CS246: Final Project
CC3K+

J283Sun,Y59Qi,K42Shi
20733095,20721758,20744862

Introduction:

This is the report for our final project CC3K+. It is a developed based on Rogue. Consisting different kinds of enemy, player and five floor. Each floor have five chamber. The player needs to kill the enemy and get the compass to the next floor. Until the fifth floor, the player will win the game. There also have different buff, debuff, treasure.

Overview:

We divide the game into four three different parts: character(include enemy and player), board(cell, chamber and floor) and Item and incorporate high cohesion and low coupling.

Designing detail and class break down:

In Class Game:

std::vector<Floor *> floors: for creating five level of floors
currentFloor: for checking which floor the player is on
action: for output the player action, whether the player is pick an item or hit an enemy, also the enemy's action if it hit the player.
ifWin: check if the player is on the fifth floor and win the game
ifDie: check if the player's HP is 0 and whether to end the game
ifQuit:check if the player wants to quit the game
gameflow(): line by line move the enemy and hit the player and print the text
movePlayer(string): move the play in the chamber and stairs and also get to the next floor
pickUp(string): turn and pick up the item on the cell
attack(string): attack the enemy
quit(): whether to quit the game
gameOver(): check the player win or lose or quit
checkOver():return the states of player

In Class Player:

HP, Atk, Def: the player's HP, Attack and defence
x,y: the position of the player
money: the money player owns
ifOwnBarrier: does the play the barrier suite
race: whether the player is a human, elves, dwarf, orc
increaseHP(int): add the player's HP
decreaseHP(int): decrease the player's HP
Def, Atk increase and decrease is similar as the HP
setHP: for set the HP for the player (for other class to use)
similar set function for money, x, y
getHP: get the HP of the player, (for other class to use)
similar get function for money, x, y, race, own Barrier or not
attacked(int): calculate the HP decreased that hit by an enemy
ownBarrier(): whether the play get the barrier suite

In Class Enemy:

x, y, HP, Def is similar as class Player

type: the enemy's type, V for vampire, T for Troll, W for Werewolf, G for Goblin, P for Phoenix, M for merchant, D for Dragon

compass: whether get the compass to the next floor

ifAttack: whether the enemy will attack the player

ifMove: whether the enemy will move

getHP(): get the enemy's HP

similar get function for Def, Atk, X, Y, compass, type, ifattack, if move

changeIfAttack: check the enemy whether to change attack mode

changeHoleState: check the dragon hole's state

setX,setY,setCompass is similar as player's set function

attack: calculate the HP that hitten by a player

In Class Floor:

Floor: save all cells' pointers in 2D vector (used for layout)

Chamber: save cells' pointers chamber by chamber (use for spawn items and chracters)

Player: save player's pointer

Stair: save the cell that is a stair

generatePlayer(char)(int): read in a type of player and spawn a character on the floor and return which chamber the player spwans

generateEnemy()(void): generate 20 enemies on the floor by some probability distribution

generatePotion()(void): generate 10 potions by some probability distribution

generateTreasure()(void): generate 10 golds by some probability diestrion

generateStair()(void): generate a stair and does not in the same chamber with player

printFloor()(void): print the floor

generateFloor()(void): generate a whole floor(by using private methods: generateEnemy, generatePotion, generateTreasure, generateStair)

In Class Cell:

int X,Y represents the corresponding positon in the 2D vector(floor)

ifOccupied(bool): check if the cell is occupied by any items or enemies

type(char): store the type of the cell(wall, floor tile, doorway or passage)

ifAccess(bool): if the cell is a stair, check if the player can access it

ifPlayer(bool): check if the player is on this cell

Player(pointer to a Player): store the pointer to Player

ifEnemy(bool): check if the enemy is on this cell

enemy(pointer to a Enemy): store the pointer to Enemy

ifItem(bool): check if any Item is on this cell

item(pointer to a Player): store the pointer to Item

setEnemy(pointer to Enemy)(void): set the enemy on the cell

setPlayer(pointer to Player)(void): set the player on the cell

freePlayer(void)(void): move the player from the cell

freeEnemy(void)(void): move the enemy from the cell

freeItem(void)(void): move the item from the cell

In Class Item:
Int x,y represent the corrdinate
Name: the name of the item, P(potion), G(treasure), C(compass), B(barrier suit)
Picked up:whether it can be picked up or not
Char printItem(): print the name of item
getX(),getY(): accessor for other class
ifPickup(): if the item can be picked up or not
changePickUp: change the pickup states
setX,setY:for other class to change the value of x and y
whatisit: check for the name of the item
getValue: set to 0

Design:

The final and the first doesn't have a lot difference. The way we split our program are very similar. Separate them into Enemy, character, item, floor. Then use the Game to connect with the main function. We also change the raw pointers to the shared pointers, so didn't to worry about the memory leak much.

The change we accommodate is treasure and potion. At first, we think that the treasure is for the whole game and potion for the same floor, so we decided to split into two parts. Then, we saw the Barrier suite and compass, these two item, which is sound more independent. Shouldn't be put into any of treasure or potion. So we decide to create a larger class, which is Item, to include potion, treasure, barrier suite and compass.

For the DD2, we considered more precise functions than DD1, but the big frame is the same.

The way we accommodate change:

We split each category into small category for high cohesion and low coupling. Split character into four different races, which means we can add also add the fifth race. Similar for enemy. We let each enemy have its own property, which means we can also add the property to each enemy, and also some special property. For item we get two independent item and two larger class: treasure and potion. Potion is for the current floor and treasure can be inherited. This structure can let us add new staff into the game.

QA:

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: The solution is that set the four different races of characters as four derived classes of the abstract super class called Character. When adding additional classes, just add more derived classes under the super class. Therefore, it is easy to add additional classes.

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

Answer: Firstly, set a vector of pointers to seven different enemies as the field in floor class. Then, create a public method called generatingEnemy, where the constructors of corresponding classes will be called and generate the enemies.

Yes, it is different from how to generate the player character. It is because there is only one

player character and many different enemies. The enemies also have to generate randomly. For different possibilities of each enemy. Merchant also have to decide attack or not attack.

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: All the enemies are the subclasses of an abstract enemy class. Add these properties as the public method of subclasses and use it when there is combat between character and enemies. Also might use the override function.

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: Decorator pattern can be chosen since we can add and observe the changes the HP state of the character at the runtime.

5. How could you generate items so that the generation of Treasure, Potions, and major items reuses as much code as possible? That is for example, how would you structure your system so that the generation of a potion and then generation of treasure does not duplicate
code? How could you reuse the code used to protect both dragon hordes and the Barrier Suit?

Answer: Have a template method in floor class that generates every different type of items.

Final 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: In teams, we need good communication. Each person has different ideas, we have to communicate the big idea at the beginning to avoid structure problems. Additionally, since we split the work, we need to borrow others function. We should let others know and also if they need our function, we need to have good communication for what specifically this function do. If one of three of us didn't finish the work, it will also drag other two people's work speed. So we should follow the plan more. If I worked alone, I think I will spend more time on structure making, since use different patterns will receive lots of time and reduce the work load.

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

Answer: We should consider using pattern at the beginning of our work, rather than doing it and thinking the pattern. And also we should understand pattern earlier, it will increase our working speed.