Chamber Crawler 3000

Design Documentation

Cheng Qiwei(20553126)

Zhu Yandong(20588720)

Overview:

- **Explanation of classes**
- **Explanation of Patterns**
- **Changing from Due Day 1**
- **Description of Strategies we used in our Project**
- **Time Management**
- **Questions and Answers For Due Day 1**
- **Questions and Answers For Due Day 2**

1 Explanation of classes

Floor:

- Floor is a class which red and store a map for the game.
- It has a filed theFloor which is the map of the game, and it is a vector of vector of cell(a class which will be introduced after).
- It also contains integer fields which store the maximum number of enemies, treasures, potions and the current number of enemies, treasures and potions, and an integer filed which represent whether merchant will attack player character.
- It provides the public methods that return the number of different types objects in the mp and the methods which initialize the player character, enemies, potions and floor stair.

Cell:

- Cell is a class which represent each cell in the 25*79 cells map in theFloor.
- each cell store a pointer of info(a class which will be introduced later).
- Cell has methods that return the pointer of info to move pointer of info to other cell and Cell also has methods to check the information of each pointer of info.

Info:

- This is the class which represent the information which is be stored in each cell.
- All info are objects which are located on the map.
- This class does not have any fields and only have one virtual method that return the string field type which is the name of the type of the objects.

Pc(subclass of info):

- Pc class is the class which store all attributes of all player characters(such as hp, atk, def, gold, maxhp, type and etc). Pc have methods that attack non-player character, use potions, pick golds and return all its fields.
- Pc have two types constructors, and each subclass(which represent a specific race of player) will call a constructor of pc.
- Pc is a decorator of all its subclasses.

(All following classes are subclasses of pc)

— **Shade**: a subclass of pc, it represent race shade and it has a unique ctor of pc in its own ctor. It has a string field type and a method return its type. Shade has talent skill that give 50% extra score when game is ending.
— **Vampire**: s subclass of pc, it represent race vampire and it has a unique ctor of pc in its own ctor. It has a string field type and a method return its type. Vampire has skill that gain 5 HP when attack npc except dwarf(when attacking dwarf, losing 5 HP instead of gain 5 HP).
— **Troll**: a subclasses of pc, it represent race troll and it has a unique ctor of pc in its own ctor. It has a string field type and a method return its type. Troll restores 5 HP after each turn.
— **Drow**: a subclasses of pc, it represent race drow and it has a unique ctor of pc in its own ctor. It has a string field type and a method return its type. Drow have 50% extra effect from all potions.
— **Goblin**: a subclasses of pc, it represent race goblin and it has a unique ctor of pc in its own ctor. It has a string field type and a method return its type. Goblin steals 5 golds when attacking non-player characters.
— **Saber**: a subclasses of pc, it represent race saber and it has a unique ctor of pc in its own ctor. It has a string field type and a method return its type. Saber will reduce 5 Damage(will consume at least 1 Damage) form non-player characters' attacking.
— **DeathKnight**: a subclasses of pc, it represent race deathknight and it has a unique ctor of pc in its own ctor. It has a string field type and a method return its type. I will have one time resurgence when player character died and restores 100 HP.

NPC(subclass of info):

- NPC is a class which store all attributes of any type of non-player characters, and NPC is a decorator class for all non-player character classes.
- NPC has integer fields(such as atk, def, hp and etc), methods that return current npc's atk, def and other attributes.
- NPC is a decorator for all non-player character classes.

(All following classes are subclasses of npc)

— **human**: a subclass of npc, it represent human race and it has a unique ctor of npc in its own ctor. It has a string field type and a method return its type. Human will drop two piles of golds on map when human dead.(Two treasure classes will be initialized and the human class will be deleted)

— **orc**: a subclass of npc, it represent orc race and it has a unique ctor of npc in its own ctor. It has a string field type and a method return its type. Orc will have 50% extra damage on goblin.
— **dwarf**: a subclass of npc, it represent dwarf race and it has a unique ctor of npc in its own ctor. It has a string field type and a method return its type. Vampire will lose 5HP each turn when attacking dwarf.
— **elf**: a subclass of npc, it represent elf race and it has a unique ctor of npc in its own ctor. It has a string field type and a method return its type. Elf will attack all player character twice except drow.
— **merchant**: a subclass of npc, it represent merchant race and it has a unique ctor of npc in its own ctor. It has a string field type and a method return its type. Merchant's has its own static integer field which is called hostile. This field determine whether merchant will be hostile to player character. When merchant class be removed, a treasure class will be initialized on theFloor.
— **Dragon**: a subclass of npc, it represent dragon race and it has a unique ctor of npc in its own ctor. It has a string field type and a method return its type. Dragon won't be initialized singly. It will be initialized when a special treasure class is initialized. Dragon class has a filed to store the pointer of the treasure class which is protected by the dragon.
— **Halfling**: a subclass of npc, it represent halfing race and it has a unique ctor of npc in its own ctor. It has a string field type and a method return its type. When player character call the attacknpc method to this Non-player character, there is 50% chance miss(deal 0 damage).
— **Boss**: a subclass of npc, it represent Boss race and it has a unique ctor of npc in its own ctor. It has a string field type and a method return its type. The ctor is related to the current floor number and the boss will be more stronger when the floor is higher. In the bonus part, after the floor boss be removed, then the stair to next floor will be initialized.

Potion(subclass of info):

- Potion is a class store the type and methods that set or change type of all potions.
- Potions is a decorator of all specific potions.

(All following classes are subclasses of potion)

— **PH**: the player character will lose 10 HP if PC call use potion method with PH class.
— **RH**: the player character will restore 10 HP if pc call use potion method with RH class.
— **BA**: the player character will increase 10 atk when pc call use potions method with this class.
— **BD**: the player character will increase 10 def when pc call use potion method with this class.
— **WA**: the player character will reduce 10 atk if pc call potion using method with WA class.
— **WD**: the player character will reduce 10 def when pc call potion using method with WD class.

Treasure(subclass of info):

- Treasure class are the piles of gold which located on the game map
- Treasure class store an integer field that is the value of the specific pile of gold and methods that set or return the value of treasure class

(All following classes are subclasses of treasure)

— **smallT:** smallT is the class which represent the small pile go gold and its value field is 1.
— **humanT:** humanT is the class which represent the normal pile of gold and its value filed is 2. When a human class be removed, 2 humanT class will be initialized.

— **merchantT:** merchantT is the class which represent the pile of gold which has value filed is 4. When a merchant class be removed, a new merchantT class will be initialized.
— **dragonT:** dragonT is the class which represent the pile of gold which is a dragon hoard. Its value field is 6. It has a pointer of dragon field which is called protector and it has methods to call its protector attack the player character when pc nears it.

Skill:

- skill is a class that represent the ability of play character. All player character has a filed skillets which is a vector of skill.
- skill store the name of the skill and have method that add new skill or check whether the consume string is the name of one skill in the skilllist.

(All following classes are subclasses of skill)

— **shadenative:** the talent skill that shade race have, have 50% extra score.
— **drownative:** the talent skill that drow race have, have 50% extra effect on medicine.
— **trollnative:** the talent skill that troll race have, restore 5 HP at the end of each turn.
— **goblinnative:** the talent skill that goblin has, steal 5 golds from each attacking.
— **vampirenative:** the talent skill that vampire has, gain 5 HP when attack npc except dwarf.
— **deathknightnative:** the talent skill that death knight has, it will resurgence one time if pc first died and restore 100 HP.
— **sabernative:** the talent skill that sober class have, use invisible air to reduce at most 5 Damage and will consume at least 1 Damage.
— **luckyseven:** the skill that shade can learn and has 1/7 chance to kill a npc imdiately and have 1/7 chance to pick 7 times gold.
— **poisonbody:** the npc will lose 5 HP if it attack the pc which have this skill
— **grow:** the player character will increase 1-5 atk or def by killing a npc.
— **handofMidas:** when pc pick a pile gold, it will create a pile of gold with the same value.
— **bloodrage:** pc will restore 30%(up to 50HP) of HP that the npc just be killed by pc.
— **excalibur:** pc will have 25% chance to deals a triple damage to a npc and restore the same amount HP.
— **Frostmourne:** when pc attack the npc, ignore the def that npc have, treat it as 0 def.

2 Explanation of Patterns

Observer Pattern:

- used for dragon and dragon treasure, if pc moves near the dragon treasure, dragon treasure will notify the dragon to attack pc who near the dragon hoard
- used for merchants, if pc attack merchant, then single merchant will notify all merchants to be hostile to pc.

Strategy Pattern:

- used for member method usspotion in pc class, if the player character use a potion, it will check the potion type, then choose the correct strategy which represent the effect of the potion

Visitor Pattern:

- used for member method attack npc and npc attack in pc class, when some specific npc will attack different races pc, they will call specific npc attack to attack player character.

Decortor Pattern:

- used for pc class, potion class, npc class, skill class and treasure class, if we want to add a new player character, let it be the subclass of pc. Then, the ctor of pc will let the new race be created quite easy. Similarly, npc, skill, potion and treasure are all decorator and they are helpful for creating new type of potion, non-player character, pile of gold and player skill ability.

3 Changing from Due Day 1

1. Use strategy pattern for use potion method in pc class.
2. Add several new subclass of pc and npc for achieving the bonus part. Deathknight and sober class are new subclass of pc which are new race of player character. Boss is the new subclass of npc. New decorator class skill is created and it will show the skill ability of each player character race. 14 subclasses of skill are added since each player character race has 2 skills.

4 Description of Strategies we used in our Project

For our whole project, we divided it into three parts. The first part is floor and cell part. The second part is info part, and the last part is control part.

The floor and cell part has two classes which are floor and cell. Cell is the basic component which composes the floor. There is a field theFloor which is a vector<vector<Cell>> is the whole floor of our game. The floor class will be responsible for display the floor and call the method in cell to do the commands in the game. Cell is the basic component and each cell has the field curinfo which is storing a pointer of info which represent the object which is located on this cell. Since the info class can return the type of the object(will discuss below), thus, each Cell can check the type of the object and print the char which represent the the object. Therefore, the way floor class print the whole floor is just use for loop to print each Cell's curinfo. Cell also have a field move to represent the whether object in the cell has moved. This is used for all non-player character classes except dragon, since they will auto move or auto attack player character each turn.

The info part is the class which represent all the objects in the game, include player, non-player characters, potions and treasures. Info is an abstract class and only have a method that will return the subclass' type. Pc, npc, potion and treasure are four different subclasses under the info class. These four classes are all decorators and storing the common information of their subclasses. Human, merchant and other enemies are subclasses of npc and these classes will only store the enemy type. Similarly, vampire, drow and other player character are subclasses of pc. RH, PH and all other potions are subclasses of potion, and dragon hoard, merchant hoard are subclasses of treasure. All the bottom class will only store their type. The methods of player characters attack non-player characters or the methods of player character use potions are all in pc, potion, npc, treasure. We design like this is because we think all types player character are able to attack all types non-player characters. We do not need to write them in each play characters' class because that need us to implement a lot of repeated

methods in each subclasses. Thus, we summarize these methods and implement them in decorator classes.

The last part is the control part. This part only have one class which called control. This part has no any classes and responsible for the logic in our game. This part only have several help functions. For example, the function command is reading command through stdin and call floor's method to move player character or attack non-player characters. It also have the function movenpc, and this function will move non-player character through their auto move each turn. Besides this, after we initial a floor in the main function, we need to initial player character, non-player characters, potions and treasures, and all of these initializing functions are all in control part. Mapping reading function is also in the control part as well.

5 Time Management

For time management, we deciding to finish the the floor/cell and info parts of project in first week. Then we will implement the control part and bonus part in the second week.

Nov 19 ~ Nov 24:

We need to write our plan of attack and UML first. Then, we want to finish the implementation of floor/cell part and info part before Nov.24.

In specific, we will discussed together at Nov 19 about the plan of attack and finished the UML at Nov 20. Then we will use 3 days to implement two parts individually and combine the two parts together and debugging after combining them together at Nov 24.

Subtasks:
floor/cell part (Zhu Yandong)
info part (Cheng Qiwei)

Nov 25 ~ Dec 2:

We will implement the control part and bonus part during the second week.

In specific, we plan to implement the move, attack, potion using command and command reading functions and all initializing functions in Nov 25. We will implement the npc auto move and auto attack logic function in Nov 26. In Nov 27, we will implement the bool function to decide whether player will lose and function about go to next floor through stairs and the function will remove the death enemies on the floor. In Nov 28, we will implement the map reading function and staring bonus part. The whole second week will used for bonus part in plan. If we haven't implement the basic part before the second week, we also can use the second week to continue the part we do not finished in the first week.

6 Questions and Answers For Due Day 1

Question1: 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?

Answer:

As we discussed in the first part of our plan, we use info class as our superclass. Info class only has one method which called return type. The method will return the type of the object. Then, class pc is the way we generate each race. Pc is one of the subclass of class info. Pc is also a decorator for all the player characters. Pc class has storing all common attributes

that all player characters have, such as atk, def, hp and gold. Also, we write all the common methods which all player characters should have in pc class, such as attack non-player character classes or using potions. Thus, Pc classes has all attributes and methods of player character. After implement pc class, the last part is the specific player character classes, such as vampire class, shade class and etc. These classes are subclass of the decorator (which is pc class). The ctor of these class only consume a string which is their type name and initialize a pc. The superclass info can return these classes' type.

Our method is very easy to add a new race. The new race class will be subclass of our pc class. We just need write a new class with the new race's name. The new class will store a string which is its type and a method which return its type. The ctor of the new class is its type and a pc. All other attributes will be set when ctor initial its pc part, and the new class will be able to attack or use potions since it is a subclass of pc class(pc is decorator and it has attack and potion using methods).

Question2: How does your system handle generating different enemies? Is it different from how you generate the player character? Why or why not?

Answer:

It is possible that we use the same method to add non-player characters in our pc class, since enemies have similar attributes. We could just make all enemy classes be subclass of pc and use pc as decorator again. Even through it is possible, our group had discussed seriously and denied this method. We want to build a new decorator only for non-player characters. Thus, it is different with generating player characters because we use another decorator.

The main reason we write another decorator is because if we use same decorator for both player character and non-player character. Then, if we add attack method and potion using method to the decorator, the non-player character will be able to attack themselves and using potions which are not allowed in the game logic. Even we won't call non-player characters to use these methods, but we won't know what will actually happen when we implementing the game logic. Thus, we decide not to write these dangerous method that may increase the difficulty of debugging game logic part.

Question3: How could you implement the various abilities for the enemy characters? Do you use the same techniques as for the player character races? Explain.

Answer:

we use conditional logic to add enemies and races' skills. If vampire will use skills to attack dwarf, then in the method attackinpc in pc class, we will add a condition like **if((this->type == "vampire") && (npc->type == "dwarf"))** then add what will happen in the condition. Similarly, if some race a can get double gold after kill some enemy, we will write code like **if(type == a){ get double gold;}**. That will be our way to implement all skills of all player character and non-player characters.

It could be a bit slow because we will check the type when we attack or be attack every time, but since each race or enemy has different skills. We must add these conditional part to check the type. Thus, I don't think we need to add extra class for skills and extra skill classes will increase the efficiency of our code.

Question4: The Decorator and Strategy 8 patterns are possible candidates to model the effects of potions, so that we do not need to explicitly track 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.

Answer:

decorator is a design pattern that wrap new methods on the original methods and it can wrap any number of times if you want. strategy pattern is choose one of the n strategies which you need to implement. Both patterns are fine to work. I think decorator will be better if we need to use two potions in one turn, but the game logic does not allow us to do this. Strategy pattern also works because we just choose the effect by checking the potions' type. I think they all work and no one is better than the other, but I will choose decorator since decorator may work if player consume several potions in one turn and it may be used in bonus part(DLC).

Question5: 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:

As I mentioned before, we have a superclass info. Potion and Treasure are subclass of info and specific potions and treasures(such as dragon hoard) are subclasses of Potion and Treasure. Thus, info can return the type of potion and treasure for printing on the floor. The method of return type can be reused several times whatever it is a potion or a treasure.

7 Questions and Answers For Due Day 2

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:

This project teaches us that we need to separate our tasks clearly. If we do not separate our tasks, then it is common that one person is writing and the other person is waiting and have nothing to implement. separate the whole project in different modules, and both team members can be responsible for some module that do not need other module to teat. Each member implement their own part of modules and test them by themselves. Then, after every member finishing implementation, they can put their work together and that will be much more effective than one person coding, if he finished, the other person continue coding on previous one's code.

Documentation and discussion are very important for team work. If one member do not write documentation, other member must spend a lot of time to read his code and that will waste a lot of time. Thus, every member must write clarify documentation. Discussion among members is also important, every member need to know their partners want what type of input and output for their modules' methods. If the team is lack of discussion, each member will need to waste extra time for changing their code to achieve their members' requirements.

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

Answer:

We will read the requirements on pdf file more carefully to check the requirements before we start coding but not after we finish coding.

We will double check each members' requirements for their parts' functions' input and output type requirements, but not changing methods after finish implement them.