CC3k Plan of Attack

Yuxin Zhu y322zhu
Tianhao Wang t326wang
Jiayang Fan j52fan

Timeline Breakdown

Week1: July. 10th – July. 14th

In the first week, our goal was to extract all the important information from the project guideline and contemplate the classes as well as their relationships that are necessary for this project.

From **July. 10th to July 12th**, we read the requirements of the project carefully, highlighted the key information and made some notes to create the framework of the project.
We decided to decompose the relatively complicate project into a few components, and we define the main interface as **Floor**.

1. The display of the game board (Floor, chambers, wall, etc), by *Observer Pattern*.
2. The generation of Potions & Treasures, by *Factory Pattern*
3. The generation of Players & Enemies, by *Factory Pattern*
4. The utility of Potions & Gold, by *Strategy Pattern*.
5. The action of characters (Attack & Move), by *Visitor pattern.*
6. Special properties and functions (dragons, merchants, …)

The basic plan for UML started on July 13th. We discussed together and add the main classes to it. The framework of UML is expected to be finished by **July 14th**.

By **July 14th**, we will divide the questions to each group member, but will discussed together to ensure there are no contradictions between the designs for different components.

Week2: July. 17th – July. 21ed

We aim to complete all of the .h files of classes and subclasses from **July 15th to July 16th** following our initial UML and plan. If there are unexpected difficulties of implementation or excellent new ideas during the implementation, we may update our UML after discussion.

In terms of the main part of implementation (.cc files), we will split the work evenly and according to the strengths. The implementation will start on July 18th – July 22ed.

1. Floor & Main function (Framework) is expected to be completed by **July 17th.**
2. Items & Basic Characters classes is expected to be completed by **July 18th**.
3. Specific Items & Characters classes is expected to be completed by **July 19th**.
4. Actions is expected to be completed by **July 20th**.
5. Special properties & functions is expected to be completed by **July 21th**.
6. Testing & bonus features will be focused after main implementation.

We will modify the UML whenever there is a change in our design of classes. This is the rough outline of our plan, and we will try to stick to it.

Questions

Question (1). How could your design your system so that each race could be easily generated? Additionally, how difficult does such a solution make adding additional races?

Answer: In order to allow each race to be generated easily, we decided to use *factory pattern*. We are going to create a "Player" abstract class, and all of the specific race classes are subclasses of it. "Player" class will specify all of the common attributes of the races such as HP, Atk, Def and the number of gold. Also "Player" class will create virtual methods such as attack and move, and the subclasses will modify/override them and add their own ethnicity features. So the generation of races is considerably easy, and when the user want to start the game, the constructor operation of races will be called depending on the decision of user. Our main interface, Floor class, will be the factory of player.

In term of adding additional races, all we need to do is to make a new subclass of "Player" and add a new option for the generating method in Floor class.

Question (2). How does your system handle generating different enemies? Is it different from how you generate the player character? Why or why not?

Answer: Very similar to "Player" class, we will use *factory pattern* and implement an "Enemy" abstract class, which is the parent class of all of the specific enemy types such as trolls, merchants, etc. The only difference is that the factory of "Enemy" is the 5 merchants in the floor so that it would be easier to control the behavior of enemies and follow other restrictions. When main function notifies the chambers to create an Enemy, random numbers will be used to ensure the possibilities of occurrence of enemies will be equal.

Question (3). 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 of the enemies will have Attack & beAtkBy method, inherited and override from "Character" superclass. We need to make sure that the combination of different defender types and different attacker types will have different effect based on the information and features of types. Therefore, we decided to employ *visitor pattern* and make use of double dispatching. In terms of "Player" types, the method is seems to be the same now.

Question (4). The *Decorator* and *Strategy 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: Decorator creates a decorated object with additional information (adding specific potion effects) to specific object, leaving others unchanged. However, the decorated object was constructed as a new object, rather than modifying the original object. In this case, **removing** the effect will be convoluted. Strategy pattern defines all behaviors and dispatch at runtime.

Question (5). 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 use a class Chamber as the factories of generating Treasures and Potions. When we want to create an item, the main function will let Chamber to call the constructing function. Currently, we decide not to build a common superclass of treasure and potion, since we can't find these two items much in common. So, potion and treasure will both inherit Object class together with character, since all of them will have a position on the floor.