Name: Christopher Matian
OID: 933308644
Date: 11/03/2017

Initial Design Strategy and File Structure

My initial design strategy revolved around breaking up the program into sections and building it piece by piece. I used the grading rubric as a list of steps to take as I built out each character class. I also wanted to try out some new things with the program's various menus. For the menus I want them to be dynamic in that I can just push a string to a vector, parse through the vector, and use a showMenu() function to display each value that's parsed.

Menus would use a format like so:

1. Declare a vector<string> mainMenu.
2. Use mainMenu.push_back("some string") to push strings (the menu option) into them.
3. Use a showMenu(vectorName) function that displays the vector that's passed in as an argument.

Creating the menus like this would allow me to have a single showMenu function that I can call which takes in any vector of strings as an argument. I don't anticipate to use anything else but strings in vectors so there's no need to worry about using templates.

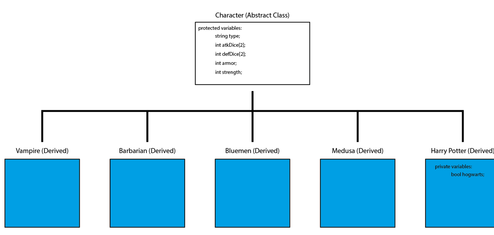
For file structure I plan to break down the project into several files for menus, validation, main, creatures, and combat. A more detailed look would be something like this:

Header File	Function File
	main.cpp
validation.hpp	validation.cpp
menu.hpp	menu.cpp
combat.hpp	combat.cpp
character.hpp	character.cpp
vampire.hpp	vampire.cpp
barbarian.hpp	barbarian.cpp
bluemen.hpp	bluemen.cpp
medusa.hpp	medusa.cpp
harrypotter.hpp	harrypotter.cpp

The game flow should work like so:

1. Main.cpp is declares the menu objects and queries the player for general information (what characters to fight with - two selections).
2. A Combat object is constructed and we pass in characters that the player wanted to simulate combat between.
3. The combat simulator will take the passed in characters and simulate combat between them using a recursive function. Each character should fight until death (i.e. a winner is declared).
4. Delete the characters just before querying the player as to whether they would like to play again or not (this can help reset the results). This step might not even be necessary since the object is initialized at the beginning, so we're probably going to overwrite and initialize the information each game loop.
5. If the player wishes to continue we allow the do-while loop to continue running, otherwise we exit the loop and the program exits.

Simple Class Hierarchy Diagram



Design Changes, Problems, and Final Strategy

The design changes that I made were quite varied. First, I changed my **validateBounds()** function to handle user input as well rather than just return a boolean. I made it so that it returned an *integer* based on the *min* and *max* values, and then I used that int to determine which *menu option* the user wanted to select.

Within my combat hpp/cpp file, I opted to not use a class because I simply didn't see the need for creating *another* class for handling the combat simulation. Instead, I could just use it as storage spot for the prototype and **combat()** function. This helped clear out the Main.cpp file so it wasn't too cluttered by the prototype initialization and the function declaration at the bottom.

For my Character class, I was originally going to just override each and every attack and defend function because I had set them as pure virtual. However, I realized that this would be rather inefficient because I would end up repeating a lot of code. So, I made a base virtual function that each derived character class would have access to. In the event they needed to override that class, I would simply create a virtual function in the derived class file and have that override the original, base-class file. Additionally, I refactored my derived class constructors to simply be empty and set / inherit the base-class' protected variables. This cleaned up my code quite a bit since I didn't need to worry about setting these variables at run-time (i.e. when the user selects them and a new object is created), so I could just set them as the default constructor variables. My code was originally passing in the variable information through the new object call within the main file.

Additionally, I used a similar strategy that I had used in my project 2 where I created two arrays (atkDice and defDice) both set to hold 2 values. The first index would hold the total number of dice being rolled, and the second index would hold the total sides each dice had (the max bounds). I then created two associated roll functions for both attack and defend, and used the values of these dice to create a random number for the corresponding attack and defend functions. By doing this, I didn't need to worry about creating a bunch of different attack/defend virtual functions for each derived class and set the RNG roll for each one. By doing things this way, I could easily control who rolled what and how many of dice dice, all by calling two simple functions that referenced their stored dice arrays.

A big change I also made was abstracting medusa's glare ability as doing a ton of damage. Originally, my attack function was returning a boolean value based on whether or not the attack killed the defender. The problem with this was that I had no easy way of utilizing the recursive function I had in mind because it would immediately exit the function and return that Medusa had won. In theory this is OK but not when she's fighting a Vampire or Harry Potter, both of which have special abilities that override her glare ability. If I had returned false through the attack function into the recursive function, she would have ignored the special abilities of both characters which is incorrect behavior.

So, I instead made the virtual defend function a boolean, and determined the result of each combat **after** we had determined whether or not the defender had been killed, rather than whether or not the attacker had killed the defender. It might sound like the same thing but the order of operation just changed. Doing things this way essentially allowed me further control over who died and how I displayed that information, and abstracting Medusa's ability as doing raw damage allowed me to simulate her glare ability. When she hits the roll (12) for glare, she kills anyone that isn't a Vampire using charm, or Harry Potter with his Hogwarts ability.

Test Cases:

	Input Values	Driver Functions	Expected Outcome	Observed Outcome
or	-10	validateBounds(intent);	Should catch the bad submission and query the user again.	`Please enter a value that corresponds with the menu item above.`
or	e	validateBounds(intent);	Should catch the bad submission and query the user again.	`Please enter a value that corresponds with the menu item above.`
or	eeee	validateBounds(intent);	Should catch the bad submission and query the user again.	`Please enter a value that corresponds with the menu item above. (x4)`

	NA	combat() function;	Even match up.	`Character 2 (Vampire) is the victor!`
				
				`End of Simulation Health Statistics.`
				`Character 2 (Vampire)  health: 15`
				`Character 1 (Vampire)  health: 0`

	NA	combat() function;	Even match up.	`Character 2 (Barbarian) is the victor!`
				
				`End of Simulation Health Statistics.`
				`Character 2 (Barbarian)  health: 9`
				`Character 1 (Barbarian)  health: 0`

	NA	combat() function;	Even match up.	`Character 1 (Blue Men) is the victor!`
				
				`End of Simulation Health Statistics.`
				`Character 1 (Blue Men)  health: 11`
				`Character 2 (Blue Men)  health: 0`

	NA	combat() function;	Even match up.	`Character 1 (Medusa) is the victor!`
				
				`End of Simulation Health Statistics.`
				`Character 1 (Medusa)  health: 8`
				`Character 2 (Medusa)  health: 0`

	NA	combat() function;	Even match up.	`Character 1 (Harry Potter) is the victor!`
				
				`End of Simulation Health Statistics.`
				`Character 1 (Harry Potter)  health: 3`
				`Character 2 (Harry Potter)  health: 0`

	NA	combat() function	Vampire CHARM should override and ignore GLARE.	`Character 2 (Medusa)  attacks:`
				
				`The attacking Medusa has 5 strength and 3 armor.`
				
				`The attacking Medusa made an attack roll of 6`
				`The attacking Medusa made an attack roll of 6`
				`The attacking Medusa dealt 12 damage total.`
				
				`Medusa rolled a total of 12 with both dice and successfully used her ~~GLARE~~ ability. Avert your gaze - quickly!`
				`The defending Vampire has 18 strength.`
				
				`The defending Vampire used ~~CHARM~~ and completely avoided taking damage.`
				
				`The defending Vampire has 18 strength remaining.`

t	NA	combat() function	Harry Potter HOGWARTS should override and ignore first instance of death from GLARE.	`Character 1 (Medusa)  attacks:`
				
				`The attacking Medusa has 5 strength and 3 armor.`
				
				`The attacking Medusa made an attack roll of 6`
				`The attacking Medusa made an attack roll of 6`
				`The attacking Medusa dealt 12 damage total.`
				
				`Medusa rolled a total of 12 with both dice and successfully used her ~~GLARE~~ ability. Avert your gaze - quickly!`
				`The defending Harry Potter has 7 strength and 0 armor.`
				
				`The defending Harry Potter made a defense roll of 5`
				`The defending Harry Potter made a defense roll of 6`
				`The defending Harry Potter's armor reduced the attack by 0`
				`The defending Harry Potter reduced the attack to 39 damage.`
				`The defending Harry Potter has 0 strength remaining.`
				`The defending Harry Potter has been slain in glorious combat!`
				`The defending Harry Potter was slain but managed to use ~~HOGWARTS~~. He came back to life with 20 hitpoints - wicked!`

	NA	combat() function	Each time a Bluemen loses 4 health the defense dice is reduced by 1 (up to a maximum of two times).	`Character 2 (Blue Men)  attacks:`
				
				`The attacking Blue Men has 12 strength and 3 armor.`
				
				`The attacking Blue Men made an attack roll of 10`
				`The attacking Blue Men made an attack roll of 8`
				`The attacking Blue Men dealt 18 damage total.`
				
				`The defending Blue Men has 12 strength and 3 armor.`
				
				`The defending Blue Men made a defense roll of 1`
				`The defending Blue Men made a defense roll of 1`
				`The defending Blue Men made a defense roll of 2`
				`The defending Blue Men's armor reduced the attack by 3`
				`The defending Blue Men reduced the attack to 11 damage.`
				`The defending Blue Men has 1 strength remaining.`
				`Due to the ~~MOD~~ special ability, the Bluemen have lost a blue man and their defense has been reduced!`

th	NA	combat() function	Same as above.	`Character 2 (Blue Men)  attacks:`
				
				`The attacking Blue Men has 6 strength and 3 armor.`
				
				`The attacking Blue Men made an attack roll of 8`
				`The attacking Blue Men made an attack roll of 10`
				`The attacking Blue Men dealt 18 damage total.`
				
				`The defending Blue Men has 12 strength and 3 armor.`
				
				`The defending Blue Men made a defense roll of 2`
				`The defending Blue Men made a defense roll of 2`
				`The defending Blue Men made a defense roll of 3`
				`The defending Blue Men's armor reduced the attack by 3`
				`The defending Blue Men reduced the attack to 8 damage.`
				`The defending Blue Men has 4 strength remaining.`
				`Due to the ~~MOD~~ special ability, the Bluemen have lost a blue man and their defense has been reduced to 2D6!`
				`Due to the ~~MOD~~ special ability, the Bluemen have lost a blue man and their defense has been reduced to 1D6!`