Plan of attack:

Week 1 Nov 20-27

During the first week, we will be discussing the general direction for the first two days and try to finish all the coding by the end of the week. We have initially decided that the work will be split into two parts, coding and documentation. Documentation will be work first for the purposes of fully understand what to do.

In the first week, we will be mainly targeting the comprehension of the game for the first 3 days, which most of the time will be spent on UML; therefore, the first completion of UML is expected by Wednesday Nov 25th. And we will be completing the coding of the game for the rest of the week.

We are going to breaking the coding into three parts, which are characters (heroes and enemies), items (potions and treasures), and the game board (including the controller). The first two parts will be expected to complete in two days whereas the game board part needs more time to complete due to the complexity of the controller part. So it is expected to spend our weekend on implementing the controller. And it is expected to be completed on Nov 29th.

With the given resources, i.e. the entire design pattern we have learned in class and assignment, and the complete understanding of how should the code be implemented and interact each other, we should be able to finish the project by the end of the first week. Design and structure of the game may change as we progress; consequently, the UML will be kept track within the week of the progress. However, the UML that is created before coding will be kept for the first submission of the documentation.

Week 2 Nov 23-Dec 5

Since all the code has been fully implemented, we are just going to focus on debugging and adding additional implementation in the second week. In the first two days of the Week 2, we will be mainly focus on creating tests and debugging the code. If there is still some time left, we will try to add bonus implementation to our game.

Summary:

All the code required by CC3K, will be completed on Nov 29. Debugging will happen shortly after we finish coding. The time left will be spent on Downloadable Content.

Questions

**Player Character**

**Question 1** 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?

A method of generating the race will be implemented in Controller class. It asks for an input form user and creates the corresponding object to the private field of controller. Each races is inherited from Hero class (abstract) which will is inherited from an abstract class Character. Since all the common field and methods of characters in cc3k are implemented in Hero and Character class, all the repetition of coding is avoided. Thus, it is not difficult to add an additional race. All you need to do is to add another class that is inherited from Hero and add this additional choice to the method that generates the race in Controller class.

**Enemies**

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

In Game class (a class that is used to create objects used for the game), a method will be created for generating enemies. The enemy generated by the method will be stored in a vector in Game class for the further uses. All the enemy character classes will be inherited from Enemy class, which is inherited from Character. The reason we inherited both enemy class and hero class to character class is that we would like to keep track and be able to maintain the two main classes of characters; Moreover, both heroes and enemies have relatively same properties, i.e. HP, DEF, ATK and names, we would be really like to reuse all the getters and setters for the two inherited classes, which could save a huge amount of spaces. Methods and private fields can be added based on their functionalities. Hero characters and enemy characters are designed to be inherited from Hero class and Enemy class respectively.

**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.

The data of HP/ATK/DEF will be kept in Character class. For enemy characters, their ability will be mainly stored in Enemy class, except for dragon and merchant. For those two enemy parties, special fields may be created for them. While all the common method such as combat is implemented in enemy class, special methods and fields are expected to be added to dragon and merchant classes due to their special functionality. A similar technique is used for player character races for the purpose of maintainability.

**Potions**

**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.

The Strategy pattern encapsulates various algorithms that can be used on objects. This design pattern allows user to be able to change the implementation of something used at runtime. And the Decorator pattern allows user to be able to add functionality to an object at run time.

For the assignment we are working on right now, decorator would work better since there are only two types of potions that used in the game (positive or negative). By using decorator pattern, it is more flexible to build the potion. However if you are planning to add more items or more complicated functionality to the game, strategy pattern will be a better pattern to use for the purpose of readability and maintainability.

**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 do not duplicate code?

Similar to the generation of enemies, a template pattern (vector of Treasure pointer and Potion pointer) is used for the generation of Treasure and Potions. To minimize the duplication code of generation of poison and treasure, we want to make Potion and Treasure as separate classes that both inherited from the abstract class item. A ‘generateItems’ method will be implemented in Game. This way, part of the duplicate code happened in notifying the game is limited, and code will be well organized.