Matthew Solbrack
cs162/400 Intro to Computer Science
Reflection Project 3 - Fantasy Combat Game

Design:

Classes (And the functions that belong in them)

Character (Base/Abstract Class):

```
    virtual string getType () = 0 // Pure virtual function
    virtual getAttackNumberOfDice () = 0 // Pure virtual function
    virtual getAttackSidesOfDice () = 0 // Pure virtual function
    virtual getDefenseNumberOfDice() = 0 // Pure virtual function
    virtual getDefenseSidesOfDice() = 0 // Pure virtual function
    virtual getArmor()  = 0 // Pure virtual function
    virtual getStrengthPoints() = 0 // Pure virtual Function
    virtual string getSpecialAbilities() = 0 // Pure virtual Function
```

```
Barbarian : public character ; takes all above functions
Vampire: public character ; takes all above functions
Blue Men : public character ; takes all above functions
    getMob
Medusa : public character ; takes all above functions
    getGlare
Harry Potter: public character ; takes all above functions
    getHogwarts
```

Menu:

```
    Member Variables: int playerOneScore, int playerTwoScore;
                      character * playerOne, character * playerTwo
    void menuOne()
            1.  Play game
            2.  Exit the game

    void menuTwo()
        To play the game you must pick two characters from the following list:
        1. Vampire
        2. Barbarian
        3. Blue Men
        4. Medusa
        5. Harry Potter
        Please choose your 1st character [1 through 5]:
        Please choose your 2nd character [1 through 5]:
```

void menuThree()

1. Play again
2. Exit the game

void gamePlayMenu()

1. Attacker type.
2. Defender type, armor, strength point.
3. The attacker's attack dice roll.
4. The defender’s defend dice roll.
5. The total inflicted damage calculation.
6. The defender’s updated strength point amount after subtracting damage.

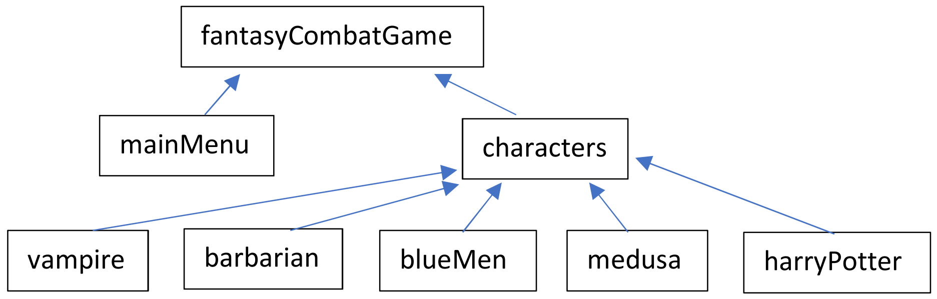
dice:

dice(int numberOfSides)
int twelveSided();
int sixSided();
int tenSided();

<u>Test Table:</u>

Test Case	**Input Values**	**Expected Outcomes**	**Observed Outcomes**
Input Negative	-1, -22, -5	“Try Again”	“Try Again”
Input at 0	0	”Try Again”	“Try Again”
Input in Current Range	When Prompted: 1 or 2 When Able: 3,4,5,6	Should go through the program and bring you to the menu to run the simulation again and again and again	The game executed as desired.
Input low	2, 3, 4, 5	Should go through the program and bring you to the menu to run the simulation again.	The game executed as desired.
input extremely High	1000, 99, 55	”Try Again”	”Try Again”, this input is too high for this game
other than integer	#, W,	“Try Again”	“Try Again”

Class Hierarchy



Reflection:

This was a fun program to build. There were a few changes that I made to the original design to make the program work more efficient. I will explain these changes below.

Special Abilities - While I was writing the program I decided to take out the special abilities from the character classes. It seemed easier to write them directly into the main game play class. It might be different if there were 10's to 100's of characters with special abilities that overlapped\. Then it might make sense to keep them in their perspective character class. But, since there were only 4 special characters, I feel that it was easier to just put those abilities directly into the main game play.

Dice Class – Taking out the dice class just made sense. It did not save much time or space to separate out the dice roll from the main game play class.

All the other classes and functions, not mentioned above, fell right into place. I feel comfortable with building classes and polymorphic/virtual functions. In my mind they work kind of like an excel spread sheet. The base class is sort of like the titles of the columns for the spreadsheet. And, the children classes, are the actual data. It all works very well.