Project 3

In this project, our class was tasked to utilize inheritance to create a fantasy combat game. A base class would be used as the foundation for five other derived classes, and all creatures utilized the same functions, attack(), defend() and damageReceived() throughout the game. Some of these creatures had an inherent advantage through the special abilities they possessed. In the game, a user selected two creatures to combat each other. The outcome of the game was announced if one or both creatures died in combat and the user would then have the option to exit the game or play again.

My design for this program is to make a creature base class and have the Medusa, Harry Potter, Vampire, Blue Men, and Barbarian all stem from this base class. I created an attack roll function in my creature class, which would get the attack value from the offender and a synonymous defend roll function to get the defense roll from the defender. I also created a damage received function, which acts as an extension of the defense function. I divided the functionality of the defend function because I wanted to make the attack and defense function responsible for generating dice values and make the damageReceived() function use these values to calculate the damage inflicted on the defender. These three functions were shared amongst all creatures.

I made several revisions to my original design to address problems that arose during implementation. I initially made my damageReceived() function, attack() function, and defend() function pure virtual in my creature class, but noticed that the contents of the functions were shared by almost all my creatures. The functions only deviated on occasion for some of the creatures. To reduce the errors when copy and pasting my code and minimize the redundant aspects of my program, I decided to make these three functions virtual and hosted them all in my creature class. I made my creature class abstract by making the destructor pure virtual. I also did not have an isalive() function in my original design. This presented a problem because I initially wanted to make by play() function take place until one of the character's strength points became less than or equal to zero. However, upon implementing my code, I realized it was necessary to know when a creature was alive versus when its strength values were less than or equal to zero to account for Harry Potter's second life. Thus, I created an isalive() function, which was based on an additional life variable I created. This variable was created in my creature class as a protected member variable and was set to one for all creatures except Harry Potter. For Harry Potter, the variable was set to two. The isalive() function returned true if the life was greater than zero and false otherwise and more importantly, allowed by play function to work continuously until a creature died versus when a creature's strength points reached zero.

My final design begins by showing a menu to a user. A user is give the option to start the game or exit. If they choose to start the game, they get taken to another menu that allows them to choose two characters to fight or return to the main menu. The user can choose the same characters to fight as well. Once two players are chosen, one character is randomly selected to attack first. Afterwards, the play() function takes places, which allows both creatures to battle one another until one of them dies. The damage inflicted is calculated by the battle function. The battle function takes two creature pointers as arguments. The first argument is reserved for the attacker and the second argument is reserved for the defender. The attacker die roll values and the defender die roll values are taken as arguments to the damage received function, which is responsible for discerning how much damage to deduct from the defender's strength. If both creatures are still alive, the battle function executes again; this time the defender takes the attacker’s formers position and the attacker takes the defender’s former position in the function call. The creature therefore alternate attacking one another until one is victorious or a draw happens. The special abilities are also all accounted within each creature class without any dependencies or checkers throughout the program. The charm is elicited 50% of the time through a random feature in the vampire’s damage received function, which allows the creature to avoid any damages. Medusa’s charm takes place within the creature’s attack function whenever a 12 is rolled. The Blue Men utilize the get strength function to lose a die for every four damages points received. Harry Potter uses the damage received function to regenerate his life after his first death.

There are a few random elements in the program that are addressed. The player to initiate the attack is chosen randomly through the equation `randNumber = rand() % 4; if` randNumber is 1 or 3, player 1 attacks first. If randNumber is 0 or 2, player 2 attacks first. Other random elements take place when the attack and defense die are rolled. In both situations, the attack or defense value are obtained using a for loop, that account for the number of die sides and number of die. For example, if a creature has a defense die with six sides and has two defense die, then a random number will be generated using the equation rand() % defenseDieSides + 1. This equation will execute twice since there are two die and the sum of the values from the equation will constitute the defense die roll value. A similar method is used to elicit the glare and charm special abilities. If the attack die randomly is set to the value 12, then the glare ability with enact. For the charm ability, the random fifty percent chance the power is enacted takes place using the same methodology used to determine which player attacks first. To ensure that the random elements executes properly in my program, I seeded srand my main function.

I addressed other random elements in my program by observing the likelihood of how often a special ability would occur, so I could display the desired output and debug it accordingly. For the blue men, I printed each time the character lost a die in the combat to ensure the die was being lost correctly. For the vampire's charm, I ran around twenty combats to ensure the charm would happen around 50% of the time per game. For Medusa's glare, I ran approximately twenty rounds and as expected, saw the glare appear occasionally. When it did, the other character died. Harry Potter, however, regenerated if he lost just his first life. I also played Vampire and Medusa against multiple rounds until I could see glare being acted against charm. As expected, the charm trumped the glare.

Creating this program required addressing random elements and utilizing polymorphism and inheritance. Addressing these factors facilitated building out the remaining aspects of this program, which aimed to integrated the former elements.

Test Plan

The test plan covers all combat scenarios amongst the creatures, including when the creatures fight themselves. To comprehensively test my creatures, I ran several tests until all special abilities were provoked. I also made cout statement for the rand() functions in the initial debugging phase to ensure reasonable values were outputted and were within the designated ranges.

The obtained test plan results are sensible. The Blue Men is more inclined to beat any other creature because of its high die values. The vampire's charm gets invoked almost half the time, and Medusa's glare takes place occasionally. In addition, Barbarian and Medusa are both weaker creatures that have a higher likelihood of losing.

I addressed the random factor in this game my making my special characters play multiple rounds against one another until their abilities were invoked. I addressed the random factor when Harry and Medusa fight by making the players combat each other around 12 times. Since Medusa has an 8% chance of using glare, I entered these creatures in multiple combats against each other until glare was invoked. When Medusa used glare in one of the combats, Harry was brought back to life. I also pitted Medusa and Vampire for five rounds of combat until glare and charm were both used in the same round. As expected, vampire incurred zero damages when charm was set against glare. I was able to observe the Blue Men lose all their die when they played against themselves.

Test Case	**Driver Functions**	**Observed Outcome**
Vampire vs. Vampire	play(), battle(), attack(), defend(), damageReceived()	13 rounds; charm evoked 7 times total. Player 1 wins.
Harry Potter vs. Harry Potter	play(), battle(), attack(), defend(), damageReceived()	40 rounds; Player 2 and Player 1 lose their first lives; Player 2 wins.
Blue Men vs. Blue Men	play(), battle(), attack(), defend(), damageReceived()	8 rounds; Player 2 wins. Player 1 blue man incurred 5 points of damage and lost one die. Expected this to have less rounds as both blue men roll bigger attack and defense values in the beginning.
Blue Men vs. Blue Men	play(), battle(), attack(), defend(), damageReceived()	12 rounds. Player 2 wins. Player 1 loses two die.
Barbarian vs. Barbarian	play(), battle(), attack(), defend(), damageReceived()	8 rounds; Player 2 wins. No armor so rounds are short. Die values within appropriate range 1-12.
Medusa vs. Medusa	play(), battle(), attack(), defend(), damageReceived()	8 rounds; player 2 wins. Glare took place once from player 2. Attack die values within appropriate range 1-12. Defense die within appropriate range 1-6.

Vampire vs. Barbarian	play(), battle(), attack(), defend(), damageReceived()	4 rounds; Vampire wins. Charm took place once. Rolls within appropriate range.
Vampire vs. Blue Men	play(), battle(), attack(), defend(), damageReceived()	10 rounds; blue men wins because its attack roll has a higher rolling capacity than vampire.
Vampire vs. Medusa	play(), battle(), attack(), defend(), damageReceived()	12 rounds. Vampire wins. Charm invoked 4 times.
Vampire vs. Medusa	play(), battle(), attack(), defend(), damageReceived()	21 rounds. Medusa wins. Medusa used glare in the bbeginning, but vampire blocked it with charm.
Vampire vs. Harry Potter	play(), battle(), attack(), defend(), damageReceived()	37 rounds. Vampire wins. Harry loses life in round 17. Charm invoked 12 times.
Barbarian vs. Vampire	play(), battle(), attack(), defend(), damageReceived()	17 rounds. Vampire wins. Charm used 6 times.
Barbarian vs. Blue Men	play(), battle(), attack(), defend(), damageReceived()	8 rounds. blue men wins because its attack roll has a higher rolling capacity than barbarian.
Barbarian vs. Medusa	play(), battle(), attack(), defend(), damageReceived()	6 rounds. Barbarian wins. Understandable since medusa has lowest strength point amongst character and GLARE wasn't enacted.
Barbarian vs. Harry Potter	play(), battle(), attack(), defend(), damageReceived()	38 rounds. Harry Potter wins. Harry loses a life.
Blue Men vs. Vampire	play(), battle(), attack(), defend(), damageReceived()	22 rounds. Blue men wins. Charm enacted 12 times. Blue Men lost one die.
Blue Men vs. Barbarian	play(), battle(), attack(), defend(), damageReceived()	1 round. Blue men wins by rolling 18 attack die.
Blue Men vs. Medusa	play(), battle(), attack(), defend(), damageReceived()	3 rounds. Blue men wins.
Blue Men vs. Harry Potter	play(), battle(), attack(), defend(), damageReceived()	12 rounds. Blue men wins

Harry Potter vs. Vampire	play(), battle(), attack(), defend(), damageReceived()	Harry Potter wins. HP lost one life. Vampire used charm 12 times.
Harry Potter vs. Barbarian	play(), battle(), attack(), defend(), damageReceived()	13 round. Harry potter wins.
Harry Potter vs. Medusa	play(), battle(), attack(), defend(), damageReceived()	14 rounds.Medusa wins. Harry potter loses first life is round 7. Medusa uses glare afterwards.
Harry Potter vs. Medusa	play(), battle(), attack(), defend(), damageReceived()	8 rounds. Harry potters wins. Harry Potter keeps both lives.
Harry Potter vs. Blue Men	play(), battle(), attack(), defend(), damageReceived()	9 rounds. Blue men wins.