The Game of ChamberCrawler3000+

By Henglin(Henry) Huang and Di Ma

Part.1 Overview of the structure

1. Main Classes:
    a. Game
    b. Floor
    c. Chamber
    d. BatteFile
    e. State
    f. Cell
    g. Content
2. Main subclasses:
    a. Character
        i. Hero: Elf, Orc, Human, Dwarf, Max
        ii. Enemy: Troll, Dragon, Vampire, Werewolf, Goblin, Merchant, Phoenix, Henry
    b. Event: Potion, Treasure, Barrier, Compass
    c. Terrain: Wall, Passage, Tile, Doorway

Description of each classes

1.a Game

Game holds floor and accept input from user, transfer a command into floor, output result while the game running. Which is the major part to manipulate all of the IO stuff.

1.b Floor

In order to achieve high cohesion and low coupling, we use the Floor class to finish most of the calculation and open a few public ports. Floor is responsible to generate all 79*

25 Cells. Use chamber's public function to generate character, monster and items. By using BattleFile to manipulate the combat part between PC and NPC. Meanwhile, itself can complete the action between PC and items. It will also update the PC's information to State.

1.c Chamber

By using the address from the cell and mutate cell's information to generate different stuffs like character, monster or items. It will also refresh the floor, delete everything generated.

1.d BattleFile

Whenever a PC is within radius of 1 with a hostile enemy, BattleFile will finish the combat and return the result to floor.

1.e State

State is used to store all the information of PC.

1.f Cell

Cell stores the coordinate of itself, and owns a content pointer which contains the information of what the cell has.

1.g Content

As the cell's information will continuously change which makes unpredictable memory error. We use the Content class the store the property/information of cell. Content has three subclasses which are character, event and terrain. We will introduce theses three classes later.

2.a Character

Character is an abstract class which has two abstract subclasses, enemy and hero. Enemy has different subclasses, each of them represents one kind of the enemy. Hero has different subclasses as well, each of them represents one kind of race.

2.b Event

Event is an abstract class which has some subclasses, each of them are one kind of special event which will trigger some ability to happen.

2.c Terrain

Terrain is an abstract class which has some subclasses, each of them represent different terrain.

Part2. Design:

We used the Observer design pattern for our project.
Game, floor and cell are observers. They accept information from subjects in different situations. We highly concentrated most functionality into floor classes to achieve high cohesion and low coupling.
Floor will notify cell that which chamber this cell is in, meanwhile, the cell will notify the adjacent cells (within radius of 1) that I'm in this chamber.
Every time the PC moves, floor will notify the cell has the PC and make the move. At the same time, that cell owns the PC will also notify the adjacent cells in order to occurring the following situation,

1. A monster is within 1 radius of PC, they will notify floor to deal with what should happens next (which will be the floor responsibility to notify BattleFile to let them combat).
2. If potion is within 1 radius of gold, they will notify floor there is a potion, and do the following according to what floor should do next.
3. Pc within 1 radius of dragon horde, it will notify the dragon horde cell, which will make the dragon be angry. Then dragon will be hostile to pc.

Following will cause the floor to notify game that the game needs to end

1. If the PC move to a stair, it will notify floor once PC reach stair five times.

2. If the PC died while combating, BattleFile will notify floor and floor notify game to end.

The main difference between the original design and the final design is that we use the BattleFile class to deal with all combat stuff. The other difference is that we use the Content class as an intermediate between cell and three abstract classes, so that we could avoid some memory problem and make our program more flexible.

Part3. Updated UML

Please see UML pdf.

Part 4. Resilience to Change

Resilience to Change:

1. Adding new races or enemies

   Solution: It is just easy as adding a new subclass of hero or enemy. If the new enemy has special ability, then we just add few lines code in the BattleFile to fulfill the requirement
2. Adding new items

   Solution: It is also easy. We just add new subclasses of event.

Part 4. Answer to Questions

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 classes?(Same as original)

We create an abstract class called character, and all different races are subclasses of the subclass of character called hero. Different races have their own attribute and initial data. Every game we create a hero, we just simply call the constructor.

When we want to add a new race, we just need to create a new subclass for this specific race with its attribute.

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

It is almost the same as generating player character. Enemy is still an abstract class, and all type of enemies such as Werewolf etc are subclasses. There's no big difference from the way generate the player character. Because they are all subclasses of character.

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

We create a class called BattleFile. We will perform different abilities while combating by different implementation.

Question. 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? (Same as original)

We are going to use observer design pattern for this.

Question. 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? (Same as original)

We can simply let Treasure, Potions be the subclasses of the same superclass. So that when we are generating, for example, treasure, we just let the type of Treasure to be argument.

Part 5. Extra Credit Feature

We add new race called Max who has (Hp: 200, Atk: 100, Def: 100). Denoted in map as “!”.

We also add Henry as a new enemy has (Hp: 5000, Atk: 0, Def: 0). Denoted in map as “?”.

Part 5. Final Questions

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?

Thinking and working together is always better, faster and more efficient than working alone. It was a big surprise for both us that what an amount of work we have done. To transferring our knowledge of basic C++ and some design patterns into a real game (though it is really simple compare to real world game), we need to continuously think and practise of what we know and finish the code in the way we can manage. We also seek for help from considerable online material of C++ and watch tons of YouTube design pattern videos.

The most important lesson we’ve learned from finishing this program is that the design is always the highest priority. In this case, UML was our first thing to accomplish. A bad design will lead to lot of struggles. Breaking big project into small pieces, and then discuss who is responsible for which parts. Since everyone has their own weakness in programming, we divided the project into parts making sure each one are doing the part he is good at.

At the same time as we completed the project, we also understood the knowledge we learned. And in the process, we found that the problem, we shared ideas, shared knowledge, and shared learning resources. Through this project, two of us who we did not know before have become good friends. This is also a great harvest for working together, where you can meet friends and share fresh ideas.

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

If we could have a chance to do the project all over again, we would definitely put more work on the estimation of the workload. Since the unbalance workload, we did not follow the

schedule and even lead some arguments between us. Due to this reason, we didn’t finish the bonus part as we expected which is a pity for us.

We will also strictly follow good programming style, such as avoid using both CamelCase and underscore_case to define function name or some meaningless name.