Camille Marcojos

CS 162 – Fall 2019

Project 4: Fantasy Combat Game Part 2

Since this was an extension from last week’s project, my design stayed roughly the same and I only had to make some slight changes to my project to meet the requirements. However, that did not mean that I did not experience some challenges in completing the assignment.

First I added the Queue class from my Lab 7 assignment to the program so that I can have a container for the two teams and the losers. I changed the QueueNode struct so that it contained a pointer to a Character and can then hold the fighters. Then I created Queue objects for each team and the losers. I also added a pointer to a Character in the beginning of the program which was dynamically allocated when a character was created.

In my main program, the game flow changed so that the user was first asked how many fighters they wanted for each team and then they would choose each character’s type and name. The new Character was then assigned to a pointer to a Character and then added to the queue. Then the program displays the line up for both teams and then battle commences for the first fighter.

The battle game flow also changed so that now each fighter on both teams fought each other. After each round, the winning team got a point and the losing team lost a point. Then the program checks whether the defending fighter died. If not, then the battle continued. If the fighter did die, then they were placed in the loser container and their team would be checked if it was empty. If the team was not empty, then the next fighter on both teams fought. If it was empty, then the tournament ended. At the conclusion of the tournament, the winning team was displayed and then the user is asked if they would like the losers printed. Finally, the user is asked if they wanted to end the game or replay.

I did have a memory leak problem in this program which changed my initial design slightly. As a result, I added a vector which holds all the dynamically allocated fighters that were created in the beginning of the program for the two teams. I did this so that I can cycle through the stack and delete the allocated memory at the end of the program. I am not sure if that is the most elegant solution, however, I did not have a memory leak after the change.

Harry Potter

- Variable: lives

Medusa

- Variable: glare

- override attack

Blue Men

- Variable: defense die 1; defense die 2; defense die 3

- override defense

Barbarian

Vampire

- Variable: charm

- override defense

Character Class: Base Class

* Member variables: attack, defense, armor, strength, *characteristics?*, special ability, dice, damage, strength
* Functions: attack\*, defense\* required
  + Get attacker type
  + Get defender type
  + Get attack dice roll – attack()
  + Get defend dice roll – defense()
  + Get armor
  + Set damage
  + Get damage
  + Set strength
  + Get strength
  + Roll dice