Jeffrey Schachtsick
May 16, 2015
CS 162 – Assignment 4
Test Plan

Understanding

In this assignment, I will be creating a program that will be building off of a previous assignment with polymorphism. In the previous assignment, I created a class hierarchy which class objects would be created. These class objects had attributes and behaviors similar to each other, but different values and were thus child classes of the parent Character class. In the program the user was able to create these character objects and then be able to fight them in like an arena battle.

This program will take the same idea of creating character objects, but will incorporate to a two player game. One player will be able to create a static list structure of creatures and fight them against another player's list structure of characters. If the user's fighter wins, the fighter gets healed to some percent, moves to the back of the line to wait for the next fight. The loser moves into the loser pile as a placement. At the end the first, second, and third place fighters will be displayed. Also, the total points will be displayed.

How the fighters become placed is really up to me as well as the total points, but the object is to be able to arrange the structures by placing the character objects using both a FIFO (first-in-first-out) and FILO (first-in-last-out) methods. The FIFO and FILO data structures is a goal of this assignment, along with being able to modify the parent abstract class.

Testing

Test Title	Description	Input	Expected Output
Create lineups	A user is asked how many fighters will be for each side and the program will create static space for each player side and space for the loser pile.	User enters 3 for each side	There is 3 available nodes for player 1, 3 available nodes for player 2, and 6 nodes for the loser pile.
Create a full lineup	A user setup a lineup of 6. Enough for each of the charcters plus an extra Blue Man. Blue man rule states only 1 cannot fight.	User enters a Barbarian, Reptile, Gargoyle, 2 Blue men, and Shadow with names and types.	Each character is displayed with attributes for the Player's lineup.
Character is a winner	6 characters are made. A fight between two opposing sides is made between characters, until 1 of the fighters strength runs out. The winner gets to recover 50% strength and is placed at the bottom of the node list.	Reptile is fighting a Barbarian. Reptile wins with 4 strength left.	List of player's fighters. 1) Shadow – 12 strength 2) Gargoyle – 12 strength 3) Reptile – 6 strength (Reptile moved to bottom, 2 points accumulated for 50% health recovery, 10 add bonus points to players total score)

Character is first loser	A fight between two opposing sides is made between characters, until 1 of the fighters strength runs out. The first loser gets to be placed at the top of the loser pile.	Reptile is fighting a Barbarian. Reptile wins with 4 strength left.	Placement Pile: 6th place - Barbarian
Character is a loser other than first loser	6 characters are made. A fight between two opposing sides is made between characters, until 1 of the fighters strength runs out. The winner gets to recover 50% strength and is placed at the bottom of the node list. The loser gets to be placed at the next spot of the loser pile.	Reptile is fighting a Gargoyle. Reptile wins with 4 strength left.	Placement Pile: 6th place – Barbarian 5th place – Blue Man 4th place – Shadow 3rd place - Gargoyle
Battle for first place	6 characters are made. A fight between two opposing sides is made between characters, until 1 of the fighters strength runs out. The winner gets to recover 50% strength and is placed at the bottom of the node list. With no other characters left, the loser gets placed in 2nd and the winner is 1st.	Reptile is fighting a Blue Man. Reptile wins with 4 strength left.	Placement Pile: 6th place – Barbarian 5th place – Blue Man 4th place – Shadow 3rd place - Gargoyle 2nd place – Blue Man 1st place – Reptile
No more players left for a player	6 characters are made for both players, but a player runs out of fighters. Fighters with the most strength win.	Reptile is fighting a Gargoyle. Reptile wins with 4 strength left.	Placement Pile: 6th place – Barbarian 5th place – Blue Man 4th place – Shadow 3rd place - Gargoyle 2nd place – Reptile with 6 1st place – Blue Man with 12

Design

Class: Character

Behavior:

+ virtual attack(): int

+ virtual defend(): int

Class: Goblin :public Character

Attributes:

- isAlive: bool
- attackDice[]: vector <int>
- defenseDice[]: vector <int>
- armor[]: vector <int>
- strength: int

Behavior:

+ attack(): int
+ defend(): int

Class: Barbarian: public Character

Attributes:

- isAlive: bool
- attackDice[]: vector <int>
- defenseDice[]: vector <int>
- armor[]: vector <int>
- strength: int

Behavior:

+ attack(): int
+ defend(): int

Class: Reptile : public Character

Attributes:

- isAlive: bool
- attackDice[]: vector <int>
- defenseDice[]: vector <int>
- armor[]: vector <int>
- strength: int

Behavior:

+ attack(): int
+ defend(): int

Class: Blue : public Character

Attributes:

- isAlive: bool
- attackDice[]: vector <int>
- defenseDice[]: vector <int>
- armor[]: vector <int>
- strength: int
- bluePop: int

Behavior:

+ attack(): int

+ defend(): int

Class: Shadow : public Character

Attributes:

- isAlive: bool
- attackDice[]: vector <int>
- defenseDice[]: vector <int>
- armor[]: vector <int>
- strength: int

Behavior:

+ attack(): int
+ defend(): int

Add a fighter

- Drop menu of which character to create
- User chooses from list
- Create character, add to player list of fighters in FILO.

Add a fighter to placement

- If the head is Null,
  - add fighter to head-> next
- Else
  While placement node-> next != NULL
    Move to the next node
  Add fighter to node -> next

Remove a fighter

- Ask user, which character to remove
- If character exists,
  remove character from list
- Else
  Does not exist

Stage a fight

- User needs at least 2 alive characters in list, exit otherwise.
- Ask user for attacking character (loop, until correct character is entered)
- Ask user for defending character (loop, until correct character is entered)
- Call the attack method for attacking character
- Call the defend method for defending character
- Get armor and strength points from defending character

- Subtract attack total from defense total
- If total < 0
    Subtract this total from armor. Armor remains same.
    If the new total < 0
    Subtract new total from strength
    If strength <= 0
        Character is not alive.

Display the player fighters
- Go through each character in list and display character attributes in class.

Display the placement fighters
- Go through each fighter in list and display fighter attributes in class

Main
- Welcome the user
- Ask the user how many fighters per each player.
- Create static structures for player 1, player 2, and multiply the user input by two to create the Placement pile structure.
- Loop: add fighters for player 1.
- Loop: add fighters for player2.
- Loop: until a player runs out of fighters
    - stage a fight(taking player 1's #1 in list, taking player 2's #1 in list)
    - add loser to placement list
    - move winner to end of winner player's list, add recovery health
    - Calculate total points.
    - Display each player's list.
- Display placement list and total points for each player
- Exit the program

May 21, 2015

Reflection

What did you learn about the problem as you went? Why or how did you learn it?

In this assignment, I needed to create a program that was ultimately going to accomplish the goal of creating static structures or ADTs (Abstract Data Types). These static structures were going to be created by asking the user to enter a value and this value was going to represent the size or how many nodes were going to be in this structure. Each node was going to be either some NULL value until I add an object to it, or I was going to delete the contents of the node to make it NULL. There were some types that I could use, but this assignment specifically wanted to have 2 queues for the order of fighters for each player, and a stack that was to represent the placement order of winners and losers.

In learning how to accomplish this goal in creating ADTs, I turned to the assigned readings for assistance. I had found some great examples on how to do methods essential for both the queue and

stack. I was able to implement those examples more or less into the code, but I needed to modify a bit with objects I had.

What tests didn't work out the way you expected? What alterations did you have to make to your program due to failed tests? How could your planned tests have been more complete?

I had found my tests were working as I had expected after implementation of code from the design. I did forget on my tests document, that I needed to implement some tests on when a player is going to win over another. I think that was missed when I was writing up the table from the many bullets of requirements from the assignment.

What was missing or needed to be altered from your initial design, and why?

When I initially created this document for my understanding, test cases, and design, I had originally thought I was going to implement with the topic of Linked lists. This topic was assigned in the previous week and where the assignment could also be started. I had originally thought this was going to be the case, because the previous assignment had that same sort of timeline.

However, as I came to realize after working on implementing the code was quite difficult and I started to work on stacks and queues topic in the following week, that I needed to change how I did things. After working through the stacks and queues topic, was able to create a new design with tests and paper and implement that to code. I've been finding it's really unclear with the assignments when I can/should start on the assignments.

My Characters header and implementation files really did not need much change which was a good thing. The only things that I needed to work on with stacks and queues is the implementation of classes involving the player's queue and the loser pile stack.

What problems did you encounter during implementation? How were you able to solve those problems? What outside sources (sites, books, or other materials) did you find helpful?

The major problem to figure out was how to move objects from nodes in the queue around or into the stack when I needed to. Once I had the methods that would do such things as peek, pop, push, I could use those just about anywhere to move my objects around. I haven't seen any examples where I could move objects to other nodes, so this was a see if it works kind of thing. Luckily it all kind of worked out correctly. The resources for these methods I had found in the Stacks and Queues Chapter reading material.

Can you generalize any parts of your problem solving experience in a way that might help you on future assignments?

I think the one thing that really helped me get through understanding the overall objective for this assignment was the reading material in the course textbook. I did some research online, and there were some shorter ways to do all of this, but I think it is nice to be able to create your own methods and learn how to do it yourself. Plus, the reading had some handy examples on creating the methods. So, going through the reading material is a very helpful thing to do when trying to accomplish assignments.