Christopher Merrill
CS162
11/8/2016

Assignment 4 Reflection Document
(Design, Test Plan, Test Results and Reflection)

DESIGN OVERVIEW

I plan on utilizing my creature classes from assignment 3, along with my Combat class (which ran the game logic for a head-to-head battle). I will also use my queue class from Lab 6 to handle the team lineups, and my stack class from Lab 6 for the loser pile. I will then create a tournament class which will handle the following: populating the team lineups, pairing up creatures in head-to- head battles, updating the lineups/loser pile, and determining the winner.

Note: My heal method is within the Combat class. My "Hero" class is the creature class, I didn't realize they were referred to as creatures when I completed assignment 3. (So when you see reference to Hero in my code, know that I meant Creature. Sorry for any confusion!)

PSEUDOCODE

MAIN.CPP:

Display the main menu with options to play or exit
If the user decides to play
    Ask the user how many heros should be on each team
    Create a new Tournament class object
    Run the Tournament class play function
    Ask the user if they want to play again

TOURNAMENT CLASS:

Data Members:

- The number of heros in each lineup
- Team A's points
- Team B's points
- Team A's lineup (queue-like structure)
- Team B's lineup (queue-like structure)
- Loser Pile (stack-like structure)

Member Functions:

- A method to populate each team's lineup
- A method to run the main game loop
- A method to determine the winner

Populate Line-Ups Function:

Display a menu that lists the creature types
Ask the user to enter items from the list to populate team A's lineup
The user will enter integer values which correspond to the 5 creature types
When a creature is selected, it will be pushed into team A's lineup (queue)
Ask the user to enter items from the list to populate team B's lineup
The user will enter integer values which correspond to the 5 creature types
When a creature is selected, it will be pushed into team B's lineup (queue)

Play Function:

Display the starting line-ups
Start the game loop:
Grab the first creatures in each team's queue
Create a new Combat class object, using the first creatures in each team's lineup
Run the Combat play function
If Team A's creature won
Push the winning creature to the back of Team A's lineup
Pop the winning creature from the front of Team A's lineup
Push the losing creature to the loser pile
Pop the losing creature from the front of Team B's line-up
Give team A one point
If Team B's creature won
Push the winning creature to the back of Team B's line-up
Pop the winning creature from the front of Team B's line-up
Push the losing creature to the loser pile
Pop the losing creature from the front of Team A's lineup
Give team B one point
(Repeat the above code until one team runs out of creatures)
Declare the winner of the tournament and display the results
Ask the user if they would like to see the contents of the loser pile
If the user wants to see the loser pile
Clear the contents of the loser pile using "pop", and display the name as the creatures pop
Clean up memory by deleting any remaining creatures

Determine the Winner Function:

If Team A has more points than team B
Declare Team A the winner and display the final scores
If Team B has more points than team A
Declare Team B the winner and display the final scores

TEST PLAN and TEST RESULTS:

Test	Results	Comments
Input Validation: Check that the input validation works correctly in main.cpp and in Tournament.cpp by entering the following each time the program asks for an int: -1, 0, 100, F, @	**-1:** The program asks for a nonnegative value. **0:** The program asks for an integer greater than zero. **100:** The program asks for an integer within the range of the particular menu function. **F:** The program asks for an integer value. **@:** The program asks for an integer value.	No changes necessary.
Starting Lineup: Run the program with 1, 5, and 10 creature on each team of varying creatures and verify that the starting lineup is displayed correctly and in the correct order.	**1 Creature:** The program correctly displays each team's respective creature. **5 Creatures:** The program correctly displays each team's respective creatures in the correct order. **10 Creatures:** The program correctly displays each team's respective creatures in the correct order.	No changes necessary.
Updated Lineup: Add cout statements at the end of each round that display the updated lineup and loser pile after each round. This is to verify following: The winner is moved to the back of the lineup, the loser is removed from the	The winning creature was successfully moved to the end of its team's line up. The losing creature was successfully removed from its team's lineup. The losing creature was successfully added to the top of the loser pile.	No changes necessary.

lineup, the loser is added to the loser pile.		
Correct Winner: Run three tournaments and follow the rounds to assure that the final scores are correct and that the correct winner is declared.	In all three runs, the scores were correct and the correct winner was declared.	No changes necessary.
Heal Function: Verify that the healing process is working correctly by adding cout statements before and after each heal that show the battling creature's strength points.	The heal function is working correctly. The creatures are healed the correct amount of strength after each round, and when a creature fights a second time its starting strength is where it left off in the previous round.	No changes necessary.
Loser Pile Print: Play through three tournaments and verify that the loser pile is displayed in the correct order at the end of each tournament. (The loser pile is a stack-structure and the contents are displayed by popping each creature out of the pile, which means it should be displayed in reverse order.)	The loser pile was successfully printed in the correct order after each of the three tournaments. Also the creatures were displayed with their correct team names.	No changes necessary.

***Note:** The logic that controls the die rolling and creature powers in each round was already thoroughly tested in Assignment 3, so I excluded any die rolling tests from this test plan.

REFLECTION:

The additions that were added to this project after Assignment 3 were as followed: the inclusion of stack and queue structures to handle the team lineups and loser pile, a tournament class was added to handle the mechanics of the tournament structure, and an updated main.cpp file to

drive the program. For the most part the logic that controlled the head-to-head battles was left alone, and the creature classes were left entirely alone.

I did change the Combat class from assignment 3 by removing all of the *cout* statements that displayed the results of each die roll. This was to make the results of the overall tournament more clear in the display. I also updated the Combat constructor so that it took two pointers-to-heros rather than two integer values, so that I could pull the pointer-to-heros straight from the front of each teams lineup to initiate a round. This meant that I also had to update my Queue class from Lab 6 by adding a “getHead” function that returns the current head of a team’s lineup. The only other addition I made to the Combat class was the addition of a heal function, so that I could easily heal the winner of each round.

Aside from these changes (which were all to previously created aspects of the assignment), I didn’t have to make any major changes to my Tournament class or main.cpp file. For the most part I stuck to the design plan above. The one thing I did have to do was add two data members (aElements and bElements) that held the number of elements in each team’s lineup at the start of the tournament. This allowed me to sort through the team lineups more easily in my play function.

Overall I found my program to be successful with only minor changes to my original design. Luckily I was able to salvage the majority of the work done in Assignment 3 and Lab 6 with only slight modifications, which really drove in the importance of writing code that can be reused in future projects!