Candidate Number : 183773

My goal was to design a game that takes inspiration from classic dragon quest and final fantasy games. A protagonist making his/her way through a world with multiple levels. From a story perspective, the protagonist works his/her way through each town helping them defeat their local enemy boss in different places. Once defeated the user can move onto the next one. This all provides experience and gold for the user to buy items and level up to be strong enough to defeat the final boss on the top of the mountain. As of yet I have not had enough time to implement the storyline but I have a good idea on how I would do so.

In the game the user can roam through a world with 29 maps. Not all unique at this point, but once I implement the storyline the similarities will disappear. The world is multi-leveled, i.e. there are maps within maps. You can travel on the world map and then enter a town and then enter a shop or a house. The user can also purchase items, these include weapons, shields, armour and healing potions. All of the above help improve the users battle statistics. Battle is the main component of the game. The program generates random enemies, with strength based on location, and the user is given the option to fight, use an item or run away. Outcome is based on user statistics and decision. This, along with adventure, is the core of the game. Since this is not a small game and allows for progression, by visiting the bed at home or in a hotel the user saves the game to a txt file which can be easily restored (by providing the same name and choosing load at startup) at a later date.

A class diagram is available within the code files. (class diagram.png)

I did not use the provided startup code. I knew the fundamental part of the game was moving around, followed by battle. To encourage adventure I needed a sufficiently large world. I created a small tool to create matrices, which I used to create the matrix for each map.

* I created the map class to create and conveniently store all the world map data.
* I created the location class to have a dynamic object moving through the maps. As a user would.
* I then created the character class, this was just to store all character statistics in one place.
* I created the player class to store the bulk of data, as the whole game links back to the player. I connected a location object to the player class, to have “the player” dynamically moving through the world.
* I created the enemy class for all my enemy bots to inherit from. This along with all the enemy sub classes allow for the easy creation of new enemies.
* With the main character and enemies, I now created the fight class. Each fight object is an event where some enemy battles some player. This also takes care of post-battle details, such as if a player wins: gaining EXP, looting gold and items and if a player loses: resting to the last save location.
* I then created items. These would be stored within the users items dictionary. Some items (pots) can be used while in battle, while others (weapons, armour, shields) sit in the bag but if initialised correctly using the add\_item method will boost user stats.
* I then created the shop class. These allow me to create different shops with different products and unique keys, so I can place them as I desire on the maps. When a player interacts with one of these, items can be “purchased”.
* I then created the bed class. This simple class allows the user to reset hit points and save the game. This links to a function within the player class that writes all vital game data to a txt file.
* I created the game class to make it easier to start new games and load up old ones. New ones being set with standard user attributes and old ones being loaded up via a parsing method on the imported txt file.

I encountered a few problems with this coursework. The first and most difficult being linking up the maps to allow for jumping maps. This itself was fine but the position where the user would enter the new map was always the same. To fix this I created a new attribute; a list of doors. The algorithm searches for a matching key in the list of doors for the new map and then sets entry for the user at one space to the right.

My algorithm for opening save game data was also interesting. Since the dictionary of items was stored as a string, I decided to instead store item keys and when opening, clean the data, search for key in all keys and create a new list of items. I also stored dictionary values in a separate list just to make the cleaning process easier and rejoined them to the keys later.

I created a table for system level testing which contains all the fundamental features of the game. I had a few issues but I fixed them all and then re-did each test. At this point they have all passed. I also used unit testing. I created a random key generator to input random movements. Along with this I pre-set all interactions to “no”. The program moved around quickly and terminated with an index error next to a specific door multiple times. I checked that map and found that the door was incorrectly labeled. After fixing this I re-ran the program for 5 minutes and terminated when no error was found. The excel sheet for system level testing and the map testing files are also included.