Ryan DiRezze
CS 162 400
Project 4: Fantasy Combat Game
November 18, 2018

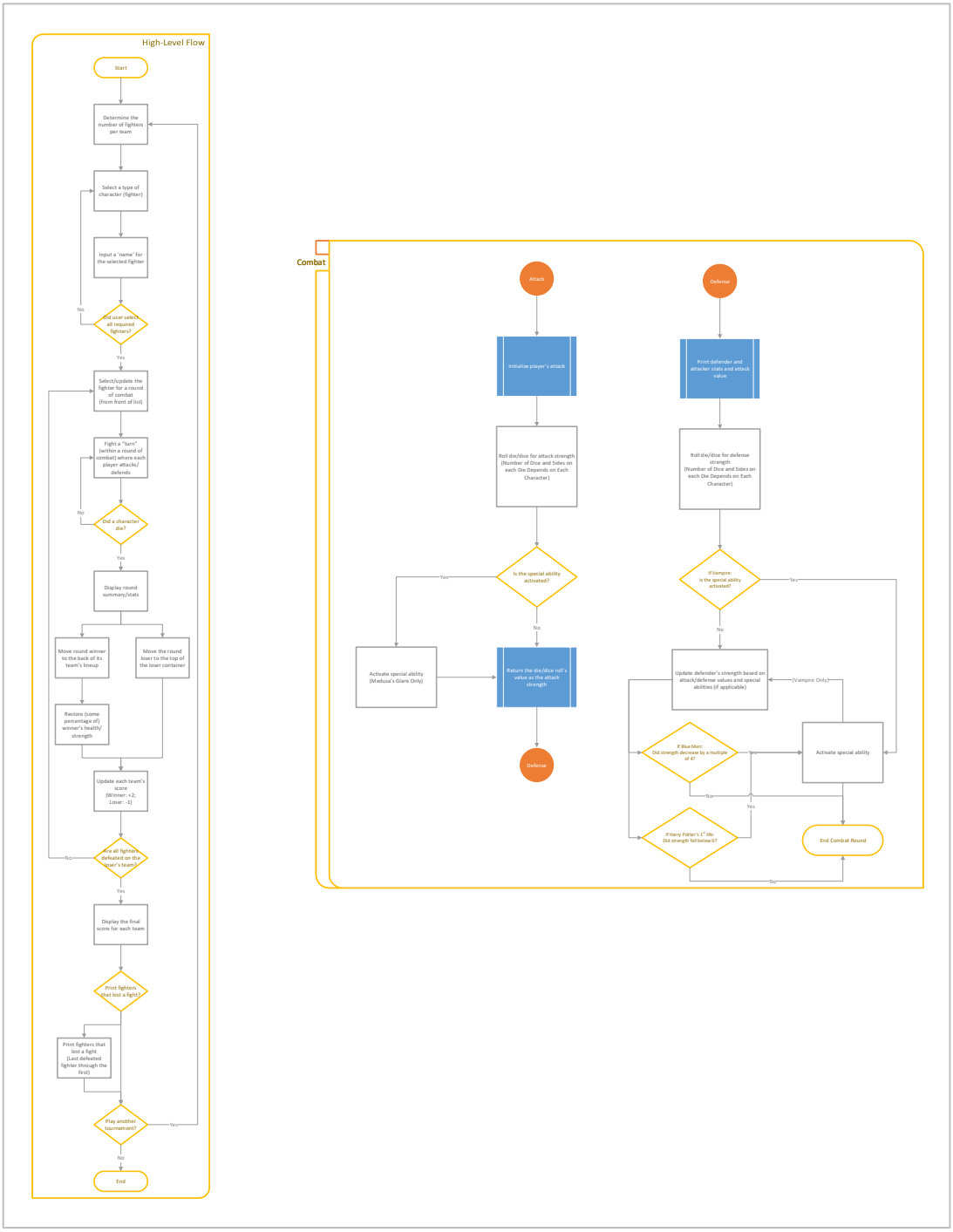
PROJECT 4 (DESIGN + REFLECTION)

Project 4 (Fantasy Combat Tournament Game) design, test cases, and reflection

TABLE OF CONTENTS

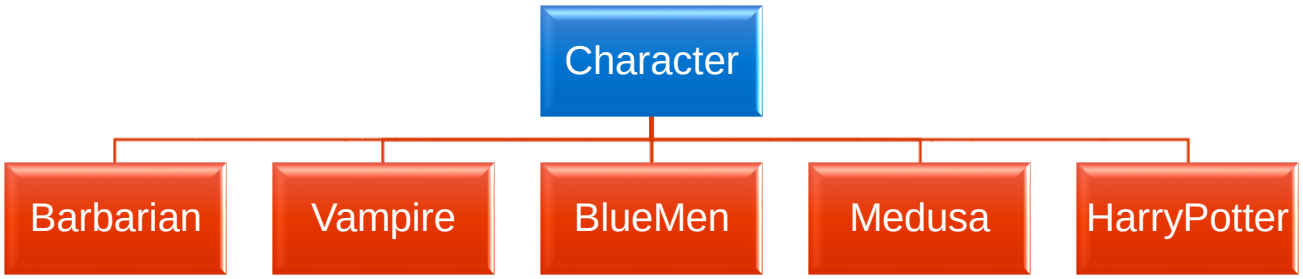
DESIGN / FLOWCHART

The below flowchart (on the left) illustrates the high-level flow of this program, including a more detailed flowchart (on the right) to illustrate a round of combat. A copy of the below flowcharts is also included within the assignment .zip file.



CLASS HIERARCHY

The below diagram illustrates the use of inheritance within this program. As shown below, *Barbarian, Vampire, BlueMen, Medusa, and HarryPotter* classes inherit most, or all, of their member variables and functions from the base class, *Character*.



REFLECTION

This assignment reflection will cover the following content: (1) design changes while developing and implementing the assignment's program, (2) problems that were experienced and how those problems were overcome, and (3) what was learned from this assignment.

Project 4 was particularly challenging because two different programs (circular linked list and project 3) were essentially merged together to create Project 4. I experienced quite a bit of challenges with this assignment, like managing memory and preventing memory leaks, but I was able to get the program to operate as intended.

One particular challenge that I experienced was the development of a circular linked list with characters, or fighters, in order to create the proper teams, or groups of people, that were required within the program. A design change that I had to make was changing the value's type from integer to *Character*, which is the parent class of the different types of fighters. Next, I experienced problems with creating the destructor for an object of type *Queue*.

The problem was that I received a segmentation fault when trying to run the program on FLIP. I overcome this problem by carefully structuring the destructor to free memory for one character, then node, at a time. I would repeat this process until the loop encountered a node with a value of *null*. What I learned from this assignment is just how important memory management is, and I learned the importance of carefully freeing the intended object(s). For example, I received the segmentation fault because I was likely trying to access memory that I shouldn't have been.

Lastly, I learned the importance of modularization within a program. After encountering bugs within my program, I found the program exceptionally difficult to debug because my functions were quite large, and thus, even after pinpointing an error to a particular function, the problem(s) within my code was not clear at first; I had to perform additional analysis in order to determine the root cause of the error(s) that my program encountered. Going forward, I intend to utilize functions wherever possible to modularize my code. By modularizing the code of my program, I'll enable myself with more meaningful messages when debugging my program, which would point me to a particular function within the program that would be causing issues.

TEST CASES

The below test table includes the: test plan, expected output, and actual output. See the assignment .zip file for a larger image of the below test table within a PDF file.

Test Case	Input Value(s)	Driver Functions	Expected Outcome(s)	Observed Outcome(s)
Incorrect input to start/exit the game	Input = %, 5, *, P, l	void introMenu(int& numFighters, bool& status)	Notify the user that the input is wrong & ask for another input	Notify the user that the input is wrong & ask for another input
Correct input to start/exit the game	Input = 1, 2	void introMenu(int& numFighters, bool& status)	Start the program or exit the program	Start the program or exit the program
Incorrect input for the number of fighters per team	Input = *, &, @, l, P	void introMenu(int& numFighters, bool& status)	Notify the user that the input is wrong & ask for another input	Notify the user that the input is wrong & ask for another input
Correct input for the number of fighters per team	Input = 1, 2, 3, 4, 5, ...	void introMenu(int& numFighters, bool& status)	Accept input & progress forward to request fighter character type selections	Accept input & progress forward to request fighter character type selections
Incorrect input for teams 1 & 2 character selections within the menu	Input = %, &, *, (,), #, $, k, p, Y	void GameMenu(int& option1)	Notifies the user of an incorrect input and loops back to the question, prompting the user for input	Notifies the user of an incorrect input and loops back to the question, prompting the user for input
Correct input for teams 1 & 2 character selections within the menu	Input = 1, 2, 3, 4, 5, 6	void GameMenu(int& option1)	Accepts character selections for teams 1 & 2 (all character menu choices must be acceptable if chosen) until the correct number of players per team is chosen	Accepts character selections for teams 1 & 2 (all character menu choices must be acceptable if chosen) until the correct number of players per team is chosen
Input for character names	Input = %, 3, l, P	player1->setName(name1)	Updates a character's name	Updates a character's name
Incorrect input to display characters in the loser pile	Input = %, f, L, y, N, @	losers->reversePrint()	Notifies the user of an incorrect input and loops back to the question, prompting the user for input	Notifies the user of an incorrect input and loops back to the question, prompting the user for input
Correct input to display characters in the loser pile	Input = 1, 2	losers->reversePrint()	Prints the list of defeated fighters or does not list the defeated fighters, then asks the user if he/she wants to play another game	Prints the list of defeated fighters or does not list the defeated fighters, then asks the user if he/she wants to play another game
Incorrect input when prompted to play another game (Yes / No)	Input = %, &, *, (,), #, $, 0, 5, 3, 7, k, p, A, a	mainMenuReturn(char&)... User must enter a valid character for the yes or no prompt to play another game	Notifies the user of an incorrect input and loops back to the question, prompting the user for input	Notifies the user of an incorrect input and loops back to the question, prompting the user for input
Correct input when prompted to play another game (Yes / No)	Input = 1, 2	mainMenuReturn(char&)... User must enter a valid character for the yes or no prompt to play another game	Y'/'y': Return to the menu & requests the user to select two characters for combat 'N'/'n': End the program	Y'/'y': Return to the menu & requests the user to select two characters for combat 'N'/'n': End the program