Module: CMP-7000A – Applications Programming
Assignment: R002–Game Coding and Testing Presentation

Set by: Debbie Taylor, Debbie.taylor@uea.ac.uk
Date set: 14 July 2023
Value: 70%

Date due: 08 August 2023 15:00
Returned by: 05 September 2023
Submission: Blackboard
Checked by: Eleanor Leist, e.leist@uea.ac.uk

Learning outcomes

- Apply skills to write applications in a modern programming language
- Test software using existing techniques, to ensure your 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. ***This R002 reassessment game must be a different scenario to the one you coded in assignment 002, during the Autumn 2022-2023 semester, and you must not reuse any code from your original submission.***

To evidence the learning objectives, you must produce working Python game code (entirely coded by yourself with no content from the internet, other sources, a previous course, or your 002 assignment). You also need to record a maximum 10-minute MP4 (max 720P) video that shows your game working from both a gamer and code perspective. The video also needs to evidence system unit testing and think aloud user testing, by showing and discussing completed test plans.

Description

You should code a new R002 unique command-line MUD game, using Python coding techniques**. You must not use any code found on the internet (including GitHub, YouTube, ChatGPT, etc.), or any other external sources, including any previous courses you have taken, or from assignment 002; you must code this new unique game entirely by yourself for this R002 reassessment.**

If you are not sure if your game is fully compliant with MUD game specification requirements, please email the MO Debbie Taylor on Debbie.taylor@uea.ac.uk to check.

The game should resemble a Multi-User adventure Dungeon game (also known as MUD game). 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. You should 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 should see within a MUD game:

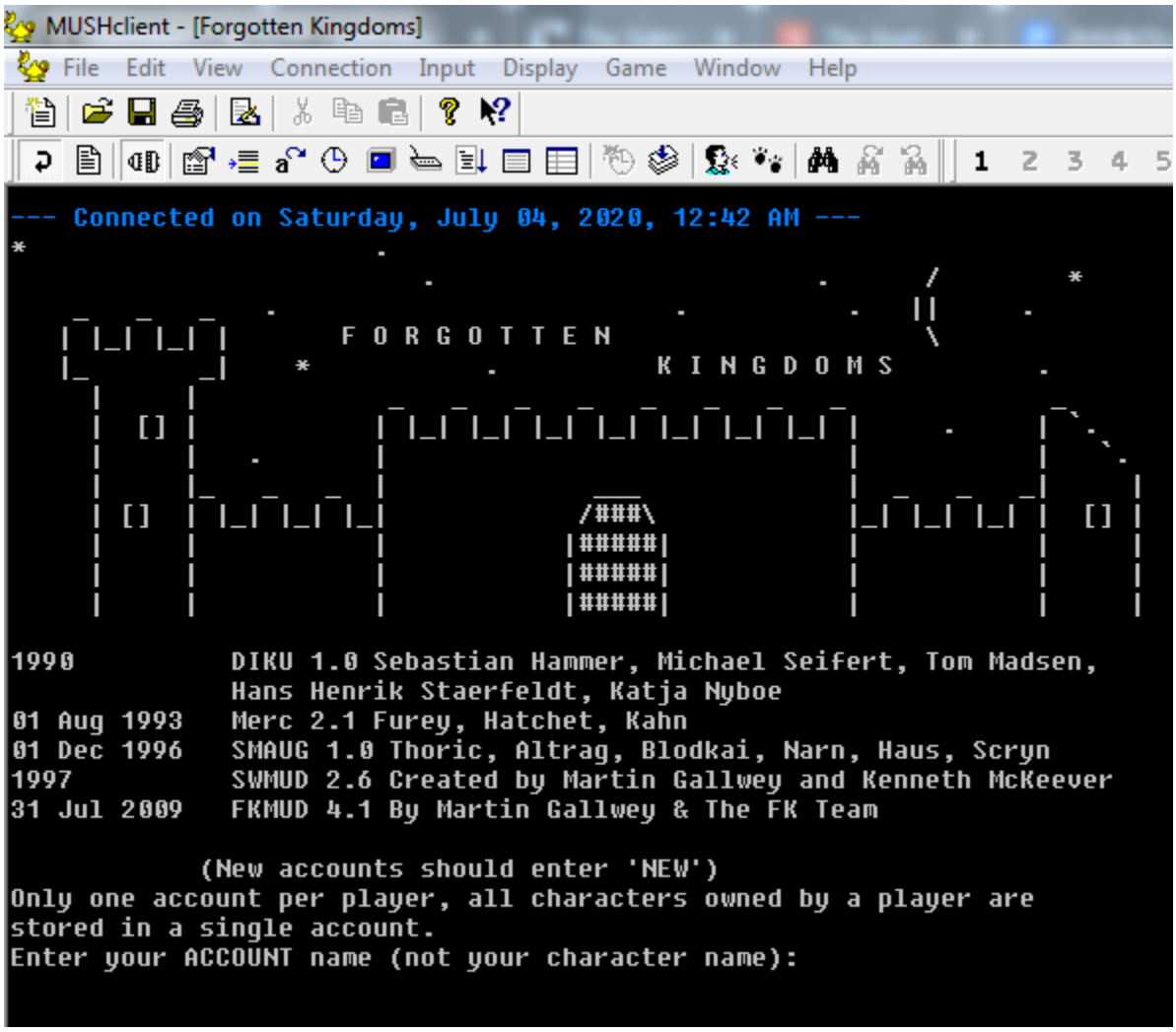


Figure 1 – Example MUD Game, showing a gamer starting a game

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 customise 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 item/s if they choose to do so.

Saving/loading progress:
Manually code classes or function that allow you to save and load progress during 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, with the highest score at the top, and 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.

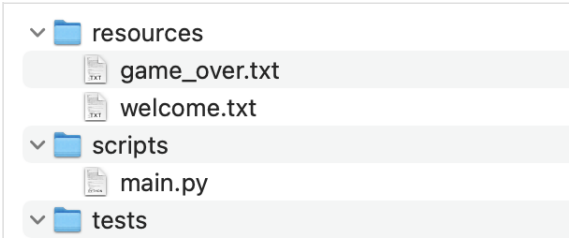
Testing:
You are expected to perform and discuss white box system unit testing using Python unittest, along with think aloud user testing.

Ensure both are recorded on a valid test plan. Blank test plans are available in 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

You need to store 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:



Additional areas to consider for your game coding:

Extra marks are available for coding additional functionality specific to your game. A few example ideas given below should prompt you to think about areas to develop, as that you produce a unique piece of work. You will need to justify these decisions during your video.

Visualisations:

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 a visual textual picture of the castle to the console terminal.

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

Equipment:

You might want to consider customising 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 items 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.

Help option:

What happens if a gamer gets stuck and doesn't know what to input or what options are available for them to select?

To evidence the above R002 reassessment code and testing processes, you must record a maximum 10-minute MP4 (max 720P) working video. This must show the code working from both a gamer user perspective and the code itself.

You must also show the testing test plans and discuss both python system unit testing and user think aloud testing.

If the video exceeds 10 minutes any information discussed after this time will not be marked. You **must not alter the playback speed of the video to remain under the 10-minute limit.**

You must not use any code found on the internet (including GitHub, YouTube, ChatGPT, etc.), from a previous course you have taken, or your 002

assignment; you must code this game entirely by yourself specifically for this R002 reassessment, to achieve the learning objectives.

R

elationship to formative assessment

The skills needed to complete this assignment relate to the lectures and labs from week 3 onwards. You can also use the feedback provided for assignment 002.

Deliverables

You must upload a .zip file to the R002 submission point on Blackboard, containing all the source files for your python code, a copy of your PowerPoint slides and a maximum 10-minute single MP4 video (max 720P) file, evidencing the code working from a gamer and code perspective, along with test plans for system unit testing and think aloud user testing.

If the video exceeds 10 minutes any information after this will not be marked. You must not alter the playback speed of the video to remain under the 10-minute limit.

The zip file must use the following naming format of studentID.zip e.g.*10000000.zip.*

Please ensure you upload a working video, as you will not be asked to provide another copy if the uploaded version does not play when being marked.

Resources

Lecture slides and lab sessions from week 3 onwards, are highly relevant to this work. You can also use the feedback provided for assignment 002.

You can use any video editing software you choose but the upload must include a single video file for the entire presentation from start to finish, not a recording per slide. You can show a watermark across the video, but only if this does not stop the markers being able to view your working system and code.

If you have questions, please email the MO Debbie Taylor on Debbie.taylor@uea.ac.uk.

Plagiarism, collusion, and contract cheating

The University takes academic integrity very seriously. You must not commit plagiarism, collusion, or contract cheating in your submitted work. Our Policy on Plagiarism, Collusion, and Contract Cheating explains:

- what is meant by the terms ‘plagiarism’, ‘collusion’, and ‘contract cheating’
- how to avoid plagiarism, collusion, and contract cheating
- using a proof reader
- what will happen if we suspect that you have breached the policy.

It is essential that you read this policy and you undertake (or refresh your memory of) our school’s training on this. You can find the policy and related guidance here:

https://my.uea.ac.uk/departments/learning-and-teaching/students/academic-cycle/regulations-and-discipline/plagiarism-awareness

The policy allows us to make some rules specific to this assessment. Note that:

In this assessment, working with others is *not* permitted. All aspects of your submission, including but not limited to: research, design, development and writing, must be your own work according to your own understanding of topics. Please pay careful attention to the definitions of contract cheating, plagiarism and collusion in the policy and ask your module organiser if you are unsure about anything.

Marking scheme

Professional programmers are required to produce maintainable code that is easy for *others* to understand, easy to debug when (rather than "if") bug reports are received and easy to extend, therefore this is expected from you too.

Marks will be awarded according to the proportion of the specifications successfully implemented, programming style (indentation, good choice of identifiers, commenting etc.), and appropriate use of programming constructs.

See next page.

Marking scheme: R002 - Game Coding and Testing Presentation

Marker Name	Student Number

Marking Details	Mark %	Marking Comments
Input and Output Input easy to use = 3 Output clear and understandable = 3 Consistency = 3	9%	
Character Creation Each option (max 3 options) = 2 Options called throughout the game = 4	10%	
Interaction Easy to figure out what to do = 3 Interact with locations and items etc. = 3 Find/pick up items throughout game = 3 Exception handling = 6	15%	
Saving/Loading Save to text/CSV/Json file = 4 Save game at any point and save to file = 4 Reload game from saved point = 3 Save and load a multi-user leader board = 4	15%	
Additional Features Each feature (max 4) = 4	16%	
Testing Evidence System unittest = 5 User testing think aloud = 5	10%	
Code Quality Naming of components = 1 Effective functions = 3 Effective classes/class hierarchies = 4 Effective loops = 3 Effective use of data types = 1 Effective code layout = 1 File structure = 4 Comments = 2	19%	
Video Quality Content = 2 Delivery and sound = 2 Structure = 2	6%	

Total Score =