# Simulation Project Report

**Fundamentals Of Programming – COMP1005 2022 S2 FOP Assignment**

**Swamp of Life**

**Name : Malith Pramuditha**

**Curtin ID : 20926076**

## Overview

The goal of this project was to create a model of a swamp filled with ducks, shrimp, and newts to show how they interact with one another and how their behavior changes depending on the time of day. And how their movements would be carried out on a landscape that had been created to fluctuate in height. The goal of the program was to have distinct behaviors and characteristics for each duck, shrimp, and newt, which was accomplished by using object-oriented programming to create classes for each duck, shrimp, and newt and specifying attributes to implement these features specifically for each individual animal. And how the duck would be affected by landmarks like water, food, and so forth, The programming was also supposed to imitate shrimp movement and how other creatures influence animal movement. As a result, the software includes elements like water levels, and how the animals interact with one another and with one other. Since various choices would produce greatly diverse situations, a variety of parameters can be provided as command lines without limiting the simulation to a certain scenario. Using software packages like matplotlib.pyplot, random, and animation, pygame which were utilized to create the program, the behavior of swamp creatures was simulated based on the many criteria that were given.

## User Guide

Before running the code for the first time, a user should be aware of a number of things. To execute the function with waterlevel and duck, the input file main.py is used within the code. Python is used to provide a demonstration of the life cycle and movement of ducks. Additionally, newt.py was created to display the motions and life cycle of newts. Shrimp.py was used to simulate the lifecycle of shrimps, algae, and motions. PY was used to display immobile things. the main code is not required to be submitted by the user. Before running the code, the user must be aware of the command line inputs that may be made in order to receive different simulations based on the sets of inputs provided. If the user does not give any inputs the values for the parameters will be used, the parameters are ducks(noOfDucks), shrimps(noOfshrimp), newt(noOfnewt) simulation. which has to be a given as the number of animals otherwise the program would not be able to recognize the input as the code is made to take in otherwise. The program should be able to save the plot of the simulation though which was expected but in the program I wasn't able to implement that cause everytime I ran through it only saved the last timesteps' plot so that requires debugging.and I didnt figure out what was wrong with it. All in all a user should be able to simulate the program through different scenarios once these factors are known.

## Traceability matrix

| Feature | Requirements | Design | Test |
|---|---|---|---|
| 1)Movements | | | |
| 1.1)Create duck object | Duck objects must be initialized using default data | self.Attributes in duck.py | [PASSED] Create duck objects without error in duck.py |
| 1.2)Create shrimp object | Shrimp objects must be initialized using default data | self.Attributes in shrimp.py | [PASSED] Create shrimp objects without error in shrimp.py |
| 1.3)Create newt object | newt objects must be initialized using default data | self.Attributes in newt.py | [PASSED] Create newt objects without error in newt.py |
| 1.4) duck move | Duck must need to move | Self.speed in duck py line 09 –line 17 , main.py | [PASSED] duck is moving without error in main.py |

|  |  |  |  |
|---|---|---|---|
| 1.5)shrimp move | shrimp must need to move | Self.speed in shrimp.py line 9 – 17 in main.py | [PASSED] shrimp is moving without error in main.py |
| 1.6)newt move | Newt must need to move | Self.speed in newt.py line 9 - 18 | [PASSED] newt is moving without error in main.py |
| 2)Interaction |  |  |  |
| 2.1)Interaction when different animals meet | N/A | Not Implemented | N/A |
| 2.2)recognizing animals when near | N/A | Not Implemented | N/A |
| 3)Life cycle |  |  |  |
| 3.1) duck life cycle | Duck life cycle must need to activate | In duck.py line 9 - 63 | [PASSED] Duck life cycle activate without error in main.py |
| 3.2)shrimp life cycle | Shrimp life cycle must need to activate | In shrimp.py line 9 - 62 | shrimp life cycle activate without error in main.py |
| 3.3) newt life cycle | newt life cycle must need to activate | In newt.py line 9- 60 | newt life cycle activate without error in main.py |
| 4) non-moving objects | Algae must not need to move | Self.attributes in algae.py | Algae not moving without error in main.py |
|  |  |  |  |

## Introduction

The purpose of this project, as stated in the assignment specifications, was to simulate animal behavior in a swamp within a terrain or grid while taking into account various factors, such as how their movement makes a difference according to grid heights and how they would react with other animals when combinations of animals come into contact. In order to implement the simulation and produce an output that could be useful to monitor and analyze behavior on a real-life level, it was necessary to take into account these and a great number of other factors. This would have given us an idea of how the movements and life cycles of duck, shrimp, and newt differ due to various factors. There were six key parts we were expected to implement through the code in relation to the needed extensions listed in the assignment. The class files duck.py, newt.py, shrimp.py, and algae.py were used to call in the maincode (main.py) to create different animal objects by giving various attributes such as movements, life cycle, and a unique life cycle to make that three animals different from each other. The first section of this was object behavior and it is implemented through Object Orientation and class methods. While other properties were generated at random with relation to the class files, the locations of the animals were assigned at random using random, which was what required to be called through the class. The notion of providing each animal a distinct life cycle was put into practice in the class file. And within this file, the make self method was used to divide the animals into three groups according to their types. This was done using the swamlife.py example code, which was adopted as a model and changed to fit the application. We were expected to incorporate how an animal's lifespan altered every day in the extension. This functionality could not be used since I was unable to implement that in the code to the main code. When it comes to the second category of characteristics, movement, we had to allow the user to choose the amount of animals on which the movement and interaction of the animals is dependent. Pygame is used to achieve this, enabling the user to enter the Number of animals entered into the command line, as provided in the user manual's instructions on how to modify the command line parameters. Assigning a maximum number of animals that could travel from different places was how this element was accomplished. After importing the random function, it was used to create the actual animal movement, and PracTest3's code character was used to prevent the animals from straying outside the grid's bounds. The .py files' code was used and applied appropriately to build this functionality, and "yebbleLife base," one of the provided example scripts, served as inspiration for the random movement. The discussion of files and grids in Prac05 provided inspiration for creating the grid. The array that will eventually be used to depict the landscape is filled with the relevant values as the algorithm iterates line by line through the list. After that, we must put into practice how cats and other animals interact. Creatures must first be able to detect the presence of other animals in order to carry out the interaction.

All of the significant growth phases that occur over the course of a animal's lifespan are described by a life cycle. It covers the breeding of new species members. Life cycles usually go in a circle and end up back where they started, as their name indicates. Duck spawn, for instance, develops into duck eggs. They go through many stages as they develop into adults. Since all living things reproduce, they all have life cycles. These can come in a variety of shapes, but they generally have the same structure. The last step of an animal's life cycle—death—was supposed to be implemented in this section, but I was unable to do it. Algae, water, ducks, shrimp, and newts, which are essentially regarded to be waterlevel, were to be integrated inside this area. The distance gave each animal a unique watermark, whose coordinates were retrieved from a commonFunctions.py file in Python, which was then utilized to visualize the unique cells under consideration. This method was used to plot the places inside the code once the main.py routines had been utilized, changed, and adjusted to fit the code. The interaction with the positions was done inside the code under the presumption that if an animal changed positions, it would return to its original location and continue to move. We were asked to improve the visualization of the simulation in the last segment, which I accomplished by invoking the make def() function in the duck.py, newt.py, and shrimp.py programs creating the plot and placing it on the simulation's plot. And this makes it much clearer in the simulation to distinguish between the various animals. For the simulation to work properly when charting the animals in the grid, the rows and columns must be reversed, therefore this component was accomplished using the PracTest3s' code and one of the example outputs that were provided (PasciroExample.py). Monitoring and comprehension are much aided by this, and in this area we also had to show a log of occurrences or simulation data.

Another feature that we had to incorporate was the ability to store the simulation state as a plot image or a, which I was only able to achieve partially since I only saved the simulation state for the most recent timestep, as opposed to storing each plot image throughout each iteration. Overall, the software meets the majority of the criteria listed in the section, and according to the user guide, the user may simulate a variety of situations by entering different combinations of parameters into the program's customized command line, which is based on Prac07.

## Methodology

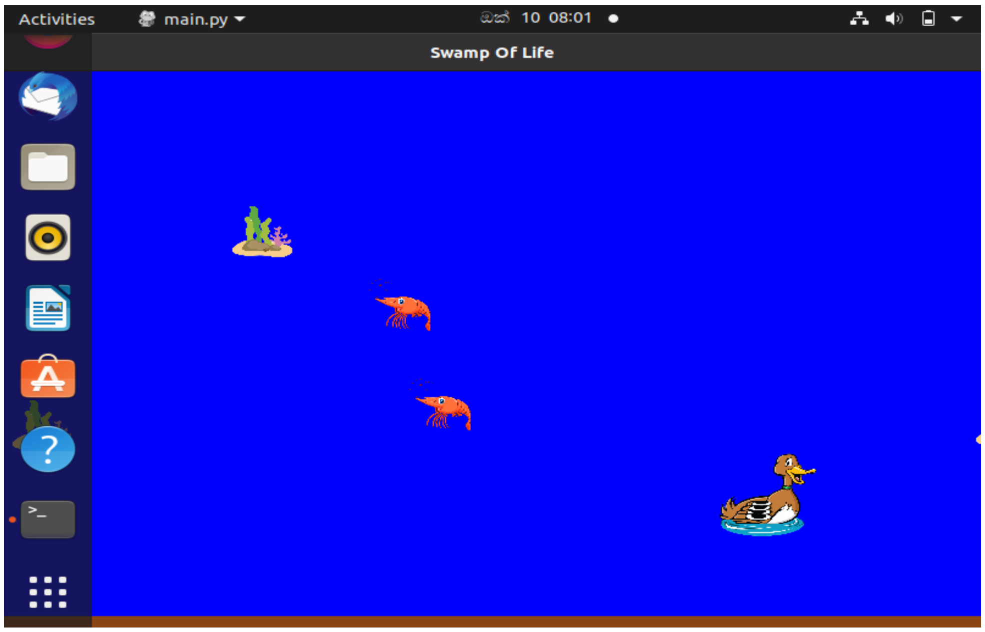
There will be a few minor differences between the three simulation-created scenarios when they are repeated due to the code's unexpected random execution of some parts, but overall, the outcomes should be reasonably comparable. The command that must be entered for the first simulation scenario is just "python3 main.py," which will run the program and start it with a population of 2 ducks, 4 shrimp, and 5 newts. Since beginning locations are determined by applying a random function, results would undoubtedly vary, but the lesson learned would mostly remain the same. The second scenario's instruction was "python3 main.py," which allowed the entry of 3 ducks, 4 shrimps, and 5 newts. You should type "python3 main.py" in the third scenario. You can input 2 ducks, 4 shrimps, and 3 newts here as well; the parameters are taken in the same order, accordingly. When comparing these three scenarios, the first one has the fewest interactions between ducks and newts since the number of two is small and lowers the likelihood of their engaging. In the second simulation, there is a higher frequency of newt interactions. and shrimp, as one could anticipate given the high starting quantity. The third scenario has the smallest initial population so far but appears to produce results in terms of interaction that are somewhat similar. This is because the neighborhood is Von, and the population is high, interaction is limited to a certain extent because cats can only interact to a certain extent.

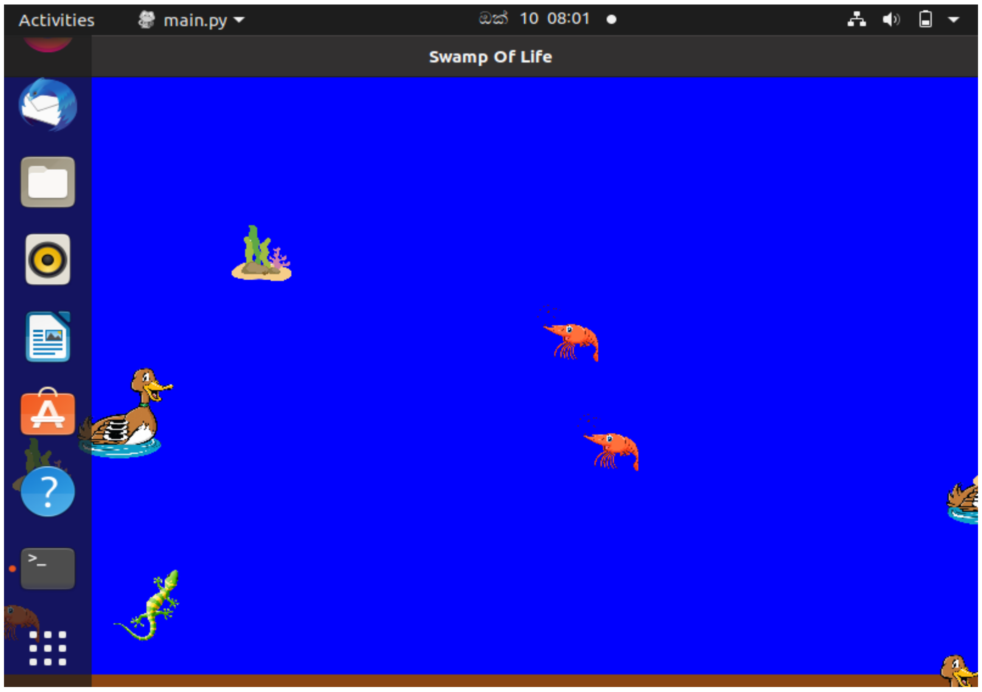
Instead of in Moor, notice the many fewer ducks and newts near them. Since the simulation needs to be able to simulate numerous scenarios with varying parameters, it is evident from these three scenarios that differen of parameters produce different sorts of results.

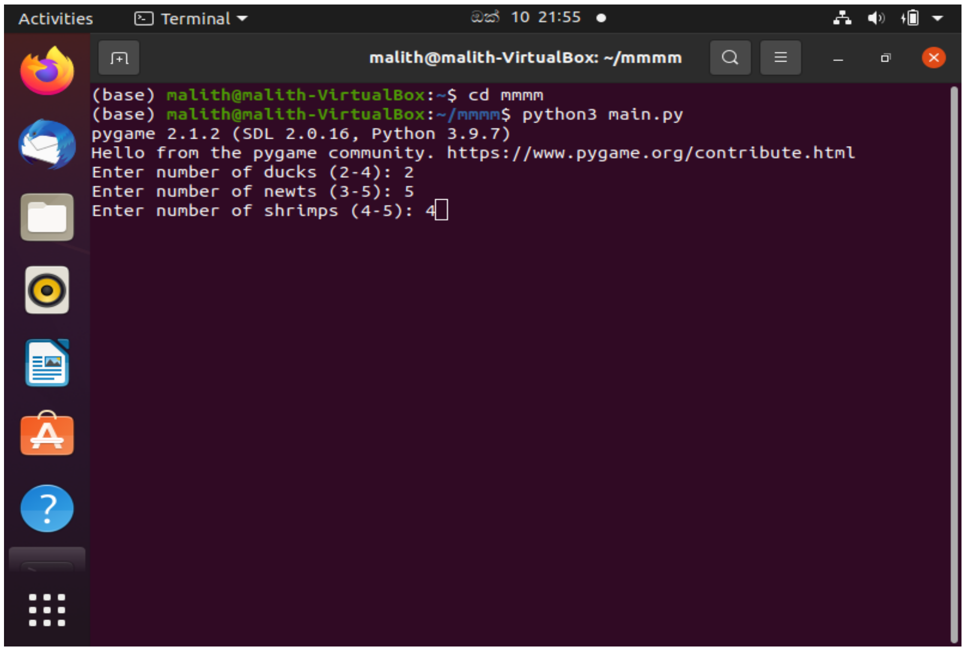
## Results

Simulation results,

Simulation 1:



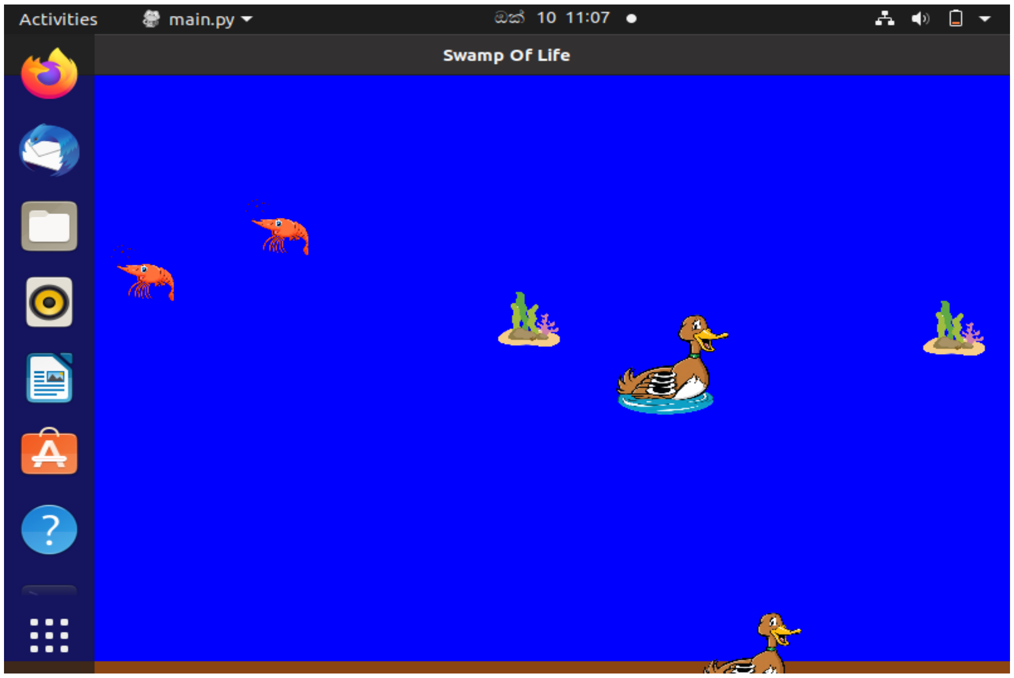


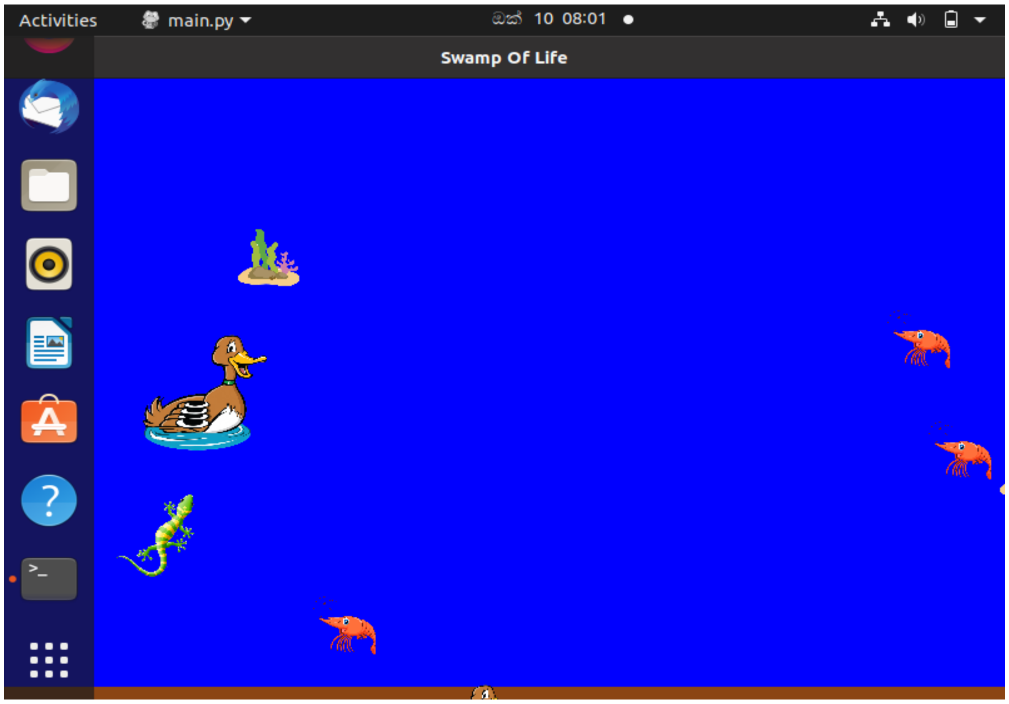


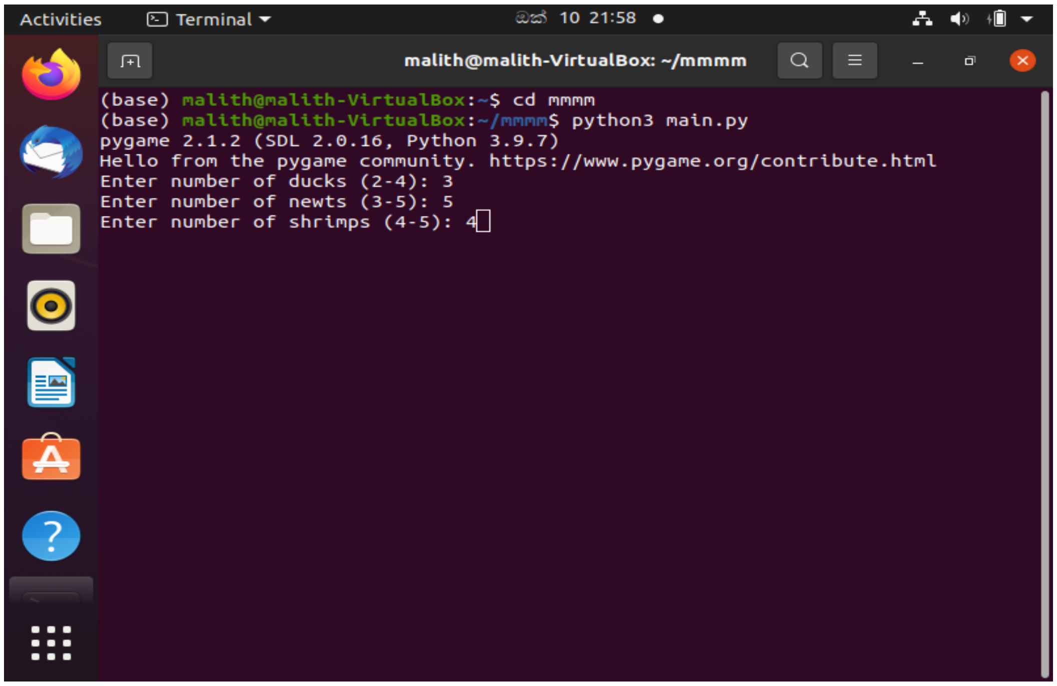
```
(base) malith@malith-VirtualBox:~$ cd mmmm
(base) malith@malith-VirtualBox:~/mmmm$ python3 main.py
pygame 2.1.2 (SDL 2.0.16, Python 3.9.7)
Hello from the pygame community. https://www.pygame.org/contribute.html
Enter number of ducks (2-4): 2
Enter number of newts (3-5): 5
Enter number of shrimps (4-5): 4
```

This is the simulation for a 2 ducks, 4 shrimps, 5 newts and the neighbourhood is moor. Given parameters immediately affect the simulation and outcome since alternative parameter settings produce wildly dissimilar outcomes.

## Simulation 2:



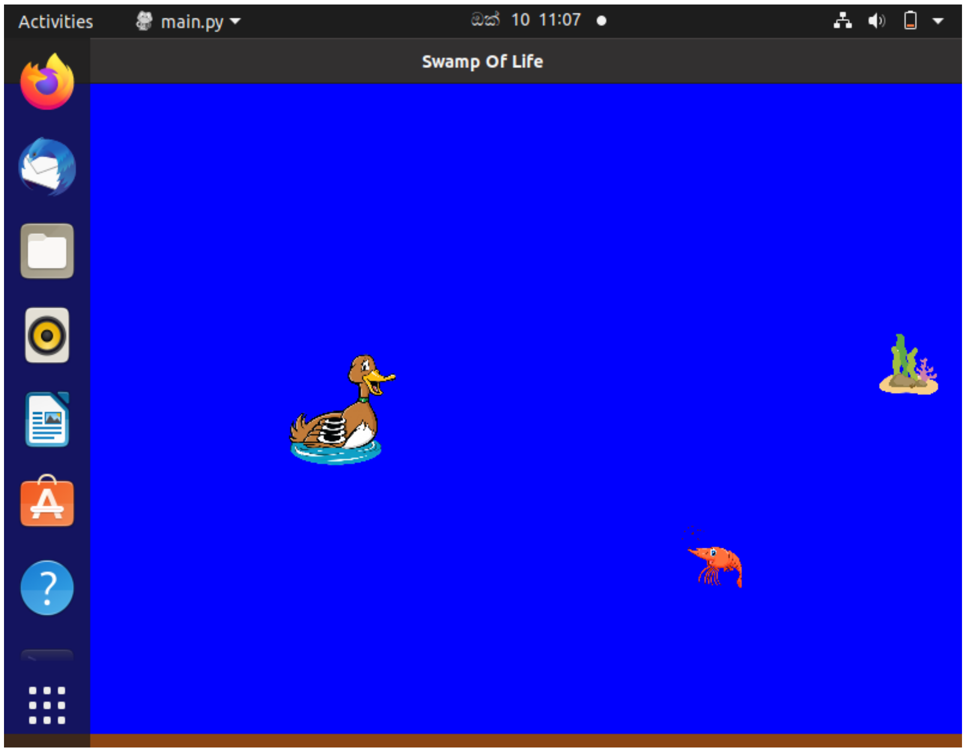


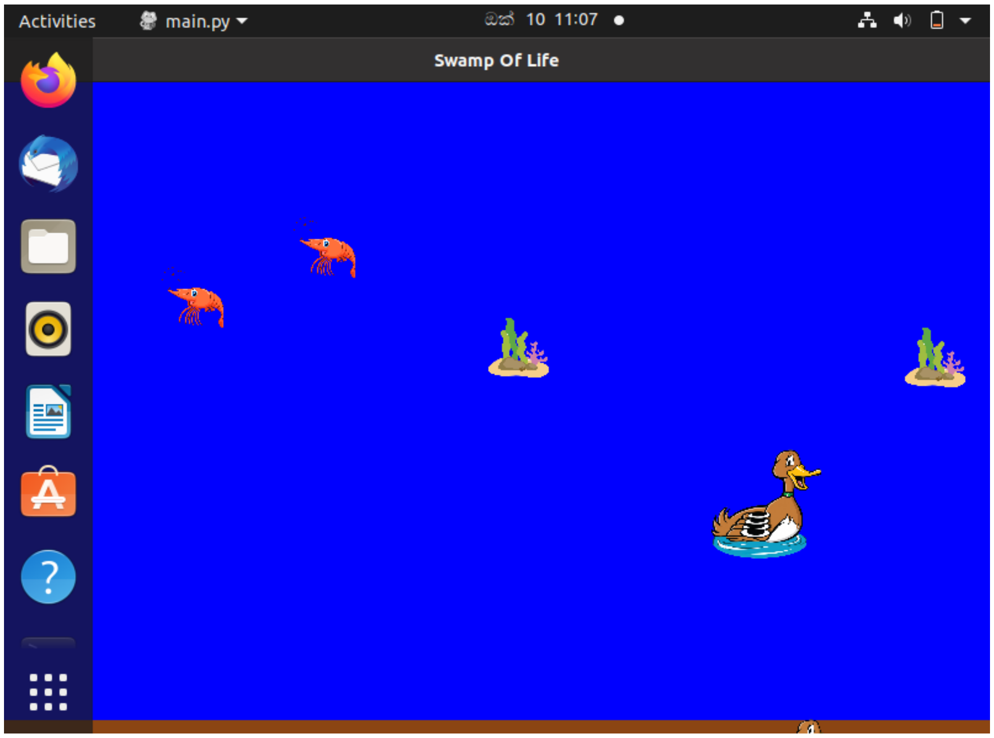


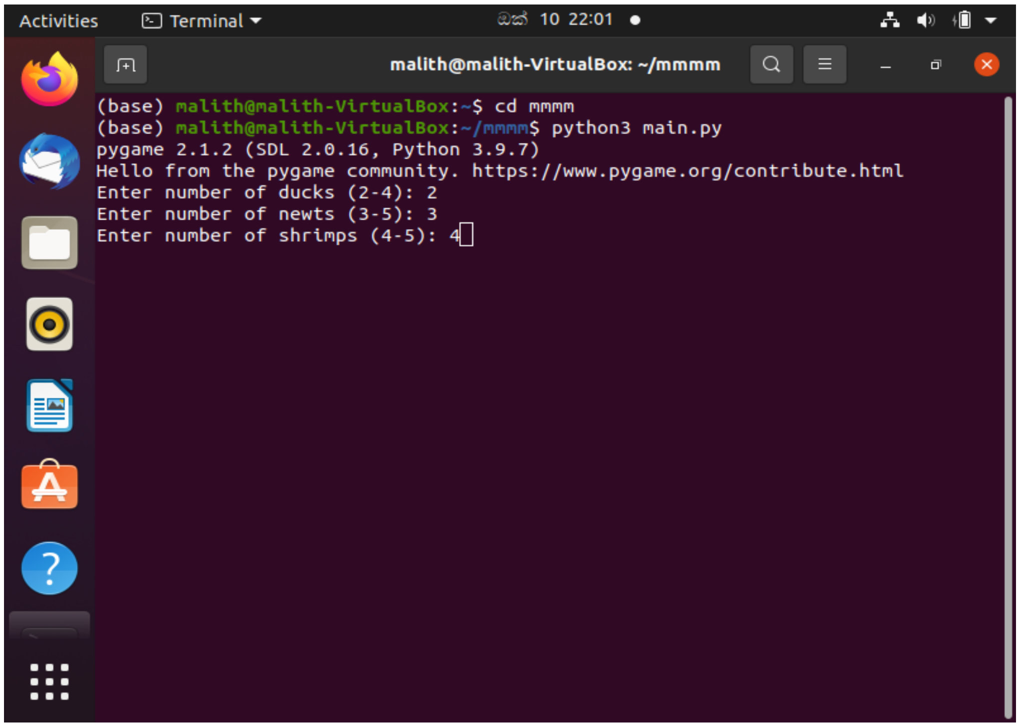
```
(base) malith@malith-VirtualBox:~$ cd mmmm
(base) malith@malith-VirtualBox:~/mmmm$ python3 main.py
pygame 2.1.2 (SDL 2.0.16, Python 3.9.7)
Hello from the pygame community. https://www.pygame.org/contribute.html
Enter number of ducks (2-4): 3
Enter number of newts (3-5): 5
Enter number of shrimps (4-5): 4
```

This is the simulation for a 3 ducks, 4 shrimps, 5 newts and the neighbourhood is moor. Given parameters immediately affect the simulation and outcome since alternative parameter settings produce wildly dissimilar outcomes.

## Simulation 3:







```
(base) malith@malith-VirtualBox:~$ cd mmmm
(base) malith@malith-VirtualBox:~/mmmm$ python3 main.py
pygame 2.1.2 (SDL 2.0.16, Python 3.9.7)
Hello from the pygame community. https://www.pygame.org/contribute.html
Enter number of ducks (2-4): 2
Enter number of newts (3-5): 3
Enter number of shrimps (4-5): 4
```

This is the simulation for a 2 ducks, 4 shrimps, 3newts and the neighbourhood is moor. Given parameters immediately affect the simulation and outcome since alternative parameter settings produce wildly dissimilar outcomes.

## Conclusion and Future work

Thus, a great deal of knowledge can be gained from the program, and the animal life in the swamp may be partially analyzed in relation to many parameters. The three aforementioned situations illustrate how various settings may have an impact on both the simulation's output and the outcomes as a whole. The main takeaway is that the inputted parameters alter the simulation's overall picture, allowing the user to roughly understand the life cycles of ducks, shrimp, and newts. I was unable to implement all features, but overall, the majority of them are covered, and the simulation will be much more accurate once these missing elements are added.

## References

- COMP1005. 2021. “shrimp.py” Assignment sample code, COMP1005 Fundamentals of Programming, Semester 2, 2022.

- COMP1005. 2022. “pasciroExample.py” Assignment sample code, COMP1005 Fundamentals of Programming, Semester 2, 2022.
- COMP1005. 2022. “swamp.py” Assignment sample code, COMP1005 Fundamentals of Programming, Semester 2, 2022.
- COMP1005. 2022. “swamplife.py” Assignment sample code, COMP1005 Fundamentals ofProgramming, Semester 2, 2022.
- COMP1005. 2022. “yebbleLife_base.py” Assignment sample code, COMP1005 Fundamentals of Programming, Semester 2, 2022
- COMP1005. 2022. “test3.py”, “characters.py” PracTest03 code, COMP1005 Fundamentalsof Programming, Semester 2, 2021

Document version: 1.1 (2015-11-15)

Curtin University – Department of Computing

# Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

| | | | |
|---|---|---|---|
| Last name: | HewaWandige | Student ID: | 20926076 |
| Other name(s): | Malith Pramuditha | | |
| Unit name: | Fundamentals of Programming | Unit ID: | COMP1005 |
| Lecturer / unit coordinator: | Dr. Valerie Maxville | Tutor: | Ms. Umanga |
| Date of submission: | 11.10.2022 | Which assignment? | FOP Ass (Leave blank if the unit has only one assignment.) |

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Signature: Malith

Date of signature: 11.10.2022

*(By submitting this form, you indicate that you agree with all the above text.)*