Jason Goldfine-Middleton
932675265

CS 162 Assignment 4 Reflection

Design Plan:

The goal of this project is to take the Creature hierarchy from Assignment 3 and mesh it with a client 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. Each round the types of Creatures fighting and the winner are 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. Players will be awarded points for each victory. Additionally, Goblins will receive double points while Blue Men will receive half points for each win. The three Creatures who earn the most points will be printed out along with all the creatures that were KOed after the tournament ends.

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 CombatRunner client class that will run the tournament sequence, keep track of which Creatures died and in what order, and print battle results. The only modifications to the original Creature hierarchy will be in the implementation of the revive() member function, a points earned private data member, and a new inherited function to allow adjustment of the points earned.

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 Creature's score at the end of the tournament. As the notion of "best finishers" is hazy, I have decided to make the term correspond to the last Creatures standing. This will be determined by which three Creatures were put into the losers pile last. Note that this works because in my implementation, all Creatures will ultimately end up in the losers pile, even if they never were knocked out.

Creature class:
Changes to protected members:

- int points

Changes to public members:

- Constructor
    - move srand() to the CombatRunner client constructor
    - initialize points to 0
- int get_points()
    - return points

Goblin class:
Changes to public members:

- virtual void revive()

- restore strength to 8
- add 4 points

Barbarian class:
Changes to public members:

- virtual void revive()
  - restore a random amount of strength between 3 and 9 or max strength out
  - add 2 points

Reptile class:
Changes to public members:

- virtual void revive()
  - restore a random amount of strength between 6 and 12 or max strength out
  - add 2 points

BlueMen class:
Changes to public members:

- virtual void revive()
  - restore a random amount of strength between 2 and 6 or max strength out
  - add 1 point

Shadow class:
Changes to public members:

- virtual void revive()
  - restore a random amount of strength between 2 and 8 or max strength out
  - add 2 points

Stack and Queue classes:
No changes to Lab 6 code aside from changing all “int” variables to “Creature*” variables

CombatRunner class:
Private members:

- Queue* side1
  - Contains pointers to player 1’s lineup
- Queue* side2
  - Contains pointers to player 2’s lineup
- Stack* losers1
  - Pile of player 1 pointers to Creatures who were KOed
- Stack* losers2
  - Pile of player 2 pointers to Creatures who were KOed
- Stack* losers
  - Pile of all pointers to Creatures who were KOed
- int cur_sim

    - current matchup number
- int side1pts
    - sum of all points for player 1's Creatures
- int side2pts
    - sum of all points for player 2's Creatures
- static const bool P1_WON = true
- static const bool P2_WON = false
- vector<int> defeat_sides
    - each time a Creature dies, either a 1 or 2 is pushed back, depending on which team he was on

Public members:

- CombatRunner(Queue* s1, Queue* s2)
    - get random seed for Creatures
    - side1 = s1
    - side2 = s2
    - losers1 = new Stack
    - losers2 = new Stack
    - losers = new Stack
    - cur_sim = side1pts = side2pts = 0
- ~CombatRunner()
    - delete losers1, losers2, and losers
- bool run_tournament()
    - Creature *p1
    - Creature *p2
    - while (true)
        - If (p1 = side1->remove() && p1 == NULL)
            - while (p2 = side2->remove() && p2 != NULL)
                - losers2->add(p2)
                - losers->add(p2)
                - side2pts += p2->get_points()
                - defeat_sides.push_back(2);
            - return P2_WON
        - else if (p2 = side2->remove() && p2 == NULL)
            - while (p1 = side1->remove() && p1 != NULL)
                - losers1->add(p1)
                - losers->add(p1)
                - side1pts += p1->get_points()
                - defeat_sides.push_back(1);
            - return P1_WON
        - else // still fighters left from both sides
            - bool result = play_match(p1, p2)
            - if (result == P1_WON)

        - revive p1
        - side1->add(p1)
        - losers2->add(p2)
        - losers->add(p2)
        - side2pts += p2->get_points()
        - defeat_sides.push_back(2);
    - else
        - revive p2
        - side2->add(p2)
        - losers1->add(p1)
        - losers->add(p1)
        - side1pts += p1->get_points()
        - defeat_sides.push_back(1);
- bool play_match(Creature* p1, Creature* p2)
    - while (true)
        - call attack() for p1
        - call defend() for p2
        - call take_damage() for p2
            - double damage if p2 is Goblin and cut opponent's achilles but opponent is also Goblin
            - if p2->is_dead()
                - return P1_WON
        - call attack() for p2
        - call defend() for p1
        - call take_damage() for p1
            - double damage if p1 is Goblin and cut opponent's achilles but opponent is also Goblin
            - if p1->is_dead()
                - return P2_WON
- void print_outcome()
    - bool winner = run_tournament()
    - if (winner == P1_WON)
        - print player 1's team defeated the opposition
    - else
        - print player 2's team defeated the opposition
    - print total points for each player
    - print_top_three()
- void print_top_three()
    - print last three standing in order first through third from the losers stack
    - print which team each was on by looking at last three values in defeat_sides

<u>main():</u>

- Create two queues
- Ask how many Creatures per team
    - Must be at least 2
- Prompt user for team 1
    - Loop getting chars
        - vector1.push_back(make_creature(char))
- Prompt user for team 2
    - Loop as above but use vector2 to populate second queue
- Create new CombatRunner object and pass in queues
- Run tournament and print results
- Free all memory held by pointers in the two vectors
- Exit

Creature* make_creature(char c):

- switch(c)
    - g = Goblin
    - b = Barbarian
    - r = Reptile
    - m = BlueMen
    - s = Shadow
- return new Creature pointer to derived object

Testing Plan:

As a result of using the Creature class, its derived classes, and the Stack and Queue classes from Lab 6 almost unmodified, I will focus the majority of my unit testing on the main.cpp file and the CombatRunner class. With respect to the classes being pulled in from Assignment 3, I need to ensure that the Creature::get_points() 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 "utilities" namespace to filter out invalid values.

Otherwise, the bulk of the changes were in the CombatRunner class. Since I simply renamed a working function from Assignment 3 to CombatRunner::play_match(), I was confident that the function already worked correctly. In the CombatRunner::print_top_three() function, I anticipate that some care will be needed to ensure the vector bounds are respected. I will test this by creating an analogous standalone function and ensuring that the structural logic is correct before throwing it over the fire in the class. The biggest function is

CombatRunner::run_tournament(). As this function is so big and intricate, I will employ print statements during the scaffolding process.

Testing Results:

It appears that I underestimated the importance of unit testing the get_points() function because, by saving it until last, I ended up struggling to understand why I was getting point totals in the hundred thousands. Since the get_points() function simply returns the value stored in the protected "points" member variable, it hit me that I had forgotten to initialize "points" in the Creature constructor, leading me to uninitialized garbageville. ☺

In main(), I had some issues when trying to ensure that user selection of Creature types worked correctly. Unfortunately, the issue was twofold. In my initial implementation I had forgotten to reset the value of char "c" after each std::cin.get(). This led to my while-loop conditions failing immediately, preventing the user from entering more values. The second problem I discovered was that I was unnecessarily clearing the buffer after each character was read. This caused an issue when I tested inputting all the characters as a single string, pressing ENTER only after all the characters were on-screen. After fixing these issues, the results were as shown below.

What tested?	Which input values?	Expected Results	Actual Results
Just pressing ENTER	ENTER	Wait for rest of chars	Wait for rest of chars
Entering with spaces	s m r b g ENTER	Works	Works
Entering with no spaces	mmrbg ENTER	Not sure	Works
Entering invalid chars	n x 9 SPACE 5 y	Wait for correct chars	Wait for correct chars
Entering too few chars	s b g ENTER	Wait for rest of chars	Wait for rest of chars
Entering invalid chars with no spaces	xxxop	Wait for correct chars	Wait for correct chars

I also took advantage of the command line tool "valgrind" to verify that all heap memory was correctly deallocated by the end of program execution. There were no issues. Specifically, on one typical run valgrind's output included the following:

==18638== HEAP SUMMARY:
==18638== in use at exit: 0 bytes in 0 blocks

Jason Goldfine-Middleton
932675265

==18638== total heap usage: 72 allocs, 72 frees, 1,752 bytes allocated

One of the first things I did when beginning work on the CombatRunner class was create an analogous standalone function to its member print_top_three(). This allowed me to ensure I didn't access any out-of-bound vector indices when looping. Once I was confident about that, I put the actual function in place.

While working on the constructor for the CombatRunner class, I discovered that it would be very difficult to have any member variables that were references. My original plan was for side1 and side2 to be references to Queues, but compiler errors when I tried to assign reference parameters to those reference members suggested I had better just use pointers instead. That was the largest change in my original design plan for this project.

To wrap things up, I will say that the print statements in the run_tournament() method didn't end up being too important, but I did use them to check each place where get_points() was called after seeing results like "Player 1: 4, Player 2: 345266". As I mentioned before, that was silly old me forgetting to initialize the "points" member variable in the Creature class.

The actual tournament functions well. As expected, the more Blue Men a team has, the more likely it will win. In fact, teams with Blue Men or The Shadows absolutely annihilate the opposition. One thing to keep in mind is that Player 1 always attacks first, giving him the advantage even when both teams have identical lineups. Below are some sample runs.

Player 1	**Player 2**	**Player 1 Total Points**	**Player 2 Total Points**	**Last One Standing**	**Expected Winner (LOS)**
gbgbgb	rmrmrm	0	9	P2	P2
ssssss	ssssss	10	4	P1	P1
ssssss	ssssss	8	6	P1	P1
rrrrrr	rrrrrr	10	12	P2	P1
bbbrrr	bbbbbb	10	6	P1	P1
bbbbbb	bbbbbb	8	8	P1	P1
bbbbbb	bbbbbb	6	12	P2	P1
gggggg	gggggg	16	16	P1	P1
mmmmmm	mmmmmm	2	6	P2	P1
gggmmm	brrrrb	5	10	P2	Not sure
gbrms	smrbg	7	7	P2	Not sure
smr	brm	2	2	P1	Not sure
ssrrmbgs	gggmggbb	12	5	P1	P1
ggbbrrmmss	rrbbggssmm	11	15	P1	P1