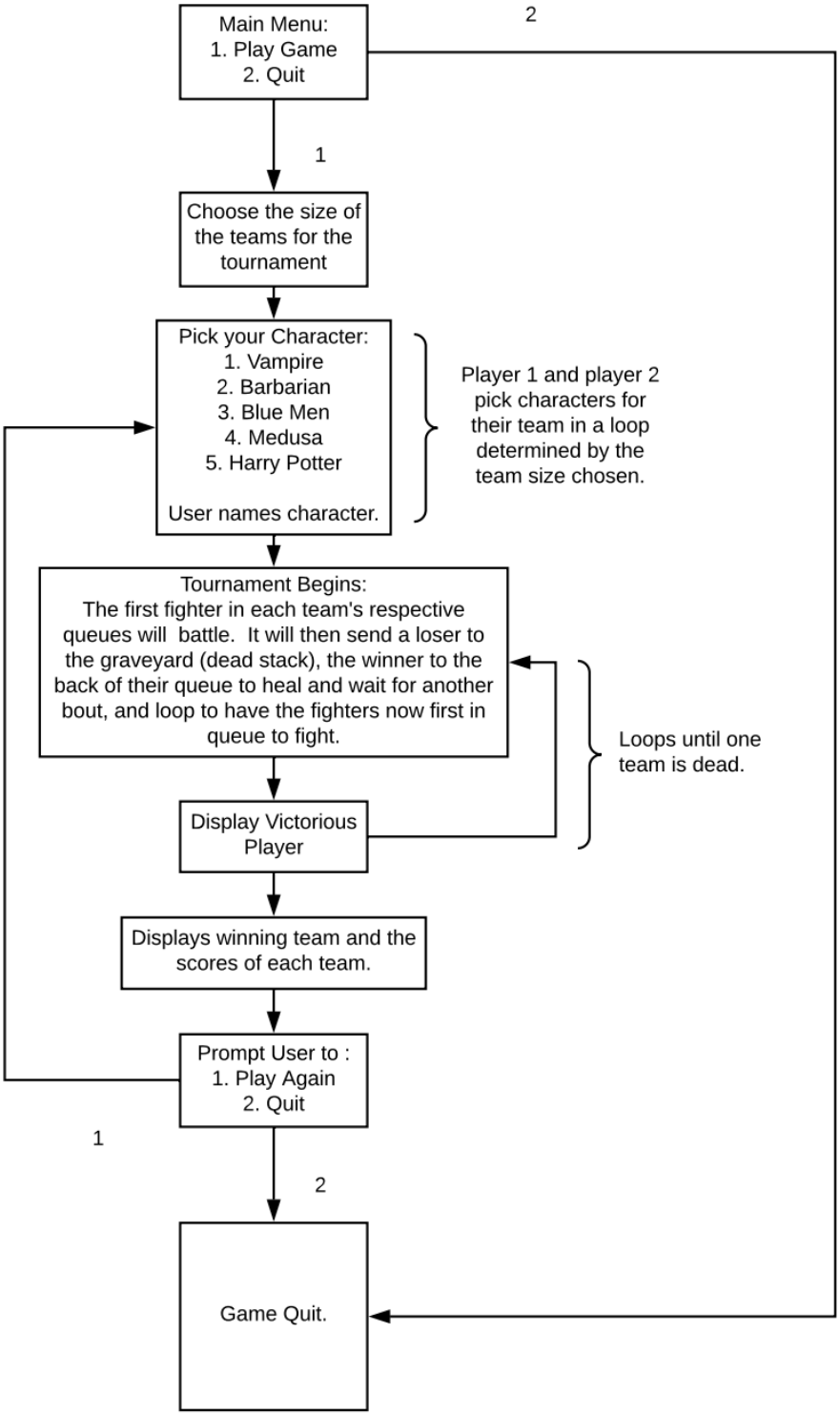
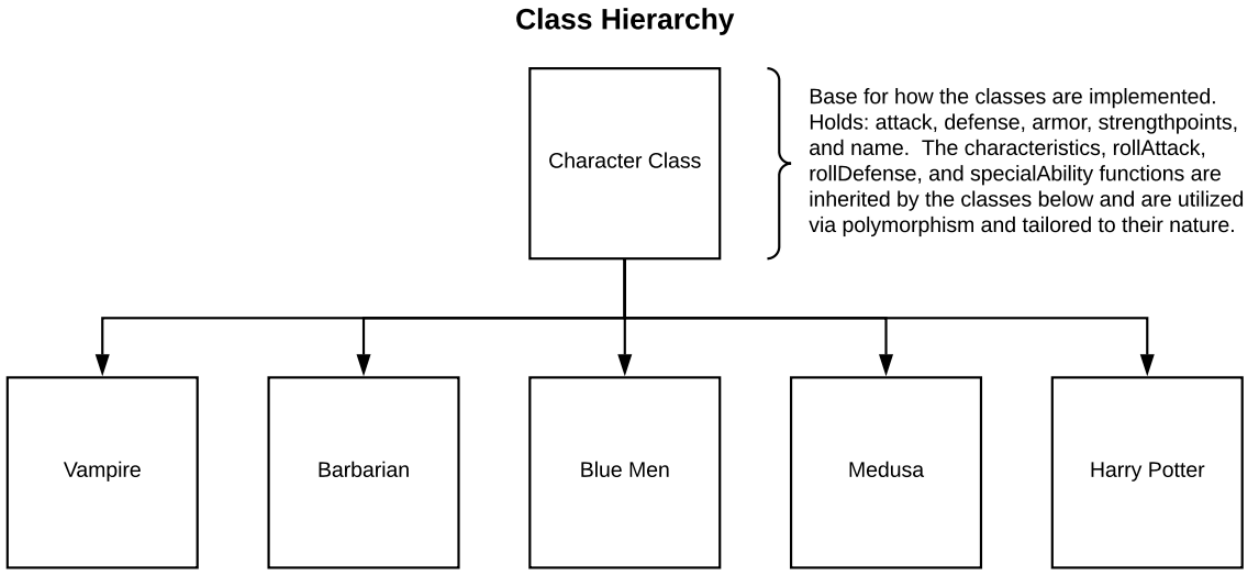
Devin Gendron
Project 4 – Fantasy Combat Tournament

Flowchart:



Hierarchy:



Vampire:Constructor initializes attack (0), defense (0), armor (1), strengthPoints(18), and name (Vampire).
Characteristics: Suave, debonair, but vicious and surprisingly resilient.
rollAttack:Rolls one 12 sided die.
rollDefense: Rolls one 6 sided die.
specialAbility: Charm - 50% of the defense rolls activate charm special ability where the Vampire will not be attacked.

Barbarian: Constructor initializes attack (0), defense (0), armor (0), strengthPoints(12), and name (Barbarian).
Characteristics: Think Conan or Hercules from the movies. Big sword, big muscles, bare torso.
rollAttack: Rolls two 6 sided die.
rollDefense: Rolls two six sided die.
specialAbility: No special ability.

Blue Men: Constructor initializes attack (0), defense (0), armor (3), strengthPoints(12), and name (Blue Men).
Characteristics: They are small, 6 inch tall, but fast and tough. They are hard to hit so they can take some damage. They can also do a LOT of damage when they crawl inside enemies' armor or clothing.
rollAttack: Rolls two 10 sided die.
rollDefense: Rolls die depended on special ability. Rolls three six sided defense die, then 2, then 1. Rolls dependent on strengthPoints.
specialAbility: SP > 8 = three defense rolls, SP > 4 = two defense rolls, SP > 0 && SP < 4 = 1 defense roll.

Medusa: Constructor initializes attack (0), defense (0), armor (3), strengthPoints(8), and name (Medusa).
Characteristics: Scrawny lady with snakes for hair which helps her during combat. Just don't look at her!
rollAttack: Rolls two 6 sided die.
rollDefense: Rolls one 6 sided die.
specialAbility: If attack roll == 12, then Medusa uses glare and turns her opponent to stone.

Harry Potter: Constructor initializes attack (0), defense (0), armor (0), strengthPoints(10), and name (Harry Potter).
Characteristics: Harry Potter is a wizard.
rollAttack: Rolls two 6 sided die.
rollDefense: Rolls two 6 sided die.
specialAbility: Hogwarts ability - Harry returns to life the first time he dies.

PseudoCode:

Main()

//Menu function
//switch statements to play game or quit
//If game is played, ask user for size of teams
//ask user to pick team character types and name them
//enter teams into combat simulation
//rotate and empty queues as necessary
//display winner

Menu()

//print statements
//use inputvalidation func to receive proper return
//return value

playAgainMenu() //user chooses if they would like to play the game again

//print statements
//use inputvalidation func to receive proper return
//return value

charMenu()

//using inputval(); to choose character types.

Combat()

//sets characters to their players (Player 1 and Player 2)
//using loop, have player 1 attack and player 2 defend, then if player 2 is alive, they attack player 1.
//While combat ensues, print information on the characters as they battle.
//If there is a victorious player, display who won and who lost.
//else, ask they user to continue the combat.

-This loops until there is a victor and the loop is ended.

Queue()

//Creates team object (Team 1 and Team 2)
//Allows teams to be cycled through, remove team members, and can send them to the Back

Node()

//linked list container for the dead. Acts as a stack.

Test Plan:

Menu();

Test Case	Expected Outcomes	Observed Outcomes
Input letters	Inputvalidation(); returns "Incorrect entry…". Restarts loop.	Incorrect entry statement printed, loops to restart input request.
Input symbols or space	Inputvalidation(); returns "Incorrect entry…". Restarts loop.	Incorrect entry statement printed, loops to restart input request.
Input too low	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Input too high	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Empty input	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Correct Input	Counter for exit increments, print correct entry statement. Program continues.	Returns value to main to continue with program

playAgainMenu();

Test Case	Expected Outcomes	Observed Outcomes
Input letters	Inputvalidation(); returns "Incorrect entry…". Restarts loop.	Incorrect entry statement printed, loops to restart input request.
Input symbols or space	Inputvalidation(); returns "Incorrect entry…". Restarts loop.	Incorrect entry statement printed, loops to restart input request.
Input too low	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Input too high	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Empty input	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Correct Input	Counter for exit increments, print correct entry statement. Program continues.	Returns value to main to continue with program

charMenu();

Test Case	Expected Outcomes	Observed Outcomes
Input letters	Inputvalidation(); returns "Incorrect entry…". Restarts loop.	Incorrect entry statement printed, loops to restart input request.
Input symbols or space	Inputvalidation(); returns "Incorrect entry…". Restarts loop.	Incorrect entry statement printed, loops to restart input request.
Input too low	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Input too high	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Empty input	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Correct Input	Counter for exit increments, print correct entry statement. Program continues.	Returns value to main to continue with program

Medusa vs Harry

Test Case	Expected Outcomes	Observed Outcomes
Medusa uses glare	Harry survives using hogwarts	Harry survived.
Medusa uses glare again	Harry dies after using up his Hogwarts ability	Harry dies.

Vampire vs Medusa

Test Case	Expected Outcomes	Observed Outcomes
Medusa uses glare	Vampire uses charm	Vampire survives
Medusa uses glare	Vampire uses no ability	Vampire dies

Reg damage vs Harry

Test Case	Expected Outcomes	Observed Outcomes
Char attacks harry	Harry survives using hogwarts	Harry survived.
Char attacks harry	Harry dies after using up his Hogwarts ability	Harry dies.

Reg damage vs Vampire

Test Case	Expected Outcomes	Observed Outcomes
Char attacks vampire	Vampire uses charm	Vampire survives
Char attacks vampire	Vampire uses no ability	Vampire dies

Medusa vs Char

Test Case	Expected Outcomes	Observed Outcomes
Medusa uses glare	Character dies	Char died

Blue Men taking damage

Test Case	Expected Outcomes	Observed Outcomes
Blue Men takes 4 points of dmg	They lose a defense die	Defense die is lost
Blue Men take 8 points of total damage	They lose two defense die	Defense die is lost

Add to team queue

Test Case	Expected Outcomes	Observed Outcomes
Receives character object	Character gets added to back	As expected

Set Player

Test Case	Expected Outcomes	Observed Outcomes
Receives character object from front of queue	Sets character for combat sequence	As expected

Character wins

Test Case	Expected Outcomes	Observed Outcomes
Move character to back of team queue	Character becomes last in queue	As expected
Recover strength points	Random amount of strength points is recovered in regards to amount missing	As expected
Sets opponent to graveyard	Opponent enters dead container	As expected
Remove enemy from their team queue	Dead enemy is removed from their team queue	As expected
Increment winning team's points	Increment points for winner	As expected

Game ends

Test Case	Expected Outcomes	Observed Outcomes
Display winner and team scores	Information printed to screen	As expected
User can choose to display list of dead or skip	Traverse dead stack and print or skip to end of game	As expected
Deletes all objects from teams	All dynamically allocated objects are deleted from team queues	As expected
Deletes all dead objects from dead	All dynamically allocated objects are deleted from dead stack	As expected

Reflection:

This reflection will mainly go over design decisions rather than design changes. I felt super comfortable with Lab 6 and Lab 7, so creating a queue and linked-list stack for my project wasn't difficult to implement. Project 3 also gave me little trouble, so the culmination of these assignments in Project 4 was fairly straight forward. I was happy with how I was able to take Project 3 and implement the tournament format around and within it. I got it running in just a day after designing the changes and only slight debugging followed - so I didn't plan any further implementation.

So, how did my design work? In my Project 3 implementation, I had created two character objects that were passed into the combat class that were then set to "Player 1" and "Player 2" character objects. These objects were then used in my combat class to battle each other and determine a winner. Using this current implementation in Project 4, the user enters the size of the teams, picks and names characters who are then added to a team queue. The head of the queues that represent each team are then passed into the set player function in the combat class. This then has them battle like in Project 3 and returns the winner and loser of each duel. Depending on who is victorious and who was vanquished, they are moved to the back of their queue to recover and the loser is sent to the graveyard or "dead stack" which is a doubly-linked list and is removed from their team queue. The program then loops and sets the new heads of the queue to battle. As each battle comes to an end, points are accrued for the victors. When an entire team is dead, the tournament ends and displays the victors and the point distribution.

I'm particularly proud of this implementation, because I realized I could keep my combat function pretty much the same and just alter my main to take care of the tournament functionality. All I really needed to do was have the team queues properly rotate through and pass their characters into battle accordingly. Then, dead characters would need to be moved to the graveyard and properly removed from their queues.

Overall, the last few weeks of this class have been really good for me in that I feel I really understand pointers, classes, inheritance, and polymorphism. Knowing how important these

key subjects are and the fact that I am getting better and better at implementing them is very reassuring as I move further into this program. If I were to do anything differently for Project 4, I would try and streamline my code to do more with less. It is pretty long and I think with more time, I would be able to write cleaner more succinct code that accomplished what I did in this project.