
        self.analysis_button_position = [1300,300]

        smallfont = pygame.font.SysFont("Corbel",40)
        self.pause_text = smallfont.render("Pause" , True , WHITE)
        self.save_text = smallfont.render("Save" , True , WHITE)
        self.quit_text = smallfont.render("Quit", True, WHITE)
        self.analysis_text = smallfont.render("Analysis", True, WHITE)

        self.x_coordinate_values = {}
        self.y_coordinate_values = {}

        self.draw_border()
        self.draw_buttons()

        while True:
            mouse = pygame.mouse.get_pos()
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    pygame.quit()
                    sys.exit()

                if event.type == pygame.MOUSEBUTTONDOWN:
                    if self.pause_button_position[0] <= mouse[0] <= self.pause_button_position[0]+140 and
self.pause_button_position[1] <= mouse[1] <= self.pause_button_position[1]+40:
                        self.pause()

                    if self.save_button_position[0] <= mouse[0] <= self.save_button_position[0]+140 and
self.save_button_position[1] <= mouse[1] <= self.save_button_position[1]+40:
                        self.save()

                    if self.quit_button_position[0] <= mouse[0] <= self.quit_button_position[0]+140 and
self.quit_button_position[1] <= mouse[1] <= self.quit_button_position[1]+40:
                        self.quit_simulation()

                    if self.analysis_button_position[0] <= mouse[0] <= self.analysis_button_position[0]+180
and self.analysis_button_position[1] <= mouse[1] <= self.analysis_button_position[1]+40:
                        self.analysis()

            pygame.display.update()

    def draw_border(self):
        border = pygame.Rect(0, 0, self.SURFACE_WIDTH, self.SURFACE_HEIGHT)
        pygame.draw.rect(self.simulation_surface, WHITE, border, 1)
self.surface.blit(self.simulation_surface, (int(self.SCREEN_WIDTH*(1/5)),50))
```

```
    def draw_buttons(self):

pygame.draw.rect(self.surface,WHITE,[self.pause_button_position[0],self.pause_button_position[1],14
0,40],1)
        self.surface.blit(self.pause_text ,
(self.pause_button_position[0]+30,self.pause_button_position[1]))


pygame.draw.rect(self.surface,WHITE,[self.save_button_position[0],self.save_button_position[1],140,
40],1)
        self.surface.blit(self.save_text , (self.save_button_position[0]+30,self.save_button_position[1]))


pygame.draw.rect(self.surface,WHITE,[self.quit_button_position[0],self.quit_button_position[1],140,40]
,1)
        self.surface.blit(self.quit_text , (self.quit_button_position[0]+30,self.quit_button_position[1]))


pygame.draw.rect(self.surface,WHITE,[self.analysis_button_position[0],self.analysis_button_position[
1],180,40],1)
        self.surface.blit(self.analysis_text ,
(self.analysis_button_position[0]+30,self.analysis_button_position[1]))

    def pause(self):
        print("Pause")

    def save(self):
        print("Save")

    def quit_simulation(self):
        Program.current_simulation = None
        Home_screen()

    def analysis(self):
        print("Analysis")
```

Simulation Screen I created:



The image above shows the simulation screen.It consists of the simulation in the central square and some smaller buttons around this centre display. These will eventually be linked up once the simulation is written (The quit button is currently half introduced).

Quick Test of the Simulations Screen Version:

When I hit the quit button, the program will quit the simulation screen.

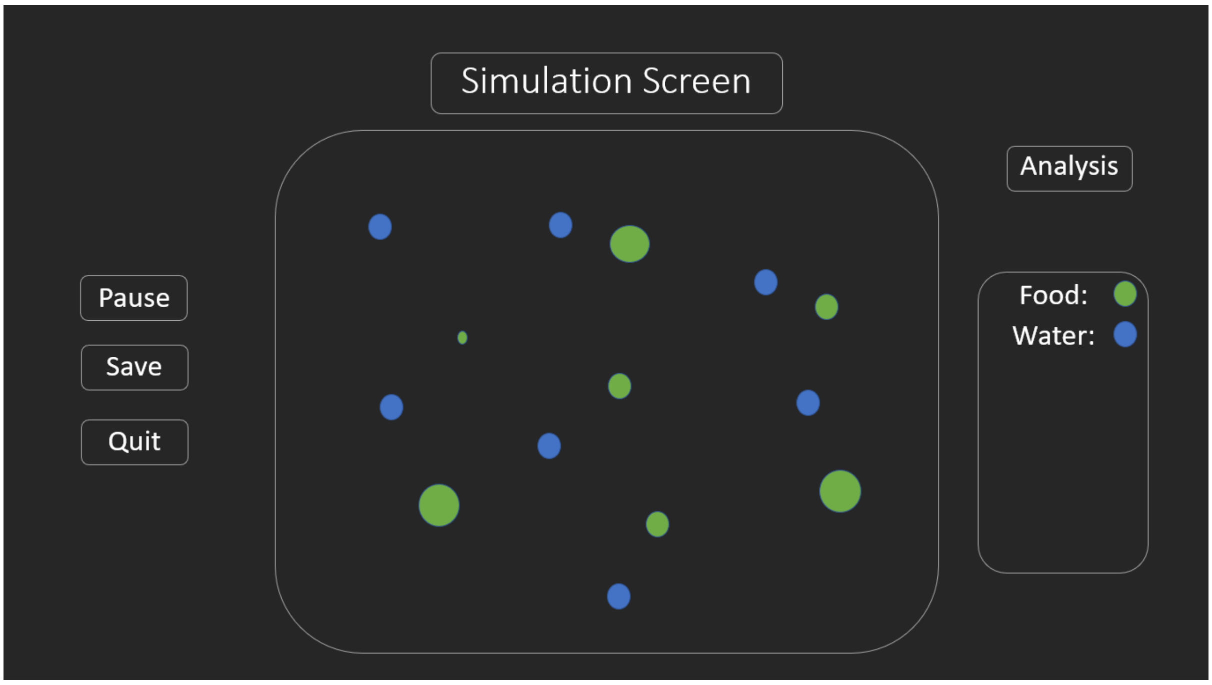
When I hit the pause button, the program prints “Pause” as expected.

When I hit the save button, the program prints “Save” as expected.

When I hit the analysis button, the program prints “Analysis” as expected.

Setting up the Map on Simulation Screen (Version 0.7.2):

Proposed Simulation Screen Design:



Overview:

The idea is that my simulation will have 4 different food types. There will be water, which always supplies a single resource. I also have veg which will provide the organisms the first n value of its nutritional code where n changes based on how well adapted that organism is to consuming the veg food type. This is a non depletable resource (it doest run out). The final food type, meat is initially set up with a nutritional value and works similarly to the fruit food type, except that once the nutritional value reaches 0 the food item is deleted.

New meat items are created whenever an organism is killed/dies. When this happens, its chromosome will become the nutritional value of the new meat item. These food items apart from the water, will all be stored in the database. In this version I will create and display the tiles. The nutritional values and the deleting of meat will all be carried out in the eating food section (version 0.9.4)

The simulation will use Tile objects which inherit from Simulation_Tile to display the items on the tiles via the draw_tile method. This will allow me to have an array of Tile objects which I can just loop through and call draw_tile on.

Algorithm Explanation:

The method set_up_grid - This will create a Tile object in the database for each of the tiles that I plan to display on the map.

The method position_water - This will find a blank tile and it will then change the tile type of that tile within the database to water

The method position_veg - This will find a blank tile and will then change the tile type of that tile within the database to Veg. It will also create a Food object and establish the relationship between the Tile and Food objects within the database.

The method position_meat - This will perform a task similar to that of position_veg.

The method create_tile - This will go through each of the tiles in the database that belong to the current simulation. It will then run through them creating the appropriate Tile from Simulation Tile and its children. It does so by using the tile_type that is stored in the database for each tile.

The method draw_tiles - This will loop through all of the items in the tile_array. It then calls the method .draw_tile for each. Since the child Tile classes inherit from Simulation_Tile, they will all have the method draw_tile.

The method draw_buttons - This will use the button positions as set in __init__ to place the buttons in the correct place on the screen.

The method draw_tile in the Tile classes - This will display the image of the tile (as set in __init__) in the correct spot on the screen.

Version 0.7.2 code:

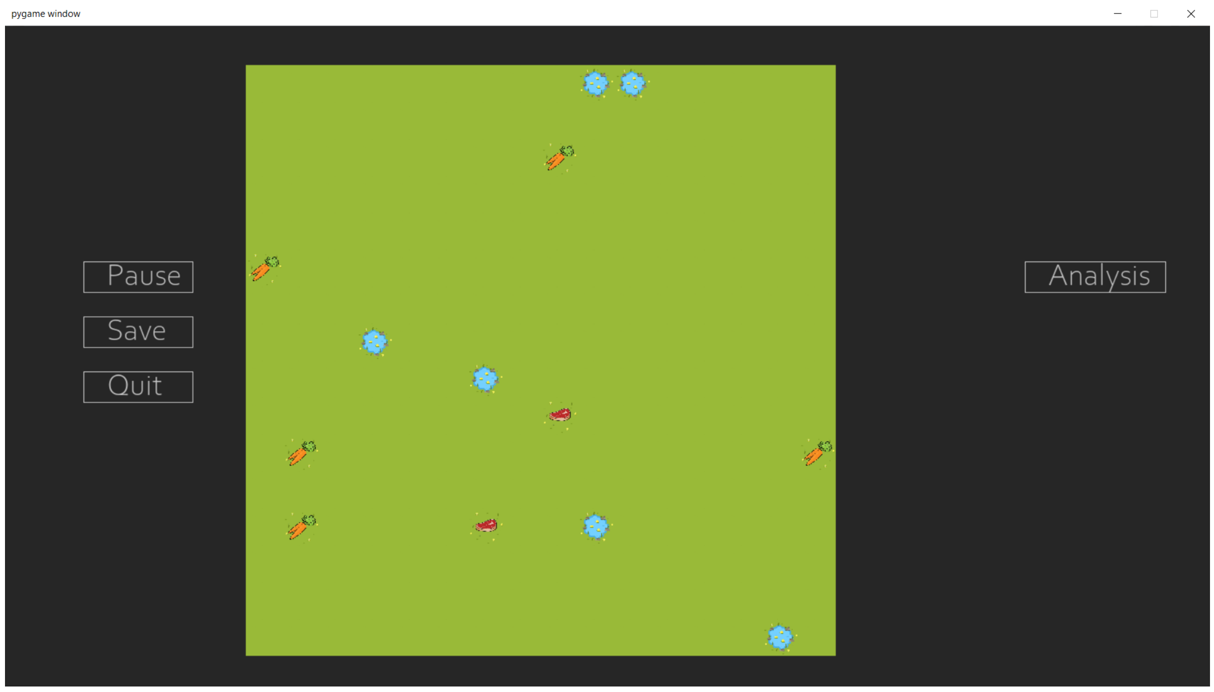
```
class Simulation_Screen():
    def __init__(self):
        self.simulation = Program.current_simulation
        self.simulation_map = self.simulation.map_relationship[0]
        self.map_size = self.simulation_map.map_size

        self.screen_width = window.winfo_screenwidth()
        self.screen_height = window.winfo_screenheight()
        self.surface = pygame.display.set_mode((self.screen_width, self.screen_height))
        self.surface.fill(BLACK)

        self.surface_height = int(self.screen_height-100)
        self.block_size = int(self.surface_height/self.simulation_map.map_size)
        self.surface_width = int(self.block_size*self.simulation_map.map_size)

        surface=self.surface
        self.simulation_surface = pygame.surface.Surface((self.surface_width, self.surface_height))
        self.simulation_surface.fill(BLACK)

        self.block_size = int(self.surface_height/self.simulation_map.map_size)

        self.pause_button_position = [100,300]
        self.save_button_position = [100, 370]
        self.quit_button_position = [100,440]
        self.analysis_button_position = [1300,300]

        smallfont = pygame.font.SysFont("Corbel",40)
        self.pause_text = smallfont.render("Pause" , True , WHITE)
        self.save_text = smallfont.render("Save" , True , WHITE)
        self.quit_text = smallfont.render("Quit", True, WHITE)
        self.analysis_text = smallfont.render("Analysis", True, WHITE)

        self.tile_array = [[None for i in range(self.map_size)] for j in range(self.map_size)]

        if len(self.simulation.tile_relationship) == 0:
            self.set_up_grid()
            self.position_water()
            self.position_veg()
            self.position_meat()
        self.create_tiles()
        self.draw_buttons()
        self.draw_tiles()


        while True:
            mouse = pygame.mouse.get_pos()
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    pygame.quit()
                    sys.exit()

                if event.type == pygame.MOUSEBUTTONDOWN:
                    if self.pause_button_position[0] <= mouse[0] <= self.pause_button_position[0]+140 and
self.pause_button_position[1] <= mouse[1] <= self.pause_button_position[1]+40:
                        self.pause()

                    if self.save_button_position[0] <= mouse[0] <= self.save_button_position[0]+140 and
self.save_button_position[1] <= mouse[1] <= self.save_button_position[1]+40:
```

```
                        self.save()

                    if self.quit_button_position[0] <= mouse[0] <= self.quit_button_position[0]+140 and
self.quit_button_position[1] <= mouse[1] <= self.quit_button_position[1]+40:
                        self.quit_simulation()

                    if self.analysis_button_position[0] <= mouse[0] <= self.analysis_button_position[0]+180
and self.analysis_button_position[1] <= mouse[1] <= self.analysis_button_position[1]+40:
                        self.analysis()

            pygame.display.update()

    def set_up_grid(self):
        for x in range(self.map_size):
            for y in range(self.map_size):
                tile = Tile(x_coordinate=x, y_coordinate=y, x_position=x*self.block_size,
y_position=y*self.block_size, tile_type="Blank", simulation_relationship=self.simulation,
food_relationship=None, organism_relationship=None)
                session.add(tile)
                session.commit()

    def position_water(self):
        for i in range(self.simulation_map.number_of_water_sources):
            x = random.randint(0, self.map_size-1)
            y = random.randint(0, self.map_size-1)
            tile_at_xy = session.query(Tile).filter(Tile.x_coordinate == x).filter(Tile.y_coordinate ==
y).filter(Tile.simulation_relationship == Program.current_simulation).one()
            while tile_at_xy.tile_type != "Blank":
                x = random.randint(0, self.map_size-1)
                y = random.randint(0, self.map_size-1)
                tile_at_xy = session.query(Tile).filter(Tile.x_coordinate == x).filter(Tile.y_coordinate ==
y).filter(Tile.simulation_relationship == Program.current_simulation).one()
            tile_at_xy.tile_type = "Water"
            session.commit()

    def position_veg(self):
        proportion_of_veg =
int(self.simulation_map.meat_to_veg_ratio.split(":")[1])/(int(self.simulation_map.meat_to_veg_ratio.spli
t(":")[0])+int(self.simulation_map.meat_to_veg_ratio.split(":")[1]))
        for i in range(round(self.simulation_map.number_of_food_sources*proportion_of_veg)):
            x = random.randint(0, self.map_size-1)
            y = random.randint(0, self.map_size-1)
            tile_at_xy = session.query(Tile).filter(Tile.x_coordinate == x).filter(Tile.y_coordinate ==
y).filter(Tile.simulation_relationship == Program.current_simulation).one()
            while tile_at_xy.tile_type != "Blank":
                x = random.randint(0, self.map_size-1)
                y = random.randint(0, self.map_size-1)
                tile_at_xy = session.query(Tile).filter(Tile.x_coordinate == x).filter(Tile.y_coordinate ==
y).filter(Tile.simulation_relationship == Program.current_simulation).one()
            tile_at_xy.tile_type = "Veg"
            veg_item = Food(food_type="Veg", nutritional_value=("a" +
random.choice(["b","b","b","b","bb"]) + "d"*random.randint(0,1)), map_id=self.simulation_map.id)
            session.add(veg_item)
            print(veg_item)
            session.commit()

    def position_meat(self):
        proportion_of_meat =
int(self.simulation_map.meat_to_veg_ratio.split(":")[0])/(int(self.simulation_map.meat_to_veg_ratio.spli
t(":")[0])+int(self.simulation_map.meat_to_veg_ratio.split(":")[1]))
```

```
        for i in range(round(self.simulation_map.number_of_food_sources*proportion_of_meat)):
            x = random.randint(0, self.map_size-1)
            y = random.randint(0, self.map_size-1)
            tile_at_xy = session.query(Tile).filter(Tile.x_coordinate == x).filter(Tile.y_coordinate ==
y).filter(Tile.simulation_relationship == Program.current_simulation).one()
            while tile_at_xy.tile_type != "Blank":
                x = random.randint(0, self.map_size-1)
                y = random.randint(0, self.map_size-1)
                tile_at_xy = session.query(Tile).filter(Tile.x_coordinate == x).filter(Tile.y_coordinate ==
y).filter(Tile.simulation_relationship == Program.current_simulation).one()
            tile_at_xy.tile_type = "Meat"
            meat_item = Food(food_type="Meat", nutritional_value=("a" +
random.choice(["c","c","c","c","cc"]) + "d"*random.randint(0,1)), map_id=self.simulation_map.id)
            session.add(meat_item)
            print(meat_item)
            session.commit()

    def create_tiles(self):
        for x in range(self.map_size):
            for y in range(self.map_size):
                tile_at_xy = session.query(Tile).filter(Tile.x_coordinate == x).filter(Tile.y_coordinate ==
y).filter(Tile.simulation_relationship == Program.current_simulation).one()
                if tile_at_xy.tile_type == "Blank":
                    self.tile_array[x][y] = Simulation_Tile(self.surface, self.simulation_surface, self.block_size,
x*self.block_size, y*self.block_size, self.screen_width)
                elif tile_at_xy.tile_type == "Water":
                    self.tile_array[x][y] = Water_Tile(self.surface, self.simulation_surface, self.block_size,
x*self.block_size, y*self.block_size, self.screen_width)
                elif tile_at_xy.tile_type == "Veg":
                    self.tile_array[x][y] = Veg_Tile(self.surface, self.simulation_surface, self.block_size,
x*self.block_size, y*self.block_size, self.screen_width)
                elif tile_at_xy.tile_type == "Meat":
                    self.tile_array[x][y] = Meat_Tile(self.surface, self.simulation_surface, self.block_size,
x*self.block_size, y*self.block_size, self.screen_width)

    def draw_tiles(self):
        for x in range(self.map_size):
            for y in range(self.map_size):
                self.tile_array[x][y].draw_tile()

    def draw_buttons(self):

pygame.draw.rect(self.surface,WHITE,[self.pause_button_position[0],self.pause_button_position[1],14
0,40],1)
        self.surface.blit(self.pause_text ,
(self.pause_button_position[0]+30,self.pause_button_position[1]))


pygame.draw.rect(self.surface,WHITE,[self.save_button_position[0],self.save_button_position[1],140,
40],1)
        self.surface.blit(self.save_text , (self.save_button_position[0]+30,self.save_button_position[1]))


pygame.draw.rect(self.surface,WHITE,[self.quit_button_position[0],self.quit_button_position[1],140,40]
,1)
        self.surface.blit(self.quit_text , (self.quit_button_position[0]+30,self.quit_button_position[1]))


pygame.draw.rect(self.surface,WHITE,[self.analysis_button_position[0],self.analysis_button_position[
1],180,40],1)
```

```
        self.surface.blit(self.analysis_text ,
(self.analysis_button_position[0]+30,self.analysis_button_position[1]))

    def pause(self):
        print("Pause")

    def save(self):
        print("Save")

    def quit_simulation(self):
        Program.current_simulation = None
        Home_screen()

    def analysis(self):
        print("Analysis")

class Simulation_Tile:
    def __init__(self, surface, simulation_surface, block_size, x_position, y_position, screen_width):
        self.surface = surface
        self.simulation_surface = simulation_surface
        self.block_size = block_size
        self.x_position = x_position
        self.y_position = y_position
        self.screen_width = screen_width
        self.image = "Grass.png"
        self.tile_type = "Blank"

    def draw_tile(self):
        image_to_draw = pygame.image.load(self.image).convert_alpha()
        scaled_image = pygame.transform.scale(image_to_draw, (self.block_size, self.block_size))
        self.simulation_surface.blit(scaled_image, pygame.Rect(self.x_position, self.y_position,
self.block_size, self.block_size).topleft)
        self.surface.blit(self.simulation_surface, (int(self.screen_width*(1/5)),50))

class Water_Tile(Simulation_Tile):
    def __init__(self, surface, simulation_surface, block_size, x_position, y_position, screen_width):
        super().__init__(surface, simulation_surface, block_size, x_position, y_position, screen_width)
        self.image = "Water_on_grass.png"
        self.tile_type = "Water"

class Veg_Tile(Simulation_Tile):
    def __init__(self, surface, simulation_surface, block_size, x_position, y_position, screen_width):
        super().__init__(surface, simulation_surface, block_size, x_position, y_position, screen_width)
        self.image = "Veg_on_grass.png"
        self.tile_type = "Veg"

class Meat_Tile(Simulation_Tile):
    def __init__(self, surface, simulation_surface, block_size, x_position, y_position, screen_width):
        super().__init__(surface, simulation_surface, block_size, x_position, y_position, screen_width)
        self.image = "Meat_on_grass.png"
        self.tile_type = "Meat"
```

Simulation Screen I created:



The image above shows the simulation screen with the food and water added. This will act as the map on which the simulation takes place.

I found the images I used at the following addresses. I then changed the background to make the colours match:

- https://www.alamy.com/stock-photo-cartoon-vector-garden-pond-illustration-with-water-plants-and-animals-148311766.html
- https://www.pinterest.co.uk/pin/597008494332118089/
- https://www.jing.fm/iclip/TbThwx_clipart-of-protein-meat-and-beef/

Quick Test of the Simulations Screen Version:

The tiles size is proportional to the screen's size and to the number of tiles in the map. The size of the tiles changes as the number of tiles is increased.

The correct number of water tiles are drawn.

There are no overlapping tiles.

The screen displays the correct number of meat and veg tiles according to the number of food tiles and the ratio of meat to veg. This also changes when the values are changed.

Creating Species for the Simulation (Version 0.8.1):

Overview:

In this version, I will create a standardised initial species chromosome. I will then randomly select out of the list of starting standard chromosomes. Once the whole simulation is working I can add more standard chromosomes to increase diversity if needed.

The structure of my chromosome will be defined as follows:

Tags | Control | Exchange (Possible Extension)

These sections are then split into different genes. Each of the genes will have a specific function. (The processes which use these genes will all be explained within their respective versions). New genes may be added in later versions.

The tags section is to be subdivided into the following sections:

1. Attack Tag - This is the tag that the organism will use to attack other organisms with.
2. Defence Tag - This is the tag that the organism will use to defend itself from other organisms that attack it.

The control section is to be subdivided into the following sections:

1. Reproduction Code - This is the tag that the organism will use to try and reproduce with other organisms.
2. Hide Tag - This is the tag that the organism will use to hide from other organisms that are trying to attack it.
3. Consumer Code - This is the tag that the organism will use to eat food.

The exchange section is used for an organism to create resources that it needs. I will add this as an extension if there is time at the end of development.

I will then randomize the chromosome letter by letter in order to make every simulation slightly different to start with. This randomisation will be carried out via the randomise_chromosome method. The randomisation will be carried out using random.randint, the values used for the randomisation will likely change during testing to make the simulation work better.

The data for each species will then be stored in my database. In this version I also implement the method find_colour, which is a way of finding the colour for the organism according to the different sections of its chromosome.

Algorithm Explanation:

The method create_species - In this method, the program will randomly select from two premade chromosomes that I designed to work fairly well with the simulation. It will then split the selected chromosome into the three different sections (Tags,Control,Exchange) and will pass these onto randomize_chromosome method. This will return a new version of the chromosome which has been randomised. The program will then create a new species in the database where the average_chromosome is the chromosome we just created.

The method randomise_chromosome - This method will take the different sections of the chromosome as input. It will then go through each of these sections, changing each of the characters with a set probability. It will then assemble the new chromosome and return it.

The method find_colour - This method also takes the three different sections of the chromosome as an input. Each of the 3 sections will encode the 3 different values of RGB colours. The R value will be calculated using the Tags section, the G value will be calculated using the Control section and the B value will be calculated using the Exchange section. The method will calculate each of these values by adding the ascii values of each of the letters and then finding the remainder when divided by 256. The value returned will be a string of the form (number, number, number).

Version 0.8.1 code:

```
def create_species(self):
    standard_chromosomes = ["abbaca,aabab|b##db|abccc","bbadbc,abaa|a|ac,ac"]
    for species in range(self.simulation.starting_number_of_species):
        new_species_chromosome =
standard_chromosomes[random.randint(0,len(standard_chromosomes)-1)]
        tag_section = new_species_chromosome.split("|")[0]
        control_section = new_species_chromosome.split("|")[1]
        exchange_section = new_species_chromosome.split("|")[2]
        new_species_chromosome = self.randomize_chromosome(tag_section, control_section,
exchange_section)
        tag_section = new_species_chromosome.split("|")[0]
        control_section = new_species_chromosome.split("|")[1]
        exchange_section = new_species_chromosome.split("|")[2]
        new_species_colour = self.find_colour(tag_section, control_section, exchange_section)
        new_species_reproduction_code = "1"
        new_species = Species(average_reproduction_code=new_species_reproduction_code,
average_chromosome=new_species_chromosome, average_species_colour=new_species_colour,
simulation_id=self.simulation.id)
        session.add(new_species)
        session.commit()

  def randomize_chromosome(self, tag_section, control_section, exchange_section):
    new_chromosome = ""
    for character in list(tag_section):
        if character == ",":
            new_chromosome+=","
        else:
            random_value = random.randint(0,10)
            if random_value<8:
                new_chromosome+=character
            elif character == "a":
                available_letters = ["b","c","d"]
                new_chromosome+=available_letters[random_value-8]
            elif character == "b":
                available_letters = ["a","c","d"]
                new_chromosome+=available_letters[random_value-8]
            elif character == "c":
                available_letters = ["a","b","d"]
                new_chromosome+=available_letters[random_value-8]
            elif character == "d":
                available_letters = ["a","b","c"]
                new_chromosome+=available_letters[random_value-8]
    new_chromosome+="|"
    for character in list(control_section):
        if character == ",":
            new_chromosome+=","
        else:
            random_value = random.randint(0,20)
            if random_value<16:
                new_chromosome+=character
            elif character == "a":
                available_letters = ["b","b","d","c","#"]
                new_chromosome+=available_letters[random_value-16]
            elif character == "b":
                available_letters = ["c","c","a","d","#"]
                new_chromosome+=available_letters[random_value-16]
            elif character == "c":
                available_letters = ["d","d","b","a","#"]
                new_chromosome+=available_letters[random_value-16]
            elif character == "d":
                available_letters = ["a","a","c","b","#"]
                new_chromosome+=available_letters[random_value-16]
            elif character == "#":
                available_letters = ["a","b","c","d","#"]
                new_chromosome+=available_letters[random_value-16]
    new_chromosome+="|"
    for character in list(exchange_section):
        if character == ",":
            new_chromosome+=","
        else:
            random_value = random.randint(0,20)
            if random_value<16:
                new_chromosome+=character
            elif character == "a":
                available_letters = ["b","b","d","c","#"]
                new_chromosome+=available_letters[random_value-16]
            elif character == "b":
                available_letters = ["c","c","a","d","#"]
                new_chromosome+=available_letters[random_value-16]
            elif character == "c":
                available_letters = ["d","d","b","a","#"]
                new_chromosome+=available_letters[random_value-16]
            elif character == "d":
                available_letters = ["a","a","c","b","#"]
                new_chromosome+=available_letters[random_value-16]
            elif character == "#":
                available_letters = ["a","b","c","d","#"]
                new_chromosome+=available_letters[random_value-16]
    return(new_chromosome)

def find_colour(self, tag_section, control_section, exchange_section):
```

```
R_value = 0
for value in list(tag_section):
    if value != ",":
        R_value += ord(value)
R_value = R_value%256
G_value = 0
for value in list(control_section):
    if value != ",":
        G_value += ord(value)
G_value = G_value%256
B_value = 0
for condition in list(exchange_section):
    for value in list(condition):
        if value != ",":
            B_value += ord(value)
B_value = G_value%256
return("("+str(R_value)+","+str(G_value)+","+str(B_value)+")")
```

Quick Test of the Species Creation Version:

When I create a new simulation, the correct number of species are displayed according to the number of species entered in the simulation detail entry screen.

The RGB values of each species are different since they have different genes.

Within each species' colour, I noticed that the G value and B value were always the same which was unexpected. When I looked over my code I recognised a misspelling in the following line:

```
B_value = G_value%256
```

Which I replaced with:

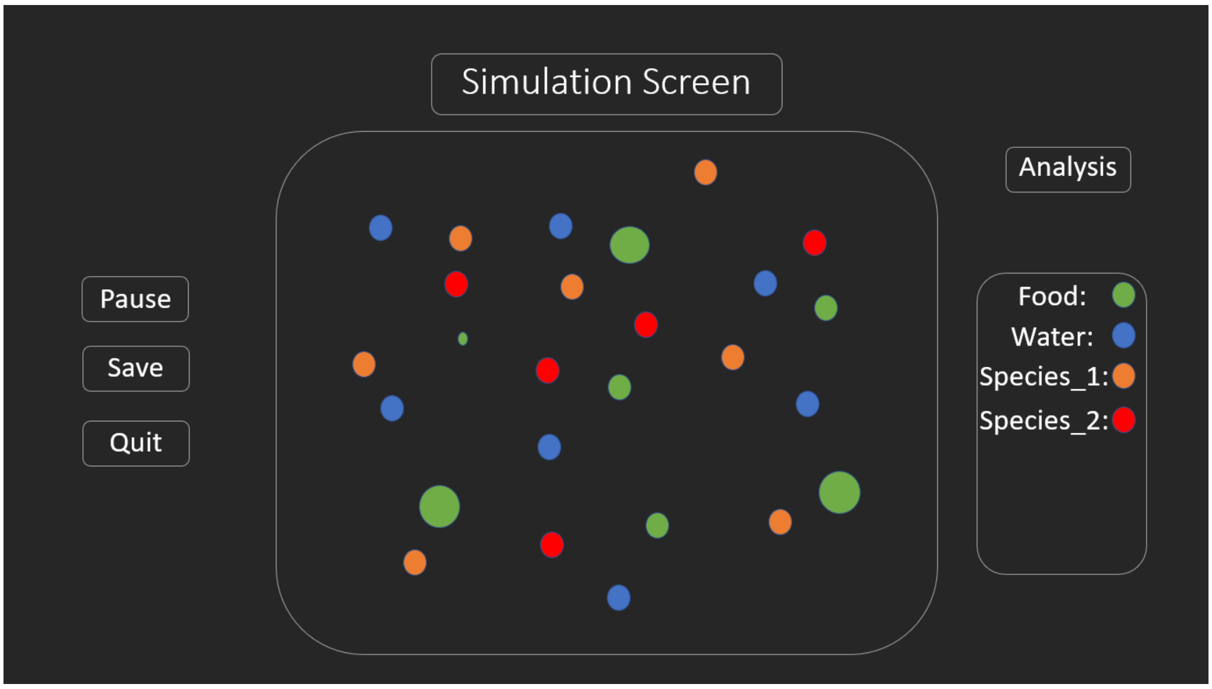
```
B_value = B_value%256
```

Creating and Displaying Organisms (Version 0.8.2 and 0.8.3):

Overview:

In this version I will display the organisms on the simulation screen that I created. Their positions and data will be collected from the database and then correctly displayed.

Proposed Simulation Screen Design:



Algorithm Explanation:

The method create_organisms - In this method, the program will cycle through all of the organisms that have to be made. It will then randomly try different tile positions until the tile on that position has a tile type “Blank”. It will then, according to their species, create them, setting their chromosomes equal to the species they belong to average_chromosome.

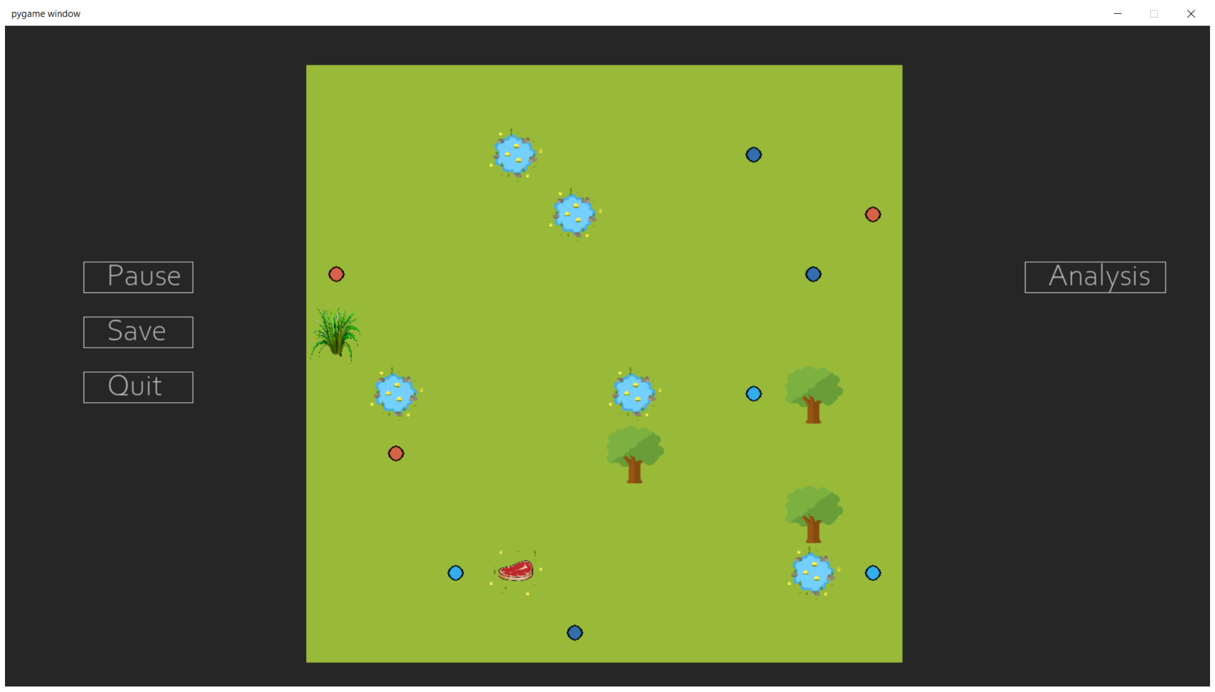
I have also created the new class Organism Tile which inherits from Simulation_Tile. The method draw_organism, will select the colour of the organism on that tile and will draw a circle of that colour on the screen at the correct position. This method does not overwrite the draw_tile method from Simulation Tile. The draw_tile is still used but the image is set to blank.png so the tile looks to be a photo of the organism on the green background but is instead just an overlay on top.

Version 0.8.2 and 0.8.3 code:

```
def create_organisms(self):
    species = session.query(Species).filter(Species.simulation_relationship==self.simulation).all()
    for current_species in species:
        for i in
range(int(self.simulation.starting_number_of_organisms/self.simulation.starting_number_of_species)):
            x = random.randint(0, self.map_size-1)
            y = random.randint(0, self.map_size-1)
            tile_at_xy = session.query(Tile).filter(Tile.x_coordinate == x).filter(Tile.y_coordinate ==
y).filter(Tile.simulation_relationship == Program.current_simulation).one()
            while tile_at_xy.tile_type != "Blank":
                x = random.randint(0, self.map_size-1)
```

```
                    y = random.randint(0, self.map_size-1)
                    tile_at_xy = session.query(Tile).filter(Tile.x_coordinate == x).filter(Tile.y_coordinate ==
y).filter(Tile.simulation_relationship == Program.current_simulation).one()
                tile_at_xy.tile_type = "Organism"
                new_organism = Organism(chromosome=current_species.average_chromosome,
reservoir="", colour=current_species.average_species_colour,
reproduction_code=current_species.average_reproduction_code, x_position=x, y_position=y,
species_id=current_species.id, simulation_id=self.simulation.id)
                session.add(new_organism)
                session.commit()


class Organism_Tile(Simulation_Tile):
    def __init__(self, simulation, surface, simulation_surface, block_size, x_position, y_position,
x_coordinate, y_coordinate, screen_width):
        super().__init__(surface, simulation_surface, block_size, x_position, y_position, x_coordinate,
y_coordinate, screen_width)
        organism_on_tile =
session.query(Organism).filter(Organism.x_position==x_coordinate).filter(Organism.y_position==y_co
ordinate).filter(Organism.simulation_id==simulation.id).one()
        r, g, b = organism_on_tile.colour.strip().split(",")
        self.block_size = block_size
        self.colour_tuple = (int(r),int(g),int(b))
        self.simulation_surface = simulation_surface
        self.x_position = x_position
        self.y_position = y_position
        self.tile_type = "Organism"
        self.image = "Grass.png"
        self.block_size = block_size
    def draw_organism(self):
        pygame.draw.circle(self.simulation_surface, self.colour_tuple,
(int(self.x_position+(self.block_size/2)),int(self.y_position+(self.block_size/2))), int(self.block_size/7))
        pygame.draw.circle(self.simulation_surface, (0,0,0),
(int(self.x_position+(self.block_size/2)),int(self.y_position+(self.block_size/2))), int(self.block_size/7),
1)
```

Simulation Screen I created:



In this version of the code as well as adding the organisms (seen as circles on the simulation map), I also changed some of the images for the different food types. I also added a new food type fruit which was implemented in the same way as the other foods. As can be seen in the image there are currently three different colours that correspond to the three starting species which all have different genes.

Quick Test of the Simulations Screen Version:

There are three different species as indicated in the detail entry screen for this simulation.

The number of organisms in the simulation is one less than expected. However, this is because the starting organisms are spread evenly between the species so if the number of starting organisms is not fully divisible by the number of starting species, then we expect to have less organisms that stated in the detail entry screen.

The organisms have all been displayed with their correct colours and in unique squares.

The organisms are displayed as the right size.

Moving Organisms (Version 0.9.1):

Overview:

In this version I will go through the organism and randomly select a movement direction to one of their adjacent tiles. The organism's position attributes will then be modified accordingly in the database and will then be displayed on the correct tile once the create_tiles method is called again.

Algorithm Explanation:

The move_organisms method - This method will randomly select the direction in which the organism will move. There are 5 different options for the movement, up, down, right, left and staying still. This method will then call the corresponding method according to the direction in which the organism moves.

The move_up method - The move up method will first check that the tile above the organism is of tile type “Blank”. If it is not then the organism will remain still. If the tile is blank, then the organism’s position in the database will be updated. The organism relationship for the tile the organism was originally on will be set to None and the tile it moves to will have it’s organism_relationship set to the organism that is moved. The tile types are also updated. The other move methods work using the same idea.

I also added the current_simulation_state variable to the simualtion_screen. This will then determine whether the simulation is started or paused or playing. This is changed on the click of the start/pause/play button. This variable also controls which version of the start/pause/play button is displayed.

Version 0.9.1 Code:

```
self.simulation_state = 0
        self.simulation = Program.current_simulation
        self.simulation_map = self.simulation.map_relationship[0]
        self.map_size = self.simulation_map.map_size

        self.screen_width = window.winfo_screenwidth()
        self.screen_height = window.winfo_screenheight()
        self.surface = pygame.display.set_mode((self.screen_width, self.screen_height))
        self.surface.fill(BLACK)

        self.surface_height = int(self.screen_height-100)
        self.block_size = int(self.surface_height/self.simulation_map.map_size)
        self.surface_width = int(self.block_size*self.simulation_map.map_size)

        surface=self.surface
        self.simulation_surface = pygame.surface.Surface((self.surface_width, self.surface_height))
        self.simulation_surface.fill(BLACK)

        self.block_size = int(self.surface_height/self.simulation_map.map_size)

        self.pause_button_position = [100,300]
        self.save_button_position = [100, 370]
        self.quit_button_position = [100,440]
        self.analysis_button_position = [1300,300]

        smallfont = pygame.font.SysFont("Corbel",40)
        self.start_text = smallfont.render("Start" , True , WHITE)
        self.pause_text = smallfont.render("Pause" , True , WHITE)
        self.play_text = smallfont.render("Play" , True , WHITE)
        self.save_text = smallfont.render("Save" , True , WHITE)
        self.quit_text = smallfont.render("Quit", True, WHITE)
        self.analysis_text = smallfont.render("Analysis", True, WHITE)

        self.start_time = time.time()
        self.interval = 1.0/self.simulation.length_of_turn
        self.tick = self.interval / 50.0
        self.count = 0

        self.tile_array = [[None for i in range(self.map_size)] for j in range(self.map_size)]

        self.organisms = []

        if len(self.simulation.tile_relationship) == 0:
            self.set_up_grid()
            self.position_water()
            self.position_veg_and_fruit()
            self.position_meat()
            self.create_species()
            self.create_organisms()
        self.create_tiles()
        self.draw_buttons()
        self.draw_tiles()


        while True:
            mouse = pygame.mouse.get_pos()
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    pygame.quit()
                    sys.exit()

                if event.type == pygame.MOUSEBUTTONDOWN:
                    if self.pause_button_position[0] <= mouse[0] <= self.pause_button_position[0]+140 and
self.pause_button_position[1] <= mouse[1] <= self.pause_button_position[1]+40:
                        if self.simulation_state==0:
                            self.start()
                        elif self.simulation_state==1:
                            self.pause()
                        elif self.simulation_state==2:
                            self.play()

                    if self.save_button_position[0] <= mouse[0] <= self.save_button_position[0]+140 and
self.save_button_position[1] <= mouse[1] <= self.save_button_position[1]+40:
                        self.save()

                    if self.quit_button_position[0] <= mouse[0] <= self.quit_button_position[0]+140 and
self.quit_button_position[1] <= mouse[1] <= self.quit_button_position[1]+40:
                        self.quit_simulation()

                    if self.analysis_button_position[0] <= mouse[0] <= self.analysis_button_position[0]+180
and self.analysis_button_position[1] <= mouse[1] <= self.analysis_button_position[1]+40:
```

```
                        self.analysis()
                pygame.display.update()


                if self.simulation_state == 1:
                    if time.time() >= self.start_time + self.interval * self.count:
                        print("Moving all organisms")
                        self.move_organisms()
                        self.tile_array = [[None for i in range(self.map_size)] for j in range(self.map_size)]
                        print("Creating Tiles")
                        self.create_tiles()
                        print("Drawing Tiles")
                        self.draw_tiles()
                        self.count += 1
                    time.sleep(self.tick)


        def move_organisms(self):
            for organism_to_move in self.organisms:
                movement_direction = random.randint(0,4)
                if movement_direction == 0:
                    movement_direction = self.move_up(organism_to_move)

                elif movement_direction == 1:
                    movement_direction = self.move_right(organism_to_move)

                elif movement_direction == 2:
                    movement_direction = self.move_down(organism_to_move)

                elif movement_direction == 3:
                    movement_direction = self.move_left(organism_to_move)

                elif movement_direction == 4:
                    pass

        def move_up(self, organism_to_move):
            if organism_to_move.y_position > 0:
                if
    self.tile_array[organism_to_move.x_position][organism_to_move.y_position-1].tile_type=="Blank":
                    tile_to_move_from = organism_to_move.tile_relationship
                    tile_to_move_to =
    session.query(Tile).filter(Tile.x_coordinate==organism_to_move.x_position).filter(Tile.y_coordinate==o
    rganism_to_move.y_position-1).filter(Tile.simulation_relationship==self.simulation).one()
                    tile_to_move_from.tile_type = "Blank"
                    tile_to_move_to.tile_type = "Organism"
                    tile_to_move_from.organism_relationship = None
                    tile_to_move_to.organism_id = None
                    tile_to_move_to.organism_relationship = organism_to_move
                    tile_to_move_to.organism_id = organism_to_move.id
                    organism_to_move.y_position -= 1
                    session.commit()
                    return 0
                else:
                    return 4
            else:
                return(4)

        def move_right(self, organism_to_move):
            if organism_to_move.x_position < self.map_size-1:
```

```
            if
self.tile_array[organism_to_move.x_position+1][organism_to_move.y_position].tile_type=="Blank":
                tile_to_move_from = organism_to_move.tile_relationship
                tile_to_move_to =
session.query(Tile).filter(Tile.x_coordinate==organism_to_move.x_position+1).filter(Tile.y_coordinate=
=organism_to_move.y_position).filter(Tile.simulation_relationship==self.simulation).one()
                tile_to_move_from.tile_type = "Blank"
                tile_to_move_to.tile_type = "Organism"
                tile_to_move_from.organism_relationship = None
                tile_to_move_to.organism_id = None
                tile_to_move_to.organism_relationship = organism_to_move
                tile_to_move_to.organism_id = organism_to_move.id
                organism_to_move.x_position += 1
                session.commit()
                return 1
            else:
                return 4
        else:
            return(4)

    def move_down(self, organism_to_move):
        if organism_to_move.y_position < self.map_size-1:
            if
self.tile_array[organism_to_move.x_position][organism_to_move.y_position+1].tile_type=="Blank":
                tile_to_move_from = organism_to_move.tile_relationship
                tile_to_move_to =
session.query(Tile).filter(Tile.x_coordinate==organism_to_move.x_position).filter(Tile.y_coordinate==o
rganism_to_move.y_position+1).filter(Tile.simulation_relationship==self.simulation).one()
                tile_to_move_from.tile_type = "Blank"
                tile_to_move_to.tile_type = "Organism"
                tile_to_move_from.organism_relationship = None
                tile_to_move_to.organism_id = None
                tile_to_move_to.organism_relationship = organism_to_move
                tile_to_move_to.organism_id = organism_to_move.id
                organism_to_move.y_position += 1
                session.commit()
                return 2
            else:
                return 4
        else:
            return(4)

    def move_left(self, organism_to_move):
        if organism_to_move.x_position > 0:
            if
self.tile_array[organism_to_move.x_position-1][organism_to_move.y_position].tile_type=="Blank":
                tile_to_move_from = organism_to_move.tile_relationship
                tile_to_move_to =
session.query(Tile).filter(Tile.x_coordinate==organism_to_move.x_position-1).filter(Tile.y_coordinate=
=organism_to_move.y_position).filter(Tile.simulation_relationship==self.simulation).one()
                tile_to_move_from.tile_type = "Blank"
                tile_to_move_to.tile_type = "Organism"
                tile_to_move_from.organism_relationship = None
                tile_to_move_to.organism_id = None
                tile_to_move_to.organism_relationship = organism_to_move
                tile_to_move_to.organism_id = organism_to_move.id
                organism_to_move.x_position -= 1
                session.commit()
                return 3
            else:
```

```
            return 4
        else:
            return(4)

class Organism_Tile(Simulation_Tile):
    def __init__(self, simulation, surface, simulation_surface, block_size, x_position, y_position,
x_coordinate, y_coordinate, screen_width):
        super().__init__(surface, simulation_surface, block_size, x_position, y_position, x_coordinate,
y_coordinate, screen_width)
        organism_on_tile =
session.query(Organism).filter(Organism.x_position==x_coordinate).filter(Organism.y_position==y_co
ordinate).filter(Organism.simulation_id==simulation.id).one()
        r, g, b = organism_on_tile.colour.strip().split(",")
        self.block_size = block_size
        self.colour_tuple = (int(r),int(g),int(b))
        self.simulation_surface = simulation_surface
        self.x_position = x_position
        self.y_position = y_position
        self.tile_type = "Organism"
        self.block_size = block_size

def start(self):
        self.simulation_state = 1
        self.draw_buttons()
        self.organisms = session.query(Organism).filter(Organism.simulation_id==self.simulation.id).all()

    def pause(self):
        self.simulation_state = 2
        self.draw_buttons()
        print("Pause")

    def play(self):
        self.simulation_state = 1
        self.draw_buttons()
        print("Play")
```

Quick Test of the Organisms Movement Version:

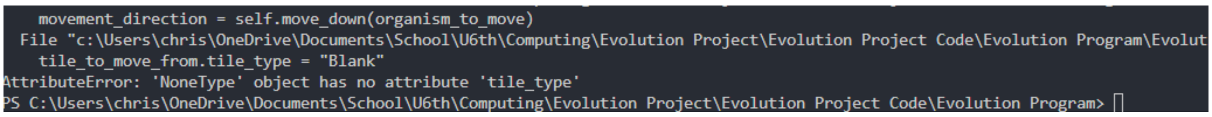
When I hit the start button, the organisms will start to randomly move around the screen and the start button will turn to a pause button.
The organisms will not be displayed on top of each other or on tiles of types that aren't blank.

When I hit the pause button, the organisms stop moving and the pause button turns to a play button.

When i hit the play button, the organisms will start to randomly move around the screen and the start button will turn to a pause button.

If I quit the simulation, and then return, the simulation will be at the state I quit at.

However when I ran the program for a couple of moves, i got the following error:



```
    movement_direction = self.move_down(organism_to_move)
  File "c:\Users\chris\OneDrive\Documents\School\U6th\Computing\Evolution Project\Evolution Project Code\Evolution Program\Evolut
    tile_to_move_from.tile_type = "Blank"
AttributeError: 'NoneType' object has no attribute 'tile_type'
PS C:\Users\chris\OneDrive\Documents\School\U6th\Computing\Evolution Project\Evolution Project Code\Evolution Program> 
```

After debugging my program, I found the problem was occurring because two organisms were moving to the same location. Since my database is designed so that only one organism can be related to a tile, one previously moved organism would have the tile_relationship set to None. When i then tried to use the .tile_type attribute, it would say none type has no attribute tile_type. The error was due to me not having updated the self.tiles_array so when the second organism checked whether the spot was free, it was so it then moved there. In order to fix this, I changed the move_up/right/down/left methods as below:

```
def move_up(self, organism_to_move):
        if organism_to_move.y_position > 0:
            if
self.tile_array[organism_to_move.x_position][organism_to_move.y_position-1].tile_type=="Blank":
                tile_to_move_from = organism_to_move.tile_relationship
                tile_to_move_to =
session.query(Tile).filter(Tile.x_coordinate==organism_to_move.x_position).filter(Tile.y_coordinate==o
rganism_to_move.y_position-1).filter(Tile.simulation_relationship==self.simulation).one()
                tile_to_move_from.tile_type = "Blank"
                tile_to_move_to.tile_type = "Organism"

self.tile_array[organism_to_move.x_position][organism_to_move.y_position].tile_type="Blank"

self.tile_array[organism_to_move.x_position][organism_to_move.y_position-1].tile_type="Organism"
                tile_to_move_from.organism_relationship = None
                tile_to_move_from.organism_id = None
                tile_to_move_to.organism_relationship = organism_to_move
                tile_to_move_to.organism_id = organism_to_move.id
                organism_to_move.y_position -= 1
                session.commit()
                return 0
            else:
                return 4
        else:
            return(4)

    def move_right(self, organism_to_move):
        if organism_to_move.x_position < self.map_size-1:
            if
self.tile_array[organism_to_move.x_position+1][organism_to_move.y_position].tile_type=="Blank":
                tile_to_move_from = organism_to_move.tile_relationship
                tile_to_move_to =
session.query(Tile).filter(Tile.x_coordinate==organism_to_move.x_position+1).filter(Tile.y_coordinate=
=organism_to_move.y_position).filter(Tile.simulation_relationship==self.simulation).one()
                tile_to_move_from.tile_type = "Blank"
                tile_to_move_to.tile_type = "Organism"

self.tile_array[organism_to_move.x_position][organism_to_move.y_position].tile_type="Blank"

self.tile_array[organism_to_move.x_position+1][organism_to_move.y_position].tile_type="Organism"
                tile_to_move_from.organism_relationship = None
                tile_to_move_from.organism_id = None
                tile_to_move_to.organism_relationship = organism_to_move
                tile_to_move_to.organism_id = organism_to_move.id
                organism_to_move.x_position += 1
                session.commit()
                return 1
            else:
                return 4
        else:
            return(4)

    def move_down(self, organism_to_move):
        if organism_to_move.y_position < self.map_size-1:
            if
self.tile_array[organism_to_move.x_position][organism_to_move.y_position+1].tile_type=="Blank":
                tile_to_move_from = organism_to_move.tile_relationship
                tile_to_move_to =
session.query(Tile).filter(Tile.x_coordinate==organism_to_move.x_position).filter(Tile.y_coordinate==o
rganism_to_move.y_position+1).filter(Tile.simulation_relationship==self.simulation).one()
                tile_to_move_from.tile_type = "Blank"
                tile_to_move_to.tile_type = "Organism"

self.tile_array[organism_to_move.x_position][organism_to_move.y_position].tile_type="Blank"
self.tile_array[organism_to_move.x_position][organism_to_move.y_position+1].tile_type="Organism"
                tile_to_move_from.organism_relationship = None
                tile_to_move_from.organism_id = None
                tile_to_move_to.organism_relationship = organism_to_move
                tile_to_move_to.organism_id = organism_to_move.id
                organism_to_move.y_position += 1
                session.commit()
                return 2
            else:
                return 4
        else:
            return(4)

    def move_left(self, organism_to_move):
        if organism_to_move.x_position > 0:
            if
self.tile_array[organism_to_move.x_position-1][organism_to_move.y_position].tile_type=="Blank":
                tile_to_move_from = organism_to_move.tile_relationship
                tile_to_move_to =
session.query(Tile).filter(Tile.x_coordinate==organism_to_move.x_position-1).filter(Tile.y_coordinate=
=organism_to_move.y_position).filter(Tile.simulation_relationship==self.simulation).one()
                tile_to_move_from.tile_type = "Blank"
                tile_to_move_to.tile_type = "Organism"

self.tile_array[organism_to_move.x_position][organism_to_move.y_position].tile_type="Blank"

self.tile_array[organism_to_move.x_position-1][organism_to_move.y_position].tile_type="Organism"
                tile_to_move_from.organism_relationship = None
                tile_to_move_from.organism_id = None
                tile_to_move_to.organism_relationship = organism_to_move
                tile_to_move_to.organism_id = organism_to_move.id
                organism_to_move.x_position -= 1
                session.commit()
                return 3
            else:
                return 4
        else:
            return 4
```

Checking the Tiles Around Organisms and Reacting to Them (Version 0.9.2 and 0.9.3):

Overview:

In order to check the tiles around the organism, I must give each tile a set of neighbours. Therefore I will check through each tile and add all its neighbouring tiles to a self.neighbours array within each tile. I will then cycle through these neighbouring tiles and check their types. I will then use certain conditions to choose the action that the organism will perform.

Algorithm Explanation:

The method find_tiles_neighbours - This method will find the tile in above, to the right, below and to the left of each of the tiles in the simulation (for coroner/side tiles only the tiles that exist are added).

The method check_area_around - This method will look at the tiles around the organism. It will then react accordingly depending on the tile type of that tile.

Version 0.9.2 and 0.9.3 Code:

```
class Simulation_Tile():
    def __init__(self, surface, simulation_surface, block_size, x_position, y_position, x_coordinate,
y_coordinate, screen_width):
        self.surface = surface
        self.simulation_surface = simulation_surface
        self.block_size = block_size
        self.x_coordinate = x_coordinate
        self.y_coordinate = y_coordinate
        self.x_position = x_position
        self.y_position = y_position
        self.screen_width = screen_width
        self.image = "Grass.png"
        self.tile_type = "Blank"
        self.neighbouring_tiles = []

def find_tile_neighbours(self):
        for tile_row in self.tile_array:
            for tile in tile_row:
                if tile.x_coordinate > 0:
                    tile.neighbouring_tiles.append(self.tile_array[tile.x_coordinate-1][tile.y_coordinate])
                if tile.x_coordinate < self.map_size-1:
                    tile.neighbouring_tiles.append(self.tile_array[tile.x_coordinate+1][tile.y_coordinate])
                if tile.y_coordinate > 0:
                    tile.neighbouring_tiles.append(self.tile_array[tile.x_coordinate][tile.y_coordinate-1])
                if tile.y_coordinate < self.map_size-1:
                    tile.neighbouring_tiles.append(self.tile_array[tile.x_coordinate][tile.y_coordinate+1])


    if self.simulation_state == 1:
                self.move_organisms()
                self.tile_array = [[None for i in range(self.map_size)] for j in range(self.map_size)]
                self.create_tiles()
                self.draw_tiles()
                self.find_tile_neighbours()
                for tile in self.organism_tiles:
                    self.check_area_around(tile)
                self.count += 1

    def check_area_around(self, tile):
        for neighbour in tile.neighbouring_tiles:
            action_performed = False
            if neighbour.tile_type == "Organism":
                if neighbour.organism_on_tile.reproduction_code ==
tile.organism_on_tile.reproduction_code:
                    action_performed = True
                    print("Reproduce")
                else:
                    print("Organisms interact")
                    action_performed = True
            elif neighbour.tile_type == "Meat" or neighbour.tile_type == "Veg" or neighbour.tile_type ==
"Fruit" and action_performed == False:
                print("Eating food")
                action_performed = True
            elif neighbour.tile_type == "Water" and action_performed == False:
                action_performed = True
                print("Drink water")
```

Quick Test of the Checking and Reaction Version:

When I print the values of each tile’s neighbours, the program will output the correct tile items.

The program prints a series of commands, (“drink water”, “eat food”, etc) for each organism.

The number of commands outputted is equal to the number of organisms in the simulation.

Eating Food (Version 0.9.4):

Overview:

For consuming food, i will compare the consumption code of the organism with the consumption code of the food. For every character that matches I will add 2 to the number of resources added to the organism. If it is longer or a hashtag I will only add 1. If it is a mismatch i will subtract 1. I will then collect the relevant number of resources off the food consumed and depending on its type may change or delete it.

I will also have a fruit resource which works very similarly to the veg and meat resource. Once the nutritional value is empty, the tree is set to fruitless so the fruitless tree is displayed. The nutritional value will be reset once the turn is over (turn is set by the user not every time organisms move).

Algorithm Explanation:

The method eat_food - This method is responsible for the feeding mechanisms of the organisms. When this method is called on a tile, it will first check the tile_type of the tile. It will then react accordingly.

The program will first compare each of the values in the organisms consumer code with the consumption code of the food (the later is always a standard value for a food type). For every character that matches, 2 will be added to the total number of resources that will be eaten at the end of the calculation. If the consumption code character is a # then 1 will be added. Once all the characters have been checked, if the total is positive, then the first n resources in the food's nutritional value will be added to the organisms reservoir, where n is the total number of resources that will be eaten.

If the food type is meat then the resources will be permanently removed from the meat Food object. It will then check if the nutritional_value is empty. If it is, the tile will be converted to a blank tile.

If the food type is fruit then it will work the same as the meat. The difference is that instead of deleting it, the fruit tile will become fruitless until the nutritional_value is reset at the end of the turn.

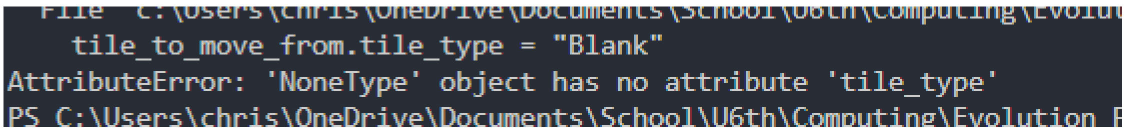
The veg will not have resources removed from its nutritional value.

Version 0.9.4 Code:

```
def eat_food(self, organism, food):
    food_consumption_code = list(food.consumption_code)
    organism_consumer_code = list(organism.chromosome.split("|")[1].split(",")[2])
    total = 0
    if food.food_type == "Meat":
        for i in range(len(organism_consumer_code)):
            if i<len(food_consumption_code)-1:
                if organism_consumer_code[i]==food_consumption_code[i]:
                        total+=2
                if organism_consumer_code[i]=="#":
                    total+=1
        if total > 0:
            organism.reservoir += food.nutritional_value[0:total]
            food.nutritional_value = food.nutritional_value[total:]
            if len(food.nutritional_value)==0:
                food.food_type = "Deleted"
                food.tile_relationship.tile_type = "Blank"
                food.tile_relationship.food_relationship = None
                print(food.tile_relationship)
                food.tile_relationship.tile_id = None
                session.delete(food)
            session.commit()
    elif food.food_type == "Fruit":
        for i in range(len(organism_consumer_code)):
            if i<len(food_consumption_code)-1:
                if organism_consumer_code[i]==food_consumption_code[i]:
                    total+=2
                if organism_consumer_code[i]=="#":
                    total+=1
        if total > 0:
            organism.reservoir += food.nutritional_value[0:total]
            food.nutritional_value = food.nutritional_value[total:]
            session.commit()
        if len(food.nutritional_value)==0:
            food.tile_relationship.tile_type = "Tree"
            food.food_type = "Tree"
            session.commit()

    elif food.food_type == "Veg":
        for i in range(len(organism_consumer_code)):
            if i<len(food_consumption_code)-1:
                if organism_consumer_code[i]==food_consumption_code[i]:
                    total+=2
                if organism_consumer_code[i]=="#":
                    total+=1
        if total > 0:
            organism.reservoir += food.nutritional_value[0:total]
            session.commit()

class Fruit_Tile(Simulation_Tile):
    def __init__(self, food_map, surface, simulation_surface, block_size, x_position, y_position,
x_coordinate, y_coordinate, screen_width, fruit_on_tree):
    super().__init__(surface, simulation_surface, block_size, x_position, y_position, x_coordinate,
y_coordinate, screen_width)
    self.food_on_tile =
session.query(Food).filter(Food.x_coordinate==x_coordinate).filter(Food.y_coordinate==y_coordinate)
.filter(Food.map_id==food_map.id).one()
    if fruit_on_tree:
        self.image = "Fruit_on_grass_with_fruit.png"
    else:
        self.image = "Fruit_on_grass.png"
    self.tile_type = "Fruit"
```

Quick Test of the Eating Version:

When I ran the code above, I got an error shown below:



```
  File "C:\Users\chris\OneDrive\Documents\School\U6th\Computing\Evolut
    tile_to_move_from.tile_type = "Blank"
AttributeError: 'NoneType' object has no attribute 'tile_type'
PS C:\Users\chris\OneDrive\Documents\School\U6th\Computing\Evolution F
```

In order to sort this, I reworked my code to change where I set the tile_relationship.food_relationship to None.

The new adapted code is shown below:

```
def eat_food(self, organism, food):
    food_consumption_code = list(food.consumption_code)
    organism_consumer_code = list(organism.chromosome.split("|")[1].split(",")[2])
    total = 0
    if food.food_type == "Meat":
        for i in range(len(organism_consumer_code)):
            if i<len(food_consumption_code)-1:
                if organism_consumer_code[i]==food_consumption_code[i]:
                    total+=2
                if organism_consumer_code[i]=="#":
                    total+=1
        if total > 0:
            organism.reservoir += food.nutritional_value[0:total]
            food.nutritional_value = food.nutritional_value[total:]
            if len(food.nutritional_value)==0:
                food.food_type = "Deleted"
                food.tile_relationship.tile_type = "Blank"
                food.tile_relationship.tile_id = None
                food.tile_relationship.food_relationship = None
                session.delete(food)
            session.commit()
    elif food.food_type == "Fruit":
        for i in range(len(organism_consumer_code)):
            if i<len(food_consumption_code)-1:
                if organism_consumer_code[i]==food_consumption_code[i]:
                    total+=2
                if organism_consumer_code[i]=="#":
                    total+=1
        if total > 0:
            organism.reservoir += food.nutritional_value[0:total]
            food.nutritional_value = food.nutritional_value[total:]
            session.commit()
        if len(food.nutritional_value)==0:
            food.tile_relationship.tile_type = "Tree"
            food.food_type = "Tree"
            session.commit()
```

Fighting (Version 0.9.5):

Overview:

When organisms are adjacent to other organisms, they will either reproduce or fight. If two organisms fight, then the attack and defence tags of the attacking and defending organisms will be used. This will work similarly to eating. When an organism attacks another, there is a chance that the other organism will hide rather than getting involved in the fight. This will be done using the hide tag within the control section of the chromosome.

When an organism attacks another organism, the attack code of the attacker will be compared to the defence code of the defender. For every character that the attacker matches with the defender, they will add a 2 to the total number of resources they will take from the defender. For every extra character or hashtag that the attacker has, they will add 1 to their total. However if the two characters don't match then 1 will be subtracted from the total.

If after the attack, the organism that was defending will lose the number of resources specified by the total from their reservoir. If they have insufficient food in their reservoir then the organism will be killed. When an organism is killed, it will be converted to a meat food item. Its genome will be converted to the nutritional value of the meat item.

Algorithm Explanation:

The method hide - This method works by checking the hide tag in the control section of the defending organism against the attack tag of the attacking organism. If the hide tag is longer or any of the characters don't match, then the organis, will not hide and the attack will take place.

The method attack - When this method is called, it takes two organism objects from the database as inputs. It will start by finding the attack and defence codes in the first and second organisms respectively. It will then calculate the number of resources that must be given to the attacker as described in the overview. If the value is positive, the required resources will be removed from the defenders reservoir and added to the attackers. It will then save these changes in the database. If the length of the defenders reservoir is 0 then the convert_to_meat method is called.

The method convert_to_meat - This method is called when an organism is killed. The method will cycle through the dead organisms' chromosome and remove all the pipes and commas. It will then use the remaining characters as the nutritional_value for the new meat object. It will also then delete the organism object and convert the tile to blank.

Version 0.9.5 Code:

```
def hide(self, organism_1, organism_2):
    hide = False
    attack_tag = list(organism_1.chromosome.split("|")[0].split(",")[0])
    hide_tag = list(organism_2.chromosome.split("|")[1].split(",")[0])
    if len(hide_tag) < len(attack_tag):
        for i in range(len(hide_tag)):
            if hide_tag[i] != attack_tag[i] and hide_tag[i] != "#":
                hide = True
    else:
        hide = True
    return(hide)

  def attack(self, organism_1, organism_2):
    value = 0
    attack_tag_elements = list(organism_1.chromosome.split("|")[0].split(",")[0])
    defence_tag_elements = list(organism_2.chromosome.split("|")[0].split(",")[1])
```

```
        for i in range(max(len(attack_tag_elements), len(defence_tag_elements))):
            if i <= len(attack_tag_elements)-1:
                if i > len(defence_tag_elements)-1:
                    value += 1
                elif attack_tag_elements[i] == defence_tag_elements[i]:
                    value += 2
                elif attack_tag_elements[i] == "#":
                    value += 1
                else:
                    value -= 1
        if value > 0:
            resources_given = organism_2.reservoir[0:value]
            organism_2.reservoir = organism_2.reservoir[value:]
            organism_1.reservoir += resources_given
            session.commit()
            if len(organism_2.reservoir) == 0:
                self.convert_to_meat(organism_2)

    def convert_to_meat(self, organism_to_convert):
        nutritional_value=""
        for letter in list(organism_to_convert.chromosome):
            if letter != "," and letter != "|":
                nutritional_value += letter
        new_food = Food(food_type="Meat", nutritional_value=nutritional_value,
consumption_code="cdccdddccd", x_coordinate=organism_to_convert.x_position,
y_coordinate=organism_to_convert.y_position, map_id=self.simulation_map.id,
map_relationship=self.simulation_map, tile_relationship=organism_to_convert.tile_relationship)

        tile = organism_to_convert.tile_relationship
        organism_to_convert.tile_relationship.tile_type = "Meat"
        tile.organism_relationship = None
        tile.organism_id = None
        tile.food_relationship = new_food
        tile.food_id = new_food.id
        session.add(new_food)
        session.commit()

        organism_to_convert.species_relationship = None
        organism_to_convert.species_id = None
        organism_to_convert.simulation_relationship = None
        organism_to_convert.simulation_id = None
        self.organisms.remove(organism_to_convert)

        session.delete(organism_to_convert)
        session.commit()
```

Quick Test of the Fighting Version:

When I run the simulation, if I print what happens at each part of the code, I will see a mixture of attacks, hides, and kills messages.

When I examine the messages the outputs are as expected for each situation in which two organisms fight as described above.

However when I ran the simulation for long periods of time, I got an error as shown below:

I was unable to recreate this error after it happened as it only happened on very rare occasions.

This error was due to two organisms attacking the same organism. Since I previously only reset/changed the self.tiles_array in the create_tiles method, when the two organisms attacked within the same movement turn, if the first organism had killed the other organism, when the second organism tried to attack it, it would pass the conditions since I hadn't changed the value of the tile in self.tiles_array. To fix this issue I added to the convert to meat method as shown below:

```
def convert_to_meat(self, organism_to_convert):
        nutritional_value=""
        for letter in list(organism_to_convert.chromosome):
            if letter != "," and letter != "|":
                nutritional_value += letter
        new_food = Food(food_type="Meat", nutritional_value=nutritional_value,
consumption_code="cdccdddccd", x_coordinate=organism_to_convert.x_position,
y_coordinate=organism_to_convert.y_position, map_id=self.simulation_map.id,
map_relationship=self.simulation_map, tile_relationship=organism_to_convert.tile_relationship)

        tile = organism_to_convert.tile_relationship
        organism_to_convert.tile_relationship.tile_type = "Meat"
        tile.organism_relationship = None
        tile.organism_id = None
        tile.food_relationship = new_food
        tile.food_id = new_food.id
        session.add(new_food)
        session.commit()

        self.tile_array[organism_to_convert.x_position][organism_to_convert.y_position].tile_type =
Meat_Tile(self.simulation_map, self.surface, self.simulation_surface, self.block_size,
organism_to_convert.x_position*self.block_size, organism_to_convert.y_position*self.block_size,
organism_to_convert.x_position, organism_to_convert.y_position, self.screen_width)

        organism_to_convert.species_relationship = None
        organism_to_convert.species_id = None
        organism_to_convert.simulation_relationship = None
        organism_to_convert.simulation_id = None
        self.organisms.remove(organism_to_convert)

        session.delete(organism_to_convert)
        session.commit()
```

Reproducing (Version 0.9.6):

Overview:

The idea for organism reproduction is that the two adjacent organisms’ reproduction tags must be at least 75% the same (This value may alter to make the simulation work better). The two organisms must also each have enough enough resources within their reservoir to create half of their own gene. If these conditions are met (these are calculated by the methods match and ready_to_reproduce), then the organisms will start reproducing.

When two organisms reproduce, they will “cross over” (they will select a part of one and a part of another):

```
        Organism_1:                   aa,aa|aaa,aaaa,aaa|aaa,aaa
        Organism_2:                   bb,bb|bbb,bbbb,bbb|bbb,bbb
If crossover point at index 6                 V
        Child Chromosome:             aa,aa|aab,bbbb,bbb|bbb,bbb
```

The child organisms chromosome will then undergo random mutation in order to change some of the features randomly to generate innovative new tags.

```
        Child Chromosome before randomisation:        aa,aa|aab,bbbb,bbb|bbb,bbb
        Child Chromosome after randomisation:         aa,ca|aab,bcbb,bbcb|bdb,bab
```

Algorithm Explanation:

The method match - This cycles through all of the characters in the two given organisms’ reproduction codes. It will then calculate the number of characters that match. If the number of matching characters is equal or higher to 75% of the two chromosomes then the method returns True, otherwise it returns False.

The reproduce method - This method starts by creating a filtered version of the two organisms’ chromosomes, without pipes and commas. It will then check if the two organisms are ready to reproduce via the ready_to_reproduce method. If they are ready, then a random crossover point is calculated. If this crossover point lies on a pipe or a comma then chromosomes are split accordingly. They are then recombined to make a new chromosome for their offspring. If the crossover point is not on a pipe or comma then the new organism’s chromosome is equal to parent 1’s. The chromosome produced is then randomized using a randomized chromosome and then a new organism is created in any blank spaces around organism_1. The relevant data (eg position and tile relationship) is all updated in the database. It will then return True if the conditions are all met and the organisms reproduce or False if they are not.

The method ready_to_reproduce - This method is responsible for checking whether the organisms have enough food in their reservoirs to create their half of the offspring’s chromosome. It will go through letter by letter, trying to remove the letter. If there is an error because the letter is not in the list then has_resources is set to False and so returns False

Version 0.9.6 Code:

```
def match(self, organism_1_code, organism_2_code):
    match = False
    total = 0
    for i in range(min(len(organism_1_code), len(organism_2_code))):
        if list(organism_1_code)[i]==list(organism_2_code)[i]:
            total+=1
        if total >= 0.75*min(len(organism_1_code), len(organism_2_code)):
            match = True
        return match

    def reproduce(self, organism_1, organism_2):
        commas_filtered_1 = list(filter(lambda a: a != ",", list(organism_1.chromosome)))
        organism_1_chromosome = list(filter(lambda a: a != "|", commas_filtered_1))
        commas_filtered_2 = list(filter(lambda a: a != ",", list(organism_2.chromosome)))
        organism_2_chromosome = list(filter(lambda a: a != "|", commas_filtered_2))
        if self.ready_to_reproduce(organism_1, organism_2, organism_1_chromosome,
organism_2_chromosome):
            crossover_point = random.randint(1, min(len(organism_1.chromosome)-1,
len(organism_2.chromosome)-1))
            if list(organism_1.chromosome)[crossover_point] == "|" or
list(organism_1.chromosome)[crossover_point] == ",":
                new_organism_chromosome = organism_1.chromosome[0:crossover_point] +
organism_2.chromosome[crossover_point:]
            else:
                new_organism_chromosome = organism_1.chromosome
            tag_section = new_organism_chromosome.split("|")[0]
            control_section = new_organism_chromosome.split("|")[1]
            exchange_section = new_organism_chromosome.split("|")[2]
            new_organism_chromosome = self.randomize_chromosome(tag_section, control_section,
exchange_section)

            organism_placed = False
            for tile in self.tile_array[organism_1.x_position][organism_1.y_position].neighbouring_tiles:
                if tile.tile_type == "Blank" and organism_placed == False:
                    new_organism = Organism(chromosome=new_organism_chromosome,
reservoir="abcd", colour=self.find_colour(tag_section, control_section, exchange_section),
reproduction_code=control_section.split(",")[0], x_position=tile.x_coordinate,
y_position=tile.y_coordinate, species_id=organism_1.species_id,
species_relationship=organism_1.species_relationship, simulation_id=self.simulation.id,
simulation_relationship=self.simulation)
                    session.add(new_organism)
                    tile_at_xy = session.query(Tile).filter(Tile.x_coordinate ==
tile.x_coordinate).filter(Tile.y_coordinate == tile.y_coordinate).filter(Tile.simulation_relationship ==
Program.current_simulation).one()
                    tile_at_xy.organism_id = new_organism.id
                    tile_at_xy.organism_relationship = new_organism
                    tile_at_xy.tile_type = "Organism"
                    session.commit()
                    organism_placed = True
                    self.organisms.append(new_organism)

self.tile_array[new_organism.x_position][new_organism.y_position].tile_type=Organism_Tile(self.simu
lation_map, self.surface, self.simulation_surface, self.block_size,
new_organism.x_position*self.block_size, new_organism.y_position*self.block_size,
new_organism.x_position, new_organism.y_position, self.screen_width, tile_at_xy)
            if organism_placed == False:
                return False
            return True
        else:
            return False

    def ready_to_reproduce(self, organism_1, organism_2, chromosome_value_1,
chromosome_value_2):
        has_resources = True
        reservoir_copy_1 = list(organism_1.reservoir)
        reservoir_copy_2 = list(organism_2.reservoir)
```

```
for character in chromosome_value_1[:int(len(chromosome_value_1))]:
    try:
        reservoir_copy_1.remove(character)
    except:
        has_resources = False
for character in chromosome_value_2[int(len(chromosome_value_1)):]:
    try:
        reservoir_copy_2.remove(character)
    except:
        has_resources = False
return has_resources
```

Quick Test of the Reproducing Version:

When two organisms meet, a new organism is correctly created and displayed on the simulation screen.

A new organism will only appear on a blank square.

Ending the turn (Version 0.9.7):

Overview:

After the number of turns as specified by the user have passed, all the fruits will reset their nutritional value and be converted back to fruit trees. In this version I also ended up removing the save functionality since it didn't really add any value to the user experience and was unstable.

Algorithm Explanation:

The method end_turn - When this method is called, it will get all the tree food objects in the database for the current simulation. It will then change the tile_type to fruit and reset the nutritional value to the initial fruit nutritional value. It will similarly reset the nutritional value of fruit food objects.

The end method above is called only when the number of turns specified by the user in the creation of the simulation. This is controlled in the while True section of the simulation_screen __init__.

Version 0.9.7 Code:

```
if self.simulation_state == 1:
        self.organism_tiles = []
        self.move_organisms()
        self.tile_array = [[None for i in range(self.map_size)] for j in range(self.map_size)]
        self.create_tiles()
        self.draw_tiles()
        self.find_tile_neighbours()
        for tile in self.organism_tiles:
            self.check_area_around(tile)
        self.count += 1
```

```
            if self.current_turn%self.simulation_length_of_turn == 0:
                self.end_turn()
            self.current_turn += 1

def end_turn(self):
        trees =
session.query(Food).filter(Food.food_type=="Tree").filter(Food.map_relationship==self.simulation_ma
p).all()
        fruits =
session.query(Food).filter(Food.food_type=="Fruit").filter(Food.map_relationship==self.simulation_ma
p).all()
        for tree in trees:
            tree.nutritional_value = "cabdcababcacdacadadcacdacadadc"
            tree.food_type = "Fruit"
            tree.tile_relationship.tile_type = "Fruit"
            session.commit()
        for fruit in fruits:
            fruit.nutritional_value = "cabdcababcacdacadadcacdacadadc"
            session.commit()
        self.simulation.current_turn+=1
        session.commit()
```

Quick Test of the the Ending of Turn Version:

When the correct number of turns have passed, the fruit trees change image from fruitless to having fruit.

Removing Resources (Version 0.9.8):

Overview:

Every time the user moves, they must lose resources. In order to penalise older organisms, the number of resources taken will be equal to the number of turns that they have been alive for. This will mirror the way that in nature, organisms age and become less efficient.

Algorithm Explanation:

The method end_turn - In this version of the end_turn method, I add the functionality where the organisms will gain age. This will then be used to remove that many resources each time the organism moves. These resources are removed in the simulation_screen's while True section of __init__.

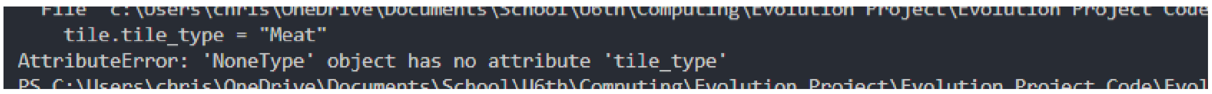
Version 0.9.8 Code:

```
def __init__(self):
        self.simulation_state = 0
        self.simulation = Program.current_simulation
        self.simulation_map = self.simulation.map_relationship[0]
        self.map_size = self.simulation_map.map_size
        self.current_turn = 1
        self.simulation_length_of_turn = self.simulation.length_of_turn

        self.screen_width = window.winfo_screenwidth()
        self.screen_height = window.winfo_screenheight()
        self.surface = pygame.display.set_mode((self.screen_width, self.screen_height))
        self.surface.fill(BLACK)

        self.surface_height = int(self.screen_height-100)
        self.block_size = int(self.surface_height/self.simulation_map.map_size)
        self.surface_width = int(self.block_size*self.simulation_map.map_size)

        surface=self.surface
        self.simulation_surface = pygame.surface.Surface((self.surface_width, self.surface_height))
        self.simulation_surface.fill(BLACK)

        self.block_size = int(self.surface_height/self.simulation_map.map_size)

        self.pause_button_position = [100,300]
        self.save_button_position = [100, 370]
        self.quit_button_position = [100,440]
        self.analysis_button_position = [1300,300]

        smallfont = pygame.font.SysFont("Corbel",40)
        self.start_text = smallfont.render("Start" , True , WHITE)
        self.pause_text = smallfont.render("Pause" , True , WHITE)
        self.play_text = smallfont.render("Play" , True , WHITE)
        self.save_text = smallfont.render("Save" , True , WHITE)
        self.quit_text = smallfont.render("Quit", True, WHITE)
        self.analysis_text = smallfont.render("Analysis", True, WHITE)

        self.start_time = time.time()
        self.interval = 1.0/self.simulation.length_of_turn
        self.tick = self.interval / 50.0
        self.count = 0

        self.tile_array = [[None for i in range(self.map_size)] for j in range(self.map_size)]

        self.organisms = []
        self.organism_tiles = []

        if len(self.simulation.tile_relationship) == 0:
            self.set_up_grid()
            self.position_water()
            self.position_veg_and_fruit()
            self.position_meat()
            self.create_species()
            self.create_organisms()
        self.create_tiles()
        self.find_tile_neighbours()
        self.draw_buttons()
        self.draw_tiles()

        while True:
            mouse = pygame.mouse.get_pos()
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    pygame.quit()
                    sys.exit()

                if event.type == pygame.MOUSEBUTTONDOWN:
                if self.pause_button_position[0] <= mouse[0] <= self.pause_button_position[0]+140 and
self.pause_button_position[1] <= mouse[1] <= self.pause_button_position[1]+40:
                    if self.simulation_state==0:
                        self.start()
                    elif self.simulation_state==1:
                        self.pause()
                    elif self.simulation_state==2:
                        self.play()

                if self.quit_button_position[0] <= mouse[0] <= self.quit_button_position[0]+140 and
self.quit_button_position[1] <= mouse[1] <= self.quit_button_position[1]+40:
                    self.quit_simulation()

                if self.analysis_button_position[0] <= mouse[0] <= self.analysis_button_position[0]+180
and self.analysis_button_position[1] <= mouse[1] <= self.analysis_button_position[1]+40:
                    self.analysis()
            pygame.display.update()


            if self.simulation_state == 1:
                self.organism_tiles = []
                self.move_organisms()
                self.tile_array = [[None for i in range(self.map_size)] for j in range(self.map_size)]
                self.create_tiles()
                self.draw_tiles()
                self.find_tile_neighbours()
                for tile in self.organism_tiles:
                    tile.organism_on_tile.reservoir =
tile.organism_on_tile.reservoir[tile.organism_on_tile.age:]
                    session.commit()
                    if len(tile.organism_on_tile.reservoir) == 0:
                        self.convert_to_meat(tile.organism_on_tile)
                    else:
                        self.check_area_around(tile)
                self.count += 1
                if self.current_turn%self.simulation_length_of_turn == 0:
                    self.end_turn()
                self.current_turn += 1

def end_turn(self):
        trees =
session.query(Food).filter(Food.food_type=="Tree").filter(Food.map_relationship==self.simulation_ma
p).all()
        fruits =
session.query(Food).filter(Food.food_type=="Fruit").filter(Food.map_relationship==self.simulation_ma
p).all()
        for tree in trees:
            tree.nutritional_value = "cabdcababcacdacadadcacdacadadc"
            tree.food_type = "Fruit"
            tree.tile_relationship.tile_type = "Fruit"
            session.commit()
        for fruit in fruits:
            fruit.nutritional_value = "cabdcababcacdacadadcacdacadadc"
            session.commit()
        self.simulation.current_turn+=1
        organisms =
session.query(Organism).filter(Organism.simulation_relationship==self.simulation).all()
        for organism in organisms:
            organism.age += 1
            session.commit()
```

Quick Test of the Resource Removal Version:

When I run the simulation, the organisms will be converted to meat images on the screen at different points (depending on whether they have consumed more or less food in their lives).

However when I ran the simulation for longer periods of time I got the following error:



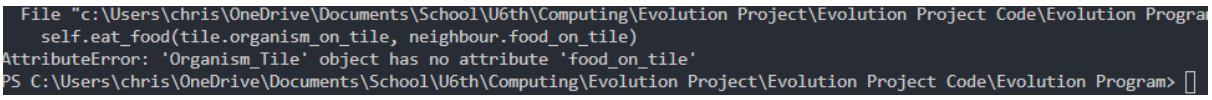
```
  File "C:\Users\chris\OneDrive\Documents\School\U6th\Computing\Evolution Project\Evolution Project Code
    tile.tile_type = "Meat"
AttributeError: 'NoneType' object has no attribute 'tile_type'
PS C:\Users\chris\OneDrive\Documents\School\U6th\Computing\Evolution Project\Evolution Project Code\Evol
```

This error was due to an organism being killed by an organism that executes its commands before the organism that it eats. The organism that was attacked will then execute its commands. When it does so if the organism has already been converted to a meat, it won't have a tile relationship which is later used in the convert to meat method which it uses as it has run out of food in its reservoir. I solved this error by modifying convert to meat as shown below:

```
def convert_to_meat(self, organism_to_convert):
        nutritional_value=""
        for letter in list(organism_to_convert.chromosome):
            if letter != "," and letter != "|":
                nutritional_value += letter
        new_food = Food(food_type="Meat", nutritional_value=nutritional_value,
consumption_code="cdccdddccd", x_coordinate=organism_to_convert.x_position,
y_coordinate=organism_to_convert.y_position, map_id=self.simulation_map.id,
map_relationship=self.simulation_map, tile_relationship=organism_to_convert.tile_relationship)

        tile = organism_to_convert.tile_relationship
        tile.tile_type = "Meat"
        tile.organism_relationship = None
        tile.organism_id = None
        tile.food_relationship = new_food
        tile.food_id = new_food.id
        session.add(new_food)
        session.commit()


self.organism_tiles.remove(self.tile_array[organism_to_convert.x_position][organism_to_convert.y_po
sition])

        self.tile_array[organism_to_convert.x_position][organism_to_convert.y_position].tile_type =
"Meat"
        self.tile_array[organism_to_convert.x_position][organism_to_convert.y_position] =
Meat_Tile(self.simulation_map, self.surface, self.simulation_surface, self.block_size,
organism_to_convert.x_position*self.block_size, organism_to_convert.y_position*self.block_size,
organism_to_convert.x_position, organism_to_convert.y_position, self.screen_width)

        organism_to_convert.species_relationship = None
        organism_to_convert.species_id = None
        organism_to_convert.simulation_relationship = None
        organism_to_convert.simulation_id = None
```

```
    self.organisms.remove(organism_to_convert)

    session.delete(organism_to_convert)
    session.commit()
```

I also had a separate error when I ran the simulation for longer periods of time. I got the following error:



```
  File "c:\Users\chris\OneDrive\Documents\School\U6th\Computing\Evolution Project\Evolution Project Code\Evolution Progra
    self.eat_food(tile.organism_on_tile, neighbour.food_on_tile)
AttributeError: 'Organism_Tile' object has no attribute 'food_on_tile'
PS C:\Users\chris\OneDrive\Documents\School\U6th\Computing\Evolution Project\Evolution Project Code\Evolution Program> 
```

This error was due to the situation where an organism is killed/dies and then another neighbouring organism attempts to eat it in the same turn. Since the value for the tile's tile_type attribute was modified, the organism would correctly treat the tile as a food tile. However in the organism that is eating's tile, the list neighbouring tiles was not updated so when the neighbouring tile object was selected it was of type Organism_Tile rather than Meat_Tile. In order to fix this issue I modified the convert-to_meat method as shown below:

```
def convert_to_meat(self, organism_to_convert):
    nutritional_value=""
    for letter in list(organism_to_convert.chromosome):
        if letter != "," and letter != "|":
            nutritional_value += letter
    new_food = Food(food_type="Meat", nutritional_value=nutritional_value,
consumption_code="cdccdddccd", x_coordinate=organism_to_convert.x_position,
y_coordinate=organism_to_convert.y_position, map_id=self.simulation_map.id,
map_relationship=self.simulation_map, tile_relationship=organism_to_convert.tile_relationship)

    tile = organism_to_convert.tile_relationship
    tile.tile_type = "Meat"
    tile.organism_relationship = None
    tile.organism_id = None
    tile.food_relationship = new_food
    tile.food_id = new_food.id
    session.add(new_food)
    session.commit()


self.organism_tiles.remove(self.tile_array[organism_to_convert.x_position][organism_to_convert.y_po
sition])
    meat_tile = Meat_Tile(self.simulation_map, self.surface, self.simulation_surface, self.block_size,
organism_to_convert.x_position*self.block_size, organism_to_convert.y_position*self.block_size,
organism_to_convert.x_position, organism_to_convert.y_position, self.screen_width)

    neighbours =
self.tile_array[organism_to_convert.x_position][organism_to_convert.y_position].neighbouring_tiles
    for neighbour in neighbours:

neighbour.neighbouring_tiles[neighbour.neighbouring_tiles.index(self.tile_array[organism_to_convert.
x_position][organism_to_convert.y_position])] = meat_tile
    self.tile_array[organism_to_convert.x_position][organism_to_convert.y_position] = meat_tile
    self.tile_array[organism_to_convert.x_position][organism_to_convert.y_position].tile_type =
"Meat"

self.tile_array[organism_to_convert.x_position][organism_to_convert.y_position].neighbouring_tiles =
neighbours

    organism_to_convert.species_relationship = None
    organism_to_convert.species_id = None
    organism_to_convert.simulation_relationship = None
```

```
        organism_to_convert.simulation_id = None
        self.organisms.remove(organism_to_convert)

        session.delete(organism_to_convert)
        session.commit()
```

Tweaking (Version 0.9.9):

Overview:

For the tweaking of my code, I have decided to add the following extensions/changes/final fixes:

- Correctly validate the user input for the simulation creation.
- If attacks/food consumption is zero or negative then skip to the next option in the check area around method.
- I will modify the random mutation to allow for extra letters to be added.

Algorithm Explanation:

The method randomize_chromosome - for this version of the randomise_chromosome method, I added the condition where if the random.randint value used is equal to a number then it will add a new ran

Version 0.9.9 Code:

Validation of the user's simulation customisation details.

```
def collect_input_data(self):
        data = self.menu.get_input_data(recursive=False)
        simulation_name = data[self.simulation_name.get_id()]
        date_created = datetime.now()
        length_of_turn = data[self.length_of_turn.get_id()]
        starting_number_of_organisms = data[self.starting_number_of_organisms.get_id()]
        starting_number_of_species = data[self.starting_number_of_species.get_id()]

        map_size = data[self.map_size.get_id()]
        number_of_food_sources = data[self.number_of_food_sources.get_id()]
        meat_to_veg_ratio = data[self.meat_to_veg_ratio.get_id()]
        number_of_water_sources = data[self.number_of_water_sources.get_id()]
        if self.validate_data(length_of_turn, starting_number_of_organisms,
starting_number_of_species, map_size, number_of_food_sources, meat_to_veg_ratio,
number_of_water_sources):
            self.generate(simulation_name, date_created, length_of_turn, starting_number_of_organisms,
starting_number_of_species, map_size, number_of_food_sources, meat_to_veg_ratio,
number_of_water_sources)
        else:
            Custom_Detail_Entry_screen()

    def validate_data(self, length_of_turn, starting_number_of_organisms,
starting_number_of_species, map_size, number_of_food_sources, meat_to_veg_ratio,
number_of_water_sources):
        try:
            int(length_of_turn)
            int(starting_number_of_organisms)
            int(starting_number_of_species)
            int(map_size)
            int(number_of_food_sources)
            int(number_of_water_sources)
        except:
            return False
        if int(length_of_turn) <= 0 or int(starting_number_of_organisms) <= 0 or
int(starting_number_of_species) <= 0 or int(map_size) <= 0 or int(number_of_food_sources) < 0 or
int(number_of_water_sources) < 0:
            return False
        ratio = meat_to_veg_ratio.split(":")
        if len(ratio) != 2:
            return False
        try:
            int(ratio[0])
            int(ratio[1])
            if int(ratio[0]) < 0 or int(ratio[1]) < 0 or (ratio[0] == "0" and ratio[1] == "0"):
                return False
        except:
            return False
        if int(starting_number_of_species) > int(starting_number_of_organisms):
            return False
        if int(starting_number_of_organisms) + int(number_of_food_sources) +
int(number_of_water_sources) > int(map_size)**2:
            return False
        return True
```

Allowing the organism to execute a separate command if attack or eating provides 0 food (I also made the defending organism hiding cost the attacking organism’s turn):

```
    def check_area_around(self, tile):
        for neighbour in tile.neighbouring_tiles:
            action_performed = False
            if neighbour.tile_type == "Organism":
                if self.match(neighbour.organism_on_tile.reproduction_code,
tile.organism_on_tile.reproduction_code):
                    action_performed = self.reproduce(tile.organism_on_tile, neighbour.organism_on_tile)
                else:
                    if self.hide(tile.organism_on_tile, neighbour.organism_on_tile):
                        action_performed=True
                    else:
                        action_performed = self.attack(tile.organism_on_tile, neighbour.organism_on_tile)
            elif neighbour.tile_type == "Meat" or neighbour.tile_type == "Veg" or neighbour.tile_type ==
"Fruit" and action_performed == False:
                action_performed = self.eat_food(tile.organism_on_tile, neighbour.food_on_tile)
            elif neighbour.tile_type == "Water" and action_performed == False:
                self.drink_water(tile.organism_on_tile)
                action_performed = True

    def attack(self, organism_1, organism_2):
        value = 0
        attack_tag_elements = list(organism_1.chromosome.split("|")[0].split(",")[0])
        defence_tag_elements = list(organism_2.chromosome.split("|")[0].split(",")[1])
        for i in range(max(len(attack_tag_elements), len(defence_tag_elements))):
            if i <= len(attack_tag_elements)-1:
                if i > len(defence_tag_elements)-1:
                    value += 1
                elif attack_tag_elements[i] == defence_tag_elements[i]:
                    value += 3
                elif attack_tag_elements[i] == "#":
                    value += 2
                else:
                    value -= 1
        if value > 0:
            resources_given = organism_2.reservoir[0:value]
            organism_2.reservoir = organism_2.reservoir[value:]
            organism_1.reservoir += resources_given
            session.commit()
            if len(organism_2.reservoir) == 0:
                self.convert_to_meat(organism_2)
            return True
        else:
            return False

    def eat_food(self, organism, food):
        food_consumption_code = list(food.consumption_code)
        organism_consumer_code = list(organism.chromosome.split("|")[1].split(",")[2])
        total = 0
        if food.food_type == "Meat":
            for i in range(len(organism_consumer_code)):
                if i<len(food_consumption_code)-1:
                    if organism_consumer_code[i]==food_consumption_code[i]:
                            total+=2
                    if organism_consumer_code[i]=="#":
                        total+=1
            if total > 0:
                organism.reservoir += food.nutritional_value[0:total]
                food.nutritional_value = food.nutritional_value[total:]
                if len(food.nutritional_value)==0:
                    food.food_type = "Deleted"
                    food.tile_relationship.tile_type = "Blank"
                    food.tile_relationship.tile_id = None
                    food.tile_relationship.food_relationship = None
                    session.delete(food)
                session.commit()
                return True

        elif food.food_type == "Fruit":
            for i in range(len(organism_consumer_code)):
                if i<len(food_consumption_code)-1:
                    if organism_consumer_code[i]==food_consumption_code[i]:
                        total+=2
                    if organism_consumer_code[i]=="#":
                        total+=1
            if total > 0:
                organism.reservoir += food.nutritional_value[0:total]
                food.nutritional_value = food.nutritional_value[total:]
                session.commit()
            if len(food.nutritional_value)==0:
                food.tile_relationship.tile_type = "Tree"
                food.food_type = "Tree"
                session.commit()
            if total > 0:
                return True
        elif food.food_type == "Veg":
            for i in range(len(organism_consumer_code)):
                if i<len(food_consumption_code)-1:
                    if organism_consumer_code[i]==food_consumption_code[i]:
                        total+=2
                    if organism_consumer_code[i]=="#":
                        total+=1
            if total > 0:
                organism.reservoir += food.nutritional_value[0:total]
                session.commit()
                return True

        return False
```

Allowing the randomisation of the chromosome to add a new letter to the chromosome.

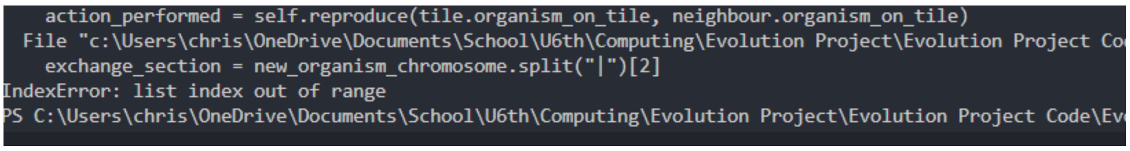
```
def randomize_chromosome(self, tag_section, control_section, exchange_section):
        new_chromosome = ""
        for character in list(tag_section):
            if character == ",":
                new_chromosome+=","
            else:
                random_value = random.randint(0,10)
                if random_value<8:
                    new_chromosome+=character
                elif character == "a":
                    available_letters = ["b","c","d"]
                    new_chromosome+=available_letters[random_value-8]
                elif character == "b":
                    available_letters = ["a","c","d"]
                    new_chromosome+=available_letters[random_value-8]
                elif character == "c":
                    available_letters = ["a","b","d"]
                    new_chromosome+=available_letters[random_value-8]
                elif character == "d":
                    available_letters = ["a","b","c"]
                    new_chromosome+=available_letters[random_value-8]
                if random_value==10:
                    new_chromosome+=["a","b","c","d"][random.randint(0,3)]
        new_chromosome+="|"
        for character in list(control_section):
            if character == ",":
                new_chromosome+=","
            else:
                random_value = random.randint(0,20)
                if random_value<16:
                    new_chromosome+=character
                elif character == "a":
                    available_letters = ["b","b","d","c","#"]
                    new_chromosome+=available_letters[random_value-16]
                elif character == "b":
                    available_letters = ["c","c","a","d","#"]
                    new_chromosome+=available_letters[random_value-16]
                elif character == "c":
                    available_letters = ["d","d","b","a","#"]
                    new_chromosome+=available_letters[random_value-16]
                elif character == "d":
                    available_letters = ["a","a","c","b","#"]
                    new_chromosome+=available_letters[random_value-16]
                elif character == "#":
                    available_letters = ["a","b","c","d","#"]
                    new_chromosome+=available_letters[random_value-16]
                if random_value==20:
                    new_chromosome+=["a","b","c","d","#"][random.randint(0,4)]
        new_chromosome+="|"
        for character in list(exchange_section):
            if character == ",":
                new_chromosome+=","
            else:
                random_value = random.randint(0,20)
                if random_value<16:
                    new_chromosome+=character
                elif character == "a":
                    available_letters = ["b","b","d","c","#"]
                    new_chromosome+=available_letters[random_value-16]
                elif character == "b":
                    available_letters = ["c","c","a","d","#"]
                    new_chromosome+=available_letters[random_value-16]
                elif character == "c":
                    available_letters = ["d","d","b","a","#"]
                    new_chromosome+=available_letters[random_value-16]
                elif character == "d":
                    available_letters = ["a","a","c","b","#"]
                    new_chromosome+=available_letters[random_value-16]
                elif character == "#":
                    available_letters = ["a","b","c","d","#"]
                    new_chromosome+=available_letters[random_value-16]
                if random_value==20:
                    new_chromosome+=["a","b","c","d","#"][random.randint(0,4)]
        return(new_chromosome)
```

Quick Test of the Tweaks Version:

When I try and input data that seems invalid, the program will not send me to the simulation screen but will instead reset the custom detail entry screen.

The organisms were able to survive for longer suggesting that they were more efficiently consuming food.

When I ran the simulation for longer periods of time and two organisms reproduced I got the following error:



```
    action_performed = self.reproduce(tile.organism_on_tile, neighbour.organism_on_tile)
  File "c:\Users\chris\OneDrive\Documents\School\U6th\Computing\Evolution Project\Evolution Project Co
    exchange_section = new_organism_chromosome.split("|")[2]
IndexError: list index out of range
PS C:\Users\chris\OneDrive\Documents\School\U6th\Computing\Evolution Project\Evolution Project Code\Ev
```

This error was caused by the crossing over in reproduce method as if two organisms with different length chromosomes reproduced and their chromosomes crossed over the positions of the commas and pipes wouldn’t be the same and so extra commas and pipes could be added or correct commas and pipes could be removed. I solved this by modifying the crossing over section of the reproduce method as shown below:

```
def reproduce(self, organism_1, organism_2):
    commas_filtered_1 = list(filter(lambda a: a != ",", list(organism_1.chromosome)))
    organism_1_chromosome = list(filter(lambda a: a != "|", commas_filtered_1))
    commas_filtered_2 = list(filter(lambda a: a != ",", list(organism_2.chromosome)))
    organism_2_chromosome = list(filter(lambda a: a != "|", commas_filtered_2))
    if self.ready_to_reproduce(organism_1, organism_2, organism_1_chromosome,
organism_2_chromosome):
        crossover_point = random.randint(1, min(len(organism_1.chromosome)-1,
len(organism_2.chromosome)-1))
        if list(organism_1.chromosome)[crossover_point] == "|" or
list(organism_1.chromosome)[crossover_point] == ",":
            chromosome_1_pipes = organism_1.chromosome.split("|")
            chromosome_2_pipes = organism_2.chromosome.split("|")
            crossover_point = random.randint(0,4)
            if crossover_point ==  1 or crossover_point == 4:
                if crossover_point == 1:
                    new_organism_chromosome =
chromosome_1_pipes[0]+"|"+chromosome_2_pipes[1]+"|"+chromosome_2_pipes[2]
                elif crossover_point == 4:
                    new_organism_chromosome =
chromosome_1_pipes[0]+"|"+chromosome_1_pipes[1]+"|"+chromosome_2_pipes[2]

            elif crossover_point == 0 or crossover_point == 2 or crossover_point == 3:
                if crossover_point == 0:
                    new_organism_chromosome =
chromosome_1_pipes[0].split(",")[0]+","+chromosome_2_pipes[0].split(",")[1]+"|"+chromosome_2_pip
es[1]+"|"+chromosome_2_pipes[2]
                elif crossover_point == 2:
                    new_organism_chromosome =
chromosome_1_pipes[0]+"|"+chromosome_1_pipes[1].split(",")[0]+","+chromosome_2_pipes[1].split(",
")[1]+","+chromosome_2_pipes[1].split(",")[2]+"|"+chromosome_2_pipes[2]
                elif crossover_point == 3:
                    new_organism_chromosome =
chromosome_1_pipes[0]+"|"+chromosome_1_pipes[1].split(",")[0]+","+chromosome_1_pipes[1].split(",
")[1]+","+chromosome_2_pipes[1].split(",")[2]+"|"+chromosome_2_pipes[2]
        else:
            new_organism_chromosome = organism_1.chromosome.split("|")[0] + "|" +
organism_2.chromosome.split("|")[1] + "|" + organism_1.chromosome.split("|")[2]
        tag_section = new_organism_chromosome.split("|")[0]
        control_section = new_organism_chromosome.split("|")[1]
        exchange_section = new_organism_chromosome.split("|")[2]
        new_organism_chromosome = self.randomize_chromosome(tag_section, control_section,
exchange_section)

        organism_placed = False
        for tile in self.tile_array[organism_1.x_position][organism_1.y_position].neighbouring_tiles:
            if tile.tile_type == "Blank" and organism_placed == False:
                new_organism = Organism(chromosome=new_organism_chromosome,
reservoir="abcd", colour=self.find_colour(tag_section, control_section, exchange_section),
reproduction_code=control_section.split(",")[0], x_position=tile.x_coordinate,
y_position=tile.y_coordinate, age=0, species_id=organism_1.species_id,
species_relationship=organism_1.species_relationship, simulation_id=self.simulation.id,
simulation_relationship=self.simulation)
                session.add(new_organism)
```

```
                    tile_at_xy = session.query(Tile).filter(Tile.x_coordinate ==
tile.x_coordinate).filter(Tile.y_coordinate == tile.y_coordinate).filter(Tile.simulation_relationship ==
Program.current_simulation).one()
                    tile_at_xy.organism_id = new_organism.id
                    tile_at_xy.organism_relationship = new_organism
                    tile_at_xy.tile_type = "Organism"
                    session.commit()
                    organism_placed = True
                    self.organisms.append(new_organism)

self.tile_array[new_organism.x_position][new_organism.y_position].tile_type=Organism_Tile(self.simu
lation_map, self.surface, self.simulation_surface, self.block_size,
new_organism.x_position*self.block_size, new_organism.y_position*self.block_size,
new_organism.x_position, new_organism.y_position, self.screen_width, tile_at_xy)
            if organism_placed == False:
                return False
            return True
        else:
            return False
```

Plotting Graphs (Version 1.0):

Overview:

For the plotting of my graphs, I plan to display a series of three graphs. The user should be able to select which graph they wish to display on the screen. The three graphs are described below. They are plotted using matplotlib:

Number of organisms of a species Vs Time:

In this plot I will be displaying the number of organisms that were alive at the end of each turn for each of the individual species. The different coloured lines will correspond to different species.

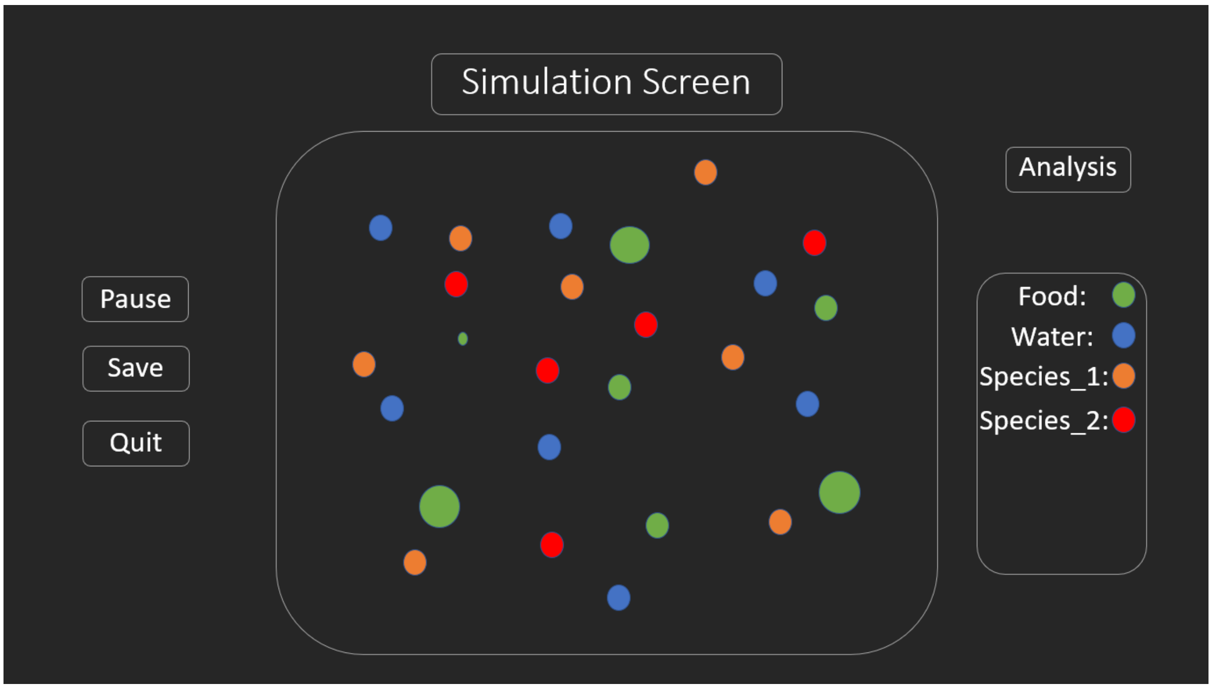
Average Age of organisms of a species Vs Time:

In this plot I will be displaying the average number of turns that the organisms from a species have been alive for. This is also equivalent to the number of resources being deducted after every movement period.

Average Variance from originals' chromosomes of a species Vs Time:

In this plot I will be displaying the average number of characters in the organisms currently in the simulation that belong to the species being drawn that differ from the chromosome of the original organisms of that species.

Proposed Graph Screen Design:



Algorithm Explanation:

The method end_turn - In this version of the end turn method, I carry out a series of calculations for each of the species in the simulation. I then add these details to a new Simulation_History in the database. The calculations are as follows:

I add all the ages of the organisms that belong to that species and I then divide by the number of organisms in that species.

For the other calculation I find the number of characters that are different per organism from the original chromosome and then divide by the number of organisms.

The method draw_graph - When this method is called, start by finding the number of species that need to be plotted on each graph. It will then cycle through to that range. I will then collect all of the species_history and will append the relevant values for the species_history in the current turn being plotted.

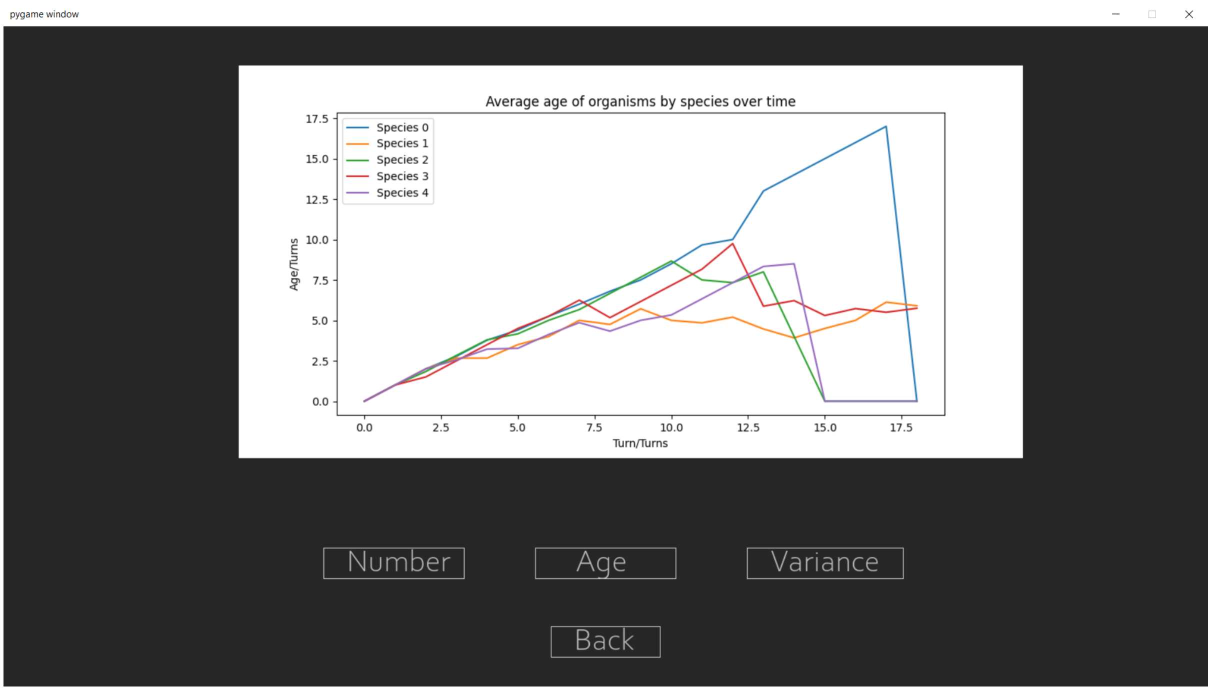
I also update reproduce and convert_to_meat to add or subtract one from the species' number_of_organisms.

Version 1.0 Code:

```
def end_turn(self):
        trees =
session.query(Food).filter(Food.food_type=="Tree").filter(Food.map_relationship==self.simulation_ma
p).all()
        fruits =
session.query(Food).filter(Food.food_type=="Fruit").filter(Food.map_relationship==self.simulation_ma
p).all()
        for tree in trees:
            tree.nutritional_value = "cabdcababcacdacadadcacdacadadc"
            tree.food_type = "Fruit"
            tree.tile_relationship.tile_type = "Fruit"
            session.commit()
        for fruit in fruits:
            fruit.nutritional_value = "cabdcababcacdacadadcacdacadadc"
            session.commit()
        self.simulation.current_turn+=1
        session.commit()
        organisms =
session.query(Organism).filter(Organism.simulation_relationship==self.simulation).all()
        for organism in organisms:
            organism.age += 1
            session.commit()
        for species in self.simulation.species_relationship:
            average_age_total = 0
            average_variance_total = 0
            for organism in species.organism_relationship:
                average_age_total += organism.age
                for i in range(min(len(organism.chromosome),len(species.average_chromosome))):
                    if list(organism.chromosome)[i] != list(species.average_chromosome)[i]:
                        average_variance_total+=1
                average_variance_total +=
abs(len(organism.chromosome)-len(species.average_chromosome))
            try:
                average_age = average_age_total/len(species.organism_relationship)
            except:
                average_age = 0
            try:
                average_variance = average_variance_total/len(species.organism_relationship)
            except:
                average_variance = 0
            new_species_history = Species_History(turn_number = self.simulation.current_turn,
number_of_organisms=len(species.organism_relationship),
average_age_of_organisms=average_age, average_variance_from_originals=average_variance,
species_relationship=species, species_id=species.id)
            session.add(new_species_history)
            session.commit()

class Graph_Screen():
    def __init__(self):
        self.current_graph = 1
        self.screen_width = window.winfo_screenwidth()
        self.screen_height = window.winfo_screenheight()
        self.surface = pygame.display.set_mode((self.screen_width, self.screen_height))
        self.surface.fill(BLACK)

        self.surface_height = int(self.screen_height-100)
        self.surface_width = int(self.screen_width-200)
        surface=self.surface
        self.simulation_surface = pygame.surface.Surface((self.surface_width, self.surface_height))
        self.simulation_surface.fill(BLACK)

        self.quit_button_position = [int(self.screen_width/2)-70, int(self.screen_height)-100]
        self.graph_1_button_position = [int(self.screen_width/2)-360, int(self.screen_height)-200]
        self.graph_2_button_position = [int(self.screen_width/2)-90, int(self.screen_height)-200]
        self.graph_3_button_position = [int(self.screen_width/2) + 180, int(self.screen_height)-200]
        smallfont = pygame.font.SysFont("Corbel",40)
        self.quit_text = smallfont.render("Back", True, WHITE)
        self.graph_1_text = smallfont.render("Number ", True, WHITE)
        self.graph_2_text = smallfont.render("   Age    ", True, WHITE)
        self.graph_3_text = smallfont.render("Variance ", True, WHITE)

        self.draw_graph()
        self.draw_buttons()

        while True:
            mouse = pygame.mouse.get_pos()
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    plt.close(self.fig)
                    pygame.quit()
                    sys.exit()

                if event.type == pygame.MOUSEBUTTONDOWN:
                    if self.quit_button_position[0] <= mouse[0] <= self.quit_button_position[0]+140 and
self.quit_button_position[1] <= mouse[1] <= self.quit_button_position[1]+40:
                        self.quit_graphing()
                    if self.graph_1_button_position[0] <= mouse[0] <= self.graph_1_button_position[0]+1180
and self.graph_1_button_position[1] <= mouse[1] <= self.graph_1_button_position[1]+40:
                        self.current_graph = 1
                        self.draw_graph()
                    if self.graph_2_button_position[0] <= mouse[0] <= self.graph_2_button_position[0]+180
and self.graph_2_button_position[1] <= mouse[1] <= self.graph_2_button_position[1]+40:
                        self.current_graph = 2
                        self.draw_graph()
                    if self.graph_3_button_position[0] <= mouse[0] <= self.graph_3_button_position[0]+200
and self.graph_3_button_position[1] <= mouse[1] <= self.graph_3_button_position[1]+40:
                        self.current_graph = 3
                        self.draw_graph()
            pygame.display.update()

    def draw_graph(self):
        matplotlib.use("Agg")
        self.fig = pylab.figure(figsize=[int(self.screen_width/150),int(self.screen_height/150)],dpi=100)
        ax = self.fig.gca()

        if self.current_graph == 1:
            number_of_species = Program.current_simulation.starting_number_of_species
            for species in range(number_of_species):
                x = list(range(0, Program.current_simulation.current_turn+1))
                y = []
                for turn in range(Program.current_simulation.current_turn+1):
                    species_history_item =
session.query(Species_History).filter(Species_History.species_relationship ==
Program.current_simulation.species_relationship[species]).filter(Species_History.turn_number ==
turn).one()
                    y.append(species_history_item.number_of_organisms)
                label="Species " + str(species)
                plt.plot(x, y, label = label)
            ax.set(xlabel='Turn/Turns', ylabel='Number of organisms')
            plt.title('Number of organism by species over time')
            plt.legend()
        if self.current_graph == 2:
            number_of_species = Program.current_simulation.starting_number_of_species
            for species in range(number_of_species):
                x = list(range(0, Program.current_simulation.current_turn+1))
                y = []
                for turn in range(Program.current_simulation.current_turn+1):
                    species_history_item =
session.query(Species_History).filter(Species_History.species_relationship ==
Program.current_simulation.species_relationship[species]).filter(Species_History.turn_number ==
turn).one()
                    y.append(species_history_item.average_age_of_organisms)
                label="Species " + str(species)
                plt.plot(x, y, label = label)
            ax.set(xlabel='Turn/Turns', ylabel='Age/Turns')
            plt.title('Average age of organisms by species over time')
            plt.legend()
        if self.current_graph == 3:
            number_of_species = Program.current_simulation.starting_number_of_species
            for species in range(number_of_species):
                x = list(range(0, Program.current_simulation.current_turn+1))
                y = []
                for turn in range(Program.current_simulation.current_turn+1):
                    species_history_item =
session.query(Species_History).filter(Species_History.species_relationship ==
Program.current_simulation.species_relationship[species]).filter(Species_History.turn_number ==
turn).one()
                    y.append(species_history_item.average_variance_from_originals)
                label="Species " + str(species)
                plt.plot(x, y, label = label)
            ax.set(xlabel='Turn/Turns', ylabel='Variation from species\' original chromosome')
            plt.title('Genetic variance from original organisms over time')
            plt.legend()

        canvas = agg.FigureCanvasAgg(self.fig)
        canvas.draw()
        renderer = canvas.get_renderer()
        raw_data = renderer.tostring_rgb()

        size = canvas.get_width_height()

        surf = pygame.image.fromstring(raw_data, size, "RGB")
        self.surface.blit(surf, (300,50))
        pygame.display.flip()

    def draw_buttons(self):

pygame.draw.rect(self.surface,WHITE,[self.quit_button_position[0],self.quit_button_position[1],140,40]
,1)
        self.surface.blit(self.quit_text , (self.quit_button_position[0]+30,self.quit_button_position[1]))


pygame.draw.rect(self.surface,WHITE,[self.graph_1_button_position[0],self.graph_1_button_position[
1],180,40],1)
        self.surface.blit(self.graph_1_text ,
(self.graph_1_button_position[0]+30,self.graph_1_button_position[1]))
```

```
pygame.draw.rect(self.surface,WHITE,[self.graph_2_button_position[0],self.graph_2_button_position[
1],180,40],1)
        self.surface.blit(self.graph_2_text ,
(self.graph_2_button_position[0]+30,self.graph_2_button_position[1]))


pygame.draw.rect(self.surface,WHITE,[self.graph_3_button_position[0],self.graph_3_button_position[
1],200,40],1)
        self.surface.blit(self.graph_3_text ,
(self.graph_3_button_position[0]+30,self.graph_3_button_position[1]))

    def quit_graphing(self):
        plt.close(self.fig)
        Simulation_Screen()
```

Graph Screen I created:



Testing the Graphs:

When I have run the simulation for a while, and then try to plot the graphs, they seem to be correct for that simulation that was run.

The data is plotted for the whole history of the species, not just the section that was just run.

Finalised Version:

Requirements:

Python Version 3.8.0
Pygame Version 1.8.1
Pygame_menu Version 4.0.1
Matplotlib Version 3.4.0
Numpy Version 1.20.2
Tkinter Version 3.9.2
Sqlalchemy Version 1.4.3

Database Schema:

```
import datetime
import sqlalchemy
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, relationship
from sqlalchemy import Column, Integer, String, ForeignKey, DateTime, Float

Base = declarative_base()

engine = create_engine('sqlite:///evolution_simulator.db', echo=True)
Session = sessionmaker(bind=engine)
Base.metadata.create_all(engine)
session = Session()

class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    name = Column(String(20))
    username = Column(String(20))
    password = Column(String(20))

    simulations_relationship = relationship("Simulation", back_populates='user_relationship')

    def __repr__(self):
        return "<User(id: {0}, name: {1}, username: {2}, password: {3})>".format(self.id, self.name,
self.username, self.password)

class Simulation(Base):
    __tablename__ = "simulations"
    id = Column(Integer, primary_key=True)
    simulation_name = Column(String(50))
    date_last_used = Column(DateTime)
    current_turn = Column(Integer)
    length_of_turn = Column(Integer)
    starting_number_of_organisms = Column(Integer)
    starting_number_of_species = Column(Integer)
    current_number_of_organisms = Column(Integer)

    user_id = Column(ForeignKey("users.id"))
    user_relationship = relationship("User", back_populates='simulations_relationship')

    species_relationship = relationship("Species", back_populates='simulation_relationship')

    organism_relationship = relationship("Organism", back_populates='simulation_relationship')

    map_relationship = relationship("Map", back_populates='simulation_relationship')

    tile_relationship = relationship("Tile", back_populates='simulation_relationship')

    def __repr__(self):
        return "<Simulation(id: {0}, simulation name: {1}, date last used: {2}, current turn: {3}, length of
turn: {4}, current organisms: {5}, user: {6})>".format(self.id, self.simulation_name, self.date_last_used,
self.current_turn, self.length_of_turn, self.current_number_of_organisms, self.user_id)

class Species(Base):
    __tablename__ = "species"
    id = Column(Integer, primary_key=True)
    average_reproduction_code = Column(String(50))
    average_chromosome = Column(String(100))
    average_species_colour = Column(String(10))

    simulation_id = Column(ForeignKey("simulations.id"))
    simulation_relationship = relationship("Simulation", back_populates='species_relationship')

    organism_relationship = relationship("Organism", back_populates='species_relationship')

    species_history_relationship = relationship("Species_History",
back_populates='species_relationship')

    def __repr__(self):
        return "<Species(species id: {0}, reproduction code: {1}, species colour: {2}, simulation:
{3})>".format(self.id, self.average_reproduction_code, self.average_species_colour, self.simulation_id)

class Species_History(Base):
    __tablename__ = "species_history"
    id = Column(Integer, primary_key=True)
    turn_number = Column(Integer)
    number_of_organisms = Column(Integer)
    average_age_of_organisms = Column(Float)
    average_variance_from_originals = Column(Float)

    species_id = Column(ForeignKey("species.id"))
    species_relationship = relationship("Species", back_populates='species_history_relationship')
```

```
class Organism(Base):
    __tablename__ = "organisms"
    id = Column(Integer, primary_key=True)
    chromosome = Column(String(1000))
    reservoir = Column(String(1000))
    colour = Column(String(10))
    reproduction_code = Column(String(50))
    x_position = Column(Integer)
    y_position = Column(Integer)
    age = Column(Integer)

    species_id = Column(ForeignKey("species.id"))
    species_relationship = relationship("Species", back_populates='organism_relationship')

    simulation_id = Column(ForeignKey("simulations.id"))
    simulation_relationship = relationship("Simulation", back_populates='organism_relationship')

    tile_relationship = relationship("Tile", uselist=False, back_populates='organism_relationship')

    def __repr__(self):
        return "<Organism(id: {0}, chromosome: {1}, reservoir: {2}, colour: {3}, reproduction_code: {4},
position: ({5}:{6}), age: {7}, tile relationship: {8})>".format(self.id, self.chromosome, self.reservoir,
self.colour, self.reproduction_code, self.x_position, self.y_position, self.age, self.tile_relationship)

class Map(Base):
    __tablename__ = "maps"
    id = Column(Integer, primary_key=True)
    map_size = Column(Integer)
    number_of_food_sources = Column(Integer)
    meat_to_veg_ratio = Column(String(4))
    number_of_water_sources = Column(Integer)

    simulation_id = Column(ForeignKey("simulations.id"))
    simulation_relationship = relationship("Simulation", back_populates='map_relationship')

    food_relationship = relationship("Food", back_populates="map_relationship")

    def __repr__(self):
        return "<Map(map id: {0}, map size: {1}, number of food sources: {2}, number of water sources:
{3}, simulation id: {4})>".format(self.id, self.map_size, self.number_of_food_sources,
self.number_of_water_sources, self.simulation_id)

class Food(Base):
    __tablename__ = "foods"
    id = Column(Integer, primary_key=True)
    food_type = Column(Integer)
    nutritional_value = Column(Integer)
    consumption_code = Column(String(20))
    x_coordinate = Column(Integer)
    y_coordinate = Column(Integer)

    map_id = Column(ForeignKey("maps.id"))
    map_relationship = relationship("Map", back_populates='food_relationship')

    tile_relationship = relationship("Tile", uselist=False, back_populates='food_relationship')

    def __repr__(self):
        return "<Food(food id: {0}, food type: {1}, nutritional value: {2}, consumption code: {3}, map id:
{4})>".format(self.id, self.food_type, self.nutritional_value, self.consumption_code, self.map_id)
```

```
class Tile(Base):
    __tablename__ = "tiles"
    id = Column(Integer, primary_key=True)
    x_coordinate = Column(Integer)
    y_coordinate = Column(Integer)
    x_position = Column(Integer)
    y_position = Column(Integer)
    tile_type = Column(String(20))

    simulation_id = Column(ForeignKey("simulations.id"))
    simulation_relationship = relationship("Simulation", back_populates='tile_relationship')

    food_id = Column(ForeignKey("foods.id"))
    food_relationship = relationship("Food", back_populates='tile_relationship')

    organism_id = Column(ForeignKey("organisms.id"))
    organism_relationship = relationship("Organism", back_populates='tile_relationship')

    def __repr__(self):
        return "<Tile(tile id: {0}, x coordinate: {1}, y coordinate: {2}, position: ({3}:{4}), tile type:
{5})>".format(self.id, self.x_coordinate, self.y_coordinate, self.x_position, self.y_position, self.tile_type)

Base.metadata.create_all(engine)
session = Session()
session.commit()
```

Program Code:

```
import random
import pygame
import pygame_menu
import time
import matplotlib
import matplotlib.backends.backend_agg as agg
import matplotlib.pyplot as plt
import pylab
import numpy as np
from pygame.locals import *
from tkinter import *
window = Tk()
pygame.init()
from Evolution_simulator_schema_v6 import User, Simulation, Species, Species_History, Organism,
Map, Food, Tile
from sqlalchemy.orm import sessionmaker
from sqlalchemy import create_engine
from datetime import datetime, date
engine = create_engine("sqlite:///evolution_simulator.db", echo=False)
session = sessionmaker(bind=engine)()
BLACK = (38,38,38)
WHITE = (200,200,200)
GREEN = (153,186,56)
class Program():
        def __init__(self):
        self.user_logged_in = None
        self.current_simulation = None
        Welcome_screen()
class Welcome_screen():
        def __init__(self):
        SCREEN_WIDTH = window.winfo_screenwidth()
        SCREEN_HEIGHT = window.winfo_screenheight()
        surface = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT), FULLSCREEN)
        self.menu = pygame_menu.Menu(SCREEN_HEIGHT, SCREEN_WIDTH, "Evolution
Simulator", theme=pygame_menu.themes.THEME_DARK)
        self.menu.add_label("Evolution Simulator")
        self.menu.add_button("Sign-Up", self.sign_in)
        self.menu.add_button("Log In", self.log_in)
        self.menu.add_button("Quit", pygame_menu.events.EXIT)
        self.menu.mainloop(surface)
        def sign_in(self):
        Signup_screen()
        def log_in(self):
        Login_screen()
class Signup_screen():
        def __init__(self):
        SCREEN_WIDTH = window.winfo_screenwidth()
        SCREEN_HEIGHT = window.winfo_screenheight()
        surface = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT), FULLSCREEN)
        self.menu = pygame_menu.Menu(SCREEN_HEIGHT, SCREEN_WIDTH, "Sign Up",
theme=pygame_menu.themes.THEME_DARK)
        self.input_name = self.menu.add_text_input("Name:  ", default="")
        self.input_username = self.menu.add_text_input("Username:  ", default="")
        self.input_password = self.menu.add_text_input("Password:  ", default="", password=True)
        self.input_repeat_password = self.menu.add_text_input("Repeat Password:   ", default="",
password=True)
        self.menu.add_label("")
        self.menu.add_button("Sign Up", self.sign_up)
        self.menu.add_button("Back", self.welcome_screen)
        self.menu.mainloop(surface)
        def welcome_screen(self):
        Welcome_screen()
        def home_screen(self):
        Home_screen()
        def sign_up(self):
        data = self.menu.get_input_data(recursive=False)
        input_name = data[self.input_name.get_id()]
        input_username = data[self.input_username.get_id()]
        input_password = data[self.input_password.get_id()]
        input_repeat_password = data[self.input_repeat_password.get_id()]

        self.validate_data(input_name, input_username, input_password, input_repeat_password)
        def validate_data(self, name, username, password, repeat_password):
        usernames = session.query(User).filter(User.username == username).all()
        if len(usernames) != 0:
        Signup_screen()
        if password != repeat_password:
        Signup_screen()
        self.add_data_to_database(name, username, password, repeat_password)
        def add_data_to_database(self, name, username, password, repeat_password):
        user = User(name=name, username=username, password=password)
        session.add(user)
        session.commit()
        Program.user_logged_in = user
        self.home_screen()
class Login_screen():
        def __init__(self):
        SCREEN_WIDTH = window.winfo_screenwidth()
        SCREEN_HEIGHT = window.winfo_screenheight()
        surface = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT), FULLSCREEN)
        self.menu = pygame_menu.Menu(SCREEN_HEIGHT, SCREEN_WIDTH, "Log In",
theme=pygame_menu.themes.THEME_DARK)
```

```
        self.username = self.menu.add_text_input("Username:  ", default="")
        self.password = self.menu.add_text_input("Password:  ", default="", password=True)
        self.menu.add_button("Log In", self.log_in)
        self.menu.add_button("Back", self.welcome_screen)
        self.menu.mainloop(surface)
        def welcome_screen(self):
        Welcome_screen()

        def home_screen(self):
        Home_screen()
        def log_in(self):
        data = self.menu.get_input_data(recursive=False)
        username = data[self.username.get_id()]
        password = data[self.password.get_id()]
        user = session.query(User).filter(User.username == username).one_or_none()
        if user == None:
        Login_screen()
        if user.password  == password:
        Program.user_logged_in = user
        self.home_screen()
        else:
        Login_screen()
class Home_screen():
        def __init__(self):
        SCREEN_WIDTH = window.winfo_screenwidth()
        SCREEN_HEIGHT = window.winfo_screenheight()
        surface = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT), FULLSCREEN)
        self.menu = pygame_menu.Menu(SCREEN_HEIGHT, SCREEN_WIDTH, "Home",
theme=pygame_menu.themes.THEME_DARK)
        self.menu.add_label("Welcome back " + Program.user_logged_in.name)
        self.menu.add_button("New Simulation", self.new_simulation_screen)
        self.menu.add_button("Saved Simulations", self.saved_simulations_screen)
        self.menu.add_button("Log Out", self.log_out)
        self.menu.mainloop(surface)
        def welcome_screen(self):
        Welcome_screen()
        def new_simulation_screen(self):
        New_simulation_screen()
        def saved_simulations_screen(self):
        Saved_Simulations_screen()
        def log_out(self):
        Program.user_logged_in = None
        self.welcome_screen()
class New_simulation_screen():
        def __init__(self):
        self.current_generation_type = "Custom"
        SCREEN_WIDTH = window.winfo_screenwidth()
        SCREEN_HEIGHT = window.winfo_screenheight()
        surface = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT), FULLSCREEN)
        self.menu = pygame_menu.Menu(SCREEN_HEIGHT, SCREEN_WIDTH, "Simulation
Generation", theme=pygame_menu.themes.THEME_DARK)
        self.menu.add_selector("Simulation Variables:", [("        Custom", 1), ("Randomized", 2)],
onchange=self.change_input)
        self.menu.add_button("Create Simulation", self.create_simulation)
        self.menu.add_button("Back to Home Screen", self.home_screen)
        self.menu.mainloop(surface)
        def home_screen(self):
        Home_screen()
        def generate_simulation(self):
        Simulation_Screen()
        def change_input(self, val_1, val_2):
        if val_2 == 1:
        self.current_generation_type = "Custom"
        if val_2 == 2:
        self.current_generation_type = "Random"
        def create_simulation(self):
        if self.current_generation_type == "Custom":
        Custom_Detail_Entry_screen()
        elif self.current_generation_type == "Random":
        self.randomly_generate()
        def randomly_generate(self):
        simulation_name = "Simulation_" + str(date.today())
        date_created = datetime.now()
        length_of_turn = random.randint(5,20)
        starting_number_of_organisms = random.randint(5,20)
        starting_number_of_species = random.randint(2,10)

        map_size = random.randint(5,30)
        number_of_food_sources = random.randint(2,10)
        meat_to_veg_ratio = str(random.randint(1,5))+":"+str(random.randint(1,5))
        number_of_water_sources = random.randint(2,10)
        simulation = Simulation(simulation_name=simulation_name, date_last_used=date_created,
current_turn=0, length_of_turn=length_of_turn,
starting_number_of_organisms=starting_number_of_organisms,
starting_number_of_species=starting_number_of_species,
current_number_of_organisms=starting_number_of_organisms, user_id=Program.user_logged_in.id,
user_relationship=Program.user_logged_in)
        session.add(simulation)
        session.commit()
        map_to_add = Map(map_size=map_size,
number_of_food_sources=number_of_food_sources, meat_to_veg_ratio=meat_to_veg_ratio,
number_of_water_sources=number_of_water_sources, simulation_id=simulation.id,
simulation_relationship=simulation)
        session.add(map_to_add)
        session.commit()
        Program.current_simulation = simulation

        self.generate_simulation()
class Custom_Detail_Entry_screen():
        def __init__(self):
        SCREEN_WIDTH = window.winfo_screenwidth()
        SCREEN_HEIGHT = window.winfo_screenheight()
        surface = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT), FULLSCREEN)
        self.menu = pygame_menu.Menu(SCREEN_HEIGHT, SCREEN_WIDTH, "Custom Detail
Entry", theme=pygame_menu.themes.THEME_DARK)
        self.menu.add_label("Simulation Details")
        self.simulation_name = self.menu.add_text_input("Name of Simulation:  ",
default="Simulation_")
        self.length_of_turn = self.menu.add_text_input("Length of Turn:  ", default="10")
        self.starting_number_of_organisms = self.menu.add_text_input("Starting Number of
Organisms:  ", default="20")
        self.starting_number_of_species = self.menu.add_text_input("Starting Number of Species:  ",
default="4")
        self.menu.add_label("")
        self.menu.add_label("Map Details")
        self.map_size = self.menu.add_text_input("Map Size:  ", default="10")
        self.number_of_food_sources = self.menu.add_text_input("Number of Food Sources:  ",
default="20")
        self.meat_to_veg_ratio = self.menu.add_text_input("Meat to Veg Ratio:  ", default="1:4")
        self.number_of_water_sources = self.menu.add_text_input("Number of Water Sources:  ",
default="10")
        self.menu.add_label("")
        self.menu.add_button("Generate", self.collect_input_data)
        self.menu.add_button("Back", self.home_screen)
        self.menu.mainloop(surface)
        def home_screen(self):
        Home_screen()
        def generating_screen(self):
        Simulation_Screen()
        def collect_input_data(self):
        data = self.menu.get_input_data(recursive=False)
        simulation_name = data[self.simulation_name.get_id()]
        date_created = datetime.now()
        length_of_turn = data[self.length_of_turn.get_id()]
        starting_number_of_organisms = data[self.starting_number_of_organisms.get_id()]
        starting_number_of_species = data[self.starting_number_of_species.get_id()]

        map_size = data[self.map_size.get_id()]
        number_of_food_sources = data[self.number_of_food_sources.get_id()]
        meat_to_veg_ratio = data[self.meat_to_veg_ratio.get_id()]
        number_of_water_sources = data[self.number_of_water_sources.get_id()]
        if self.validate_data(length_of_turn, starting_number_of_organisms,
starting_number_of_species, map_size, number_of_food_sources, meat_to_veg_ratio,
number_of_water_sources):
        self.generate(simulation_name, date_created, length_of_turn,
starting_number_of_organisms, starting_number_of_species, map_size, number_of_food_sources,
meat_to_veg_ratio, number_of_water_sources)
        else:
        Custom_Detail_Entry_screen()

        def validate_data(self, length_of_turn, starting_number_of_organisms,
starting_number_of_species, map_size, number_of_food_sources, meat_to_veg_ratio,
number_of_water_sources):
        try:
        int(length_of_turn)
        int(starting_number_of_organisms)
        int(starting_number_of_species)
        int(map_size)
        int(number_of_food_sources)
        int(number_of_water_sources)
        except:
        return False
        if int(length_of_turn) <= 0 or int(starting_number_of_organisms) <= 0 or
int(starting_number_of_species) <= 0 or int(map_size) <= 0 or int(number_of_food_sources) < 0 or
int(number_of_water_sources) < 0:
        return False
        ratio = meat_to_veg_ratio.split(":")
        if len(ratio) != 2:
        return False
        try:
        int(ratio[0])
        int(ratio[1])
        if int(ratio[0]) < 0 or int(ratio[1]) < 0 or (ratio[0] == "0" and ratio[1] == "0"):
        return False
        except:
        return False
        if int(starting_number_of_species) > int(starting_number_of_organisms):
        return False
        if int(starting_number_of_organisms) + int(number_of_food_sources) +
int(number_of_water_sources) > int(map_size)**2:
        return False
        return True
        def generate(self, simulation_name, date_created, lenth_of_turn,
starting_number_of_organisms, starting_number_of_species, map_size, number_of_food_sources,
meat_to_veg_ratio, number_of_water_sources):
        current_simulations = session.query(Simulation).filter(Simulation.user_relationship ==
Program.user_logged_in).order_by(Simulation.date_last_used.desc()).all()
        if len(current_simulations) >= 3:
        session.delete(current_simulations[-1])
        simulation = Simulation(simulation_name=simulation_name, date_last_used=date_created,
current_turn=0, length_of_turn=lenth_of_turn,
starting_number_of_organisms=starting_number_of_organisms,
starting_number_of_species=starting_number_of_species,
current_number_of_organisms=starting_number_of_organisms, user_id=Program.user_logged_in.id,
user_relationship=Program.user_logged_in)
        map_to_add = Map(map_size=map_size,
number_of_food_sources=number_of_food_sources, meat_to_veg_ratio=meat_to_veg_ratio,
number_of_water_sources=number_of_water_sources, simulation_id=simulation.id,
simulation_relationship=simulation)
        session.add(simulation)
        session.commit()
        Program.current_simulation = simulation
        self.generating_screen()
class Saved_Simulations_screen():
        def __init__(self):
        SCREEN_WIDTH = window.winfo_screenwidth()
        SCREEN_HEIGHT = window.winfo_screenheight()
        surface = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT), FULLSCREEN)
        self.menu = pygame_menu.Menu(SCREEN_HEIGHT, SCREEN_WIDTH, "Saved
Simulations", theme=pygame_menu.themes.THEME_DARK)
        user_simulations = session.query(Simulation).filter(Simulation.user_relationship ==
Program.user_logged_in).order_by(Simulation.date_last_used.desc()).all()
        if len(user_simulations) == 0:
        self.menu.add_label("You have no saved simulations")
        else:
        self.menu.add_button(user_simulations[0].simulation_name, self.load_simulation_1)
        try:
        self.menu.add_button(user_simulations[1].simulation_name, self.load_simulation_2)
        except:
        pass
        try:
        self.menu.add_button(user_simulations[2].simulation_name, self.load_simulation_3)
        except:
        pass
        self.menu.add_button("Back", self.home_screen)
        self.menu.mainloop(surface)
        def home_screen(self):
        Home_screen()
        def load_simulation_1(self):
        simulation = session.query(Simulation).filter(Simulation.user_relationship ==
Program.user_logged_in).order_by(Simulation.date_last_used.desc()).first()
        simulation.date_last_used = datetime.now()
        session.commit()
        Program.current_simulation=simulation
        self.generate()

        def load_simulation_2(self):
```

```
        simulations = session.query(Simulation).filter(Simulation.user_relationship ==
Program.user_logged_in).order_by(Simulation.date_last_used.desc()).all()
        simulation = simulations[1]
        simulation.date_last_used = datetime.now()
        session.commit()
        Program.current_simulation=simulation
        self.generate()
        def load_simulation_3(self):
        simulations = session.query(Simulation).filter(Simulation.user_relationship ==
Program.user_logged_in).order_by(Simulation.date_last_used.desc()).all()
        simulation = simulations[2]
        simulation.date_last_used = datetime.now()
        session.commit()
        Program.current_simulation=simulation
        self.generate()
        def generate(self):
        Simulation_Screen()
class Simulation_Screen():
        def __init__(self):
        self.simulation_state = 0
        self.simulation = Program.current_simulation
        self.simulation_map = self.simulation.map_relationship[0]
        self.map_size = self.simulation_map.map_size
        self.current_turn = 1
        self.simulation_length_of_turn = self.simulation.length_of_turn
        self.screen_width = window.winfo_screenwidth()
        self.screen_height = window.winfo_screenheight()
        self.surface = pygame.display.set_mode((self.screen_width, self.screen_height))
        self.surface.fill(BLACK)
        self.surface_height = int(self.screen_height-100)
        self.block_size = int(self.surface_height/self.simulation_map.map_size)
        self.surface_width = int(self.block_size*self.simulation_map.map_size)

        surface=self.surface
        self.simulation_surface = pygame.surface.Surface((self.surface_width, self.surface_height))
        self.simulation_surface.fill(BLACK)
        self.pause_button_position = [100,300]
        self.quit_button_position = [100, 370]
        self.analysis_button_position = [1300,300]
        smallfont = pygame.font.SysFont("Corbel",40)
        self.start_text = smallfont.render("Start" , True , WHITE)
        self.pause_text = smallfont.render("Pause" , True , WHITE)
        self.play_text = smallfont.render("Play" , True , WHITE)
        self.quit_text = smallfont.render("Quit", True, WHITE)
        self.analysis_text = smallfont.render("Analysis", True, WHITE)
        self.tile_array = [[None for i in range(self.map_size)] for j in range(self.map_size)]
        self.organisms = []
        self.organism_tiles = []
        if len(self.simulation.tile_relationship) == 0:
        self.set_up_grid()
        self.position_water()
        self.position_veg_and_fruit()
        self.position_meat()
        self.create_species()
        self.create_organisms()
        for species in
session.query(Species).filter(Species.simulation_relationship==self.simulation).all():
        new_species_history = Species_History(turn_number = self.simulation.current_turn,
number_of_organisms=len(species.organism_relationship), average_age_of_organisms=0,
average_variance_from_originals=0, species_relationship=species, species_id=species.id)
```

```
            session.add(new_species_history)
            session.commit()
            self.create_tiles()
            self.find_tile_neighbours()
            self.draw_buttons()
            self.draw_tiles()
            while True:
            mouse = pygame.mouse.get_pos()
            for event in pygame.event.get():
            if event.type == pygame.QUIT:
                        pygame.quit()
                        sys.exit()

            if event.type == pygame.MOUSEBUTTONDOWN:
                        if self.pause_button_position[0] <= mouse[0] <= self.pause_button_position[0]+140
and self.pause_button_position[1] <= mouse[1] <= self.pause_button_position[1]+40:
                        if self.simulation_state==0:
                        self.start()
                        elif self.simulation_state==1:
                        self.pause()
                        elif self.simulation_state==2:
                        self.play()
                        if self.quit_button_position[0] <= mouse[0] <= self.quit_button_position[0]+140 and
self.quit_button_position[1] <= mouse[1] <= self.quit_button_position[1]+40:
                        self.quit_simulation()
                        if self.analysis_button_position[0] <= mouse[0] <=
self.analysis_button_position[0]+180 and self.analysis_button_position[1] <= mouse[1] <=
self.analysis_button_position[1]+40:
                        self.analysis()
            pygame.display.update()

            if self.simulation_state == 1:
            self.organism_tiles = []
            self.move_organisms()
            self.tile_array = [[None for i in range(self.map_size)] for j in range(self.map_size)]
            self.create_tiles()
            self.draw_tiles()
            self.find_tile_neighbours()
            for tile in self.organism_tiles:
                        tile.organism_on_tile.reservoir =
tile.organism_on_tile.reservoir[tile.organism_on_tile.age:]
                        session.commit()
                        if len(tile.organism_on_tile.reservoir) == 0:
                        self.convert_to_meat(tile.organism_on_tile)
                        else:
                        self.check_area_around(tile)
            if self.current_turn%self.simulation_length_of_turn == 0:
                        self.end_turn()
            self.current_turn += 1

            def set_up_grid(self):
            for x in range(self.map_size):
            for y in range(self.map_size):
            tile = Tile(x_coordinate=x, y_coordinate=y, x_position=x*self.block_size,
y_position=y*self.block_size, tile_type="Blank", simulation_id=self.simulation.id,
simulation_relationship=self.simulation, food_id=None, food_relationship=None, organism_id=None,
organism_relationship=None)
            session.add(tile)
            session.commit()
            def position_water(self):
```

```
            for i in range(self.simulation_map.number_of_water_sources):
            x = random.randint(0, self.map_size-1)
            y = random.randint(0, self.map_size-1)
            tile_at_xy = session.query(Tile).filter(Tile.x_coordinate == x).filter(Tile.y_coordinate ==
y).filter(Tile.simulation_relationship == Program.current_simulation).one()
            while tile_at_xy.tile_type != "Blank":
            x = random.randint(0, self.map_size-1)
            y = random.randint(0, self.map_size-1)
            tile_at_xy = session.query(Tile).filter(Tile.x_coordinate == x).filter(Tile.y_coordinate ==
y).filter(Tile.simulation_relationship == Program.current_simulation).one()
            tile_at_xy.tile_type = "Water"
            session.commit()

            def position_veg_and_fruit(self):
            proportion_of_veg =
int(self.simulation_map.meat_to_veg_ratio.split(":")[1])/(int(self.simulation_map.meat_to_veg_ratio.spli
t(":")[0])+int(self.simulation_map.meat_to_veg_ratio.split(":")[1]))
            for i in range(round(self.simulation_map.number_of_food_sources*proportion_of_veg)):
            x = random.randint(0, self.map_size-1)
            y = random.randint(0, self.map_size-1)
            tile_at_xy = session.query(Tile).filter(Tile.x_coordinate == x).filter(Tile.y_coordinate ==
y).filter(Tile.simulation_relationship == Program.current_simulation).one()
            while tile_at_xy.tile_type != "Blank":
            x = random.randint(0, self.map_size-1)
            y = random.randint(0, self.map_size-1)
            tile_at_xy = session.query(Tile).filter(Tile.x_coordinate == x).filter(Tile.y_coordinate ==
y).filter(Tile.simulation_relationship == Program.current_simulation).one()
            if random.randint(0,10) <= 5:
            tile_at_xy.tile_type = "Veg"
            veg_item = Food(food_type="Veg", nutritional_value="acdacadadcacdacadadcacdacadadc",
consumption_code="abbcabcbdb", x_coordinate=x, y_coordinate=y, map_id=self.simulation_map.id,
map_relationship=self.simulation_map, tile_relationship=tile_at_xy)
            session.add(veg_item)
            tile_at_xy.food_relationship=veg_item
            tile_at_xy.food_id=veg_item.id
            else:
            tile_at_xy.tile_type = "Fruit"
            fruit_item = Food(food_type="Fruit", nutritional_value="cabdcababccabdcababccabdcababc",
consumption_code = "addcacbcdc", x_coordinate=x, y_coordinate=y, map_id=self.simulation_map.id,
map_relationship=self.simulation_map, tile_relationship=tile_at_xy)
            session.add(fruit_item)
            tile_at_xy.food_relationship=fruit_item
            tile_at_xy.food_id=fruit_item.id
            session.commit()
            def position_meat(self):
            proportion_of_meat =
int(self.simulation_map.meat_to_veg_ratio.split(":")[0])/(int(self.simulation_map.meat_to_veg_ratio.spli
t(":")[0])+int(self.simulation_map.meat_to_veg_ratio.split(":")[1]))
            for i in range(round(self.simulation_map.number_of_food_sources*proportion_of_meat)):
            x = random.randint(0, self.map_size-1)
            y = random.randint(0, self.map_size-1)
            tile_at_xy = session.query(Tile).filter(Tile.x_coordinate == x).filter(Tile.y_coordinate ==
y).filter(Tile.simulation_relationship == self.simulation).one()
            while tile_at_xy.tile_type != "Blank":
            x = random.randint(0, self.map_size-1)
            y = random.randint(0, self.map_size-1)
            tile_at_xy = session.query(Tile).filter(Tile.x_coordinate == x).filter(Tile.y_coordinate ==
y).filter(Tile.simulation_relationship == self.simulation).one()
            tile_at_xy.tile_type = "Meat"
            meat_item = Food(food_type="Meat", nutritional_value="ddbddabadc",
consumption_code="cdccdddccd", x_coordinate=x, y_coordinate=y, map_id=self.simulation_map.id,
map_relationship=self.simulation_map, tile_relationship=tile_at_xy)
            tile_at_xy.food_relationship=meat_item
            tile_at_xy.food_id=meat_item.id
            session.add(meat_item)
            session.commit()
            def create_species(self):
            standard_chromosomes =
["abbaca,cdcc|baadb,aaaa,abc|abccc","abadcc,bbbb|a,bbbb,abdc|ac,ac"]
            for species in range(self.simulation.starting_number_of_species):
            new_species_chromosome =
standard_chromosomes[random.randint(0,len(standard_chromosomes)-1)]
            tag_section = new_species_chromosome.split("|")[0]
            control_section = new_species_chromosome.split("|")[1]
            exchange_section = new_species_chromosome.split("|")[2]
            new_species_chromosome = self.randomize_chromosome(tag_section, control_section,
exchange_section)
            tag_section = new_species_chromosome.split("|")[0]
            control_section = new_species_chromosome.split("|")[1]
            exchange_section = new_species_chromosome.split("|")[2]
            new_species_colour = self.find_colour(tag_section, control_section, exchange_section)
            new_species_reproduction_code = control_section.split(",")[1]
            new_species = Species(average_chromosome=new_species_chromosome,
average_species_colour=new_species_colour,
average_reproduction_code=new_species_reproduction_code, simulation_id=self.simulation.id,
simulation_relationship=self.simulation)
            session.add(new_species)
            session.commit()
            def create_organisms(self):
            species = session.query(Species).filter(Species.simulation_relationship==self.simulation).all()
            for current_species in species:
            for i in
range(int(self.simulation.starting_number_of_organisms/self.simulation.starting_number_of_species)):
            x = random.randint(0, self.map_size-1)
            y = random.randint(0, self.map_size-1)
            tile_at_xy = session.query(Tile).filter(Tile.x_coordinate == x).filter(Tile.y_coordinate ==
y).filter(Tile.simulation_relationship == Program.current_simulation).one()
            while tile_at_xy.tile_type != "Blank":
                        x = random.randint(0, self.map_size-1)
                        y = random.randint(0, self.map_size-1)
                        tile_at_xy = session.query(Tile).filter(Tile.x_coordinate == x).filter(Tile.y_coordinate ==
y).filter(Tile.simulation_relationship == Program.current_simulation).one()
            tile_at_xy.tile_type = "Organism"
            new_organism = Organism(chromosome=current_species.average_chromosome,
reservoir="abcd", colour=current_species.average_species_colour,
reproduction_code=current_species.average_reproduction_code, x_position=x, y_position=y, age=0,
species_id=current_species.id, species_relationship=current_species,
simulation_id=self.simulation.id, simulation_relationship=self.simulation)
            session.add(new_organism)
            tile_at_xy.organism_id = new_organism.id
            tile_at_xy.organism_relationship = new_organism
            session.commit()
            def randomize_chromosome(self, tag_section, control_section, exchange_section):
            new_chromosome = ""
            for character in list(tag_section):
            if character == ",":
            new_chromosome+=","
            else:
            random_value = random.randint(0,10)
```

```
if random_value<8:
        new_chromosome+=character
elif character == "a":
        available_letters = ["b","c","d"]
        new_chromosome+=available_letters[random_value-8]
elif character == "b":
        available_letters = ["a","c","d"]
        new_chromosome+=available_letters[random_value-8]
elif character == "c":
        available_letters = ["a","b","d"]
        new_chromosome+=available_letters[random_value-8]
elif character == "d":
        available_letters = ["a","b","c"]
        new_chromosome+=available_letters[random_value-8]
if random_value==10:
        new_chromosome+=["a","b","c","d"][random.randint(0,3)]
new_chromosome+="|"
for character in list(control_section):
if character == ",":
new_chromosome+=","
else:
random_value = random.randint(0,20)
if random_value<16:
        new_chromosome+=character
elif character == "a":
        available_letters = ["b","b","d","c","#"]
        new_chromosome+=available_letters[random_value-16]
elif character == "b":
        available_letters = ["c","c","a","d","#"]
        new_chromosome+=available_letters[random_value-16]
elif character == "c":
        available_letters = ["d","d","b","a","#"]
        new_chromosome+=available_letters[random_value-16]
elif character == "d":
        available_letters = ["a","a","c","b","#"]
        new_chromosome+=available_letters[random_value-16]
elif character == "#":
        available_letters = ["a","b","c","d","#"]
        new_chromosome+=available_letters[random_value-16]
if random_value==20:
        new_chromosome+=["a","b","c","d","#"][random.randint(0,4)]
new_chromosome+="|"
for character in list(exchange_section):
if character == ",":
new_chromosome+=","
else:
random_value = random.randint(0,20)
if random_value<16:
        new_chromosome+=character
elif character == "a":
        available_letters = ["b","b","d","c","#"]
        new_chromosome+=available_letters[random_value-16]
elif character == "b":
        available_letters = ["c","c","a","d","#"]
        new_chromosome+=available_letters[random_value-16]
elif character == "c":
        available_letters = ["d","d","b","a","#"]
        new_chromosome+=available_letters[random_value-16]
elif character == "d":
        available_letters = ["a","a","c","b","#"]
```

```
                new_chromosome+=available_letters[random_value-16]
        elif character == "#":
                available_letters = ["a","b","c","d","#"]
                new_chromosome+=available_letters[random_value-16]
        if random_value==20:
                new_chromosome+=["a","b","c","d","#"][random.randint(0,4)]
        return(new_chromosome)
        def find_colour(self, tag_section, control_section, exchange_section):
        R_value = 0
        for value in list(tag_section):
        if value != ",":
        R_value += ord(value)
        R_value = R_value%256

        G_value = 0
        for value in list(control_section):
        if value != ",":
        G_value += ord(value)
        G_value = G_value%256
        B_value = 0
        for condition in list(exchange_section):
        for value in list(condition):
        if value != ",":
                B_value += ord(value)
        B_value = B_value%256
        return(str(R_value)+","+str(G_value)+","+str(B_value))
        def create_tiles(self):
        for x in range(self.map_size):
        for y in range(self.map_size):
        tile_at_xy = session.query(Tile).filter(Tile.x_coordinate == x).filter(Tile.y_coordinate ==
y).filter(Tile.simulation_relationship == Program.current_simulation).one()
        if tile_at_xy.tile_type == "Water":
                self.tile_array[x][y] = Water_Tile(self.surface, self.simulation_surface, self.block_size,
x*self.block_size, y*self.block_size, x, y, self.screen_width)
        elif tile_at_xy.tile_type == "Veg":
                self.tile_array[x][y] = Veg_Tile(self.simulation_map, self.surface,
self.simulation_surface, self.block_size, x*self.block_size, y*self.block_size, x, y, self.screen_width)
        elif tile_at_xy.tile_type == "Fruit":
                self.tile_array[x][y] = Fruit_Tile(self.simulation_map, self.surface,
self.simulation_surface, self.block_size, x*self.block_size, y*self.block_size, x, y, self.screen_width,
True)
        elif tile_at_xy.tile_type == "Tree":
                self.tile_array[x][y] = Fruit_Tile(self.simulation_map, self.surface,
self.simulation_surface, self.block_size, x*self.block_size, y*self.block_size, x, y, self.screen_width,
False)
        elif tile_at_xy.tile_type == "Meat":
                self.tile_array[x][y] = Meat_Tile(self.simulation_map, self.surface,
self.simulation_surface, self.block_size, x*self.block_size, y*self.block_size, x, y, self.screen_width)
        elif tile_at_xy.tile_type == "Organism":
                self.tile_array[x][y] = Organism_Tile(self.simulation_map, self.surface,
self.simulation_surface, self.block_size, x*self.block_size, y*self.block_size, x, y, self.screen_width,
tile_at_xy)
                self.organism_tiles.append(self.tile_array[x][y])
        else:
                self.tile_array[x][y] = Simulation_Tile(self.surface, self.simulation_surface,
self.block_size, x*self.block_size, y*self.block_size, x, y, self.screen_width)
        def find_tile_neighbours(self):
        for tile_row in self.tile_array:
        for tile in tile_row:
        if tile.x_coordinate > 0:
                tile.neighbouring_tiles.append(self.tile_array[tile.x_coordinate-1][tile.y_coordinate])
        if tile.x_coordinate < self.map_size-1:
                tile.neighbouring_tiles.append(self.tile_array[tile.x_coordinate+1][tile.y_coordinate])
        if tile.y_coordinate > 0:
                tile.neighbouring_tiles.append(self.tile_array[tile.x_coordinate][tile.y_coordinate-1])
        if tile.y_coordinate < self.map_size-1:
                tile.neighbouring_tiles.append(self.tile_array[tile.x_coordinate][tile.y_coordinate+1])
        def draw_tiles(self):
        for x in range(self.map_size):
        for y in range(self.map_size):
        self.tile_array[x][y].draw_tile()
        if self.tile_array[x][y].tile_type == "Organism":
                self.tile_array[x][y].draw_organism()
        def draw_buttons(self):
        if self.simulation_state == 0:

pygame.draw.rect(self.surface,BLACK,[self.pause_button_position[0],self.pause_button_position[1],14
0,40])

pygame.draw.rect(self.surface,WHITE,[self.pause_button_position[0],self.pause_button_position[1],14
0,40],1)
        self.surface.blit(self.start_text ,
(self.pause_button_position[0]+30,self.pause_button_position[1]))
        elif self.simulation_state == 1:

pygame.draw.rect(self.surface,BLACK,[self.pause_button_position[0],self.pause_button_position[1],14
0,40])

pygame.draw.rect(self.surface,WHITE,[self.pause_button_position[0],self.pause_button_position[1],14
0,40],1)
        self.surface.blit(self.pause_text ,
(self.pause_button_position[0]+30,self.pause_button_position[1]))
        elif self.simulation_state == 2:

pygame.draw.rect(self.surface,BLACK,[self.pause_button_position[0],self.pause_button_position[1],14
0,40])

pygame.draw.rect(self.surface,WHITE,[self.pause_button_position[0],self.pause_button_position[1],14
0,40],1)
        self.surface.blit(self.play_text ,
(self.pause_button_position[0]+30,self.pause_button_position[1]))

pygame.draw.rect(self.surface,WHITE,[self.quit_button_position[0],self.quit_button_position[1],140,40]
,1)
        self.surface.blit(self.quit_text , (self.quit_button_position[0]+30,self.quit_button_position[1]))

pygame.draw.rect(self.surface,WHITE,[self.analysis_button_position[0],self.analysis_button_position[
1],180,40],1)
        self.surface.blit(self.analysis_text ,
(self.analysis_button_position[0]+30,self.analysis_button_position[1]))
        def move_organisms(self):
        for organism_to_move in self.organisms:
        movement_direction = random.randint(0,4)
        if movement_direction == 0:
        movement_direction = self.move_up(organism_to_move)

        elif movement_direction == 1:
        movement_direction = self.move_right(organism_to_move)

        elif movement_direction == 2:
```

```
        movement_direction = self.move_down(organism_to_move)
        elif movement_direction == 3:
        movement_direction = self.move_left(organism_to_move)
        elif movement_direction == 4:
        pass
        def move_up(self, organism_to_move):
        if organism_to_move.y_position > 0:
        if
self.tile_array[organism_to_move.x_position][organism_to_move.y_position-1].tile_type=="Blank":
        tile_to_move_from = organism_to_move.tile_relationship
        tile_to_move_to =
session.query(Tile).filter(Tile.x_coordinate==organism_to_move.x_position).filter(Tile.y_coordinate==o
rganism_to_move.y_position-1).filter(Tile.simulation_relationship==self.simulation).one()
        tile_to_move_from.tile_type = "Blank"
        tile_to_move_to.tile_type = "Organism"
        self.tile_array[organism_to_move.x_position][organism_to_move.y_position].tile_type="Blank"

self.tile_array[organism_to_move.x_position][organism_to_move.y_position-1].tile_type="Organism"
        tile_to_move_from.organism_relationship = None
        tile_to_move_from.organism_id = None
        tile_to_move_to.organism_relationship = organism_to_move
        tile_to_move_to.organism_id = organism_to_move.id
        organism_to_move.y_position -= 1
        session.commit()
        return 0
        else:
        return 4
        else:
        return(4)
        def move_right(self, organism_to_move):
        if organism_to_move.x_position < self.map_size-1:
        if
self.tile_array[organism_to_move.x_position+1][organism_to_move.y_position].tile_type=="Blank":
        tile_to_move_from = organism_to_move.tile_relationship
        tile_to_move_to =
session.query(Tile).filter(Tile.x_coordinate==organism_to_move.x_position+1).filter(Tile.y_coordinate=
=organism_to_move.y_position).filter(Tile.simulation_relationship==self.simulation).one()
        tile_to_move_from.tile_type = "Blank"
        tile_to_move_to.tile_type = "Organism"
        self.tile_array[organism_to_move.x_position][organism_to_move.y_position].tile_type="Blank"

self.tile_array[organism_to_move.x_position+1][organism_to_move.y_position].tile_type="Organism"
        tile_to_move_from.organism_relationship = None
        tile_to_move_from.organism_id = None
        tile_to_move_to.organism_relationship = organism_to_move
        tile_to_move_to.organism_id = organism_to_move.id
        organism_to_move.x_position += 1
        session.commit()
        return 1
        else:
        return 4
        else:
        return(4)
        def move_down(self, organism_to_move):
        if organism_to_move.y_position < self.map_size-1:
        if
self.tile_array[organism_to_move.x_position][organism_to_move.y_position+1].tile_type=="Blank":
        tile_to_move_from = organism_to_move.tile_relationship
```

```
            tile_to_move_to =
session.query(Tile).filter(Tile.x_coordinate==organism_to_move.x_position).filter(Tile.y_coordinate==o
rganism_to_move.y_position+1).filter(Tile.simulation_relationship==self.simulation).one()
            tile_to_move_from.tile_type = "Blank"
            tile_to_move_to.tile_type = "Organism"
            self.tile_array[organism_to_move.x_position][organism_to_move.y_position].tile_type="Blank"

self.tile_array[organism_to_move.x_position][organism_to_move.y_position+1].tile_type="Organism"
            tile_to_move_from.organism_relationship = None
            tile_to_move_from.organism_id = None
            tile_to_move_to.organism_relationship = organism_to_move
            tile_to_move_to.organism_id = organism_to_move.id
            organism_to_move.y_position += 1
            session.commit()
            return 2
            else:
            return 4
            else:
            return(4)
            def move_left(self, organism_to_move):
            if organism_to_move.x_position > 0:
            if
self.tile_array[organism_to_move.x_position-1][organism_to_move.y_position].tile_type=="Blank":
            tile_to_move_from = organism_to_move.tile_relationship
            tile_to_move_to =
session.query(Tile).filter(Tile.x_coordinate==organism_to_move.x_position-1).filter(Tile.y_coordinate=
=organism_to_move.y_position).filter(Tile.simulation_relationship==self.simulation).one()
            tile_to_move_from.tile_type = "Blank"
            tile_to_move_to.tile_type = "Organism"
            self.tile_array[organism_to_move.x_position][organism_to_move.y_position].tile_type="Blank"

self.tile_array[organism_to_move.x_position-1][organism_to_move.y_position].tile_type="Organism"
            tile_to_move_from.organism_relationship = None
            tile_to_move_from.organism_id = None
            tile_to_move_to.organism_relationship = organism_to_move
            tile_to_move_to.organism_id = organism_to_move.id
            organism_to_move.x_position -= 1
            session.commit()
            return 3
            else:
            return 4
            else:
            return 4

            def check_area_around(self, tile):
            for neighbour in tile.neighbouring_tiles:
            action_performed = False
            if neighbour.tile_type == "Organism":
            if self.match(neighbour.organism_on_tile.reproduction_code,
tile.organism_on_tile.reproduction_code):
                        action_performed = self.reproduce(tile.organism_on_tile, neighbour.organism_on_tile)
            else:
                        if self.hide(tile.organism_on_tile, neighbour.organism_on_tile):
                        action_performed=True
                        else:
                        action_performed = self.attack(tile.organism_on_tile, neighbour.organism_on_tile)
            elif neighbour.tile_type == "Meat" or neighbour.tile_type == "Veg" or neighbour.tile_type ==
"Fruit" and action_performed == False:
            action_performed = self.eat_food(tile.organism_on_tile, neighbour.food_on_tile)
            elif neighbour.tile_type == "Water" and action_performed == False:
```

```
            self.drink_water(tile.organism_on_tile)
            action_performed = True
            def match(self, organism_1_code, organism_2_code):
            match = False
            total = 0
            for i in range(min(len(organism_1_code), len(organism_2_code))):
            if list(organism_1_code)[i]==list(organism_2_code)[i]:
            total+=1
            if total >= 0.75*min(len(organism_1_code), len(organism_2_code)):
            match = True
            return match
            def reproduce(self, organism_1, organism_2):
            commas_filtered_1 = list(filter(lambda a: a != ",", list(organism_1.chromosome)))
            organism_1_chromosome = list(filter(lambda a: a != "|", commas_filtered_1))
            commas_filtered_2 = list(filter(lambda a: a != ",", list(organism_2.chromosome)))
            organism_2_chromosome = list(filter(lambda a: a != "|", commas_filtered_2))
            if self.ready_to_reproduce(organism_1, organism_2, organism_1_chromosome,
organism_2_chromosome):
            crossover_point = random.randint(1, min(len(organism_1.chromosome)-1,
len(organism_2.chromosome)-1))
            if list(organism_1.chromosome)[crossover_point] == "|" or
list(organism_1.chromosome)[crossover_point] == ",":
            chromosome_1_pipes = organism_1.chromosome.split("|")
            chromosome_2_pipes = organism_2.chromosome.split("|")
            crossover_point = random.randint(0,4)
            if crossover_point ==  1 or crossover_point == 4:
                        if crossover_point == 1:
                        new_organism_chromosome =
chromosome_1_pipes[0]+"|"+chromosome_2_pipes[1]+"|"+chromosome_2_pipes[2]
                        elif crossover_point == 4:
                        new_organism_chromosome =
chromosome_1_pipes[0]+"|"+chromosome_1_pipes[1]+"|"+chromosome_2_pipes[2]
            elif crossover_point == 0 or crossover_point == 2 or crossover_point == 3:
                        if crossover_point == 0:
                        new_organism_chromosome =
chromosome_1_pipes[0].split(",")[0]+","+chromosome_2_pipes[0].split(",")[1]+"|"+chromosome_2_pip
es[1]+"|"+chromosome_2_pipes[2]
                        elif crossover_point == 2:
                        new_organism_chromosome =
chromosome_1_pipes[0]+"|"+chromosome_1_pipes[1].split(",")[0]+","+chromosome_2_pipes[1].split(",
")[1]+","+chromosome_2_pipes[1].split(",")[2]+"|"+chromosome_2_pipes[2]
                        elif crossover_point == 3:
                        new_organism_chromosome =
chromosome_1_pipes[0]+"|"+chromosome_1_pipes[1].split(",")[0]+","+chromosome_1_pipes[1].split(",
")[1]+","+chromosome_2_pipes[1].split(",")[2]+"|"+chromosome_2_pipes[2]
            else:
            new_organism_chromosome = organism_1.chromosome.split("|")[0] + "|" +
organism_2.chromosome.split("|")[1] + "|" + organism_1.chromosome.split("|")[2]
            tag_section = new_organism_chromosome.split("|")[0]
            control_section = new_organism_chromosome.split("|")[1]
            exchange_section = new_organism_chromosome.split("|")[2]
            new_organism_chromosome = self.randomize_chromosome(tag_section, control_section,
exchange_section)
            organism_placed = False
            for tile in self.tile_array[organism_1.x_position][organism_1.y_position].neighbouring_tiles:
            if tile.tile_type == "Blank" and organism_placed == False:
                        new_organism = Organism(chromosome=new_organism_chromosome,
reservoir="abcd", colour=self.find_colour(tag_section, control_section, exchange_section),
reproduction_code=control_section.split(",")[0], x_position=tile.x_coordinate,
y_position=tile.y_coordinate, age=0, species_id=organism_1.species_id,
species_relationship=organism_1.species_relationship, simulation_id=self.simulation.id,
simulation_relationship=self.simulation)
                        session.add(new_organism)
                        tile_at_xy = session.query(Tile).filter(Tile.x_coordinate ==
tile.x_coordinate).filter(Tile.y_coordinate == tile.y_coordinate).filter(Tile.simulation_relationship ==
Program.current_simulation).one()
                        tile_at_xy.organism_id = new_organism.id
                        tile_at_xy.organism_relationship = new_organism
                        tile_at_xy.tile_type = "Organism"
                        self.simulation.current_number_of_organisms+=1
                        session.commit()
                        organism_placed = True
                        self.organisms.append(new_organism)

self.tile_array[new_organism.x_position][new_organism.y_position].tile_type=Organism_Tile(self.simu
lation_map, self.surface, self.simulation_surface, self.block_size,
new_organism.x_position*self.block_size, new_organism.y_position*self.block_size,
new_organism.x_position, new_organism.y_position, self.screen_width, tile_at_xy)
            if organism_placed == False:
            return False
            return True
            else:
            return False
            def ready_to_reproduce(self, organism_1, organism_2, chromosome_value_1,
chromosome_value_2):
            has_resources = True
            reservoir_copy_1 = list(organism_1.reservoir)
            reservoir_copy_2 = list(organism_2.reservoir)
            for character in chromosome_value_1[:int(len(chromosome_value_1))]:
            try:
            reservoir_copy_1.remove(character)
            except:
            has_resources = False
            for character in chromosome_value_2[int(len(chromosome_value_1)):]:
            try:
            reservoir_copy_2.remove(character)
            except:
            has_resources = False
            return has_resources

            def hide(self, organism_1, organism_2):
            hide = False
            attack_tag = list(organism_1.chromosome.split("|")[0].split(",")[0])
            hide_tag = list(organism_2.chromosome.split("|")[1].split(",")[0])
            if len(hide_tag) < len(attack_tag):
            for i in range(len(hide_tag)):
            if hide_tag[i] != attack_tag[i] and hide_tag[i] != "#":
                        hide = True
            else:
            hide = True
            return(hide)
            def attack(self, organism_1, organism_2):
            value = 0
            attack_tag_elements = list(organism_1.chromosome.split("|")[0].split(",")[0])
            defence_tag_elements = list(organism_2.chromosome.split("|")[0].split(",")[1])
            for i in range(max(len(attack_tag_elements), len(defence_tag_elements))):
            if i <= len(attack_tag_elements)-1:
            if i > len(defence_tag_elements)-1:
                        value += 1
            elif attack_tag_elements[i] == defence_tag_elements[i]:
                    value += 3
            elif attack_tag_elements[i] == "#":
                    value += 2
            else:
                    value -= 1
            if value > 0:
            resources_given = organism_2.reservoir[0:value]
            organism_2.reservoir = organism_2.reservoir[value:]
            organism_1.reservoir += resources_given
            session.commit()
            if len(organism_2.reservoir) == 0:
            self.convert_to_meat(organism_2)
            return True
            else:
            return False
            def convert_to_meat(self, organism_to_convert):
            nutritional_value=""
            for letter in list(organism_to_convert.chromosome):
            if letter != "," and letter != "|":
            nutritional_value += letter
            new_food = Food(food_type="Meat", nutritional_value=nutritional_value,
consumption_code="cdccdddccd", x_coordinate=organism_to_convert.x_position,
y_coordinate=organism_to_convert.y_position, map_id=self.simulation_map.id,
map_relationship=self.simulation_map, tile_relationship=organism_to_convert.tile_relationship)

            tile = organism_to_convert.tile_relationship
            tile.tile_type = "Meat"
            tile.organism_relationship = None
            tile.organism_id = None
            tile.food_relationship = new_food
            tile.food_id = new_food.id
            session.add(new_food)
            session.commit()

self.organism_tiles.remove(self.tile_array[organism_to_convert.x_position][organism_to_convert.y_po
sition])
            meat_tile = Meat_Tile(self.simulation_map, self.surface, self.simulation_surface,
self.block_size, organism_to_convert.x_position*self.block_size,
organism_to_convert.y_position*self.block_size, organism_to_convert.x_position,
organism_to_convert.y_position, self.screen_width)

            neighbours =
self.tile_array[organism_to_convert.x_position][organism_to_convert.y_position].neighbouring_tiles
            for neighbour in neighbours:

neighbour.neighbouring_tiles[neighbour.neighbouring_tiles.index(self.tile_array[organism_to_convert.
x_position][organism_to_convert.y_position])] = meat_tile
            self.tile_array[organism_to_convert.x_position][organism_to_convert.y_position] = meat_tile
            self.tile_array[organism_to_convert.x_position][organism_to_convert.y_position].tile_type =
"Meat"

self.tile_array[organism_to_convert.x_position][organism_to_convert.y_position].neighbouring_tiles =
neighbours
            organism_to_convert.species_relationship = None
            organism_to_convert.species_id = None
            organism_to_convert.simulation_relationship = None
            organism_to_convert.simulation_id = None
            self.organisms.remove(organism_to_convert)
            session.delete(organism_to_convert)
            self.simulation.current_number_of_organisms-=1
```

```
session.commit()

def eat_food(self, organism, food):
food_consumption_code = list(food.consumption_code)
organism_consumer_code = list(organism.chromosome.split("|")[1].split(",")[2])
total = 0
if food.food_type == "Meat":
for i in range(len(organism_consumer_code)):
if i<len(food_consumption_code)-1:
        if organism_consumer_code[i]==food_consumption_code[i]:
        total+=2
        if organism_consumer_code[i]=="#":
        total+=1
if total > 0:
organism.reservoir += food.nutritional_value[0:total]
food.nutritional_value = food.nutritional_value[total:]
if len(food.nutritional_value)==0:
        food.food_type = "Deleted"
        food.tile_relationship.tile_type = "Blank"
        food.tile_relationship.tile_id = None
        food.tile_relationship.food_relationship = None
        session.delete(food)
session.commit()
return True
elif food.food_type == "Fruit":
for i in range(len(organism_consumer_code)):
if i<len(food_consumption_code)-1:
        if organism_consumer_code[i]==food_consumption_code[i]:
        total+=2
        if organism_consumer_code[i]=="#":
        total+=1
if total > 0:
organism.reservoir += food.nutritional_value[0:total]
food.nutritional_value = food.nutritional_value[total:]
session.commit()
if len(food.nutritional_value)==0:
food.tile_relationship.tile_type = "Tree"
food.food_type = "Tree"
session.commit()
if total > 0:
return True

elif food.food_type == "Veg":
for i in range(len(organism_consumer_code)):
if i<len(food_consumption_code)-1:
        if organism_consumer_code[i]==food_consumption_code[i]:
        total+=2
        if organism_consumer_code[i]=="#":
        total+=1
if total > 0:
organism.reservoir += food.nutritional_value[0:total]
session.commit()
return True

return False
def drink_water(self, organism):
organism.reservoir += "##"
session.commit()
def end_turn(self):
        trees =
session.query(Food).filter(Food.food_type=="Tree").filter(Food.map_relationship==self.simulation_ma
p).all()
        fruits =
session.query(Food).filter(Food.food_type=="Fruit").filter(Food.map_relationship==self.simulation_ma
p).all()
        for tree in trees:
        tree.nutritional_value = "cabdcababcacdacadadcacdacadadc"
        tree.food_type = "Fruit"
        tree.tile_relationship.tile_type = "Fruit"
        session.commit()
        for fruit in fruits:
        fruit.nutritional_value = "cabdcababcacdacadadcacdacadadc"
        session.commit()
        self.simulation.current_turn+=1
        session.commit()
        organisms =
session.query(Organism).filter(Organism.simulation_relationship==self.simulation).all()
        for organism in organisms:
        organism.age += 1
        session.commit()
        for species in self.simulation.species_relationship:
        average_age_total = 0
        average_variance_total = 0
        for organism in species.organism_relationship:
        average_age_total += organism.age
        for i in range(min(len(organism.chromosome),len(species.average_chromosome))):
                if list(organism.chromosome)[i] != list(species.average_chromosome)[i]:
                average_variance_total+=1
        average_variance_total +=
abs(len(organism.chromosome)-len(species.average_chromosome))
        try:
        average_age = average_age_total/len(species.organism_relationship)
        except:
        average_age = 0
        try:
        average_variance = average_variance_total/len(species.organism_relationship)
        except:
        average_variance = 0
        new_species_history = Species_History(turn_number = self.simulation.current_turn,
number_of_organisms=len(species.organism_relationship),
average_age_of_organisms=average_age, average_variance_from_originals=average_variance,
species_relationship=species, species_id=species.id)
        session.add(new_species_history)
        session.commit()
        def start(self):
        self.simulation_state = 1
        self.draw_buttons()
        self.organisms =
session.query(Organism).filter(Organism.simulation_id==self.simulation.id).all()
        def pause(self):
        self.simulation_state = 2
        self.draw_buttons()
        def play(self):
        self.simulation_state = 1
        self.draw_buttons()
        def quit_simulation(self):
        Program.current_simulation = None
        Home_screen()
        def analysis(self):
        self.simulation_state = 2
        self.draw_buttons()
        Graph_Screen()
class Simulation_Tile():
        def __init__(self, surface, simulation_surface, block_size, x_position, y_position,
x_coordinate, y_coordinate, screen_width):
        self.surface = surface
        self.simulation_surface = simulation_surface
        self.block_size = block_size
        self.x_coordinate = x_coordinate
        self.y_coordinate = y_coordinate
        self.x_position = x_position
        self.y_position = y_position
        self.screen_width = screen_width
        self.image = "Grass.png"
        self.tile_type = "Blank"
        self.neighbouring_tiles = []
        def draw_tile(self):
        image_to_draw = pygame.image.load(self.image).convert_alpha()
        scaled_image = pygame.transform.scale(image_to_draw, (self.block_size, self.block_size))
        self.simulation_surface.blit(scaled_image, pygame.Rect(self.x_position, self.y_position,
self.block_size, self.block_size).topleft)
        self.surface.blit(self.simulation_surface, (int(self.screen_width*(1/4)),50))
class Water_Tile(Simulation_Tile):
        def __init__(self, surface, simulation_surface, block_size, x_position, y_position,
x_coordinate, y_coordinate, screen_width):
        super().__init__(surface, simulation_surface, block_size, x_position, y_position, x_coordinate,
y_coordinate, screen_width)
        self.image = "Water_on_grass.png"
        self.tile_type = "Water"
class Veg_Tile(Simulation_Tile):
        def __init__(self, food_map, surface, simulation_surface, block_size, x_position, y_position,
x_coordinate, y_coordinate, screen_width):
        super().__init__(surface, simulation_surface, block_size, x_position, y_position, x_coordinate,
y_coordinate, screen_width)
        self.food_on_tile =
session.query(Food).filter(Food.x_coordinate==x_coordinate).filter(Food.y_coordinate==y_coordinate)
.filter(Food.map_id==food_map.id).one()
        self.image = "Grass_on_grass.png"
        self.tile_type = "Veg"
class Fruit_Tile(Simulation_Tile):
        def __init__(self, food_map, surface, simulation_surface, block_size, x_position, y_position,
x_coordinate, y_coordinate, screen_width, fruit_on_tree):
        super().__init__(surface, simulation_surface, block_size, x_position, y_position, x_coordinate,
y_coordinate, screen_width)
        self.food_on_tile =
session.query(Food).filter(Food.x_coordinate==x_coordinate).filter(Food.y_coordinate==y_coordinate)
.filter(Food.map_id==food_map.id).one()
        if fruit_on_tree:
        self.image = "Fruit_on_grass_with_fruit.png"
        else:
        self.image = "Fruit_on_grass.png"
        self.tile_type = "Fruit"
class Meat_Tile(Simulation_Tile):
        def __init__(self, food_map, surface, simulation_surface, block_size, x_position, y_position,
x_coordinate, y_coordinate, screen_width):
        super().__init__(surface, simulation_surface, block_size, x_position, y_position, x_coordinate,
y_coordinate, screen_width)
```

```
        self.food_on_tile =
session.query(Food).filter(Food.x_coordinate==x_coordinate).filter(Food.y_coordinate==y_coordinate)
.filter(Food.map_id==food_map.id).one()
        self.image = "Meat_on_grass.png"
        self.tile_type = "Meat"
class Organism_Tile(Simulation_Tile):
        def __init__(self, simulation, surface, simulation_surface, block_size, x_position, y_position,
x_coordinate, y_coordinate, screen_width, tile_at_xy):
        super().__init__(surface, simulation_surface, block_size, x_position, y_position, x_coordinate,
y_coordinate, screen_width)
        self.organism_on_tile = tile_at_xy.organism_relationship
        r, g, b = self.organism_on_tile.colour.strip().split(",")
        self.block_size = block_size
        self.colour_tuple = (int(r),int(g),int(b))
        self.simulation_surface = simulation_surface
        self.x_position = x_position
        self.y_position = y_position
        self.tile_type = "Organism"
        self.block_size = block_size
        def draw_organism(self):
        pygame.draw.circle(self.simulation_surface, self.colour_tuple,
(int(self.x_position+(self.block_size/2)),int(self.y_position+(self.block_size/2))), int(self.block_size/7))
        pygame.draw.circle(self.simulation_surface, (0,0,0),
(int(self.x_position+(self.block_size/2)),int(self.y_position+(self.block_size/2))), int(self.block_size/7),
1)
class Graph_Screen():
        def __init__(self):
        self.current_graph = 1
        self.screen_width = window.winfo_screenwidth()
        self.screen_height = window.winfo_screenheight()
        self.surface = pygame.display.set_mode((self.screen_width, self.screen_height))
        self.surface.fill(BLACK)
        self.surface_height = int(self.screen_height-100)
        self.surface_width = int(self.screen_width-200)

        surface=self.surface
        self.simulation_surface = pygame.surface.Surface((self.surface_width, self.surface_height))
        self.simulation_surface.fill(BLACK)
        self.quit_button_position = [int(self.screen_width/2)-70, int(self.screen_height)-100]
        self.graph_1_button_position = [int(self.screen_width/2)-360, int(self.screen_height)-200]
        self.graph_2_button_position = [int(self.screen_width/2)-90, int(self.screen_height)-200]
        self.graph_3_button_position = [int(self.screen_width/2) + 180, int(self.screen_height)-200]
        smallfont = pygame.font.SysFont("Corbel",40)
        self.quit_text = smallfont.render("Back", True, WHITE)
        self.graph_1_text = smallfont.render("Number ", True, WHITE)
        self.graph_2_text = smallfont.render("   Age    ", True, WHITE)
        self.graph_3_text = smallfont.render("Variance ", True, WHITE)
        self.draw_graph()
        self.draw_buttons()
        while True:
        mouse = pygame.mouse.get_pos()
        for event in pygame.event.get():
        if event.type == pygame.QUIT:
                plt.close(self.fig)
                pygame.quit()
                sys.exit()

        if event.type == pygame.MOUSEBUTTONDOWN:
                if self.quit_button_position[0] <= mouse[0] <= self.quit_button_position[0]+140 and
self.quit_button_position[1] <= mouse[1] <= self.quit_button_position[1]+40:
```

```
                    self.quit_graphing()
                    if self.graph_1_button_position[0] <= mouse[0] <=
self.graph_1_button_position[0]+1180 and self.graph_1_button_position[1] <= mouse[1] <=
self.graph_1_button_position[1]+40:
                    self.current_graph = 1
                    self.draw_graph()
                    if self.graph_2_button_position[0] <= mouse[0] <=
self.graph_2_button_position[0]+180 and self.graph_2_button_position[1] <= mouse[1] <=
self.graph_2_button_position[1]+40:
                    self.current_graph = 2
                    self.draw_graph()
                    if self.graph_3_button_position[0] <= mouse[0] <=
self.graph_3_button_position[0]+200 and self.graph_3_button_position[1] <= mouse[1] <=
self.graph_3_button_position[1]+40:
                    self.current_graph = 3
                    self.draw_graph()
          pygame.display.update()
          def draw_graph(self):
          matplotlib.use("Agg")
          self.fig = pylab.figure(figsize=[int(self.screen_width/150),int(self.screen_height/150)],dpi=100)
          ax = self.fig.gca()

          if self.current_graph == 1:
          number_of_species = Program.current_simulation.starting_number_of_species
          for species in range(number_of_species):
          x = list(range(0, Program.current_simulation.current_turn+1))
          y = []
          for turn in range(Program.current_simulation.current_turn+1):
                    species_history_item =
session.query(Species_History).filter(Species_History.species_relationship ==
Program.current_simulation.species_relationship[species]).filter(Species_History.turn_number ==
turn).one()
                    y.append(species_history_item.number_of_organisms)
          label="Species " + str(species)
          plt.plot(x, y, label = label)
          ax.set(xlabel='Turn/Turns', ylabel='Number of organisms')
          plt.title('Number of organism by species over time')
          plt.legend()
          if self.current_graph == 2:
          number_of_species = Program.current_simulation.starting_number_of_species
          for species in range(number_of_species):
          x = list(range(0, Program.current_simulation.current_turn+1))
          y = []
          for turn in range(Program.current_simulation.current_turn+1):
                    species_history_item =
session.query(Species_History).filter(Species_History.species_relationship ==
Program.current_simulation.species_relationship[species]).filter(Species_History.turn_number ==
turn).one()
                    y.append(species_history_item.average_age_of_organisms)
          label="Species " + str(species)
          plt.plot(x, y, label = label)
          ax.set(xlabel='Turn/Turns', ylabel='Age/Turns')
          plt.title('Average age of organisms by species over time')
          plt.legend()
          if self.current_graph == 3:
          number_of_species = Program.current_simulation.starting_number_of_species
          for species in range(number_of_species):
          x = list(range(0, Program.current_simulation.current_turn+1))
          y = []
          for turn in range(Program.current_simulation.current_turn+1):
```

```
                        species_history_item =
session.query(Species_History).filter(Species_History.species_relationship ==
Program.current_simulation.species_relationship[species]).filter(Species_History.turn_number ==
turn).one()
                        y.append(species_history_item.average_variance_from_originals)
            label="Species " + str(species)
            plt.plot(x, y, label = label)
            ax.set(xlabel='Turn/Turns', ylabel='Variation from species\' original chromosome')
            plt.title('Genetic variance from original organisms over time')
            plt.legend()
            canvas = agg.FigureCanvasAgg(self.fig)
            canvas.draw()
            renderer = canvas.get_renderer()
            raw_data = renderer.tostring_rgb()
            size = canvas.get_width_height()
            surf = pygame.image.fromstring(raw_data, size, "RGB")
            self.surface.blit(surf, (300,50))
            pygame.display.flip()

            def draw_buttons(self):

pygame.draw.rect(self.surface,WHITE,[self.quit_button_position[0],self.quit_button_position[1],140,40]
,1)
            self.surface.blit(self.quit_text , (self.quit_button_position[0]+30,self.quit_button_position[1]))

pygame.draw.rect(self.surface,WHITE,[self.graph_1_button_position[0],self.graph_1_button_position[
1],180,40],1)
            self.surface.blit(self.graph_1_text ,
(self.graph_1_button_position[0]+30,self.graph_1_button_position[1]))

pygame.draw.rect(self.surface,WHITE,[self.graph_2_button_position[0],self.graph_2_button_position[
1],180,40],1)
            self.surface.blit(self.graph_2_text ,
(self.graph_2_button_position[0]+30,self.graph_2_button_position[1]))

pygame.draw.rect(self.surface,WHITE,[self.graph_3_button_position[0],self.graph_3_button_position[
1],200,40],1)
            self.surface.blit(self.graph_3_text ,
(self.graph_3_button_position[0]+30,self.graph_3_button_position[1]))
            def quit_graphing(self):
            plt.close(self.fig)
            Simulation_Screen()
Program()
```

Testing:

Testing Strategy:

In order to test my program and show that the functionalities all work as desired in the final version, I will carry out both Black and White box testing for each of the different screens of my program.

Testing the Welcome Screen:

Testing the Sign Up Button:	
Purpose	To ensure the signup button works correctly.
Description of Test	I will hit the signup button.
Expected Result	I will be sent to the signup screen.
Actual Result	I am sent to the signup screen.

Testing the Login Button:	
Purpose	To ensure the login button works correctly.
Description of Test	I will hit the login button
Expected Result	I will be sent to the login screen.
Actual Result	I am sent to the login screen.

Testing the Quit Button:	
Purpose	To ensure the quit button works correctly.
Description of Test	I will hit the quit button.
Expected Result	The window will close and the program will end.
Actual Result	The window closes and the program ends

Additional Video Evidence:

https://www.youtube.com/watch?v=X9kyMruoYhM

Testing the Sign-Up Screen:

Testing the Sign Up Button:	
Purpose	To ensure the signup button works correctly.
Description of Test	I will hit the signup button.
Expected Result	I will be sent to the Home screen of the user I made.
Actual Result	I am sent to the Home screen of the user I made.

Testing the Data Entry Validation:	
Purpose	To ensure that only valid data is added to the database.
Description of Test	When I enter the new user's data, the test data below will be the input.
Test Data	A. A username that is not already used and a password that is correctly repeated. (The name may be blank). B. A username that is already used and a password that is correctly repeated. C. A username that is not already used and a password that is incorrectly repeated.
Expected Result	A. The user will be sent to the home screen when the signup button is pressed. B. The user will be returned to a blank signup screen. C. The user will be returned to a blank signup screen.
Actual Result	A. The user is sent to the home screen when the signup button is pressed. B. The user is returned to a blank signup screen. C. The user is returned to a blank signup screen.

Testing the Data Entry:	
Purpose	To ensure the data is correctly added to the database.
Description of Test	I will print all the users in the simulation before and after the user signs up
Expected Result	There should be a new user with the correct details in the returned list.
Actual Result	The window closes and the program ends

Additional Video Evidence:

https://www.youtube.com/watch?v=90vXr1SdEnk

Testing the Login Screen:

Testing the Login Button:	
Purpose	To ensure the login button works correctly.
Description of Test	I will hit the login button.
Expected Result	I will be sent to the Home screen of the user I made.
Actual Result	I am sent to the Home screen of the user I made.

Testing the Data Entry Validation:	
Purpose	To ensure that only valid data is added to the database.
Description of Test	When I enter the new user’s data, the test data below will be the input.
Test Data	A. The username and password for a user in my database. B. A username of a user in my database with an incorrect password. C. A correct password for a user in my database and an incorrect username.

Expected Result	A. The user will be sent to the home screen when the login button is pressed. B. The user will be returned to a blank login screen. C. The user will be returned to a blank login screen.
Actual Result	A. The user is sent to the home screen when the login button is pressed. B. The user is returned to a blank login screen. C. The user is returned to a blank login screen.

Testing the Data Entry:	
Purpose	To ensure the correct user is selected from the database.
Description of Test	I will sign in as a user in the database.
Expected Result	The name of the user should be correctly displayed at the top.
Actual Result	The name of the user is correctly displayed at the top.

Additional Video Evidence:

https://www.youtube.com/watch?v=JMmjM6KEnPw

Testing the Home Screen:

Testing the New Simulation Button:	
Purpose	To ensure the new simulation button works correctly.
Description of Test	I will hit the new simulation button.
Expected Result	I will be sent to the new simulation screen.
Actual Result	I am sent to the new simulation screen.

Testing the Saved Simulations Button:	
Purpose	To ensure that the saved simulations button works correctly.
Description of Test	I will hit the saved simulations button
Expected Result	I will be sent to the saved simulations screen
Actual Result	I am sent to the saved simulations screen

Testing the Logout Button:	
Purpose	To ensure the quit button works correctly..
Description of Test	I will hit the logout button
Expected Result	I will be returned to the welcome screen.
Actual Result	I am returned to the welcome screen.

Additional Video Evidence:

https://www.youtube.com/watch?v=TyIlo3prd1s

Testing the Simulation Type Selection Screen:

Testing the Simulation Type Selector:	
Purpose	To ensure the selector works correctly.
Description of Test	I will hit the new simulation button on both custom and randomized.
Expected Result	For the custom I will be taken to the custom detail entry screen. For the randomized simulation I will be taken to the simulation screen.
Actual Result	For the custom detail entry I am sent to the custom detail entry screen. For the randomized simulation I am sent to the simulation screen.

Testing the Back Button:	
Purpose	To ensure that the back button works correctly.
Description of Test	I will hit the back button.
Expected Result	I will be sent to the home screen
Actual Result	I am sent to the home screen

Additional Video Evidence:

https://youtu.be/aRRUxh339Vk

Testing the Custom Data Entry Screen:

Testing the Data Entry Validation:	
Purpose	To ensure that only valid data is added to the database.
Description of Test	When I enter the new simulation's data, the test data below will be the input.
Test Data	A. A valid set of data. B. A valid set of data other than the length of turn which is zero. C. A valid set of data other than the length of turn which is negative. D. A valid set of data other than the starting number of organisms which is zero. E. A valid set of data other than the starting number of organisms which is negative. F. A valid set of data other than the starting number of species which is zero. G. A valid set of data other than the starting number of species which is negative. H. A valid set of data other than the starting number of species which is larger than the number of organisms. I. A valid set of data other than the map size which is zero.

	J. A valid set of data other than the map size which is negative. K. A valid set of data other than the number of food sources which is negative. L. A valid set of data other than the meat to veg ratio which is of form string:integer. M. A valid set of data other than the meat to veg ratio which is of form integer:string. N. A valid set of data other than the meat to veg ratio which is 0:0. O. A valid set of data where meat to veg ratio which is integer:0. P. A valid set of data where meat to veg ratio which is 0:integer Q. A valid set of data other than the number of water sources which is negative. R. A valid set of data other than that the tiles to be made with items on them is larger than the number of tiles in the map.
Expected Result	A. I will be sent to the simulation created with the given data. B. I will be sent to a blank custom detail entry screen. C. I will be sent to a blank custom detail entry screen. D. I will be sent to a blank custom detail entry screen. E. I will be sent to a blank custom detail entry screen. F. I will be sent to a blank custom detail entry screen. G. I will be sent to a blank custom detail entry screen. H. I will be sent to a blank custom detail entry screen. I. I will be sent to a blank custom detail entry screen. J. I will be sent to a blank custom detail entry screen. K. I will be sent to a blank custom detail entry screen. L. I will be sent to a blank custom detail entry screen. M. I will be sent to a blank custom detail entry screen. N. I will be sent to a blank custom detail entry screen. O. I will be sent to the simulation created with the given data. P. I will be sent to the simulation created with the given data. Q. I will be sent to a blank custom detail entry screen. R. I will be sent to a blank custom detail entry screen.
Actual Result	A. I am sent to the simulation created with the given data. B. I am sent to a blank custom detail entry screen. C. I am sent to a blank custom detail entry screen. D. I am sent to a blank custom detail entry screen. E. I am sent to a blank custom detail entry screen.

	F. I am sent to a blank custom detail entry screen. G. I am sent to a blank custom detail entry screen. H. I am sent to a blank custom detail entry screen. I. I am sent to a blank custom detail entry screen. J. I am sent to a blank custom detail entry screen. K. I am sent to a blank custom detail entry screen. L. I am sent to a blank custom detail entry screen. M. I am sent to a blank custom detail entry screen. N. I am sent to a blank custom detail entry screen. O. I am sent to the simulation created with the given data. P. I am sent to the simulation created with the given data. Q. I am sent to a blank custom detail entry screen. R. I am sent to a blank custom detail entry screen.

Testing the Generate button:	
Purpose	To ensure the selector works correctly.
Description of Test	I will hit the new simulation button on both custom and randomized.
Expected Result	For the custom I will be taken to the custom detail entry screen. For the randomized simulation I will be taken to the simulation screen.
Actual Result	I am sent to the simulation screen for the simulation I created

Testing the Back Button:	
Purpose	To ensure that the back button works correctly.
Description of Test	I will hit the back button.
Expected Result	I will be sent to the home screen
Actual Result	I am sent to the home screen

Additional Video Evidence:

https://www.youtube.com/watch?v=aRRUxh339Vk

Testing the Saved Simulations Screen:

Testing that Simulations Displayed:	
Purpose	To ensure only the three latest used simulations are shown on the screen.
Description of Test	I will create three simulations. I will then create a fourth.
Expected Result	The first simulation I created should not be displayed on the screen.
Actual Result	The first simulation I created is not displayed on the screen.

Testing the Simulation Displayed order:	
Purpose	To ensure that the simulations are displayed in the correct order.
Description of Test	I will click on the third simulation.
Expected Result	When I return to the saved simulations screen the previously third simulation should be displayed at the top. The other two should both have moved down.
Actual Result	When I return to the saved simulations screen the previously third simulation is displayed at the top. The other two have both moved down.

Testing the Simulation linked to each button:	
Purpose	To ensure that the correct simulation is loaded by its respective button.
Description of Test	A. I will click on the top simulation. B. I will click on the middle simulation. C. I will click on the bottom simulation. D. I will create a new simulation and then go to saved simulations and click on the top one.
Expected Result	A. The top simulation is displayed. B. The middle simulation is displayed.

	C. The bottom simulation is displayed. D. The new top simulation is displayed.
Actual Result	A. The top simulation is displayed. B. The middle simulation is displayed. C. The bottom simulation is displayed. D. The new top simulation is displayed.

Additional Video Evidence:

https://www.youtube.com/watch?v=HUTO6XdGPMg

Testing the Randomised Generation:

Testing the Simulation Created is Random.	
Purpose	To check that the randomised simulation has a name of the date it was created and random simulation details.
Description of Test	I will first clear the database. I will hit the generate the new randomised simulation. I will then create a second simulation.
Expected Result	I should be sent to the simulation screen with a new unique (not a standard) simulation. The name in saved simulations should be today's date (29/03/2021). The second simulation should be different to the first (not the name).
Actual Result	I am sent to a simulation screen with a new unique simulation. When I go to the simulations_screen

Additional Video Evidence:

https://www.youtube.com/watch?v=iey4XBV2pBQ

Testing the Simulation Screen:

Testing the simulation runs well and that the organisms evolve	
Purpose	To ensure there are no bugs/crashes

Description of Test	I will run a simulation and show how over time the organisms will change.
Expected Result	To see how certain organisms that are better adapted to the environment they are in will reproduce more and become a more successful species.
Actual Result	Shown in video

Video Evidence:

https://www.youtube.com/watch?v=PZBhb9dZVyo

Testing the Graph Screen:

Testing the Graph Buttons.	
Purpose	To check that the correct graph is being displayed when the buttons are clicked.
Description of Test	A. I will click the Number button B. I will click the Age button C. I will click the Variance button
Expected Result	A. I should see the number vs time graph displayed on the screen B. I should see the age vs time graph displayed on the screen C. I should see the variance vs time graph displayed on the screen
Actual Result	A. I see the number vs time graph displayed on the screen B. I see the age vs time graph displayed on the screen C. I see the variance vs time graph displayed on the screen

Testing the Number vs Time graph.	
Purpose	To check that the data being displayed by the number vs time graph is correct according to the simulation

Description of Test	A. I will run the simulation. When I see an organism being born, I will wait for the turn to end and then hit the analysis button. B. I will run the simulation. When I see an organism die, I will wait for the turn to end and then hit the analysis button.
Expected Result	A. I should see one of the lines for the species increase by one. B. I should see one of the lines for the species decrease by one.
Actual Result	A. I see one of the lines for the species increase by one. B. I see one of the lines for the species decrease by one.

Testing the Average Age vs Time graph.	
Purpose	To check that the data being displayed by the average age vs time graph is correct according to the simulation
Description of Test	A. I will run the simulation. When I see an organism being born, I will wait for the turn to end and then hit the analysis button. I will then click on the age button to display the average age time graph B. I will run the simulation. If no organism is born or dies, I will wait for the turn to end and then hit the analysis button. I will then click on the age button to display the average age time graph C. I will run the simulation. When I see an organism die, I will wait for the turn to end and then hit the analysis button. I will then click on the age button to display the average age time graph
Expected Result	A. I should see one of the lines for the species decrease since there is a new organism that has age 0 so average should be lower. B. The age of the organisms should increase as no organisms have died so they have all aged. C. I should see one of the lines for the species decrease since there is a new organism that has age 0 so average should be lower.

Actual Result	A. I see one of the lines for the species decrease since there is a new organism that has age 0 so average should be lower. B. The age of the organisms increases as no organisms have died so they have all aged. C. I see one of the lines for the species decrease since there is a new organism that has age 0 so average should be lower.

Testing the Average Variance vs Time graph.	
Purpose	To check that the data being displayed by the average variance vs time graph is correct according to the simulation
Description of Test	I will run the simulation, when a new organism is born, I will wait till the end of the turn and then I will hit analysis. I will then click on the Variance button and look at the Variance time graph.
Expected Result	The variance from the starting organism should have changed up or down according to the chromosome of the new organism
Actual Result	The variance line has changed value

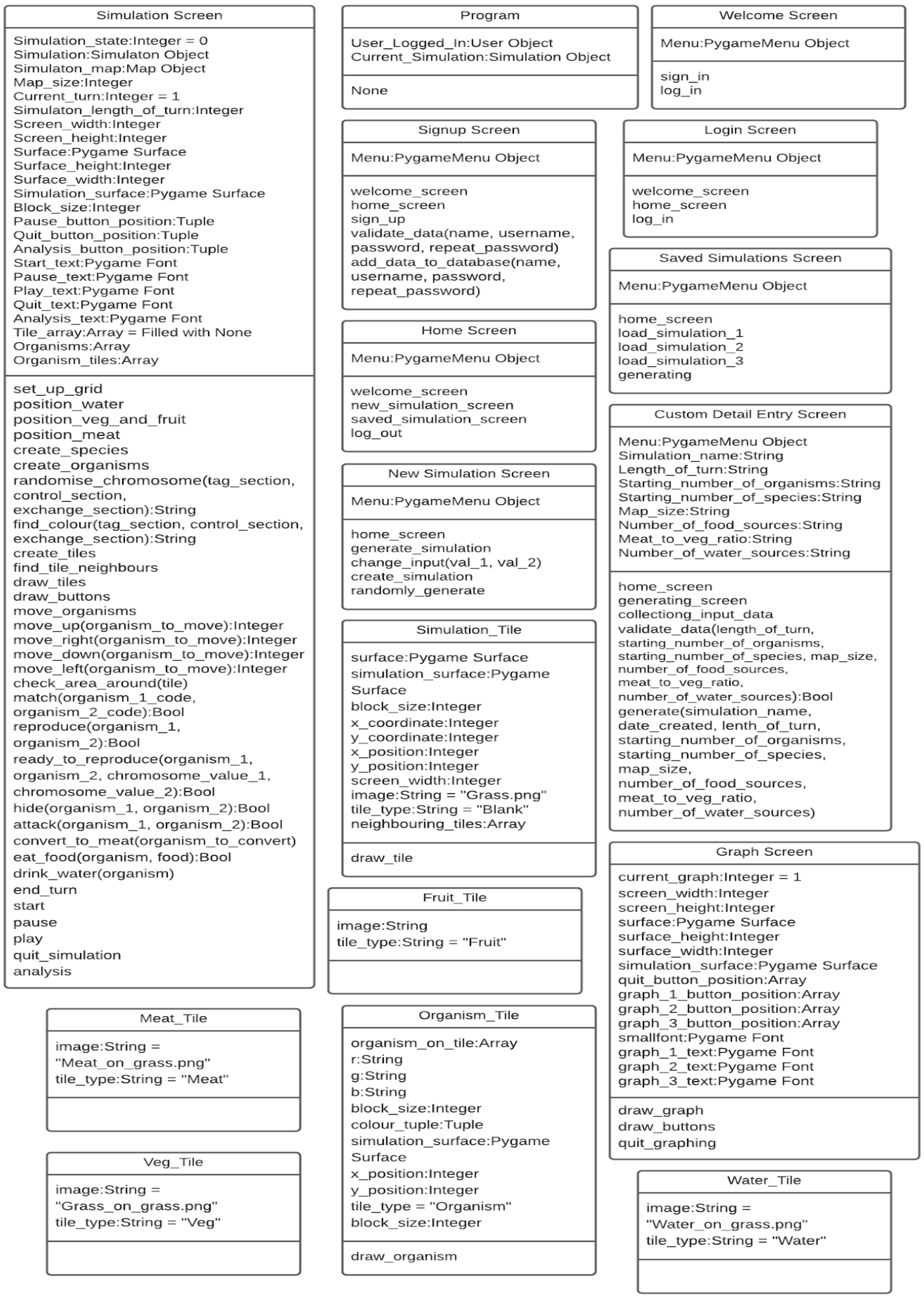
Additional Video Evidence:

https://www.youtube.com/watch?v=JuNuLxEc9yM

Evaluation:

UML of final version:

Below are the UML classes for my program with the attributes and methods:



Simulation Screen

Simulation_state:Integer = 0
Simulation:Simulaton Object
Simulaton_map:Map Object
Map_size:Integer
Current_turn:Integer = 1
Simulaton_length_of_turn:Integer
Screen_width:Integer
Screen_height:Integer
Surface:Pygame Surface
Surface_height:Integer
Surface_width:Integer
Simulation_surface:Pygame Surface
Block_size:Integer
Pause_button_position:Tuple
Quit_button_position:Tuple
Analysis_button_position:Tuple
Start_text:Pygame Font
Pause_text:Pygame Font
Play_text:Pygame Font
Quit_text:Pygame Font
Analysis_text:Pygame Font
Tile_array:Array = Filled with None
Organisms:Array
Organism_tiles:Array

set_up_grid
position_water
position_veg_and_fruit
position_meat
create_species
create_organisms
randomise_chromosome(tag_section, control_section, exchange_section):String
find_colour(tag_section, control_section, exchange_section):String
create_tiles
find_tile_neighbours
draw_tiles
draw_buttons
move_organisms
move_up(organism_to_move):Integer
move_right(organism_to_move):Integer
move_down(organism_to_move):Integer
move_left(organism_to_move):Integer
check_area_around(tile)
match(organism_1_code, organism_2_code):Bool
reproduce(organism_1, organism_2):Bool
ready_to_reproduce(organism_1, organism_2, chromosome_value_1, chromosome_value_2):Bool
hide(organism_1, organism_2):Bool
attack(organism_1, organism_2):Bool
convert_to_meat(organism_to_convert)
eat_food(organism, food):Bool
drink_water(organism)
end_turn
start
pause
play
quit_simulation
analysis

Program

User_Logged_In:User Object
Current_Simulation:Simulation Object

None

Welcome Screen

Menu:PygameMenu Object

sign_in
log_in

Signup Screen

Menu:PygameMenu Object

welcome_screen
home_screen
sign_up
validate_data(name, username, password, repeat_password)
add_data_to_database(name, username, password, repeat_password)

Login Screen

Menu:PygameMenu Object

welcome_screen
home_screen
log_in

Saved Simulations Screen

Menu:PygameMenu Object

home_screen
load_simulation_1
load_simulation_2
load_simulation_3
generating

Home Screen

Menu:PygameMenu Object

welcome_screen
new_simulation_screen
saved_simulation_screen
log_out

Custom Detail Entry Screen

Menu:PygameMenu Object
Simulation_name:String
Length_of_turn:String
Starting_number_of_organisms:String
Starting_number_of_species:String
Map_size:String
Number_of_food_sources:String
Meat_to_veg_ratio:String
Number_of_water_sources:String

home_screen
generating_screen
collectiong_input_data
validate_data(length_of_turn, starting_number_of_organisms, starting_number_of_species, map_size, number_of_food_sources, meat_to_veg_ratio, number_of_water_sources):Bool
generate(simulation_name, date_created, lenth_of_turn, starting_number_of_organisms, starting_number_of_species, map_size, number_of_food_sources, meat_to_veg_ratio, number_of_water_sources)

New Simulation Screen

Menu:PygameMenu Object

home_screen
generate_simulation
change_input(val_1, val_2)
create_simulation
randomly_generate

Simulation_Tile

surface:Pygame Surface
simulation_surface:Pygame Surface
block_size:Integer
x_coordinate:Integer
y_coordinate:Integer
x_position:Integer
y_position:Integer
screen_width:Integer
image:String = "Grass.png"
tile_type:String = "Blank"
neighbouring_tiles:Array

draw_tile

Fruit_Tile

image:String
tile_type:String = "Fruit"

Graph Screen

current_graph:Integer = 1
screen_width:Integer
screen_height:Integer
surface:Pygame Surface
surface_height:Integer
surface_width:Integer
simulation_surface:Pygame Surface
quit_button_position:Array
graph_1_button_position:Array
graph_2_button_position:Array
graph_3_button_position:Array
smallfont:Pygame Font
graph_1_text:Pygame Font
graph_2_text:Pygame Font
graph_3_text:Pygame Font

draw_graph
draw_buttons
quit_graphing

Meat_Tile

image:String = "Meat_on_grass.png"
tile_type:String = "Meat"

Organism_Tile

organism_on_tile:Array
r:String
g:String
b:String
block_size:Integer
colour_tuple:Tuple
simulation_surface:Pygame Surface
x_position:Integer
y_position:Integer
tile_type = "Organism"
block_size:Integer

draw_organism

Veg_Tile

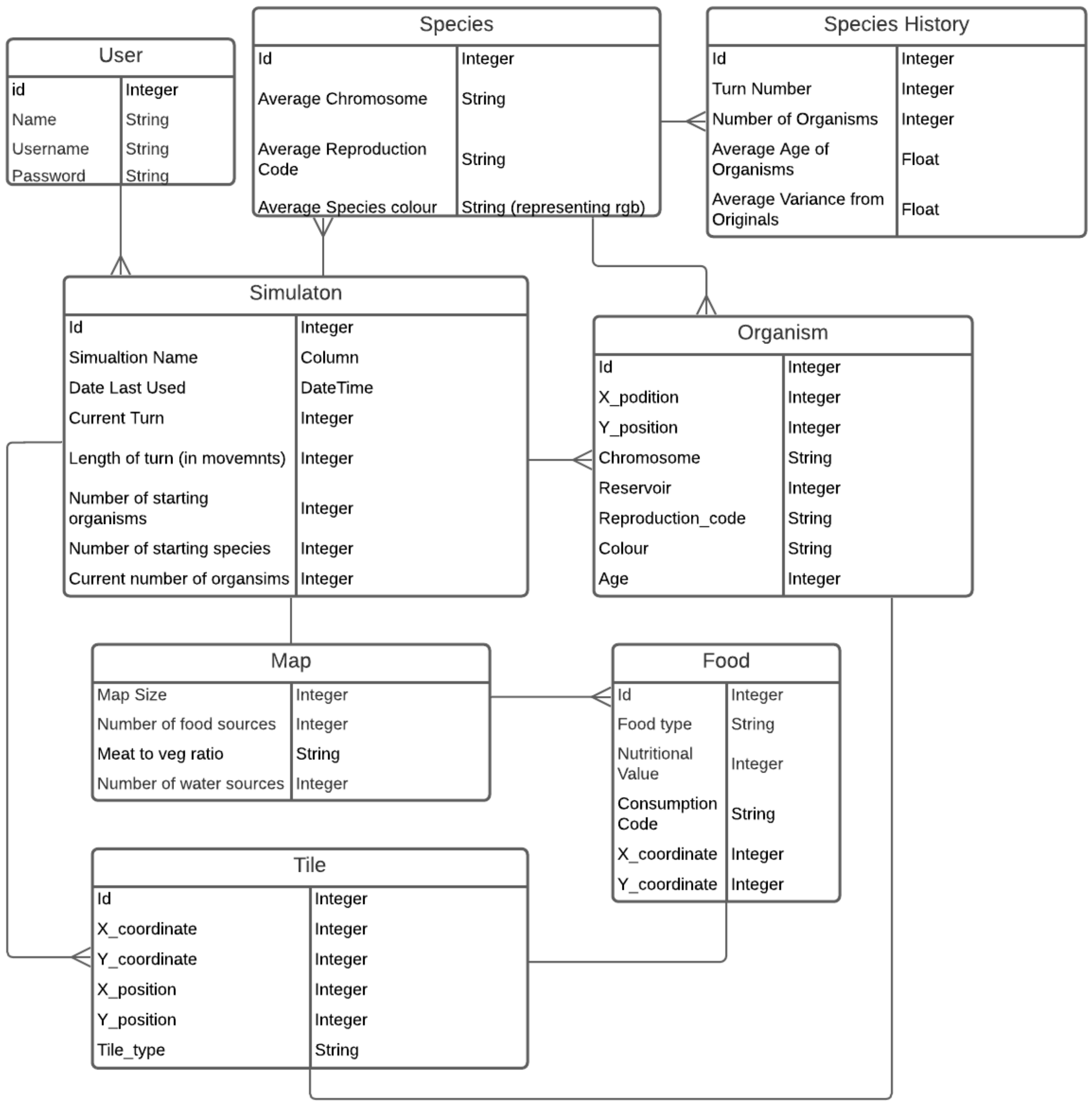
image:String = "Grass_on_grass.png"
tile_type:String = "Veg"

Water_Tile

image:String = "Water_on_grass.png"
tile_type:String = "Water"

Database Structure of final version:

Below is the structure of my final database:



Performance Vs Objectives:

No.	Objective	Performance Criteria	Evaluation
1	Have multiple different environments to show how through evolution the creatures will tend to specialise for different environments.	The user must be able to see the difference between the multiple areas and my program must be able to show how the different areas have creatures specialised to them.	--COMPLETELY ACHIEVED-- The user can create different environments via the custom detail entry screen. They can select the size of the map, the number of food and water and the number of organisms and species. These will all be displayed differently on the screen so the user can see that they are different. The user is also able to view the different specialisations of organisms as they are colour coded according to their chromosome, so similar colours have similar specialisations.
2	Creatures will have to collect food and water to survive.	Creatures must move around and try to collect the things that they require to survive.	--COMPLETELY ACHIEVED-- The organisms will die if their reservoir is empty.
3	For creatures to reproduce they must find a mate (a compatible creature).	Creatures will not be able to pass on their “genes” and reproduce unless they have contact with a fellow member of their species.	--COMPLETELY ACHIEVED-- Creatures are able to reproduce and create new organisms. The organism they create will have chromosome components from each of the two parents.
	Users will be able to have an account.	Users must be able to sign up and log in.	--COMPLETELY ACHIEVED--

4			Users have accounts which they can create and delete. They are able to sign in and look at their saved simulations or create new simulations which only they will be able to view.
5	Allow users to pause and save one of their simulations and to continue them at will.	Users must be able to reload their most recently saved simulation.	--COMPLETELY ACHIEVED-- Users are able to pause and play their simulations. The simulations are automatically saved as they go so there is no need for saving. Users are able to load previous simulations from the saved simulation screen. They can then continue and return to their home screen at will.
6	2D graphics to allow users to view the simulation.	The simulations must show how the creatures move around and behave.	--COMPLETELY ACHIEVED-- The simulations are displayed on the simulation screen. The organisms are represented using coloured circles. The organisms will move about the screen for the user to see. Food tiles will also appear, disappear or reset when they must and the user is able to view these changes on the simulation screen.
	Graphing of data to see how the simulations attributes change over time (e.g. a family tree).	The program will create graphs to show the relationship between organisms and their	--COMPLETELY ACHIEVED-- The graphs created for the user are easy to read and display useful information. They display

7		properties over time. This must be easy to use.	the information for all the different species in the simulation for which the graphs are created. They are particularly good at showing the ways in which the subtleties of evolution work such as a higher deviation from the original chromosome with subsequent generations.

Evaluation Interview:

For my evaluation, I decided to re-interview Miss Cuss, who I originally interviewed at the start of my project. I allowed her to have a play around with the program and then asked the following questions:

- **Does the program I have created fulfill your expectations?**

- Absolutely, the simulation has a great mixture of a fun interactive design and some really interesting detail. This would be especially for the older years as the analysis section can be used to identify the relationships between species and how they interact with one another.

- **Do you think that the simulation could be used in the classroom for teaching?**

- I think it could be really great as a fun laptop/ict lesson for the students. They could play around with it and try and see who gets the best simulation which highlights the relationships best.

- **What year groups do you think could benefit the most from the program?**

-This would probably be best for GCSE/A level students as the analysis would be quite complex for younger years. However they could always have a go at using it.

- **Do you think that the students would find it fun and interactive?**

- Definitely, I think that sort of simulation and interactive activities always interest students. I think they would really enjoy it.

- **If I had more development time are there any other extensions you would like added?**

- Maybe a more visual way of showing why certain organisms survived/ reproduced. That way it could add the extra level of learning to it. Maybe a sort of simulation log which outlines all the interactions between the organisms in the simulation.

- **Would you recommend this simulation?**

- Yes, I think it's great and that it could definitely be a fun way of teaching students about evolution. It was also very clear and easy to use. It was easy to navigate and very smooth.

Possible Extensions and Improvements:

If I had more time, I would likely implement the following extensions:

- I would give the organisms different speeds according to a new speed tag.
- I would implement the exchange conditions to allow the organisms to convert readily available chromosome components (a,b,c,d,#) to less readily available ones.
- I would likely try and speed up the rate at which the organisms move and interact with tiles around them to make the simulation more streamlined. In order to do this I could use the algorithm described below:

  In order to make my simulation more efficient in terms of time complexity, I would likely import all the data from the database at the start of every movement turn and store it in appropriate classes with the same format as the database. I would then create the relevant objects and add them to arrays. I could then quickly access these instead of the database.