Tristan Santiago
February 18, 2018
Project 3 Reflection
CS162-400

DESIGN STAGE

I started this project very low-tech, trying to determine which characteristics should be a part of the base class. I eventually decided on including name, armor, strength, and status. From there, I started defining the functions I would use to set and obtain this information. I also included an attack and defense roll function for the base class, as every character would require those two functions to play the game. From there, I began working on the barbarian class, as it was the simplest of all the classes in the game. I borrowed a lot of logic from my dice project earlier on in the quarter, which I found translated quite well into this program. From there, I needed to begin testing the barbarian class.

MENU DESIGN

As I was designing the program's menu, I found it a lot easier to work by simply including the assignment's instructions and rules of the game directly in main, so I could look at it, rather than switching back and forth from my IDE to Canvas to read about the project. This led me to just coding the rules of the game and the character classes into the program's menu. This wasn't planned, but happened organically as I was trying to fulfill the project's specifications. Although the project didn't specify the need to include the game's rules, I felt it would be a better experience for the user, and decided to make it a personal design choice (while also using it as a reference myself).

EARLY TESTING

This phase consisted of two barbarians fighting head-to-head to test the base class functions and make sure the barbarian class was inheriting properly. Although boring, the game functioned well, and provided a great template that I would use to build the other classes in the game.

Early on, I moved from automatically instantiating my barbarian objects at the start of the program to only creating them when the user selected the barbarian as their character. Again, this served as the perfect template for adding classes later on, while also following the program's specifications. I feel like making this change early on was actually a wiser choice, saving me the trouble of having to redo this step in later stages of the program.

In the early stages, combat continued forever between the two barbarians. It was at this point that I realized I had not yet implemented a win condition, so that's when I began writing the code I would use to continue combat until one of the characters ran out of strength. Again, I was able to reuse a lot of code from my dice project here. The logic I used for pointing to each player in the dice game and whether or not the player was using a normal or loaded die translated perfectly. Using it for this project was simply a matter of applying the same logic for character name, armor, strength, etc. This was also the perfect opportunity for me to incorporate the logic used to prompt the user whether or not they wanted to play the game again, which I did here to avoid confusion with brackets later on. I found it a lot easier to use a huge do-while loop for running the game until the user wanted to stop and program the entire game into it, rather than try to implement the do-while loop after everything else. I have run into a lot of minor issues with figuring out which brackets belong to which statements and loops in the past, so this design choice really simplified things for me.

INPUT VALIDATION

Prompting the user for their choices in the program of course required testing for valid input, so I found it easier to just implement input validation as soon as I began doing so, using the methods I've used practically all quarter. This choice was also great for testing during the development phase and saved me a lot of time in the long run, as I didn't have to go back and test code that I knew was working.

IMPLEMENTATION OF ADDITIONAL CLASSES

Using the barbarian class as a template, it was rather easy to create the other classes used in the game. The only changes I really needed to make were to the name, starting stats, and which dice they used. Again, dice logic was adopted from the dice project. Implementing the logic into this project was simply a matter of changing the number of sides of the dice. I created each class, one at a time, first testing combat against barbarians and then against the created itself. This part of the development phase didn't really take long at all, because I deliberately hadn't implemented special abilities yet. Once I got all classes implemented, I ran many combat tests just to make sure everything from dice rolls to user input was working properly before moving on to special abilities.

SPECIAL ABILITIES

I deliberately avoided implementing special abilities until I got everything else tested and working properly. I started with the vampire's special ability, as it seemed to be the easiest to write. I basically thought of charm as a coin toss, giving the ability a 50% chance to trigger. If the charm works, it gives the vampire 9999 defense. Harry Potter's ability was next. I used a variable to determine whether or not his ability had been triggered or not, since it made sense to me that it could only be used once per fight. If triggered, it updates his strength to twenty, and marks his lives to 2. The third ability was for Medusa. It actually made common sense that her attack did 9999 damage. That way it ensures the vampire can charm her glare, but that her glare will trump and defensive roll made, killing her opponent immediately. The glare is just looking for the two die to total to 12. The final ability was the blue men. Initially I wasn't sure how to go about this - if I should create/remove dice based off of how many strength points they have, or something along those lines. I ultimately found I would just create the dice regardless, and then only roll them based on their strength. Presently they are working based off of a range of numbers. If strength is greater than 8, 3 dice, between 4 and 8, 2, below 4, 1 dice.