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,

Display final result

Prompt user to repeat

**Node struct**

Character \*

string name

Next node

Previous node

**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)

name()

**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 |
| 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 |
| Fighters are deleted after combat ends | - | Game setup functions | Fighters are deleted after combat ends | Fighters are deleted after combat ends |

**Reflection**

This project was not nearly as time consuming or difficult as Zoo. I’m starting to feel a lot more comfortable with pointers and polymorphism.

* As per your suggestion, I created the Character and Barbarian classes first, then made a basic main function to test attacking/defending/death. I had an issue where my Barbarian objects were taking the junk armor and SP values from the Character constructor, and eventually figured out it was because I redeclared armor and SP in the Barbarian.hpp. Once I got that figured out it was smooth sailing.
* Differentiating between two different Barbarians was solved by adding a Name string to each fighter type, and appending 1 and 2 if the chosen fighters had the same name.
* I added Vampire next. Implementing Charm in the class itself was fairly simple, but designing the attack function to recognize when Charm activated took me a few minutes. I eventually settled on having Charm return a specific, ridiculously large number for the defense roll was the easiest to implement. Having figured that out, I used the same approach for Medusa’s Glare.
* The last hurdle was Harry Potter, and I eventually settled on including a bool in Character for whether a fighter is “revivable” or not. With all the pieces in place, the rest went pretty smoothly.