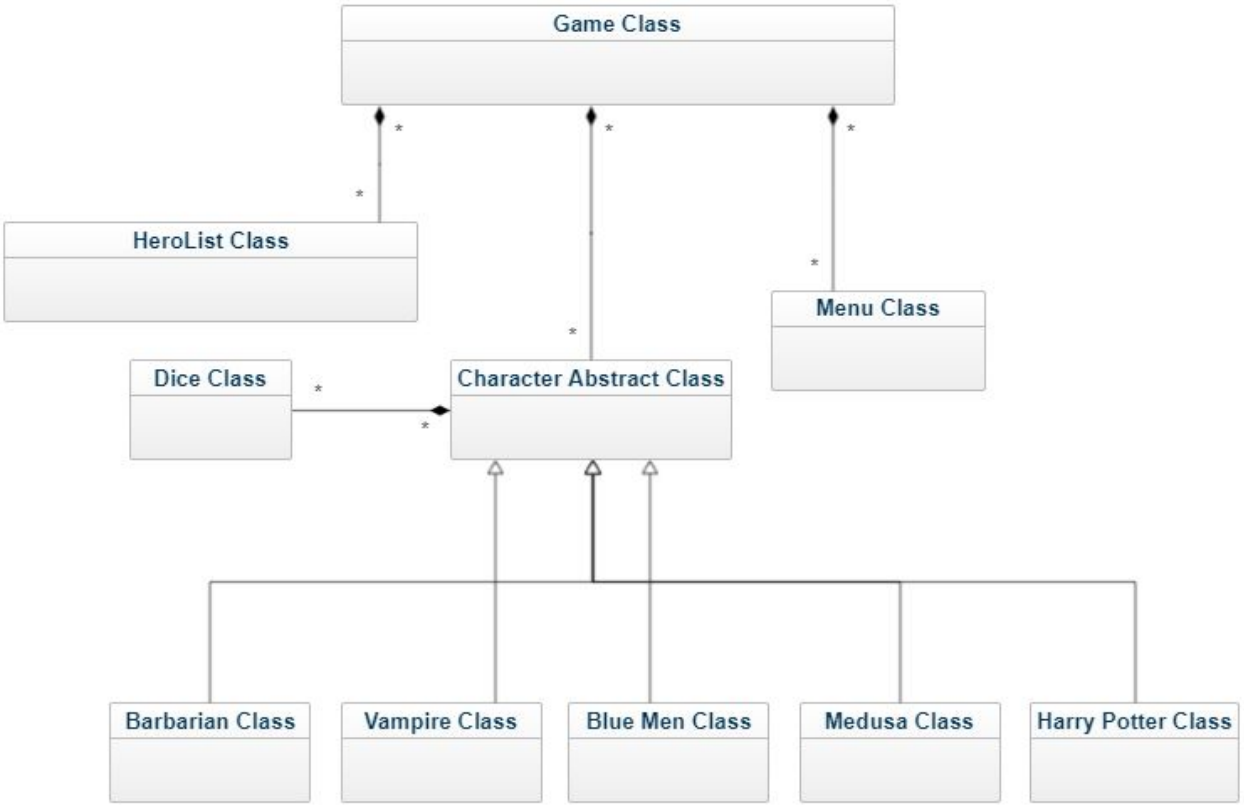
DESIGN & REFLECTION DOCUMENT

SANDRO AGUILAR
PROJECT 4- SECTION 400

March 1, 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.



I initially had trouble deciding how to implement the HeroList class. At first, my plan was to have it inside of the game class because then I would be able to have the list class communicate across all other objects within the game class. However, it was daunting thinking about having to pass objects to functions since I would have to make sure that I am interacting with the same object within the function and not a copy of it. I then thought of having the list class as inheriting from the game class however I found that although it seemed to be working fine, I found it awkward and confusing doing it this way. Therefore, for my initial design, I will stick with the HeroList class as an object within the game class.

All of my other classes stayed the same compared to project 3. The Character class is still an object inside of the Game class and it is the base class for various character types. The menu will also be created inside of the Game class.

TEST TABLE

This test plan is provided to show how I tested the program. I decided to keep the same table from project 3 however I appended an additional section to test the additional work done for this project, particularly the node list. However, I kept memory leaks since my initial tests indicated that memory was leaking in various places.

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	Input < 1 Input > 2	Any input that is not 1 - 5 should be	Select 1 = Vampire Select 2 = Barbarian

Screen	Input = any characters, spaces, and number combination I.e., 1 1 1, i.1 10000, -100000, 11.1, 11. 12	rejected. The appropriate characters should be selected.	Select 3 = Blue Men Select 4 = Medusa Select 5 = Harry Potter
Characters Selected	N/A	A screen should appear that confirms the characters selected for combat	Characters selected are the 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	N/A	Attacker should display the calculated attack power.	Each round shows the attack power of the attacker. The starting strength of

calculations		Defender should display calculated defense power. Defender should display proper armor, if applicable. Defender's strength points should be properly updated to show any damage taken in combat.	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 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	When a player's strength reaches 0, the round is immediately over even if player 2 did not get to attack.

		continue. If a player is attacked first and dies, that player can no longer attack and the round should end immediately.	
PROJECT 4 TEST TABLE			
Set total players for each team including the same type of character.	Input < 1 Input > 2 Input = any characters, spaces, and number combination I.e., 1 1 1, i.1 10000, -100000, 11.1, 11. 12	A user should be able to enter any number of characters for each team. This should include the same type of characters.	The program accepts any number of characters for each team including the same type of character multiple times.
Total matches in tournament	N/A	A tournament should match fighters from each team until there are no more fighters available to pair and fight. For example, if there are 2 fighters left in team A and none in team B, then no more battles should take place.	Tournament ends as soon as one team runs out of fighters to fight.
The score for each team should be properly computed	2+ points for winner team -1 points for loser team	Every winner should add 2 points to the team. Each loser should subtract 1 point from the team.	The score is properly computed. A team's score is allowed to go negative.
Determining the winning team.	2+ points for winner team -1 points for loser team	The team with the most points should win; the team with the least points should lose; teams with the same number of points should tie.	The winning team is properly identified. The losing team is properly identified. Teams with equal points results in a tie.
Display the loser list	Input < 1 Input > 2 Input = any characters, spaces, and number combination I.e., 1 1 1, i.1 10000, -100000, 11.1, 11. 12	If a user selects 1, the defeated Heroes should be displayed. Since the defeated heroes are held in a stack structure, the first hero to be displayed should be the last hero added to the stack.	The defeated heroes are displayed in a LIFO method consistent with a stack data structure.

Display the correct defeated heroes in the loser's list	N/A	The defeated heroes should be added to the losers list and it should display the proper losers.	The correct defeated heroes are displayed when the function to display the losers list is called.
Move the winning hero to the back of their list	N/A	The winner of each match should be moved back of their list so that the next hero can fight.	The winning hero is added to the back of the list. This can be confirmed by seeing who fights next and or by traversing the list to see the order of the characters in it.
Restore winner's health	N/A	The winner should have their health restored in a random fashion ranging from 10% to 100%. Their health should not be restored to exceed past their allowable health.	The winning hero's health is restored after the match and then moved to the back of their list. The winner's health is not restored past their maximum health allowable.
Memory Leaks	N/A	No memory leaks should be present after running multiple games with all players multiple times.	No memory leaks were identified on Valgrind on the Flip server

REFLECTION

The implementation of the HeroList class turned out to be challenging especially at the beginning. However, as I went over the the architecture of the program, I was able to better understand of how the classes should interact with one another.

The difficult part was understanding how to pass information back and forth between the game class and the HeroList class. I learned in the group project that whenever we are passing object around in functions, we need to pass them by reference if we need to interact with the same object and not a copy of the object otherwise bugs will occur big time. Once I figured out how to do this, I was able to implement the HeroList fairly easily after that.

I implemented queue lists for each player team since they are easy to understand and make sense to implement for this use case. A stack was used to hold the defeated heroes since the specifications require that the defeated heroes be placed on top of the list and it is fairly easy to add them to a stack. Also, displaying them is fairly simple when you want to display them in a LIFO method.

The most difficult aspect of this project was debugging the memory leaks that I had encountered. When I completed project 3, I was able to resolve the memory leaks so I knew that I must have introduced them somewhere in this project when adding the linked lists. The approach I took to resolve them involved displaying the memory addresses of the list nodes I was creating and using Valgrind. Initially, I was able to figure out how to delete the nodes since it was similar to Lab 6 however I still had memory leaks. Then I realized that since I also had character objects within the nodes, I needed to delete those as well. I added code to delete the characters during the traversal that deletes the nodes and I proceeded to test using Valgrind and although the memory leak significantly decreased, there was still a leak remaining. I realized that just like in project 3, I was probably not deleting the old characters if a user decided to play again. The old characters ended up getting lost somewhere so then my destructor was not deleting them since by then they were probably unreachable. My solution was to delete the characters after the tournament was complete but before the program was exited. This would delete any memory being used by the prior characters and allow for all memory to be deleted after ending the program. After experimenting with this solution for a while, I was able to get it to work and my game was memory-leak free.

I really liked implementing and using the linked list data structures in this project. It really helped to reinforce how to use most of the linked lists that we studied and it gave me a chance to practice them even more.