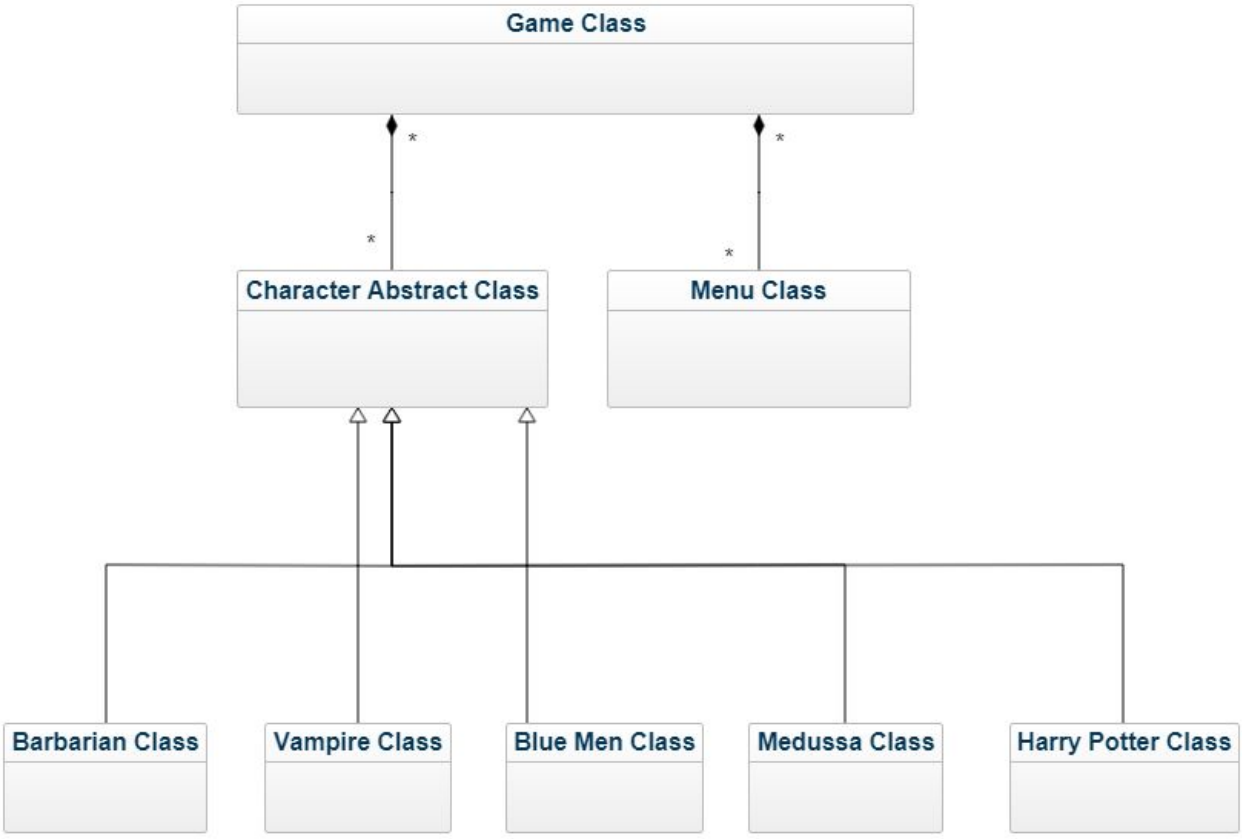
DESIGN & REFLECTION DOCUMENT

SANDRO AGUILAR
PROJECT 3 - SECTION 400

February 15, 2019

DESIGN

I began my project by starting with an overall design using an UML diagram so that I could begin thinking about how to plan the overall design of my program.



My initial design shows that the Character class is an abstract class and the actual specific characters (Vampire, Barbarian, Blue men, Medusa, Harry Potter) all inherit from Character. The Character base class will be an object inside of the Game class. The menu will also be created inside of the Game class

I decided to Finally, since I will be dynamically allocating animals to various arrays, I have to make sure to deallocate them in the proper location which in this case, would be the Zoo class.

TEST TABLE

This test plan is provided to show how I tested the program. My goal was to create a test plan that is robust enough to provide assurance that the program behaves like it is meant to. The bulk of the test plan makes sure that the game parameters are properly set as this is what drives the rest of the program. Testing for memory leaks was also a part of my tests.

Test Case	**Input Values**	**Expected Output**	**Actual Output**
Main Menu (start game and exit)	Input < 1 Input > 2 Input = any characters, spaces, and number combination I.e., 1 1 1, i.1 10000, -100000, 11.1, 11. 12	Any input that is not 1 or 2 should be rejected. Selecting 1 starts the game and selecting 2 quits the game.	Only characters 1 or 2 are accepted, all others rejected. Selecting 1 starts the game and selecting 2 quits the game.
Character Selection Screen	Input < 1 Input > 2 Input = any characters, spaces, and number combination I.e., 1 1 1, i.1 10000, -100000, 11.1, 11. 12	Any input that is not 1 - 5 should be rejected. The appropriate characters should be selected.	Select 1 = Vampire Select 2 = Barbarian Select 3 = Blue Men Select 4 = Medusa Select 5 = Harry Potter
Characters	N/A	A screen should	Characters selected are the

Selected		appear that confirms the characters selected for combat	ones that appear on the menu screen.
Combat Rounds	N/A	The game should track the combat rounds and the characters should attack each other in the order initially selected.	Each combat round is tracked and displayed by an integer that increments by 1 each time. Each character attacks each other in the order initially selected
Combat Operations	N/A	When player 1 attacks, player 2 should defend. When player 2 attacks, player 1 should defend. This should happen in the same round.	Player 1 attacks player 2 Player 2 defends Player 2 attacks player 1 Player 1 defends
Combat Operations - Attack Power	N/A	Each character should attack with the proper attack dice settings. Barbarian - 2d6 Vampire - 1d12 Blue Men - 2d10 Medusa - 2d6 Harry Potter - 2d6	Attack dice power does not exceed specifications required by each time of player. Barbarian - 2d6 Vampire - 1d12 Blue Men - 2d10 Medusa - 2d6 Harry Potter - 2d6
Combat Operations - Defense Power	N/A	Each character should defend with the proper defend dice settings. Barbarian - 2d6 Vampire - 1d12 Blue Men - 2d10 Medusa - 2d6 Harry Potter - 2d6	Defence dice power does not exceed specifications required by each time of player. Barbarian - 2d6 Vampire - 1d6 Blue Men - 3d6 Medusa - 1d6 Harry Potter - 2d6
Combat Operations - Attack and defense calculations	N/A	Attacker should display the calculated attack power. Defender should display calculated defense power. Defender should display proper armor, if applicable.	Each round shows the attack power of the attacker. The starting strength of the defender is displayed. Defense roll is calculated by defender and armor points are added to it to mitigate damage received. Ending strength points are updated to show the correct remaining strength after taking into account any

		Defender’s strength points should be properly updated to show any damage taken in combat.	damage received.
Special Powers - Vampire	N/A	Vampire should have a 50% chance of charming opponent into not attacking	Vampire charm ability activates 50% of the time. When Medusa uses Glare, Charm trumps Glare.
Special Powers - Blue Men	N/A	Blue men defense dice should be adjusted so that once dice is removed for every 4 points of damaged received.	Blue men has defense die of 3d6 for strength points 8-12. Blue men has defense die of 2d6 when strength points are 4-7. Blue men has defense die of 2d6 when strength points are 0-3.
Special Powers - Medusa	N/A	Medusa should use her glare special ability when she rolls a 12 and instantly defeat her opponent. However, Vampire’s Charm trumps Glare	Medusa’s glare is activated when she roles a 12. Vampire’s Charm trump’s Medusa’s glare.
Special Powers - Harry Potter	N/A	Harry Potter should starts off with 10 strength points. Should Harry Potter die, he should revive and have 20 strength points. Harry Potter can only revive once.	Harry Potter starts off with 10 strength points. When Harry Potter dies, he revives and has 20 strength points. Harry Potter revives only once.
Player Death	N/A	A player whose strength points reaches 0 first should die and no more rounds should continue. If a player is attacked first and dies, that player can no longer attack and the round should end immediately.	When a player’s strength reaches 0, the round is immediately over even if player 2 did not get to attack.
Memory Leaks	N/A	No memory leaks should be present	No memory leaks were identified on Valgrind on

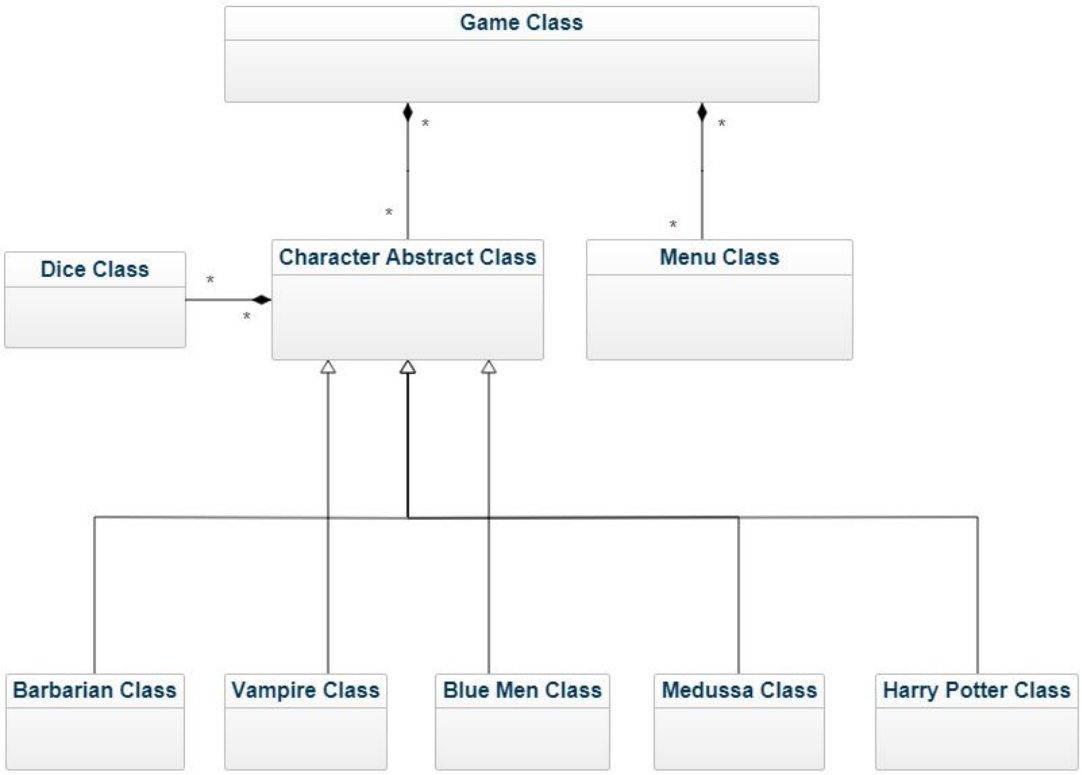
		after running multiple games with all players multiple times.	the Flip server

REFLECTION

During the implementation of the combat simulation project, I found that I had to go back many times to research topics that I have already studied such as building constructors and other random syntax for doing certain things. I think this is because many times I will use a certain syntax or concept once or twice and I don't really get to practice it on a daily basis so it is harder to retain. However, I did notice that when going back and doing my research, I was more proficient in finding the resources I needed online. I also found it a lot easier to implement the code since I was already familiar with it.

One major change that I made to my code was to add a new class that was not previously shown in my initial design. After reading the project a few times and starting the project, I realized that a dice would be necessary and therefore I decided to re-use my dice class from the prior class in this project. This definitely saved me some time however it did create new challenges since each type of character has certain dice requirements for both attack and defense. This required that I apply new concepts into my design such as virtual functions in my abstract character class. This allows me to build any type of dice for each character where the type of dice created will be determined at run time depending on the characters selected. Another challenging aspect about the dice was thinking of a way to create the dice I needed for each different character. Although it is easy to create any type of dice, the problem I was trying to solve was how to create a data structure that would hold various kinds of dice since some characters can use up to three types of the same dice. I decided that using a 2-D array was a good option because it is easy to keep track of the dice and is easy to scale as needed per player. However, this presented more challenges with memory management as I will describe further below. Here is an updated UML class diagram after revising the class structure.

UPDATED CLASS HIERARCHY DIAGRAM



A challenging aspect of this game was implementing the special attack powers of each character. This part was challenging because the character abilities are both offensive and defensive in nature. This required special considerations as to when a player attacks and defends and the order of operations depending on the player type.

Another challenging aspect of this project was managing memory. I think that the way I designed the game required extra consideration on how to manage the memory I was obtaining from the heap to instantiate the types of players being chosen as well as the memory I was obtaining for the dice in each player. The most challenging issue with memory that I had was figuring out why my player objects were not properly being freed in memory. At first I would have thought that deallocating the memory to the dice would be more complicated since they are in arrays however I did not see any issues with that. Instead, Valgrind was telling me that the players were not all being deleted. This was at first

confusing since I only have two player objects. The way I resolved this was testing to see what would happen if I only played one combat simulation and looking at the Valgrind results and then playing 2 or more combat simulations. When I played one combat simulation, there were no leaks. However, if I played 2 or more combat simulations, Valgrind would start showing memory leaks. This immediately helped me in identifying the problem: when the game ends, if a player decides to play again, a loop will create new players again. This calls the new keyword again and keeps doing so until a player quits. The way I resolved it was by moving the delete operations from the destructor and into the loop where the players could be deleted after combat. I ran Valgrind again, player many rounds with many different players, and no memory leaks were found.