University of Waterloo

CC3K Game Development Plan

Yinong Wang, Raymond Tan, Qifan Zhu

Attack Plan

Part 1: UML and Schedule
March 23 - March 27:

For the first week, the group members will have meetings and start to design the overall picture for the project. The initial draft of UML diagram will be done by March 27. The diagram is subject to change depending on problems that might occur later while the game is being constructed and while header files are being written.

Part 2: Basic Design and Basic Implementation
March 27 - March 30:

Begin to construct and add all the methods and class fields on the header files(.h) based on the UML.The main goal for the first week is to build the skeleton structure of the game, including the basic floor map, objects generation,player movement, enemy interaction, and game flow. All three members must implement the basics first before adding any additional features.

After the interface for all classes are completed, some new design pattern will be added on the UML, such as the visitor pattern for adding weapons and armor. For this week, we will be only implementing the very basic elements of the game, such as initializing the game and tracking enemies action when player move and etc.

Here is a basic idea of spawning the objects on the game grid by using the random number generator and using the chamber separation algorithm. First, we implemented an algorithm, which returns a pointer, to store all coordinates of each chamber in a two dimension array of pairs with the first value is int and second value is also an int. This algorithm is very useful for spawning object, since we just need to generate a number among all five chambers and randomly choose a coordinate in the chosen chamber and check if that coordinate is valid to spawn. For tracking enemies actions, the observer design pattern can establish the relationships between player, enemies and text display.

On March 30th, objects generation, player movement, and enemy interaction must be implemented. Debugging, additional features, DLC will be added later on.

Part 3: Additional Required Contents, DLC and Public Testing
March 30 - April 3

All the implementations will be finished, including all classes of players, enemies and items with additional DLC. The WASD control DLC will be added.

Through factory design pattern, we provide flexibility of potentially adding new floors with different difficulty (e.g. different spawning pattern).

Inventory for player will be introduced and player will be able to store items in their inventory (i.e. Potion can be saved for later use). Through decorator design pattern, we can easily add more buff to player by wrapping player in wrapper class. Thus, weapons and armors and other equipments will be introduced to increase player's ability. More precisely, some of the decorator wrappers will be peeled off from player to ensure only the required permanent effects follow player into the next floor.

Simple AI for enemy will be implemented. Enemy will start chasing player rather than randomly wandering once player is within a certain distance.
Trading system will be implemented which allows player to buy some items from Merchants in exchange of gold and Merchants will also be buying items from Player and giving them back gold.

More player race and enemy race will be added if time allows.

At last, we will debug all modules implemented as well as the entire basic game; debug each DLC as well as the entire enhanced game; match the actual program with our UML; run tests in all possible scenarios.

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?

Since all races have a common make() method, we can use the factory method pattern to generate the races and let the factory to decide which races needed to be created.

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

In our UML, you can see that character is a superclass of both player class and enemy class since they have common fields. The way to generate enemies and player is very similar. When the createEnemies() method is invoked in the floor class to spawn enemies, each created enemy is attached to the player's observers list.

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.

We could use the visitor pattern for double dispatching. Implement overloading methods of attack for player, one is fighting against common enemies and the second method for the one who has special ability.

Question: The Decorator and Strategy pattern are possible canadians to model the effects of potions, so the we do not need to explicitly track with 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.

In our opinion, Decorator Pattern is the best fit for adding potion effects to the player.When a player use the potion, potion can be a decorator and wrap it onto the player. When the player go to the next floor, those temporary potions' effects are gone and. we could just simply unwrap that potions. The disadvantage of using Strategy Pattern is we might have to add extract fields to the potions and we might have to explicitly track the potions that the player consumed.

How could you generate items so the 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 then generation of treasure does not duplicate code?

In our UML, potion class and gold class are the subclasses of item class. We can use the Template Method pattern for code reusability. By implementing a template method that takes any type of items we can generate each item easily.

Taks	**Name**	**Expected Due Date**

UML	Raymond, Yinong, Qifan	March 27th, 2017(Done)
Floor.h(Generates enemies, player and objects) Floor.cc	Raymond	March 30th(Mostly Done)
GameGrid.cc GameGrid.h main.cc	Yinong Wang	March 30th(Done)
TextDisplay.h TexDisplay.cc (displaying the map and keep track of objects' moments)	Qifan	March 30th(Mostly Done)
Character.h Character.cc Item.h Item.cc	Raymond, Yinong, Qifan	March 30th(Little)
Chamber.h Chamber.cc (Store each chamber's coordinates)	Yinong	March 30th(Done)
Debug	Raymond,Yinong, Qifu	April 3rd(Not started)
Additional features	Raymond,Yinong, Qifu	April 3rd(Not started)