Zheng Fang & Ruijie Zhang
To: Professor Nomir Naeem and ISA Kirsten Bradley
March 26, 2015

CS 246 Final Project

Plan of Attack

1. Subject of Project

For this project, we have chosen "Chamber Crawler 3000" as our subject after some serious considerations. The biggest reason is because of interest, we are all big fans of RPC (Role Playing Control) games, and we have played many of them since the age of Windows 95. Some of our all-time favourite collections including "Contra" and "Pokemon" . For me(Ruijie Zhang), I personally have an old wooden PC kept at home just to play those games, because most of them are unfortunately no longer supported by the current system. So it is pretty exciting to see ourselves create and implement a game like "Chamber Crawler 3000".

2. The Big Picture of Attacking

Our plan of attacking includes five steps. The first step of this Project is to create the board that needed for the game, which is the cornerstone of everything else. And the next step would be to categorize all other elements in the game, such as: races of players, different kinds of enemies, different kinds of potions and creating corresponding classes for them. Our third step would be implementing all the necessary logics that needed to link the board and other elements, such as "player moving on the board", "battle system", "using potions", "restart, exit and score" ,"random generation" and e.t.c. After completing that, there goes our fourth step ,

“Special cases handling”. We will handle some special situations such “Dragon Horde”, “Merchant” and all the other different skills of different races and enemies. And then it comes our last step, which is the fifth step, “playing the game and bug detecting”. In other words, the ‘Quality Assurance” process. This will probably be the most fun QA job that will ever exist in this planet!

The following is our possible schedule:

-Mar 28th-29th Creating the board and linking cells together. (Together)

-Mar 30th Creating the player class , enemy class and potion class. (By Zheng Fang)

-Mar 31st Creating the logic to make player move in the board. (By Ruijie Zhang)

-April 1st Create logic to random generate player, stair and enemy. (By Zheng Fang)

-April 2nd Create logic of attack and random moving. (By Ruijie Zhang)

-April 3rd Create logic of entering and quit and restarting the game. (By Zheng Fang)

-April 4th Create the skill class and link everything together. (By Ruijie Zhang)

-April 5th-6th Testing and bonus features. (Together)

-April 6th Demo (Together)

Also, considering the complexity of the game, we will do “unit testing” as we go on. As this will avoid strange bugs in the end as much possible and make our lives so much easier. Hopefully, we can finish early and try some bonus features such as Ncurses and other fun stuff!

3. The Implementation Details

For the big structure of the game, we decided to use the design of Question 2 from Assignment 4, the “Floodlit game”. Our core class is called “Cell”, which we will use to structure the board. The game board is 25 * 79, which means we will have 25 * 79 cell objects connected together to form the board. And we will use the Observer pattern, and created a class called “TextDisplay”, who is the “Observer” of all the cells. And each time there is a change in any of the cells, it will notify “TextDisplay” and update the picture displayed in the game. And Cells have an array of pointers of cells, which is used to keep all the information about its “Neighbours”. This could be extremely useful when it comes to battle, as the “Enemies” will have to check if the player has moved near them and attack the player.

Additionally, the cells have a pointer to Content, which is abstract class we create to add all other elements of the game, such as different races, different kinds of enemies and potions and

treasures. So Content is the base class of several different classes, such as "Characters" and "Treasures". The "Characters" class is the base class of two different classes, such as "Enemies" and "Player" and both of them are abstract classes, and the base class of different classes like "Human", "Dwarf"', "Elves" ,"Orc", "Troll"" and e.t.c. We decided to make "Potions" to be a derived class from Players because we will use "Decorator Pattern" in this case and we will discuss the reason in our "Questions & Answers" part. Potion is an abstract class and is also the base class of "Positive Potion" and "Negative potion". And they are abstract classes as well and have derived classes such as"PosItive HP", "NegativeHp", "PositiveAtk", "NegativeAtk" and e.t.c.

And then, we will have a class called "Board" which actually "owns" all the cells. And we will created a class called Game which "has a" board , a player and a vector of of pointers to enemies. Also it has some useful methods to help the game run smoothly. Also it hides a lot implementation details from our "main.cc", so we don't have a huge chunk code in our "main function".

Furthermore, we used "Singleton Pattern" on board, game and player class to make sure there can only be one game, one board and one player. To implement the skills, we will create a class called "Skill", which is the base class of all the different skills. And we will add a field of in our Character class, which is an array of pointer to "skills", so the skills can be used when needed.

To handle the effects of the potions, we will use "Decorator Pattern". We will make the "Potions" be the subclass of "Character", and then we can link them one by one to our player, so we do not need explicitly track which potion have the player used.

The above is our initial thought of how to implement the game, and it might change as we go through the project. For further details of the design, please see our "UML.pdf".

4. Questions & Answers

Question.1 How could your design your system so that each race could be easily generated? Additionally, how difficult does such a solution make adding additional classes?

Answer:

In order to easily generate each race, we create a base class called "Player" and set each race in the derived classes. The base Class have some common fields that all derived class will have :

-race

-Hp
-Def
-Atk

and some common methods that will be used by the by all derived class such:

-attack
-calculating damage
-updating character Hp

In this way, we do not need redefine those methods again and again for each class.

Also, it allows us to add additional classes pretty easily. we just need add one more derived class from "Player" without modifying code from other modules.

Question.2 How does your system handle generating different enemies? Is it different from how you generate the player character? Why or why not?

Answer:

To generate different enemies, we first use rand to generate random number between 1 to 18. A specific enemy will be generated according to the random numbers that are generated.

-Werewolf will be generated if the random number is one of 1,2,3,4.
-Vampire will be generated if the random number is one of 5,6,7
-Goblin will be generated if the random number is one of 8,9,10,11,12
-Troll will be generated if the random number is one of 13,14
-Phoenix will be generated if the random number is one of 15,16
-Merchant will be generated if the random number is one of 17, 18

Finally, we use "for loop" to spawn random enemies 20 times at random coordinates.

The way to generate different enemies is basically the same from generating the player character except the player's race can be intentionally chosen at the beginning , while the enemies' race are randomly generated. And similarity is that both enemies and player characters are spawned at random coordinates. We have a such method in Class "Board" that is responsible to generate random and valid location in the Board.

In fact, since the functionality of "enemies" and "player" are almost the same, so we created a abstract class called "Character" and make "Enemy" and "Player" to be the subclass of "Character". In this way, we can even reuse more code as we can just define common fields such as:

-race
-Hp
-Def
-Atk

and common methods such as:

-attack
-calculating damage
-updating character Hp

in our super class "Character", so that we can save time and make our code shorter and better.

Also there is one more difference that the player cannot be spawned in the same chamber as the stair, while there is no such restriction about Enemies.

Question.3 How could you implement special abilities for different enemies. For example, gold stealing for goblins, health regeneration for trolls, health stealing for vampires, etc.?

Answer:

We will create a super class called "Skill" and derive different sub class from it which represents different skills. Each "Character", which are "Enemy" and "Player", will have an array of pointer to skill objects, that will store the skills they have. It allows us to create and add new skills to player pretty easily. It also allows us to reuse code for similar skills, such as "Hostile" for both "Merchant" and "Dragon". The "Hostile" class will have methods to return a boolean value that determines whether "Dragon" and "Merchant" are hostile or not.

-Merchant: create class "Hostile" to that has a method to determine whether Merchant is hostile or not.
-Dragon: reuse the class "Hostile" to that has a method to determine whether Dragon is hostile or not.

-Goblins: Create a class called "stealing gold" that has a method that if Goblins are one block away from the player character, the amount of gold that the character has will decrease for each move.
-Trolls: Create a class that has a method that if their health are between 1 to 119, increase amount of the health after each player's move.
and similarly for Vampires, WereWolf, and Phoenix.

The good thing is, we can get creative and mix the skills for player and enemies, for example, a player character can have the health regenerating skill as well. We don't need change anything, we just add the pointer of a "HealthRestore" object to the array of skills inside of the player. Or even more fun, player can buy a skill from Merchant with certain amount of gold. These could all be easily achieved by just creating a "Skill" object and give the pointer to the player and add it in the "skill array".

Question.4 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?

Answer:

We use the Decorator Pattern. We make the Potion to be a derived class of Player, and then we can link all the potions together and eventually link to a player race. So if we want to know the Atk for a player, we just call a method called getAtk(), and it will go through the chain and recursively calculate the final Atk. And having a new potion is just adding a new potion object to the end of the Chain. We do not need to explicitly track which potion we have had. When entering a new floor, we just delete all these potion objects and create a new chain.

Question.5 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 "Potion" are the derived class that belong to one big base class "Content". And the way to generate a content is to first randomly pick a valid non-occupied spot in one of the five chambers and then add different content to it. So we decide to implement one function to find the random location, because we will need it again and again.

We have a method in our Board class that returns the address of a random cell in one of the five chambers. This function is used over and over again for generating enemies, Potions and Treasures. So if the random cell is non-occupied, we will add content to it, or we will call this function again to find a new cell. In this way, this function could help with all five needs: generating player, generating stair, generating enemies, generating treasures and generating potions. We do not need implement similar function for five times.