David An
Project 4: Design, Test, and Reflection

Design: The main goal for this program was to create a combat tournament using the classes that we created in Project 3. By using the abstract class of Characters, two teams are built by the user by storing each Character object in their respective lineups (implemented with a queue). This requires me to create a new QueueNode and Queue class to support the storage of Character objects instead of integers. In addition to using Queues to keep track of each team's lineup, I will have to create a Stack to store the defeated characters in order to display them at the end of the tournament. Like the Queue implementation I will have to have them accommodate the holding of Character objects instead of integers. Some changes I will have to make to the Character classes includes a hasFought member variable to check to see if the Character has already fought during the battle phase of the tournament – this is used to see if the character requires recovery or not. In addition, I will need to create a name member variable for each character so that each character can be kept track of, even if repeating characters are used in each lineup. Another change I will have to implement is a recover() method so that characters that have fought already are able to recover some of their strength points when they are called to fight again.

Test Case	Input Values	Functions	Expected	Observed
Input too low	Input < 1	begin() [Menu] battle() [Menu]	Re-prompt for new entry	Re-prompt for new entry
Input in correct range	1 <= Input <= 20	^	Appropriate number of characters is chosen/appropriate option is chosen	Appropriate number of characters is chosen/appropriate option is chosen
Input extreme low	Input = 1	^	1 character is prompted to be entered into each team/Vampire is chosen	1 character is prompted to be entered into each team/Vampire is chosen
Input extreme high	Input = 20	^	20 characters are prompted to be entered into each team	20 characters are prompted to be entered into each team
Input too high	Input > 20	^	Re-prompt for new entry	Re-prompt for new entry
Input is incorrect type	Input = String	^	Re-prompt for new entry	Re-prompt for new entry

Reflection: I began by changing the Character and Queue classes to reflect what I needed in the new format. This included adding in the previously mentioned member variables and setter/getter member functions for each new member variable. I also needed to change the type of value held by the QueueNodes from integers to Character pointers, meaning that I needed to change all of the next() prev() variables to either set or return pointers to Character objects as well. I then created the menu to display instructions and validate user input. I used a for loop to determine the number of Characters that are to be added into each team, adding in a character to each team during each iteration of the loop. I then created the battle phase of the menu which is where the teams fight eachother until one of the lineups is empty (meaning all characters that were once in the lineup are dead). Whenever a character dies, I made sure to add them into the loser Stack that I set up at the beginning of the battle phase. The most challenging aspect of implementing the battle phase of the menu function was creating the recover() member function for the Character class. Since I was to restore a percentage of the lost strength points, I had to create a new member variable that stored the original SP amount for each Character, since each Character holds a different beginning SP. I then had to calculate the difference between the Character's current SP amount and its' original SP amount, and randomly allocate a percentage of that difference to be added back to the current SP amount. A hiccup I ran into was how to type cast the randomly generated number into a double in order to get the percentage I wanted. I eventually decided simply generate a random double from 0 to 1 (which would act as the percentage to be recovered), and calculate the SP to be recovered by the Character. This amount would then be added to the current SP amount and this new result would be set as the Character's new current SP.