Peter Moldenhauer
moldenhp@oregonstate.edu
11/13/16

CS 162 Assignment 4 Reflection

Program Requirements:

- Create a tournament between two teams (lineups) of creatures using last assignment's creature classes
- Have user enter specific input
  - The number of fighters both players will use
  - Team names
  - The type of creatures for each team (lineup)
- Have two creatures fight at a time for each round
  - The winning creature gets put at the back of its team's lineup
    - Lineup order cannot be changed once players are entered and re-added
  - The losing creature gets put into the container for the creatures who lost
    - Last loser should be at the top and the first loser will be at bottom
  - The winning creature will have some strength points restored before being put back into the team lineup (use polymorphism for this!)
  - Maintain a score between the two teams based on the win/loss of each round
- After each round, display which type of creatures fought and which won
- At the end of the tournament, display the final total points for each team and which team won the tournament.
  - Make sure to handle ties somehow
- Provide an option to display the contents of the loser pile

Design Plan:

The goal of this project is to take the creature hierarchy from Assignment 3 and transform it into a program that pits two user-selected teams of creature objects against each other, making use of the linked-list implementations of stacks and queues. The gameplay is to be divided up into individual matchups between two creatures from opposite teams. In each round, the types of creatures fighting and the winner is to be displayed. After a winner is determined, the winner is placed at the back of the lineup to continue fighting and the loser is sent to a loser pile and removed from the field of battle. The teams will be awarded points for each round victory. When the tournament is over, the list of the creatures in the losing pile will be available to be displayed to the user.

In order to produce this scenario, I will make use of the creature class and its five derived classes, a stack class and queue class, as well as a run combat class that will run the tournament sequence. The stack class will hold the creatures who lose throughout the tournament and the queue class will hold each team's lineup of creatures to fight in the tournament. The run combat class will not only run the tournament sequence but it will also keep a running total of each team’s points, place each creature in the correct container after each round (queue or stack), print the results of the tournament and display all of the creatures who lost in battle throughout the tournament. The only new modifications to the original creature hierarchy will be in the implementation of the revive function and the ability to store a team name to a particular creature object. The new revive function that will be added will allow (if certain conditions are met) for a partial increase in strength points to a winning creature before being added back in the team’s lineup.

The winning team is considered to be the one who has at least one creature standing after repeated battle cycles. Both teams will have a total score, determined by adding up each team’s scores after each round at the end of the tournament. If by chance there is a tie game (scores are equal) at the end of the tournament, then the winner of the tournament will be determined by a simulated coin flip. The first team will “call” the coin flip and based on the flip, the winning team will be determined.

<u>Description of classes:</u>

Creature class:
- Changes to protected members:
    - teamName
- Changes to functions
    - Virtual revive function

Vampire class:
- Changes to functions:
    - teamName initialized in setData function
    - Revive function defined uniquely to class

Barbarian class:
- Changes to functions:
    - teamName initialized in setData function
    - Revive function defined uniquely to class

BlueMan class:
- Changes to functions:
    - teamName initialized in setData function
    - Revive function defined uniquely to class

Medusa class:
- Changes to functions:
    - teamName initialized in setData function
    - Revive function defined uniquely to class

HarryPotter class:
- Changes to functions:

        - teamName initialized in setData function
        - Revive function defined uniquely to class

Stack and Queue classes:
- No changes to Lab 6 code aside from changing all "int" variables to "creature*" variables

RunCombat class:
- Private members:
    - Team1Name and team2Name (strings)
    - P1Won, and p2won (bools)
    - *side1 and *side2 (Queue's)
    - Side1pts, side2pts and numRounds (ints)
    - *losers (Stack)
- Public functions
    - Constructor that has each team's creature lineups and team names as parameters
    - Destructor that frees up memory allocated for stack
    - playMatch that runs the tournament between both teams
    - finalScore displays the final results of the tournament
    - printLosers displays the losing creatures throughout the tournament

Main function:
- Display program title and instructions to user
- Get user input
    - Team names
    - Number of creatures on each team
    - Starting lineup of creatures for each team
- Store user selected creature lineup's in queues for each team
- Create and call RunCombat class object
    - Run tournament
    - Display results
    - Display losing creatures

Testing Plan:

As a result of using the creature class, its derived classes, and the stack and queue classes from the previous assignment and lab, I will focus the majority of my unit testing on the main.cpp file and the RunCombat class. With respect to the classes being pulled in from Assignment 3, I need to ensure that the teamNames variables and the revive function works as expected.

I plan to program the functions in main.cpp first so that I will have a basic skeleton of the program flow down. The points that I will test most rigorously are the user inputs for the types of creatures in each player's lineup. I don't believe I will have to test the user input for the size of each lineup because I will be using auxiliary functions written for previous assignments and stored in the “functions.h and functions.cpp” files to filter out invalid values that the user may possibly input. I will also have to make sure that the creatures being entered in for each team’s lineup is added and removed in the lineups in correct order. Since the lineup’s will be a queue like container, the first creature in the lineup will be the first to be taken out of the lineup. I am expecting very few errors in this part to do the fact that I am reusing my code from the Lab 6 program.

The bulk of the changes in this assignment will take place in the RunCombat class. Since I was certain that the code for individual battles between two creatures worked fine (designed for Assignment 3) I would simply have to expand this code into incorporating multiple battles among many more creatures based on the user input of each team’s lineup. Throughout the process of building the function to run the tournament, I will use incremental development to test my code as I build it. I will also employ print statements during this process to keep track of the scores of each round, the wins/losses of each round, the correct creatures are lined up to fight each other at the right time and the right creatures are placed in the correct “containers” (queue or stack) after each matchup.

The finalScore function in the RunCombat class will have to display the correct final total points for each team as well as declare the winner of the tournament. I will have to manually record the points gained for each team for each battle in a given tournament and make sure that my calculated point values match up with my program’s generated point values. Even if the points do match up, I will still have to look at each round in a given tournament to make sure that the points are being calculated correctly. In addition, I have to make sure that the team that the program declares as the winner is indeed the winner of the tournament. To test and confirm that my program is running as it should, I need to run numerous tests using numerous different creature matchups to see consistency among how the scores and tournament winners are calculated.

Lastly, I need to make sure that the function in the RunCombat class which displays the list of creatures who lost not only display the correct creatures but displays them in the proper order. Since these creatures will be contained in a stack like structure, the first creature entered in this list of losing creatures should be the last creature outputted in the list. I will need to keep track of which creatures started in what order for each team and then follow the tournament process to keep track of the order in which each losing creature is added to the list. In the end I need to make sure that this list of losing creatures to be displayed is correct and in the correct order. Similar to the queue like container for the lineup’s, I am expecting very few errors in the stack container to do the fact that I am reusing my code from the Lab 6 program.

Testing Results:

Overall, the testing process of my final program went fairly smoothly. I can most likely attribute this to the fact that I used the tactic of incremental development throughout my program as well as reusing code from previous assignments that have already been tested to work properly (code from assignment 3 and lab 6). One of the ways that I tested the revive function in each creature subclass was to completely remove the function from a tournament and compare it with running a tournament with the same exact lineup of creatures. I did not address this in my initial testing plan but realized through the course of testing other parts of the program that this would be a good tactic to use. The tournaments that had the revive function removed changed the outcomes of the tournament compared to running tournaments with the revive function included. The round/tournament results generated from my program matched up consistently with my manual calculations as I ran through multiple tournaments with different creature matchups each time. There were a few spelling and output formatting errors that I needed to correct but the actual calculations were spot on. One thing that came up in the testing phase which wasn't necessarily an error, was keeping track of the same creatures fighting each other each on different teams. I needed to keep track of which specific team's creature was the winner or loser. To solve this, I added the team name in parenthesis behind each creature name so it was clear as to what creature was on what team.

I ran numerous tournaments over and over again to check the calculations as well as the revive function, the creatures being placed in the correct "containers", correct creatures fought each other in the correct order, correct final tournament results were displayed and the correct creatures were displayed in order from the losing creature "container". The tests and results of these tests can be seen in the chart below:

Test Description	**Expected outcome**	**Observed outcome**
Creatures are added to lineup queue properly and removed from queue in correct order	The first creature added to a team's lineup queue will be the first creature removed	As expected
Creatures are added to the loser stack properly and removed from the stack in correct order at end of tournament	The first creature that loses in tournament will be the last creature that will be displayed in the final "loser creature list"	As expected
Compare the allowable creatures to be entered for each team to the number of creatures that the user enters at the start of the program	The user should only be allowed to enter the number of creatures in the lineup for a team that is the same number that is entered at the start of the program.	As expected
Compare the user entered team names throughout the outputted team names in the tournament	The user entered team names should align with the correct lineups, scores and results throughout the tournament	As expected
The tournament continues for a correct number or rounds	The tournament lasts at minimum the number or rounds that there are creatures on each team (Ex. two creatures on each team, the tournament last for at least two rounds)	As expected

Point values are calculated correctly	A team receives 1 point for each win in a given round and 0 points for a loss in a given round (each round there is 1 winner and 1 loser).	As expected
Revive function works properly	If a creature wins a round, that creature will have a slight increase in strength points when it is added back into the lineup	As expected
Revive function has an effect on the tournament gameplay	Using the revive function (versus not using it) in a tournament causes the length of the tournament to increase	As expected
Winner of tournament is correct	The team with the highest score from the rounds will be declared as the winner	As expected
“Stronger” creatures won more often than “weak” creatures	Teams that we made up of Blue Men and Harry Potter will win more often than teams made up of Vampire and Medusa	As expected

To sum up the testing phase in this assignment, I must point out that the print statements in the playMatch function did help to keep track of the calculations and flow of a given tournament. The “gameplay” of each tournament flows and functions well. As expected, the more BlueMen creatures that a given team has, the more likely it will win (along with HarryPotter). In fact, teams with BlueMen or HarryPotter win every time when matched up with teams without these two creatures. One thing to keep in mind throughout this testing and gameplay is that player 1’s team always attacks first. This gives this team the advantage even when both teams have identical lineup’s. The following chart shows some of the sample tournaments that I ran and the outcomes of the tournaments:

Team 1	**Team 2**	**Team 1 total points**	**Team 2 total points**	**Last one standing**	**Expected Winner**
V,V,V,V,V	B,B,B,B,B	5	2	V	V
V,V,V,V,V	BM,BM,BM,BM, BM	4	5	BM	BM
V,V,V,V,V	M,M,M,M,M	5	0	V	V
V,V,V,V,V	HP,HP,HP,HP,HP	0	5	HP	HP
B,B,B,B,B	BM,BM,BM,BM, BM	0	5	BM	BM
B,B,B,B,B	M,M,M,M,M	5	1	B	B
B,B,B,B,B	HP,HP,HP,HP,HP	0	5	HP	HP

BM,BM,BM,BM, BM	M,M,M,M,M	5	0	BM	BM
BM,BM,BM,BM, BM	HP,HP,HP,HP,HP	5	0	BM	BM
M,M,M,M,M	HP,HP,HP,HP,HP	0	5	HP	HP
BM,BM,BM,BM, BM	BM,BM,BM,BM, BM	5	4	BM	BM
HP,HP,HP,HP,HP	HP,HP,HP,HP,HP	5	4	HP	HP
BM,HP,BM,HP,BM	M,B,M,B,M	5	0	BM	BM

Problems encountered and how they were resolved:

Overall, I did not run into too many problems throughout the process of building this program. As noted earlier, one of my issues that I had to deal with was keeping track of which creature belonged to which team if both teams had the same creature fighting. This was solved by adding in the team names next to each creature that was displayed. Another problem that I had to deal with was how to figure out how to determine if a queue or stack was empty. For example, if a lineup queue was empty then it means that there are no more available creatures in the lineup and another battle round should not take place. I ultimately decided to tackle this by each time I removed a creature object from the queue, I tested to see if it was equal to NULL. If it was equal to NULL then this meant that the queue was empty and I should not proceed with another round. I applied the same logic to the stack for the list of creatures that lost throughout the tournament. Lastly, I had a problem when I tested my final program on the school server (FLIP). When I did this, my program would not compile due to a “variable array length error”. I normally write my code up in an IDE on a windows computer and then transfer it to FLIP to test. Even though my program ran fine in the IDE, it would not compile due to this error I was getting on FLIP. I looked up this error message on the stack overflow website and soon found the problem. Initially I was creating an array of pointers for the lineup for each team. I would then use the array of pointers to create the lineup as a queue container. However, from what I found on stack overflow, I needed to dynamically create these arrays to hold the pointers to creature objects to eventually be made into the lineup queues. I therefore had to dynamically create a double pointer for each array – a pointer to the array that held pointers to creature objects.

Final Thoughts:

This assignment showed me the importance of having well written code that could be reused. It probably would have taken twice as long (both in design and testing) if I hadn’t already nicely compartmentalized and tested well documented code. This assignment also provided practical uses for the queue and stack containers. I now have a better sense for the things I might use a queue for and things I might use a stack for. All in all, I thought this was a good assignment to learn from and it was fun to implement.