Ahmad El-Bobou

162

Fall 2018

Project 4 Design

This project's object is to turn Project 3 into a tournament style game. All character classes will be included from Project 3. The main things that will change a few. The overall structure of the battle is the same, but we must have many fighters from each team fight against each other. To store the data, I will reuse the Queue class that I created for Lab 7. What must be changed in the class is the value stored by the node will be substituted for a pointer to a character. This is so we can dynamically add characters to it. All of the member functions that interact with the value must be modified and the pointers to the characters must in each node must be deleted when the queue is destroyed. This is a simple addition to the destructor function. In order to give the characters individual names, I will modify the character class. I will give the class a string variable named name. I will create a member function that takes an input and sets that input as the characters name. This member function will be called whenever a new character node is created.

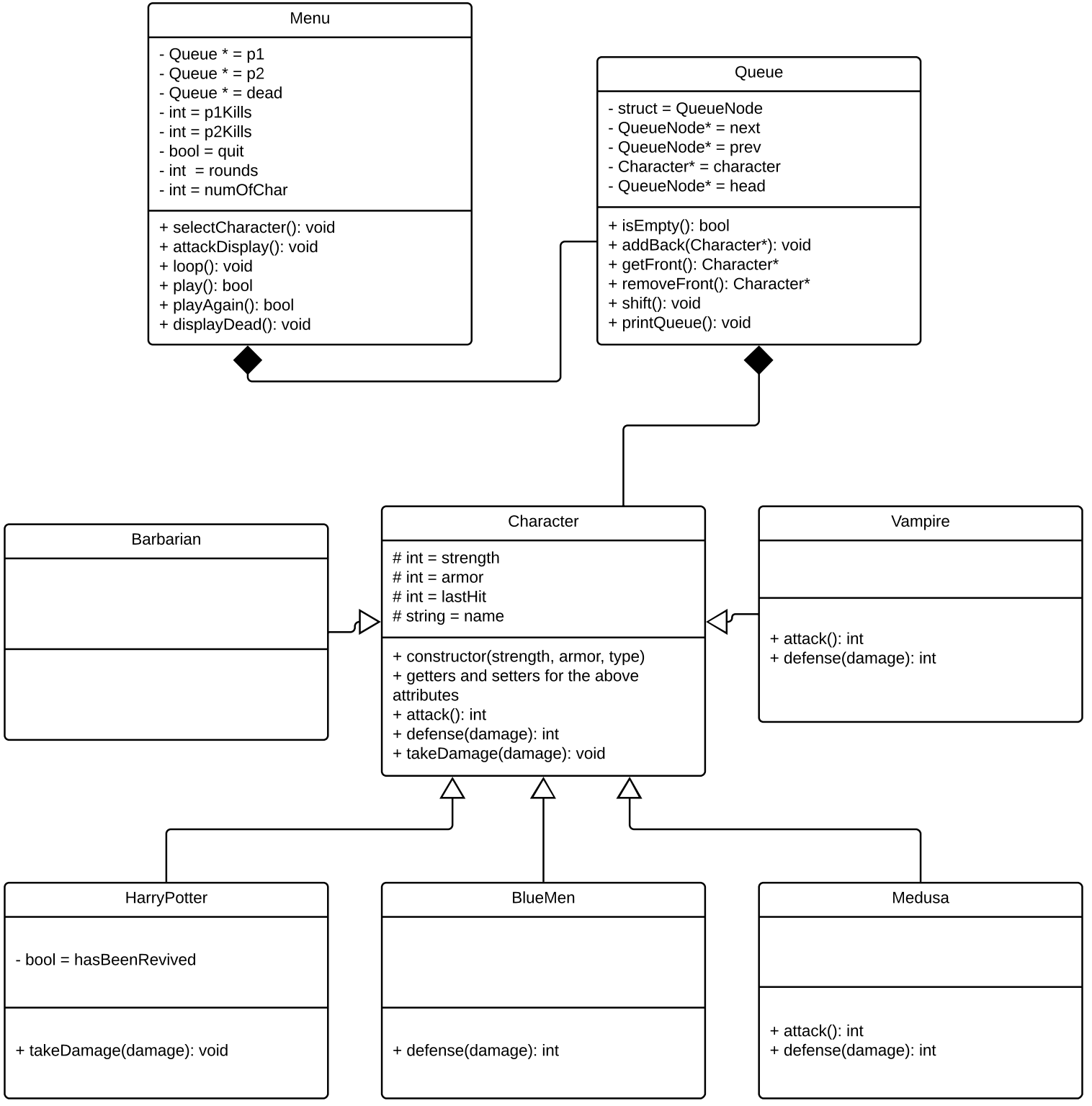
The main mechanics of the game will be stored in the Menu class. This class will create 3 Queue objects: 2 for the players and one for the dead characters. The fighting mechanic will function similarly to the project 3 fighting mechanics except the display of the information will be greatly reduced. Each battle will only display the characters and then the victor. The characters that die will be passed through the removeFront function of the player Queue object and directly in the addBack function of the Queue object for the dead characters. The scoring system is very simple. When a character of a player wins a battle, its player receives a point. At the end of the competition, the player with the most points wins. After the tournament, the player will have the

option to display the dead characters. This will be done by calling the Queue removeFront function and placing all of the nodes into an array. The array will then be printed backwards to show the characters who died most recently first.

Testing

Test case	Driver	Input	Expected Outcome	Observed Outcome
Menu function	Tested the menu function in isolation to see if each input assigned the correct character and Queue objects take the proper amount of characters	5 1-5	Queue with 5 nodes 1 - Vampire 2. Barbarian 3. Bluemen 4. Medusa 5. Harry Potter	Queue with 5 nodes 1 - Vampire 2. Barbarian 3. Bluemen 4. Medusa 5. Harry Potter
Character::setName() is correctly called in the construction of classes that inherit from Character	Created each character in main in isolation		Each character instantiated requested a name	Each character instantiated requested a name :wq
Checked several characters of character of the same and different type for the same player		Ran the simulation many times choosing different and same characters inside the same Queues to check for errors	No errors	No errors
Start game		1	Game begins	Game begins
End game at start		0	Game ends	Game ends
End game loop		0	Game ends	Game ends
Replay game		1	Game restarts and asks play to choose characters again	Game restarts and asks play to choose characters again

Run with valgrind to check for memory leaks		Valgrind ./output Create 3 characters each player	No leaks	No leaks
Run on flip to test for errors		Run make command	Program compiles with no errors and runs properly	Program compiles with no errors and runs properly
Input Validation functions have already been tested				


Menu
- Queue * = p1
- Queue * = p2
- Queue * = dead
- int = p1Kills
- int = p2Kills
- bool = quit
- int = rounds
- int = numOfChar
+ selectCharacter(): void
+ attackDisplay(): void
+ loop(): void
+ play(): bool
+ playAgain(): bool
+ displayDead(): void
Queue
- struct = QueueNode
- QueueNode* = next
- QueueNode* = prev
- Character* = character
- QueueNode* = head
+ isEmpty(): bool
+ addBack(Character*): void
+ getFront(): Character*
+ removeFront(): Character*
+ shift(): void
+ printQueue(): void
Barbarian
Character
int = strength
int = armor
int = lastHit
string = name
+ constructor(strength, armor, type)
+ getters and setters for the above attributes
+ attack(): int
+ defense(damage): int
+ takeDamage(damage): void
Vampire
+ attack(): int
+ defense(damage): int
HarryPotter
- bool = hasBeenRevived
+ takeDamage(damage): void
BlueMen
+ defense(damage): int
Medusa
+ attack(): int
+ defense(damage): int

Reflection

I really enjoyed this project because it gave me an opportunity to reuse a look of code. As a software developer, I am sure I will spend lots of time scrolling through old code that needs to be repurposed or used in a different context. Even though I was just using my own code, I found it a great learning experience. Because I joined two previous assignments, Lab 7 and Project 3, I was even able to review some concepts that we've gone over earlier in the program.

I changed very little from my original design. I opted to add the addName function of the character class to the constructor to save code being repeated often through the project. This may be a terrible idea. Generally, I think, input output should not be handled by classes. This makes the class non reusable outside of a console setting.

Another thing I decided to change was how I printed the dead players. I first wanted to create a dynamic array, but I decided the go the easy route and use a vector. Because the vector will take care of its own memory deallocation, this makes my job a little bit easier. Thinking back, maybe I should have created a function that prints the queue backwards. That would have given me a nice queue object to work with later.