CS 162 - Proj4: Fantasy Combat Tournament - Design + Reflection Document
Jeovani Vela
3/2/18

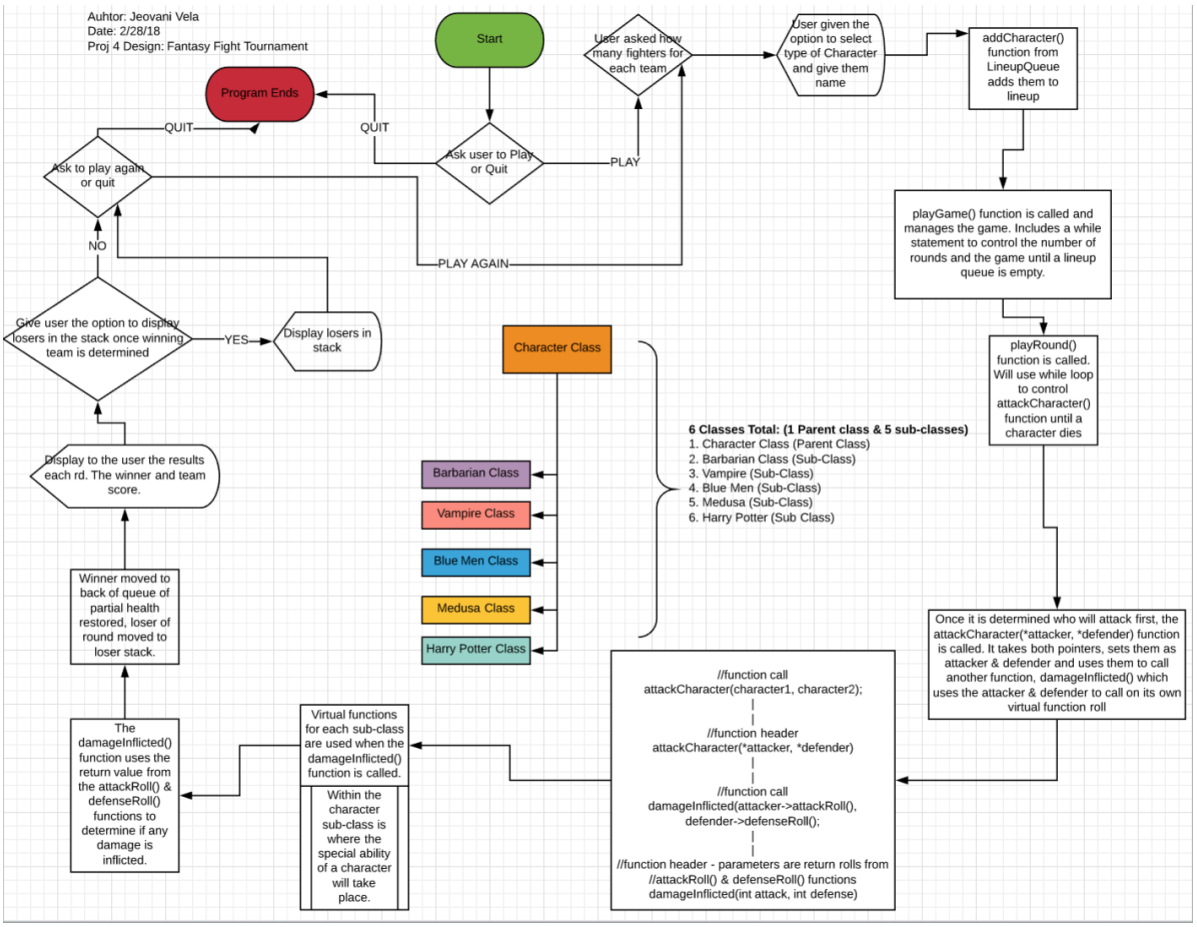
Project 4 was unique as it enabled you to build off Project 3 to re-create the game based on a fighting game amongst fantasy characters into a fighting tournament format. Lineups are created with a queue and a stack serving as a container for characters in the loser column.

Requirements:

-Utilize existing files from Project 3 to expand game into a fighting tournament.
-Must use a stack and queue to serve as "containers" in the game.
    -Enables you to set lineups and move the loser of each round into the "loser stack".
-Winner at the front of each lineup moves to the back of the lineup and has health partially restored.
-Loser of the round is moved to the Loser stack.
-Display the results of each round: Character types, names, winner, loser, and score.
    -Including which team won the tournament, even if there is a tie.
-Ask user if they would like to display the Loser stack.
-Ask user to play again or quit.
    -If quit, program ends.
    -If continue, game starts over and user can re-choose which characters to fight.

Design:

Utilizing the existing files from Project 3, I expanded my program to include a couple more classes to contain the functions and member variables necessary to perform a fantasy fighting tournament. I first created the ListNode struct which would serve as the nodes to be used to traverse the lineup in the queue and the loser stack. Member variables included pointers and a special pointer to type Character as this is the type of data the node would hold. Next I



began to work on my queue class. I created a class named LineupQueue which contained all the member variables and functions necessary to create a lineup, remove a Character from the lineup if they lost in a round and to move the winner of the round to the back of the lineup. After my queue was sorted out, I re-modified my menu so that when a user selected a Character and gave it a name, it would then include a function from the LineupQueue class that would add that Character to a lineup. The queue is set as a "first-in, first-out" scenario, with winners moving to the back of the lineup and having their health partially restored. After this was set, I created my stack class, LoserStack. This class would serve as the container where the losers from each round would be held to later be displayed at the user's discretion. Its member variables consisted of pointers type ListNodes and a couple of members functions including adding a losing Character to the stack and displaying the losers in the stack. The stack was set up as a "last-in, first-out" container. Finally, I re-formatted a couple of member functions in the Character class in order to handle some of the new functions included in the program and to add a function that enabled the program to restore some strength to the winner of the round in the fighting tournament. After user is prompted with the option of displaying the losers in the stack, I changed my playAgain() function slightly to enable the user to continue playing the game with the results being reset or giving he/she the option to quit the program entirely.

Test Plan:

Test Case	Input Values	Driver Functions	Expected Outcomes	Observed Outcomes
startMenu()- User prompted with 2 choices: Play or Quit -user selects quit	`sdgh, <>fdg, 235., 566, fdsg67!, Q, 2`	Input validator will only accept a number 1 or 2.	User will get message stating input is incorrect and choose either 1 or 2	Program does not continue until user enters correct number. User enters 2 and program ends,
startMenu()- User prompted with 2 choices: Play or Quit -user selects play	`edged, 1111, 11, daagfsd, NMN*U*^, ()sdaf, 1`	Input validator will only accept a number 1 or 2.	User will get message stating input is incorrect and choose either 1 or 2.	Program does not continue until user enters correct number, 1 or 2. Program continues after user inputs 1.
displayMenu()- User prompted to enter a number for the number of fighters on each team	`rewt., fads, sfg45, <gfh78, >?", 0, 3`	Input validator will only accept a positive integer that is greater than 0.	User will get message stating incorrect selection. Enter a positive number greater than zero.	Program does not continue until the user enters a positive number greater than zero. In this case, user selects 3.
displayMenu() User prompted to select a character to fight, 5 selections	`0, dfsg., 99, 100, dfsg, -4, 3`	Input validator will only accept a number between 1 and 5	User gets error stating invalid choice. Select a number between 1 and 5.	Program does not continue until user selects a character between 1 and 5. User selects 3.
displayMenu() Check if name given by user can handle a space.	`Selection 4 is made- for nickname entered: YES YES YES`	Must have getline(cin, ID) in order for program to handle a name with space(s).	Program will continue without issue because getline() is used.	Program continues without issue prompting user to continue selecting characters and giving them nicknames.
displayWinner() -User asked to display loser stack. -if yes, printLoserStack() function called	`fdsgdsfg., 546345. 546. dghf.., fsg, 1111, hello, y`	Input validator will only accept a 'Y' or 'N" to continue.	Error message to user will be displayed to select either a 'Y' or 'N' to continue.	User selects 'y', program continues and displays losers in stack.

Test Case	Input Values	Driver Functions	Expected Outcomes	Observed Outcomes
playAgain() -User asked to continue playing or quit.	`sfdgs, dgfs., <><, 111.324., afdsg., p`	Input validator will only accept a 'P' or 'Q' to continue.	User gets error message until either a 'P' or 'Q' is selected,	User selects 'P', program continues, and game resets prompting user to choose the number of fighters for each team.
playAgain() -User asked to continue playing or quit.	`fcvbv,sdfg/, f5436354, {} DFFS, q`	Input validator will only accept a 'P' or 'Q' to continue.	User gets error message until either a 'P' or 'Q' is selected,	User selects 'q' and program ends without issue.

Reflection:

This project was much more complicated to re-format than I expected. It was not complicated creating my struct for my ListNode, however when I implemented my LineupQueue class I ran into a number of memory leak issues. Initially I made the mistake of deallocating each new Character object in the displayMenu() function after having the Character passed in the addCharacter() function for the class LineupQueue. The leaks got a bit worse when my LoserStack class was implemented as more memory was being allocated, but not deallocated at the correct places. Thus, I had to create a destructor in my Tourney class that would call on certain class pop functions that would remove a node from a class or queue. This solved a good portion of the memory leaks. Others were solved by going back and deleting where I had created new character objects or nodes during the testing of my code. I also had to reformat my Game class from Project 3 as most of the functions utilized for the game had to be moved to the new class Tourney, considering all the members functions and variables that had now been included in the program. Additionally, I made the mistake of using the #include header file declarations incorrectly in some files which led to much of the "does not name a type" error. By including all header files in some of the newer classes created and not placing them in a certain order, at compile time the program was having difficulty determining data types as it traversed the program. Thus, I had to go back through each file and reformat much of the #include header files in order to resolve the issue.

After running into more complications than expected, I think it would have been beneficial to me to re-write some aspects of the program from scratch instead of going through each file and slowly making changes here and there, while forgetting about other places. Maybe over time my programming abilities will adjust and re-formatting existing programs will not be as difficult for me, however this project served as a valid experience of what it would be like to reformat a large program for a type of update or change. I need to learn to keep a log and write down every change that was made as I struggled to keep up with so many files to go back and forth from.