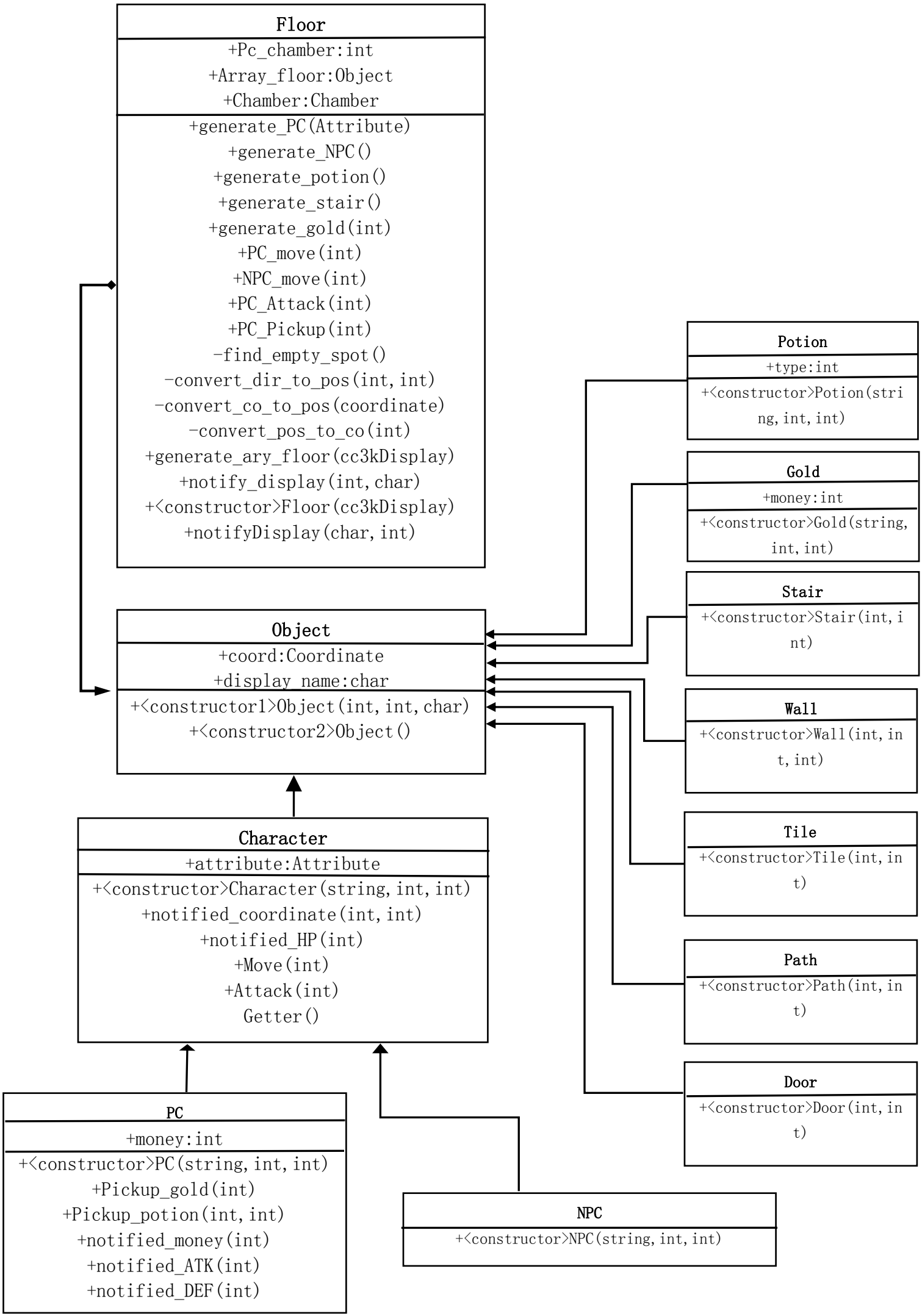
CS246
Assignment 5

Jinchao Lin (j68lin)
Zijie Zhang (z289zhan)

Plan of Attack
Of
The Game of ChamberCrawler3000


Floor
+Pc_chamber:int
+Array_floor:Object
+Chamber:Chamber
+generate_PC(Attribute)
+generate_NPC()
+generate_potion()
+generate_stair()
+generate_gold(int)
+PC_move(int)
+NPC_move(int)
+PC_Attack(int)
+PC_Pickup(int)
-find_empty_spot()
-convert_dir_to_pos(int,int)
-convert_co_to_pos(coordinate)
-convert_pos_to_co(int)
+generate_ary_floor(cc3kDisplay)
+notify_display(int,char)
+<constructor>Floor(cc3kDisplay)
+notifyDisplay(char,int)
Object
+coord:Coordinate
+display_name:char
+<constructor1>Object(int,int,char)
+<constructor2>Object()
Character
+attribute:Attribute
+<constructor>Character(string,int,int)
+notified_coordinate(int,int)
+notified_HP(int)
+Move(int)
+Attack(int)
Getter()
PC
+money:int
+<constructor>PC(string,int,int)
+Pickup_gold(int)
+Pickup_potion(int,int)
+notified_money(int)
+notified_ATK(int)
+notified_DEF(int)
NPC
+<constructor>NPC(string,int,int)
Potion
+type:int
+<constructor>Potion(string,int,int)
Gold
+money:int
+<constructor>Gold(string,int,int)
Stair
+<constructor>Stair(int,int)
Wall
+<constructor>Wall(int,int,int)
Tile
+<constructor>Tile(int,int)
Path
+<constructor>Path(int,int)
Door
+<constructor>Door(int,int)

Plan of Attack:

At the high level, we will use "MVC" design pattern to implement our program (Game of Chamber Crawler 3000). We plan to break our code into four separate compiled file, which are floor.(h.cc), object.(h, cc), display.(h, cc) and control.(h,cc).

The objects inside display file will play the role of "Viewer" in the "MVC" pattern. The objects inside floor file and object file will play the role of "Model" in the "MVC" pattern. The objects inside control file will play the role of "Controller" in the "MVC" pattern. The controller will interact with standard input to accept user command and transfer the user command into corresponding function call. In our program, the controller will be solely interact with floor object, which will lives inside floor file. The floor file plays a central role in our program, which is responsible to accept controller's command, make corresponding change for itself and associate objects, and notify "Viewer" (in our program it is display object) with any necessary change.

In term of responsibility, the floor will again play the central role in our program. It's the place where all cell (wall, path, door, tail) and character are stored. It's also responsible to interpret any command input from the controller and check the command's feasibility (for example, make sure the PC's moving is at the within valid space). If a command pass the feasibility check, the floor will make corresponding method call to change the object's status, and post the change to the display (Viewer).

The program has built-in default map. But it also has the option to take map as input file from outside. The display object will have corresponding constructor to build display object from outside map. We then build the corresponding floor object based on the display object using floor's constructor.

In term of data storage, all elements are stored in an array in both floor object and display object. Inside floor object, the array has type object, which is the base class of any character and cell and items. Inside the display object, the array has type char, which

is a symbolic representation of corresponding object. There will be mapping functions to map the coordinate from a 2d board into the position in a 1d array and the other way around. There is “pretty_print” method inside display object to print the 1d array to stdout as a nice game board.

For the enhancement, we consider to add “weapons” as a buffer to the characters. NPC will carry weapon will certain probability. PC will get the weapon when they slay NPC. The PC will also have the ability to carry weapon, up to a limit. The weapon will increase PC’s attributes, and grant PC with special combat skill. We will also add special ability to certain NPC, like goblin will has the money stealing ability, the vampire will have the HP stealing ability. We plan to implement these features using the Visitor design pattern. We may also consider using decorator pattern to add more add-on functionality to PC. We also consider giving PC with recover ability. The PC will gradually recover their lost HP in certain number of steps.

Breakdown of project:

We will keep doing pair coding in our program, therefore both of us will know the big picture as well as the implementing detail of the program.

To make debugging easier, we will first build the display class and controller class, and therefore we can have input and output ability. We plan to finish controller and display class before Thursday (July 24). We will then build our “Model” gradually; hopefully we will finish the main part of project before Saturday (July 26).

Answer to the Question in the list:

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?

Answer: We plan to create a struct called “Attribute” to store all Character’s HP, ATK, DEF, and race string. We will use this to store information systemically for both

PC and NPC. We will also create constant Attribute struct to store the race information for each races. In this case, to add class, we need to make change of the Attribute's fields. This task may be easy or not depends on if we are adding new fields. We can also consider to use decorator pattern to implement class, which is involved some modification to the Character class to create a base interface.

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

Answer: We do this by two steps. First we call function to decide which race of enemy we should generate based on given probability. Second we call find location function to decide where should we put the enemy. In second step, we first decide which chamber to put our enemy with equal probability, then we decide exactly in which tail we should put our enemy.

PC's race is determined by player, and therefore won't have the first step as we generate NPC. Beside this, PC is the first object to be generated. Therefore its location is easy to determine. In generating NPC, because there are already some objects in the tails, we may fail to find a suitable location at the first try and therefore need to call find location function many times to find a good spot.

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: Since each enemy is implemented as an independent class, we can simply add new method to the class to make the NPC has special ability. In the combat, the NPC's method will call notify function inside PC to make change of PC's status.

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: We don't use any design pattern. After we get into a new floor, we simply call reset method in floor object to reset the PC's ATK and DEF.

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 do not duplicate code?

Answer: we will implement a common helper function called "find_empty_spot()". This function will be called by any methods that need to randomly generate any object, including Character, Potion and treasures and stairs. Of course, the generator of each objects will then check if the "find_empyt_spot()" function's return value is usable. If not, call it again, until we get a usable address. Therefore we won't have duplicate code.