Final Design outline

Through out the days of working on this project we took upon many different approaches towards completing this project, first with taking multiple approaches towards creating a board until finally agreeing upon one. We began by creating a class that contains the entire layout of the floor, whether it may be a wall or just a blank area. Followed by that we created a class that controlled each individual cell, which contained three subclasses for; non passable cells such as walls; passages such as hallways; and finally just regular tiles that were contained within a chamber.

Then came the entities that were contained within the map, we split these up into character classes and the in game items. First we created a general class for all character on the board and split that up into two subclasses of Players and Enemies. With each one of those came a decorator class that gave you a choice of different player classes and also gave a variety of enemies. We had a similar approach towards items, with a main class for general items and two subclasses for treasure and potions. The player and the enemy classes were link together as well in order to allow interaction between the two on the game board.

The testing factor that came in was just as planned, with each addition test for any type of bugs that may pop up and assure all the required specifications are on point. Then after completing the majority of the project we ran a plethora of tests to see if any point of the board collapses and if the game ending objectives can be completed.

Questions:

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

We'll use a base class called Hero, and a subclass for each race. That way each race can be easily generated and the rest of the classes don't need to know what race my character is. This also makes it easy to add additional classes that need to know about the player character, because you can just refer to the Hero class.

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

We'll have a base class called Enemy, and a subclass for each type of enemy - just like the player character structure above. Any code that needs to apply to any type of enemy can refer to Enemy, and anything that depends on individual enemy types can use the subclasses. This is similar to the way we generate the player character, because we need similar functionality from both - some classes will want to refer to any type of race/enemy and some classes will want to refer to specific types of race/enemies.

How could you implement special abilities for different enemies. For example, gold stealing for goblins, health regeneration for trolls, health stealing for vampires, etc.?

There could be a virtual method in the base Enemy class describing what miscellaneous actions that enemy takes per turn, which has an empty body if the enemy doesn't have a special ability. Every turn, we can execute that enemy's special ability function, and that enemy's special ability will execute if it has one.

What design pattern could you use to model the effects of temporary potions (Wound/Boost/Atk/Def) so that you do not need to explicitly track which potions the player character has consumed on any particular floor?

The Decorator pattern is well-suited to implementing this functionality. The player character can be decorated with potion effects, and then the Hero object doesn't need to know which potions he consumed - only what effects still linger, which have been set by which potions decorate it.

We ended up not using a decorator pattern but instead put in a switch into the constructor that set the type of potion, and sorted the effects in the actually function that used the potion.

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?

We can make a base class for a Thing That Can Be Generated, and then run the same generation code when some type of treasure or potion needs to be created (or even some other type of generatable item, if we choose to implement more later).

What lessons did this project teach you about developing software in teams?

It showed how different style of development can merge together to bring a new and better perspective to a project that may have not been clear to any individual. Each individual brought unique ideas and solutions to any particular problem as well as the whole project all together. At times it got tough to work together since both parties wanted to do things their own way and the final code would have random working bits that refused to compile as a whole.

What would you have done differently if you had the chance to start over?

Take a longer time to discuss how each part should be done and decide on one method for each .cc file, so it can then be merged together as a whole.