University of Waterloo

CS246 Final Project - CC3K
Plan of attack

Kai Sun

Yuan Zhong

Ao Liu

August 4, 2020

PLAN OF ATTACK

	Tasks	Responsible	Estimated Time	Current Status
1	Draw UML	Kai	5 hrs	Finished
2	Plan of Attack	Ao	2 hrs	Finished
3	Implement Header files	Yuan, Ao	2 hrs	Unfinished
4	Implement Floor.cc Chamber.cc TextDisplay.cc	Yuan, Kai	7 hrs	Unfinished
5	Implement State.cc Item.cc Character.cc	Kai	5 hrs	Unfinished
6	Implement Player.cc Enemy.cc (include derived classes)	Ao	8 hrs	Unfinished
7	Implement Potion.cc Treasure.cc (include derived classes)	Ao, Kai	8 hrs	Unfinished
8	Implement Buff.cc main.cc	Ao	3 hrs	Unfinished
9	Debugging	Kai	10 hrs	Unfinished
10	Testing	Yuan	3 hrs	Unfinished
11	Implement graphics	Kai	1 hr	Unfinished

Summary:

When designing the program, we are planning to use smart pointers to avoid memory leak. Also, as shared_ptr has use_count method which would be useful for debugging. However, when cyclic references case comes up, we would switch back to normal * pointers to avoid this fundamental flaw.

When we design the main part of the program, there are three essential classes which are the core of this game. They are Floor class, Character class and Item class. Based on the UML we draw, the floor class consists of the player, next floor, enemy, items, Chambers and textdisplay which is used to display the map and current floor.

Our design is that each Floor has a composition relationship with all Items, Chambers, Enemies and Textdisplay where Floor has an aggregation relationship with Player.

The Item will be the abstract class of Potion and Treasure, containing the number which represents potion effect for Potion and the gold number for Treasures. In this way, we can improve our project design structure by having less reuse code.

The Character class is an abstract class of Player and Enemy and they have common fields which are health, attack, defense. Each Player and Enemy also has their own special ability which will be activated during the combat.

Each Player has a field Buff which is like bonus attack or defense for the player in the current floor. When the player picks up the potion, it will add the potion effect to the Buff based on which kind of potion it is. At the end of each floor, since attack potion and defense potion only last in the current floor, Buff will reset its attack field and defense field to zero except health.

We designed the TextDisplay class to display the game when the game starts, and the player enters the floor. The current game information including floor, chamber, characters, etc. will be printed parallelly.

Questions:

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

Answer:

We can use inheritance to design the system as each race has common fields which are Health Points (HP), Attack (Atk) and Defense (Def). The Factory Method design pattern provides an interface for object creation, but lets the subclasses decide which object to create. Let Character be a superclass. We can make new races as the subclass to inherit the common methods and fields, so it is easy to add additional races.

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

Answer:

By making Character as a superclass and Race, Enemy as subclasses of Character, it would be similarly to generate different enemies compared to generating the player character since each enemy also has common fields.

3. 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:

As attacks on different enemies will trigger different abilities on enemies, we could use visitor pattern to implement attack which is an overloading method. In a combat, our visitor can automatically recognize the player and the enemy by doubling dispatch and applying the desired effect to both player and enemy.

4. The Decorator and Strategy 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 discussing the advantages/disadvantages of the two patterns.

Answer:

I think the Decorator pattern would work better for consuming potions by the player character on each floor. When potion is used by a player, positive or negative bonuses on HP, ATK, or Def will be applied to the player by applying potion as decorator. At the end of each floor, we could undecorate potions which had temporary effects. With a decorator pattern, potions can be easily attached or detached to the player character. The drawback is that it will result in many little objects which look similar. Strategy pattern is harder for detaching when at the end of each floor.

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:

We can make Item as an abstract class, so Treasure and Potion are subclasses which inherit from Item. Since Treasure represents the number of gold and Potion represents the number of different effects, thus, they both have a common field which represents number, thus we can let our abstract Item class contain a protected number data field in order to avoid duplicate code.