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:	Moktan	Student ID:	21522132
Other name(s):	Nima Dorji		
Unit name:	Fundamentals of Programming	Unit ID:	COMP5005
Lecturer / unit coordinator:	Valerie Maxville	Tutor:	
Date of submission:	18/10/2023	Which assignment?	(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: [illegible]

Date of signature: 18/10/2023

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

Report

Simulation of Nature using Python

Name: Nima Dorji Moktan

Student ID: 21522132

DECLARATION OF ORIGINALITY

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 plagiarize 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.

Abstract

This project on Simulation of Natural Environment using Python aims to create a comprehensive and interactive simulation of a virtual garden ecosystem. In this simulation, it is designed to emulate the behaviors and interactions of various entities within a dynamic environment. This simulation integrates various entities, including ants, butterflies, ladybugs, caterpillars, plants, rocks, and raindrops, to offer users an engaging experience that mirrors the intricate relationships and behaviors found in natural environments.

Introduction

To understand the complex dynamics of natural ecosystems is important for comprehending and appreciating the delicate balance that sustains life on our planet. This project endeavors to create a virtual representation of a garden environment where various entities interact within a complex ecological system. By modeling the behaviors and relationships between different organisms and elements, the project aims to provide insights into the complexities of nature and the interplay between living and non-living components within an ecosystem.

Objectives and scope

The primary goal of the Simulation of Natural Environment using Python project is to create a realistic representation of a dynamic garden ecosystem. It aims to simulate the behaviors and interactions of diverse entities within the environment, considering their responses to external factors such as terrain features and simulated weather conditions. Additionally, the project seeks to facilitate a user-friendly interface for individuals to observe, study, and gain insights into the complexities of natural ecosystems.

Methodology

The implementation of the project revolves around an object-oriented approach. The project comprises distinct classes representing different entities and elements within the simulated ecosystem. These classes integrate specific attributes and methods defining the behaviors, movements, and interactions of the corresponding entities. Moreover, the project incorporates terrain data from a CSV file to visualize the garden landscape. The simulation operates within a controlled time step loop, dynamically updating the positions and movements of entities based on predefined rules (logical movement of entities based on their characteristics) and environmental constraints.

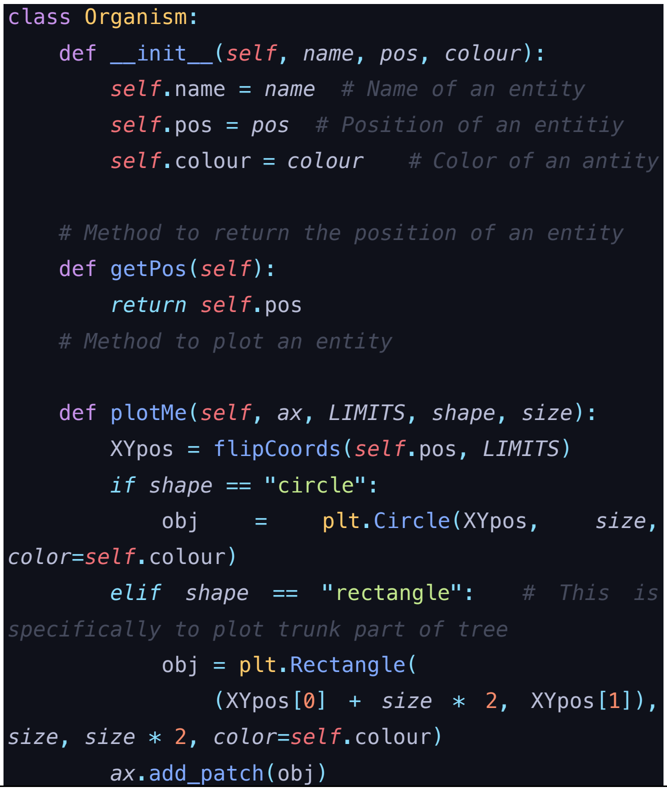
User Guide

The implementation of this project is at basic level, so to engage and explore the Simulation, users can follow these simple steps:

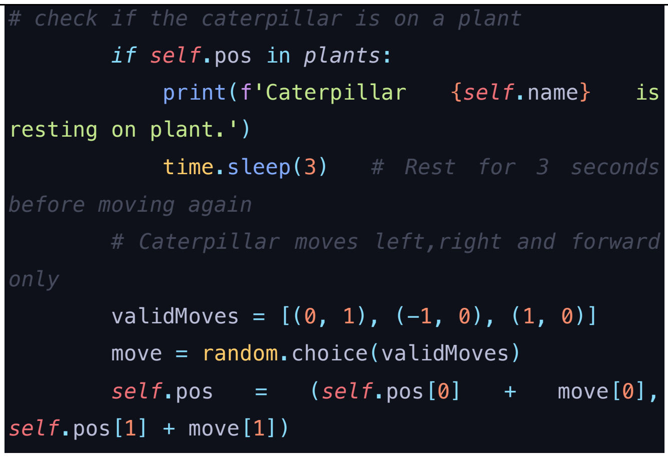
- **Installation and Setup**: Ensure that Python is installed and then clone the project repository from the designated source (Note: Make sure that all the files mentioned in the README are available and in a single folder).
- **Execution**: Run the main script to start the simulation.
- **Interaction**: Observe the behaviors of simulated entities within the garden environment and their responses to environmental stimuli.
- **Exploration**: Analyze the relationships between entities, terrain features, and environmental factors such as rainfall.
- **Termination**: The simulation terminates automatically after a certain period of timestamp.

Traceability Matrix

The Traceability Matrix correlates the project requirements to the implemented features and components, ensuring coherence between the defined objectives and realized functionalities. This allows for a comprehensive assessment of the project's performance and adherence to the initial specifications. The entire process of coding and testing the feature is captured using a git version control which is included in the project folder and GitHub repository.

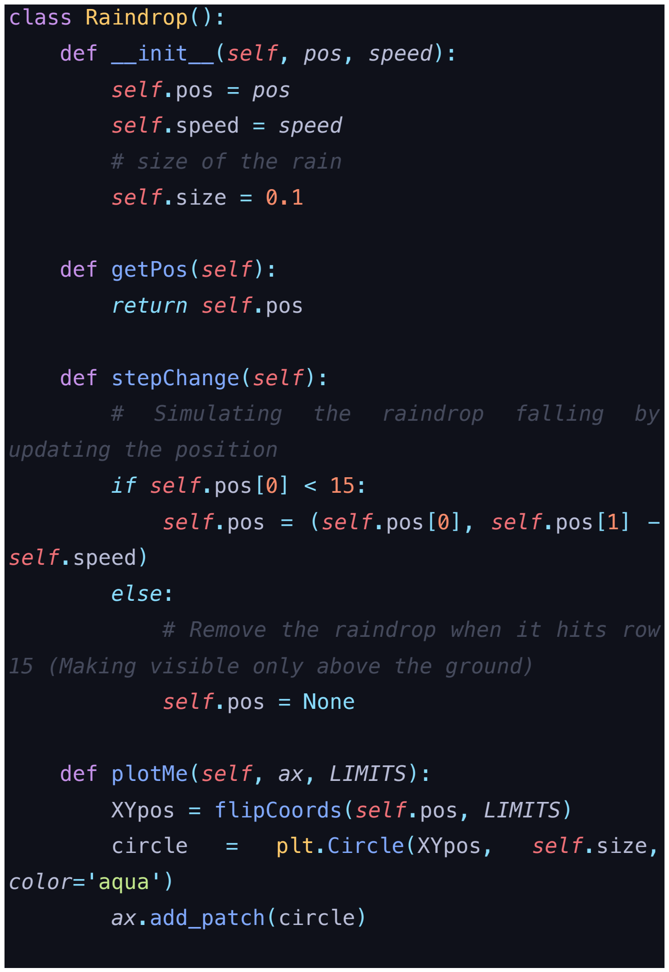
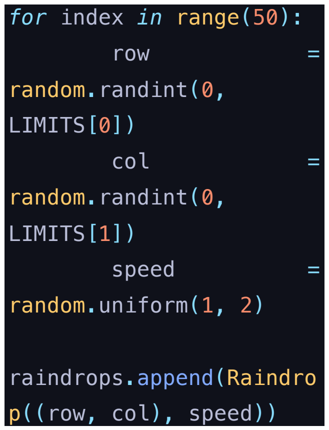
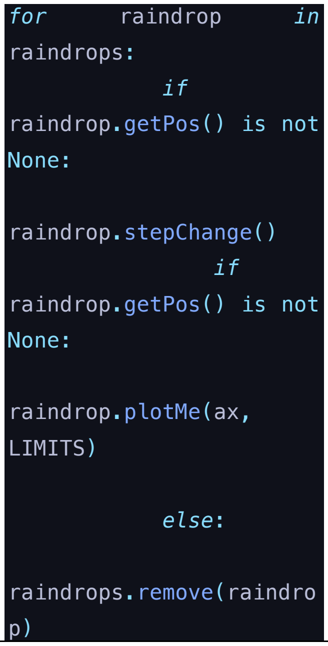
Requirement	**Feature**	**Code reference**	**Test reference**	**Completion date**
Entity Design	Designing the object skeleton, its component and methods related to it.		The super class is inherited in the succeeding classes which is tested working	10/10/2023

```python
class Organism:
    def __init__(self, name, pos, colour):
        self.name = name  # Name of an entity
        self.pos = pos  # Position of an entitiy
        self.colour = colour    # Color of an antity

    # Method to return the position of an entity
    def getPos(self):
        return self.pos
    # Method to plot an entity

    def plotMe(self, ax, LIMITS, shape, size):
        XYpos = flipCoords(self.pos, LIMITS)
        if shape == "circle":
            obj    =    plt.Circle(XYpos,    size,
color=self.colour)
        elif  shape  ==  "rectangle":   #  This  is
specifically to plot trunk part of tree
            obj = plt.Rectangle(
                (XYpos[0]  +  size  *  2,  XYpos[1]),
size, size * 2, color=self.colour)
        ax.add_patch(obj)
```

Entity Behavior Simulation	Implementation of distinct behaviors for ants, butterflies, ladybugs, and caterpillars
	
	Note: In this feature, I am trying to add a movement feature to the caterpillar. If the caterpillar encounters a plant, they can rest and eat the leaves for a certain time. Also, caterpillar moves in three directions (left, right and forward only)
		
		`# check if the caterpillar is on a plant`
		`        if self.pos in plants:`
		`            print(f'Caterpillar {self.name} is resting on plant.')`
		`            time.sleep(3)  # Rest for 3 seconds before moving again`
		`        # Caterpillar moves left,right and forward only`
		`        validMoves = [(0, 1), (-1, 0), (1, 0)]`
		`        move = random.choice(validMoves)`
		`        self.pos = (self.pos[0] + move[0], self.pos[1] + move[1])`
			This code snippet was tested in two forms, one using the visual when the simulation is run and one by printing the valid moves after each timestamp (loop). Looking at the simulation, the caterpillar moves only in three directions defined by the valid moves. Also, the caterpillar stops when it encounters a plant. This test verifies that the code snippet is working.	08/10/2023

Guided Behavior Simulation	The ant is guided to make tunnel or hole only in the specified direction and if ant encounters obstacle (rocks), the ants will change the direction of movement and change the course.	`def stepChange(self, subgrid, rocks):`
		`        validMoves = [(0, 0), (-1, 0), (1, 0), (0, -1), (0, 1)]`
		`        possible_moves = []`
		`        for move in validMoves:`
		`            new_pos = (self.pos[0] + move[0], self.pos[1] + move[1])`
		`            if new_pos not in rocks:`
		`                # The ant will move in only tunnel`
		`                for a in range(len(subgrid)):`
		`                    for b in range(len(subgrid)):`
		`                        if subgrid[a, b] == 0:`
		`possible_moves.append((a-1, b-1))`
		
		`        if possible_moves and len(possible_moves) > 0:`
		`            move = random.choice(possible_moves)`
		`            self.pos = (self.pos[0] + move[0], self.pos[1] + move[1])`
			The code snippet is validated by observing the simulation.	07/10/2023

Terrain Representation	Integration of terrain data from CSV file for visual representation	`# Getting the terrain value from CSV file`
		`    tlist = []`
		`    terrain_obj = open('garden3.csv', 'r')`
		`    for line in terrain_obj:`
		`        line_s = line.strip()`
		`        # using list comprehension to convert each value into integer value`
		`        ints = [int(x) for x in line_s.split(',')]`
		`        tlist.append(ints)`
		`    terrain_obj.close()`
		`    # End of reading value from csv`
			There is visible change in the background of the simulation.	30/09/2023

Simulation of Rain	The simulation of rain has been implemented in this project		 …. 	08/10/2023

```python
class Raindrop():
    def __init__(self, pos, speed):
        self.pos = pos
        self.speed = speed
        # size of the rain
        self.size = 0.1

    def getPos(self):
        return self.pos

    def stepChange(self):
        # Simulating the raindrop falling by updating the position
        if self.pos[0] < 15:
            self.pos = (self.pos[0], self.pos[1] - self.speed)
        else:
            # Remove the raindrop when it hits row 15 (Making visible only above the ground)
            self.pos = None

    def plotMe(self, ax, LIMITS):
        XYpos = flipCoords(self.pos, LIMITS)
        circle = plt.Circle(XYpos, self.size, color='aqua')
        ax.add_patch(circle)
```

```python
for index in range(50):
    row = random.randint(0, LIMITS[0])
    col = random.randint(0, LIMITS[1])
    speed = random.uniform(1, 2)

    raindrops.append(Raindrop((row, col), speed))
```

….

```python
for raindrop in raindrops:
    if raindrop.getPos() is not None:
        raindrop.stepChange()
        if raindrop.getPos() is not None:
            raindrop.plotMe(ax, LIMITS)

    else:
        raindrops.remove(raindrop)
```

Implementation Details

Super Class Organism:

The ‘Organism’ superclass acts as the foundational class for various entities in the garden. It provides a common structure for entities like Ants, Butterflies, Ladybugs, and Caterpillars. This superclass contains shared characteristics such as the name, position, and color of each entity. Additionally, it offers essential methods such as ‘getPos()’ for retrieving the position and ‘plotMe()’ for visual representation within the garden. By serving as a parent class, it ensures that fundamental attributes and functionalities are inherited and utilized consistently by the specific entities.

Other Entities (Ant, Butterfly, Ladybug, Caterpillar):

Each of these entities inherits from the ‘Organism’ superclass and possesses its own distinct traits and behaviors. For instance, the ‘Ant’ class includes methods for managing its movements in the garden. Similarly, the ‘Butterfly,’ ‘Ladybug,’ and ‘Caterpillar’ classes exhibit unique behaviors, such as resting and segmented movement. These classes build upon the attributes and methods provided by the superclass, extending and implementing them to suit the specific characteristics and actions of each entity type.

Entity Behavior:

The behavior of each entity is defined within their respective classes, outlining how they interact with the garden environment. For example, the ‘Butterfly,’ ‘Ladybug,’ and ‘Caterpillar’ classes demonstrate resting behavior when they encounter plants, mimicking natural behavior. These behaviors are implemented using a blend of random movements and interactions with the garden terrain, plants, and other entities.

Terrain Representation:

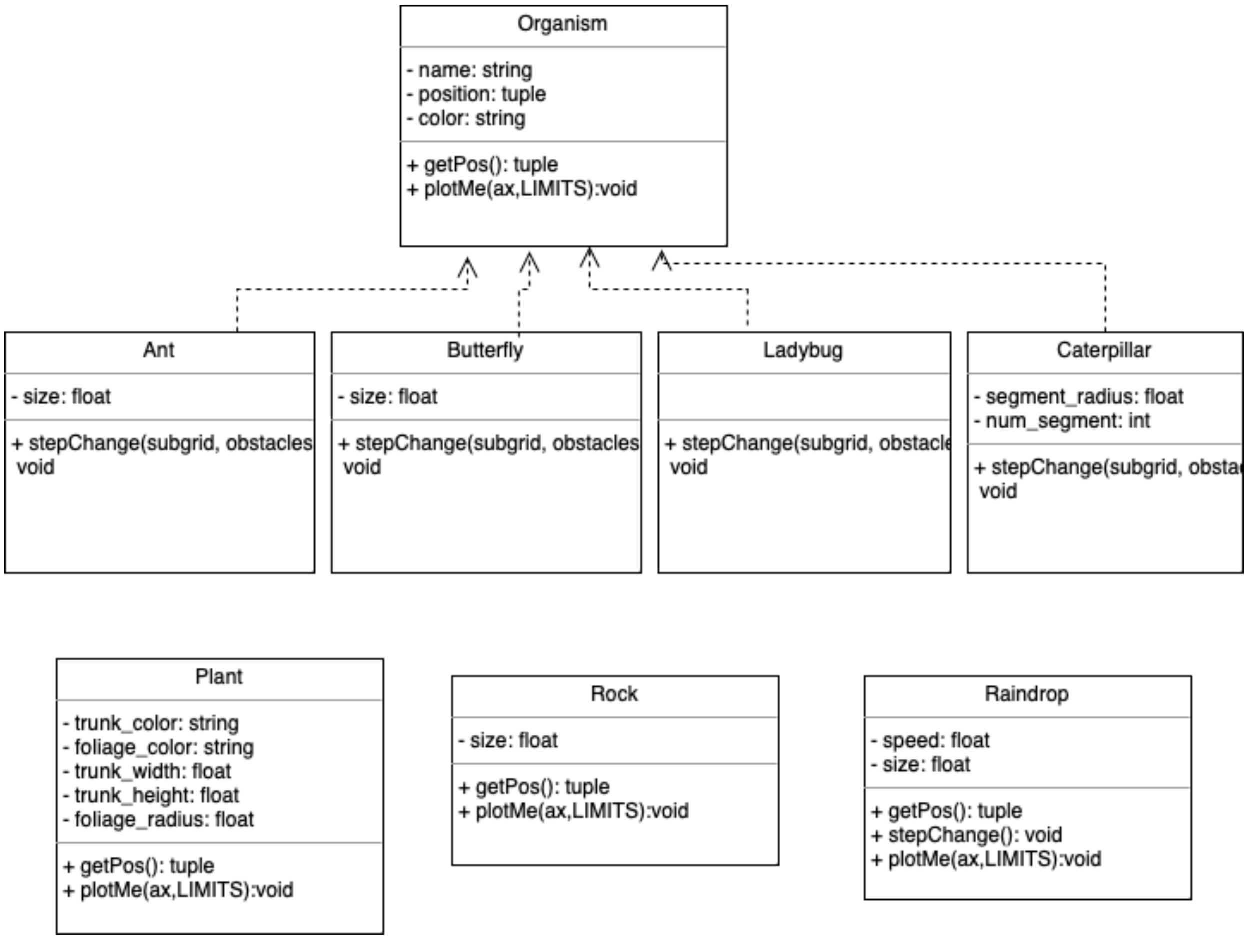
The ‘Plant’ and ‘Rock’ classes are responsible for visually representing plants and rocks within the garden. ‘Plant’ class defines attributes like trunk color, foliage color, trunk width, and foliage radius, creating a detailed visual representation of a plant. Similarly, the ‘Rock’ class specifies the color and size of the rock for rendering in the garden. These classes utilize methods like ‘plotMe()’ to effectively display these objects within the garden based on their defined characteristics.

Rain Simulation:

The ‘Raindrop’ class facilitates the simulation of rain within the garden environment. It replicates the movement of raindrops using a defined speed and updates their position accordingly. The ‘plotMe()’ method is used to visually represent the raindrops with a specific size and color. The rain simulation adds a dynamic element to the garden, enhancing the overall visual experience and contributing to the realism of the simulation environment.

Through the combined use of these components, the garden simulation becomes more engaging and immersive, offering a comprehensive and interactive representation of natural ecosystems.

UML Diagram



Simulation Dynamics

The Simulation of Natural Environment in Python project orchestrates simulation dynamics to ensure a realistic and engaging user experience. Within the controlled time step loop, each entity's behavior adheres to specific rules and constraints, enabling dynamic movements, interactions, and responses to environmental stimuli. Collision detection algorithms and environmental interaction models are integrated to replicate the intricate dynamics observed in real-world ecosystems.

Showcase of Features

The project highlights several key features that contribute to its immersive and educational value:

- **Realistic Entity Behaviors**: Accurate emulation of behaviors, movements, and interactions of simulated entities within the garden environment, providing insights into the complexities of natural ecosystems.
- **Interactive Terrain Visualization**: Detailed representation of the garden's topography and landscape features, allowing users to observe the impact of terrain elements on entity movements and interactions.
- **Dynamic Environmental Simulation**: Real-time representation of environmental factors such as rainfall, displaying the influence of weather conditions on the ecosystem and its inhabitants, thereby enriching the overall simulation experience.

Analysis

The Simulation of Natural Environment in Python project has demonstrated effectiveness as a valuable tool for educational and research purposes. The comprehensive implementation of entity behaviors, terrain visualization, and dynamic environmental simulation has provided users with a holistic understanding of the intricacies of natural ecosystems. Further refinements and enhancements could be considered, including the integration of additional environmental factors, expansion of entity interaction models, and the implementation of advanced visualization techniques to improve the simulation's fidelity and user experience.

Future Extensions and Developments

Future developments of the Simulation of Natural Environment in Python project could include the incorporation of more complex ecological relationships and environmental interactions. These developments might encompass the integration of predator-prey relationships, seasonal variations, and climate change simulations to create a more dynamic and realistic virtual ecosystem. Additionally, advanced data visualization techniques and improved user interactivity could enhance the overall user experience and educational value of the simulation.

Conclusion

In conclusion, the Simulation of Natural Environment in Python project has effectively emulated the intricate dynamics of a garden ecosystem. Through its accurate representation of entity behaviors, sophisticated terrain visualization, and dynamic environmental simulation, the project serves as a valuable platform for individuals interested in the study and appreciation of natural ecosystems. With potential enhancements and developments, the project remains a versatile and educational tool for understanding the complexities of natural environments.

References

Matplotlib. (n.d.). Rain simulation. Matplotlib: Visualization with Python. Retrieved October 1, 2023, from https://matplotlib.org/stable/gallery/animation/rain.html

Visual Paradigm International. (n.d.). UML Class Diagram Tutorial. Visual Paradigm: Unified Modeling Language (UML) Guide. Retrieved October 5, 2023, from https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/

Maxville, V. (2023). Modelling the World with Objects [Slides]. Blackboard.

Maxville, V. (2023). Relationships in Object-Orientation [Slides]. Blackboard.

Maxville, V. (2023). Assignment Starter Code [Source code]

Moktan, N.D. (2023). PracTest 2 [Source code]. Practical Test. https://lms.curtin.edu.au/webapps/assignment/uploadAssignment?content_id=_11459060_1&course_id=_125873_1&group_id=&mode=view

Moktan, N.D. (2023). PracTest 5 [Source code]. Practical Test. Moktan, N.D. (2023). PracTest 2 [Source code]. Practical Test. https://lms.curtin.edu.au/webapps/assignment/uploadAssignment?content_id=_11459060_1&course_id=_125873_1&group_id=&mode=view

Declaration:

I acknowledge the use of ChatGPT (https://chat.openai.com/chat) in the preparation and writing of my report. I have used ChatGPT to assist with:

- Generating the outline of my project report to assist with the report writing process.

The following prompts were input into ChatGPT:

- Give me an outline of writing a report for the coding project.