Ryan Mann
CS 162
Project 4
25 November 2019

Design

This project is a bit different than the others, with a different design paradigm, so I approached it in a slightly different, and overall I believe, healthier way. The project focused on using old code to modify and implement in an updated process, synthesizing with new data storage techniques we have learned. Due to this, it is noted what we have and what we need to accomplish the problem, and that forms the backbone of how this was developed. The same format will be followed with slight modifications/annotations.

main:

- Create teams
    - Create fighter * number per team
- Introduce
- Combat loop
    - Change references
- Post combat functions
    - Increment points
    - Add to losers
    - Delete from losing team
    - addBack() to winning team
    - Delete from winning team
    - Recover winner
- Tally points and display winner
- Display loser pile
- Deallocate memory
- Prompt play

character(s):

- Change constructors
- Add recovery function

list/node:

- Research new format
- No memory leaks

- addBack/Front, deleteBack/Front, printList
- p1, p2, losers

charMenu:

- combatMenu()
    - Same start menu from project 3
- charMenu()
    - Streamline character creation, return a character object pointer

Reflection:

character.hpp:

Need: Add recovery function

For the character header file, which is an abstract class through which the others inherit, I did not have to make many modifications, merely adding a virtual recovery function for the inherited classes to use. In addition, I removed the setStats() function, and replaced it with standard constructors, because I did the last project very tired and this is better.

medusa/barbarian/vampire/harrypotter/bluemen.cpp/hpp:

Need: Add recovery function, add ability to name

For each individual character, I needed to implement the recovery function and add a way to name the character. These were both simple. For the recovery function I had it modify the current strength by 1.3 and for the naming, I replaced setStats() with standard constructors, with the parametrized taking name as an argument.

node.cpp/hpp

For the node class, I scrapped my earlier implementations of linked lists and started anew with a fresh format, tailored to this project. I had a shaky understanding of these concepts until I started to implement them. The nodes accept a character and are used to create the linked list.

charLinkList.cpp/hpp

This implementation of a linked list is my most complete yet. I had difficulty with this concept until doing copious amounts of research, examining many different forms. I am unsure why but this was the most difficult concept for me so far in CS. This list accepted nodes that each contained a character. This list is used to hold each lineup and additionally a separate list is used for the losers.

charMenu.cpp/hpp

Need: Select characters menu, create player differentiation

This is the set of menu functions I used on the project, and it contains three parts. First off, a structure holding linked lists is created to allow for player differentiation. Then, two menus

are created, one to prompt game continuation, and one to allow for character selection and creation.

charMain.cpp

Have: Character functions, Combat loop, Menus

Need: List containers, allow multiple fighters, losers container, lists to reference

The first thing I realized about this project is that I had most of it already done in project 3, and the parts that I did not would not be hard to implement. The major changes I noticed were a creation of a list, which through a lot of research I found better fundamentals in how to implement; the allowing of multiple characters in a team setup, a losers container, and changing the references in main to the lists instead of objects. I started slowly modifying the program to include these. Allowing multiple fighters required creation of player differentiation (through a player structure) and the streamlining of character creation and implementation by putting into a function charMenu(), which returned the character created as an object, to be added to lists. The losers container was the same thing, though one of my several issues was there, which I will talk about later. Using the lists as reference in main was pretty easy, just the same logical implementation. Several problems I encountered were actually in reference to main. In my update function, which will later be better modularized, I mixed up the two lists and ended up having recover called to the wrong character, which caused an issue when it was called to an empty list. I ran through the debug tool to find that the issue was on Node calling getVal(), cluing me in to a reference to something nonexistent, so I looked harder and laughed as I realized I mixed them up. The second issue I encountered was an issue with points. I wanted to implement the points as part of the player struct, but unfortunately I had some bugs with the implementation and am not sure how to fix it elegantly, so I will attend office hours. I ended up just using a similar implementation to project 3, but I dislike it and want to make it cleaner. The third issue I found was the lack of inclusion of ->getVal() in the printList() function call in main. It was returning hexadecimal representations in the console print so I immediately knew it was an issue with what I was printing. I just printed the getName() return instead of the getName()->getVal(), which was a silly error on my part. I am very fortunate I got a 100 on project 3, which meant I didn't have to mod my combat function at all. Overall, this project was cool because I enjoyed seeing how I can use old code to make a new project. After looking at the final project, I am thinking excitedly of implementing some combat into it; maybe like a colosseum or escape dungeon crawler.

Class Hierarchy Diagram:

- Character
    - Vampire
    - Barbarian
    - BlueMen
    - Medusa
    - HarryPotter
- Linked List (contains)
    - Node

Test Plan:

Tests involving the workings of each individual character that were specified and tested in the previous program are not listed here, they all work and combat flows as normal. Any changes made will be noted.

character(s):

Test	**Result**
Vampire creates	TRUE
Barbarian creates	TRUE
Bluemen creates	TRUE
Medusa creates	TRUE
Harrypotter creates	TRUE
Vampire recovers	TRUE
Barbarian recovers	TRUE
Bluemen recovers	TRUE
Medusa recovers	TRUE
Harrypotter recovers	TRUE

node/linkedlist

Test	**Result**
Node creates	TRUE
setVal()	TRUE
setNext()	TRUE
setPrev()	TRUE
getVal()	TRUE
getNext()	TRUE

getPrev()	TRUE
List creates	TRUE
addFront()	TRUE
addBack()	TRUE
deleteFront()	TRUE
deleteBack()	TRUE
printList()	TRUE
isEmpty()	TRUE
getFront()	TRUE
getBack()	TRUE
deAllocate()	TRUE

menu

Test	**Result**
p1 created	TRUE
p2 created	TRUE
combatMenu()	TRUE
charMenu()	TRUE
Creates and returns vampire	TRUE
Creates and returns barbarian	TRUE
Creates and returns bluemen	TRUE
Creates and returns medusa	TRUE
Creates and returns harrypotter	TRUE

main

Test	**Result**
Srand initializes	TRUE
combatMenu() prompt	TRUE
Objects created	TRUE
Team 1 prompted and created	TRUE
Team 2 prompted and created	TRUE
Introduced fighters	TRUE
Combat loop works (noted in header)	TRUE
p2 list updates	TRUE
p2 points	TRUE
p1 list updates	TRUE
p1 points	TRUE
Points displayed	TRUE
Winner displayed	TRUE
Losers displayed	TRUE
Memory freed	TRUE
Play reprompted	TRUE
Exit works	TRUE