Ryan DiRezze
CS 162 400
Project 3: Fantasy Combat Game
November 4, 2018

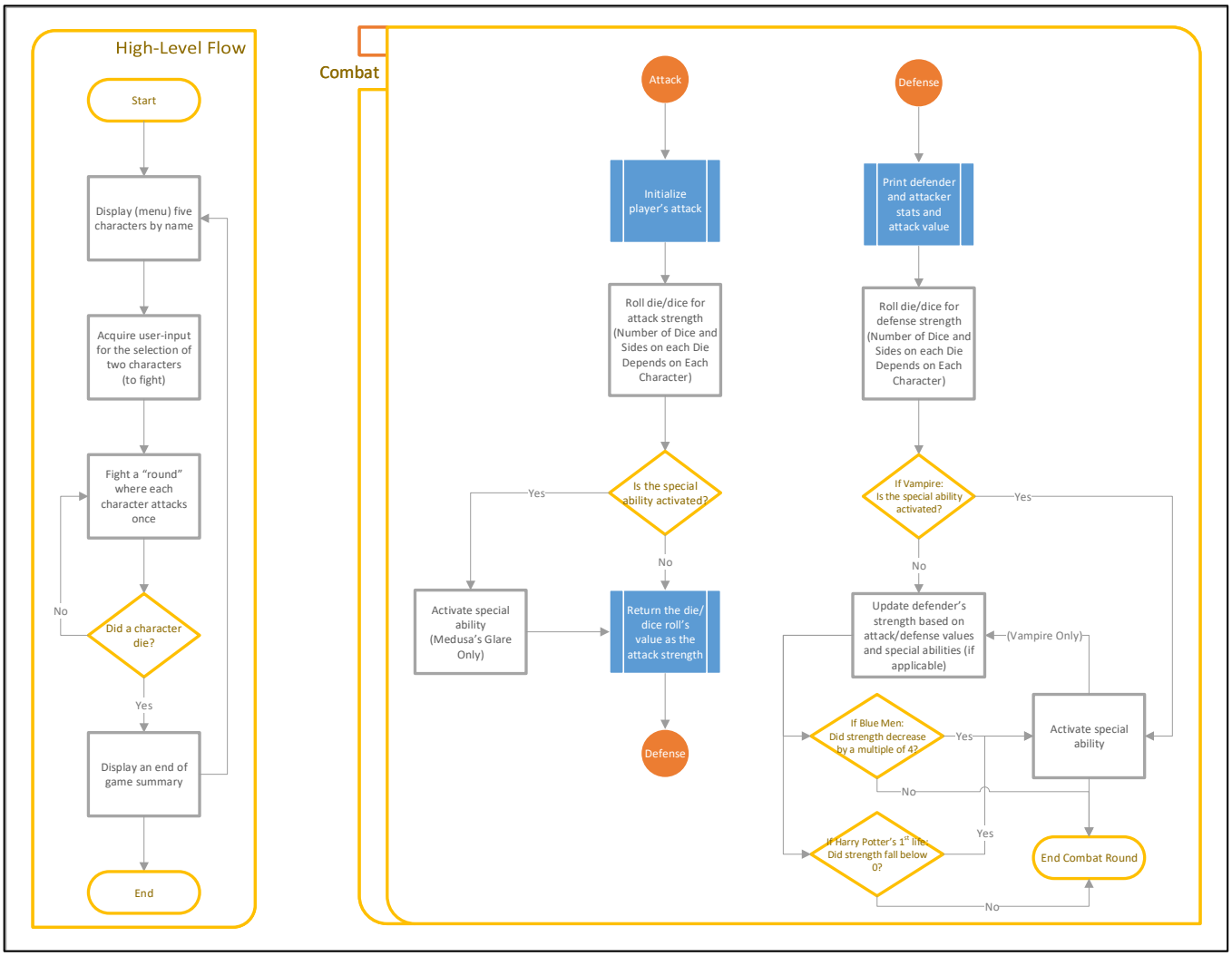
PROJECT 3 (DESIGN + REFLECTION)

Project 3 (Fantasy Combat Game) design, test cases, and reflection

TABLE OF CONTENTS

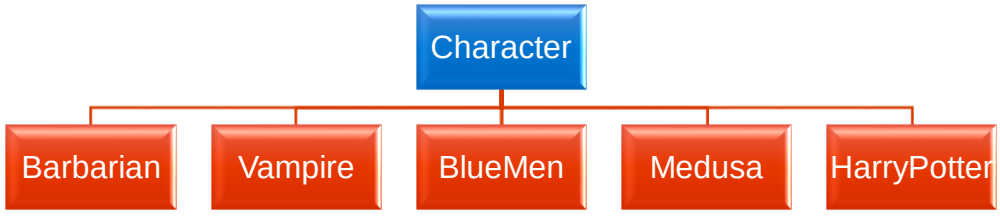
DESIGN / FLOWCHART

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.

When I started this assignment by drafting the first draft of the process map, which is shown on the first page of this reflection, I began with a "high-level" diagram to illustrate the overall sequence of events, like: printing program information, acquiring user input, class hierarchy, the "rounds" of combat within this program, and a decision point that determines whether each "game", which consists of "rounds" (of combat) until a character dies. Once I drafted a baseline design for how the program should operate, I felt comfortable to begin programming the classes of all characters while implementing inheritance.

A challenge that I faced was how to properly implement each of the characters' special abilities. Since each special ability is unique to the character, I wasn't sure of the best way to incorporate the requirements into the program. After reviewing all special abilities, I noticed that all functionality was related to when a character attacks or defends, which makes sense due to the nature of the program. Thus, I decided to implement the special abilities within each specific character's attack or defense function. I learned that some customization is acceptable within "like" functions between characters. By using slightly "customized" attack and defense functions, I was able to easily implement the requirements.

A second challenge that I faced was how to implement the Harry Potter character's special ability, requiring two lives with different numbers of strength, which reflects the "health points" of a character. The requirement states that Harry Potter must have 10 strength, and if Harry Potter's strength falls to zero, he must have his strength reset to 20. However, Harry Potter only has two lives, and thus, he would die if his strength fell to zero, after his strength was reset to 20. I overcame that challenge by implementing a unique member variable for the *HarryPotter* class called, "lives," which was programmed to initialize with a value of two. This way, I was able to program an if statement within the *HarryPotter* class' defense function that would execute if the character's strength fell to zero, or below, and the character's lives must be equal to two. If the if statement's criteria were met, Harry Potter's strength would be reset to 20 while also decreasing Harry Potter's "lives" variable by one. After Harry Potter's "revival", he would "die" if, the *HarryPotter* object's strength value reached zero or below, ending the "game" between the two characters. Again, I learned that customization is acceptable between children classes when using inheritance with a base class. Since the *HarryPotter* class was the only class that required two lives, I chose to not put the "lives" member variable within the base class, and I chose to add the "lives" member variable solely within the *HarryPotter* class.

This assignment's required use of polymorphism was very powerful, and I gained some great perspective on how polymorphism and inheritance can be used within C++ programs to implement creative and unique functionality, like the functionality within this assignment. Once the concept of polymorphism and inheritance was understood, I was able to design and implement this program with relative ease. A major takeaway of this assignment was when you should choose to design and implement certain member variables and functions within a base class versus an inherited class. Lastly, I learned that C++ can be used to implement some pretty cool programs due to powerful functionality, like polymorphism and inheritance.

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 for player 1 & 2 character selections within the menu	Input = %, &, *, (,), #, $, 0, 5, 3, 7, k, p, Y	fantasyGameMenu(char&, char&)... User must select a valid menu option (character/letter) from the following: a, b, c, d, e, f	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 player 1 & 2 character selections within the menu (Uppercase & Lowercase)	Input = 'A', 'a', 'B', 'b', 'C', 'c', 'D', 'd', 'E', 'e', 'F', 'f'	fantasyGameMenu(char&, char&)... User must select a valid menu option (character/letter) from the following: a, b, c, d, e, f	Accepts character selections for players 1 & 2 (all character menu choices must be acceptable if chosen) and runs the program until the game (fight with 1+ rounds) completes	Accepts character selections for players 1 & 2 (all character menu choices must be acceptable if chosen) and runs the program until the game (fight with 1+ rounds) completes
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 = 'Y', 'y', 'N', 'n'	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