Rowan Simmons
Project 3 Design/Reflection

<u>Character Information</u>

<u>Vampire:</u>
Attack: 1 dice, 12 sides
Defense: 1 dice, 6 sides. *can charm opponent into not attacking, so 50% chance their opponent doesn't attack them.
Armor: 1
Strength Points: 18

<u>Barbarian:</u>
Attack: 2 dice, 6 sides (d1+d2 = attack)
Defense: 2 dice, 6 sides (d1+d2 = attack)
Armor: 0
Strength Points: 12

<u>Blue Men:</u>
Attack: 2 dice, 10 sides (d1+d2 = attack)
Defense: 3 dice, 6 sides (d1+d2+d3 = attack)
Armor: 3
Strength Points: 12 *for ever four points of damage, they lose one defense die

<u>Medusa:</u>
Attack: 2 dice, 6 sides (d1+d2 = attack) * if medusa rolls a 12, target gets turned to stone and medusa wins. If medusa rolls a 12 on harry potter on his first life, he comes back to life
Defense: 1 dice, 6 sides
Armor: 3
Strength Points: 8

<u>Harry Potter:</u>
Attack: 2 dice, 6 sides (d1+d2 = attack)
Defense: 2 dice, 6 sides (d1+d2 = attack)
Armor: 0
Strength Points: 10/20 * If his strength reaches 0 or below, he recovers and his strength becomes 20. if he dies again, he is dead.

Notes:
1) if medusa uses “glare” on harry potter on his first life, then harry potter comes back to life after using “hogwarts”
2) if the vampire’s “charm” ability activates when Medusa uses “glare”, the Vampire’s charm trumps Medusa’s glare
3) The characters are unbalanced intentionally, which helps with debugging.

Game directions

1) between two characters, ends when one dies
2) each round consists of two attacks, one for each character. for each attack, attacker and defender both generate dice rolls.
3) actual damage calculation: damage = attacker's roll - defenders roll - defender's armor. damage should be no less than zero- if it is, set it back to 0.
4) the value of that damage is subtracted from the defender's strength points.

Example: character A rolls 8, 10, which is an attack of 18. Character B rolls 5, 6, which is a defense of 11. Character b has an armor of 3. so damage inflicted from A to B = 18-11-3=4

If character B has a strength point of 3 during that round, the new strength point would be 3-4 = -1, which means character B dies.

5) decided who starts attacking who is own decision.

Classes

Main:
starts program

Menu:
Displays 5 characters by name, and prompts user to select two characters to fight each other. characters of the same type can fight.
Displays:
1) attacker type
2) Defender type, armor, and strength point
3) the attacker's attack dice role
4) The defender's defense dice roll
5) total inflected damage calculation
6) defender's updated strength point amount after subtracting damage

After fight is over, ask user to either play again or exit game.

Character:

Variables:
Number of attack dice
Attack dice sides number
Number of defense dice
Defense dice sides number
Armor points
Strength points
Name

Functions:
setters/getters
rollAttack
rollDefense
reduceStrength

<u>Vampire/Barbarian/BlueMen/Medusa/HarryPotter:</u>

Functions:
rollAttack (overrides)
rollDefense(overrides)
reduceStrength (overrides)

Game play

main->Menu
Menu: menu->validate->start->main

Class Hierarchy:
Parent class: Character
Child classes: Vampire, Barbarian, BlueMen, Medusa, HarryPotter

Character: holds virtual roll attack, roll defense, and reduce strength functions
*if any child class holds a virtual rollAttack, rollDefense, or reduceStrength function, it overrides the one in Character
*classes with functions that will override Character:
-All: reduceStrength(), rollAttack(), rollDefense()

Test Table

Test Case	Input	Driver Functions	Expected Outcomes	Observed Outcomes

Player1: vampire
Player2: barbarian
	1
	2
		if () if else()
		while() loop
			-loops until player strength points <=0
			-vampire charm activates ~50% of the time
			-outputs player type, strength, armor, rolls, who is attacking who, and winner
				-loops until player strength points <=0
				-vampire charm activates ~50% of the time
				-outputs player type, strength, armor, rolls, who is attacking who, and winner

Test Case	Input	Driver Functions	Expected Outcomes	Observed Outcomes

player1: BlueMen
Player2: Medusa
	3
	4
		if () if else()
		while() loop
			-loops until player strength points <=0
			-Blue Men lose 1 die per 4 strength points lost
			-Medusa uses “glare” when attack roll=12
			-outputs player type, strength, armor, rolls, who is attacking who, and winner
				-loops until player strength points <=0
				-Blue Men lose 1 die per 4 strength points lost
				-Medusa uses “glare” when attack roll=12
				-outputs player type, strength, armor, rolls, who is attacking who, and winner

player1: Medusa
player2: HarryPotter
	4
	5
		if () if else()
		while() loop
			-loops until player strength points <=0
			-Medusa uses “glare” when attack roll=12
			-Harry Potter’s “Hogwarts” trumps “glare” only once
			-outputs player type, strength, armor, rolls, who is attacking who, and winner
				-loops until player strength points <=0
				-Medusa uses “glare” when attack roll=12
				-Harry Potter’s “Hogwarts” trumps “glare” only once
				-outputs player type, strength, armor, rolls, who is attacking who, and winner

player1: Vampire
player2: Medusa
	1
	4
		if () if else()
		while() loop
			-loops until player strength points <=0
			-Medusa uses “glare” when attack roll=12
			-vampire charm activates ~50% of the time
			-vampire charm trumps glare
			-outputs player type, strength, armor, rolls, who is attacking who, and winner
				-loops until player strength points <=0
				-Medusa uses “glare” when attack roll=12
				-vampire charm activates ~50% of the time
				-vampire charm trumps glare
				-outputs player type, strength, armor, rolls, who is attacking who, and winner

Test Case	Input	Driver Functions	Expected Outcomes	Observed Outcomes

player1: BlueMen
player2: BlueMen
	3
	3
		if () if else()
		while() loop
			-loops until player strength points <=0
			-Blue Men lose 1 die per 4 strength points lost
			-outputs player type, strength, armor, rolls, who is attacking who, and winner
			-two of the same player type are able to play each other
				-loops until player strength points <=0
				-Blue Men lose 1 die per 4 strength points lost
				-outputs player type, strength, armor, rolls, who is attacking who, and winner
				-two of the same player type are able to play each other

player1: vampire
player2: vampire
	1
	1
		if () if else()
		while() loop
			-loops until player strength points <=0
			-vampire charm activates ~50% of the time
			-outputs player type, strength, armor, rolls, who is attacking who, and winner
			-two of the same player type are able to play each other
				-loops until player strength points <=0
				-vampire charm activates ~50% of the time
				-outputs player type, strength, armor, rolls, who is attacking who, and winner
				-two of the same player type are able to play each other

player1: medusa
player2: medusa
	4
	4
		if () if else()
		while() loop
			-loops until player strength points <=0
			-Medusa uses “glare” when attack roll=12
			-outputs player type, strength, armor, rolls, who is attacking who, and winner
			-two of same type are able to play each other
				-loops until player strength points <=0
				-Medusa uses “glare” when attack roll=12
				-outputs player type, strength, armor, rolls, who is attacking who, and winner
				-two of same type are able to play each other

Test Case	Input	Driver Functions	Expected Outcomes	Observed Outcomes

player1: BlueMen
player2: HarryPotter
	3
	5
		if () if else()
		while() loop
			-loops until player strength points <=0
			-Blue Men lose 1 die per 4 strength points lost
			-Harry Potter's "Hogwarts" used only once
			-outputs player type, strength, armor, rolls, who is attacking who, and winner
				-loops until player strength points <=0
				-Blue Men lose 1 die per 4 strength points lost
				-Harry Potter's "Hogwarts" used only once
				-outputs player type, strength, armor, rolls, who is attacking who, and winner

Play or exit game:

Test Case	Input	Driver Functions	Expected outcomes	Observed outcomes
Input &*	input=&*	Validate() while loop	invalid, loops back and gets user input	Invalid, loops back and gets user input
Input = 1.5	input = 1.5	Validate() while loop	invalid, loops back and gets user input	Invalid, loops back and gets user input
input = 1	input = 1	Validate() while loop	Valid, goes to menu	valid, goes to menu
input = 2	input = 2	Validate() while loop	Valid, game ends	valid, game ends

Character selection input:

Test Case	Input	Driver Functions	Expected outcomes	Observed outcomes
Input &*	input=&*	Validate() while loop	invalid, loops back and gets user input	Invalid, loops back and gets user input
Input = 1.5	input = 1.5	Validate() while loop	invalid, loops back and gets user input	Invalid, loops back and gets user input
input = 1	input = 1	Validate() while loop	Valid, proceed to next question. Registers as Vampire.	valid, proceed to next question. Registers as Vampire.

Test Case	Input	Driver Functions	Expected outcomes	Observed outcomes
input = 2	input = 2	Validate() while loop	Valid, proceed to next question. Registers as Barbarian.	valid, proceed to next question. Registers as Barbarian.
input = 3	input = 3	Validate() while loop	Valid, proceed to next question. Registers as Blue Men.	valid, proceed to next question. Registers as Blue Men.
Input = 4	input = 4	Validate() while loop	Valid, proceed to next question. Registers as Medusa.	valid, proceed to next question. Registers as Medusa.
input = 5	input=5	Validate() while loop	Valid, proceed to next question. Registers as Harry Potter.	valid, proceed to next question. Registers as Harry Potter.

Reflection:

I mostly stuck with my original design. I originally planned to override each rollAttack and rollDefense function in each individual character class, but decided that was redundant and changed my plan to handle that in the Character class, except for Medusa. Medusa's "glare" ability had to be handled by the rollAttack() function, so I added an override function in the Medusa class. All other abilities I was able to accommodate using the reduceStrength() function that was an override for all classes. For Medusa's rollAttack() function, I had some trouble getting glare to work without returning a value, so I ended up just returning a large value that would always kill the opponent. The Blue Men gave me the most trouble. I decided to handle their ability in the reduceStrength() function, and separated the dice elimination into two categories: if their strength points were between 4 and 8, or less than 4. Between 4 and 8 they lost 1 die, and less than 4 they lost 2 dice. I found that to be a better solution than going by damage, because if they lost 8 points of damage during one round, they had to lose 2 dice instead of 1, so an (if damage >=4, subtract one die) wouldn't work properly. Additionally, I had a hard time figuring out how to have Harry Potter's "Hogwarts" override Medusa's "glare" only once, and also to only update his strength to 20 once while playing any opponent. I ended up making a private variable in the harry potter class to track how many times he used it, to make sure it was only activated once.

I found it clunky to put the actual game logic in the menu function, but the directions explicitly said to do that so I didn't want to do anything else. I would have preferred to separate the menu into two classes, menu and game, and handle the game play in a game class.