YEAR: 2023

Saf Flatters

**Eden Simulation: Ecosystem Animation in Python**

Project Development Report

Contents

1. Introduction
2. User Guide

2.1 Dependencies

2.2 Command line Arguments

1. Background
2. Object Classes – Animals and Food
   1. Ants
   2. Butterflies
   3. Caterpillars
   4. Lizards
   5. Worms
   6. Flowers
   7. Fossils
3. Events
   1. Object Classes - Raining (Event)
   2. Initialisation and End
4. Traceability Matrix
5. UML Class diagram
6. Exception Handling
7. Code Techniques from COMP1005
8. References: Code Techniques using references ]
9. Conclusion and Future Work
10. **Introduction**

This project presents Eden, a Python-based simulation of a dynamic ecosystem, developed as a way to explore Python programming and coding fundamentals learned through independent research and short courses.

Built using Python3 and Matplotlib, Eden visualises a cross-sectional garden world populated with interactive entities such as ants, butterflies, caterpillars, lizards, worms, flowers, and fossils. The environment is divided into an above-ground and below-ground view, featuring elements like trees, plants, rocks, tunnels, clouds, and a sun.

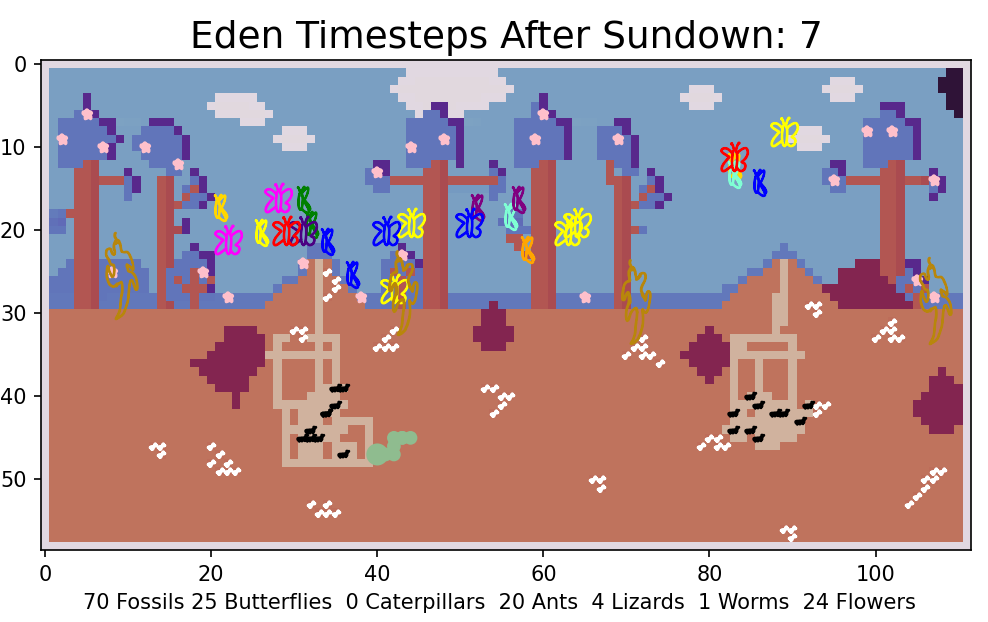
Users can customise their experience by selecting day or night mode and specifying the number of simulation timesteps. A rain event spanning 15 timesteps dynamically alters animal behaviour and environmental features, such as tunnel flooding and terrain restoration. Throughout the simulation, a real-time terminal log records animal lifecycles, food interactions, and ecosystem changes.

Eden integrates key programming concepts including object-oriented programming, user input and validation, command-line argument handling, exception handling, and animated visualization. Animal behaviours adapt to environmental triggers: ants dig and seek fossils, butterflies feed and evade predators, caterpillars undergo life cycle changes, and worms grow, die, and transform into fossils.

The project served both as a creative exercise and a technical foundation, consolidating core Python skills in a dynamic, visual, and interactive format.

A screenshot of a video game

Description automatically generated

****

1. **User Guide**

The files required to run the simulation include:

A screen shot of a computer program

Description automatically generatedREADME

Eden.py - Class file

playEden.py - main python script file (this is the file to quote to run simulation)

worldscene.csv - contains background

alive.csv - contains initial animals

Critters FOLDER - for plotting hand drawn animals and food

Requirements.txt

**2.1 Dependencies**

The dependencies are noted in requirements.txt

The svgpath dependencies are for the visualisation of some of the objects (drawn by myself and converted to SVG).

Download the dependencies on your virtual machine `pip install -r requirements.txt`

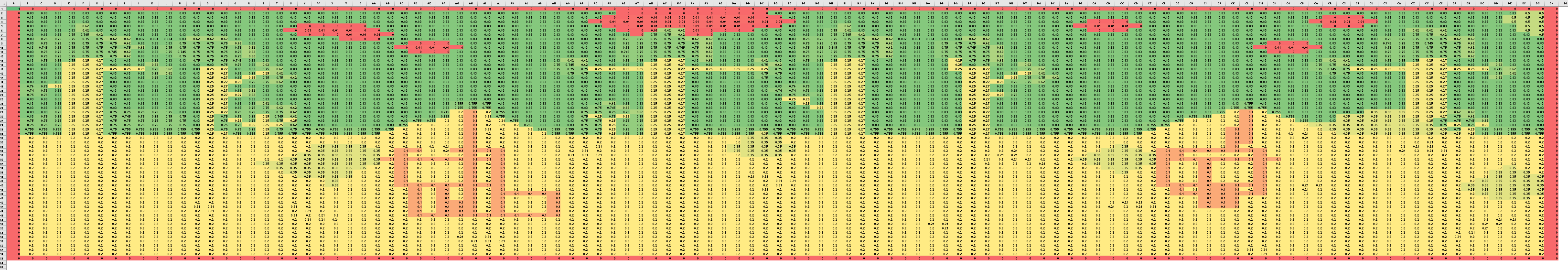
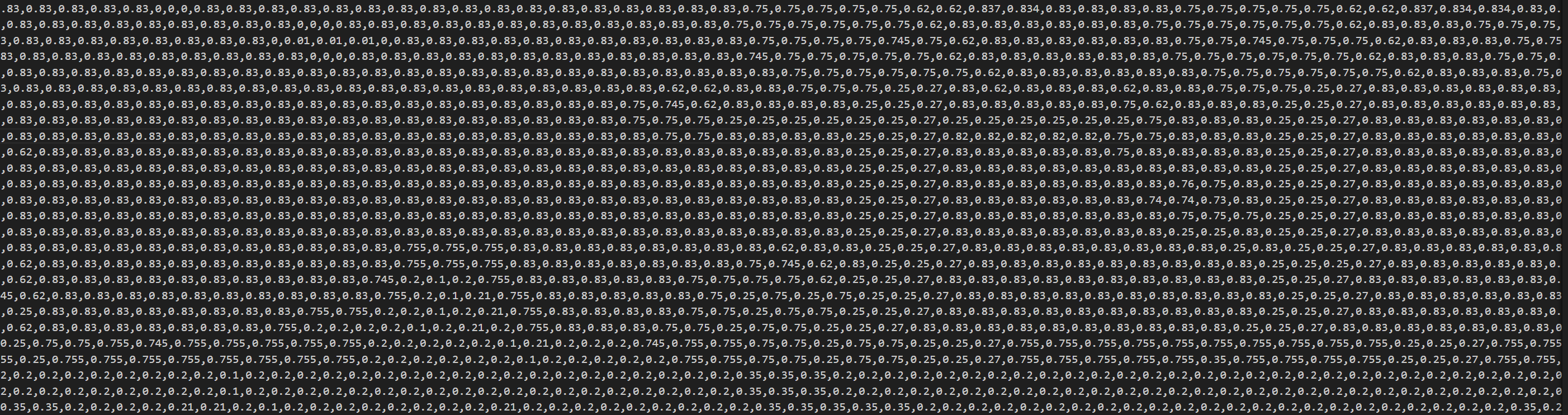
**2.2 Command Arguments:**

When initialised, the user is asked for the number of timesteps they would like to watch (3 timesteps per second) and whether they would like to watch it in Day mode or Night mode.

The number of timesteps and what mode (Day/Night) can be customised by command arguments. If there isn’t two command arguments or they are in the wrong format, it will revert to asking the user questions. (Questions are protected by Exception Handling)

For example; "python3 playEden.py 100 D" <----- 100 timesteps, Day Mode

1. **Background**

To create the background, I used lots of different environmental objects. I made it in excel, using Conditional Formatting colour scales which gave me red, yellow, green scaled cells to work with. I first created a colour chart and tested it in python to work out what colours the Reverse Terrain colour scheme would give me as the MatPlotLib website was limiting. I even made shadows on the trees and clouds away from the sun for visualisation. I also made it wide screen (rather than square or vertical) for gamer friendly screens.

Conditional Formatting in Excel for building alive.csv

VS Code screenshot (section) of alive.csv

A pixel art of a video game

Description automatically generated

Eden background colour legend:

0-0.01 = clouds or border

0.1 = tunnel (existing or object created)

0.2 = ground

0.21 = fossil loc (existing or object created)

0.25-0.27 = tree trunk or branch

0.35 = rock

0.5 = sun

0.62-0.75 = bush or leaves

0.7 = worm tail (object created)

0.745 = flower loc

0.755 = grass

0.83 = blue sky

Screenshot of Eden background (during the day) with no objects

1. **Object Classes – Animals and Food**

This section provides a detailed overview of the various objects and object classes within the Eden simulation. Each class is described, highlighting its characteristics, behaviours, and interactions.

Super classes are: Animal Class, Food Class and Event Class.

**A pixelated image of a group of sheep

Description automatically generated**

**4.1 Ants**

(Inheritance from Animal Class)

**Terrain:** Ants inhabit terrain with values ranging from 0.1 to 0.21 (underground or within tunnels). They are able to move through tunnels or dig new ones in the ground. Ants cannot move through worms (worms create a 0.7 terrain trail as they move). Ants move within Von Nuemann Neighbourhood movements.

**Food:** Ants actively seek out the object “fossils” for food when they are hungry, which is set as an interval from their previous meal. They also consume fossils they encounter even when not actively searching for them.

**Interactions:** Ants avoid worms. Until worm dies and becomes ant food (fossils). Fossils disappear once ants are on top of them (after they have seeked them out or stumbled upon them)

**Rain:** During rain events, ants move twice as fast. They crawl, dig and eat fossils twice their normal speed.

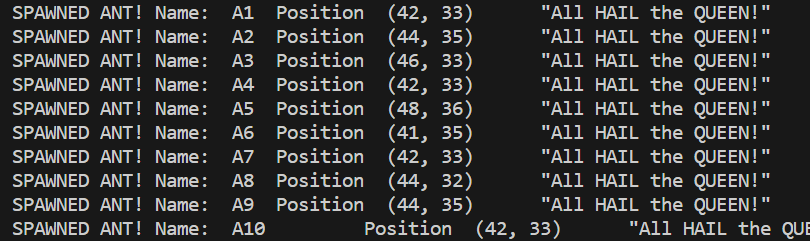
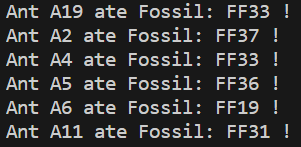
**Born/Die:** The number of ants in the environment remains the same from the beginning and are imported from the alive.csv file stating their name eg. “A1”, their position, their status and they say “All HAIL the QUEEN!”

**A black and grey caterpillar

Description automatically generatedPlot:** Ants are represented in the simulation with SVG drawings converted from drawings I drew on my ipad. See “Code Techniques from References” section for more detail. Zorder = 5 (larger the number the further to the top it is plotted) so worm would not be hindered when being plotted but fossils do not get plotted on top. The number of ants at a timestep is plotted on the x-axis title.

Ant drawn by me (PNG before being converted to SVG)

**Log:**

****

**Future work:** Future enhancements include the introduction of more ants during the simulation such as a birth process from a queen ant. I would like for ants to bring the queen ant fossils for her to eat. I would also like to code ants to be able to die. This could be during encounters with worms or from starvation when their "time\_since\_fossil" exceeds a certain threshold. I have set these up by having “status: alive” and would do the birth and death similar to how the butterflies are coded.

**A group of butterflies and a cross

Description automatically generated**

* 1. **Butterflies**

(Inheritance from Animal Class)

**Terrain:** Butterflies are capable of flying anywhere within the environment, excluding ground, rocks, and tunnels with terrain values exceeding 0.35. Butterflies move within Moore Neighbourhood movements.

**Food:** When hungry, they seek out flowers for nectar, once they have landed on a flower, they are not hungry for a set interval and fly randomly until they hungry again.

**Interactions:** Butterflies may become prey for lizards if they are within reach, either in front of the lizard's head or directly on top of the lizard. Caterpillars also turn into butterflies.

**Rain:** During the rain, butterflies hunger is turned off and they fly lower towards the ground to hide in the grass (Valid moves are only down (straight, left, right) or stay still). This increased the likelihood a lizard will eat them also.

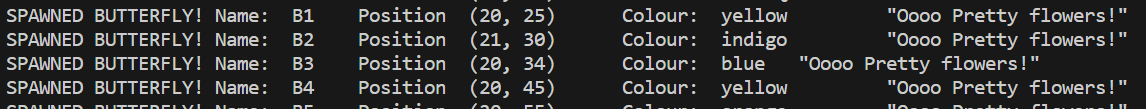
**Born/Die:** The number of butterflies in the environment at the beginning of the simulation is imported from the alive.csv file. This changes as lizards consume butterflies (the butterflies die). Butterflies are born from caterpillars. (see Caterpillar). When they are born they stating their name eg. “B1”, their position, their colour and “Ooooo Pretty flowers”.

**A drawing of a person holding a object

Description automatically generatedA drawing of a butterfly

Description automatically generatedPlot:** Butterflies are visually represented with two SVG images (drawn by me) and are plotted in two different ways to make it appear they are flapping their wings when flying. This is achieved by splitting the butterflies into two groups – those with an even first digit in their name and those with an odd first digit (using slicing). Then determining whether the timestep is odd and even, each group is plotted open or closed. See “Code Techniques from References” section for more detail about inputting SVG as markers. Zorder = 4 (so butterflies are plotted on top of flowers but under lizards). The number of butterflies at a timestep is plotted on the x-axis title.

Butterfly open and Butterfly closed drawn by me (PNG before being converted to SVG)

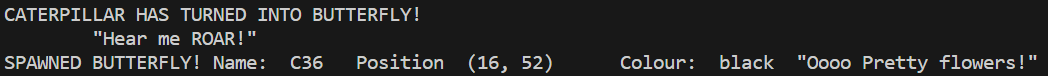
**Log: **

**A black background with white text

Description automatically generated**

**A black background with white text

Description automatically generated**

****

**Future work:** To include butterflies coupling to spark the introduction of caterpillars

A pixelated tree with a yellow oval

Description automatically generatedA pixelated tree with a red circle

Description automatically generated

* 1. **Caterpillars**

(Inheritance from Animal Class)

**Terrain**: Caterpillars can only move left and right and can not access the sky (0.8). This restricts them to one branch, leaves and tree trunk at a specific height.   
**Born/Die:** Caterpillars are born at a random location on one tree plotted in the simulation. They are born once the butterfly population in reduced from its original number of butterflies. Only one caterpillar is born at a time and takes 16 seconds for it to mature into a butterfly so generally the butterfly population will lower over time. Ther Caterpillar looks like a caterpillar for 9 seconds, then transforms into a cocoon which is plotted in the position the Caterpillar was in at metamorphosis. The cocoon then hangs there for 5 seconds before changing into a butterfly, which joins the butterfly class.

**Plot:** Caterpillars are visually represented as horizontal Ellipse patches with green edges and red fills. Cocoons are plotted as a vertical Ellipse patch - olive colour. The number of caterpillars at a timestep is plotted on the x-axis title.

**Log:**



A black background with white text

Description automatically generated

A black background with white text

Description automatically generated

**Future work**: Potential future developments include the introduction of multiple caterpillars, additional birth locations, caterpillars growing with age, the use of SVG images for caterpillar representation, and caterpillars actively seeking leaves as food.

**A brown line on a brown surface

Description automatically generated with medium confidence**

* 1. **Lizards**

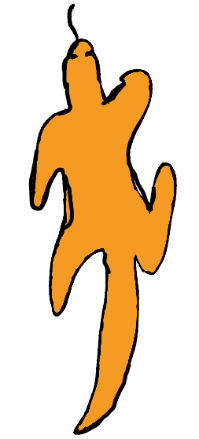
(Inheritance from Animal Class)

**Terrain:** Lizards have the ability to move through grass, climb trees, and navigate rocks within the environment (0.24 - 0.79). Lizards move within Von Nuemann Neighbourhood movements.

**Food:** Lizards eat butterflies when they are in reach (either above their head or on top of them)

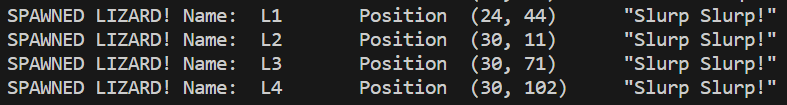
**Interactions:** Lizards eat butterflies

**Born/Die:** Only the initial number of lizards from alive.csv coexist in the simulation prevent a steep decline in the butterfly population. They do not die.

**Plot:** Lizards are visually represented using two SVG images (drawn by me), simulating their walking or climbing movements. This is achieved by splitting the lizards into two groups – those with an even first digit in their name and those with an odd first digit (using slicing). Then determining whether the timestep is odd and even, each group is plotted right leg infront or left leg infront. See “Code Techniques from References” section for more detail about inputting SVG as markers. Zorder = 7 (so lizards are in front of butterflies and flowers). The number of lizards at a timestep is plotted on the x-axis title.

Lizard right and Lizard left drawn by me (PNG before being converted to SVG)

**Log:**



A black background with white text

Description automatically generated

**Future work:** Future enhancements may include the introduction of lizard birth processes, deaths due to starvation, chasing food, changes in colour during rain events and sleeping on rocks.

* 1. A green and white circle

     Description automatically generated**Worms**

Inheritance from Animal Class)

**Terrain:** Worms can only move underground (ground and tunnels). < 0 and > 0.21. They can not move through rocks or through themselves (they change the terrain to 0.7 to make sure they can’t move through themselves, or ants move through them). Worms move within Von Neumann Neighbourhoods movement options.

**Food:** It is assumed the worm is eating dirt in the ground as it moves.

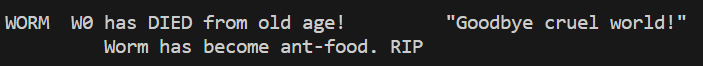
**Interactions:** While alive, the worm restricts ant movements. When dead, the worm becomes fossils for the ants to eat.

A pixelated image of a game

Description automatically generated**Born/Die:** Only one worm exists at a time. With each timestep, the worm grows longer. At a random length, the worm dies, leaving behind fossils in all the areas the worm tail was in. These fossils are eaten by ants. After dying, a new worm is born – in a random location underground to start growing longer again.

**Plot:** The worm is plotted as MatPlotLib “darkseagreen” circle patches that are added for every movement. The head is bigger than the tail. As the worm dies, there is one timestep where the fossils and the worms body are both shown to show that it’s being replaced with fossils. The number of worms in the environment is indicated on the x-axis title.

**Log:**

****

****

**Future work:** Worms actively hunting ants or trashing tunnels made by the ants. Coming to the surface during the rain. The use of SVG images for representing the worm.

* 1. A pixelated trees with pink flowers

     Description automatically generated**Flowers**

(Inheritance from Food Class)

**Terrain:** Flowers are positioned in the terrain everywhere that it is 0.745. These locations are situated in trees and on the grass.

**Food:** Flowers provide food for butterflies when the butterflies are hungry. They are permanent do not disappear like the fossils.

**Interactions:** With butterflies

A purple flower with yellow center

Description automatically generated**Plot:** Flowers are plotted using an SVG image that has been drawn by me. See “Code Techniques from References” section for more detail about inputting SVG as markers. Zorder = 1 (so butterflies and lizards are plotted in front of the flowers). The number of flowers at a timestep is plotted on the x-axis title.

**Log:**

A black background with white text

Description automatically generated

**Future work:** The amount of nectar in each flower to be tracked and they could die when they are used up. The could change colour depending on nectar levels.

**A white dots on a brown background

Description automatically generated**

* 1. A pixelated image of a game

     Description automatically generated**Fossils**

(Inheritance from Food Class)

**Terrain:** Fossils are positioned in the terrain everywhere that it is 0.21. These locations are situated underground.

**Food:** Fossils provide food for ants when the ants are hungry. Once eaten they are removed from the plot. Fossils are created by dying worms and are plotted everywhere the worm tail has been after it dies.

**Interactions:** With ants and worms

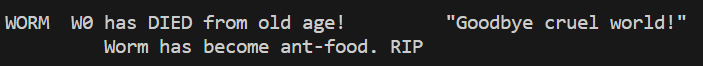
A grey bone with black outline

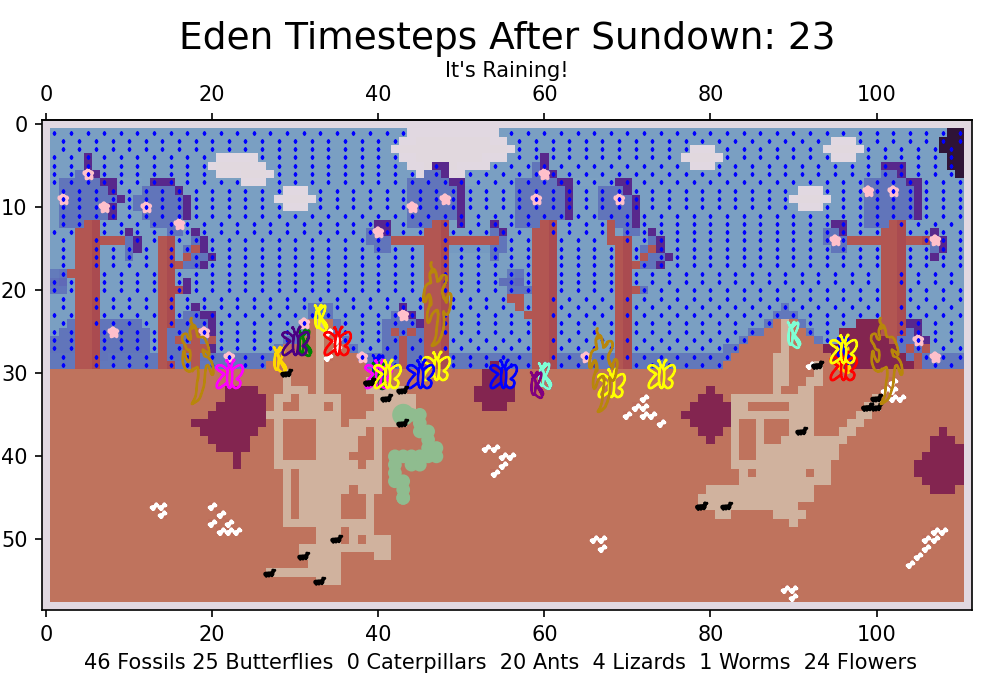
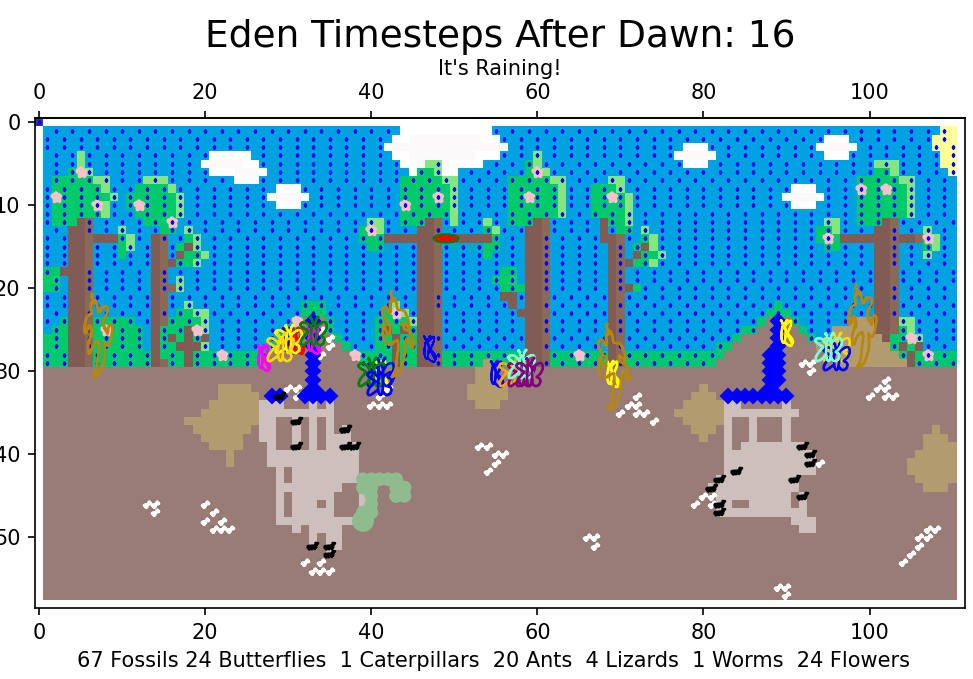
Description automatically generated**Plot:** Fossils are plotted using an SVG image that has been drawn by me. See “Code Techniques from References” section for more detail about inputting SVG as markers. The number of fossils at a timestep is plotted on the x-axis title.

**Log:**

**A black background with white text

Description automatically generated**

****

1. **Events – Rain Class, Initialisation and End**

**5.1 Rain**

(Inheritance from Event Class)

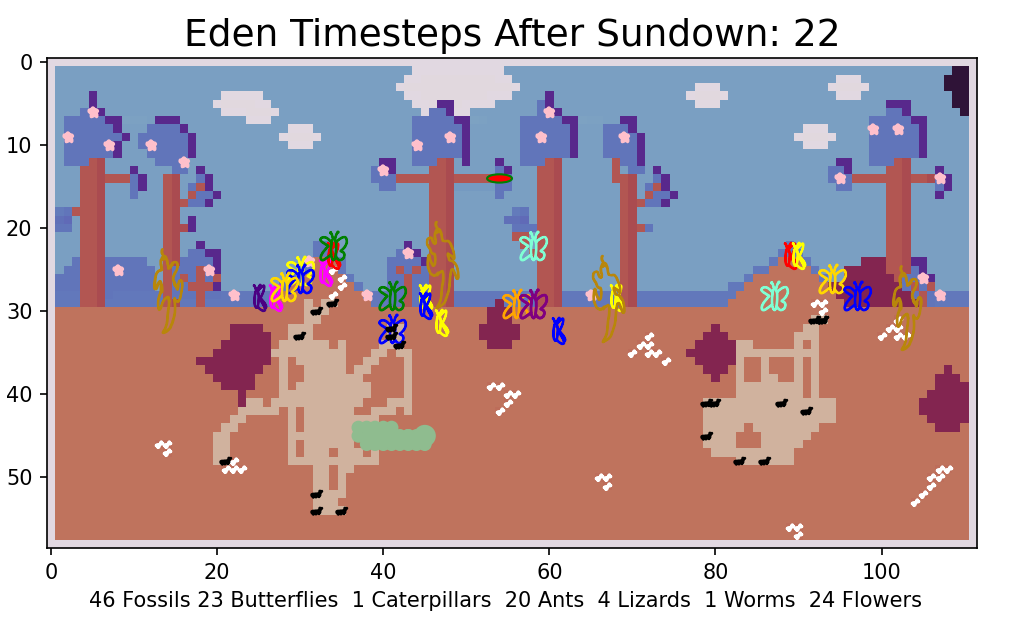
Rain is initiated for a set amount of timesteps (recommended set to 15) and begins at a specified timestep during the simulation. The rain event is a scatterplot due to computational limitations (I found plotting raindrops as individual objects to be resource-intensive).

When it rains, it triggers changes in behaviour in Ants and Butterflies (Ants move twice the speed and Butterflies seek the grass).

To simulate falling raindrops, the drops are plotted in two separate scatterplots, alternating at odd and even timesteps. A subtitle on the plot under the title “Eden Timesteps” says “It’s raining!”.

The log also prints “MA! THE RAINS ARE HERE!”.

The rain also triggers flooding in the tunnels which when the rain finishes, and the ground dries out, the tunnels are gone. As shown in the red circle below:



**5.2 Initialisation & End**

Before the simulation starts, if command line arguments haven’t been used, it asks the user “How may timesteps do you want to play? (3 timesteps per second): ”. This gives the user flexibility to choose how long they want to run the simulation for. Then it asks the user “Is it Day or Night?”. This gives the user flexibility to choose which colour map they want to run the simulation in. One appears as it’s day time and the other as it’s night.

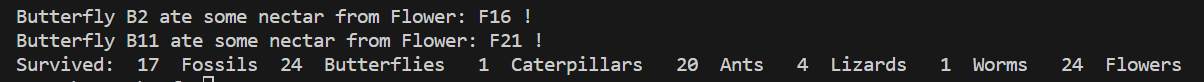
Both of these inputs are protected by ValueError Exception Handling.

A black background with white text

Description automatically generated

The log then prints all the animals from alive.csv and states how many animals and food to start.

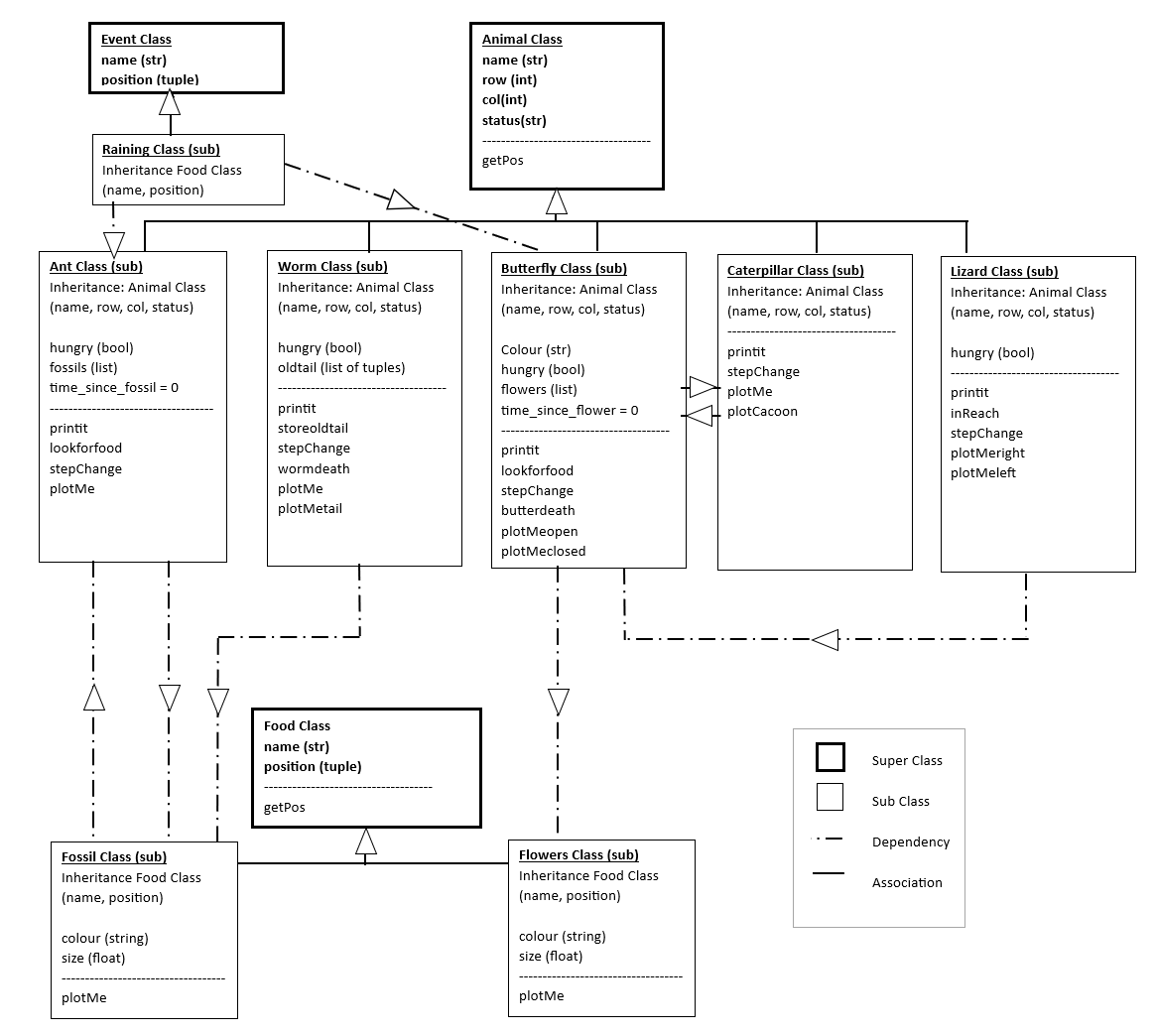
Once the timesteps chosen have been exhausted, the log prints the number of animals and food that survived the simulation.



1. **Traceability Matrix**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#Code ID (commented in code)** | **Feature** | **Action** | **Files associated** | **Tested (Green: OK, Red: Needs investigation)** |
| **1** | **Background Terrain & Initial Start up** | Read in backdrop worldscene.csv file that has floats, split by commas | playEden.py, worldscene.csv | Via Testing simulation |
| 1.1 | *Exception Handling* | FileNotFoundError  sys.exit(1) | playEden.py, worldscene.csv | Test incorrect files/input |
| 1.2 | Daytime | Sundial input chooses cmap plotted  Plt title “Dawn” | playEden.py | Via Testing simulation |
| 1.3 | Nighttime | Sundial input chooses cmap plotted  Plt title “Sundown” | playEden.py | Via Testing simulation |
| **2** | **Initial start up** | Ask user for number of timesteps and day/night | playEden.py | Via Testing simulation |
| 2.1 | *Exception Handling*  For both inputs | Value Error and upper() | playEden.py | Test incorrect files/input |
| **3** | **Object lists** | Make animal lists from alive.csv | playEden.py, alive.csv | Print to screen |
| 3.1 | *Exception Handling for alive.csv* | FileNotFoundError  sys.exit(1) | playEden.py, alive.csv | Test incorrect files/input |
| 3.2 | *Exception Handling*  *For all .svg* | FileNotFoundError  sys.exit(1) | playEden.py, all SVG files in critters | Test incorrect files/input |
| **4** | **Commandline Args** |  |  |  |
|  |  | No. of Timesteps | playEden.py | Via Testing simulation |
|  |  | Day or Night mode | playEden.py | Via Testing simulation |
|  |  | Rain Start timestep | playEden.py | Not enough time |
|  |  | Rain Stop timestep | playEden.py | Not enough time |
| **5** | **Animals** |  |  |  |
| 5.1 | Animal Class | Init and getPos | Eden.py | Via Testing simulation |
| 5.2 | Ants | Object lists | playEden.py, Eden.py | Print to screen |
|  |  | Object class - and inheritance from Animal class | Eden.py | Via Testing simulation |
|  |  | Behaviour functions: lookforfood, stepChange, | Eden.py | Via Testing simulation |
|  |  | dig tunnels - changing the terrain to 0.1 | playEden.py, worldscene.csv, Eden.py | Via Testing simulation |
|  |  | raindance - plotting twice per timestep | playEden.py, worldscene.csv, Eden.py | Via Testing simulation |
|  |  | Visualisation - SVG image | ant.svg, Eden.py, playEden.py | Via Testing simulation |
| 5.3 | Butterflies | Object lists | playEden.py, Eden.py | Print to screen |
|  |  | Object class - and inheritance from Animal class | Eden.py | Via Testing simulation |
|  |  | Behaviour functions: lookforfood, stepChange, | Eden.py | Via Testing simulation |
|  |  | butterdeath - checking if lizards are same position or inreach | playEden.py, worldscene.csv, Eden.py | Via Testing simulation |
|  |  | plot surviving butterflies and baby butterflies | playEden.py, | Print to screen |
|  |  | raindance - valid moves changed to +rows | Eden.py | Via Testing simulation |
|  |  | Visualisation - SVG images | openbutter.svg, closedbutter.svg Eden.py, playEden.py | Via Testing simulation |
| 5.4 | Caterpillars | Object lists | playEden.py, Eden.py | Print to screen |
|  |  | Object class - and inheritance from Animal class | Eden.py | Via Testing simulation |
|  |  | Behaviour functions: lookforfood, stepChange, | Eden.py | Via Testing simulation |
|  |  | lifecycle of caterpillar - cat to cocoon to baby butter | playEden.py, worldscene.csv, Eden.py | Via Testing simulation |
|  |  | Visualisation | Eden.py, playEden.py | Via Testing simulation |
| 5.4 | Lizards | Object lists | playEden.py, Eden.py | Print to screen |
|  |  | Object class - and inheritance from Animal class | Eden.py | Via Testing simulation |
|  |  | Behaviour functions: lookforfood, stepChange, | Eden.py | Via Testing simulation |
|  |  | inReach and getPos - checking if lizards are same position or inreach of butterflies | playEden.py, worldscene.csv, Eden.py | Via Testing simulation & Print to screen |
|  |  | Visualisation - SVG images | lizardleft.svg, lizardright.svg, Eden.py, playEden.py | Via Testing simulation |
| 5.5 | Worms | Object lists | playEden.py, Eden.py | Print to screen |
|  |  | Object class - and inheritance from Animal class | Eden.py | Via Testing simulation |
|  |  | Behaviour functions: lookforfood, stepChange, | Eden.py | Via Testing simulation |
|  |  | store old tail and changing terrain to 0.7 | playEden.py, worldscene.csv, Eden.py | Print to screen |
|  |  | wormdeath - changing the terrain to be fossil terrain 0.21 | playEden.py, worldscene.csv, Eden.py | Via Testing simulation and printing worldscene.csv |
|  |  | Visualisation | Eden.py, playEden.py | Via Testing simulation |
| **6** | **Food** |  |  |  |
| 6.1 | Flowers | Object lists using terrain that = 0.745 to make list | playEden.py, Eden.py | Print to screen |
|  |  | Object class - and inheritance from Food class | Eden.py | Via Testing simulation |
|  |  | Visualisation - SVG file | flower.svg, Eden.py, playEden.py | Via Testing simulation |
| 6.2 | Fossils | Object lists using terrain that = 0.21 to make list | playEden.py, Eden.py | Print to screen |
|  |  | Object class - and inheritance from Food class | Eden.py | Via Testing simulation |
|  |  | Visualisation - SVG file | fossil.svg, Eden.py, playEden.py | Via Testing simulation |
| **7** | **Event** |  |  |  |
| 7.1 | Raining | Create raindance bool affecting ants and butterflies | Eden.py, playEden.py | Via Testing simulation |
|  |  | Flooding and Rain Visualisation | Eden.py, playEden.py | Via Testing simulation |

1. **UML Class diagram**



1. **Exception Handling**

I used FileNotFoundError and ValueError exception handling in this project.

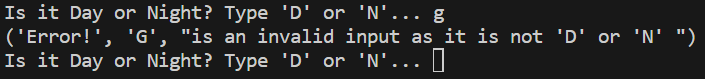
FileNotFoundError were used for importing alive.csv, worldscene.csv and svg files. These resulted in a sys.exit(1) to force the simulation to close. Here are the screenshots from testing incorrect files:

****

I used ValueError exception handling when asking the user for inputs regarding how many timesteps they want to view and whether they want day or night. I used looping so if the user put the wrong input in they could try again. Here are the screenshots from testing incorrect inputs:

**A screenshot of a computer screen

Description automatically generated**

****

1. **Code Techniques**

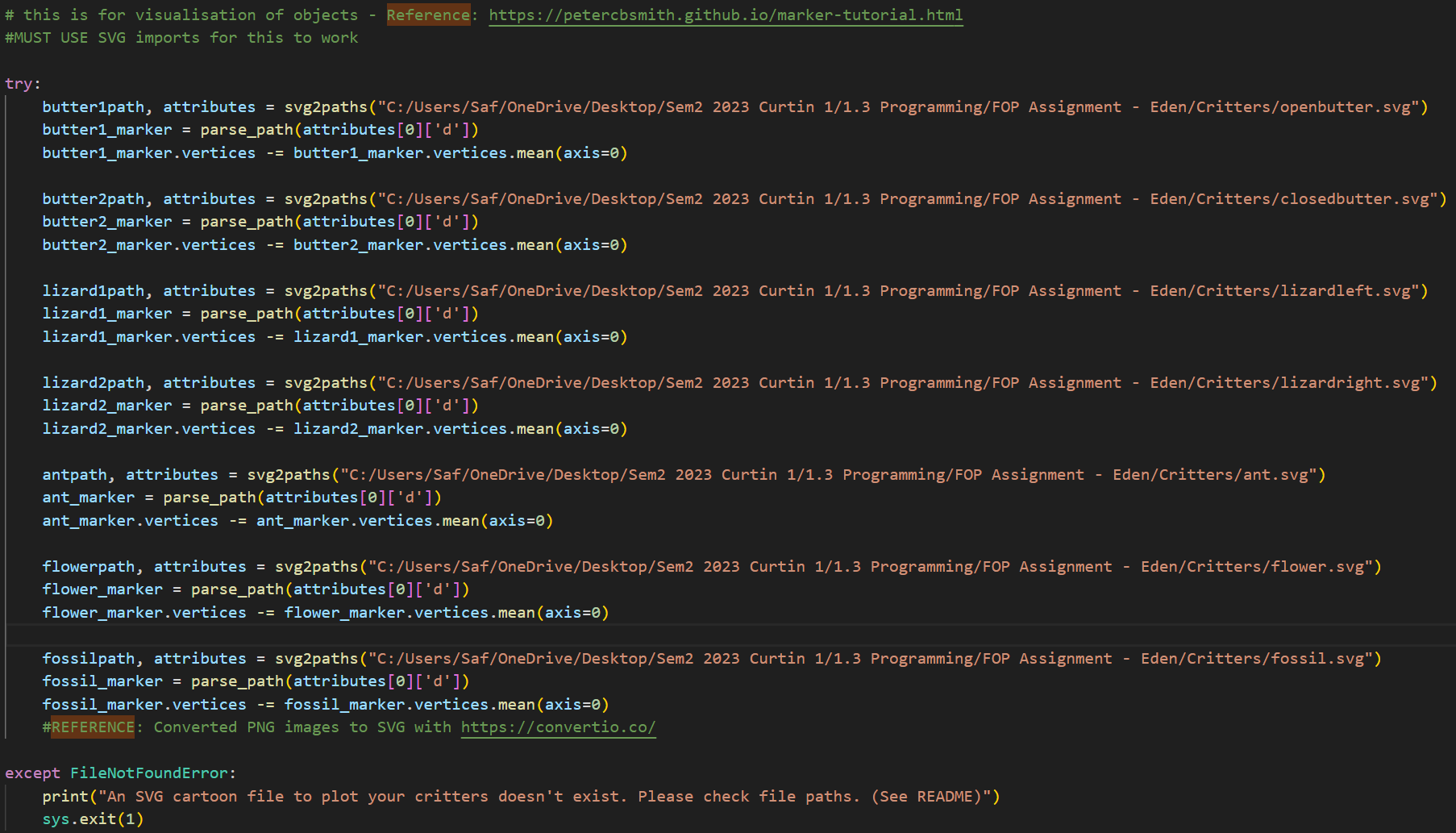
|  |  |
| --- | --- |
| Code Recommended Structure | ✔ |
| PEP-8 | ✔ |
|  |  |
| If/else/elif | ✔ |
| For loops & While loops | ✔ |
| Nested loops | ✔ |
| Integers and Floats and Strings | ✔ |
| Lists | ✔ |
| Multidimentials arrays (np.arrays) | ✔ |
| Looping through arrays | ✔ |
|  |  |
| Slicing and Indexing (start stop skip) (reverse) | ✔ |
| Arithmetic operations | ✔ |
| Comparison operations | ✔ |
|  |  |
| Asking the user for input | ✔ |
| .upper() | ✔ |
| Random number generation (randint and random.choice) | ✔ |
| Random choice, randint | ✔ |
|  |  |
| Element wise functions | ✔ |
| Array wise Functions | ✔ |
|  |  |
| Matricies in Numpy | ✔ |
| Colour Maps | ✔ |
| Different types of plots | ✔ |
| Plot titles including (strings and variables together) | ✔ |
|  |  |
| Class | ✔ |
| Class Inheritance | ✔ |
| Functions | ✔ |
| Printit definitions | ✔ |
| \_\_main\_\_ | ✔ |
|  |  |
| Opening and reading files | ✔ |
| Slicing arrays | ✔ |
| Importing Files | ✔ |
|  |  |
| Neighbourhood movements | ✔ |
|  |  |
| List comprehensions | ✔ |
|  |  |
| Exception Handling, ValueError, FileNotFound | ✔ |
| Command line arguments | ✔ |

1. **References: Code Techniques references**

Some notable code references:

To create SVG files from my drawings I required a converter: <https://convertio.co/>

I then used this blog: https://petercbsmith.github.io/marker-tutorial.html to learn how to import them into my code and plot them for the Ants, Butterflies, Lizards, Flowers, Fossils for visualisation purposes.



When plotting them I learnt from this blog: <https://saturncloud.io/blog/specifying-the-order-of-matplotlib-layers-a-guide/> about layers (zorder) to ensure I plotted my objects over each other in the order I wanted

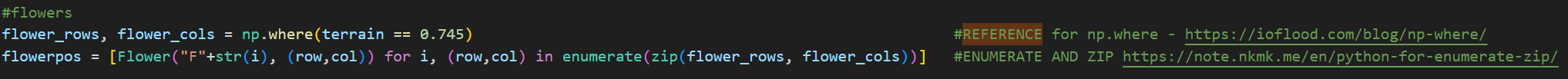


For plotting the caterpillars – I used: https://matplotlib.org/3.1.1/gallery/units/ellipse\_with\_units.html#sphx-glr-gallery-units-ellipse-with-units-py to learn about how to plot Ellipses (ovals)



For both placing the fossils, flowers and rain, I replaced my original nested loop code with np.where – NumPy function I found in the NumPy library that looks for specific conditions in an array. I learnt more about it and how to use it from <https://ioflood.com/blog/np-where>.

Rather than using a nested loop – I used the enumerate with the zip function. Zip allowed me to perform operations on both the row and the column lists. If I had more time – I would use it in the animal classes for determining Valid Moves. I used the blog <https://note.nkmk.me/en/python-for-enumerate-zip/> to work out how to put it all together for my purpose.



I wanted a better way to look for multiple terrain options when using np.where. We had learnt about the pipe operator in class being used as a union but not in this context. I found this blog https://ridwanray.medium.com/the-pipe-symbol-in-python-133239503fec that taught me what to do and why it was a good method.

****

Other techniques I used that had been expanded on my learnings:

MatPlotLib – specifically these pages:

https://matplotlib.org/stable/index.html

<https://matplotlib.org/stable/users/explain/colors/colormaps.html>

<https://matplotlib.org/stable/gallery/subplots_axes_and_figures/secondary_axis.html>

<https://matplotlib.org/stable/gallery/color/named_colors.html>

https://matplotlib.org/stable/api/markers\_api.html

Python documentation

<https://docs.python.org/3/>

NumPy documentation

<https://numpy.org/doc/1.26/>

1. **Conclusions & Future Work**

This project really helped me cement my understanding of Python and the coding techniques I picked up through short courses and my own research. Along the way, I learned a lot about debugging, reading documentation, and how to structure a bigger project properly. I also worked on writing more modular code so it would be easier to expand later.

I'm proud of how the simulation turned out — especially the visualisation side and the small creative touches like the funny quotes during animal lifecycle events. Even though I had more ambitious plans at the start, I’m happy with what I achieved in the time I had. I also feel confident that with more time, I could code everything I described in the future works section.

Going forward, I’d like to add more user interaction to make the simulation feel more like a game, and keep improving the code by using more list comprehensions, enumerate, zip, and better output handling like exporting the logs to text files. I'd also love to experiment with parameter sweeping and bigger environment changes in the future — now that I know I can build something like this, I’m excited to push my skills even further.