Clinton Hawkes
CS162-400
Project 4 Design Document

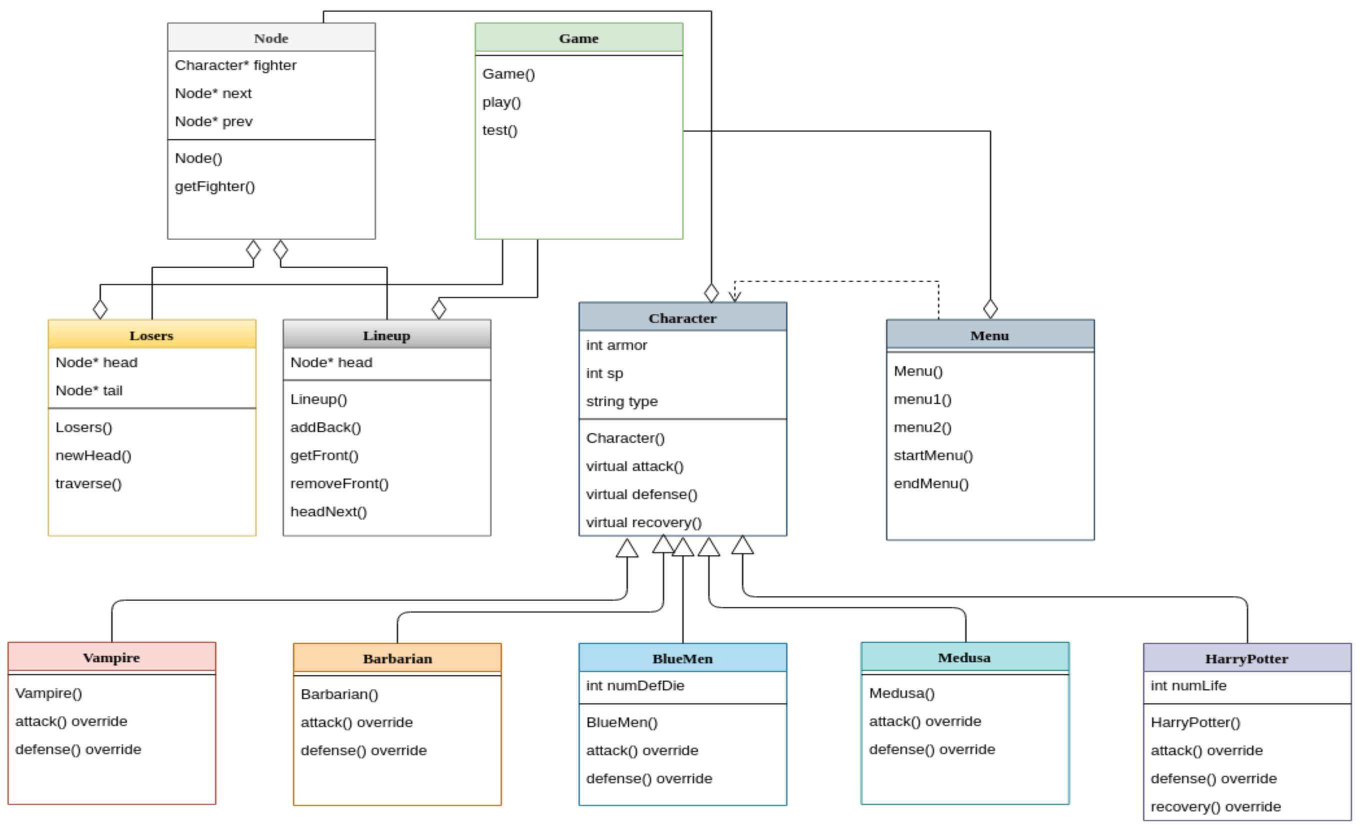
Project 4 Fantasy Combat

Design:

Program will have a main function, a Game class, a Lineup class, a Losers class, a Node class, a Character class, a Vampire class, a Barbarian class, a BlueMen class, a Medusa class, a HarryPotter class, a Menu class, a rollDie() function and an input validation function.

- 1) Main function – only used to seed the rand() function, create a Game object and call the Game::play() function to begin the tournament.
  - A) Create a bool variable "state" to hold the current state of the game (run or quit)
  - B) Create two Lineup objects(queues) that will point to team1 and team2 lineups
  - C) Create a Losers object(stack) to hold all the defeated fighters
  - D) Create a Menu object so the menus can be called
  - E) Call the startMenu() function
    - I. User chooses to play or quit
      - a) If user selects play, true is returned to the calling function
        - i. Variable "state" is set to true
      - b) If user selects quit, false is returned to the calling function
        - i. Variable "state" is set to false and program terminates
  - F) While loop created that will loop until variable "state" is set to false
    - I. User is asked how many fighters they would like on each team
      - a) User input is validated and set equal to "choice"
    - II. For loop is used to call menu1 the number of times user chose for team1 fighter count
      - a) User is asked what type of character they would like the current fighter to be
        - i. User is asked to name the current fighter
    - III. For loop is used to call menu2 the number of times user chose for team2 fighter count
      - a) User is asked what type of character they would like the current fighter to be
        - i. User is asked to name the current fighter
    - IV. While loop created that will loop until one of the teams lineup is empty
      - a) Test function from Program3 is called to simulate a battle between the first fighters of each team
        - i. While loop is created that runs until a winner is declared
          - 1) First half of round
            - A) Fighter from team1 attacks
            - B) Fighter from team2 defends
            - C) Total inflicted damage to team2 is calculated
              - I. If team2 fighter has sp <= 0, fighter from team1 wins
          - 2) Second half of round

```
            A) Fighter from team2 attacks
            B) Fighter from team1 defends
            C) Total inflicted damage to team1 is calculated
                I. If team1 fighter has sp <= 0, fighter from team2 wins
        ii. Once a winner is declared, the stats of the battle and declared winner is printed
            1) The winner recovers some of their sp points through the recovery function
        iii. The winning team is returned to the “result” variable in the play() function
    b) Switch statement base on “result” variable
        i. If team1 was the winning team
            1) Losing fighter from team2 moved to the loser stack
            2) Team2 queue is reduced by one
            3) Winning fighter from team1 moved to the back of team1 lineup
            4) Team1 gains 2 points for winning
            5) Team2 loses 1 point for losing
        ii. If team2 was the winning team
            1) Losing fighter from team1 moved to the loser stack
            2) Team1 queue is reduced by one
            3) Winning fighter from team2 moved to the back of team2 lineup
            4) Team2 gains 2 points for winning
            5) Team1 loses 1 point for losing
V. One of the team lineup is empty, so the points are tallied and compared
    a) If team1 has more points than team2
        i. Team1 is the winner
        ii. Tournament stats are displayed
    b) If team2 has more points than team1
        i. Team2 is the winner
        ii. Tournament stats are displayed
    c) If team1 and team2 have the same points
        i. It is a tie
        ii. Tournament stats are displayed
VI. User is asked if they would like a list of the losing fighters to be displayed
    a) If yes, the losing fighters are displayed from most recent loser to first loser
VII. User is asked if they would like to play again or quit
    a) If user quits, program terminates
    b) If user wants to play again
        i. All queues and stacks are erased and the program starts at the beginning
```



Most of the classes and their contents were created for previous assignments, so much of the work was already done before beginning this project. Most of my time was spent creating the Game class so the flow of the program would match the specifications. The Game class creates the containers for each team's fighters/losers, calls the menus to populate the containers with the user's choice of characters, and then issues function calls to make the fighters from each team battle. Container maintenance is then performed depending on the results of the battles.

One of the design choices I decided to implement was to use the stack class from lab 6 for the loser container and to use the queue class from lab 7 for the team lineups. I frequent the non-official slack for this course, and it appears I am in the minority with this approach. All the other students just used the queue structure for both the lineups and the loser list. They just added an extra function to make the class work for both cases. The language of the spec sheet made me favor my approach, because it strongly conveyed that a stack should be used for the losers and a queue should be used for the team lineup. Either approach works, but I wanted to stick with the code I knew was already tested and worked.

One other change that was made to the original code of these classes was to add a recovery function to the Character class. I made the recovery function a virtual function so HarryPotter could override it to fit it's unique attributes. The HarryPotter class has different levels of maximum sp, dependent upon how many lives he has left. This required the function to be able to differ in the number of sp points that were returned to the fighter if he won.

Validation function and the rollDie function have not changed from when they were used in the previous projects/labs. Testing of these two have been completed many times prior, so the testing table will not include the testing of these functions.

Since all the logic for the characters in this program were heavily tested in program 3, there will be minimal testing of the way in which the characters behave. I am assuming all the characters behave as expected, so I will mainly verify that the recovery function is working properly and that the game is scoring wins/losses correctly. I will pay attention to any patterns that emerge while testing the other functions in this program and investigate any odd behavior.

Test Plan for Fantasy Combat

Test Case	Description	Expected Result
Vampire Object Creation in Menu	Create a new Vampire	1. Vampire object should be created on the heap. 2. Member variables should be set to the values in the constructor 3. A Character pointer should be pointing to the newly created object 4. Character pointer should be returned to the play function and stored in the team1/team2 Lineup
Barabarian Object Creation in Menu	Create a new Barbarian	1. Barbarian object should be created on the heap. 2. Member variables should be set to the values in the constructor 3. A Character pointer should be pointing to the newly created object 4. Character pointer should be returned to the play function and stored in the team1/team2 Lineup
BlueMen Object Creation in Menu	Create a new BlueMen	1. BlueMen object should be created on the heap. 2. Member variables should be set to the values in the constructor 3. A Character pointer should be pointing to the newly created object 4. Character pointer should be returned to the play function and stored in the team1/team2 Lineup
Medusa Object Creation in Menu	Create a new Medusa	1. Medusa object should be created on the heap. 2. Member variables should be set to the values in the constructor 3. A Character pointer should be pointing to the newly created object 4. Character pointer should be returned to the play function and stored in the team1/team2 Lineup
HarryPotter Object Creation in Menu	Create a new HarryPotter	1. HarryPotter object should be created on the heap. 2. Member variables should be set to the values in the constructor 3. A Character pointer should be pointing to the newly created object 4. Character pointer should be returned to the play function and stored in the team1/team2 Lineup
Printout after each battle	Each battle should print out a summary	1. Battle should be numbered 2. Team 1 name and type should be displayed 3. Team 2 name and type should be displayed 4. Name of winner should be displayed 5. SP of winner at end of battle should be displayed 6. SP of winner after the recovery function is called should be displayed

Printout after the tournament ends	Tournament recap should print after one of the teams has no more fighters	1. Winner of the tournament should be displayed 2. Team 1 points should be displayed 3. Team 2 points should be displayed
Calculation of the tournament winner	Winner should be calculated after one of the teams have no more players	1. Team with the most points at the end of the tournament should be crowned the winner.
Winner moved to back of queue	Winner of each battle should call nextHead()	1. The team that won the battle should move the winning fighter to the back of the Lineup
Loser moved to stack	Loser of each battle should be moved to the Losers stack	1. After the winner of the battle is determined, the loser character should be added to the top of the Losers stack 2. The Node in the losing team Lineup holding the loser should be deleted
Loser list printout	User chooses to print out the loser list	The stack of losers should be printed to the screen with the most recent loser first and the loser of the first battle at the end.
Recovery function	Winner of each battle should call the recovery function	1. SP of winner should be greater after the function is called, unless the fighter sustained no damage during the battle. 2. SP after the recovery function should not be greater than the character’s max SP
Recovery function for HarryPotter	HarryPotter should call the recovery function if he wins	1. SP of Harry should be greater after the function is called, unless he sustained no damage during the battle. 2. If Harry is on his first life, his SP should be no greater than 10 after the recovery function is called. 3. If Harry is on his last life, his SP should be no greater than 20 after the recovery function is called.
Main Menu1	Program should start at menu1	1. User is asked to choose the character for the first player in the game. 2. The user selects the type of character they want for player 1 3. Menu2 should appear
Main Menu2	Menu should look the same as menu1 except for the selection of the first player to be displayed.	1. User is asked to choose the character for the second player in the game. 2. The first player that was chosen should be displayed in the upper right hand corner. 3. User selects the character and the tests are ran until a winner emerges.
Continue Menu	Press 1 to continue Press 2 to quit	1. Menu1 should appear and the user should have the option to choose their first player. 2. The test should terminate.

Start Menu	1. Start menu should give the option to “play” or “quit” Choose play Choose quit	When play is chosen, the game advances and asks user how many fighters they want on each team. When quit is chosen, the game terminates.
Memory Leak	1. Start and run program using valgrind. Choose play Enter number of fighters Choose fighters and enter names Let tournament run Print loser list Quit program 2. Start and run program using valgrind. Repeat the instruction in scenario 1, only “play again” instead of quit. Play 3 tournaments Quit program 3. Start program and immediately quit.	There should be no memory leaks for any of these cases.

Reflection:

There is not too much for me to discuss in this reflection. I usually have a hard time finding things to talk about, but this time there really isn’t anything to talk about. There are a lot of other thing that I need to do for the end of the quarter, so I am definitely not complaining about the easy time I had with this project. I mentioned some design choices and testing decisions in the section before the test table, so I guess that can count for some of the reflection.

Issues:

I did not have a single issue while completing this project. As I stated earlier, most of the code in this project was taken from other labs/projects and pieced together. There were minimal changes required to make everything work together, and this was the quickest project of the four in this course.