July 18 - (sy) Read and print floor, (w) command interpreter
------Read and print floor
##Reading map design from file
##Populating 2 dimensional vector to represent the map
##Creating wall, floor, pathway objects
##Printing the map to display
------Command interpreter
##Reading inputs from the player
##In the Map class, creating states and notifying observers

July 19 - () Enemy generation, () Player character movement, () Enemy character movement, () Item generation
------Enemy generation
##Generating different enemy objects
##Populating the floor with enemy objects
##Creating general purpose enemy characters with no special bonuses
------Player character movement
##Implement the base character class
##Allow the player character to respond to movement commands
------Enemy character movement
##Make enemies respond to movement command, they move randomly
##Make enemies attack the player when within one block
------Item generation
##Generate base item objects in the map

July 20 - (s) Items, (wy) Race, (sy) Enemies, (s) Combat
------Items
##Implement the effects of the items
------Race
##Implement the effect of the race modifiers
##Implement the race abilities
##Display character stats
------Enemies
##Implement enemy abilities
##Implement enemy drops
------Combat
##Implement combat system, health and damages

July 22 - Finish project
##Polish/add bonus stuff if time left

Question: 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: We make each race a subclass of a character decorator. Each race decorator overrides the necessary functions. As an example, this allows us to calculate the base damage for a combat, and then use decorator to apply a multiplier to the damage done. This way, we do not have to create new formulas to calculate damage, as classes are modifiers to the existing function of the base class. Adding additional race just involves adding new modifications to the base character class. Using a decorator also allows us to implement CLASS to the characters easier, as we just add a CLASS decorator for the player character.

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

Answer: Enemies are generated similar to the player character, but have different behaviours whenever a key is pressed. Both the player character and the enemies are subclasses of the character class. One way to think of our game is that the user is controlling the map, rather than the player character. When a movement control is issued, every character on the map reacts to the command through the observer model. The main difference is that player character's movement is not random.

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: We are in fact using a similar technique as implementing abilities for character races. We use decorators which override the default functionality of our Character class, which provides us great customizability for each class.

Question: 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: We believe the Strategy pattern is a better candidate for the effects of potions. This is because we can give a decorator pointer to each particular potion that will apply the appropriate potion effect to our player (e.g. a health boosting decorator). Using a decorator pattern, we could apply similar behaviour to the potions, but this would require us to create a new class for each potion type.

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: Treasure and potions are both subclasses of the Item class. The item class contains the code for detection of player proximity, printing functionality, and for applying decorators to the player. When generating the two, we only need to change a single line of code which determines if the class to be generated is a Treasure class or Potion.