Jared Parkinson
Assignment 4
CS 162: T. Rooker

Assignment 4 Reflections

To start off I just wanted to do a quick list of a few things you should know about the program before getting to the meat of it:

- The static Pointers to Fighters have been removed and fixed according to your last suggestions
- Many of the Creature functions that were duplicates were moved to Creature(atkRoll, defRoll). Static Data members created to store Dice per your suggestions
- Program requires 3 fighters to be entered so it always has top 3 available.
- Fighter names can be the same if you wish, this was a design choice.

Requirements:
Here is a general walkthrough of how I understood the assignment to function that I wrote in my Design document:

Program Walkthrough
1. User Input how many fighters for each side
2. Player 1 will be prompted to choose their lineup as below:
3. User Choose Fighter from list, then user types name/id of fighter (new data member creature)
4. After user chooses the Creature and Name, it will instantiate new Creature, set name of Creature
5. Once Creature is set and name is set, it will Add the creature to the Lineup Node System for Player 1
6. Once player 1 has finished, Player 2 will be prompted for the same as Player 1
7. After P2 finishes entering, tournament will start.
8. Lineup1 FRONT slot fighter 1 will battle fighter 2 from Lineup2 FRONT
9. Winner (not dead) will be placed at rear of Lineup and restored some HP
10. Loser moved to Team1 or Team2 Loser Pile
11. Winning Team will be printed to the screen, Top 3 and Loser Pile
12. Next battle will occur ad nauseum until no fighters left in one teams Lineup

In addition to this design, you must:

1. Create a Restore HP function in Creature
2. Create a Search Algorithm or method to determine Winners
3. Resolve possible issues with Fighters with special abilities. Baba Yaga needs HP reset it over max, Harry Potter can only resurrect once, Blue Men cannot go below 0 defense dice. Medusa damage amount on a roll of 12 now does just enough damage so that Baba Yaga even when maxed out on HP can still die.
4. Must use Stack and Queue linked structures for Lineup and Loser Pile

Initial Class Design: First design before changes made
TEAM CLASS: New class. Team.hpp/.cpp. This class contains the Loser Pile and the Lineup structures. Both structures reside in the one class. Each Team object is a Player. E.g. player 1 and 2.
Here is an initial design of this class:

```
Team
private:
struct qLineupNode
        Creature *fighterDataQ
        qLineupNode *next
        qLineupNode(Creature*, qLineupNode) //Constructor
qLineupNode *frontLine
qLineupNode *backLine
struct stackLoserPile
        Creature *fighterDataS
        sLoserPileNode *next
        sLoserPileNode(Creature*, sLoserPileNode) //Constructor
sLoserPile sLoserPileNode //Constructor
sLoserPileNode *botLose
```

```
public:
Team();
~Team();
//Lineup (Queue)
void addLineupNode(Creature *add); //Add node with data of Creature*
Creature* removeLineupNode(); //remove node from top of Queue FIFO
//void printRemoveQueue(); //Prints Queue and removes items
bool LineupEmptyChk(); //Checks for empty Lineup in Team (Queue)

//Loser Pile (Stack)
void addLoserNode(Creature *add); //Add node with data of Creature
Creature* removeLoserNode(); //remove node from top of stack
void printRemoveLoser(); //Prints LoserNode Stack and remove
bool LoserNodeEmptyChk(); //Checks for empty Loser Pile
void *determineWinners()
```

Initially this class was designed as the above, but I had to make significant changes to it during the programming process as I found that my methods were not efficient. I will describe those changes at the end of this reflection. TEAM has a main purpose. That purpose is to store both the Lineup and the Loser pile structs for each team. I had originally designed this to use determineWinners and traverse through all of the structures and arrange them using data members which have been now removed. These data members and functions are below:

```
REMOVED:
private
//int numFighters; //Number of Fighters desired per team
//int lineupSize; //For working with winners
//int loserSize; //Loser Pile Size
//bool topThreeFound; //Top 3 matched
//int topWin; //Highest Win
//int topAtkPoints; //Win Point System

public
//int getTopWin(); //Get Top Wins #
//int getTopAtkPts(); //Get Total Win Points
//void setTopWin(int set);
//int setTopAtkPts();
```

Function numFighters was to hold the total fighters chosen, I chose to use a different data member for this so I dropped it. lineupSize was to hold the size of the Lineup and increment as the user chose their fighters. loserSize was to increment as a new Loser node was created. Both of these sizes were supposed to be used in traversing the structures. This was a poor design. topThreeFound was to be a bool to tell me when I have the top 3 found after traversing the structs. topWin was to be set each time a fighter won and compare to find the top wins. topAtkPts was to store the highest attack points for sorting on later. Getters and Setters for wins and points were phased out. This was a whole lot of work to do something that should be simpler. The sorting of the fighters and finding of the winners for one team.

CREATURE CLASS:

Creature Class was modified to handle a few additional things that it did not previously in assignment 3. Below is the initial design of the updated Creature Class. Anything in Red was removed. Anything in Green is added.

Creature
private:
bool isDead;
int strength; //"Hitpoints" - 0 is death
int armor; //Mitigated Damage (No damage to strength)
int atkDice, defDice; //Attack Dice and Defend Dice Amount of Dice
int atkDiceType, defDiceType; //Attack Dice and Defend Dice Type of Dice (e.g. 6, 10)
int atkDiceRoll(); //Virtual Roll the Dice for Attack
int defDiceRoll(); //Virtual Roll the Dice for Defense

public:
Creature(); //Constructor
virtual ~Creature(); //Destructor
virtual int attack() = 0; //Pure Virtual Attack Action
virtual void defend(int atk) = 0; //Pure Virtual Defend Action
virtual int getStrength(); //Get Strength of Creature
virtual void setIsDead(bool state); //Set isDead bool
virtual bool getIsDead(); // get isDead bool

New Additions:

- atkDiceRoll / defDiceRoll moved to creature
- atkDice, defDice, atkDiceType, defDiceType data members added
- **Baba Yaga now can only use Soul 3 times to limit infinite fighting against another Baba Yaga

MENU CLASS:

For the menu class, my initial design was to rewrite the menu system to work for 2 players. In assignment 3 it was just for 1 player to fight once. Now the menu system will ask for a fighter number (with validation) and then show the menu system to choose fighters until you fight number has been chosen. It will show a ticker to give you an idea how many each player has left to choose. When you choose a fighter, it will call the functions in the TEAM to insert the fighter into the array and into the Lineup.

```
/*---------NEW----------*/
int fighterAmount; //Desired Fighter Amount for each team
Creature *fighter1, *fighter2; //Fighter Pointers
Team player1, player2; //Teams for Players 1&2
/*---------NEW----------*/
```

```
 /*---------NEW----------*/
 void setFighterAmount(int amt); //Set Fighter Amount
 int getFighterAmount(); //Get Fighter Amount
 void battle(); //Do Battle
 void winTeam(); //Show winning Team
 /*---------NEW----------*/
```

Private:

- fighterAmount is the desired fighter number chosen by the user
- *attacker and *defender were swapped for *fighter1 and *fighter2
- Team objects player1 and player2 are the teams with respective stats and structs

Public:

- setFighterAmount sets the amount chosen by the user
- getFighterAmount gets the amount chosen by the user
- battle() fights the two fighters at the front of the lineups until one dies

- winTeam prints out the winning team and specific stats

Updated Class Design: Updates required for poor design

TEAM CLASS:

I chose to drop the whole Struct traversing system in favor of an array system that was created on the fly dynamically when the user chose how many fighters they wanted. This design change proved to be much easier to implement and works quite well actually.

ADDED:
private
int fighterAmount: Num of fighters to choose
Creature **fightArray: Array for sorting use, printouts, and top 3 choosing
Creature *win1, win2, win3: Storing the top 3 winners for printout
public
void makeArray(int size); //New array to store all fighters as chosen
void insertArray(Creature *insert); //Insert Fighter in Array
void sortTheArray(); //Sort the Array of Fighters
void setWinners(); //Sets winners of this Team if called

New Additions:

Data members:

- fighterAmount stored the number of fighters chosen by the user
- fightArray is the array that stores the fighters as they are chosen on the fly
- win1 2 and 3 are storage for the top 3 slots

Functions:

- makeArray creates an array of size to store the fighter pointers
- insertArray adds a fighter to the array when NULL
- sortTheArray sorts the array by Wins first, then it checks if the wins are equal. If wins are equal it sorts by fighter score derived from total attack damage rolled, not dealt.
- setWinners sets the winners by checking the top 3 slots of the sorted array

MENU CLASS:

I decided that I needed to rework a bit of the Menu class to prevent less than 3 winners as stated by the assignment description. I chose to force the user to choose at least 3 fighters each team to get top 3 winners. This was because of the request to show top 3. In addition to this change, I created the winTeam() function to cut out some code in a function. There was too much code in battle() and it made it looks horribly long and confusing.

CREATURE CLASS:

Necessary changes to Creature Class included:

UPDATED:
private
std::string classType; //Class Name
int totalWins; //Track Creature Wins
int totalAttackDealt; //Track Attack Roll Dished Out
int fixedHP; // fixed max HP number
std::string fighterID; //Name or ID of Fighter
//int fighterScore; //Total Fighter Score

public
std::string getClassType(); //Get the Class Type
void setFighterID(); //Set Fighter Name or ID
std::string getFighterID();
void incWins(); //Increase wins by 1
int getWins(); //Get number of wins
int getFighterScore(); //Get Fighter Score
Removed:
int totalWins
int totalAttackDealt
//int getTotalAtk(); //Get Total Attack
//void setFighterScore(int set); //Set Fighter Score

Added:

Private:

- classType: The name of the class for printing purposes
- totalWins: track the total wins of the Creature
- totalAttackDealt: track the attack dice rolls to calculate score
- fixedHP: a HP number for comparison, specifically for 2 things: Baba Yaga to know how much HP is her max to reset and the restoreHP function for comparison.
- fighterID is the name of the fighter

Public:

- getClassType gets the Class Name
- setFighterID sets the fighter name ****** You CAN choose the same NAME if you want!!!
- getFighterID gets the fighter name
- incWins increments the win count
- getFighterScore calculates the fighter score on the fly

Removed:

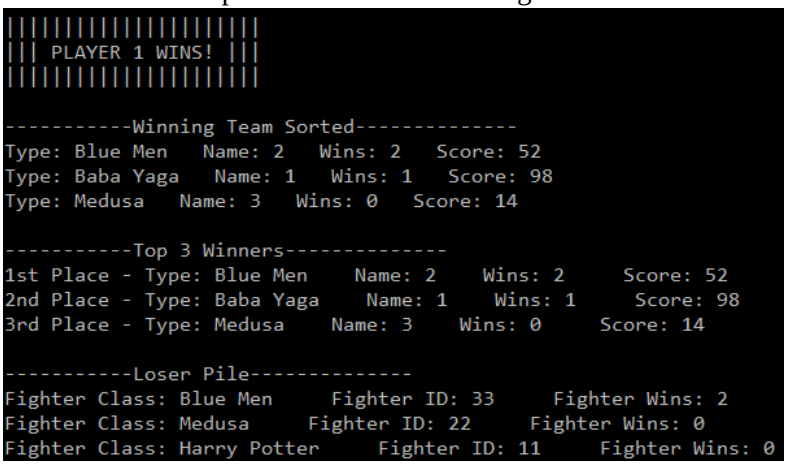
- int fighterScore was an unneeded data member
- int totalWins was an unneeded data member from poor design
- int totalAttackDealt was an unneeded data member from poor design

Testing:

Test Runs	**Action**	**Expected**	**Pass/Fail**
Menu System	Choose p1 team up to max fighters chosen	Choose up to max fighters	Init: fail, fighter count was not working 2nd pass, fixed fighter count
	Choose p2 team after p1	Choose up to max fighters same as p1	Init: pass after p1 count fixed
	Force choose 3	Force user to choose 3 fighters	Init: pass, set in do while loop
Printing	Complete a battle with 3 fighters	Print the outcome of the battle	Init: fail, the outcome was incorrect due to sorting issues 2nd: fail, the printout did NOT show the fighter Classes 3rd: pass, the classes correctly show and are in perfect order
	Printout of winner of fight	Print the winner of one single fight after each round	Init: fail, the print was not showing the fighter class 2nd: the printout was fixed and displayed class
Fighter Choices	Choose 1 of each fighter	Confirm that each fighter not only can use their special, but the proper fighter is instantiated	Init: fail, the fighters chosen were not proper instantiated 2nd: The fighter new was fixed and the

			fighter properly instantiated
	Choose more than one of the same fighter	5 barbarians can be chosen	Init: Works properly and 5 barbarians can be chosen to battle each other
	Choose fighters that are not the same for 5 rounds	Each fighter fights the other and has no issues	Init: this worked fine and no issues
	Choose Baba Yaga vs Baba to test special	Baba can only use special 3 times	Init: works fine and increments special use
	Choose Harry to test 1 use special	Harry dies once and no more	Init: works fine and harry died 1 time only in special move
Array	Test array and make sure all fighters properly are added	Array adds fighters as they are instantiated	Init: Fail, array was adding the same fighter over and over 2nd: fixed and fighters now added properly
Sort Array	Test Sorting and make sure array is sorting	Print it out and make sure sorted array is in order	Init: Fail, order was off and wins were not sorting 2nd: fail order was still off and score was not sorting 3rd: pass, fixed the sorting to use a different algorithm and it now properly sorts by wins 1st and then score
Restore HP	Test restoring HP	Quick print to make sure HP is restoring or resetting for Baba Yaga	Init: HP is properly restoring

**Here is an example Test Printout of a 3 fighter round



Final Reflections:

There were some things I wish I would have done differently on this assignment. For example:

Array vs Traversing Linked Structures: Initially, I chose to try and traverse the structures to pull the winners from the loser pile and from the lineup. This because very arduous because if there were only 2 people in the lineup, how could you get the next loser one? It was a lot to think about and a lot to implement. I was reading the forums and saw that someone had mentioned an array and I thought, OF COURSE! Why did I not think of that it is so easy! Each time a fighter is chosen it is added to the Array of each team. Then at the very end of the game, I just arrange the array by wins, then arrange by score. The scoring is a simple system of Wins + Attack dealt. This is sorted by wins first and then score.

Class Changes: I wish that I was better at knowing what will happen when I try to implement a class with the way I have designed it. I need to get better at class design to decrease my coding time. After trying to implement methods that just don't seem to be panning out, I have wasted hours of my time in that area. Then I realize I need to redesign and redo a whole portion of code because what I was trying to do was just not working out. This sets me back many hours. If I only had the foresight to know that a specific design is not good from the very beginning it would save me a lot of headache.

Data Members: I learned from this assignment that I don't actually have to have Data Members for every little thing. For example, to get the score, I simply calculate it on the fly instead of having a data member that is storing the score.

Sorting Issues: The sorting for the array was a nightmare at first. I was trying to get it to sort properly on Wins and then Scoring. I initially dropped the scoring and just tried to set the wins. That wasn't working either. Then I realized that my initial for loop was not properly traversing the array as I had thought it was. I tried a different method of array loop with the J+1 and this fixed the issue with sorting. It took me a couple hours to get this to work properly.

Final: Despite having to spend a lot of time reworking things, I feel like I learned a lot on this assignment about finding different ways to deal with things other than your first impression. First impressions are not always the best method to go with. In addition, I learned a lot about sorting and sorting on 2 separate data pieces. A double sort if you would like to call it that. I learned a lot about how to work with Structs and Pointers and how the structs all come together. I may have not gone with the traversing the structs, but I did learn a lot about moving around them. Overall, I did enjoy working on this assignment. Thanks!