Assignment 4 Bryan Konshak

Design Description:

All of my subclasses are working properly and have been exhaustively tested from assignment three. My driver program accurately allowed two creatures to fight and declare a winner. In assignment four I must take that contest and create a tournament using the queues and stacks from the previous lab.

I will need to ask the user for the names of the teams, using getline to account for spaces in user input. These names will be assigned to strings teamNameOne and teamNameTwo.

I will need to ask the user how many creatures are on each team.

I will first create arrays of pointers to creatures for the teams playerOne and playerTwo. This will be used to load the queues of creatures for the tournament. I will then use a menu that will iterate based on how many creatures are on each team using a do while menu loop, allowing user to select their team.

For each creature selected, my program will have the creature point to the selected subclass object—either barbarian, bluemen, etc.

I will create Queues (from prior assignment) for teamOne and teamTwo. I will then use the push function to add the creature to the end of the queue. I will also then display all the creatures on each team.

At this point I will have two queues with the creatures selected for each team and they will then need to fight each other.

I will have a battle function that plays the game, but first I will use two creature pointers, creatureOne and CreatureTwo, and assign them using the pop function in queue. This will pit the first two creatures in the queue against each other.

When a creature loses, it is added to my stack called losers, and the stack is displayed at the end of the game.

The battle() function has seven paramaters:

creatureOne, creatureTwo, Queue teamOne, Queue teamTwo, Stack losers, string teamNameOne, and string teamNameTwo.

It will have int variables wins1 and wins2 that tracks the total rounds won by each player. I will need to add a function to queue that indicates whether or not the queue is empty: getEmpty(). If both teamOne.getEmpty() and teamTwo.getEmpty is false, a while loop will run the battle again.

I will then need a nested while loop that is very similar to my driver function from assignment 3. This while loops if creature one's strength is not 0 and if creature two's strength is not 0. It loops until a creature dies, and it does not allow for ties.

A round of fighting takes place if neither creature's strength = 0. Player one attacks and the attack roll is displayed. Then player two's defense role is displayed. The damage inflicted is displayed via each creature's defense function, then the player's strength is displayed. Then the same happens for Player 2's attack.

If either creatureOne or creatureTwo's strength = 0, then the round will be over. Either wins1 or wins2 will be incremented and shown. The program will display the winner of the round. The losing creature will be added to the stack using push. The winning creature then gets a 1.5 multiplier boost using polymorphism in the creature class. The winning creature is then pushed to the end of the team queue. It will display the lineups of the teams afterwards.

After all results are tallied, teamOne.pop() and teamTwo.pop() are assigned to creatureOne and Two again. If either queue is empty, the while loop will end because that is a condition of the while loop and the battle royale is over.

The program will display the final tally and declare the winner, and show all the losing creatures using the stack displayList function.

TEST PLAN:

I've included the test plan from the previous assignment at the end of this document. This assured the creatures functioned correctly.

Test that names are entered as strings and spaces are allowed	FAILED then PASSED	Incorrectly used cin.ignore..
Test that array is populated correctly with creature subclass objects by cout getName function in for loop	PASSED	
Verify creatures are listed in correct order—First in first out in Queue and use displayList to show user.	PASSED	
Verify correct number of objects are created using 1, 3, and 10 for number of creatures	PASSED	
Verify game end when all items are removed from team Queue.	FAILED then PASSED	Initially getEmpty function was not initiating correct bool and indefinite loop
All one on one fighting situations tested below	PASSED	
Verify loser stack is first in last out and display, first loser listed last	PASSED	
Verify correct winner displayed	PASSED	

Verify wins1 and wins2 increments correctly	PASSED	

When game ends, wins from each team are displayed, and team with largest win total wins the contests	PASSED	
Test one creature, three, and five, verify winning team has one, three and five total wins	PASSED	
Test all combinations of fighters, one of each, all of one, and run multiple battles to verify totals are what would typically be expected (same character 50/50 split)	PASSED	

Reflections.

I really enjoyed this assignment. It tied together two key concepts—linked lists and polymorphism. I found the programs I used for stacks and queues were easily converted to creature pointers, I just had to change the data type. I had to figure out a way to end the game, and my solution was in my Queue.pop function, when the head = NULL the list was empty. I had to create a bool variable that would switch to true when the queue was empty. Then I could create a getEmpty function and include it in my while loop to run the game. As soon as the Empty variable switched to false, the loop ends and the winner was declared. I've struggled with cin and strings, and freezing up my program with spaces. I couldn't completely solve that here so I just had the teams declared before determining the size of the teams. That made more sense anyway.

I credit my driver program in the last assignment for helping me focus on the key concepts of this one. Creating the teams weren't difficult because I just had to add the for loop--I already had a way of selecting creatures to fight each other. I just had to expand that into a team. My stack and queue worked great for int values, and changing everything to pointers was a breeze. This assured everything worked correctly for the adding and removing of objects from the stack and queues.

Regaining the strength was easy to implement as I just took the winning creature's remaining strength and multiplied it by 1.5. Sometimes the result was neglible or nothing at all due to decimals rounding off of ints but I was fine with that from a design perspective.

I did initially have the user control each round but per the instructions the user shouldn't have to do anything after the initial selections, so I just changed it to a usleep and clear screen set up so you can see the results in realtime. I did like the user controlled setup better though, so I just commented that out.

I think my testing plan was thorough and accounted for any potential errors. All my simulations produced expected results once I figured out the getline for the strings and the Empty variable in my queue class.

Previous assignment's tests:

Die class has been tested and verified.

Verify sum of two attack roles returns the attack—passed

Verify defense calculates remaining strength correctly, accounting for attack, defense, and armor—passed.

Verify no negative values pass to strength via the defend function--passed

Vampire class—cout test to make sure glare is working and not changing strength when true—passed

Medusa class—test repeatedly to make sure a role of 12 ends the game with victory—passed

BlueMen class—cout all three dice roles when strength is > 8—passed 2 for > 4 passed 1 for > 0 passed

HarryPotter class—bool changes to false after first strength = 0 then strength = 20—passed

Test all classes against each other and verify approximately 50/50 spit in wins over 1000 iterations—passed

Vampire, BlueMen, and Harry Potter almost always beats barbarian. Barbarian almost always beats Medusa unless stoned.

Blue Men beats everyone almost always and their games against each other last a long time

Harry Potter battles relatively equally with vampire but is slightly stronger.

Test menu to create creature subclasses—failed. Needed to create new Creature objects then use menu to select creatures and assign to subclasses.