CS 246: ASSIGNMENT 5

CC3K Plan of Attack

Group Members:

- Rut Patel (20647619, rd4patel)
- Vibhanshu Bhardwaj (20607705, v9bhardw)

Project Breakdown

1. Read the project thoroughly and note down all required classes and their purpose. Complete the design of the UML.
   - Estimate: 2h on 23rd March

2. Write header files for all classes (as per the designed UML).
   - Estimate: 2.5h on 25th March

3. Implement basic functionalities of the following classes: GameObject, Floor, Chamber, Character, Player, Enemy, Item, Treasure and Potion. Implementing basic functionalities of these functions must be sufficient to display the game board (Floor).
   - Implement Command interpreter in GameObject Class
   - Read default layout and create 5 chambers
   - Generate Floor with player/Enemies/Potions/Gold/Stairs (all with basic functionalities) randomly placed in the chambers
   - Implement createFloor and printFloor methods.
   - Estimate: 6h on 26th March

4. Implement the loop (player.alive()) where every next move will generate new random position for every character on the floor (movement is in accordance with the guidelines).
    - Estimate: 1.5h on 27th March

5. Implement Character Class and all of its subclasses.
    - Implement attack method (Visitor design pattern)
    - Implement Player class, Enemy class and each of their derived classes completely.
    - Estimate: 2.5h on 28th March

6. Implement Item class and all of its subclasses
    - Implement Potion class, Treasure class and all of it's subclasses.
    - Estimate: 2.5h on 28th March

7. Testing
    - Estimate: 3h, on 29th March

Work Distribution:

- *Rut:* Implement main, Floor class, GameObject class, Chamber class, Potion class, Treasure class and implement methods in other classes if required.

- *Vibhanshu:* Implement Character class, Player class, Enemy class, all subclasses of Player and Enemy class, Item class, Potion class and Treasure class.
  (*Work will be completed with high collaboration while implementing every single class to ensure the smooth progress of the project*).

Answers to Questions

Question: *How could you design your system so that each race could be easily generated? Additionally, how difficult does such a solution make adding additional races?*

Answer:

(i) We will use sort of Factory Method calls to generate Enemies and Player while initializing a Floor. We've created several sub-classes for each race which inherits from its abstract Parent class.

(ii) Player and Enemy Are Abstract classes which inherits from an Abstract Super class named *Character* and every Player character and Enemy character inherits from Player class and Enemy Class which are abstract. This kind of connection between the 3 abstract classes (Character, Player and Enemy class) allows us to add additional class easily.

Question: *How does your system handle generating different enemies? Is it different from how you generate the player character? Why or why not?*

Answer: As per given guidelines, every enemy has a specific probability of getting spawned. We will use the random integer generation which generates a random number between 1 to 18 inclusive. Enemy is spawned depending on its probability and the length of interval the generated random number 'r' is in.

For instance:

- Human: [1, 4]
- Dwarf: [5, 7]
- Halfling: [8, 12]
- Elf: [13, 14]
- Orc: [15, 16]
- Merchant: [17, 18]

i.e. If 'r' is 7, Dwarf is spawned on the floor.

Dragon is spawned near a Dragon hoard only.

Yes, generating player character is different from that of enemy. The client/user selects the race of Player and that Player character is then placed on to the floor (no randomness is associated with the player character selection). After the selection, it will be assigned to a tile in any chamber just like Enemy.

Question: *How could you implement the various abilities for the enemy characters? Do you use the same techniques as for the player character races? Explain.*

Answer: All Enemy and Player races will use the default abstract character implementation of defend or attack unless it is overridden by that specific race class of attacker/defender. Therefore, all abilities for both Enemy and Player Character will be implemented using the *Visitor Design Pattern.*

Question: *The Decorator and Strategy* [8] *patterns are possible candidates to model the effects of potions, so that we do not need to explicitly track which potions the player character has consumed on any particular floor. In your opinion, which pattern would work better? Explain in detail, by weighing the advantages/disadvantages of the two patterns.*

Answer: The Decorator patter would work better in our opinion. This would allow us to augment existing functionality as well as it allows additional functionality at run time. Whereas, Strategy patter would only allow us to change the implementation at runtime. Here, Player object will be decorated by Potions which player chooses. This decorated Player object will be deleted when the floor changes and the undecorated player object will be passed on to another floor.

Question: *How could you generate items so that the generation of Treasure and Potions reuses as much code as possible? That is, how would you structure your system so that the generation of a potion and then generation of treasure does not duplicate code?*

Answer: We will generate items using the *Template Method* which acts as a factory which allows us to reuse the same code to produce objects of different types (here, potions and Treasures).
