Jakob Tucker
CSCI 1300 Project 3

Project 3: Dungeon Escape Report

How did you prepare for the project?

I prepared for this project mostly by going over some of the basic requirements described in the GitHub. Being able to display a "status update" as well as utilize at least four different classes were some of the things that I based by project around (as I'm sure many other students also did). Finalizing my project though, I had realized I had missed one requirement, which was to implement an array of a class type of my creation into another class. I solved this problem by implementing a "Result" class, which basically took all the scoring attributes garnered in the game and added a bonus if the player defeated the sorcerer. This proved to be useful, however, as when the file is appended and read form when the player chooses to look at the leaderboard, I could easily sort the array of Result objects and display the names, and scores respectively. Preparing for this project also required me to write a lot of the high-level functionality out in the code. I separated the game out into three different phases, which was first to welcome the user into the game, ask for party member names, and display the initial merchant menu. Phase two consisted of a humungous do-while loop, in which a menu was displayed to the user to move, investigate a space, fight a monster, or give up. The third phase is the end of the game, which basically just appended the results file and asked the user for their name to add to the Result class object.

How did you develop our code skeleton? In what way(s) did you use your code skeleton?

If you guys are meaning "your" code skeleton, I basically implemented a bunch of setters and getters for each class. That was basically it. I kept all the classes simple, and in turn that made my main program very large and difficult to work with. The code skeleton, frankly, could have been more thoroughly implemented so that there is not as much code in the main file. For instance, in main, I have several functions that finds the index of each type of items available to the user in a vector. While that implementation in main works fine, it would have been easier to implement those functions in the item class that I had created. That would have made calling those index-finder functions a lot easier. The attributes assigned to each of the classes though, like the "amount" for items, and the "death" Boolean for players was super useful in the utilization of each of those classes. Though, the skeleton did change substantially. I had to add a few attributes to a couple of classes, like the player class, in which I added a position attribute.

Reflect on how you could have done better or how you could have completed the project faster or more efficiently?

Kind of going back to the previous two questions, my main driver file was absurdly long. It itself was almost 3000 lines. So many of those functions could have been developed into other classes, like the getIndex[item]() functions I described in question two could have been put into the items class to simplify the code. Many of the other integral functions of the game like fightMonster() and displayNPCMenu() very well could have been put into a "game" class, so that they could be called outside of the main file, making a substantially large game class, but a much more manageable driver file, especially since that file has such a large do-while loop to continually display a menu for the user.

Did you have any false starts, or begin down a path only to have to turn back when figuring out the strategy/algorithm for your Final Project program?

I didn't personally have any major false starts. Though, I hadn't realized a map class was provided, (even though I'm pretty sure it stated so in the directions for the project), so I began creating my own map class. Thankfully though, that did not last too long. I began the project working on the displayMerchantMenu() function, which ended up being super long. I had wondered if there was a better and more efficient way to implement the menu, but I opted for the simplest solution which took up more lines. That same kind of mindset carried through the project, just so that I could get a working program free of any game breaking bugs. So, in that thinking, I hadn't really needed to turn back at any points, I just needed to alter functions slightly as the game became more complicated.