Program Name: Project 3: Fantasy Combat Game

Student Name: Robert Jones

Class: CS 162 – 400, Winter 2019

Date: February 6, 2019

Design Description:

This is a Fantasy Combat Game that contains characters of Vampire, Barbarian, Blue Men, Medusa, and Harry Potter. The characters also have their own characteristics as well as special abilities.

Two characters will be chosen by the user and they will enter in combat with each other. Combat will consist of rounds of attacks and defenses until one player dies. Each round consists of consists of an attack die roll and a defend die roll from each player which determines how much damage is taken by the players.

Player 1 will always go first with an attack and Player 2 will defend. Then, if still alive, Player 2 will go next with an attack and Player 1 will defend. The game will check if players are alive after each attack/defense sequence.

Each character type has its own characteristics of Attack Dice (number and number of sides), Defense Dice, Armor (increase defense abilities), and Strength Points. Death of a character occurs when Strength Points is less than or equal to 0. Damage from an attack is calculated as Attacker's roll mins defender's roll mins defender's armor.

Special abilities:

Character	Special Ability
Vampire	Charm: Vampires can charm an opponent into not attacking. For a given attack there is a 50% chance that their opponent does not actually attack them.
Barbarian	None
Blue Men	Mob: Blue Men are a swarm of small individuals. For every 4 points of damage, they lose one defense die.
Medusa	Glare: If a Medusa rolls a 12 when attacking then the target instantly gets turned into stone and Medusa wins! If Medusa uses Glare on Harry Potter on his first life, then Harry Potter comes back to life.
Harry Potter	Hogwarts: If Harry Potter's strength reaches 0 or below, he immediately recovers, and his total strength becomes 20. If he were to die again, then he's dead.

Inheritance / Polymorphism Design:

The program contains a Character base class. It is an abstract class in which Vampire, Blue men, Barbarian, Medusa, and Harry Potter inherit from.

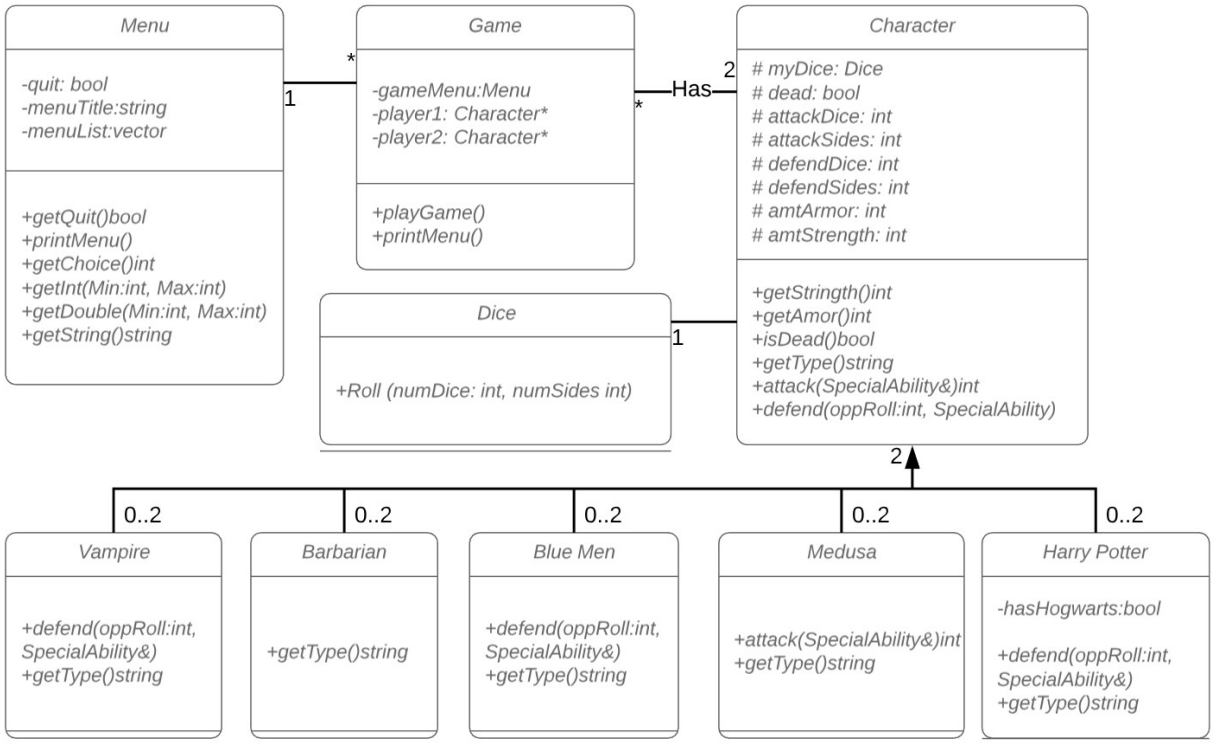
Character has the following methods:

- Constructor & Deconstructor
- Attack (Virtual)
- Defend (Virtual)
- Get Strength
- Get Amor
- Is Dead
- Get Type (Pure Virtual)

All 5 Character Types will inherit methods from the Character Class with the following specific cases of polymorphism:

- BlueMen Defend – Overrides to implement the Mob ability.
- HarryPotter Defend – Overrides to implement Hogwarts.
- Medusa Attack – Overrides to implement Glare.
- Vampire Defend – Overrides to implement Charm.
- All – Get Type – Overrides to return their character type for console output purposes only.

Class Hierarchy Diagram:



Other implementation notes:

The attack function returns an integer representing the attackers dice roll. This is passed as an argument to the defender in order to defend. To account for the special ability in the case for Medusa, the attack function accepts as an argument an enumeration of special abilities. This will allow for future implementation of other special attack abilities. If Medusa uses the Glare, the Special Abilities enumeration is flagged. This special ability use is then also passed to the defending player.

Program Flow:

#	Description	Function/Object Call
1	The main function of the program will create and loop through a general loop with two options. (1) Play the game. (2) Exit the game. The game will run first before asking the user to play again or exit.	Menu object instantiation Menu.getQuit Menu.printMenu Menu.getChoice
2	To play the game, the main function will call the Game.playGame method.	Game.playGame
3	The playGame method will start by asking the user two choose two types of characters. They can be the same or different. The method will instantiate two Character objects of the two types of characters chosen.	Game.printMenu
4	Then within the same method, a While loop will continue until one of the players is “dead”, providing for each combat round.	

Each combat round:

#	Description	Function/Object Call
1	First, the game will print out the description and stats for each player.	
2	Player 1 will “attack” Player 2. The function will return an integer value representing their attack roll. The special abilities enumeration variable will be changed if any attack special ability is used.	Character.Attack
3	Player 2 will defend this roll. The Character.defend method will accept 2 arguments: an integer for the attack roll, and the Special Abilities variable.	Character.Defend
4	The Game.playGame will check to see if any player died at this half-way point in the round by calling the Character.isDead method	Character.isDead
5	If Characters are not dead, then continue to remainder of the round by Player 2 attacking Player 1.	Character.Attack Character.Defend
6	Play will continue until one character is dead.	
7	Deconstruct the game, delete dynamically allocated memory. Return to Main Menu to ask the user to play again or exit the program.	

Testing Plan

The testing plan revolves around matching different characters with different characters and testing their special abilities, their attack rolls matches specifications, their defense roll matches specifications, and the amount of inflicted damage is calculated correctly and applied correctly.

Test Scenario	Inputs	Expected Results	Observed Results
Blue Men Mob	Blue Men vs Blue Men	Blue Men Mob affects a player's defense dice, reducing them from 3 to 2 to 1.	Successful.
Harry Potter Hogwarts	Harry Potter vs Barbarian	Harry Potter's strength points is less than 0 but regenerate to 20.	Successful.
Medusa Glare	Medusa rolls a 12 (6+6)	Medusa invokes Glare and the opponent dies. Game over.	Successful.
Vampire Charm	Vampire v Barbarian	Barbarian attacks, but no damage is inflicted because of Vampire Charm.	Successful.
Same Characters	Barbarian v Barbarian	Both characters are instantiated and fight each other.	Successful.
Vampire vs Barbarian	Vampire attack Barbarian defends.	Vampire rolls 5 Barbarian rolls 11 No damage is inflicted.	Successful.
Harry Potter vs. Blue Men	Blue Men attack Harry Potter defends	Blue Men Rolls 11 (2d10) Harry Potter rolls 4 (2d6) Damage is 7 – 0 = 7	Successful
Medusa vs. Harry Potter	Glare used within 1st life of Harry Potter.	Medusa uses Glare but Harry Potter stays a live with Hogwarts.	Successful.
Barbarian vs. Harry Potter	Harry Potter attacks Barbarian	Harry Potter rolls a 12 (2d6) Barbarian rolls a 2 (2d6) Damage is 12 – 2 – 0 = 10	Successful
Multiple games in 1 program run.	Barbarian v Barbarian Then Medusa v Harry Potter	Both fights are ran successful and independently of each other.	Valgrind produces an error of a possible memory leak.
	By the way Main was creating a Game object, the first instantiated Game object was not being deconstructed properly before the 2nd Game object was instantiated. To rectify, the Main function Menu Loop was rearranged to ensure Game objects were deconstructed then instantiated again to ensure each game is separate and all dynamically allocated memory was freed.		

Project Reflection

Problems encountered:

The primary design problem encountered was implementing the Special Abilities of each character. Because the characters are separate objects that are chosen at run-time, a solution that can handle any and all special abilities would be necessary.

For the defensive special abilities, such as the case with Blue Men, Harry Potter, and Vampire, these can be implemented within each polymorphic defend methods. However, Medusa's attack special ability required additional implementation steps.

Changes Made During Implementation

My first idea to implement the Special Abilities was to the attacking player as an argument in the defense function or pass the defending player as an argument in the attack function. However, this didn't accomplish what I hoped for.

Ultimately, I decided to use the Game class to pass information back and forth between the Characters instead of between the characters directly. This was implemented using the Special Abilities enumeration passed to each character by reference. The Character abstract class (and any polymorphic delivered classes) will then account for any special abilities used against them, particularly Medusa's Glare. This way will also allow for future implementation of additional characters or additional attacking special abilities.

Conclusion

This program served as a great example where polymorphism can reduce lines of code and complexity in code. In the case of the Barbarian class, both the Attack and Defend methods are inherited from the Character class, eliminating the need for additional code, debugging, and testing.