Clinton Hawkes
CS162-400
Project 3 Design Document

Fantasy Combat

Design:

Program will have a main function, a Character class, a Vampire class, a Barbarian class, a BlueMen class, a Medusa class, a HarryPotter class, a Menu class, a rollDie() function and an input validation function.

1) Main function – only used to seed the rand() function, create a Menu object and call the Menu::test() function to begin testing the game characters.
    A) Create a bool variable “game” to hold the current state of the game (run or quit)
    B) Create two Character pointers that will point to p1 and p2 fighters
    C) While loop created that will loop until the variable “game” is set to false
        I. Menu object calls menu1.
            a) Console screen is cleared
            b) Ask user to select the type of character for player 1 or quit
                i. If quit, return nullptr to the calling function and test will quit
                ii. If user chooses one of the five types of characters from the list
                    1) The type of character corresponding to the user’s selection is created
                        A) Character pointer pointing to new object is returned to p1
        II. Menu object calls menu2.
            a) Console screen is cleared
            b) Ask user to select the type of character for player 2 or quit
                i. If quit, return nullptr to the calling function and test will quit
                ii. If user chooses on of the five types of characters from the list
                    1) The type of character corresponding to the user’s selection is created
                        A) Character pointer pointing to new object is returned to p2
        III. While loop is setup that runs until the battle variable is false.
            a) If rnd variable is odd, p1 is the attacker and p2 is the defender
                i. Round number is printed
                ii. Statement prints informing user who the attacker is
                iii. Statement prints informing user who the defender is and states armor and sp points
                iv. p1’s attack function is called
                    1) RollDie function is called to simulate the roll of dice.
                        A) Character appropriate attack value is returned
                v. Statement prints informing user of p1’s attack value
                vi. p2’s defense function is called with p1’s attack value passed as an argument
                    1) p2’s defense function calls rollDie() to simulate rolling dice
                        A) Defense function calculates inflicted damage using Attack-Defense-Armor

B) Total inflicted damage is returned

vii. Statement prints informing user of damage inflicted on p2

viii. Statement prints with updated sp points for the defender

ix. If the defender has 0 or fewer sp points

1) Statement prints saying the defender lost

2) Battle is set to false

x. rnd variable is incremented

b) If rnd variable is even, p2 is the attacker and p1 is the defender

i. Statement prints informing user who the attacker is

ii. Statement prints informing user who the defender is and states armor/sp points

iii. p2's attack function is called

1) RollDie function is called to simulate the roll of dice.

A) Character appropriate attack value is returned

iv. Statement prints informing user of p1's attack value

v. p1's defense function is called with p1's attack value passed as an argument

1) p1's defense function calls rollDie() to simulate rolling dice

A) Defense function calculates inflicted damage using Attack-Defense-Armor

B) Total inflicted damage is returned

vi. Statement prints informing user of damage inflicted on p1

vii. Statement prints with updated sp points for the defender

viii. If the defender has 0 or fewer sp points

1) Statement prints saying the defender lost

2) Battle is set to false

ix. rnd variable is incremented

IV. Once a winner is named, the loop exits and a the winner is printed to the console

V. Object referenced by p1 is deleted and p1 is set to nullptr

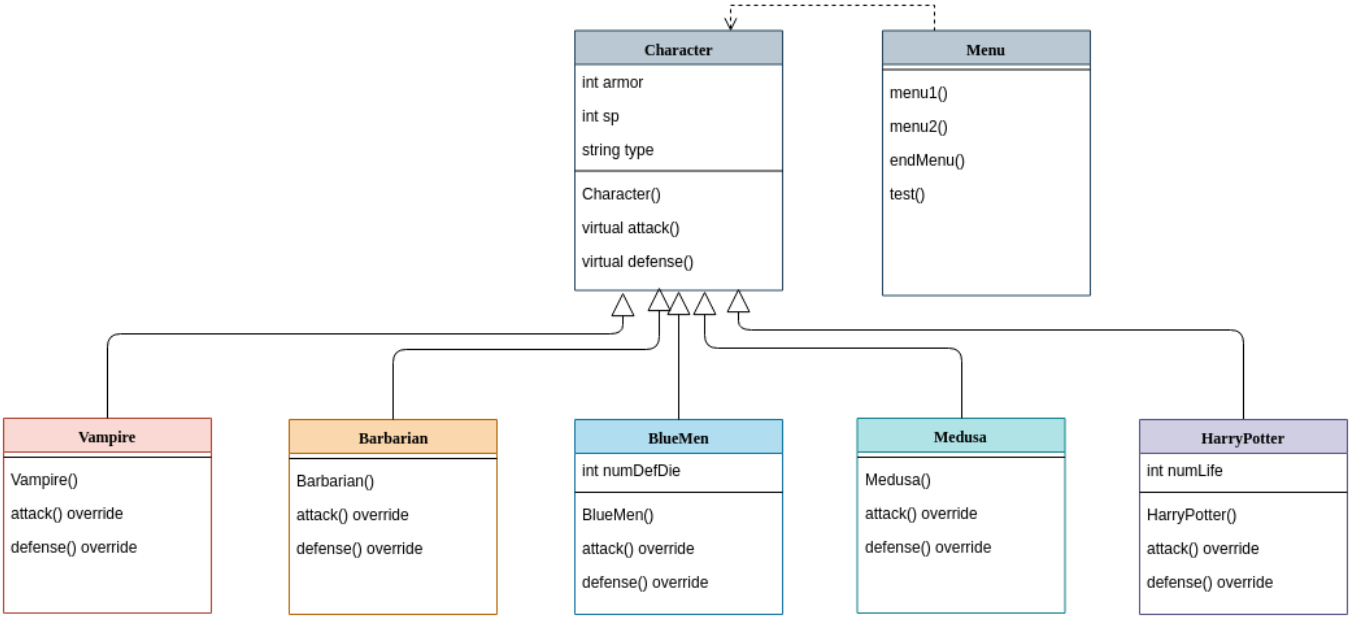
VI. Object referenced by p2 is deleted and p2 is set to nullptr

VII. endMenu is called

a) User is asked if they want to continue or quit

i. If user quits, game is set to false and program terminates

ii. If user wants to continue, the loop begins at the top



Character Class: Used as the base class for the different types of game characters.

Variables:	Functions:
Int armor	Character()
Int sp	Virtual int attack() = 0
String type	Virtual int defense() = 0
	Virtual string getType()

Vampire Class: Used to store info about a Vampire object. Derived from the Character class.

Variables:	Functions:
***Inherits Character variables	Vampire()
	Int Attack() override
	Int Defense() override

Barbarian Class: Used to store info about a Barbarian object. Derived from Character class

Variables:	Functions:
***Inherits Character variables	Barbarian()
	Int Attack() override
	Int Defense() override

BlueMen Class: Used to store info about a BlueMen object. Derived from Character class

Variables:	Functions:
***Inherits Character variables	BlueMen()
	Int Attack() override
	Int Defense() override

Medusa Class: Used to store info about a Medusa object. Derived from Character class

Variables:	Functions:
***Inherits Character variables	Medusa()
	Int Attack() override
	Int Defense() override

HarryPotter Class: Used to store info about a HarryPotter object. Derived from Character class

Variables:	Functions:
***Inherits Character variables	HarryPotter()
	Int Attack() override
	Int Defense() override

Menu Class: Used to store info about a Turtle object. Derived from Character class

Variables:	Functions:
***None	Menu()
	Character* menu1()
	Character* menu2()
	Bool endMenu()
	Void test()

Validation Function: The validation function will receive input from the user, check that the input is the correct data type, check if the input is within the specified range, and if it fails these test, the user will be prompted to enter a valid input. The validation functions will return a value of the indicated type back to the main() after all the tests are passed.

Test Plan for integerValidation()

Test Case	Input Value	Expected Result	Observed Result
Char	A, c, +, .	“Please enter a valid integer.”	“Please enter a valid integer.”
String	Four, .five, yes	“Please enter a valid integer.”	“Please enter a valid integer.”
Float	6.31, 07.0, 8.8	“Please enter a valid integer.”	“Please enter a valid integer.”
< Min (set at 1)	0, -1, -5	“Please enter a valid integer.”	“Please enter a valid integer.”
> Max (set at 4)	5, 9999	“Please enter a valid integer.”	“Please enter a valid integer.”
Lower limit (min=1)	1	Function accept and return value to main()	Function accepted 1 and returned 1 to main()
Upper limit (max=2)	2	Function accept and return value to main()	Function accepted 2 and returned 2 to main()

Test Plan for Fantasy Combat

Test Case	Description	Expected Result
Vampire Object Creation	Create a new Vampire	1. Vampire object should be created on the heap. 2. Member variables should be set to the values in the constructor 3. A Character pointer should be pointing to the newly created object
Barabarian Object Creation	Create a new Barbarian	1. Barbarian object should be created on the heap. 2. Member variables should be set to the values in the constructor 3. A Character pointer should be pointing to the newly created object
BlueMen Object Creation	Create a new BlueMen	1. BlueMen object should be created on the heap. 2. Member variables should be set to the values in the constructor 3. A Character pointer should be pointing to the newly created object
Medusa Object Creation	Create a new Medusa	1. Medusa object should be created on the heap. 2. Member variables should be set to the values in the constructor 3. A Character pointer should be pointing to the newly created object
HarryPotter Object Creation	Create a new HarryPotter	1. HarryPotter object should be created on the heap.

		2. Member variables should be set to the values in the constructor 3. A Character pointer should be pointing to the newly created object
Vampire Charm	Test Vampire against every type of character	1. When the Vampire charms, there should be a message printed to the screen and no sp points should be deducted from Vampire
BlueMen Defense	Test BlueMen against BlueMen and take note of the sp points and the defense roll.	1. When sp of BlueMen is 9, 10, 11, or 12, the defense should roll 3 dice. Range of defense is 3 to 18. 2. When sp of BlueMen is 5, 6, 7, or 8, the defense should roll 2 dice. Range of defense is 2 to 12. 3. When sp of BlueMen is 1, 2, 3, or 4, the defense should roll 1 die. The range of defense is 1 to 6.
Medusa Glare	Test Medusa against every type of character	1. When Medusa rolls an attack of 12, she should perform her glare. There should be a message printed and the attack value returned should be 99. The opponent should be defeated. 2. Special circumstance when opponent is Vampire. If Vampire charms when Medusa glares, no damage should be inflicted on Vampires. 3. One more special circumstance. If HarryPotter is on his first life and Medusa glares at him, HarryPotter will survive and return with a sp of 20.
HarryPotter Hogwarts	Test HarryPotter against every other type of character	1. If HarryPotter is on his first life and his sp drops to 0 or below, he will instantly return with an sp of 20. He will no longer have an extra life and will die if his sp reaches 0 once more.
Vampire attack/defense calculations	1. Test Vampire against every other type of character.	1. During attack, Vampire should roll the dice and return a value between 1 and 12. 2.During defense, Vampire should charm the opponent 50% of the time. If charm take place, there is no damage to Vampire this round. If charm does not take place, then total damage should be Attack-Defense-Armor. Defense is a dice roll between 1 and 6.
Barbarian attack/defense calculations	1. Test Barbarian against every other type of character.	1. During attack, Barbarian should roll the dice and return a value between 2 and 12. 2. During defense, the total damage subtracted from Barbarian's sp should be Attack-Defense-Armor. Defense is a dice roll between 2 and 12.

BlueMen attack/defense calculations	1. Test BlueMen against every other type of character.	1. During attack, BlueMen should roll the dice and return a value between 2 and 20. 2. During defense, the total damage subtracted from BlueMen's sp should be Attack-Defense-Armor. The defense is a roll of the defense die. Blue men should have 3, 6 sided dice when sp is > 8. If sp drops to 8 then BlueMen have 2, 6 sided dice. If sp drops to 4 then BlueMen have 1, 6 sided die. Defense roll should be between 1 and 18, depending on how many dice he has.
Medusa attack/defense calculations	1. Test Medusa against every other type of character.	1. During attack, Medusa should roll 2, 6 sided dice. If medusa rolls a 2 to 11, that value is returned as an attack. If Medusa rolls a 12, she performs her glare. The glare instantly defeats her opponent. (***unless vampire charms her or) If HarryPotter has another life, he will remain in the fight. 2. During defense, the total damage subtracted from Medusa's sp should be Attack-Defense-Armor. Defense is a dice roll between 1 and 6
HarryPotter attack/defense calculations	1. Test HarryPotter against every other type of character.	1. During attack, HarryPotter should roll the dice and return a value between 2 and 12. 2. During defense, the total dmage subtracted from HarryPotter's sp should be Attack-Defense-Armor. Defense should be a dice roll between 2 and 12. If Harry is hit and his sp drops to 0, he will come back to life with a sp of 20. This only happens if he is on his first life. If he dies again, that is final.
Main Menu1	Program should start at menu1	1. User is asked to choose the character for the first player in the game. 2. The user selects the type of character they want for player 1 3. Menu2 should appear
Main Menu2	Menu should look the same as menu1 except for the selection of the first player to be displayed.	1. User is asked to choose the character for the second player in the game. 2. The first player that was chosen should be displayed in the upper right hand corner. 3. User selects the character and the tests are ran until a winner emerges.
Continue Menu	Press 1 to continue Press 2 to quit	1. Menu1 should appear and the user should have the option to choose their first player. 2. The test should terminate.
RollDie function	1. Test rollDie function with multiple combinations of numDice and numSides.	Results should be between 1 and numDice*numSides.

Memory Leak	1. Start and run program using valgrind. Choose Vampire for first Choose BlueMen for second View results Press 1 to continue Choose Barbarian for first Choose Medusa for second View results Press 0 to Quit 2. Start and run program using valgrind. Continue to play until each type of player has been matched with every other type, including self Press 0 to Quit 3. Start program and immediately quit.	There should be no memory leaks for any of these cases.

Reflection:

I can finally say the projects and labs in this class are getting easier for me. I used to spend a couple days thinking about the problem and figuring out a way to implement it. I would be unsure of what to do and have to refer to the posts in Piazza. Now, I have a general idea after I read the assignment and just have to work out the details. I guess that just shows I'm learning in this class.

Issues:

The only issue I had with this project was trying to figure out what the specifications meant. When I first read the assignment I didn't know what was supposed to be included in the derived classes, the base class, what was supposed to be a member variable or a function. It was confusing. I know there were a lot of other students that had the same problem, because there have been so many posts on Piazza and slack asking for clarification. I guess that is one of the things we will run into when we work in the real world, but it was frustrating at first. Sure, looking back now it is a no brainer, but when somebody is trying to learn how to do something and the instructions aren't clear it get frustrating.

Changes:

My original design for the defense function passed the attack value as well as the type of attacker. I was reading through the spec sheet and saw that the characters should not check to see who there opponent was, so I needed to come up with a different plan. I don't know if passing the type of opponent to the defender went against the spec, but I did not want to leave it up to chance. I thought a little more about the problem and came up with the solution of assigning the glare a specific attack amount. Now when Medusa glares at her opponent there is a message printed to the console and an attack value of 99 is passed to the defender. Every defender checks to see if the attack value is 99, and if it is they instantly die. (not if vampire charms or HarryPotter hogwarts) This was a simple solution to the issue that came up.

Another change that I made to my project was moving all the test logic away from the main function. I have heard some of the TAs talk about deducting points for code being in the main function. I know there has to be some code in the main function, but I really don't know what amount is going to lower my grade. I moved everything into the Menu class under the test function. It really isn't a big deal, and it takes no more effort to put the code in a class, but I wish I knew why it is such a big deal. I had all the code you see in my test function in main because it was easier for testing when just getting started. I forgot until the end and had to move it.