Sam Heinz

s\_h267@student.usc.edu.au

Task 3: Python Project

ICT112: Programming Fundamentals

Student ID: 1166675   
Campus: Online  
Tutor’s name: Andrew Lang  
Due date: June 9th, 2023  
Submission date: June 9th, 2023  
Word Count: 1764

Table of Contents

[1 Introduction 1](#_Toc137121042)

[2 Problem Description 2](#_Toc137121043)

[3 Assumptions 3](#_Toc137121044)

[4 Flowcharts 4](#_Toc137121045)

[4.1 Initial 4](#_Toc137121046)

[4.2 Final 4](#_Toc137121047)

[5 Design 5](#_Toc137121048)

[5.1 Top-Level view 5](#_Toc137121049)

[5.2 Detailed view 5](#_Toc137121050)

[6 Test Plan 7](#_Toc137121051)

Table of Figures

[Figure 1: 4.1.1 Initial flowchart - Lucidchart. 4](file:///C:\Users\wheel\Documents\GitHub\Task-3-ICT112\docs\ICT112%20-%202023%20-%20Task%203.docx#_Toc137196232)

[Figure 2: 4.2.1 Final flowchart - Lucidchart. 4](file:///C:\Users\wheel\Documents\GitHub\Task-3-ICT112\docs\ICT112%20-%202023%20-%20Task%203.docx#_Toc137196233)

[Figure 3: 5.2.1a display\_actions function. 5](#_Toc137196234)

[Figure 4: 5.2.1b example of running code. 5](#_Toc137196235)

[Figure 5: 5.2.2 data\_store.move() function 6](file:///C:\Users\wheel\Documents\GitHub\Task-3-ICT112\docs\ICT112%20-%202023%20-%20Task%203.docx#_Toc137196236)

[Figure 6: 5.2.3 ui.shop() function 6](file:///C:\Users\wheel\Documents\GitHub\Task-3-ICT112\docs\ICT112%20-%202023%20-%20Task%203.docx#_Toc137196237)

[Figure 7: 5.2.4 extract from main, after user chooses an action. 6](file:///C:\Users\wheel\Documents\GitHub\Task-3-ICT112\docs\ICT112%20-%202023%20-%20Task%203.docx#_Toc137196238)

[Figure 8: 6.1 extract from main() showing test() function. 7](file:///C:\Users\wheel\Documents\GitHub\Task-3-ICT112\docs\ICT112%20-%202023%20-%20Task%203.docx#_Toc137196239)

[Figure 9: 6.2a lines 1-31 of TEST file. 7](file:///C:\Users\wheel\Documents\GitHub\Task-3-ICT112\docs\ICT112%20-%202023%20-%20Task%203.docx#_Toc137196241)

[Figure 10: 6.2b lines 32-62 of TEST file. 7](file:///C:\Users\wheel\Documents\GitHub\Task-3-ICT112\docs\ICT112%20-%202023%20-%20Task%203.docx#_Toc137196240)  
[Figure 11: 6.3 test() function. 7](file:///C:\Users\wheel\Documents\GitHub\Task-3-ICT112\docs\ICT112%20-%202023%20-%20Task%203.docx#_Toc137196242)

# Introduction

In this report will include a design and test plan procedure for a problem. This report will describe the product that was produced based on the problem, what assumptions were made during the process of production, planning and design aspects that were conducted and the testing system that was used to verify that the final product was to standard.

# Problem Description

As per the task description for the ‘*Multi madness text game*’:

“Your program will load room setups from a file. The setup could include names, items, room descriptions, text instructions, directions to other rooms, and hints (unlocked if you use a certain item).

After giving instructions, the program will then ask the player for actions - a player can move North, South, East, or West.

They can pick up an object (maximum of 3), they can drop an object, and can use an object to get a hint.

Keep the number of moves as the score. The game is over when the player reaches the escape room or asks to stop.

Advanced - load in a player configuration from a file with name and other details including their wallet with cash. The player pays when picking up an object and gets a portion back if they drop it before using it. At the end of the game write the updated player details to the configuration file.”

The produced product will meet all of these requirements, except for the “The player pays when picking up an object”, as the plot of the story is that the player has been snatched and is stuck within an endless loop, the player not having anything on them works better with the story, which is also why it will also change how the end requirement, “Keep the number of moves as the score. The game is over when the player reaches the escape room or asks to stop*”*, will work. As the story follows a character stuck in this endless loop without being able to escape, the escape room simply leads them back to the start, restarting the loop. Although they will have successfully made it through the game, the game won’t necessarily ‘end’. Although the user will be able to save and exit to return later.

With regards to testing, the task proposes the requirements:

“The test plan should include paths (maybe in an input file using piping). You should break the problem into subproblems and solve and test each. Example - read the file, initialise data structure, and each command. Use functions and determine the best data structure to store all the information.”

The testing and main program will both incorporate this, as both read from a file, for testing that being TEST, and for the main program being the room and player save files. The testing portion of the program will take inputs in the form of words in a text file, then use a python module to send these to the program as keyboard inputs, as if the test was like a user running it.

In the making of this program, initially a system of having each action to be inputted by the user was used, but then was replaced by the menu system as the amount of checking user inputs to see if they are valid or not is significantly less, which reduces the complexity of the code and how often code is being repeated. In the future, a system that allows unit testing to be conducted would be incorporated into this menu system so that unit testing can be conducted rather than user testing.

# Assumptions

The program that will be made to solve this problem description will have these assumptions:

* The program assumes that the program will be run on either a Windows or Unix/Linux system based on the value of ‘sys.platform’.
* The program assumes the availability of the ‘msvcrt’ module when running on Windows, and the ‘termios’ and ‘tty’ modules when running on the Linux/Unix platforms.
* The program assumes that the terminal it is being run in will have read and write capabilities in the directory it is being run from, so that it can interface with JSON files.
* The program assumes that a file called 'TEST' will be available if testing is intended to be done.
* The program assumes that the keyboard module will be available if the 'TEST' file is found.
* The program assumes that the user will be able to input keyboard presses into the terminal.
* The program assumes the availability of certain Python standard library modules such as sys, os, glob, random, and time.

# Flowcharts

## Initial

Figure : 4.1.1 Initial flowchart - Lucidchart.

## A picture containing text, diagram, plan, parallel Description automatically generatedFinal

Figure : 4.2.1 Final flowchart - Lucidchart.

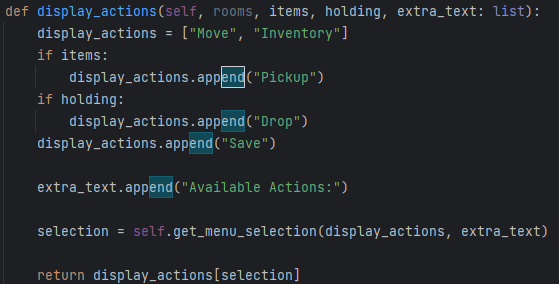
# Design

## Top-Level view

The program will begin with a menu system, explaining how the menu controls work, and offering a way to restore previous progress or create a new save, for both the rooms and the players. If a new player is created, the initial sequence will be triggered, giving the player a short introduction to what has happened to their character. Once that is finished, the main game will start, where the user has many different options for what they would like to do, such as move, use items, shop for items, drop items, and save. Each of these will have their own submenus which allow the user to interface with those options. Also, above the menu currently being shown one to three lines explaining how their usage of an item affected their experience or information about the room they are in, this is only hidden when using an item or transitioning between rooms. Once the main objective has been met, it allows the user to leave the game, or so it seems to them.

## Detailed view

This section will focus on the flows for the main menu functions. For all the main actions, it begins with the available actions, available rooms to move to, items in the current room, current items the player is holding, and extra story elements from used items. This information is passed to the UI, displaying it to the user to choose.

A screenshot of a computer

Description automatically generated with medium confidence

Figure : 5.2.1a display\_actions function. Figure : 5.2.1b example of running code.

The option that the user chooses is returned to main, which is then passed back to the UI, which runs the relevant actions. If the choice was to move, the user then sees whichever rooms are north, south, east, and west, and can choose which one to move to. If the option is to view the inventory, the user is presented with the shop, the ability to use any item they are holding, or to return to the main menu. If the option is to pick up an item, then they are presented with the ability to pick up any item in their current room, if they are holding less than three items. If they choose to drop an item, then they are presented with all their items, and show how much money they get back from dropping this item. Finally, if they choose to save, then it shows that the game is being saved and returns to main.

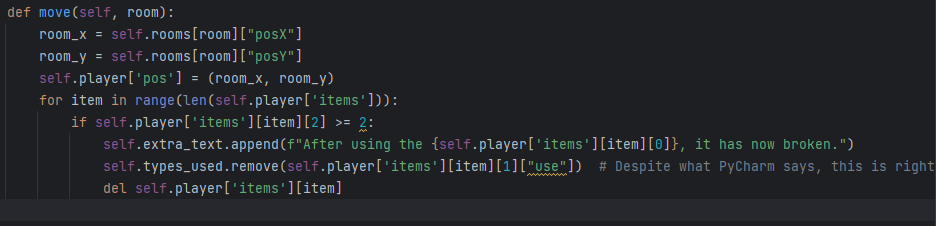
Once this has completed, main checks what was chosen by the user, and runs the relevant actions, such as with move, where it changes the players position, and checks to see if any items are set to break upon entering the next room.

Figure : 5.2.2 data\_store.move() function

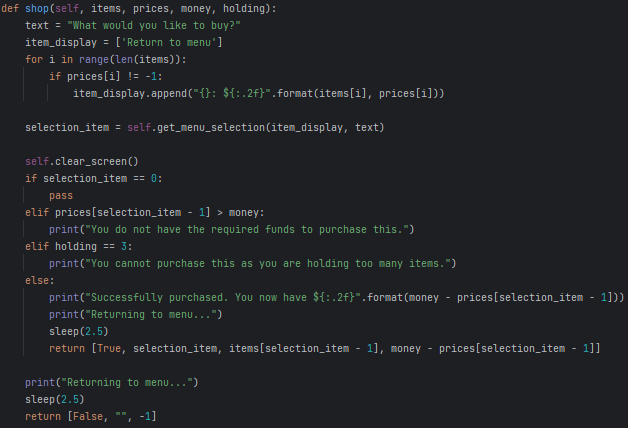
Or like with shop, where it makes the shop display to the user, and allows them to choose something to buy, then returns the results to the data\_store, to be stored.

Figure : 5.2.3 ui.shop() function

The only exception to this is with the save option, where what it does is call the data\_store to save the current state of all of the rooms and the player’s information to file, then exits utilising ‘sys.exit()’, the only warning being the print() call in the UI in the save function mentioned earlier.

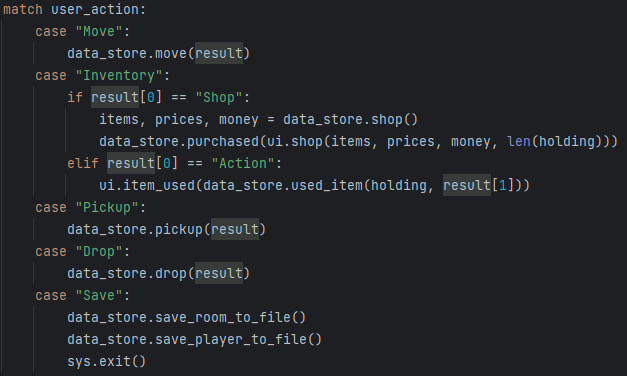


Figure : 5.2.4 extract from main, after user chooses an action.

# Test Plan

For this design, user testing was used, this is because due to everything centring around the menu system. This menu system makes it hard to utilise unit testing as most functions expect output back from the menu system. So instead, user testing will be utilised, which is providing input to the problem as if there was a user hitting the keyboard. This utilises the [keyboard module](https://pypi.org/project/keyboard/), which in one of the functions of the module, called ‘*keyboard.press(key)*’, allows for a key as a string to be passed of which it will simulate that key being pressed on the keyboard. Shown below is how it was implemented in this project:

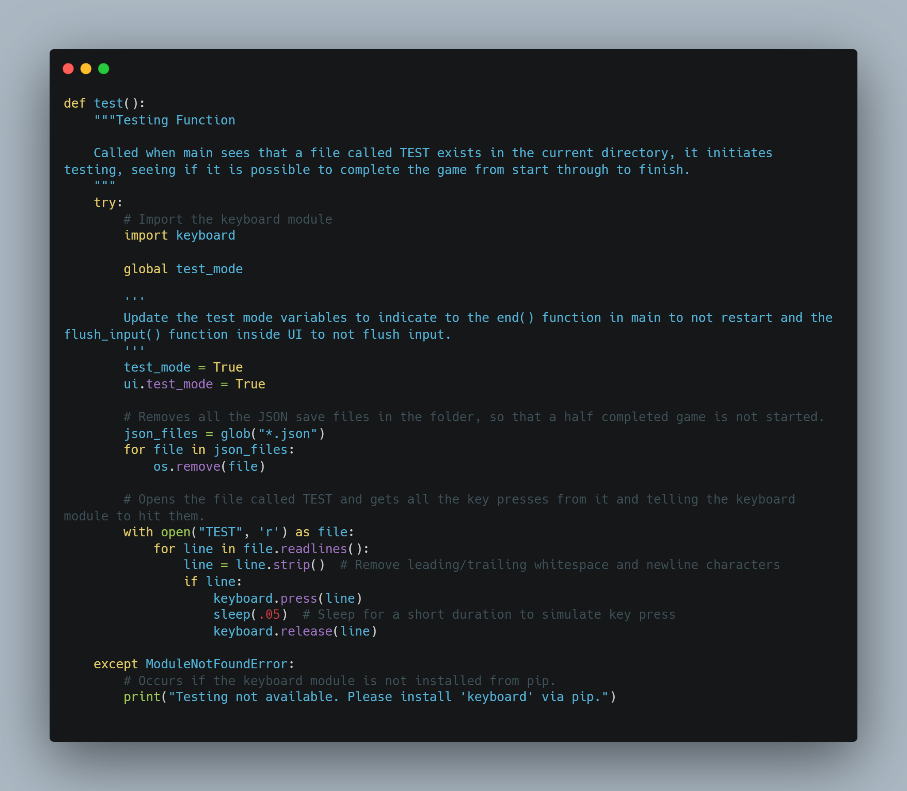
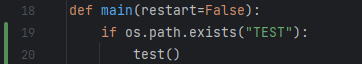


Figure : 6.1 extract from main() showing test() function.

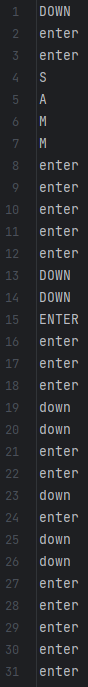
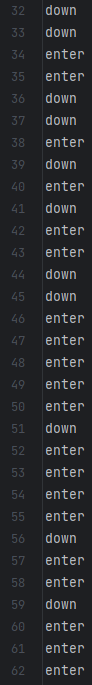


Figure 10: 6.2b lines 32-62 of TEST file.

Figure : 6.2a lines 1-31 of TEST file.

Figure : 6.3 test() function.

This function deletes all save files, then loads all the key presses at once. The call to change ‘*ui.test\_mode’* to *True, which allows for the ‘ui.flush\_input()’* function to not run, which would typically clear out the input buffer, which is a problem for this method of testing as it just fills the input buffer with all the options beforehand.

If the test is successful, it will completely run through the game and finish with it showing ‘*SUCCESS, game tested successfully from start through finish.’*, if the program gets stuck midway through, or errors out, or does not print out this end message, then a problem has occurred during testing.