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

CS 246 Final Project

Final Design

1. The Final Structure

• Cell: The fundamental element of our design is the class Cell. Cell is the element that forms the game board. The game board is 25 * 79, which means we have 25 * 79 cell objects connected together to form the board. Each cell has an array of Cell pointers , named neighbours, which stores the neighbours that is nearby of this cell. Each cell also has a Content pointer, which points whatever content the cell has. This ensures that each cell object can only have one content.

• Textdisplay: We used the "**Observer Pattern**" to help us display the board. We have a class named "Textdisplay", which is the "Observer" all the cells. So if there is any change in the cells, the Text display will be notified and print out the new board.

• Board: We used the class Board to be the class the actually have all the cell objects. In a board object, there is a 2d array of cell objects with size 25 * 79. They are all allocated on stack. Board has a pointer to Textdisplay and board is responsible to free it. We also employed the "**Singleton Pattern**" to ensure there is only one board.

• Game: We have a game class which has the board, and it is responsible for taking the input, and get the game running. It has all the logic the game needed to interpret all the commands. We also used "**Singleton Pattern**" to ensure the there is only one game going on.

• Content: Content is something the cell object owns. It is an abstract class that has many derived classes such as Character, Treasure and Potions. Each cell object only has one content pointer and thats the way we used to ensure no two contents occupy the same cell.(i.e no two objects on the same spot of the board.)

• Character: Character is the abstract class and base class for Enemy and Player. It is also a derived class from Content. It has some useful functions which will be used by Enemy and Player, such as attack() and updateHP() and some virtual functions which will be defined in Enemy and Player. Also, it has many attributes which Player and Enemy have in common, such as Hp, Def, Atk , Race, and etc.

• Enemy: Enemy is also an abstract class that has 6 derived classes , which are all concrete classes that represents each type of the enemies.

• Player: Player is an abstract class that there are 4 derived classes from Player, which represents each race that PC can choose. We employed "**Singleton Pattern**" to ensure that there is only one player in the game.

• Potion: Potion is an abstract that has 6 derived classes which represents each type of Potion. Here, we make Potion to be a derived class of Player, because we want to employee "**Decorator Pattern**" here. Each of the potion is attached to the player, so we don't need explicitly track which potions have been used in a particular floor. When the player enter the new floor, the attached potions will be deleted.

• Treasure: Treasure is an abstract class that has 4 types of Gold that would be needed in this game. It has a int which represents the gold value this horde has. Need to mention that MerchantHorde is only dropped when a merchant is killed. Also the dragonHorde will only be available to pick up when the dragon has been killed, so the dragon will have an pointer to this horde. These logics will be done in done in game.cc and board.cc and cell.cc.

2. The logic of the game

Most of the game logic is done on game.cc, board.cc , character.cc and cell.cc . The game.cc is mostly responsible interpreting the command of the player, and call corresponding methods from other files. Also, it is responsible for things as determine winning or losing condition, entering a new floor , and printing score and restarting and quitting the game. These will be done by a method called "startGame()" and it is a recursive function that has a "game loop".

The random generation methods are done by both board.cc and game.cc . The board.cc will call methods that return a pointer to a random cell the board has. And then the game.cc will

add a Content to it, such as a Player , an Enemy , a Treasure or a Potion. The randomness is done by functions and methods in board.cc .

The attack logic is done by both game.cc and character.cc . character.cc has functions to determine whether the attack misses or how much damage should be calculated and updated. And game.cc will interpret the command to pass the correct Cell* to methods in character.cc.

The moving logic and attaining gold logic is done by game.cc and cell.cc . The cell.cc has a method to move the a Content from one Cell object to another Cell object. And game.cc is responsible to let the method in cell.cc which direction should the player or the enemy move.

3. Difference between original design

Originally, we plan to make a class called "Skills". We originally wanna make the all the skills be skill objects. The benefit of doing this is we can assign different skills to different races and enemies pretty easily and make it easier to add new skills. However, due to time constraint, we are unable to make such a class, so we implemented all the skills for the all the races and enemies quite differently. Some skills, such as Elf's "Negative potion has positive effect", we changed our six classes for the potions to make it work. We have added different functions and methods in various classes to implement all the skills, that is the biggest difference between the implementation and the original design.

The other parts of the game is pretty much as we originally planned. To simplify things, we also removed two abstract class Positive Potion and Negative Potion, instead, we just have one general abstract class Potion. We have made the corresponding changes in our UML.

Questions & Answers

Question.1 What lessons did this project teach you about developing software in teams? If you worked alone, what lessons did you learn about writing large programs?

Answer: I think the most important thing for programming in groups is planning ahead and communication. Plan ahead helps us to stay organized and make sure we do not duplicate code. It also helps us make sure we are keep up with the schedule and do not need to rush it in the end.

Also, communication is also another thing we learned. We make sure our programming style is consistent, such as whether pass by value , pass by reference or pass by pointer. This avoids lots of trouble when we compile the program together.

Question.2 What would you have done differently if you had the chance to start over?

Answer: In our implementation, we make each cell has one Content pointer to ensure that there is only content in each spot. So the Content class has many derived classes and therefore needs many virtual methods. Since every time we define a virtual method, we need the implementation of that method in all the derived classes. This could be somewhat awkward. For example, for the Character Class, it has the method Attack, so there is a virtual method in Content named Attack. But then, the treasure and potion class will also need the implementation of that method, which is Attack(), otherwise it won't compile. But it does not make sense for a Potion to do attack. So we added some methods that will never be called.

One solution is to use casting. But we don't really feel safe to do that. So if we have the chance to start again, we probably try the "**Visitor Pattern**" to avoid awkward situation like this.