Project 4: Fantasy Combat Tournament
Christine Bruton - Student ID: 933010860

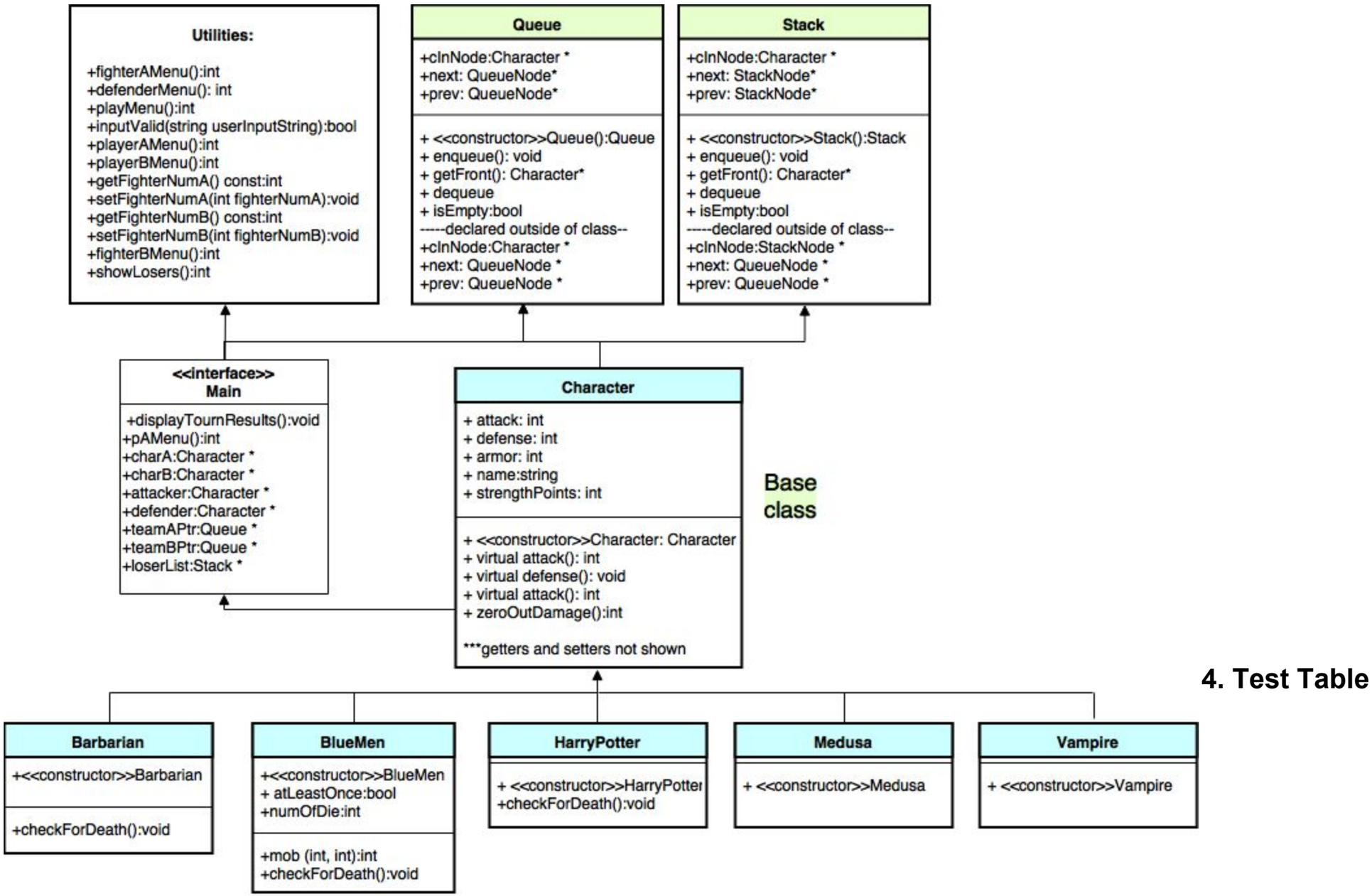
PROGRAM DESIGN DOCUMENT

1. Diagram (including hierarchy chart)

2. Test Table

3. Reflection

1. Diagram (including hierarchy chart)



4. Test Table

Input	Expected Result	Actual Result
Team A: 1 Character vs. Team B: 10 Characters	Team A loses	correct
Yes to play again menu	nodes emptied after each tournament	correct
Yes to play again menu	Score add up correctly each round	correct
Standard game	Players loaded from front of Queue	correct
Standard game	Players added to loser pile if dead	correct
Standard game	Accept any number of team members	correct

Recovery Function	Expected Result	Actual Result	Fix?
Vampire	Regain random amount of strength points between 5-7	1st: Did not regain 2nd: correct	Changed to add points instead of percentage. Cap at base strength
Medusa	Regain random amount of strength points between 8-10	1st: Regained too much 2nd: correct	Rebalanced points gained and base strength
Harry Potter	Regain random amount of strength points between 1-3	1st: correct	
Blue Men	Regain random amount of strength points between 1-2	1st: correct	
Barbarian	Regain random amount of strength points between 3-8	1st: correct	

5. Reflection

Using the last project as a basis, I commented out many of the combat messages and created displays for the round information and tournament scores. I also created a new recovery virtual function that was based on dice roll range as opposed to damage since medusa's extreme power might mess that up. Each character has a different range based on how they performed in battle during the last project (i.e. medusa has a greater recovery because she always died).

The most challenging aspect of this project was the memory management as well as successful use of the circular linked list. Once I received assistance from Khuong during the the office hours about the pointer logic, I improved my linked list functions and avoided crashes and segment faults. I also made sure they conformed to the instructions (queue-like and stack-like). However, my program still has memory leaks. After learning a bit more, I realize that I should have designed the functions to account for any dynamic memory allocation from the beginning. As a result, I could not ultimately track down all the leaks before submitting the assignment since I simply ran out of time. One problem I ran into was that the lists were not emptying out after each game. I solved this by creating them anew in each game play loop, then destroying them in the end. Overall, I will really need to take a more proactive approach to combating memory leaks since just trying to track them down at the end doesn't see to work out well for me.