Terezie Schaller (Tera)
5-14-2016
Assignment 4
Creature Combat version 2

Requirements

- Demonstrate a queue and a stack.
- Combine code from assignment 3 and Module 2.
- Game becomes a 2 player game. Players select a size of game/number of matches.
- Each player selects a lineup of the chosen size.
- Player is able to name each creature.
- The chosen creatures are entered into the lineup (queue).
- Creatures battle against each other in a FIFO order.
- A winner of each battle is determined and a score awarded.
- Losers are stored in a FILO structure (stack).
- Tie matches are not allowed based on my implementation of assignment 3. When a creature runs out of health it is not able to counter attack.
- An overall tournament winner (Team/lineup) must be determined.
- The top three creatures must be determined.

Design

Design Considerations

**Grey statements were changed during implementation

Have the user selects a number of matches. This user number builds 2 queues (one for each player) of the selected size to hold the line-up. Each queue node should contain a pointer to creature object. The user number also initiates 2 stacks (one for each player) to hold the eventual losers. (It might be better practice to grow the stack as losers are added, but just making one of size X at the beginning seems easier to implement).

The line-up should be a queue with a data member that is a pointer to creature object, but is initially set to NULL. (Note: actual creatures cannot be instantiated)

As user enters their choices of creatures, the new creature is created and the pointer set to that creature.

Once both creature queues are full, the game begins and pits creatures against each other in FIFO order.

When a creature loses a match, a pointer from the loser stack is set to that creature. The loser creature's node in the queue is removed.

Winning creatures remain in the lineup/queue.

Losing creatures (pointers) are stored in the loser stack.

The tournament is over when each creature has battled once. (Simplest choice)

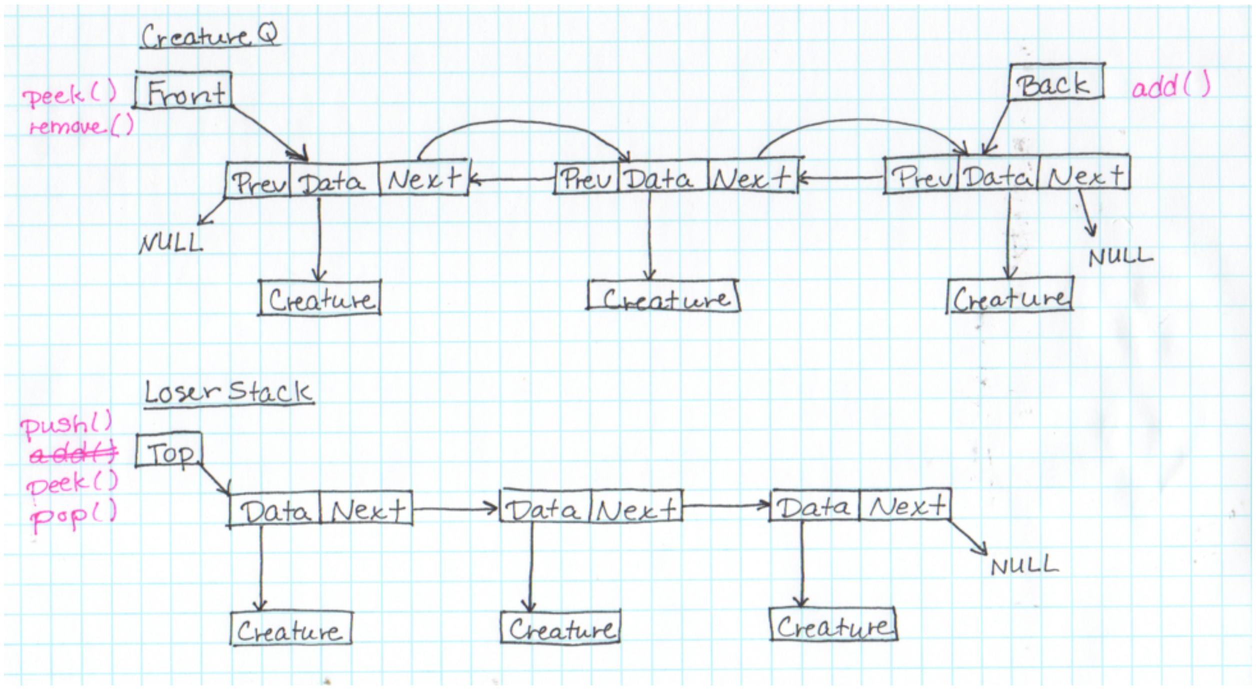
Healing effect: The winning creature of each round will have strength increases by a random number between 1 and 20. What doesn't kill us makes us stronger!

Tournament winner: Based on wins, losses. Ties are allowed if there are an even number of matches. Mention this to user in beginning of game.

Top finishers: The 3 winning creatures with the most health at the end of the tournament will be the top finishers. Write a function to step through both queues and look at health and record the creatures with the most health remaining. (This could end up being pretty tricky???) [Might end up using sorting? Sort the queue from the winning team by amount of health and declare the top 3 the winners?]

At the end of tournament, user has the option to view ranking of remaining players.

Diagram



Implementation stage 1

**Grey steps not implemented. Alternatives explained in Changes & Issues
1. Create and test a queue (circular linked list) of creature pointers.
2. Create and test a stack of creature pointers.
3. Make sure creatures can be transferred from queue to stack.
4. Make 2 queues of characters battle each other.
5. Implement heal functionality for winners. Heal() function in creature.
6. Make sure losing creatures are going into the loser stack.
7. Count the nodes in each queue to determine winning team. (Alternative: use a counter variable to count wins.)
8. Sort winning lineup queue by health.
9. Print out 3 creatures from winning team with most health.

Note: It is helpful to me to start coding, then pause and plan again. I wrote the logic for the lineup and the loser stack in Stage 1. Stage 2 involves the steps required to make something I can turn in.

Implementation stage 2

1. Compile and test early version on school server.
2. getHealth() function in creature
3. getName() and setName() function in creature
4. User prompts and menus in main
5. Switch statement to load user choices into creature queue
6. Battle algorithm
7. Make tops of each queue attack and defend
8. Transfer loser to loser stack
9. Repeat until one lineup/queue is empty
10. Announce winning team (the team with surviving players still in lineup/queue)
11. Print names of first 3 creatures in winning queue (select from top of loser stack if queue has less than 3).
12. **If there is time** sort winning queue by creature health. Creature with most health first. That way when step 11 executes, the game is returning the healthiest players as the 'top' players
13. View remaining players Y/N. Prints out rest of players in winning queue (if any) then the loser stack from top to bottom.

Testing

Unit testing

LineUp (queue)

Test Case	Input Value	Driver Function	Expected Output	Observed Output

Creatures can be added to a lineup	lineupA.add(ptr1);
	lineupA.add(ptr2);
	lineupA.add(ptr3);
		print statements in add function	“creature added” msg and a unique memory address for each creature	✓

Empty lineup	—	lineupA.length()	0	✓

Lineup with 3 creatures	lineupA.add(ptr1);
	lineupA.add(ptr2);
	lineupA.add(ptr3);
		lineupA.length()	3	✓

attack	—	lineupA.attack();	attack msg from correct type of creature (first creature in queue)	✓
defend	—	lineupA.defend();	defend msg from correct type of creature (first creature in queue)	✓
peek	Creature * ptr ptr = lineupA.peek()	ptr->attack()	attack msg from creature on top of lineup	✓

remove	lineupA.remove();
	lineupA.length();
		.remove()	2	✓

remove	lineupA.remove();
	lineupA.length();
		.remove()	1	✓

remove	lineupA.remove();	.remove()	“The queue is empty”	✓
remove	lineupA.length();	.remove()	0	✓

Heal function

Test Case	Input Value	Driver Function	Expected Output	Observed Output

heal function called through pointer	Creature *cPtr;
	cPtr =lineupA.peek();
	cPtr->heal();
		Creature::heal()	Msg:
			“Heal effect: +?? to strength.”
				✓

LoserStack

Test Case	Input Value	Driver Function	Expected Output	Observed Output

Transfer a creature pointer from the lineup to the loser stack	Creature *cPtr;
	cPtr = lineupA.peek();
	LoserStack loserA;
	loserA.push(cPtr);
	Creature *topLoser;
	topLoser =
	loserA.peek();
	topLoser->attack();
		loserStack::push()
		loserStack::peek()
			a barbarian attack message (barbarian was at the front of the lineup)	✓

Regression testing & bug fixes

Version 1 of program entered an infinite loop about 50% of the time.

Based on print out statements it appeared that this occurred when certain types of characters battling themselves.

Bug 1: Baba Yaga vs Baba Yaga: Baba does not do very much damage with attacks but her health grows when she attacks. When 2 Babas battle, the battle never ends because their health grows continuously without one ever being able to kill the other. To fix this, the Baba Yaga Soul Effect was capped at 10 uses. This gives her an advantage over adversaries, even for a few rounds of battle, but ensures the battle between 2 Babas will eventually end.

Bug 2: Harry Potter and Medusa: Battles between these characters and themselves were not terminating properly. The creatures were being added back into the lineup with a heal effect even when they had lost, resulting in an infinite loop and a never ending battle. Some of the battle logic was occurring inside a function called battle() and some of it was occurring in main(). I think some information was lost due to slicing or unexpected copy behavior and the characters were basically being unintentionally 'reset' when they left the battle function producing the undesirable behavior. I moved all of the battle logic into the battle() function and the problem was resolved.

Integration testing

Test Condition	Observed output matches expected output
Number of Characters in Line Up	✓
0 characters: error message	✓
1 character: 1 battle and 1 winner at end of tournament	✓
2 characters: battles continue until 1 queue is empty, 3 top players listed (2 from winning team, 1 from loser stack)	✓
3 characters (or more): battles continue until 1 queue is empty, 3 top players listed (3 from winning team)	✓
Creatures	✓

Entering letter or negative number when selecting type of creature results in error message, asks for input again	✓
Correct creature types are added to queue/line up	✓
Repeated creature types are unique creatures (keep track of individual info during battles)	✓
Names are correctly stored with and match selected creature type	✓
Line Ups and Battles	✓
Creatures battle in the same order they were entered with correct sub-types and names	✓
Creature special abilities: Hogwarts	✓
Creature special abilities: Soul Effect	✓
Creature special abilities: Glare	✓
Creature special abilities: Blue Men Defense	✓
Correct amount of attack power and defense applied	✓
Survivors get healed and re-enter correct queue	✓
When creature battles a second time it has correct strength (carries over from previous battle)	✓
Scoring	✓
Score updates correctly as creatures battle	✓
Winning team is based on final score	✓
Top players are drawn from winning team and then the top of loser stack as required in design	✓
Input/Output messages are clear and consistent	✓
Creature vs Self - battle terminates correctly (no infinite loops)	✓
Barbarian	✓
Harry Potter	✓
Medusa	✓
Baba Yaga	✓
Blue Men	✓

Usability Test and Creature Analysis

Player A	Player B	Winner
Barbarian	Barbarian	Barbarian (B)
Barbarian	Harry Potter	Harry Potter
Barbarian	Medusa	Barbarian
Barbarian	Baba Yaga	Baba Yaga
Barbarian	Blue Men	Blue Men
Harry Potter	Barbarian	Harry Potter
Harry Potter	Harry Potter	Harry Potter (A)
Harry Potter	Medusa	Harry Potter
Harry Potter	Baba Yaga	Harry Potter
Harry Potter	Blue Men	Blue Men
Medusa	Barbarian	Medusa
Medusa	Harry Potter	Harry Potter
Medusa	Medusa	Medusa (A)
Medusa	Baba Yaga	Baba Yaga
Medusa	Blue Men	Blue Men
Baba Yaga	Barbarian	Baba Yaga
Baba Yaga	Harry Potter	Baba Yaga
Baba Yaga	Medusa	Medusa
Baba Yaga	Baba Yaga	Baba Yaga (A)
Baba Yaga	Blue Men	Blue Men
Blue Men	Barbarian	Blue Men (A)
Blue Men	Harry Potter	Blue Men (A)
Blue Men	Medusa	Blue Men (A)
Blue Men	Baba Yaga	Blue Men (A)
Blue Men	Blue Men	Blue Men (B)

Battle Results

Creature	Wins
Barbarian	2
Harry Potter	6
Medusa	3
Baba Yaga	4
Blue Men	8

Strongest Charater: Blue Men
Weakest Character: Barbarian
Balanced Characters: Harry Potter, Medusa, Baba Yaga

Reflections

Changes, Issues, and Resolutions

1. Need to add a length function to my lineup queue both for initial testing purposes and for additional game functionality.
2. Creatures are being added to the queue, but I am having difficulty retrieving them with my remove function.
3. Creatures can be added and removed* from queue. Appropriate attack/defend functions can be accessed. *Remove is removing the node, but it should also store removed creature pointer in a variable but this is not happening. This could be a problem when I try to implement the loser stack. Maybe I need to be doing something with the CreatureNode instead of Creature? Do I need to make a copy constructor?
4. Added attack() and defend() functions to lineup. These call the attack and defend of the first creature in the lineup.
5. Divided Remove() into TWO separate functions: remove() which removes the node from the queue and peek() which returns a Creature pointer.
6. Method of picking winning team and best characters too complicated. Instead, lineups will continue battling until one lineup is empty. Use isEmpty function. The last team with a player still in the lineup will win.
7. Best/winning characters will be the remaining characters in the lineup. If the lineup has less than 3 characters, then the top of the loser stack will be the next best characters.
8. Could not make a separate function to load the lineup queue because destructors ran when function exited. Had to implement directly in main.
9. Program crashes is only 1 creature per lineup is selected. Fixed output statements to account for this case.
10. When 2 Harry Potters battle each other an infinite loop is created. For some reason both Harry's get re-added to the list even when they die. May have something to do with Hogwarts effect. Fixed battle logic; moved some logic from main into battle() function.

Lessons Learned

- Planning

I am improving at planning my programs. In the last assignment, I over planned and threw out most of my design once I started writing code. this time, rather than trying to create an outline of the whole program and write pseudo code, I focused on reusing my code from older assignments and planning a list of mini programming tasks that had to be completed. Also, it seems to help considerably when I use the simplest possible interpretation of the assignment, then add on additional features as necessary, rather than trying to get too fancy at the outset.

- Importance of Re-usable Code

I re-used code from assignment 3 and module 2. Without this, it would have been impossible to complete the assignment in the time frame given. Writing code for re-use is definitely a learned skill that you get better at with practice. The code from previous assignments came together really well for this assignment. This was not the case working on Module D, and that is probably why I struggled so much earlier in the semester.

- Encapsulation of Code

Initially, my remove function did 2 things: it removed the node from the queue and it copied the data from that node into a catch value that was entered as a parameter. However, this caused problems when I was trying to move the losing creature over to the loser stack. Instead, the remove function was separated into two functions, remove and peek. Separate functions turned out to be more useful than one function that does both.

- Inheritance

this assignment demonstrated the usefulness of inheritance. With inheritance, adding functions like getName and getHealth to creature class meant you don't have to go back and add that to all sub-types.

- Usability and Creature Analysis

The test cases described in the integration testing section were enough to determine that game 'worked,' but I wanted to play with it a little more and see how the creatures compared to each other. This would give me a chance to evaluate the look and feel and "playability" of my program. The set up made it very easy to compare larger sets of creatures. It was easy to compare how the individual creature types compared to each other. Setting up a large tournament as described in the creature analysis table gave me a chance to both compare the characters and see how the program performed at a larger scale. Clearly the Blue Men were the strongest character and the Barbarian was the weakest. Harry Potter, Medusa, and Baba Yaga all appeared to be balanced in the middle. Additionally, this somewhat validates my scoring system, even without sorting by health. Blue Men consistently ranked in the top players group, and teams with more Blue Men usually won. Likewise, Barbarians were consistently ranked at the bottom of the pack, and teams with a large number of barbarians usually lost.

- Doubly linked vs singly linked queues

The directions for module 2 called for a doubly linked queue structure, and the directions for assignment 4 called for re-using the code from module 2. However, a doubly linked structure was not required to implement the program described. Therefore, the components for a doubly linked structure are included in the program (mainly a pointer to the previous node); however, they are not used or tested. Perhaps, they will be required in a future assignment. Maybe they help with sorting or some other advanced function. Once they are required, they can be more fully developed and implemented.

Future improvements

- There is an advantage to going first (Team A). Future implementations should have some sort of ‘coin flip’ to vary which player, A or B, goes first.
- Sort the characters of winning team by strength after end of tournament, this way top player will be determined by strength/health points.
- More characters with additional interesting effects and abilities.
- Allow the users to use spells and special abilities to modify creatures abilities during gameplay. Examples: a circle of protection spell that adds bonus points to a creature’s armor, a fireball spell that eliminates multiple creatures, resurrection spell that allows player to take a creature off the loser pile, a stun or sleep spell that prevents an opponent from attacking or defending for a period of time.
- Artwork, sound, and animation. Just kidding. No way.

Acknowledgements

Thank you to Chris Snyder, Ian McQuoid, and Niza Volaire for helping with the modules as well as this assignment.

Resources

Strings
http://www.tutorialspoint.com/cplusplus/cpp_strings.htm