Programming Project 4
Justin Tromp
CS 162 – Winter 2019

Additions to programming project 3:

FantasyGame Class

Functionality Added:

Team Building - Prompt user to enter number of fighters for both teams. Once team sizes are established, prompt user to choose type of character for each fighter and prompt user to enter name for that fighter. More than 1 of same character type is possible. Order of team lineup is same as user entry.

Tournament Start – After teams are built, tournament starts. Fighter at front of each lineup will battle in same manner as project 3. Winner of each fight is placed at back of lineup in team. Loser moves to top of loser container (only one container for all losers of both teams). Winning fighter's strength value is partially restored (some percentage of damage inflicted is restored). Lineup order cannot be changed from user implementation other than placement of winners at back of lineup and moving losers to top of loser container.

Tournament End – Tournament ends when one of the teams does not have any more fighters to fight. Upon game end, results of game are printed to user and allow user to have the choice of displaying contents of loser container.

Containers: Create queue-like linked list structure as in Lab 7 for both tournament teams and use similar linked list structure for stack/pile like function of losers queue. Add additional functions to lab 7 to allow for removal of node from front of queue and placement of that node at end of queue, as well as adding to front of queue for the stack/pile like functionality of the losers list.

Variables: struct CharacterNode {CharacterNode* next & prev and Character* combatCharacters} to represent nodes in queues.

Three queue structures:
CharacterNode* combatTeam_1
CharacterNode* combatTeam_2
CharacterNode* loserQueue

Three head pointers, pointing to head of each queue:
CharacterNode* headTeam_1
CharacterNode* headTeam_2
CharacterNode* headLoserQueue

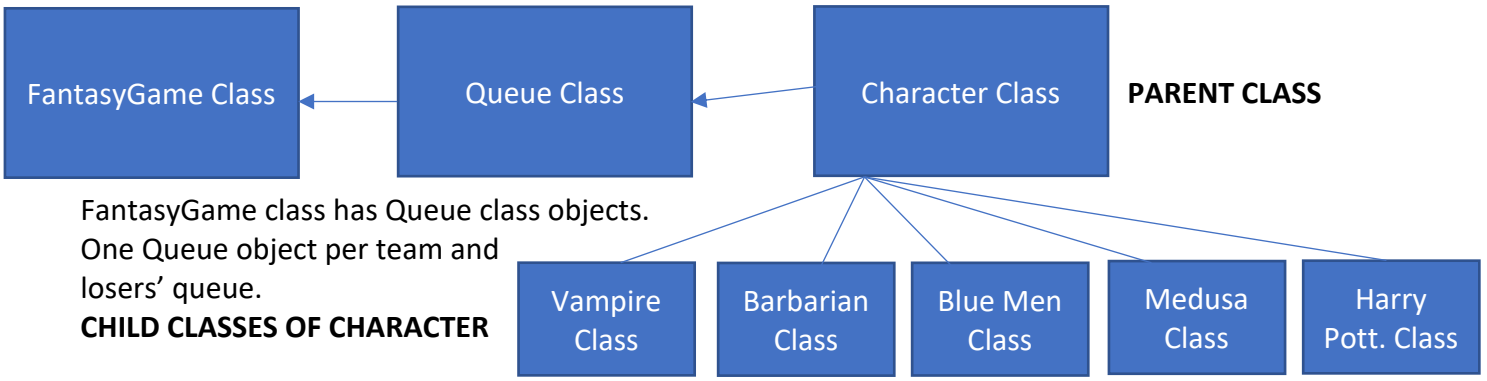
Functions:
startMenu/endMenu – Use same start/end menu functionality created in project 3.

int teamMenu – Prompts user for integer input for number of fighters for each team. Integer value is accepted/validated through present integer validation function. Validated integer value is returned to calling function.
playMenu (current implementation to choose character type in project 3) – Use same functionality as project 3, except add linked list implementation in place of vector implementation (must change addCharacter function). Function will be called for each character added to team. Additional functionality to be added: Must prompt user for name of character on team and pass name as parameter to addCharacter function.
addCharacter Function – Current function adds character as vector. I will be changing this function to add a node to queue list. Function will take additional string parameter for name of character to be added to team.
User Input Flow: Start game - > teamMenu (size of teams) - > playMenu (add characters to teams)
removeFront Function – Removes node from front of list and returns node
deleteFront Function – Delete node at front of list.
addBack Function – Adds node to back of list. Can add a new node or receive node from removeFront to add to back of list.
addFront Function – Used to add node to front of list (specifically used for losers pile)
printQueue Function – Prints out information from Queue (specifically prints all characters in Queue from front to back and team information for each fighter) for losers pile output at end. Use regular functions from Lab 7 such as isEmpty and getFront.

Game Output: Battle Output Function – Display type of character and name of two fighters battling and which character won the combat. For example: *Round 1: Team A Blue Man No.1 vs. Team B Harry Potter No.1, Harry Potter No.1 Won!*
Tournament Output Function – At end of tournament, game displays final score for each team and the winner of the tournament (or in case of tie, state that tie occurred). Scoring system will be calculated as follows: For each battle win, +2 points, for each battle loss, -1 points. At end of output, allow user to choose whether to display content of loser container. Container is printed from top to bottom (last to first). At end of tournament output, ask if user would like to play again or exit.

Character Class
Functionality Added: Recovery
Functions:
void Recover () function – Recovers percentage of lost strength points of winner of fight. Calls randomInt function to generate random integer value of 1 through 9. 1 equates to 10 percent recovery, 2 equates to 20 percent recovery, etc, up through 90 percent. Since it is partial damage recovery, 100 percent is not possible. As a design decision, if a fractional recovery value occurs such as 6.5 strength points, damage recovered is rounded down regardless of fractional value. Thus 6.8 would be 6, .9 would be 0, etc… Thus a possible recovery could be anywhere from 0 to 90 percent of the strength points lost during a fight.

Class Hierarchy Queue class QueueNode struct holds Character class object pointer.



FantasyGame class has Queue class objects.
One Queue object per team and losers' queue.
CHILD CLASSES OF CHARACTER

TESTING
MEMORY LEAK TESTS
Select 2 (exit) at start: No memory leaks
Select 1 and play tournament with 1 fighter each team: No memory leaks
Select 1 and play tournament with 2 fighters each team: No memory leaks
Select 1 and play tournament 4 fighters each team (print losers): No memory leaks
Select 1 and play tournament 4 fighters each team (do not print losers): No memory leaks
Select 1 and play tournament 4 fighters each team (print losers and detailed battle output selected): No memory leaks
Select 1 and play tournament 4 fighters each team (Play twice, print losers, and detailed battle output selected): No memory leaks

INPUT VALIDATION TESTS

Test Case	Input Value	Validate Function Used	Expected Outcomes	Observed Outcome Initial
Start Menu	0	validateIntRange	Re-prompt/err	Re-prompt/err
	3		Re-prompt/err	Re-prompt/err
	12		Re-prompt/err	Re-prompt/err
	21		Re-prompt/err	Re-prompt/err
	Enter		Re-prompt/err	Re-prompt/err
	Space		Re-prompt/err	Re-prompt/err
	1a		Re-prompt/err	Re-prompt/err

	a1		Re-prompt/err	Re-prompt/err
	1.5		Re-prompt/err	Re-prompt/err
	1 1		Re-prompt/err	Re-prompt/err
	1		Start Game	Start Game – Char/Team Selection
	2		Quit Program	Quit Program
End Menu	0	validateIntRange	Re-prompt/err	Re-prompt/err
	3		Re-prompt/err	Re-prompt/err
	12		Re-prompt/err	Re-prompt/err
	21		Re-prompt/err	Re-prompt/err
	Enter		Re-prompt/err	Re-prompt/err
	Space		Re-prompt/err	Re-prompt/err
	1a		Re-prompt/err	Re-prompt/err
	a1		Re-prompt/err	Re-prompt/err
	1.5		Re-prompt/err	Re-prompt/err
	1 1		Re-prompt/err	Re-prompt/err
	1		Start Game	Start Game Again – Empty Queues
	2		Quit Game	Quit Program
Team Menu	0	validateIntRange	Re-prompt/err	Re-prompt/err
Team size, 1-20	21		Re-prompt/err	Re-prompt/err
	20 1		Re-prompt/err	Re-prompt/err
	1 20		Re-prompt/err	Re-prompt/err
	Enter		Re-prompt/err	Re-prompt/err

	Space		Re-prompt/err	Re-prompt/err
	1a		Re-prompt/err	Re-prompt/err
	a1		Re-prompt/err	Re-prompt/err
	1.5		Re-prompt/err	Re-prompt/err
	1 1		Re-prompt/err	Re-prompt/err
	1		Set team size to 1	Set team size to 1
	20		Set team size to 20	Set team size to 20
Losers Print	0	validateIntRange	Re-prompt/err	Re-prompt/err
1 for print losers	3		Re-prompt/err	Re-prompt/err
2 for no print	12		Re-prompt/err	Re-prompt/err
	21		Re-prompt/err	Re-prompt/err
	Enter		Re-prompt/err	Re-prompt/err
	Space		Re-prompt/err	Re-prompt/err
	1a		Re-prompt/err	Re-prompt/err
	a1		Re-prompt/err	Re-prompt/err
	1.5		Re-prompt/err	Re-prompt/err
	1 1		Re-prompt/err	Re-prompt/err
	1		Print Losers	Printed Losers List
	2		Skip Print	Skipped Print
Attack Menu	0	validateIntRange	Re-prompt/err	Re-prompt/err
1 for simple output	3		Re-prompt/err	Re-prompt/err

2 for complete output	12		Re-prompt/err	Re-prompt/err
	21		Re-prompt/err	Re-prompt/err
	Enter		Re-prompt/err	Re-prompt/err
	Space		Re-prompt/err	Re-prompt/err
	1a		Re-prompt/err	Re-prompt/err
	a1		Re-prompt/err	Re-prompt/err
	1.5		Re-prompt/err	Re-prompt/err
	1 1		Re-prompt/err	Re-prompt/err
	1		Print Basic Battle Info	Printed only end of battle results
	2		Display all battle info	Prints all roll and battle information as in Project 3

Character Naming Menu – Input validation works (can only select 1 or 2) with 1 allowing user to name the fighter as they please and 2 allowing the program to name the fighter automatically with the convention as follows: Checks number of that character type in that team and names as “Barbarian No.” + the number of that character type. For example: If team 1 has 2 barbarians and a third is being added, the third will be named “Barbarian No. 3” under the naming convention if user elects to have the program name the fighter. The No. 3 is added regardless of whether the user names or has the program name the prior two barbarians.

QUEUE ORDERING TEST (ADDITIONAL TESTING IN DEBUG MODE WITH ADDRESSES CHECKS OUT AS WELL)

Team 1

Fighters Added (Type (Name)):
1. Vampire (Vampire No. 1)
2. Vampire (Vampire No. 2)
3. Vampire (Vampire Me)
4. Medusa (Medusa No. 1)

Team 2

Fighters Added:
1. Medusa (Medusa No. 1)
2. Medusa (Medusa No. 2)

3. Medusa (Medusa No. 3)
4. Harry Potter (Harry Potter No. 1)

Round 1
Winner: Team 1 – Vampire No. 1
Loser: Team 2 – Medusa No. 1

Round 2
Winner: Team 1 – Vampire No. 2
Loser: Team 2 – Medusa No. 2

Round 3
Winner: Team 1 – Vampire Me
Loser: Team 2 – Medusa No. 3

Round 4
Winner: Team 2 – Harry Potter No. 1
Loser: Team 1 - Medusa No. 1

Round 5
Winner: Team 2 – Harry Potter No. 1
Loser: Team 1 – Vampire No. 1

Round 6
Winner: Team 1 – Vampire No. 2
Loser: Team 2 – Harry Potter No. 1

Losers List
Expected:
Team 2 - Harry Potter No. 1
Team 1 – Vampire No. 1
Team 1 - Medusa No. 1
Team 2 – Medusa No. 3
Team 2 – Medusa No. 2
Team 2 – Medusa No. 1

Actual:
Team 2 - Harry Potter No. 1
Team 1 – Vampire No. 1
Team 1 - Medusa No. 1
Team 2 – Medusa No. 3
Team 2 – Medusa No. 2
Team 2 – Medusa No. 1

Fighters are moved to back when they win and losers are added to front of losers pile after loss as expected with no memory issues.

Expected Score: Team 1 – 6 Team 2 – 0
Actual Score: Team 1 – 6 Team 2 – 0

Score is calculated correctly at +2 per team win and -1 per team loss.

Retest with 1 fighter each team for edge case:
Team 1 Fighters
Vampire No. 1

Team 2 Fighters:
Vampire No. 1

Round 1:
Winner – Team 2 Vampire No. 1
Loser – Team 1 Vampire No. 1

Losers Expected:
Team 1 - Vampire No. 1

Losers Actual:
Team 1 - Vampire No. 1

Expected Score:
Team 1: -1
Team 2: 2

Actual Score:
Team 1: -1
Team 2: 2

Edge case of 1 fighter each team also works correctly with no memory issues.

TEST RECOVERY
TOURNAMENT ROUND 1
WINNER
Winning Team: Team 2
Character Type: Harry Potter
Fighter Name: Harry Potter No. 1
Winning fighter Harry Potter Harry Potter No. 1 has recovered 2 strength points.

LOSER

Losing Team: Team 1
Character Type: Medusa
Fighter Name: Medusa No. 1

TOURNAMENT ROUND 2
WINNER
Winning Team: Team 1
Character Type: Medusa
Fighter Name: Medusa No. 2
Winning fighter did not take any damage this battle. No damage was recovered.

LOSER
Losing Team: Team 2
Character Type: Medusa
Fighter Name: Medusa No. 1

TOURNAMENT ROUND 3
WINNER
Winning Team: Team 2
Character Type: Harry Potter
Fighter Name: Harry Potter No. 2
Winning fighter Harry Potter Harry Potter No. 2 has recovered 4 strength points.

LOSER
Losing Team: Team 1
Character Type: Medusa
Fighter Name: Medusa No. 3

TOURNAMENT ROUND 4
WINNER
Winning Team: Team 1
Character Type: Harry Potter
Fighter Name: Harry Potter No. 1
Winning fighter Harry Potter Harry Potter No. 1 has recovered 5 strength points.

LOSER
Losing Team: Team 2
Character Type: Barbarian
Fighter Name: Barbarian No. 1

TOURNAMENT ROUND 5
WINNER
Winning Team: Team 2
Character Type: Harry Potter
Fighter Name: Harry Potter No. 1
Winning fighter Harry Potter Harry Potter No. 1 has recovered 5 strength points.

```
LOSER
Losing Team:    Team 1
Character Type: Medusa
Fighter Name:   Medusa No. 2

TOURNAMENT ROUND 6
WINNER
Winning Team:   Team 2
Character Type: Harry Potter
Fighter Name:   Harry Potter No. 2
Winning fighter Harry Potter Harry Potter No. 2 has recovered 1
strength points.

LOSER
Losing Team:    Team 1
Character Type: Harry Potter
Fighter Name:   Harry Potter No. 1
```

Strength points recovered after each round. Functionality additionally viewed in debug mode to ensure that calculation is performed accurately. 10-90% recovery of strength points lost is possible, with a possibility of 0 recovery since float values are rounded down. Thus, 1.8 strength recovery is 1.

NOTE: In above output, I made a design change to the recovery statement and removed the character type. Although including both the character type and the character name worked for human named fighters, it created double outputs for default names.

CHARACTER SELECTION TEST –

- All characters can be chosen by typing name.
- Character types can be selected in varying caps due to "to_upper" function allowing for easier input.
- Anything other than a character's name is rejected and requests new input from user.

PROJECT 3 TEST RESULTS BELOW FOR INDIVIDUAL CHARACTER BATTLES: Due to changes in Character and child class codes to allow different output types, I did a quick re-test of calculations and special abilities. All abilities and rolls/attributes worked as expected.
GAMEPLAY TEST (ONLY TEST EACH FUNCTION, NO NEED TO TEST EVERY CHARACTER COMBINATION) – TOP ROW OF EACH TABLE IS ROUND 1 INCREMENTS ACCORDINGLY AS ATTACKER/DEFENDER ALTERNATE. LAST ROW IS FINAL ROUND IF SPECIFIED TO TEST END.

Barbarian vs. Barbarian # 1 –

Attacker	Attack Roll	Damage Inflicted	Defender	Defense Roll	Armor	Strength (Start/After)	Event Info
Barbarian	6	Expected: 0 Actual: 0	Barbarian	11	0	12/12	

Barbarian	8	Expected: 0 Actual: 0	Barbarian	9	0	12/12	
Barbarian	5	Expected: 0 Actual: 0	Barbarian	11	0	12/12	
Barbarian	10	Expected: 7 Actual:	Barbarian	3	0	12/12	**ERROR**

Changes Made: Strength was being displayed before subtracting value.

Barbarian vs. Barbarian # 2 – Check!

Attacker	Attack Roll	Damage Inflicted	Defender	Defense Roll	Armor	Strength (Start/After)	Event Info
Barbarian1	10	Expected: 2 Actual: 2	Barbarian2	8	0	12/10	
Barbarian2	8	Expected: 0 Actual: 0	Barbarian1	12	0	12/12	
Barbarian1	6	Expected: 4 Actual: 4	Barbarian2	2	0	10/6	
Barbarian2	7	Expected: 0 Actual: 0	Barbarian1	10	0	12/12	
FINAL ROUND 9: Barbarian2	10	Expected: 7 Actual: 7	Barbarian1	3	0	3/-4	**Barbarian Died**

Changes Made: Made design decision to change strength to 0 at death instead of displaying a negative strength value.

Barbarian vs. Blue Men – Check!

Attacker	Attack Roll	Damage Inflicted	Defender	Defense Roll	Armor	Strength (Start/After)	Event Info
Barbarian	12	Expected: 1 Actual: 1	Blue Men	8	3	12/11	
Blue Men	14	Expected: 6 Actual: 6	Barbarian	8	0	12/6	
Barbarian	10	Expected: 0 Actual: 0	Blue Men	13	0	11/11	
Blue Men	2	Expected: 0 Actual: 0	Barbarian	8	0	6/6	

Changes Made: Made design decision to output number of dice rolled for attack and defense.

Harry Potter vs. Blue Men (Test for Hogwarts Ability) – Check!

Attacker	Attack Roll	Damage Inflicted	Defender	Defense Roll	Armor	Strength (Start/After)	Event Info

Harry	8	Expected: 0 Actual:	Blue Men	7	3	12/12	
Blue Men	10	Expected: 6 Actual:	Harry	4	0	10/4	
Harry	9	Expected: 0 Actual:	Blue Men	12	3	12/12	
Blue Men	15	Expected: 8 Actual:	Harry	7	0	4/20	**Used Hogwarts Ability**
Final Round 20: Blue Men	19	Expected: 13 Actual: 13	Harry	6	0	1/0	**Died second time.**

Blue Men vs. Blue Men (Testing for Mob) – Initial test showed some poor wording during output. Stated that 2 blue men died on same round as death of all blue men.
Changes Made: Added if statement in mob ability function to check strength level. If strength is 0, do not state reduction of defense dice.
Blue Men vs. Blue Men (Testing for Mob) – Mob worked correctly and removed defense dice.

Vampire vs. Blue Men (Test for Vampire Charm – Modified Defense) – Check!

Attacker	Attack Roll	Damage Inflicted	Defender	Defense Roll	Armor	Strength (Start/After)	Event Info
Vampire	4	Expected: 0 Actual:	Blue Men	9	3	12/12	
Blue Men	2	Expected: 0 Actual: 0	Vampire				**Vampire Charmed**
Vampire	9	Expected: 0 Actual: 0	Blue Men	14	3	12/12	
Blue Men	18	Expected: 11 Actual: 11	Vampire	6	1	18/7	
Final Round 10: Blue Men	10	Expected: 5 Actual: 5	Vampire	4	1	1/0	**Vampire Died**

Vampire Charm – Charm worked correctly and allowed vampire to avoid attack approximately 50% of the time.

Vampire vs. Medusa (Test for Medusa Stare – Modified Attack) – Check!

Attacker	Attack Roll	Damage Inflicted	Defender	Defense Roll	Armor	Strength (Start/After)	Event Info
Vampire	7	Expected: 0 Actual: 0	Medusa	6	3	8/8	
Medusa	6	Expected: 0 Actual: 0	Vampire	6	1	18/18	

Vampire	4	Expected: 0 Actual: 0	Medusa	5	3	8/8	
Medusa	3	Expected: Charmed Actual: Charmed	Vampire	0	1	18/18	
Final Round 8: Medusa	12	Expected: Stone Actual: Stone	Vampire	N/A	1	14/0	**Vampire Died – Turned to stone**

Vampire Charm – Overrides Medusa's glare as expected in a second test.
Medusa Glare – Works on vampire. Must test on a base character defense type.
Medusa Glare vs. Harry Potter – First time does not kill Harry Potter.

```
Medusa rolled 2 dice for an attack of 12.
Medusa attempted to use glare on opponent.

Harry Potter is turned to stone.
Harry Potter took a deadly blow but he is back to life at double
strength!
```

Strength is back to 20. Second time glare is used on Harry Potter, he dies for good.
Base character defense function works with medusa as well as Harry Potter Hogwarts ability.

Harry Potter vs Harry Potter – Hogwarts ability works as expected.
Harry Potter vs Blue Men – Hogwarts ability and dice reduction works as expected.

REFLECTION

I ended up doing a major overhaul of this design. I realized as I was creating the program that I was making things a lot harder for myself by having too many things in a single class. Having a single class that held both the Queue and the FantasyGame functionality forced me to pass pointers of pointers and other such things from function to function instead of simply having a Queue object that could manipulate/maintain the queue itself for each team and the losers pile. FantasyGame class was thus split into two classes, one being for the Queue linked list itself and the functions allowing for manipulation of the Queue and the rest remained in the FantasyGame class. This allowed for a much simpler process where I did not have to have multiple pointers passing addresses to multiple functions.

In addition, I made a design decision to maintain the roll outputs for each individual battle as in Project 3 as TA input on Piazza stated either output format was acceptable for Project 4. Although it was not necessary, I felt the extra output was beneficial for testing and for user engagement. I also made the mistake of including outputs within the attack/defense functions in Project 3 in each character class. Although this kept everything contained in the attack/defense functions and it worked well for Project 3, having the output and the

calculations together in the functions made things difficult to change or make multiple output types. I was hoping to give the option of a reduced and a complete attack/defense output but doing so required an overhaul and retesting of all the Project 3 functionality on top of Project 4. This is a huge learning experience for me in the benefits of modularization. Although nothing specifically was wrong with what I did, it certainly helps to split functionality and output up in case you wish to make changes later. If output and functionality such as calculations are directly connected, even small changes can require a large overhaul. In the future, if I was to create this program again, I would start with my contributions in Project 3 and move output outside of the attack/defense functions and allow for calculations to occur independently of program output.
NOTE ON ABOVE: I went ahead and made additional changes to my attack/defense codes in order to allow for a menu option that allows the user at the start of each tournament to select for detailed or simple outputs. I accomplished this but creating a menu to allow for user selection, validating the integer selection entered by user, and then passed that selection to attack display functions. Attack display functions then passed the outputSelection of 1 or 2 (2 for detailed, 1 for simple) to the attack/defense character functions to indicate whether detailed output should be printed or ignored from within each function. I then re-tested the character functions with detailed output and debugging and made sure that special abilities and other functionalities in the program were not negatively impacted.
I made a design decision for detailed output to not include fighter name/team number for every single round of battle. The output is cluttered enough without adding 3-4 extra lines of output per battle. I felt the basic detailed output (which was not required) in addition to the end of fight output with winner/loser details was enough and met all aspects.

One major issue I had while making changes was working with passing the Queue object through functions. I spent hours trying to figure out why I kept getting read/write errors (even though the program would run fine in my IDE). It turns out that the major issue was created because I was not passing the Queue object by reference. Thus, a copy of the Queue was being passed and each time the function ended, the Queue destructor was called and essentially created a cascade of issues that were extremely hard to pinpoint. Moving forward, I want to keep better track of what I am passing as a parameter to functions when they involve major working parts of the program such as a Queue object or other class object.