Design Description

Game Play

- Play is chosen from start menu
    - Enter # of fighters for 2 teams
    - For each fighter of each team choose:
        - Character type
        - Character name
        - *allow for multiple of each character type in the respective lineups
- Linked List set up
    - Save fighters into team linked list
    - Set up loser list
    - Properly delete lists
- Fighting round
    - Fighter at the HEAD of the lineup for each team fights
    - WINNER returned to END of lineup
    - LOSER goes to TOP of loser container
        - Only ONE container total to hold ALL defeated fighters from both teams
    - WINNER's health is restored to a certain percentage
    - Lineup order canNOT be changed
        - Except for putting winner/loser in respective containers
    - OUTPUT for EACH round
        - Character Types
        - Chapter Names
        - Who won
- **Tournament ends when** a team doesn't have any more fighters
    - Results of game are printed
        - Final score for each team
        - Display the winner
        - Handle Tie
    - CHOICE to display loser container contents
        - Most recent to first
- Start Menu

Notes

- Containers
    - Must use linked lists
    - Must make the linked lists
- Recovery functions
    - Add member function to the parent class
    - Example: random roll of 5 = 50% recovery
- Start menu
    - Play
    - Exit

- Scoring System (up to me)
    - +1 for win, no points for loose
- Pointers to characters should be put into the line up

Reflection

I did my normal bullet point outline to make sure I hit all the requirements for the assignment. I also added to the hierarchy diagram (in blue). I originally intended for the team1, team 2 and loser list to all be a circular queue, but I realized this wasn't needed for the loser container. Instead, I used a regular doubly linked list from a past lab. This seem more logical for the loser container since it was a more straight forward type of list.

It took me a while to realize that I needed to alter the destructors for the lists, but that was the only major problem I faced when putting this lab together. Other than that, I had quite a few silly errors that were very easy to fix.

Test Table

Test	Input	Test Target	Expected Output	Actual Output
playAgain functions works	1. 1 2. 0	Menu::playAgain()	1. Game play continues with all values reset 2. Game exists	1. Game play continues with all values reset 2. Game exists
Players are made in team 1 or 2 appropriately	1. 3 characters for team 1 2. 3 characters for team 2	Menu::makePlayer	1. Team 1 characters play for team 1 2. Team 2 characters play for team 2	1. Team 1 characters play for team 1 2. Team 2 characters play for team 2
Losers go to front of loser container	Regular game play	Menu::playGame()	1. Loser is taken out of team list 2. Loser added to front of loser list	1. Loser is taken out of team list 2. Loser added to front of loser list

Winners go to back of respective teams	Regular game play	Menu::playGame	1. Winner put at back of team list 2. Health is restored	1. Winner put at back of team list 2. Health is restored
Option to display lists	1. 1 2. 0	Menu::playGame	1. Loser list prints 2. No printing	1. Loser list prints 2. No printing
Vampire Charm overpowers attacks	Regular game play	Vampire::defense	50% of the time: - Message for vampire charm - Opponent attack doesn't do damage	50% of the time: - Message for vampire charm - Opponent attack doesn't do damage
Blue Men loose 1 die after 4 ptr of damage	Regular game play	BlueMen::defense	- 3 dice for health > 8 - 2 dice for health > 4 - 1 die health < 5	- 3 dice for health > 8 - 2 dice for health > 4 - 1 die health < 5
Medusa glare at roll = 12	Regular game play	Medusa::attack	- when attackRoll = 12, damage = 50	- when attackRoll = 12, damage = 50
Harry Potter revives after 1st life	Regular game play	HarryPotter::defense	- after first life, health = 20	- after first life, health = 20
Vampire charm automatically kicks in with medusa stare	Regular game play	Vampire::defense	- attack = 50, attack doesn't work	- attack = 50, attack doesn't work
Each round has two attacks	Regular game play	Menu::playGame	- stats output for 2 attacks per round	- stats output for 2 attacks per round

Attack output is correct	Regular game play	Menu::playGame	Each attack outputs: - attacker type - Defender type - Defender armor pts - Defender strength pts - Attacker roll - Defender roll - Damage inflicted - Defender's remaining strength	Each attack outputs: - attacker type - Defender type - Defender armor pts - Defender strength pts - Attacker roll - Defender roll - Damage inflicted - Defender's remaining strength
Damage = Attacker's roll - defender's roll - defender's armor	Regular game play	All character defense functions	Each round the damage math is as expected	Each round the damage math is as expected
Play Again option works correctly	1. 1 2. 0	Menu::playAgain	1. New game starts with initial prompts 2. Game exits	1. New game starts with initial prompts 2. Game exits
Players are assigned correctly	1. 1 2. 2 3. 3 4. 4 5. 5	Menu::makePlayer	Player made: 1. Barbarian 2. Blue Men 3. Harry Potter 4. Medusa 5. Vampire	Player made: 1. Barbarian 2. Blue Men 3. Harry Potter 4. Medusa 5. Vampire
Any player health < 1, game ends	Regular game play	Menu::playGame	Winner message displayed	Winner message displayed
INPUT VALIDATION				
Each prompt rejects char	1. hgjhjf	isInteger()	1. Error asking for integer	1. Error asking for integer
Each prompt rejects floats	1. 1.11	isInteger()	1. Error asking for integer	1. Error asking for integer

Each prompt rejects numbers out of range	1. 4 2. -1 3. 0	isBetween()	1-2. Error asking for value in range 3. Regular game play	1-2. Error asking for value in range 3. Regular game play
FLIP				
Valgrind: no segmentation faults	valgrind lab3	Whole program	1. No errors	ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
Valgrind: No memory leaks or	valgrind --tool=memcheck --leak-check=full --show-leak-kinds=all --track-origins=yes ./ lab3	Whole program	1. No errors	ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
Complies on school server	1. g++ -std=c++0x main.cpp ant.cpp menu.cpp -o game 2. ./game	Whole program	1. nothing 2. First prompt	1. nothing 2. First prompt

Reflection

This project felt more straight forward than some of the past one, but I think that's also because the base concepts were pretty familiar. The class hierarchy diagram helped organize the various child classes and see what work needed to be done in each. With the diagram it was easy to see that the child classes used either a default attack or defense function.

I originally expected to have a Game class to control the actions of the game. However, when coding, the Menu class ended functioning as the controller for the game. This made the game easier to build since I wouldn't have to worry about passing more pointers back and forth between two classes.

Overall, this project was pretty straight forward to code and I spent much less time than usual fighting through valgrind errors.

Class Hierarchy Diagram