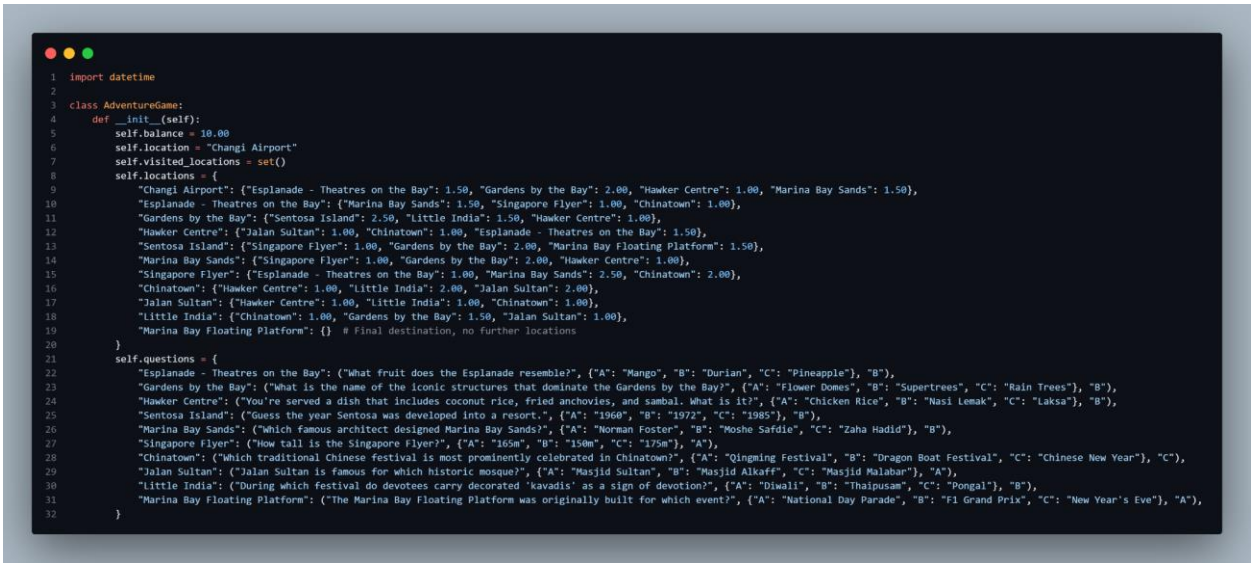
4-2 Milestone Three Enhancement Two: Algorithms and Data Structure



```python
import datetime

class AdventureGame:
    def __init__(self):
        self.balance = 10.00
        self.location = "Changi Airport"
        self.visited_locations = set()
        self.locations = {
            "Changi Airport": {"Esplanade - Theatres on the Bay": 1.50, "Gardens by the Bay": 2.00, "Hawker Centre": 1.00, "Marina Bay Sands": 1.50},
            "Esplanade - Theatres on the Bay": {"Marina Bay Sands": 1.50, "Singapore Flyer": 1.00, "Chinatown": 1.00},
            "Gardens by the Bay": {"Sentosa Island": 2.50, "Little India": 1.50, "Hawker Centre": 1.00},
            "Hawker Centre": {"Jalan Sultan": 1.00, "Chinatown": 1.00, "Esplanade - Theatres on the Bay": 1.50},
            "Sentosa Island": {"Singapore Flyer": 1.00, "Gardens by the Bay": 2.00, "Marina Bay Floating Platform": 1.50},
            "Marina Bay Sands": {"Singapore Flyer": 1.00, "Gardens by the Bay": 2.00, "Hawker Centre": 1.00},
            "Singapore Flyer": {"Esplanade - Theatres on the Bay": 1.00, "Marina Bay Sands": 2.50, "Chinatown": 2.00},
            "Chinatown": {"Hawker Centre": 1.00, "Little India": 2.00, "Jalan Sultan": 2.00},
            "Jalan Sultan": {"Hawker Centre": 1.00, "Little India": 1.00, "Chinatown": 1.00},
            "Little India": {"Chinatown": 1.00, "Gardens by the Bay": 1.50, "Jalan Sultan": 1.00},
            "Marina Bay Floating Platform": {}  # Final destination, no further locations
        }
        self.questions = {
            "Esplanade - Theatres on the Bay": ("What fruit does the Esplanade resemble?", {"A": "Mango", "B": "Durian", "C": "Pineapple"}, "B"),
            "Gardens by the Bay": ("What is the name of the iconic structures that dominate the Gardens by the Bay?", {"A": "Flower Domes", "B": "Supertrees", "C": "Rain Trees"}, "B"),
            "Hawker Centre": ("You're served a dish that includes coconut rice, fried anchovies, and sambal. What is it?", {"A": "Chicken Rice", "B": "Nasi Lemak", "C": "Laksa"}, "B"),
            "Sentosa Island": ("Guess the year Sentosa was developed into a resort.", {"A": "1960", "B": "1972", "C": "1985"}, "B"),
            "Marina Bay Sands": ("Which famous architect designed Marina Bay Sands?", {"A": "Norman Foster", "B": "Moshe Safdie", "C": "Zaha Hadid"}, "B"),
            "Singapore Flyer": ("How tall is the Singapore Flyer?", {"A": "165m", "B": "150m", "C": "175m"}, "A"),
            "Chinatown": ("Which traditional Chinese festival is most prominently celebrated in Chinatown?", {"A": "Qingming Festival", "B": "Dragon Boat Festival", "C": "Chinese New Year"}, "C"),
            "Jalan Sultan": ("Jalan Sultan is famous for which historic mosque?", {"A": "Masjid Sultan", "B": "Masjid Alkaff", "C": "Masjid Malabar"}, "A"),
            "Little India": ("During which festival do devotees carry decorated 'kavadis' as a sign of devotion?", {"A": "Diwali", "B": "Thaipusam", "C": "Pongal"}, "B"),
            "Marina Bay Floating Platform": ("The Marina Bay Floating Platform was originally built for which event?", {"A": "National Day Parade", "B": "F1 Grand Prix", "C": "New Year's Eve"}, "A"),
        }
```

1. **Briefly describe the artifact. What is it? When was it created?**

This artifact is a Choose Your Own Adventure game created in Python that was done as part of my IT140 class. Due to the mention of weapons and how it was perceived at the time of its creation, I had to deviate from the standard “game with weapons” prompt, and turn it into another type of adventure game - one where users travel through my home country of Singapore by answering trivia questions. The goal is to get from Changi Airport to the Marina Bay Floating Platform by spending the least money and visiting six places at minimum.

2. **Justify the inclusion of the artifact in your ePortfolio. Why did you select this item? What specific components of the artifact showcase your skills and abilities in algorithms and data structure? How was the artifact improved?**

I included this artifact in my ePortfolio because it demonstrates my ability to apply advanced algorithms and data structures in real-world problems. Specifically, it shows my understanding of graph traversal through Dijkstra’s Algorithm, where I effectively

implemented a solution that optimises pathfinding based on the cost (of the journeys within the game). The use of priority queue (heapq) in the algorithm illustrates my ability to manage data efficiently, ensuring that the least costly paths are prioritised.

Additionally, the inclusion of the Depth-First Search (DFS) complements my solution by allowing for a detailed exploration of possible routes, ensuring that all potential paths are considered before optimising for the cost. Combining both the DFS and Dijkstra's Algorithm shows that I can not only select appropriate algorithms but also adapt and integrate them into complex problem-solving scenarios.

```python
def dfs_collect_answers(location, locations, questions, visited):
    """
    DFS traversal to visit all locations and collect/display correct answers
    for each location's quiz question.

    Args:
        location (str): The starting location for DFS.
        locations (dict): The graph of locations and their travel costs.
        questions (dict): The questions and answers for each location.
        visited (set): A set to keep track of visited locations.
    """
    if location in visited:
        return
    visited.add(location)

    # If the location has a quiz question, show the correct answer
    if location in questions:
        question, options, correct_answer = questions[location]
        print(f"{location} - {question}")
        print(f"The correct answer is: {correct_answer}\n")

    # Recursively visit all connected locations
    for next_location in locations[location]:
        dfs_collect_answers(next_location, locations, questions, visited)


def activate_cheat(location, locations, questions):
    """
    Function to activate cheat mode and display the correct answers using DFS.

    Args:
        location (str): The current location of the player.
        locations (dict): The graph of locations.
        questions (dict): The questions and answers for each location.
    """
    print("\nCheat mode activated! Showing correct answers for all quiz questions:")
    visited = set()  # To keep track of visited locations
    dfs_collect_answers(location, locations, questions, visited)
    print("Cheat mode traversal completed!\n")
```

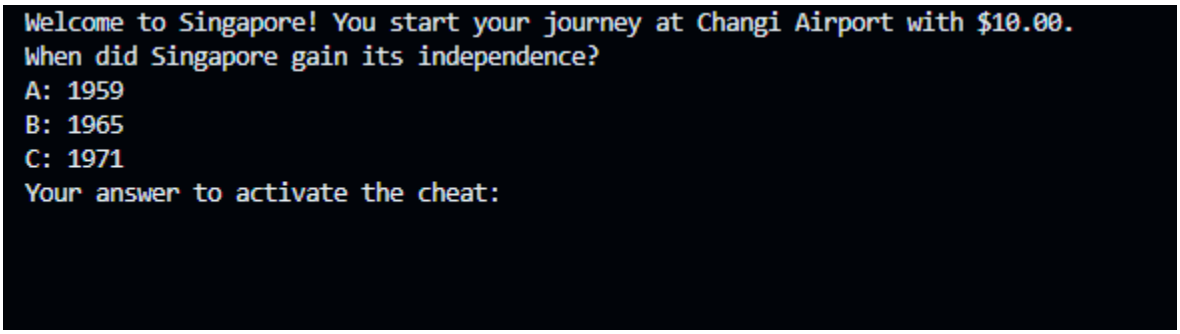
Depth-First Search Algorithm implementation.

The artifact was further improved through the separation of concerns, where I refactored the code into different modules such as `game_data.py`, `input_controls.py`, `utils.py` and `cheat_mode.py`. The modular structure enhances reusability, scalability and maintainability. It allows each component to be worked on independently. For example, Dijkstra's Algorithm is in utils.py, so that travel

logic remains separate from the game mechanics. The DFS is implemented in `cheat_mode.py`, which handles cheat mode independently. These improvements show that I am able to organise complex systems effectively and make the artifact more flexible to extension and modification.

```python
import heapq

def dijkstra_algorithm(locations, start, end, min_visits):
    """
    Finds the cheapest path from start to end using Dijkstra's Algorithm.

    Args:
        locations (dict): A graph representing locations and travel costs.
        start (str): The starting point, e.g., 'Changi Airport'.
        end (str): The destination, e.g., 'Marina Bay Floating Platform'.
        min_visits (int): Minimum number of locations to visit before reaching the end.

    Returns:
        tuple: The path with the minimum cost and the total cost.
    """
    # Initialise the priority queue (cost, location, path, visits)
    priority_queue = [(0, start, [], 0)]
    best_path = []
    min_cost = float('inf')

    # Process the queue
    while priority_queue:
        current_cost, current_location, current_path, visits = heapq.heappop(priority_queue)

        # Update the path with the current location
        current_path = current_path + [current_location]

        # Increment visits for non-start/end locations
        if current_location != start and current_location != end:
            visits += 1

        # Check if we're at the end with enough visits
        if current_location == end and visits >= min_visits:
            if current_cost < min_cost:
                min_cost = current_cost
                best_path = current_path
            continue

        # Add unvisited neighbours to the queue
        for next_location, cost in locations[current_location].items():
            if next_location not in current_path:
                heapq.heappush(priority_queue, (current_cost + cost, next_location, current_path, visits))

    # Return the best path and cost
    return best_path, min_cost
```

Figure 1: Dijkstra's Algorithm



```
Welcome to Singapore! You start your journey at Changi Airport with $10.00.
When did Singapore gain its independence?
A: 1959
B: 1965
C: 1971
Your answer to activate the cheat:
```

Figure 2: Activating the cheat.

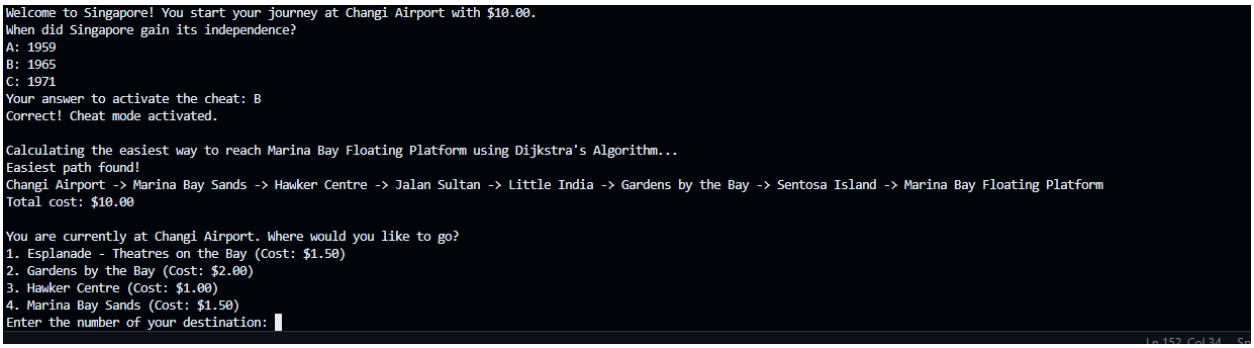


Figure 3: Cheat mode activated

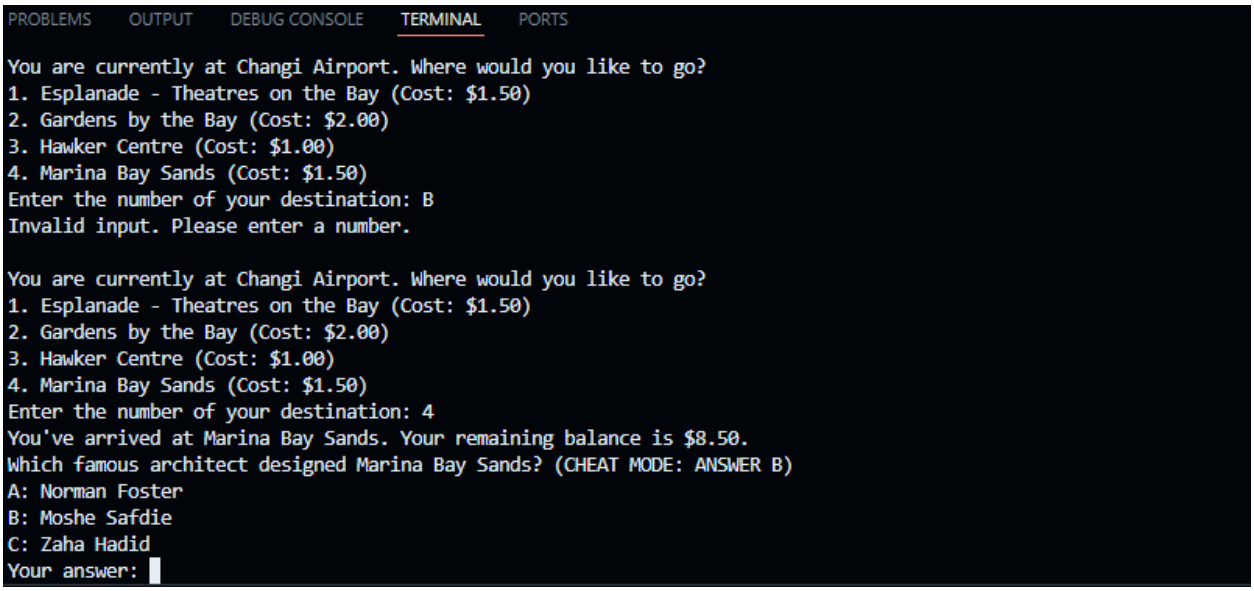


Figure 4: Cheat mode further activated

3. Did you meet the course outcomes you planned to meet with this enhancement in Module One? Do you have any updates to your outcome-coverage plans?

Having initially planned to meet Course Outcomes 1, 2 and 3, I believe that I've met them well. However, I would like to emphasise on having met Course Outcome 3 - **Design and evaluate computing solutions that solve a given problem using algorithmic principles and computer science practices and standards appropriate to its solution, while managing the trade-offs involved in design choices**. I carefully balanced trade-offs such as computational efficiency in pathfinding while ensuring

comprehensive route exploration, which reflects my ability to manage design decisions carefully and effectively. The enhancements align with the outcome I want to achieve. I also note that MAT230 - Discrete Mathematics, as well as CS 210 - Programming Languages have both contributed to my success in this enhancement as they have provided me the knowledge and skills to be able to carry out the same.

To expand on my outcome coverage, Course Outcome 1 is met through extensive and purposeful commenting throughout my code, making it easier for potential collaborators to work on specific portions of my code that they wish to. I have also containerised my work on Docker, much like my other enhancements, so that anyone who wants to run it on their computer may do so without the hassle of installing anything extra.

In Course Outcome 2, I have written a detailed ReadME document on how to access my code. The choice of language that I used in said ReadME is easy to follow and to understand. This means that anyone - regardless of their knowledge and background in technology - can freely access and experience my game without worrying about how difficult it potentially would be to set up.

No updates to my outcome coverage plan - at this stage, I have covered Course Outcomes 1, 2, 3 and 4 through both Enhancement One and Enhancement Two. Course Outcome 5 will be met through Enhancement Three with my database enhancements.

4. **Reflect on the process of enhancing and modifying the artifact. What did you learn as you were creating it and improving it? What challenges did you face?**

Firstly, I did find myself getting overwhelmed at how messy my code was in the beginning. The singular structure of the original game made it very difficult to manage. Everything was packed into a single file, so I found it quite hard to identify and isolate specific functionalities when I had to make changes to it.

Implementing algorithms like Dijkstra's Algorithm and DFS was challenging. While I had theoretical knowledge of these algorithms, applying them to real-world contexts helped me to understand the importance of thinking about efficiency and optimisation. The challenge for me was to ensure that the algorithm integrated smoothly with my existing game logic without making it unnecessarily overly complicated. I also had to pay attention to the edge cases and user experience to make sure that the game remained interactive and technically sound.

Dockerisation presented its own set of challenges. At first, I wasn't very familiar with containerising applications, but then I quickly realised the value of having a consistent environment running the game. Learning to use Docker and Docker Compose was an eye-opener in terms of how modern applications are deployed and maintained in real-world environments. Once I understood the process, I also understood how important this is going to be for future projects.