Project title: Game Coding and Testing Presentation

Skills needed:

- Apply skills to write applications in a modern programming language
- Test software using existing techniques, to ensure the code works effectively.

Specification

Overview

You are required to manually code and test a small command line game using high-level Python programming language. This game must be coded to resemble a Multi-user Adventure Dungeon (MUD) game popular in the 70s.

You can choose to code the game you designed in assignment 001 or a different one. If, however, you choose a different game please email the MO Debbie Taylor to check the new game idea matches the MUD game specification requirements.

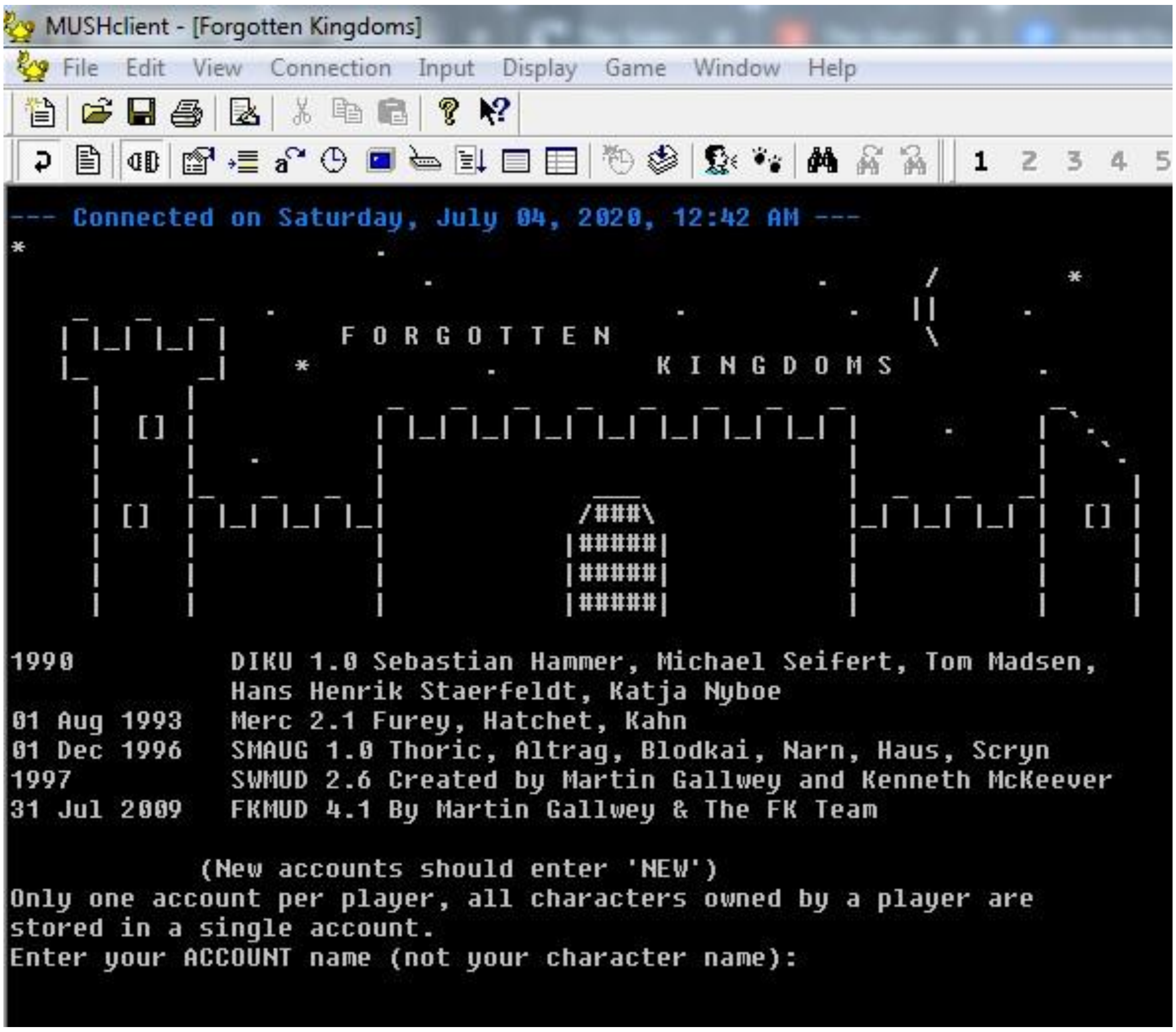
Description

You should code a unique command-line MUD game, using Python coding techniques**.**

You can choose to code the game you designed in assignment 001 or another game. If, however, you choose another game please email the MO Debbie Taylor to check it matches the MUD game specification requirements.

The game should resemble a Multi-User Adventure Dungeon game (also known as MUD games) from the 1970s. This needs to be a multi-user game, not multi-player, so you should have a single person play the game and collect information as they move throughout the game. Then evidence that different single gamers have played by producing a leader board at the end.

There are multiple examples of these types of games available on the internet, however Figure 1 below shows a basic example of one process a gamer would see within a MUD game:


MUSHclient - [Forgotten Kingdoms]
File Edit View Connection Input Display Game Window Help
1 2 3 4 5

```
--- Connected on Saturday, July 04, 2020, 12:42 AM ---

                        F O R G O T T E N
                                            K I N G D O M S

1990           DIKU 1.0 Sebastian Hammer, Michael Seifert, Tom Madsen,
               Hans Henrik Staerfeldt, Katja Nyboe
01 Aug 1993    Merc 2.1 Furey, Hatchet, Kahn
01 Dec 1996    SMAUG 1.0 Thoric, Altrag, Blodkai, Narn, Haus, Scryn
1997           SWMUD 2.6 Created by Martin Gallwey and Kenneth McKeever
31 Jul 2009    FKMUD 4.1 By Martin Gallwey & The FK Team

            (New accounts should enter 'NEW')
Only one account per player, all characters owned by a player are
stored in a single account.
Enter your ACCOUNT name (not your character name):
```

Figure 1 – Example MUD Game, showing a gamer starting a game

Tasks

The following shows the tasks you need to code for this assignment, as well as ideas for extra processes: **Input and Output:**

You will need to code input and output of textual information within your game, via the terminal console – there must not be any graphical user interface created as this is outside the scope of the module.

The input and output information will require you to give options to the gamer to choose from, and then receive input choices from them in return.
Consider consistency of output and input, potential reuse of classes and functions and what data/questions etc. you need to pass to and from the gamer.

Character creation:

Whenever a new game begins the gamer should be prompted for a series of character creation options. This will require you to manually code a class or function that allows the gamer to customize the character at the start, with multiple options for the gamer to choose from.

You will need to be able to access these option choices throughout the game. For example, if a gamer chooses a name at the start of the game you must code the game to use this multiple times throughout the game, not just at the start.

Interaction:

Manually code classes and functions that allow the gamer to interact with your uniquely created game world. This could be moving around the game world itself, accessing and receiving information about locations, weapons, items found, health points gained or lost, etc.

For example, if a gamer enters a location, they must be able to receive output information about that location and any potential items available to pick up. They should then be given options on how to pick up the items if they choose to.

Saving/Loading Progress:

Manually code classes or functions that allow you to save and load progress within the game. The progress information must be saved to a text file, CSV or Json file.

As your game progresses you should store the players location, items collected and general progress. This will enable the gamer to exit the game at any point and reload later at the saved point.

You will also need to save the gamer's score at the end of the game (win or lose) and output this information to a saved leader board, highest score at the top, lowest at the bottom. The leader board should be available to view on both the console terminal and via a text file, CSV or Json file.

Testing:

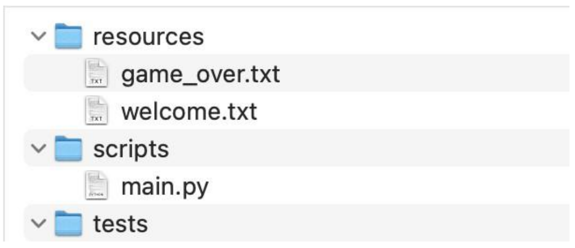
You are expected to perform and discuss white box system unit testing using Python unittest, along with think aloud user testing.

Ensure both are recorded on a valid test plan. Blank test plans will be uploaded to the week 9 Software Testing folder on Blackboard.

These test plans should then be included and discussed during the video, to evidence completion of this learning objective.

Project File Structure

I stored scripts, resources and tests in separate folders. You should also store your saved text pictures in separate classes or txt. files and use functions to load them, not just save them in the main program. See below example:



```
resources
    game_over.txt
    welcome.txt
scripts
    main.py
tests
```

Additional Areas to Consider for Game Coding:

Extra points are available for coding additional functionality specific to your game. A few **examples** given below should prompt you to think about areas to develop, so that you have a unique piece of work. You will need to justify these decisions during your video **Visualizations:**

Consider creating textual pictures and then loading the pictures at relevant points within the game world. For example, if the gamer sees a castle appear before them, you could call the picture class or function and output the visual textual picture of the castle to the console terminal.

You might also want to consider different colors for input and output text.

Equipment:

You might want to consider customizing what your character is wearing or using at different points within the game. What equipment can be found within the game and where? How can specific equipment collection impact the potential ability to add or deduct health points, etc.?

Inventory:

Do you want to allow users to collect specific items or objects that can increase health or power scores, etc.? You might want to design a system to store these items so that players can see what they have collected and interact with them. You may wish to only deal with unique items; however, you may want to consider how to handle multiples of the same item.