Robert Newton
11/07/2016
CS162

Design Plan:

The main class here is a creature class and five classes derived from the creature class. Those classes are Harry Potter, Vampire, Medusa, Barbarian and Blue Men. We are creating a fantasy combat game using these characters. Each character has characteristics pertaining to attack, defense, armor and strength points.

Type	Attack	Defense	Armor	Strength Points
Vampire	1d12	1d6*charm	1	18
Barbarian	2d6	2d6	0	12
Blue Men	2d10	3d6	3	12*mob
Medusa	2d6*glare	1d6	3	8
Harry Potter	2d6	2d6	0	10/20*hogwarts

A little explanation from the assignment explanation, the first number is the amount of dice, and the third number is the amount of sides on the dice. There will be an option for the user to choose one of these characters or get a further explanation of the characters, which we will put in our main file.

1. Suave, debonair, but vicious and surprisingly resilient!
2. Think Conan or Hercules from the movies. Big sword, big muscles, bare torso.
3. They are small (6” tall), fast and tough. So they are hard to hit and can take some damage. As for the attack value, you can do a LOT of damage when you can crawl inside the armor or clothing of your opponent.
4. Scrawny lady with snakes for hair. They help with fighting. Just don’t look at her!
5. Why are you reading this? How can you not know who Harry Potter is?

Creature Class:

Protected members:

attDice, defDice, armor, str, cre

Public:

Creature(int, int, int, int, int,string)
type()
Virtual int attack
Virtual bool defense(int)

Robert Newton
11/07/2016
CS162

Each class throughout the program will have to inherit from creature class in order to play through the fantasy game. Therefore, we must implement each class with their own take on the differences their attack, defense, armor, strength have depending on what the user chooses.

Each character has their own special effects,

*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.

*Glare: If a Medusa rolls a 12 in attack, then the target has looked her in the eyes and is turned to stone. The Medusa wins!

*Mob: The Blue Men are actually a swarm of small individuals. For every 4 points of damage (round down), they lose one defense die. For example, when they reach strength of 8 they only have 2d6 for defense.

*Hogwarts: If Harry dies (i.e. strength <=0), he immediately recovers and his total strength becomes 20. If he were to die again, then he's dead.

Testing plan:

Now my testing plan is simple, break up each class, implement them one by one, to make sure we are moving along correctly. We need to print out that the damage is going through correctly, this also means we need to make sure the user who wins is also printed out at the end of the game. I also need to test and see who wins more games, The problems I can think of, mainly in my case is getting mixed up with all the different classes in this program. I also need to make sure each roll and each special attack takes effect without issue as well. I am also going to make sure I work on my code organization this week too. I have a bad habit of putting things off, focusing on other things and other classes, or worst of all playing Skyrim.

Testing Results:

This time, the coding went pretty well, I wanted to take this moment to work on my code organization as well as my actual coding. This was a major thing I needed to personally work on this week. I made sure that all my extracurriculars were powered off, and I added a TODO plugin to my vimrc. This allowed me to continue to check things off and move through to the next issue in my realm. After finishing my code, the biggest issues I had was making sure the characters were lined up correctly, their bonuses and attacks were correct and listing out who won. The way I solved this was tracing my code, making sure the information was all lined up and finally making sure the test cases were compiling correctly. This was a test of my fundamental programming, breaking things down and printing each separate class and ensure that each function was correctly compiling.