Chamber Crawler 3000

Design Documentation

Created by Jing and Yue
Apr. 6th 2015

Overview

This design documentation contains:

- Explanation of Clases
- Explanation of Design Pattern
- Changes from due day1
- Attack plan of due day1 and actual completion time

Bonus parts added in cc3k:

- skills, different races and classes have different skills, each skill have differnt effect
- classes, and can advanced to ultimate classes
- Bosses
- inventory system, can pick, drop
- Buy and Sell from merchant
- Rune (a new type of item that will have different effect if it is kept in inventory)
- enemies abilites
- Ncurses

Explanation of Classes:

Enemy:

- This is an abstract superclass of enemies in the game cc3k(Dragon, Goblin, Merchant, Phoenix, Troll, Werewolf, Vampire).
- It contains protected fields of general attributes of enemies.
- It contains public methods such as getters and setters to reach its fields and makes the change to the field controllable. It also contains some virtual methods to realize specialization of different kinds of enemies.

(The following classes are subclasses of class Enemy)

Dragon:

- This is a concrete subclass of class Enemy
- It contains a field isHostile used to control whether the dragon is hostile.(player is in 1 radius of dragonhoard) It also contains a field t, which is a pointer to the dragonhoard
- It contains public methods (setters and getters) to reach and change its fields

- Special: Its name is Dragon and we use ‘D’ to represent it

Goblin:

- This is a concrete subclass of class Enemy
- Special: Its name is Goblin and we use ‘N’ to represent it

Merchant:

- This is a concrete subclass of class Enemy
- It contains a static field hostile, which is used to control whether the merchant is hostile(once the player attack merchant, all merchants become hostile to player)
- Special: Its name is Merchant and we use ‘M’ to represent it, and it will attack the player only when it is hostile.

Phoenix:

- This is a concrete subclass of class Enemy
- Special: Its name is Phoenix and we use ‘X’ to represent it. It increase its own Atk when it deals damage player

Troll:

- This is a concrete subclass of class Enemy
- Special: Its name is Troll and we use ‘T’ to represent it. It increase its own Def when it deals damage to player

Werewolf:

- This is a concrete subclass of class Enemy
- Special: Its name is Werewolf and we use ‘W’ to represent it.

Vampire:

- This is a concrete subclass of class Enemy
- Special: Its name is Vampire and we use ‘V’ to represent it. It increase its own HP by the amount the damage it deals to player

Item:

- This is a subclass of class Object and is a superclass of class Potion Treasure and Rune
- It contains protected field amount of general attributes of enemies.
- It contains public non virtual method to reach its amount and many virtual method to realize specialization of different kinds of items

Potion:

- This is a subclass of class Object and is a superclass of class Potion Treasure and Rune
- It contains protected field amount of general attributes of enemies.

- It contains public non virtual method to reach its amount and many virtual method to realize specialization of different kinds of items

    A: We use it to implement all attributes of potion Atk such as generate amount, and its name is determined by the amount generated. Also add control in useItem to realize that only elf can have a positive effect on its Atk even it has a negative effect on all other races

    D: We use it to implement all attributes of potion Def such as generate amount, and its name is determined by the amount generated. Also add control in useItem to realize that only elf can have a positive effect on its Def even it has a negative effect on all other races

    H: We use it to implement all attributes of potion HP such as generate amount, and its name is determined by the amount generated. Also add control in useItem to realize that only elf can have a positive effect on its HP even it has a negative effect on all other races

Treasure:

- This is a concrete subclass of class Item
- It contains private field PickState to control whether the treasure can be picked up it is used for dragon hoard. It also contains a field dragon which is a pointer points to the dragon
- It contains getters and setters to reach its private fields
- Special: Its name depend on the amount generated and we use ‘G’ to represent it.

Floor:

- We use it to generate maps and objects and display activities happend on every floor.
- It owns class Cell as the Cell of two D array in the private field of Floor
- It contains private fields used to record the objects and their number on each floor.
- It has class Treasure, Item, Enemy, Player.
- It contains public methods which has getters and setters to its private field and the method used to controll related activities on each floor. Some methods also responsible for the display.

Cell:

- It records things in each cell on map, every cell in the map can either be an Object or the corresponding cell in map

- It contains private fields structure(record the character in map at corresponding position), obj(pointer to an Object), label(the chamber number)
- It contains public methods of setters and getters to reach and change its private fields.

Object:

- It is a abstract super class of class Enemy, Item and Boss.
- It records objects positions, objects includes enemies, items and bosses.
- It contains private fields r and c, which used to record the position of objects.
- It contains public non virtual methods to setters and getters to coordinates.
- It contains public virtual methods of setters and getters to realize specialization of different kinds of objects.

Rune:

- A derived class of Item. It contains a public field called pickRune and dropRune. The effect of Rune will be on as soon as player picks it and keep it in inventory. After player drop the Rune, the effect of this rune will lost its effect on player.

Boss:

- A derived class of Enemy.
- Implement the attack method and can attack player if player is in 2 radius of the Boss.

player:

- This is a abstract superclass of class race.
- It is the only class that main function will uses to proceed game and give instructions using public methods of player class.
- Player contains an array of floor, which have the status of enemies, item, treasures, maps of each floor.
- This player class contains all the private fields attributes and data of the player character in the game such as HP, MaxHP, Atk and Def.
- Player contains a classes to pointer, which is initially set to NULL. Later when the player levels up and reaches a specific level. the player will be able to transfer to some classes(Mage/Knight/Warrior), or advance to some ultimate classes.
-

Race:

- This is a superclass of each different races, which is Orc, Elves, Dwarf and human. As soon as the game starts, the computer will ask the player to select a race for PC. Then an object of the given races will be created.
- Race contain a Skill pointer, player can use useRaceSkill method to use the racial skill. Different races are implemented to have unique racial skills.

Orc:

- This is a concrete subclass of class Race
- Able to use skill to increase its attack for a few turns.
- Gold is worth half

Elves:

- This is a concrete subclass of class Race
- Can use skills to dodge all the attacks from enemies for very short time
- All negative portions have postive effect on Elves

Human:

- This is a concrete subclass of class Race
- Use skill to increase a bit health.
- the final score of human will increase by fifty percent

Dwarf:

- This is a concrete subclass of class Race
- Use skill to increase Deffence for a short time
- the final score of human will increase by fifty percent

Classes:

- It is a classes which contained by player class and indicates which class is the player. Player are able to choose their classes when acheiving
- Its private fields contains two Skill pointers. If player have a classes, they can use classes skill by calling public method of classes.
- This is a abastrack super class. It has derived concrete classes are Magte(magic class with MP), Warrior and Knigh and a abstract abstract class ultimateClass, where we uses decorator pattern on. Each basic class have a different class skills

Mage:

- a derived class of classes.
- Contains a private field fo MP. each Mage skills consumes some MP. Skills can be used only when the the player have

UltimateClass:

- It is a abstract superclass of BloodMage, SpiritHealer, DeathKnigh, Pladin, Berserker and Ranger. Every basic class can advance to two different UltimatClasses(eg, Mage can advance to BloodMage and SpiritHealer)
- When player reach level 15, they are able to advance to a ulitmate class, where we employ a visitor pattern here.
- UltimateClass have one private field of Skill pointer and player can uses ulimate skill by calling useUskill

Skills:

- have private fields contains the name and skills description
- overload operate<< to ouput skill name and description in a proper format
- each derived skills have different effect on enemies or playerself.
    - BolldFury: increase Atk for a while
    - ShdowMeld: dodge all attack for a short time
    - StroneFome: increase Def for a period
    - HumanSpirit: increase a little HP
    - Whirlwind: attack all enmies in one radius
    - DemoralizingShout: decrease the Atck of sorounding enemies
    - FrostNova: deal little damage to enemies within 2 radius and freeze them
    - SpiritualHealing: increase player HP
    - Slam: deal huge damage to an enemy in 1 radius
    - FireBlast: deal damage to one enemy in fron up to 5 blocks away
    - BloodMagic: deal damage to enemies in 3 radius and increase player's HP
    - Vatality: increase player HP, Atk and Def
    - Flatten: deals damage and stun enemies within 1 radius
    - DeathCurse: decease Atk and Def of enemies within 3 radius
    - Blessing:increase player HP, Atk and Def

Explanation of Patterns:

Observer Pattern:

- Used among player, enemies and floor
- When player moves or enemies moves, they notifies the floor. floor will notify the cells the move the objects or characters to the given cells.
- In each turnm, floor will check the distance between player and each enemies, and give enemies will behave differently depends on the distance between the player and enemy.

Factorial Pattern:

- Used among player, floor and enemies
- When player reaches a new floor, player will generate a new floor if this floor is not been visited. Then the floor will generate enemies, whoes strength depends on the level of floor.

Visitor Pattern:

- Used between Classes and ulitimateClass.
- When player when the player reaches lv15, they are able to evolve to a ultimateClass. And the options of ultimateClasses are depends on the player's baisc class. Thus, the player class will call classes::evolve() (at player.cc:363), which is a virtual method and it is also implemented in all derived classes(Mage, Warrior, Knight)
- In each derived classes, in the function body, it calls [player:: evolve](eg. mage.cc:33) and passes this pointer as a parameter. Thus, in compile time, the function knows which derived classes is been passed as the paremeter.
- Then there are three overrided functions of [player::evolve], each of them comsume a different derived class of classes and gives differnet options of ultimateClass can be advanced.

Decorator Pattern:

- used on classes and ultimateClasses, both are abstract classes.
- ultimateClasses are derived class of classes and it contains a pointer to classes.
- Each derived classes of ultamateClass(eg. BloodMage DeathKnigh) are a decorator to the basic classes.
- When a player advances fron basic classes to ultiamteClass, the basic classes will be pointed by the classes pointer in the ultimageClass and all the basics classes methods are implemented. Thus, the player can uses basic class skills and ulitmateClass skill.

Changes(update) on due date one:

We have few changes from due date one:

- add Visitor Pattern and Decorator pattern
- did not finish some of the challenging parts

CS246 Final Project — CC3K Plan				
Steps	Content	Completion Date		Goal
Preparation	UML	Mar 22 2015	Jing, Yue	Make basic plans and design patterns
	Plan of Attack	Mar 24 2015	Yue	
Core Parts	main	Mar 23 2015	Jing, Yue	Generate maps and player to realize basic moves.
	Class:			
Core Parts	Floor	Mar 24 2015	Jing, Yue	
	Player	Mar 24 2015	Jing	
Basic Parts	Object	Mar 24 2015	Yue	Provide different races for player to choose, and randomly generate enemies and items on maps
	Enemy	Mar 25 2015	Yue, Jing	
	Enemy derived classes	Mar 25 2015	Yue	
	Race	Mar 25 2015	Jing	
	Item	Mar 26 2015	Yue	
	Item derived classes	Mar 28 2015	Yue	
Advanced Parts	Classes	Mar 26 2015	Jing	Add new features, make the game more fun. For example, add class system and inventory system, design and implement unique skills for different classes and races, including AOE effect and BUFF effect.
	Classes derived class	Mar 30 2015	Jing	
	Skills	Mar 28 2015	Jing	
	Skills derived classes	Mar 28 2015	Jing	
	Runes	Apr. 5 2015	Yue	
	Runes derived class	Apr. 5 2015	Yue	
	Boss	Mar 29 2015	Jing, Yue	
	Enemies abilities	Apr 30 2015	Yue Jing	
Challenging Parts	ncurses	Mar 29-31 2015	Jing, Yue	If time is allowed, we will try to complete the challenging part, like time system, save game and graphics
	Radom generated Map	Mar 28 2015	Jing,	

Questions (Due Day1)

2.1 Player Character

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

Answer: Each Race class is a derived class of the player class. We put the different fields between races in each Race class. Each time we assign the player to a race by constructing a new race class. It is easily to add new races to the game, because it does not touch the player and other basic part of the game.

2.2 Enemies

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

Answer: The enemies are generated when every floor is generated. However, the player is generated before the floor generated, right after the races chosen. We generate each enemy by calling new enemy class and set the coordinates on the map. For player, we just randomly pick the coordinates where it spawn.

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.?

Answer: In Enemy Class, we build a method called useSkill(), and set it to virtual. We implement this method in its derived classes of Enemy Class which are the classes of different enemies, hence in this way we have the flexibility to manipulate different skills according to different characters of these enemies.

2.3 Items

2.3.1 Potions

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?

Answer: Observer Pattern. Player has a field to a class Buff, and Buff has a method called Effect and an integer field called time . Every turn the player will check the time of Effect. If the time is non zero then execute the Effect method (i.e. poison, boost attack, if the time is 0, set the buff pointer to NULL)

2.3.2 Treasure

Question. 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 build a class Item as the super class and let class Potion and Treasure be its subclass, so that they can share some common methods and fields such as the integer field amount, and public method getAmount and getType.

Questions (Due day 2)

1. What lessons did this project teach you about developing software in teams?

- Do not underestimate the difficulty of a giant project. Before writing the code, we thought it can be done within 30-50 files with most of the bonus parts added. However, we acctually wrote more than 80 files to finish the huge project including the bonus part.
- It takes huge amount of time to compile and run the game for the first time. Our first compile and run is after the floor, player and some classes that is contained by floor and player is finished. It takes half a day to debug approximate 20 files. Thus, if we can derived the project into smaller parts, we may reduce the time on debugging
- Team work is essential in group developing software. While we are writing the code, we have one person working on the main (core) part of the project and another person working on some modules that will not change the main part. Then we will not change cour part of the program and make a mess.
- Have vision. Always trying to implement the classes, moduls in a way that is easy for teammates to modify or add new features.
- Communicatte and listen to others opinion. If the teammates does not agree on some idea, it would be better to communicate more and figure out a better to do it. Otherwise, the project will be full of bugs and hard to combine two people's work.

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

- We will read guildline, given document and examples more carefully so that we will not have a last-minute change before the due time.
- Spending more time on planning and structuring each classes and the relationship between classes instead of rushing to the coding part.