CC3k Design Document

Board

There are two separate methods within my Controller class that create the View/Board for cc3k.

- If a *map file is passed into the program as an argument* then I just store the contents of the file in a 2D vector. I then go through this 2D array and I create and push the enemies (depending on their type based on the character in the array) into an Enemies vector. I did the same thing for items. The items and enemy vectors are private fields of controller. Once the board is done being processed, I set the view's board (a 2D vector) to be the one just processed so that the controller would just need to notify the view to update when needed. When a dragon is read from the file, we search all the floor tiles around it to find it's associated dragon hoard and set the dragon hoard field of this Dragon. This is so that the dragon hoard can be picked up when the dragon dies and so that the dragon can attack the player when the player is beside the dragon hoard.
- For *random generation* of items, I had a template_map.txt file which just contained an empty 79 by 25 board. I had two 2D vectors – board and tempBoard. tempBoard would iterate through the template_map file and find all the chambers. It sets up each chamber with all of the points on the map associated with it. We then use this tempBoard and generate the Player's location, save its chamber number and generate the stairs location (making sure it isn't in the same chamber as the player). Then potions are generated. First, we randomly choose one of the chambers and then choose a random point within

the chamber. Then, I had a 'temporary' treasure array that just contained a list of 8 treasures. I organized the array to contain the treasure hoards to match their spawn rate probabilities (i.e., 1/8 for dragon hoards, 5/8 normal hoards and 2/8 for small hoards). The array contains 1 treasure with a value of 6 (dragon hoard), 5 treasures with a value of 2 (normal hoard) and 2 treasures with a value of 1 (small hoard). I then randomly choose one of the chambers, and then a point in the chamber to place one of the randomly chosen treasures of this array onto the board. The generation of enemies is similar.

Characters

Character is an abstract class that has two child classes: Player and Enemy

- *Player* is an abstract class with concrete child classes Shade, Troll, Goblin, Drow and Vampire. When a player dies, it notifies the controller which ends the game.
- *Enemy* is an abstract class with concrete child classes Elf, Orcs, Merchant, Dragon, Human, Halfling, and Dwarf. They are created by random generation (explained above) if no map is specified as an argument to a program or they are placed onto the board at the specified location when reading from a map file. Whenever an enemy dies, it notifies the controller:
    - Merchant: tells controller to drop a merchant hoard where the merchant was when it was slain
    - Human: tells controller to drop a normal hoard where the human was standing and another normal hoard somewhere within the radius of where it was when slain

- Dragon: tells controller that it's dragon hoard can now be picked up
- All other enemies: tells controller to drop a normal or small hoard by random that is automatically picked up (added to the player's score) by the player.

I implemented the visitor design pattern between the player and enemy class. This is so that player's know what type of enemy attacked them and vice versa to simulate the "specialization" of each character.

Items

Items is an abstract class that has each type of potion and treasure as concrete child classes. I implemented the visitor design pattern between the player and items. This way, whenever any of the items are picked up, the player knows what was picked up exactly and update it's stats accordingly. Then player notifies the controller about the change.

- Treasure has a method called placeDependencies() that notifies the controller to place a dragon beside the treasure if it's amount field is 6 (i.e., it is a dragon hoard). This treasure's obtainable field will initially be false because it cannot be picked up unless the dragon is slain. All other treasures' obtainable field will be true.

Gameplay

Whenever the user makes a valid move, the controller updates the view (board which further updates the caption), player, enemies and items depending on what the user did.

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?

I worked alone. I learned that it is very important to plan out how you want to design your program (i.e., what design patterns to use, how you want to structure the hierarchy and etc) because it will be very hard to change once you've started implementing things. It is always a good idea to start with a UML and a plan of attack.

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

The overall design of my game did not change from my design on Due Date 1. But if I had the chance to start over I would have started implementing cc3k by first reading a board file and then create a dummy player and try to move it around and update the board accordingly. I would then slowly implement the characters, potions and gold and place them onto the board. As my first step, I set up the design patterns between the player and enemies and player and items and found that I ultimately ended up changing most of it later on. At first, I set up attacks between every player race and every enemy type just to find out that most of them were unnecessary. I ended up implementing the 'normal' attacks between Players and Enemies and then overloading what I needed to depending on the player and enemy specializations. Also, if I had the chance to start over, I would have chosen to work with a partner. I think I could have done better on the project and would have had more time to

implement other bonus features and most importantly I would have the opportunity to learn from a fellow classmate.