Sam Heinz

s\_h267@student.usc.edu.au

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

Task 3: Python Project

ICT112: Programming Fundamentals

Table of Contents

[1 Introduction 1](#_Toc137050768)

[2 Problem Description 2](#_Toc137050769)

[3 Assumptions 3](#_Toc137050770)

[4 Design 4](#_Toc137050771)

[4.1 Top-Level 4](#_Toc137050772)

[4.2 Detailed 4](#_Toc137050773)

[5 Test Plan 5](#_Toc137050774)

[6 References 6](#_Toc137050775)

[7 Appendices 7](#_Toc137050776)

Table of Figures

[Figure 1: 4.2.1a display\_actions function. Figure 2: 4.2.1b example of running code. 4](#_Toc137059551)

[Figure 3: 4.2.2 data\_store.move() function 5](file:///C:\Users\wheel\Documents\github\Task-3-ICT112\docs\ICT112%20-%202023%20-%20Task%203.docx#_Toc137059552)

[Figure 4: 4.2.3 ui.shop() function 5](file:///C:\Users\wheel\Documents\github\Task-3-ICT112\docs\ICT112%20-%202023%20-%20Task%203.docx#_Toc137059553)

[Figure 5: 4.2.4 extract from main, after user chooses an action. 5](file:///C:\Users\wheel\Documents\github\Task-3-ICT112\docs\ICT112%20-%202023%20-%20Task%203.docx#_Toc137059554)

[Figure 6: 5.1 extract from main() showing test() function. 6](file:///C:\Users\wheel\Documents\github\Task-3-ICT112\docs\ICT112%20-%202023%20-%20Task%203.docx#_Toc137059555)

[Figure 7: 5.3 test() function. 6](file:///C:\Users\wheel\Documents\github\Task-3-ICT112\docs\ICT112%20-%202023%20-%20Task%203.docx#_Toc137059556)

[Figure 8: 5.2b lines 31-60 of TEST file. 6](file:///C:\Users\wheel\Documents\github\Task-3-ICT112\docs\ICT112%20-%202023%20-%20Task%203.docx#_Toc137059557)

[Figure 9: 5.2a lines 1-30 of TEST file. 6](file:///C:\Users\wheel\Documents\github\Task-3-ICT112\docs\ICT112%20-%202023%20-%20Task%203.docx#_Toc137059558)

# Introduction

In this report will include an overview of

# Problem Description

An outline of what problem is being solved -the scope. Also identify what you are including and what you are not.

Discuss any versions you did and would do in the future.

# 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 program 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.

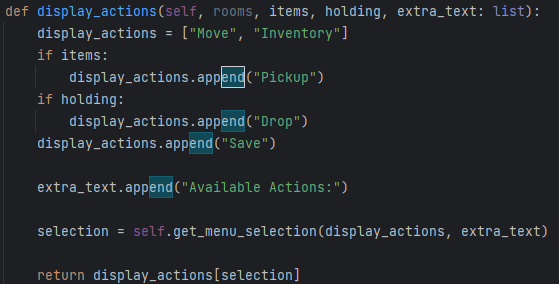
# Design

## 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 : 4.2.1a display\_actions function. Figure : 4.2.1b 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.

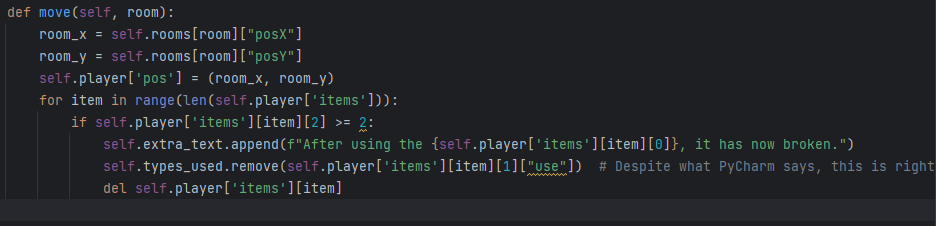
Once this has completed, main checks what was chosen by the user, and runs the relevant actions, such as with move, where it changes the players position, and checks to see if any items are set to break upon entering the next room.

Figure : 4.2.2 data\_store.move() function

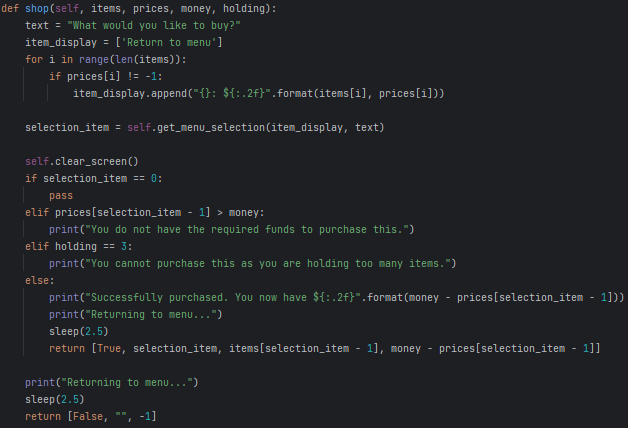
Or like with shop, where it makes the shop display to the user, and allows them to choose something to buy, then returns the results to the data\_store, to be stored.

Figure : 4.2.3 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.

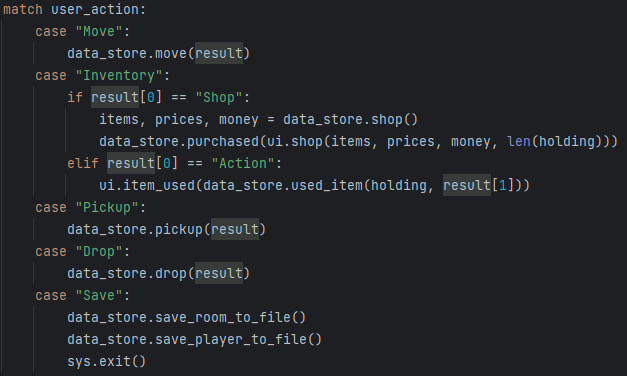


Figure : 4.2.4 extract from main, after user chooses an action.

# 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:

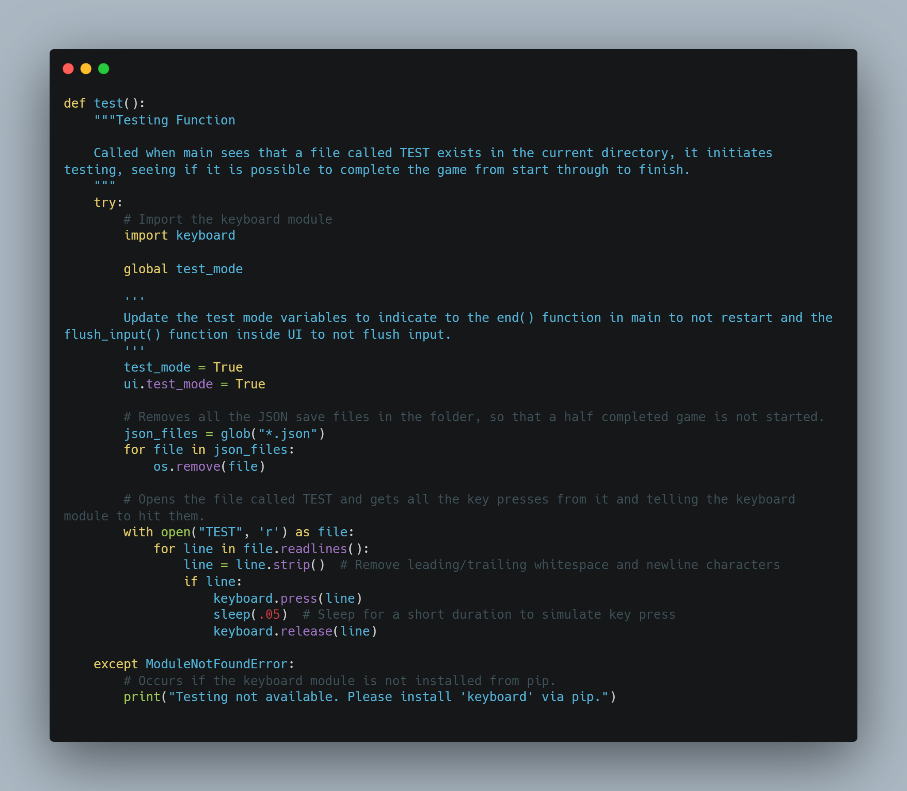
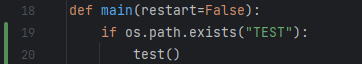


Figure : 5.1 extract from main() showing test() function.

Figure : 5.3 test() function.

A screenshot of a computer screen

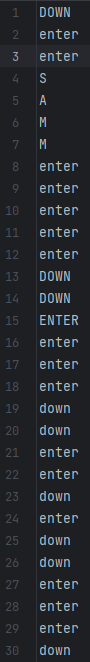
Description automatically generated with low confidence

Figure : 5.2b lines 31-60 of TEST file.

Figure : 5.2a lines 1-30 of TEST file.

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, then there is a problem with it.