Initial Plan of Attack

UML Diagram

Please see the submission under a5-uml.

Project Breakdown

SETUP		
create header files for all classes from the uml diagram	K	Nov 21
write pseudocode for gameplay logic behind the main function	K	Nov 21
DISPLAY		
create classes: board, floor	P	Nov 22
add subclasses to floor sections: cave, chamber, passage	P	Nov 22
implement decorator pattern for cell types: wall, door, stair, passage	P	Nov 22
create classes: map, info	P	Nov 22
implement observer pattern for cells so that map and info change accordingly	P	Nov 22
write function that reads from a text file and loads map	P	Nov 22
test map loading and printing functionality (decorator pattern)	P	Nov 22
test cell notification functionality (observer pattern)	P	Nov 22
CHARACTER		
create entity class with generalized base fields and methods	K	Nov 22
add subclasses to entity: character, player and enemy	K	Nov 22
implement singleton pattern for player character	K	Nov 22
introduce custom abilities for each player race and enemy type	K	Nov 22
add movement functionality to enemies and player	K	Nov 23
test movement functionality	K	Nov 23
implement combat functionality for enemies and players using the visitor pattern	P	Nov 23
test combat functionality	P	Nov 23
ITEM		
create item class with generalized base fields and methods	P	Nov 23
add subclasses to items: potion and gold	P	Nov 23
implement decorator pattern for temporary potion usage	P	Nov 23

RANDOM GENERATION		
write function that randomly selects a chamber	K	Nov 24
write function that randomly selects a cell	K	Nov 24
write function that randomly selects an enemy type	K	Nov 24
write function that randomly selects an item type	K	Nov 24
test random selection helper functions	K	Nov 24
hook up main function with command-line input	K	Nov 24
test and debug	both	Nov 27
ENHANCEMENTS		
add additional player races with custom abilities	K	Nov 29
add additional enemy types with custom abilities	K	Nov 29
add support for wasd controls	K	Nov 29
colour-code display	K	Nov 29
add additional items with custom effects	P	Nov 29
add additional weapons	P	Nov 29
add support for random floor generation	P	Nov 29
test and debug	both	Nov 30

Design Questions

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

Each character race could be easily generated using inheritance. We would have a base class called Character with a set of base fields and methods that captures the similarities between all subclasses. Then, we would create a Player subclass which inherits from Character, adding player functionality through new fields and methods or through overloading base fields and methods. Then, we would create Player subclasses which would model each character race in a similar manner. With this implementation, it would be extremely easy to add additional races simply by creating a new subclass that inherits from the Character superclass - all the base functionality would be already defined. Then, we would simply have to overloaded the base methods to customize the new class and its abilities.

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

Different enemies would be generated using the factory method. A generate enemies method will be written that will use random number generation to pick an enemy, create one and return a pointer to it. Only in this function will the constructors of enemies be called. Thus whenever an enemy is generated, we will not need to pick an enemy constructor. Instead, we would just call the factory method which ensures random enemy generation. This would be different from how the player character will be generated; since there should only be one instance of the player at any time in the game, the player is implemented as a singleton. A get player function will be used to generate or get the pointer to the player.

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 custom abilities of the enemy characters will be made through a combination of overloading base methods and the visitor pattern. All enemies will have a set of generalized functions for actions such as moving, attacking and being slain. When a race specific ability changes how these base methods work, a new function which will overload the base function will be written. Since some special abilities require the type of both the enemy and the player to be determined at runtime, the visitor pattern will be implemented to allow such interactions (for example, orcs being able to do 50% more damage to specifically goblins). The same technique will be used for the player who will also have a set of generalized base actions that will be overloaded by unique abilities for each race since there are no differences between the player and the enemy that would prevent this implementation from working.

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

The effects of temporary potions could be modelled using the decorator pattern so the effects of potions do not need to be explicitly tracked. Whenever the player consumes a temporary potion, am enhancement decorator is added to the player. This enhancement decorator will overload the getters of the attack and defence field of the player and alter the results of the getters depending on the specific potion(s) consumed. This decorator layer will allow multiple potion effects to take place without explicitly tracking the potions consumed, and when the floor is passed, the decorator layers can be discarded which will remove the effects of the temporary functions.

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 generation of treasure does not duplicate code?

Treasure and potions are both items which share several similarities. These similarities can be generalized within a base class method in order to reduce code duplication. First, the random cell selection procedure for both is the same. Thus, we can write a function that randomly selects a chamber and then randomly selects a floor cell within the chosen chamber. This random selection function would be shared and used by both items to randomize location. In addition, the type of potion or treasure generated must be random. Since the base logic for this process is the same, we can also generalize this type selection function. In this way, through generalizing the two functions - one for randomly generating the item location and one for randomly generating the item type - we can reuse as much code as possible.