Discovering Requirements

- Create a tournament using last weeks characters
  - Have two characters fight
    - Resolve a winner and a looser
    - Winner goes to back of line
    - Looser is out
  - Do the above for all characters. Figure out a first, second and third place winner
  - Figure out what side wins

Design

I will steal all of the code from last week, meaning I only have to implement the framework for which characters fight, and the framework for determining winners (and user input, but that doesn't need talked about here).

For determining fighters, I will wrap a loop around my Attack/Defend functions. It will pull members from the front of a queue object and pass them into said functions. This will determine a winner. The winner will be pop()ed and pushed back into the queue (after calling a regen function). The looser will be pop()ed and pushed onto the looser stack. This will loop through until one queue is empty.

To determine the winners, I will pop all characters from the winning queue into a priority queue that will be sorted by the number of kills the player has. I will then pop all characters from the defeated stacks, which should capture characters that were defeated last and place them in the right order. This way, queue winners with the highest score will receive headline victories, while the others with the same score will show as tying. Determining what side wins is easy, I'll just keep a counter running for each side and increment appropriately.

Testing

Much of the testing is already done. Last week I verified that all of my character classes work appropriately. This means that I only had to test to make sure my battle functions and regen functions were working appropriately. Taking out the regen function added to the length of the battle, so I know that was working well. Using a queue and stack greatly reduced the amount of testing to be done, as I can trust that the STL handles them appropriately. Basically, any errors would have been from pointer errors on my part, and I was careful to check with valgrind and a debugger to make sure I wasn't leaking memory anywhere and that the logic was working appropriately. I also simulated large scale combat with the same character on each side and found that each side won about the same number of times.

Reflection

This assignment showed how important it was to have code that could be reused. It probably would have taken twice as long (both in design and testing) if I hadn't already had nicely compartmentalized code.

This assignment also provided practical uses for STL containers. I now have a better sense for the things I might use a queue for (operating system tasks) and things I might use a stack for (maybe layering objects in a game that the character can pick up). I was also able to teach myself how to implement a priority queue, which made calculating winners super easy! All in all, this was a good assignment and fun to implement.