Project Development Reflection Document

Zecan Liu 25348914

FIT1048 Assignment 4

Motivation and Goal

From the beginning, the goal was to create a program that is effective in setting up, efficient and flexible in running. The game should be fun to play - unpredictable and offers the player full freedom.

The Design

This project practised the object-oriented programming approach. A total of seven class objects were created, together with seven Enum classes, one “tool chest” function collection, and the main game-play execution and function collection file. A brief rundown of the program design highlights is as follows - note that there are too many design details to be all listed, which could be found from comments in the actual code.

- Player class hosts many game-play functions, e.g. functions for picking up and dropping items. This is the “main object” of the game.
- The item class hosts attributes that define its efficacy on the player. The definition of all items are stored in a text file - strictly following a predefined format. The monster and location classes share the same design approach as that of the item class.
- A key design feature is that, for item, monster and location, only “model” objects are created according to the definition file. These created objects are saved in a dedicated vector as a static class attribute (using “this” pointer). This effectively clears the variable space for the program, and reduces the number of class objects generated at the same time. For example, a monster is only activated when needed.
- As a one of the typical OOD approach, the game mechanism utilises an active location as the platform, and the active player as the host. The player object initiates the game functions and chooses a next path. As a location object becomes active, the program will scan through its attribute to determine and activate the “active” item and monster or NPC.

Program Outcome and Highlights

With more than hundreds of hours spent on the project, the outcome is satisfactory. All initial design goals were met and the planned game mechanics were successfully realised.

Just some highlights of the game are as follows:

- The entire game world initialisation is randomised. Based on the player elected ame difficulty, a certain number of monsters and items are randomly allocated to locations - there will never be an identical game.
- The player purchases the starting initial gears with the given fund in the game store. Full freedom there. For each item, there could be more than one save in a single bag slot.
- After each interaction with the player, the monster resets - no identical monster will be encountered twice, even of the same kind.
- Player has the freedom to take any one or all items available in the active location. The Player could drop items from the bag any time, and decide, if not all, how many to drop.
- The game is smart. If a location has been examined, it can not be examined again. If a location monster was not defeated, it will remain there however with attributes reseted - unpredictable!
- According to the player selected game difficulty, the dragon may roam around the caves. Player needs to keep track of the dragon location.
- Additional functionalities and items that were not specified by requirements!
  - A player growth model - for each monster defeated, the player increases the attack by one.
  - New booster items that affect the player attribute but not taking up the bag space.
  - These are hidden in locations, and players need to search for them.
- NPC interaction with multiple potential outcomes.

Challenges and solutions

The main challenges were mostly associated with the core design concepts of object activation. The original concept aiming for efficiency (not having too many objects) made the realisation process really complicated. A lot of time spent in perfecting the game mechanisms and debugging. Some highlights are (there are a lot more to talk about):

- Assigning items to monsters - how to make sure all artefacts are available for the player to collect, and correctly carried. Solution: use of global variables for tracking the available attribute and available monster left.
- Taking and dropping algorithm - included the concept of number of items per slot (e.g. 10 gold per slot). Challenge was how to precisely fill the bag and drop the item by a certain amount. Solution: lots of practice of vector operators and copy by reference concept.
- -Avoid repeated examination at a location - Solution: location class boolean attribute.
- Realisation of Dragon and NPC classes - Solution: Inheritance, abstract class and polymorphism.

Future improvements

There are many aspects of the current program that could be improved.

- The better practice of class inheritance and polymorphism. This is particularly for the item class. There is not enough time to implement inheritance into item class. As a significant portion of the class build and game concept were realised even before the learning on inheritance - making it too much to alter later. However it could really use inheritance given the increased variety of items, the definition file because it is lengthy, complicated and hard to understand.
- Use of advanced data structures! For example, use of structure could significantly tidy up the space and make the program even better organised.
- Improved class initiation mechanism. This could also be somewhat improved by better practice of inheritance. Nevertheless, the definition file for each class could be improved to a way that the data are better structured and easier to understand. The program itself should be developed further to have capability of identifying wrong definition files and reporting errors. The current version would crash miserably if the definition files are wrong.

Biggest take away

The most important thing I learnt from this project for programming is that the OOD approach and a thorough planning of the program structure is everything (essentially everything included in Assignment 2). It is necessary to spend lots of time in planning and structuring the program at the start as it will be a nightmare to correct when the program gets enormous.

Appendix

A list of extra functionality attempted:

- Select a skill level - game difficulty
- A scoring system - tracking moves, monsters and updating attribute
- Dragon can move given the skill level selected
- There are three maps created.
- A game store at the start
- Roaming NPCs