**CS246 Assignment 5 CC3K: Plan of Attack**

Peter Fryer Davis (pdfryerd) 20466711  
Luke Brown (lmbrown) 20457316

**Step 1: I/O**

* Implement the observer pattern on our Floor, TextDisplay, and Cells (EmptySpace, Wall, Passage, Door, Tile)
* Once we have a working input method, we will print out the TextDisplay using a operation overload on << with Floor and TextDisplay
* Concurrently, implement a text reading mechanism that allows us to read in floor plans from a file
  + This will be done through command line arguments as well as fstream input
  + Each character in the file will be converted into a Cell of the appropriate type
  + These Cells will then be added to our Floor class
  + Floor will then notify TextDisplay what it should have at each of its positions (by calling the notifyTextDisplay() function as part of each Cell in the Floor)

Estimated Time: 2 Days (Approx 16 Hours)

**Step 2: PC Movement**

* Implement basics of Thing, Character, PC, and Human classes (other classes will be added near the end)
  + Add HP, Attack, Defense, Race, and Money to the screen
* Create Chambers and link them to Floor
* Generate PC in a chamber
* Create movement functions for PC
* Notify Floor for every time a PC moves
* Reprint the TextDisplay for every move

Estimated Time: 2 Days (Approx 16 Hours)

**Step 3: Items on Board[[1]](#footnote-1)**

* Create Item abstract class
* Create Gold abstract class
* Create NormalGold Class (other gold classes will be added later)
* Create Potion abstract class
* Create RestoreHealth Potion
* Generate 10 RestoreHealth potions and 10 NormalGold gold in the Chambers at the beginning of a Floor
* Make sure 2 Items don’t generate on the same space (and item is not on same space as PC)

Estimated Time: 1 Day (Approx 8 hours)

**Step 4: Enemies on Board**

* Create Enemy abstract class
* Create Goblin class (other enemies will be added later)
* Generate 20 Goblins throughout the 5 chambers
* Make sure 2 PC/Items/Enemies don’t generate on the same space

Estimated Time: ¼ Day (Approx 2 Hours)

**Step 5: Item Interactions**

* Implement Potion’s drinkPotion() and PC’s discoveredRH() functions
* Implement stepping on Gold which triggers the getAmount() function
* Ensure items disappear from Floor every time an Item is interacted with
* Make text actions for seeing and drinking Potions

Estimated Time: ½ Day (Approx 4 Hours)

**Step 6: Enemy Movement**

* Implement a step function that allows Enemies to move around the board every time the player interacts with the world (stepEnemy() function)

Estimated Time: ¾ Day (Approx 6 Hours)

**Step 7: Combat**

* Allow PC to attack people in any direction (1 block radius)
* Add option for Enemies to attack within the stepEnemy() function
  + The Enemy will always attack rather than moving during the stepEnemy() function if the player is within a one block radius. (Note: At this point there is no dragons or merchants so all enemies all hostile)
* When Enemies die, increase the PC Money amount
* Create combat text actions

Estimated Time: ½ Day (Approx 4 Hours)

**Step 8: Stairs**

* Generate Stairs, ensuring that PCs and Stairs are not generated in the same chamber
* Make sure that Stairs are not generated on the same Tile as Items/Enemies
* Add that when the player walks on the stair it will clear the current board of the enemies and items (but keep the player) and then generate the next level. The player is randomly positioned in the next level, but the player is constant

Estimated Time: ½ Day (Approx 4 Hours)

**Step 9: Varying Enemies/Gold**

* Add different kinds of Enemies (including Merchants)
* Different kinds of Gold (including Dragons and their placements around DragonHordes)

Estimated Time: ¼ Day (Approx 2 Hours)

**Step 10: Varying Potions**

* Different kind of Potions
* This includes the Decorator Patter on Boost and Wound points per floor

Estimated Time: ½ Day (Approx 4 Hours)

**Step 11: Varying Races**

* Add different kinds of Races
* Player can select their race at the start of the game

Estimated Time: 1/8 Day (Approx 1 Hour)

**Step 12: Start/Restart**

* Make sure that the player can restart and quit a game

Estimated Time: 1/8 Day (Approx 1 Hours)

**UML: See uml.pdf**

**Questions:**

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

We will write a superclass called PC, which is a subclass of Character, and would contain all the methods and implementations expected for a PC (getScore, getMaxHP, etc). For each Race, there would be a new class (ie Dwarf) that would overwrite certain methods based on their special abilities. We can do this by making methods virtual in PC and Character. This way, if we wanted to make a new Race, say Knight, then we would simply make the appropriate method virtual and write a new implementation for the Knight class based on their special ability.

How does your system handle generating different enemies? Is it different from how you generate the player character?

Both PCs and Enemies have a base class that they are derived from, class Character. This class has the elements that both Enemies and PCs have in common (HP, Atk, Def, strike method, etc). Enemy is an abstract class that contains all the elements that Enemies have that PCs do not, in this case, and example is the isHostile method.

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

We would implement a visitor pattern that would have access to PC and Enemy. Because the visitor pattern is very useful for accessing a class without actually adding methods to the class, this would be good for adding special abilities like the suggested ones above since these special abilities have to access both fields in PC and Enemy.

What design pattern could you use to model the effect 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 patter would be an excellent method to use in this situation since the potions take place for a certain period of time. This is also very good for stacking multiple potions on at once, resulting in several effects on the character. After a new floor is loaded, you simply eliminate any decorations to the character. This is has to be done by continuously returning the pointer to the decorated character and removing the decoration until eventually we are left with an undecorated character.

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

This can be done using an abstract spawn method as part of the Floor class. We can implement this by making the spawn function that takes a pointer to a Thing, aka an item or a person. Then this function will pick a random place on the board that is not occupied. It will do this by continuously picking a chamber first, then picking a tile within that chamber, ensuring the tile is not occupied. It will then put the Thing \* onto the tile and return a pointer to the tile if we need it (for example, storing where enemies spawn). This function will be used for spawning all subclasses of Thing. The only difference between each implementation would be the probability distributions specific to that type.

1. Note at this time none of the items will do anything this is just concerning generation. [↑](#footnote-ref-1)