# Project 3 Reflection

Byron Kooima - 932707491

CS/162

August 06, 2017

Prof. Luyao Zhang

Project 3 Reflection

## Lab Description

The main focus of this program is to implement a fantasy combat game simulation. The universe for the game contained the following Creatures:

| 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 |

The implementation of the program would start with a Creature class and flow that down to each individual creature (Vampire, Barbarian, Blue Men, Medusa, and Harry Potter). The functional design needed to include a structure for the creatures to battle each other. Each of the creatures vary only in the values contained in the above table. The special attacks will override some of the attack or defense functions. Additionally, the entire combat simulation will have a test driver to test the “round robin” arena where all creatures fight each other at least once.

## Requirements

- Make a Creature base class (abstract)
- Creature is made up of the following information:
    - strength
    - armor
    - name
    - Defense
    - Virtual Defense function
- Each Creature derived class has the following information:
    - A way to revive the creature

  - An Attack function (dice roll)
  - A Defense function (if needed)
- When O1 creature is fighting O2 creature, O1's attack function will be called and O2's defense function will be called
  - Defense function will take in attack damage and calculate the defense
  - O2's strength will be reduced by the resulting damage minus armor
- Create a menu to display the five fighters
  - User selects two fighters to fight each other
  - Can be two of the same Creature
  - Randomly select the fighter to attack first
  - Print the results of each round
- Create a test harness to walk through all possible combinations of fighters

## Class Design

| Class Name | Creature (base class) |
|---|---|
| Data Members | protected member variables:<br>(none)<br>private member variables:<br>**int strength**<br>**int armor**<br>**int defPoints**<br>**string name**<br>**string specialAttack**<br>**int dieNumAttack**<br>**int dieNumDefense**<br>**int dieSidesAttack**<br>**int dieSidesDefense**<br><br>public member variables:<br>(none) |
| Member Functions | private member functions:<br>(none)<br>protected member functions:<br>(none)<br>public member functions: |

| | |
|---|---|
| | **Creature()** – Constructor: Set default values<br>**virtual ~Creature() –** Destructor<br>**virtual int attack_roll() = 0 –** Determine the attack damage from the appropriate dice<br>**virtual void recover_strength = 0 –** Restore the creature to full strength<br><br>**string get_name** – Returns the creature name<br>**int get_strength** – Returns the creature strength<br>**int get_armor** – Returns the creature armor<br>**int get_defPoints** – Returns the creatures defense<br>**string get_special** – Returns the creatures special attack<br>**virtual int defense_roll** – Returns the total damage taken by the creature. Updates strength based on total damage taken<br>**int die_roll** – Return the random die values based on each creatures specific die criteria<br>**bool defeated** – Return a Boolean based on the current strength of the creature (<0 dead) |

| Class Name | Vampire |
|---|---|
| Data Members | protected member variables:<br>**(none)**<br>private member variables:<br>**(none)**<br>public member variables:<br>(none) |
| Member Functions | private member functions:<br>(none)<br>protected member functions:<br>(none)<br>public member functions:<br>**Vampire()** - Constructor: Set default values<br>**~Vampire()** - Destructor<br>**int attack_roll** – Returns an int based on a random roll of a single 12-sided die<br>**int defense_roll** – Takes a random roll of a single 6-sided die, adds it to the armor and then subtracts |

| | |
|---|---|
| | that from the attackers roll. Vampire has a 50% chance to cast Charm and take no damage<br>**recover_strength** – Restores the Vampires to full strength |

| Class Name | Barbarian |
|---|---|
| Data Members | protected member variables:<br>**(none)**<br>private member variables:<br>**(none)**<br>public member variables:<br>(none) |
| Member Functions | private member functions:<br>(none)<br>protected member functions:<br>(none)<br>public member functions:<br>**Barbarian()** - Constructor: Set default values<br>**~Barbarian()** - Destructor<br>**int attack_roll** – Returns an int based on a random roll of two 6-sided die<br>**recover_strength** – Restores the Barbarian to full strength |

| Class Name | BlueMen |
|---|---|
| Data Members | protected member variables:<br>**(none)**<br>private member variables:<br>**(none)**<br>public member variables:<br>(none) |
| Member Functions | private member functions:<br>(none)<br>protected member functions:<br>(none)<br>public member functions:<br>**BlueMen()** - Constructor: Set default values<br>**~BlueMen()** - Destructor<br>**int attack_roll** – Returns an int based on a random roll of two 10-sided die |

| | |
|---|---|
| | **int defense_roll** – Takes a random roll of three 6-sided die, adds it to the armor and then subtracts that from the attackers roll. Blue men lose one die for every 4 points of damage.<br>**recover_strength** – Restores the BlueMen to full strength |

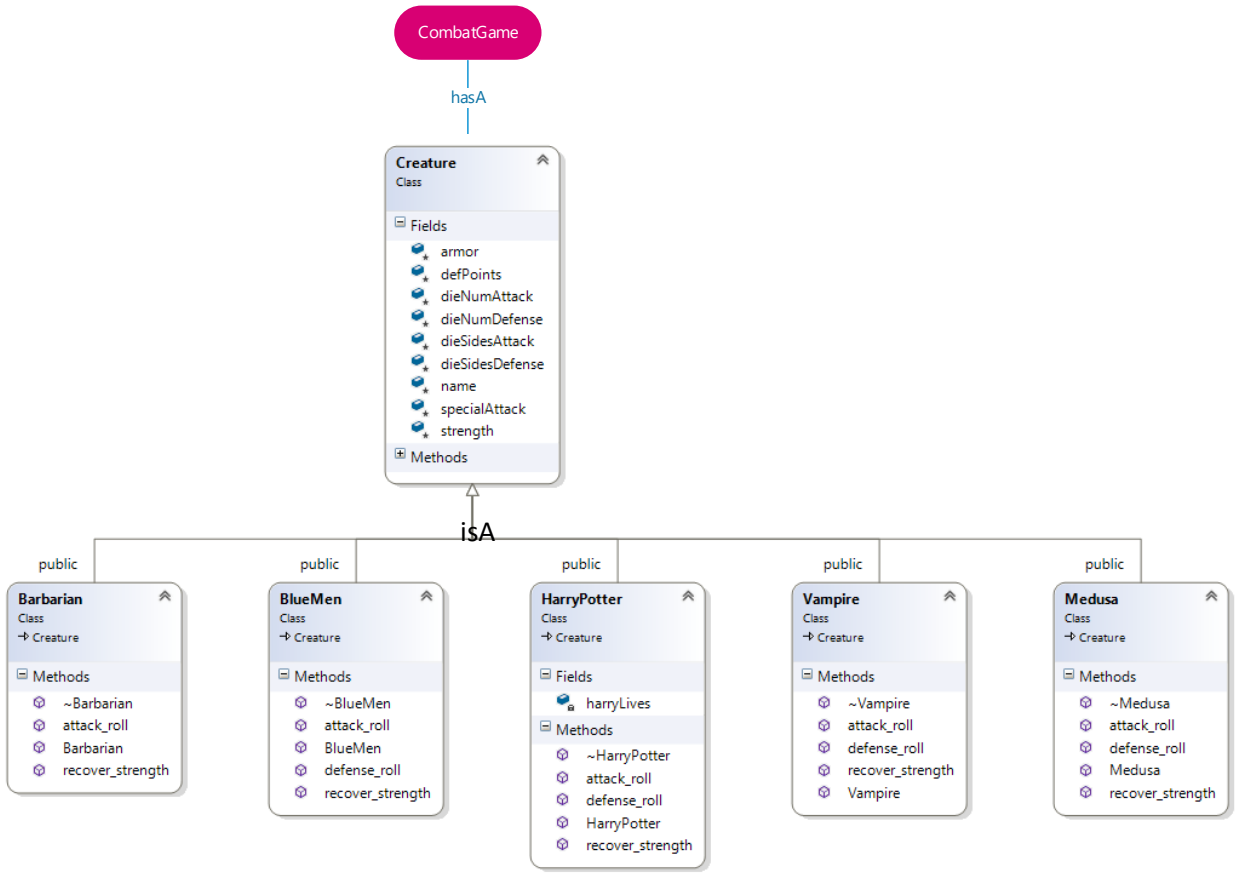
| Class Name | Medusa |
|---|---|
| Data Members | protected member variables:<br>**(none)**<br>private member variables:<br>**(none)**<br>public member variables:<br>(none) |
| Member Functions | private member functions:<br>(none)<br>protected member functions:<br>(none)<br>public member functions:<br>**Medusa()** - Constructor: Set default values<br>**~Medusa()** - Destructor<br>**int attack_roll** – Returns an int based on a random roll of two 6-sided die. Medusa will cast Glare if the roll is 12. Glare instantly defeats the opponent<br>**int defense_roll** – Takes a random roll of a single 6-sided die, adds it to the armor and then subtracts that from the attackers roll.<br>**recover_strength** – Restores the Medusa to full strength |

| Class Name | HarryPotter |
|---|---|
| Data Members | protected member variables:<br>**(none)**<br>private member variables:<br>**int harryLives** – Keeps track of Harry's lives for the Hogwarts special attack<br>public member variables:<br>(none) |
| Member Functions | private member functions:<br>(none) |

| | |
|---|---|
| | protected member functions:<br>(none)<br>public member functions:<br>**HarryPotter()** - Constructor: Set default values<br>**~HarryPotter()** - Destructor<br>**int attack_roll** – Returns an int based on a random roll of two 6-sided die<br>**int defense_roll** – Takes a random roll of two 6-sided die, adds it to the armor and then subtracts that from the attackers roll. HarryPotter revives if his strength is reduced to zero. He can only revive once.<br>**recover_strength** – Restores HarryPotter to full strength |

| Class Name | userMenu |
|---|---|
| Data Members | private member variables:<br>**int selectedChoice**<br>**vector<string> choice** |
| Member Functions | private member functions:<br>(none)<br>protected member functions:<br>(none)<br>public member functions:<br>**void add_choice(string) –** Add menu options to vector<br>**void printMenu() –** Print the menu options<br>**int makeChoice() –** Returns int for the user selection |

## Class Interactions




## Testing Plan

| Test Case | Input Values | Driver Functions | Expected Outcome | Observed Outcome |
|---|---|---|---|---|
| Input too low | **Input < 0** | **inputVerification -> SafeInput()** | **Prompt user to enter correct value** | **Prompt user to enter correct value** |
| Input not an integer | **Input = 0d** | **inputVerification -> SafeInput()** | **Prompt user to enter correct value** | **Prompt user to enter correct value** |
| Input below range | **Input = -1** | **inputVerification -> SafeInput()** | **Prompt user to enter correct value** | **Prompt user to enter correct value** |
| Input not a double | **Input = 34d** | **inputVerification -> SafeInput()** | **Prompt user to enter correct value** | **Prompt user to enter correct value** |
| User chooses option #3 to quit the program | **Input = 3** | **userMenu -> makeChoice()** | **Exit the program normally** | **Program exited normally** |

The addition aspects of my testing plan started out by taking each Creature and pitting it against another Creature of the same sub type (i.e. Vampire vs Vampire, etc.). This worked really well for allowing the Vampire to utilize the Charm attack (or defense in this case) against itself. This test initially failed since I couldn't get the Charm to override the damage caused. So even though it was still calling the Charm defense, the Vampire was still losing strength. This caused me to rethink the attack/defense design and my return statements from the defense function. After fixing it for the variations, the Charm worked as expected.

I then continued to build more and more Creatures and pitted them against a like Creature. This was the basic design of the program where a user would select each Creature from the list and pit them against each other for a single round. This worked well for initial testing where I was trying to determine if the characters variables were being set correctly. However, this did not give me good statistics of how often one Creature would overpower another.

After running a few simulations, I decided to create a test harness function that just loaded a vector with all the same creatures and iterated over loops to play 100 rounds. I did find that the randomness of the wins/losses would be lopsided at times, but over multiple attempts, the Vampire vs. Vampire evened out to ~50%.

When I added the Harry Potter creature test to my test harness, I had a lot of issues getting his lives to regenerate correctly. If he didn't use Hogwarts, his next round would start with a strength of 20. Eventually I found the correct solution for separating out the harryLives variable and keeping track of it (and resetting it) at the appropriate time.

My final test harness contained all of the creatures and played a "round robin" style where every Creature fought the other Creature. I did this by iterating between two vectors with

five Creatures each (one for each Creature sub type). The nested for loops created a tournament style with 25 rounds (See following table).

| | Vampire | Barbarian | Blue Men | Medusa | Harry Potter |
|---|---|---|---|---|---|
| Vampire | 0,0 | 0,1 | 0,2 | 0,3 | 0,4 |
| Barbarian | 1,0 | 1,1 | 1,2 | 1,3 | 1,4 |
| Blue Men | 2,0 | 2,1 | 2,2 | 2,3 | 2,4 |
| Medusa | 3,0 | 3,1 | 3,2 | 3,3 | 3,4 |
| Harry Potter | 4,0 | 4,1 | 4,2 | 4,3 | 4,4 |

## Reflection

The fantasy combat game started out simple enough. I started out just designing the Creature class and it seemed pretty straight forward. I then created a single Vampire derived class along with the initial combat main scenario. This allowed me to pit the Vampire against another Vampire just to see how they would interact. It took multiple iterations to get the initial Creatures to respond the way I intended.

I opted to create a pure virtual attack and defense functions to make sure I could grab the offense and defense roll and that it was diverse enough to handle the different Creature combinations. This initially worked very well but then I ran into problems when I tried to incorporate the “Special Attack” options for the Creatures. I also opted to have the defense roll take in the attack roll as an integer. The attack roll worked well as a purely virtual function, still keeping my Creature class abstract. However, it made sense to make the defense roll have an initial function definition that allowed the derived creatures to just utilize base class function.

One area that I found to be the most rewarding was creating the recursive function to handle the individual rounds. I mainly did this because of the requirement to randomly select the player to start the round. It took me the most time to incorporate and was probably overly complicated, but it was very awesome when it finally came together. Basically, I had to figure a way to switch the O1 and O2 Creatures back and forth within the function so that they could switch off their attacks and defenses. I had to draw out on paper the entire sequence of events before I could get the recursion to work. Then it was just a matter of making sure the variables were switching back and forth from one function call to the next. Since I had never done this with objects, I was concerned about overwriting or confusing each of the Creature member variables.

Once I finally got the recursion to work, the next challenge was getting the output to match. I came up with a cool way to check whether a round was even or odd to determine how the output would be displayed to the user. This allowed me to keep the O1 and O2 columns in line while switch attacks and defenses.

Finally, implementing the special attacks proved to be challenging because I didn't have a well thought out plan on handling those options. Since we were not allowed to have one Creature class know anything about another creature class, I had to find a solution where the Creature set a special attack variable. This worked really well and allowed me to use that output in my printing functions. The only special attack that did not work as well as I hoped was Medusa's Glare. I wanted to have her attack exactly the amount need to remove the opponent's strength. I had a solution early on, but because I refactored the attack/defense functions, I could no longer determine the defense points without first rolling the dice. To determine the defense points, I had to first roll the offense so I could feed it into the defense roll. Since I couldn't

override the strength of the opposing player, I could not get the logic to work correctly. Eventually, I just opted to set Medusa's offensive roll to 100 so that it would always defeat the opposing player (except when playing against Vampire's Charm).

Overall, this was a very challenging but fun program to implement. I learned a lot about bringing together multiple pieces of code from previous assignments. The building blocks were already there and it made me really appreciate the previous modules in this class.