Discovering Requirements

- Implement a class hierarchy, starting with a “Character” class and flowing down to other creatures that will participate in combat.
- A mechanism for combat needs to be developed.
- Certain creatures need an override of their attack or defense functions
- Simulate combat with a test driver program

Design

Create a character class with the following attributes:
- Name
- Number of attack rolls
- Number of sides of attack dice
- Number of defense rolls
- Number of sides of defense dice
- Armor value
- Two dice objects (we made a dice class in a previous lab)
- Total and current points of strength
- Is the character alive (bool)

In addition, the character needs the following protected functions:
- Roll attack dice
- Roll defense dice

And the following public functions:
- Attack
- Defend
- Reset (get the character ready for another simulation (revive might also be an appropriate name)
- Accessors for name, strength points and alive member variables.

The attack and defense functions can be designed in such a way that the attack function takes a pointer to a character. The attack function then calls the defense function of that character, with the attack roll that it generated. Attack and defense need to be virtual functions, so that the proper attack and defense functions are used in special cases.

The shadow’s cloak is easy to account for. Simply add a 50/50 chance with the rand function for the function to prematurely return.

The goblin should just have an extra attribute indicating if it cut the tendon. There should be an implementation of the attack function that will check against the name and dice roll to see if the attribute should be set to true.

Testing

Testing this by hand would be tedious. Making a test driver program is the better way to go in this case. The test driver should have a function that takes two characters as an argument and simulates combat between the two. It should look like this:

- Char 1 attacks char 2
- Check to see if both alive
- Char 2 attacks char 1
- Check to see if both alive
- repeat

The loop should break when one is dead, and a win should be recorded for whichever character is still standing. To be fair, the test driver must alternate who starts the attack each simulation. These results can be stored in a vector and then the average can be taken to find out the statistics of combat.

Reflection

This assignment was fairly straightforward, as long as one had a design in mind before starting to code. Having an idea of how all the classes fit together was immensely helpful when I actually started to type things out. It seems like my design closely matched my final product, and I don't remember there being any gotchas that I hadn't accounted for. Perhaps the most annoying thing was having to queue up all the simulations in my test driver.

My numbers turned out to be what I would expect, and matched those of my peers. In real life, there would probably be someone who worked out what the proper numbers should be, and then one would include a threshold in the test driver to either pass or fail the test. I just used nice round percentages to make sure that my characters were passing.

This was a fun project! I'm excited to see what we do with these classes in the next assignment.