

Thomas Mathew

Assignment 3: Design Document/Journal/Reflections

Many of the design decisions were essentially made for us due to the prompt of the assignment. We were to have a base class "Creature" with 5 child classes: Barbarian, Medusa, Vampire, Blue Men, and Harry Potter. In the "spirit" of inheritance, I started off by looking for ways in which the 5 child classes were similar, so that I could place those characteristics in Creature. However, each class is different in the way it attacks and defends, sometimes radically so. I also realized that Creature itself should never be instantiated, only the child classes, thus Creature only needed to have virtual functions.

One might ask, if I didn't use Creature to pass along many characteristics, why even have the inheritance relationship at all then? The answer to that became clear as I began to think about setting up the combat system. I realized that in order to have my core Battle Function work, it (like any other function) would need to know what type of parameters to expect. Obviously I would be looking to pass in objects such as a Vampire or Barbarian, but writing a function for every possibility would have been a chore - I would have needed to write 15 different functions for each scenario.

After creating my header and .cpp files for Creature, Barbarian, and Battle, I started off my testing by just declaring two Barbarian objects and passing them to the Battle function. At this point I wasn't using any pointers or really using the Creature class at all. As I will show below, I quickly started running into problems with values being lost, or as I realized later, sliced. Here are some steps I took to diagnose and eventually fix probably my biggest obstacle on this assignment:

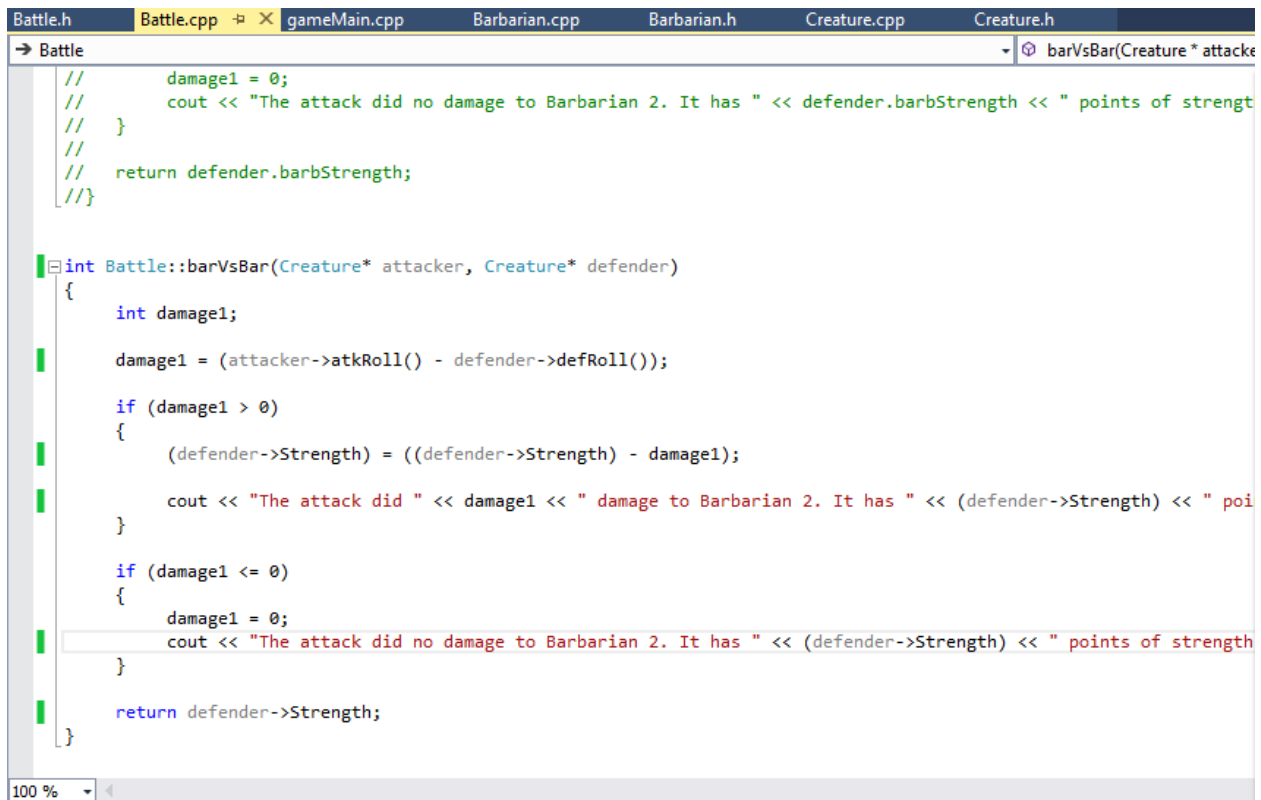
Step 1: The changes in Strength were not being saved:

```
flip3 ~/A5 22% g++ Battle.cpp Barbarian.cpp Creature.cpp gameMain.cpp -o battletest
flip3 ~/A5 23% battletest
2 barbarians will fight!
The attack did 2 damage to Barbarian 1. It has 10 points of strength left.
The attack did 0 damage to Barbarian 2. It has 12 points of Strength left.
The attack did 1 damage to Barbarian 1. It has 11 points of strength left.
The attack did 0 damage to Barbarian 2. It has 12 points of Strength left.
```

Step 2: I set up some "cout" statements to try and identify where the problem was happening.

```
current value is12
The attack did 2 damage to Barbarian 2. It has 10 points of strength left.
current value is10
current value is2
current value is2
The attack did no damage to Barbarian 2. It has 2 points of strength left.
current value is2
current value is0
current value is0
The attack did 1 damage to Barbarian 2. It has 11 points of strength left.
current value is11
current value is-11
current value is-11
The attack did no damage to Barbarian 2. It has -11 points of strength left.
current value is-11
current value is0
current value is0
The attack did no damage to Barbarian 2. It has 0 points of strength left.
```

Step 3: I realized that the program was slicing out data, the very concept we were going over in our group assignment. I had a definition for "Strength" as a variable in my Parent class, and the program was getting confused between that definition, and the "Strength" in my Barbarian class. I looked around online for examples of this problem and found that Pointers were the key in that they prevent slicing from becoming an issue. It was at this point that I realized that passing in pointers to the parent class of "Creature" would allow my battle function to be generic - that I would be able to plug in objects of any child class of parent; as long as they were defined correctly, passing by reference would allow the Battle function to use Child members correctly. Below is a screenshot of my change in this direction.



```

Battle.h  Battle.cpp  gameMain.cpp  Barbarian.cpp  Barbarian.h  Creature.cpp  Creature.h
→ Battle
//      damage1 = 0;
//      cout << "The attack did no damage to Barbarian 2. It has " << defender.barbStrength << " points of strength
//  }
//
//      return defender.barbStrength;
//}

int Battle::barVsBar(Creature* attacker, Creature* defender)
{
    int damage1;

    damage1 = (attacker->atkRoll() - defender->defRoll());

    if (damage1 > 0)
    {
        (defender->Strength) = ((defender->Strength) - damage1);

        cout << "The attack did " << damage1 << " damage to Barbarian 2. It has " << (defender->Strength) << " poi
    }

    if (damage1 <= 0)
    {
        damage1 = 0;
        cout << "The attack did no damage to Barbarian 2. It has " << (defender->Strength) << " points of strength
    }

    return defender->Strength;
}
100 %

```

Once I was able to nail down the syntax and get the battle function working with the parameters correctly, the hardest part of the assignment was over. My next steps were to introduce the other 4 child classes with all their unique features. For some classes, I found it easier to completely encase their unique behaviors within their class, while with others it was easier to use conditions in the battle function to manipulate the behavior of the program when those classes were in play. For example, I used a "harryLives" variable in Battle to make sure that any harryPotter object would only revive one time.

My biggest takeaway from this assignment is probably the power of inheritance in terms of creating versatile functions. I was able to create a Battle function that could accept 5 different types of Child objects as parameters and use their respective members correctly. Like I said previously it would have been much difficult to write out a different program for all 15 scenarios.