Fantasy Combat Tournament

**Program flow (blue text added after initial planning)**

Initialize random seed, empty queues for teams and defeated fighters

Welcome user

Prompt for number of fighters on each team

Display character selection menu

Prompt user to select fighters

(instantiate character objects, give characters names)

Combat proceeds, printing stats after each round

(loop until dead: C1 attacks, check dead, C2 attacks, check dead)

Attack: Attacker rolls attack die, defender roll defense die and calculates damage,

Survivor is added to the back of the team, loser is added to loser pile.

Display final result and score

Prompt user to repeat

**Node struct**

Character \*

string name

Next node

Previous node

Queue class

bool isEmpty()

void addBack()  
 Node \* getFront()

void removeFront()

void removeFrontandDelete()

void printQueue()

**Character class**

protected:

int armor

int strength\_points

bool canRevive()

public:

constructor() (sets armor and SP)

attack() = 0;

defend() = 0;

takeDamage()

getArmor()

getSP()

setSP()

name()

getCanRevive()

setCanRevive()

**Barbarian/BlueMen/Vampire/Harry Potter/Medusa classes**

public:

constructor() (sets armor = 0 and SP = 12)

attack() (returns result of 2d6 roll)

defend() (returns result of 2d6 roll)

**Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test case** | **Input Values** | **Affected functions** | **Expected outcomes** | **Observed outcomes** |
| Input validation | Ints are not ints, etc. | User input functions | Reprompt for correct input | Reprompt for correct input |
| Barbarian is instantiated with Barbarian stats | - | Barbarian() | Barbarian is instantiated with Barbarian stats | Barbarian is instantiated with Character junk stats. Fixed by removing duplicate armor/SP declaration in Barbarian.hpp |
| Identical characters get different names | - | Game setup functions | Identical characters get different names | Identical characters get different names |
| Damage equals attack – (defense + armor), and SP is deducted appropriately | Character rolls | Arrack()  Defend() | Damage equals attack – (defense + armor), and SP is deducted appropriately | Damage equals attack – (defense + armor), and SP is deducted appropriately |
| Dead fighters are moved to loser pile | - | playGame () | Dead fighters are moved to loser pile | Dead fighters are moved to loser pile |
| Surviving fighters are healed | - | playGame() | Fighters are healed up to max SP | Fighters are healed above max SP. Fixed by adding max SP value to Character class |
| Surviving fighters are moved to back of lineup with current SP | - | playGame() | Surviving fighters are moved to back of lineup with current SP | Surviving fighters are moved to back of lineup at full SP. Fixed by adding a copy constructor to Character class |
| The loser pile is displayed correctly upon request | - | showLosers() | The loser pile is displayed correctly upon request | The loser pile is displayed correctly upon request |
| Queues are deleted without memory leaks | - | ~Queue() | Queues are deleted without memory leaks | Queue deletion causes so many memory leaks. Fixed by adding new removeFrontandDelete() function, and editing destructor |
| Fighters are deleted after combat ends | - | Game setup functions | Fighters are deleted after combat ends | Fighters are deleted after combat ends |
| Medusa’s Glare is instant kill | 12 | Medusa.attack() | Medusa’s Glare is instant kill | Medusa’s Glare is instant kill |
| Vampire’s Charm prevents all damage, including Glare |  | Vampire.defend() | Vampire’s Charm prevents all damage, including Glare | Vampire’s Charm prevents all damage, including Glare |
| Harry Potter revives once upon death (spoilers!) |  | HarryPotter.defend() | Harry Potter revives once upon death | Harry Potter revives once upon death |
| Blue Men’s defense decreases upon damage |  | BlueMen.defend() | Blue Men’s defense decreases upon damage | Blue Men’s defense decreases upon damage |

**Reflection**

The majority of this project was straightforward, thanks to being able to reuse the code from the previous projects and recent labs. That being said, this was the first time I experienced significant memory leak issues that took a while to quash.

* Editing the Queue class from the previous project to take a Character pointer was quite simple.
* I stored the character name in the node rather than the character object itself, which I now realize is probably not the best practice. I think it would be a simple fix to add a name string to character and have the node access getter/setter methods, but it works as is and I don’t want to risk breaking things by adding changes at this stage.
* Compared to some of my peers that I’ve seen in the code sharing threads, I have a tendency to have individual functions do too many things, so I’m making an effort to modularize my code into smaller functions. I think I did a decent job this time, but my playGame() function is a bit convoluted.
* Once the game itself was up and running as per the specs, I started to tackle the (many) memory leaks. Keeping track of what was pointing to Character objects and when it was okay to delete them was confusing at first, but I was eventually able to figure it out. The breakthrough came when I figured to have two separate removeFront() functions: one that deleted the Character contained in the node, and one that didn’t. I used the one that preserved the character to move the node into different piles after combat, and the one that deleted them after the game was finished.