

Algorithm and Programming Final Project Report

“Text-Based Video Game”



Lecturer:
Jude Joseph Lamug Martinez, MCS

Report done by:
Ostein Vittorio Vellim
2602206783

Binus School of Computer Science Undergraduate Program
Universitas Bina Nusantara
Jakarta
2022

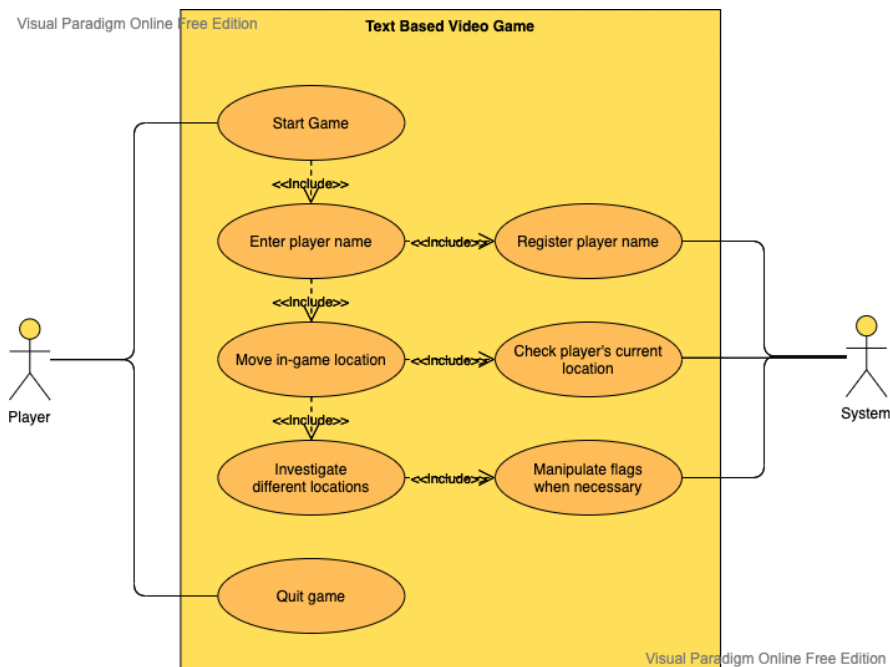
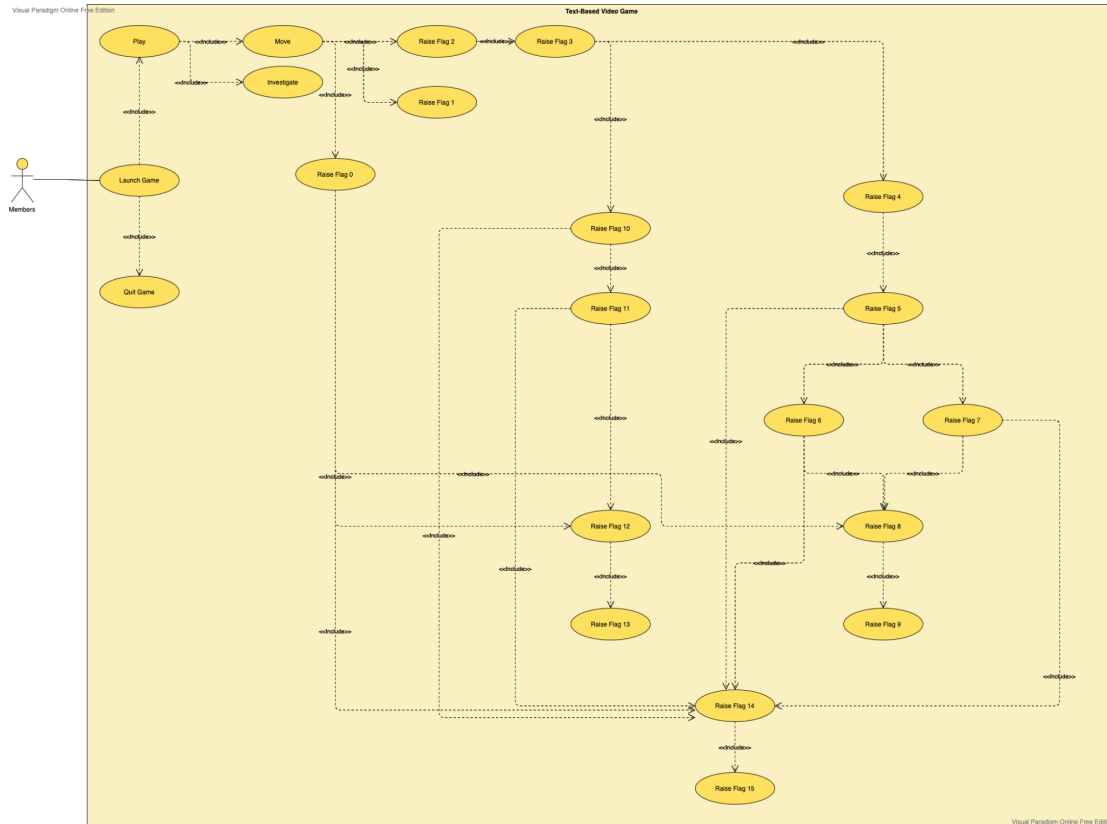
I. Brief Description

A text-based video game is a game that uses written text as its main communication method. As such, the game will relay information through text, to which the player will choose their next action also through text, often using a command prompt. Due to the simple nature of the game, developers can make the game as simple or as complex as they want. One example of a text-based video game is “The Oregon Trail”, an educational strategy game about the struggles of being a pioneer in the 19th century. In it, you guide a family in their expedition to reach Oregon, from buying supplies to surviving the harsh environment. Another thing to note is that these types of games are often possible to complete purely via text. Despite this, more modern examples of text-based games generally accompany the gameplay with matching visuals.

For my final project, I decided to create a simple story-driven text-based video game. By implementing the lessons I learned in the Algorithm and Programming course over the course of the semester. One reason I decided to start this project is that I would like to implement my hobby of writing stories into my university projects somehow, and I believe that making a text-based video game would be a good place to start.

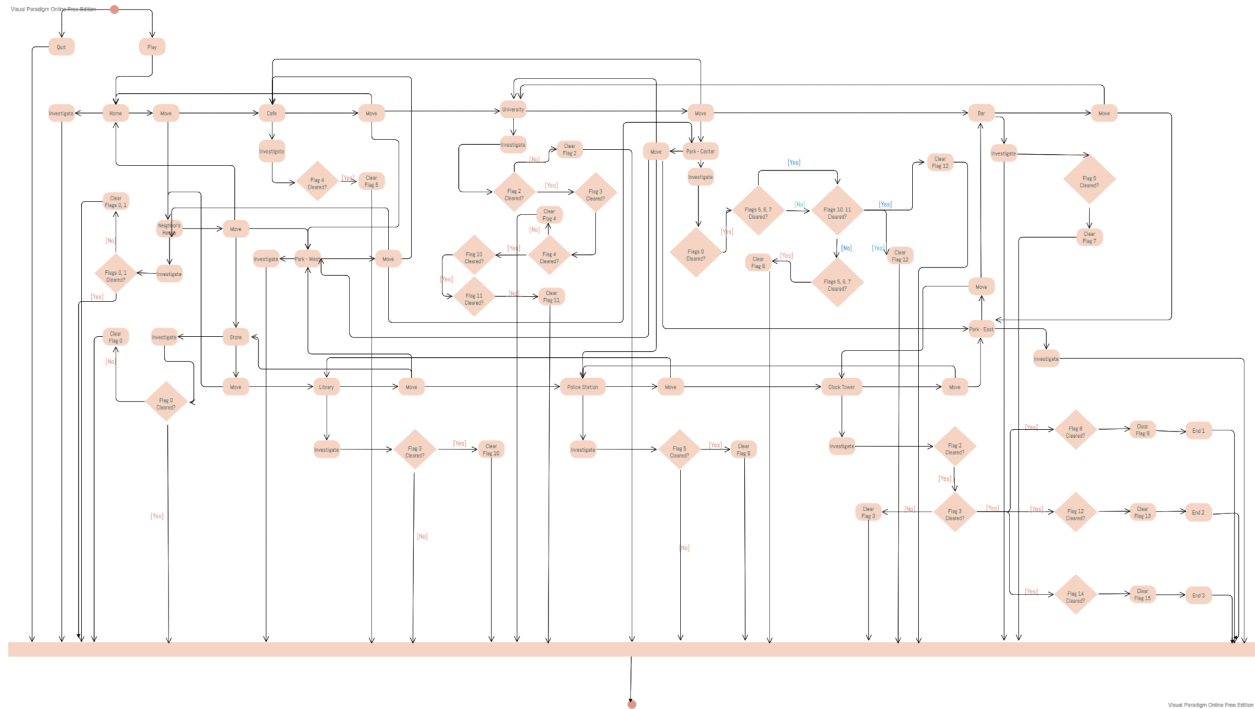
II. User Case Diagram

Attached is the UML User Case Diagram:



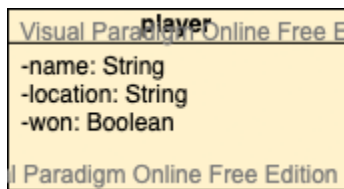
III. Activity Diagram

Attached is the UML Activity Diagram:



IV. Class Diagram

Attached is the class diagram for all the classes used in this program:



V. Modules

Modules imported in my program:

No.	Module Name	Function
1	cmd (command)	Implements command line functionality
2	textwrap	Wraps text around console border
3	sys (system)	Implements system functions which contain elements like functions and variables that can affect various parts of the code. Essentially it allows us to operate on the interpreter itself. In this program, this module imports the sys.stdout.write() and sys.stdout.flush() functions. The function sys.stdout.write() is similar to the print function in Python, while sys.stdout.flush() delays the output for the sys.stdout.write() function. In short, it prints the output one by one rather than simultaneously.
4	os (Operating System)	Implements system functions that correspond with the user's operating system. From this module, I imported the os.system('clear') in my system. This function clears the screen when the function is called.
5	time	Implements time related functions. In this program, this module is used to import time.sleep() function, which can be used to delay the output of certain lines.

VI. Essential Algorithms

My program has three essential systems that is used regularly when running the code:

1. Moving the Player

The first essential algorithm I used in this program is the move function. The move function is also one of the two available actions to the player every time the prompt function is called. When the player chooses to move, they are given four possible directions of movement: north, south, east, and west. Whichever direction they pick, they will be transported to the tile of the corresponding direction.

To execute this function, I first visualized a 4x3 grid that will represent the town, as shown below:

A1 = Home	A2 = Cafe	A3 = University	A4 = Bar
B1 = Neighbor's House	B2 = Park - West	B3 = Park - Center	B4 = Park - East
C1 = Store	C2 = Library	C3 = Police Station	C4 = Clock Tower

As shown, each tile represents a place that the player can go to. The player will be starting in their home and can move to the cafe or the neighbor's house by moving east or south respectively. In each location, certain flags can be raised in order to progress the story, which will be further in the second and third section.

To make the movement system, I created a dictionary containing all the location tiles as its keys. For the values, I implemented another dictionary containing the name of the location, description about the location, as well as the next tile for when the player wants to move up, down, left, or right from the current tile. The picture below shows the dictionary entry for A1 - Home:

```
town_map = {
  'a1': {
    NAME: 'Home',
    DESCRIPTION: "Your simple little apartment. \nIt is quite cheap for a university student like you, and there are a lot of facilities available in the building.",
    UP: 'a0',
    DOWN: 'b1',
    LEFT: 'a0',
    RIGHT: 'a2'
  }
```

However, there is a possibility for the player to move out of the playable area. For instance, if the player's current location is home, should the player move north, there wouldn't be a location for the player to go to, thus creating an error. To combat this, I added a pseudo location called A0 labeled "Border". If the player manages to enter this location, anywhere the player moves will always transport him back to A1 - Home. After that, the player can resume the game normally.

```
town_map = {
  'a0': {
    NAME: "Border",
    DESCRIPTION: "You reached the town border. move any direction to go back home.\n",
    UP: 'a1',
    DOWN: 'a1',
    LEFT: 'a1',
    RIGHT: 'a1',
  },
```

2. Investigating the Location

The second option a player can choose is investigate. There are two steps in completing this function. First, the game will check the player's location, and then the game will check if certain flags are cleared. Once those two are checked, the game will proceed with the dialogue from the location or from another character.

3. Flag System

I implemented a flag system into my game to be used as an event checker system so that each dialogue will be correct according to the progress of the story, and as such will be working in tandem alongside the investigation system. In total, I initialized 16 different flags to be used in the code.

I started by making a dictionary containing all flags numbered 00-15 and setting all of their values as false, as seen in the image attached:

```
flags = {'FLAG00': False, 'FLAG01': False, 'FLAG02': False, 'FLAG03': False,
         'FLAG04': False, 'FLAG05': False, 'FLAG06': False, 'FLAG07': False,
         'FLAG08': False, 'FLAG09': False, 'FLAG10': False, 'FLAG11': False,
         'FLAG12': False, 'FLAG13': False, 'FLAG14': False, 'FLAG15': False}
```

Attached is the flag identifier:

```
###FLAGS###
'''
FLAG00 #Shovel#
FLAG01 #Kate - Neighbor#
FLAG02 #Joseph - Friend 1#
FLAG03 #Chest - 1#
FLAG04 #Joseph - Friend 2#
FLAG05 #Cav - Barista#
FLAG06 #Archie - Sheriff#
FLAG07 #Jean - Bartender#
FLAG08 #Key 1#
FLAG09 #End 1#
FLAG10 #Elm - Librarian#
FLAG11 #Mable - Dean#
FLAG12 #Key 2#
FLAG13 #End 2#
FLAG14 #Key 3#
FLAG15 #End 3#
'''
```

As seen on the image, each flag signifies a different part of the story as well as a point of reference for the player's progression in the story. To illustrate how the flag system works, let's take a look at the flags for the location 'store':

```

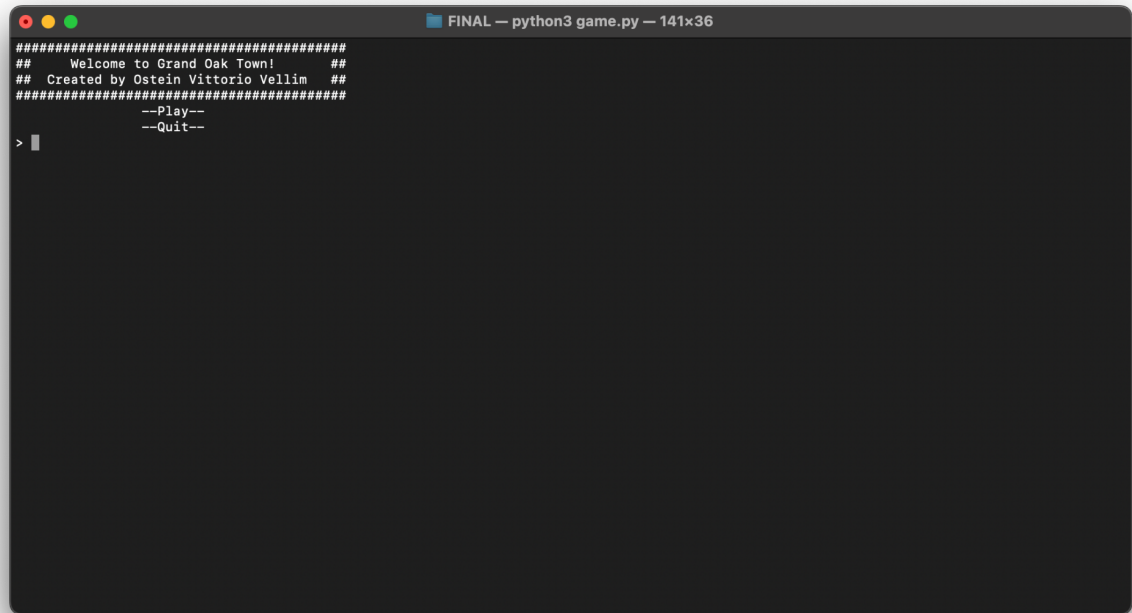
elif myPlayer.location == 'c1':
    if flags['FLAG00'] == True:
        Store1 = ("\n" + "You entered the large department store. You scanned a
        for character in Store1:
            sys.stdout.write(character)
            sys.stdout.flush()
            time.sleep(0.05)
        time.sleep(0.5)

    elif flags['FLAG00'] == False:
        Store2 = ("\n" + "While looking around the store, you spotted a shovel.
        for character in Store2:
            sys.stdout.write(character)
            sys.stdout.flush()
            time.sleep(0.05)
        time.sleep(0.5)
        flags['FLAG00'] = True

```

When the player enters the location 'store' and chooses to investigate, the game will check whether FLAG00 is cleared or not. If it is not cleared (FLAG00 = False), then the game will go through the second dialogue option. After the player clears the dialogue, the flag will be cleared (FLAG00 = True). Otherwise, if the player chooses to investigate the location with FLAG00 already cleared, then the game will go through the first dialogue, but no change will be applied. This is only a single example, and other locations with variance in the flags.

VII. Screenshots



A screenshot of a terminal window with a dark background. The window title bar at the top shows three colored circles (red, yellow, green) on the left and the text "FINAL — python3 game.py — 141x36" on the right. The terminal content consists of several lines of text: a line of 25 hash symbols, two lines of comments enclosed in double hash symbols, another line of 25 hash symbols, and two lines of options: "--Play--" and "--Quit--". At the bottom left, there is a prompt character ">" followed by a small vertical bar representing the cursor.

```
#####  
##   Welcome to Grand Oak Town!   ##  
## Created by Ostein Vittorio Vellim ##  
#####  
--Play--  
--Quit--  
> |
```

```
FINAL — python3 game.py — 141x36
"...Alright that should be everything. You can just go ahead and settle in right now."
"If you would please sign here... and write your name in here..."
> Ostein

"Alright then, Ostein, glad to work with you. Enjoy your new home!"
█
```

```
FINAL — python3 game.py — 141x36
#####
#   Welcome to Grand Oak Town   #
#####

Your home is looking a little barren, but a home's a home

You can move by typing 'move' on the console, and you can 'investigate' by typing 'investigate' in the console.

#####
# HOME #
#####

Your simple little apartment.
It is quite cheap for a university student like you, and there are a lot of facilities available in the building.
Not like you'd have time to visit them anyway.

=====
What would you like to do?
> █
```

VIII. Reflection

This project was simple in nature, but in order to keep its simplicity, it takes a lot of work. I saw a lot of my peers using more intricate modules like pygame, but I decided to make myself work with some very basic modules. Despite that, I realized that you could do a lot with the main functions itself. For instance, the map system. It is quite intuitive to use dictionaries as a navigation system, and I believe that it worked incredibly well for a simple project like mine. And I like to think of my project as that: simple.

I used to think that programmers and developers were skilled because they could work with more complex systems, more complex languages, more complex modules. But through this project, I realized that it is not entirely correct. Sure, more powerful systems can create bigger things, faster things, more effective things. But a truly skilled programmer should be able to create something out of anything; even the simpler things.

Am I a skilled programmer? No. Far from it. I could barely print out a line of code just about a year ago. But going through this program and starting and completing this project, I realized that I could be much better, and that I have grown from where I was. In the future I would be facing more challenges, all tougher than my simple creation, but the journey to grow isn't a straight road. Still, I'll enjoy the ride while it lasts.

IX. Resources and Links

1. Python - Text RPG Tutorial on Youtube by Bryan Tong

<https://www.youtube.com/playlist?list=PL1-slM0ZOosXf2oQYZpTRAoeuo0TPiGpm>