Name: Carlos Carrillo-Calderon
Date: 11/08/2015
Course: CS_162

Assignment 3

Design Description:

For this analysis, I will follow the parameters defined in the book *C++ Early Objects, 2014*. Thus, an Object-oriented analysis will be performed in order to determine the requirements for this particular system, and clarify what it must be able to do, what it needs, and how the classes created for this program are related. Also, I will specify what the classes will carry out and their responsibilities.

Description of the problem

I had to design, implement, and test a simple class hierarchy as the basis for a fantasy combat game. The game "universe" contains Goblins, Barbarians, Reptile People, Blue Men and others. Each will have characteristics for attack, defense, armor, and strength points.

Identify classes and objects

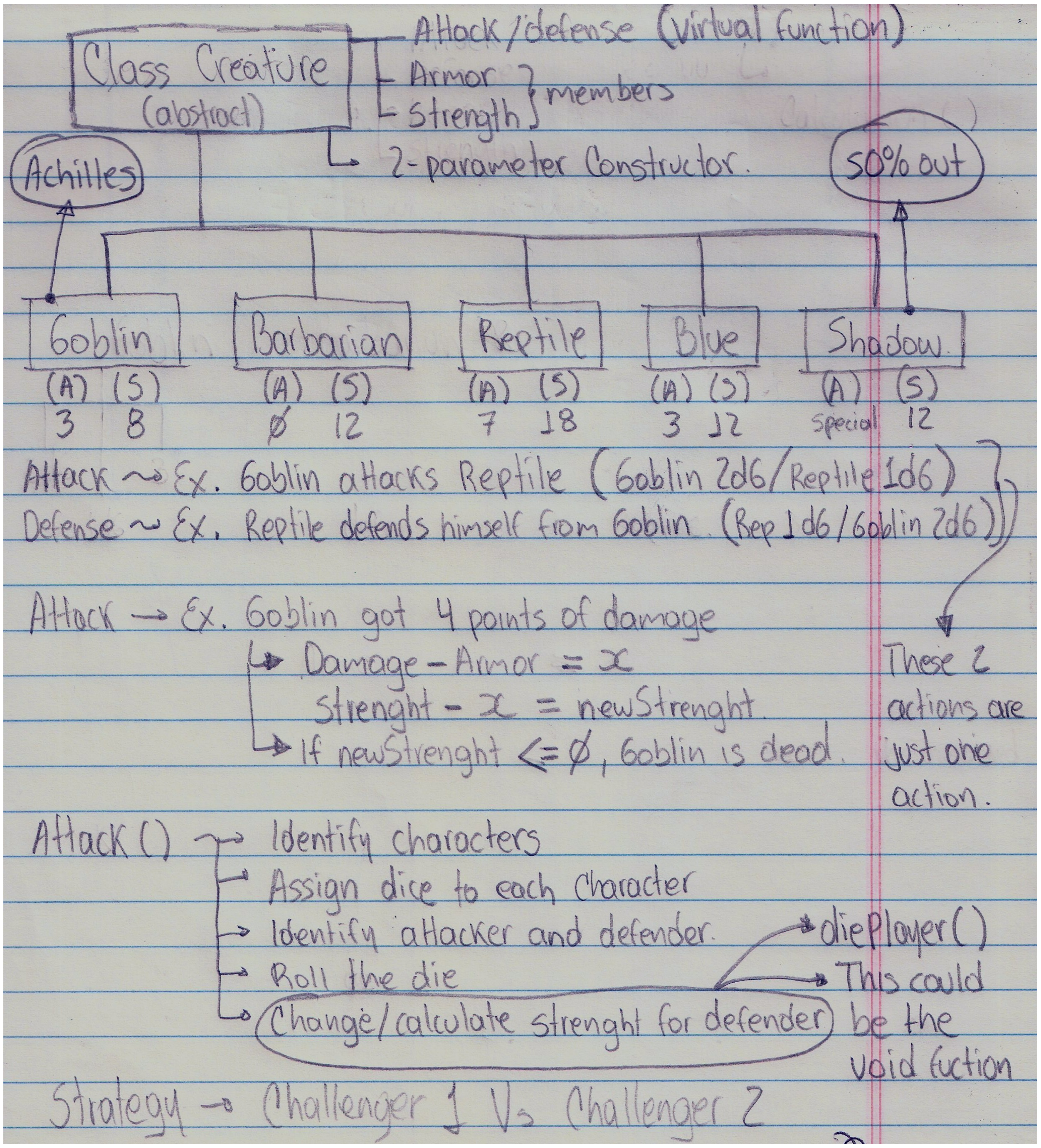
By definition, a class, in programming, is a package that consist of data and procedures that perform operations on the data, and a set of code instruction to model real-work objects (Gaddis et al, 474). For this program, it is necessary to create a base class and a subclass for each of the characters of the game. The base class will be an abstract class, so it will never be instantiated. Each subclass will vary only in the values in the table. Since each subclass starts with the same data elements, only one constructor is needed. As part of my design, I decided to create just 1 pure virtual function, and 1 void function whose purpose is not to use the constructor for processing all the data related to the characters (I took that idea from Lab6 encryption assignment).

On the other hand, I decided to have 2 class member variables: The Strength Points variable (associated to each of the characters) and the Achilles Power variable (only associated with the Goblin character).

Initial Design (very first sketch)

In order to learn the good habit of thinking about the application before starting to code, I crated this draft design (as requested) in which I started to build the whole concept of the game in my mind. Even though this draft is not very detailed, I may say this was the skeleton of the program. The first thing I obviously thought about was the base function. As I mentioned above, I had a clear idea about how a base class could be because of the program I made for Lab6. So my first idea (which I kept until the end of the project) was to create a virtual class to handle both the attack and the defense actions since I realized that it was actually unnecessary to create an attack and a defense class separately since they both actually will perform kind of the same action, which was to subtract values in order to change the Strength variable. I knew the only difference between attack and defense was the order in which the values were going to be taken depending on what character was performing the attack and what character was self-defending from the attacker.

Also, it was clear for me that the Strength variable had to definitely be a class member from the base class so all the subclasses could have access to this variable. For that reason, I would have to create a constructor for this variable anyway. Finally, I have to say that I first thought that the Armor value should also be a base class member, but later I realized I was wrong regarding that idea. Thus, this was my first sketch:



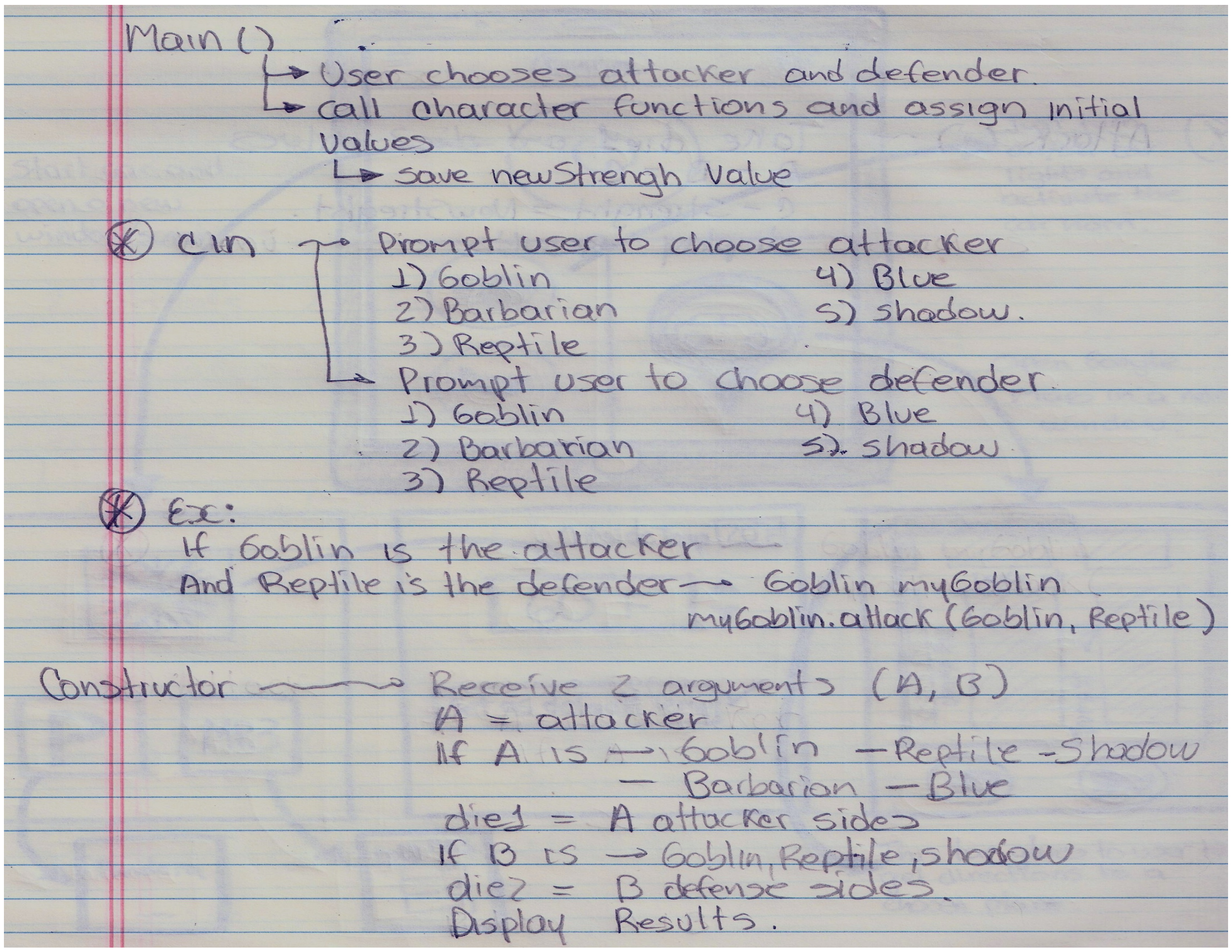
Class Creature (abstract)
Attack/defense (virtual function)
Armor
Strength
members
2-parameter Constructor.
Achilles
50% out
Goblin
Barbarian
Reptile
Blue
Shadow.
(A) (S) 3 8
(A) (S) ∅ 12
(A) (S) 7 18
(A) (S) 3 12
(A) (S) Special 12
Attack ~ Ex. Goblin attacks Reptile (Goblin 2d6/Reptile 1d6)
Defense ~ Ex. Reptile defends himself from Goblin (Rep 1d6/Goblin 2d6)
These 2 actions are just one action.
Attack → Ex. Goblin got 4 points of damage
Damage - Armor = x
strenght - x = newStrenght.
If newStrenght <= ∅, Goblin is dead.
Attack() → Identify characters
Assign dice to each Character
Identify attacker and defender.
Roll the die
Change/calculate strenght for defender
dieProyer()
This could be the void fuction
Strategy → Challenger 1 Vs Challenger 2

Another false assumption I had was to do the roll of the die in the attack function. I realized that it would be too much for the Attack() function to handle. Besides, I wouldn't have any other action to perform for my regular class function, which I called diePlayer(). Finally, I later decided to created a class member variable for the Achilles attack. This variable will change if Goblin gets an attack score of 12 when rolling the dice. For that reason, I had to include the Achilles variable in each overloading Attach() function from each subclass:

```
damage = attackRollScore/achilles – defenseRollScore;
```

Thus, the Achilles member variable will affect all the subclasses in case it changes value from 1 to 2, which would make the attacker score get halved (as required).

On the other hand, I also had to make a sketch for create a test driver program to create character objects, which would make attack and defense rolls required to show your classes work correctly. First, since the rolling die action was going to be a repetitive action, I decided to create a member function exclusively to perform this action instead of doing it every single time the user choose a particular character. As I mentioned before this function was called diePlayer() and it was only include in the base virtual class. So this was my first sketch of the testing program:



```
Main ()
    → User chooses attacker and defender.
    → call character functions and assign initial
      Values
        → save newStrengh Value

(*) cin → Prompt user to choose attacker
            1) Goblin              4) Blue
            2) Barbarian           5) shadow.
            3) Reptile
        → Prompt user to choose defender.
            1) Goblin              4) Blue
            2) Barbarian           5). shadow
            3) Reptile

(*) Ex:
    If Goblin is the attacker —
    And Reptile is the defender → Goblin myGoblin
                          myGoblin.attack (Goblin, Reptile)

Constructor → Receive 2 arguments (A, B)
              A = attacker
              If A is → Goblin  – Reptile – Shadow
                      – Barbarion – Blue
              die1 = A attacker sides
              If B is → Goblin, Reptile, shadow
              die2 = B defense sides.
              Display Results.
```

However, as expected, I had to make some change to this initial design since certain thing did not work as expected. After doing some changes (which will be mention later below) this was the final design I apply for the whole project:

Class's Attributes Definition
Now it is time to define the attributes or the data elements used to describe the objects instantiated from the classes defined above. Here are the class specifications for base class Creature and the other 5 subclasses: Barbarian, Reptile, Blue (for Blue men), Goblin, and Shadow:

```
Class name: Creature
Attributes:    fuerza     //store the Strength value for each character.
               achilles   //triggers the Achilles attack from Goblin.

Class name: Reptile
Attributes:    //the same as Crature

Class name: Blue
Attributes:    //the same as Crature

Class name: Goblin
Attributes:    //the same as Crature

Class name: Shadow
Attributes:    //the same as Crature
```

Class's Behaviors (Functions)
Since now the attributes have been defined, it is time to identify the activities or behaviors each class should perform. In software terms, these behaviors are called **functions**.

1) The functions in the **Creature** class should have these behaviors:

- Mutate, get, and return the double variable Strength (fuerza).
- Mutate the int variable achilles.
- Calculate the Strength value of each warrior/character after each attack.
- Roll the dice corresponding to each warrior/character.

Consequently, these are the functions/methods to be created for the **Creature** class:

- **setFuerza(double):** This function mutates the member variable fuerza(Strength).

- **double getFuerza():** This function returns the member variable fuerza(Strength).

- **setAchilles(int):** This function mutates the member variable achilles.

- **virtual double attacks (double, double) = 0:** This is a pure function that is overloaded or redefined according to fight features from each character.

- **Void diePlayer(int, int):** This function performs the actual die rolling according to number of dice and side corresponding to each character.

2) The only function in the **Barbarian, Reptile, Blue Men, Goblin,** and **Shadow** classes is **double attacks (double, double)** and it will obviously have very similar behaviors in each subclass, but in general this function will:

- Calculate the Strength value of each warrior/character after each attack.

UML Class Diagrams
Now that it has been possible to identify the class members and the behaviors that they should perform, it is also possible to draw a UML class diagram in order to visualize the **class members** and **functions** that each class should have.

It is important to highlight that the minus sing to the left of each attribute indicates that it is a private member. Similarly, the plus sign to the left of each function indicates that it is a public member. Thus, this could be the UML (Unified Modeling Language) diagram for the Creature class:

Creature
(protected) fuerza (protected) achilles
+Creature(): +Creature (double, int): +setFuerza(double):void +setAchilles(int):void +getFuerza():double +attacks(double, double):virtual double +diePlayer(int, int):void

Barbarian
- fuerza (inherited from Creature) - achilles (inherited from Creature)
+attacks(double, double):virtual double

Reptile
- fuerza (inherited from Creature) - achilles (inherited from Creature)
+attacks(double, double):virtual double

Blue Men
- fuerza (inherited from Creature) - achilles (inherited from Creature)
+attacks(double, double):virtual double

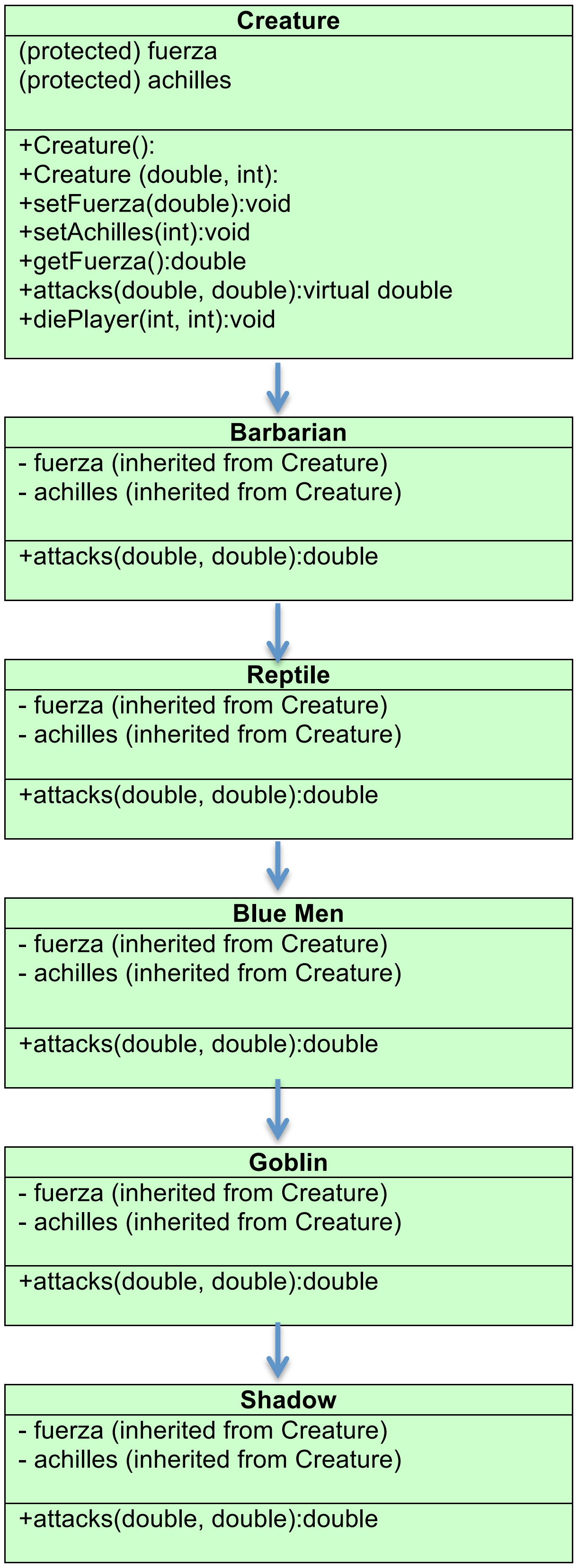
Goblin
- fuerza (inherited from Creature) - achilles (inherited from Creature)
+attacks(double, double):virtual double

Shadow
- fuerza (inherited from Creature) - achilles (inherited from Creature)
+attacks(double, double):virtual double

Relationships between classes

Now is time to determine the possible relationships that exist between the 6 classes. As established by Gaddis (2014), there could be three types of formal relationships among classes Access (Uses-a), Ownership (Has-a) and Inheritance (Is-a). In terms of access, the UML diagrams above clearly shows the private and public attributes that are related between the 6 classes. Thus, it could be said that there is an **Inheritance (Is-a)** relationship between the 6 classes. As shown in the UML diagrams above, Creature is the mother/base class of all the other classes. Actually the fact that the base class is an abstract class means that it can't work by itself since it cannot be instantiated. Reciprocally, the children classes can only work through the mother-base class. Additionally, all the class members from the base class are inherited by the subclasses, so it is possible to say that this is a perfect example of an **Inheritance (Is-a)** relationship.

This type of relationships can give us a clue of how a class ***hierarchy diagram*** may look like for this program:


Creature
(protected) fuerza
(protected) achilles
+Creature():
+Creature (double, int):
+setFuerza(double):void
+setAchilles(int):void
+getFuerza():double
+attacks(double, double):virtual double
+diePlayer(int, int):void
Barbarian
- fuerza (inherited from Creature)
- achilles (inherited from Creature)
+attacks(double, double):double
Reptile
- fuerza (inherited from Creature)
- achilles (inherited from Creature)
+attacks(double, double):double
Blue Men
- fuerza (inherited from Creature)
- achilles (inherited from Creature)
+attacks(double, double):double
Goblin
- fuerza (inherited from Creature)
- achilles (inherited from Creature)
+attacks(double, double):double
Shadow
- fuerza (inherited from Creature)
- achilles (inherited from Creature)
+attacks(double, double):double

Main Function
In order to articulate the whole design, I created this overview of the main function or driver program to create character objects, which make attack and defense rolls required to show my classes work correctly:

1. Create a main menu: In this menu I prompt the user to choose both the attacker and the defender. In order to do this, I assigned a number to each warrior in the following order:

   PLEASE SELECT THE ATTACKER:
   "Enter 1 to attack with BARBARIAN.
   "Enter 2 to attack with REPTILE."
   "Enter 3 to attack with BLUE MEN."
   "Enter 4 to attack with GOBLIN."
   "Enter 5 to attack with THE SHADOW."

   PLEASE SELECT THE RIVAL CREATURE:
   "Enter 1 to fight against BARBARIAN."
   "Enter 2 to fight against REPTILE."
   "Enter 3 to fight against BLUE MEN."
   "Enter 4 to fight against GOBLIN."
   "Enter 5 to fight against THE SHADOW."

2. Create a conditional for each possible fight: Although it was very time consuming, I believed this was the most effective way to simulate possible fight scenarios. Additionally, I decided to set the initial Strength values of the fighters inside this conditional in order to make things easier (in my opinion). For instance, if the user wants Barbarian (1) to attack Reptile (2), the corresponding conditional would be:

```
if(choice1 == 1 && choice2 == 2)
 {
     attacker = "BARBARIAN";
     defender = "REPTILE";
     strengthAttack = 12;
     strengthDefense = 18;
```

   Where **choice1** corresponds to the attacker and **choice2** corresponds to the defender.

3. Create class objects to make a single attack: For this task, I created objects in terms of the creature that was receiving the attack. The reason why I did this is because the Strength value changes only in terms of the defender but not the attacker. So this was the code I used to make one of the creatures (Reptile) attacks to another creature (Barbarian):

```
Barbarian myBarbarian(strengthDefense, 1);
myBarbarian.diePlayer(2, 1);
```

It is important to clarify that the parameter value located next to the Strength parameter is the Achilles value/parameter, which only changes to 2 when Goblin scores 12 during an attack.

4. Create class objects to make a fight: Since a fight is only a fight when the 2 contenders attack to each other, I had to come up with an idea to make that possible. And the solution was simple but effective: I switch the character numbers in the class call and the strength values in the constructor. So this is the code I used to simulate a "round" between the Reptile and Barbarian:

   ```
   Reptile attacks/Barbarian self-defends:
   strengthDefense = fuerzaDefense;
   Barbarian myBarbarian(strengthDefense, 1);
   myBarbarian.diePlayer(2, 1);
   fuerzaDefense = myBarbarian.getFuerza();

   Barbarian Attacks/Reptile self-defends:
   strengthAttack = fuerzaAttack;
   Reptile myReptile(strengthAttack, 1);
   myReptile.diePlayer(1, 2);
   fuerzaAttack = myReptile.getFuerza();
   ```

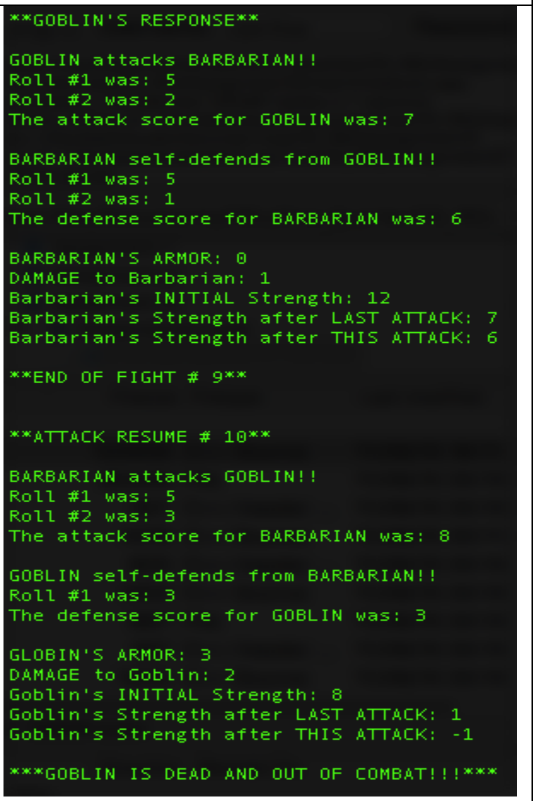
5. Create a look to continua the fight: Finally, I decided to create a loop to give the user the option to continue the fight until one of the warriors dies (Strength <0). In my original design, I planned to give the user the option to continue the fight manually or automatically until one of the warriors dies, and I could successfully implement that idea until the end of the project:

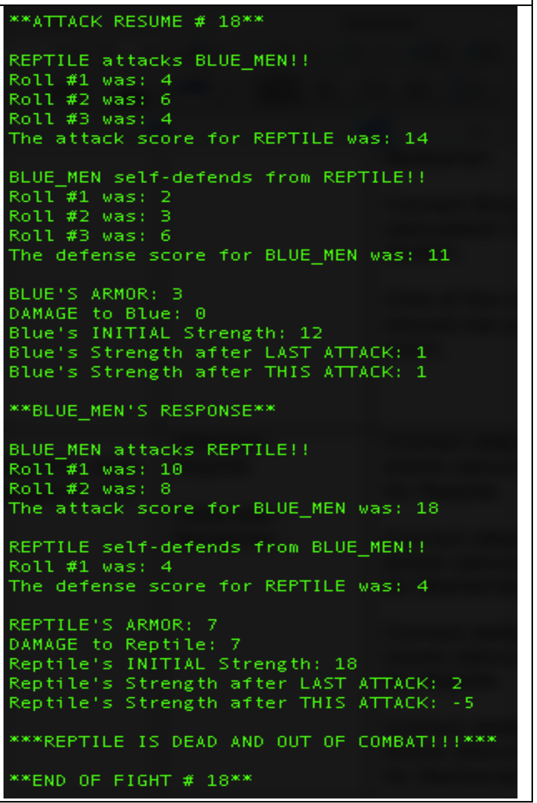
   ```
   cout<<"Enter 1 for NEXT COMBAT. ";
   cout<<"Enter 2 to automatically finish this combat to DEATH!!";
   cout<<"Enter 3 to EXIT. ";
   ```

Test Plan
In order to test this program, I created a 4-column table. In the first column, I described **what I was going to test**. In the second column, the possible **Input values** to perform the test. In the third column, the **expected outcomes**. And finally, in the forth column, the **actual values or output** that the program shows after running. This is the table I used. In my real testing process, I make each warrior had a combat to death with every other warrior. Since there are 5 warriors (5^2), I tested 25 different combats (including each character against itself). For logistic reasons, I wont include all of these 25 combats. I'll just show 1 random combat per character (but I invite the evaluator to play the game a little bit).

It is also relevant to say that the strongest fighters were Blue Men and Reptile respectively, followed by Shadow and Barbarian. And the weakest warrior (without Achilles power) was Goblin.

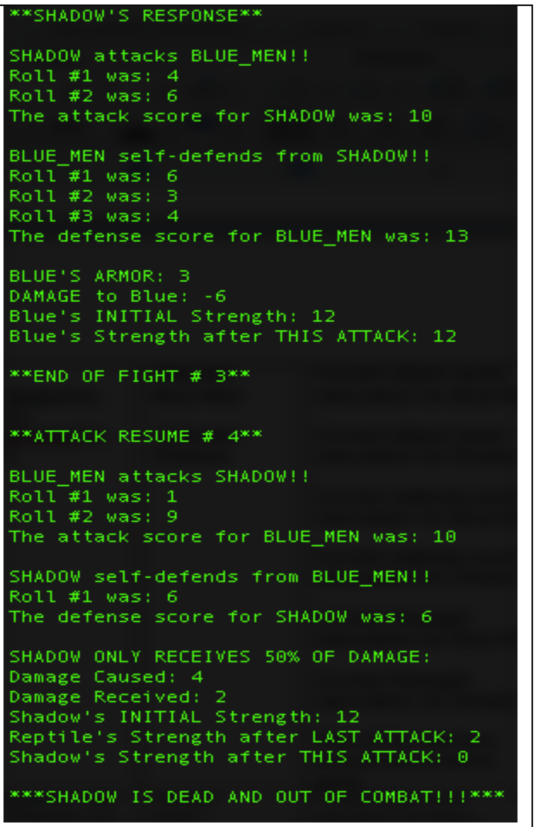
Section to be tested	Input provided	Expected Output	Actual Output

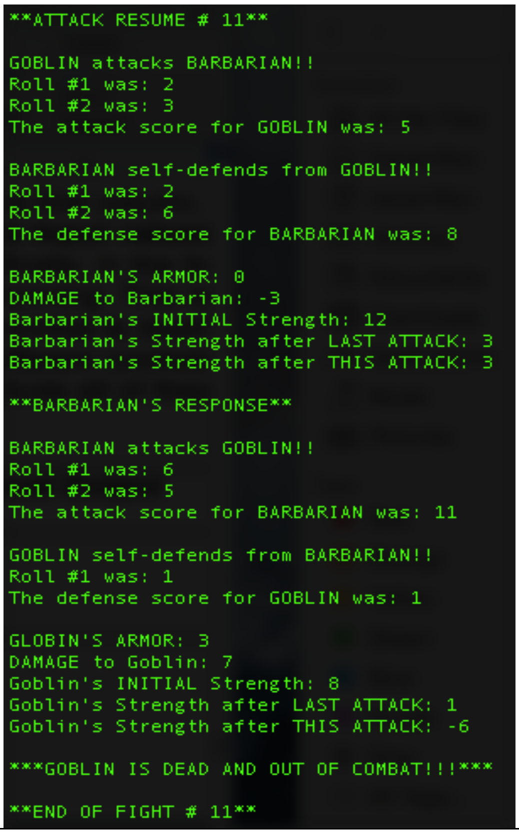
attacks(dob, dob)
diePlayer(int, int)
	Attacker:
	Barbarian
	
	Defender:
	Goblin
		Correct attack score calculation for Barbarian.
		
		Correct attack score calculation for Goblin.
		
		Correct defense score calculation for Barbarian.
		
		Correct defense score calculation for Goblin.
		
		Correct Strength calculation for Barbarian.
		
		Correct Strength calculation for Goblin.
		
		One of the warriors should die at some point.
			

attacks(dob, dob)
diePlayer(int, int)
	Attacker:
	Reptile
	
	Defender:
	Blue Men
		Correct attack score calculation for Reptile.
		
		Correct attack score calculation for Blue Men.
		
		Correct defense score calculation for Reptile.
		
		Correct defense score calculation for Blue Men.
		
		Correct Strength calculation for Reptile.
		
		Correct Strength calculation for Blue Men.
		
		One of the warriors should die at some point.
			

```
**GOBLIN'S RESPONSE**

GOBLIN attacks BARBARIAN!!
Roll #1 was: 5
Roll #2 was: 2
The attack score for GOBLIN was: 7

BARBARIAN self-defends from GOBLIN!!
Roll #1 was: 5
Roll #2 was: 1
The defense score for BARBARIAN was: 6

BARBARIAN'S ARMOR: 0
DAMAGE to Barbarian: 1
Barbarian's INITIAL Strength: 12
Barbarian's Strength after LAST ATTACK: 7
Barbarian's Strength after THIS ATTACK: 6

**END OF FIGHT # 9**


**ATTACK RESUME # 10**

BARBARIAN attacks GOBLIN!!
Roll #1 was: 5
Roll #2 was: 3
The attack score for BARBARIAN was: 8

GOBLIN self-defends from BARBARIAN!!
Roll #1 was: 3
The defense score for GOBLIN was: 3

GLOBIN'S ARMOR: 3
DAMAGE to Goblin: 2
Goblin's INITIAL Strength: 8
Goblin's Strength after LAST ATTACK: 1
Goblin's Strength after THIS ATTACK: -1

***GOBLIN IS DEAD AND OUT OF COMBAT!!!***
```

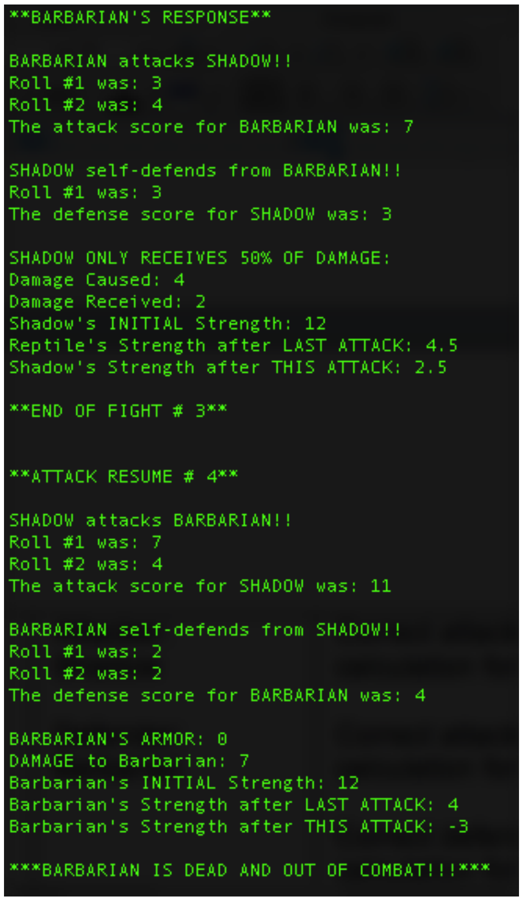
```
**ATTACK RESUME # 18**

REPTILE attacks BLUE_MEN!!
Roll #1 was: 4
Roll #2 was: 6
Roll #3 was: 4
The attack score for REPTILE was: 14

BLUE_MEN self-defends from REPTILE!!
Roll #1 was: 2
Roll #2 was: 3
Roll #3 was: 6
The defense score for BLUE_MEN was: 11

BLUE'S ARMOR: 3
DAMAGE to Blue: 0
Blue's INITIAL Strength: 12
Blue's Strength after LAST ATTACK: 1
Blue's Strength after THIS ATTACK: 1

**BLUE_MEN'S RESPONSE**

BLUE_MEN attacks REPTILE!!
Roll #1 was: 10
Roll #2 was: 8
The attack score for BLUE_MEN was: 18

REPTILE self-defends from BLUE_MEN!!
Roll #1 was: 4
The defense score for REPTILE was: 4

REPTILE'S ARMOR: 7
DAMAGE to Reptile: 7
Reptile's INITIAL Strength: 18
Reptile's Strength after LAST ATTACK: 2
Reptile's Strength after THIS ATTACK: -5

***REPTILE IS DEAD AND OUT OF COMBAT!!!***

**END OF FIGHT # 18**
```

attacks(dob, dob)
diePlayer(int, int)
	Attacker:
	Blue Men
	
	Defender:
	Shadow
		Correct attack score calculation for Blue Men.
		
		Correct attack score calculation for Shadow.
		
		Correct defense score calculation for Blue Men.
		
		Correct defense score calculation for Shadow.
		
		Correct Strength calculation for Blue Men.
		
		Correct Strength calculation for Shadow.
		
		One of the warriors should die at some point.
			

attacks(dob, dob)
diePlayer(int, int)
	Attacker:
	Goblin
	
	Defender:
	Barbarian
		Correct attack score calculation for Goblin.
		
		Correct attack score calculation for Barbarian.
		
		Correct defense score calculation for Goblin.
		
		Correct defense score calculation for Barbarian.
		
		Correct Strength calculation for Goblin.
		
		Correct Strength calculation for Barbarian.
		
		One of the warriors should die at some point.
			

```
**SHADOW'S RESPONSE**

SHADOW attacks BLUE_MEN!!
Roll #1 was: 4
Roll #2 was: 6
The attack score for SHADOW was: 10

BLUE_MEN self-defends from SHADOW!!
Roll #1 was: 6
Roll #2 was: 3
Roll #3 was: 4
The defense score for BLUE_MEN was: 13

BLUE'S ARMOR: 3
DAMAGE to Blue: -6
Blue's INITIAL Strength: 12
Blue's Strength after THIS ATTACK: 12

**END OF FIGHT # 3**


**ATTACK RESUME # 4**

BLUE_MEN attacks SHADOW!!
Roll #1 was: 1
Roll #2 was: 9
The attack score for BLUE_MEN was: 10

SHADOW self-defends from BLUE_MEN!!
Roll #1 was: 6
The defense score for SHADOW was: 6

SHADOW ONLY RECEIVES 50% OF DAMAGE:
Damage Caused: 4
Damage Received: 2
Shadow's INITIAL Strength: 12
Reptile's Strength after LAST ATTACK: 2
Shadow's Strength after THIS ATTACK: 0

***SHADOW IS DEAD AND OUT OF COMBAT!!!***
```

```
**ATTACK RESUME # 11**

GOBLIN attacks BARBARIAN!!
Roll #1 was: 2
Roll #2 was: 3
The attack score for GOBLIN was: 5

BARBARIAN self-defends from GOBLIN!!
Roll #1 was: 2
Roll #2 was: 6
The defense score for BARBARIAN was: 8

BARBARIAN'S ARMOR: 0
DAMAGE to Barbarian: -3
Barbarian's INITIAL Strength: 12
Barbarian's Strength after LAST ATTACK: 3
Barbarian's Strength after THIS ATTACK: 3

**BARBARIAN'S RESPONSE**

BARBARIAN attacks GOBLIN!!
Roll #1 was: 6
Roll #2 was: 5
The attack score for BARBARIAN was: 11

GOBLIN self-defends from BARBARIAN!!
Roll #1 was: 1
The defense score for GOBLIN was: 1

GLOBIN'S ARMOR: 3
DAMAGE to Goblin: 7
Goblin's INITIAL Strength: 8
Goblin's Strength after LAST ATTACK: 1
Goblin's Strength after THIS ATTACK: -6

***GOBLIN IS DEAD AND OUT OF COMBAT!!!***

**END OF FIGHT # 11**
```

attacks(dob, dob)
diePlayer(int, int)
	Attacker:
	Shadow
	
	Defender:
	Barbarian
		Correct attack score calculation for Shadow.
		
		Correct attack score calculation for Barbarian.
		
		Correct defense score calculation for Shadow.
		
		Correct defense score calculation for Barbarian.
		
		Correct Strength calculation for Shadow.
		
		Correct Strength calculation for Barbarian.
		
		One of the warriors should die at some point.
			

```
**BARBARIAN'S RESPONSE**

BARBARIAN attacks SHADOW!!
Roll #1 was: 3
Roll #2 was: 4
The attack score for BARBARIAN was: 7

SHADOW self-defends from BARBARIAN!!
Roll #1 was: 3
The defense score for SHADOW was: 3

SHADOW ONLY RECEIVES 50% OF DAMAGE:
Damage Caused: 4
Damage Received: 2
Shadow's INITIAL Strength: 12
Reptile's Strength after LAST ATTACK: 4.5
Shadow's Strength after THIS ATTACK: 2.5

**END OF FIGHT # 3**


**ATTACK RESUME # 4**

SHADOW attacks BARBARIAN!!
Roll #1 was: 7
Roll #2 was: 4
The attack score for SHADOW was: 11

BARBARIAN self-defends from SHADOW!!
Roll #1 was: 2
Roll #2 was: 2
The defense score for BARBARIAN was: 4

BARBARIAN'S ARMOR: 0
DAMAGE to Barbarian: 7
Barbarian's INITIAL Strength: 12
Barbarian's Strength after LAST ATTACK: 4
Barbarian's Strength after THIS ATTACK: -3

***BARBARIAN IS DEAD AND OUT OF COMBAT!!!***
```

Problems while designing and implementing your program
In general, the main problem I faced was to properly set the driver program to create character objects. Since there was several scenarios (25 in total) with different parameters, I inserted a lot bugs during this process. I spent a lot of time trying to set a combat so the program could actually be playable and graphically appreciated to see status of the warriors after each round accurately. This is actually the first time in my role as a programmer that I had to face a very tedious and complex debugging process. I may say that attention to little details was the real problem.

On the other had, I struggle a little bit deciding what the functions really should do. I was not sure if the attack class should also handle the die roll. Also, I wasn't sure if I should control all the data through the constructor or if I should use a function to make things less messy. I finally decide to use both and I don't regret to do it. Personally, I believe it was a good I idea.

Finally, I also spent a good chunk of time thinking about the initial design. I had to do a lot of reading and research. This was also my first time working with games like this. Although it was fun, it also was frustrating at some moments.

Conclusion:

In conclusion, this assignment was a great exercise to practice software design based on an OOD approach. Also, this program finally made me understand the concept of heritage, which has a lot of benefits but also certain level of complexity. In theory, if I follow these same steps when working on this type of projects, the design can be more accurate. Also, this reinforced on me the good habit of thinking about design before starting to code. In general, this was a great learning experience.