Project 4

Oliver Solorzano
CS 162

Design

1. Character Class (Abstract)
   - a. Protected Variables
     - i. Attack
     - ii. Defense
     - iii. Armor
     - iv. Strength points
   - b. Protected functions (virtual)
     - i. Attack
       - 1. Override for each subclass
         - a. Return damage pts rolled (attacked)
     - ii. Defense
       - 1. Override for each subclass
         - a. Takes damage from attacker, calculates damage inflicted, subtract from defender strength points
         - b. Damage = attacker’s roll – defender’s roll – defender’s armor
2. Vampire subclass
   - a. Attack
     - i. One rand() from 1-12
   - b. Defense
     - i. One rand() from 1-6
   - c. Armor
     - i. 1
   - d. Strength points
     - i. 18
   - e. Special
     - i. Charm
       - 1. 50% chance opponent doesn’t attack
         - a. Rand() from 1-2
           - i. If 1 proceed w/ attack
           - ii. Else if 2 skip opponent attack
3. Barbarian subclass
   - a. Attack
     - i. Two rand() from 1-6
   - b. Defense
     - i. Two rand() from 1-6
   - c. Armor
     - i. 0
   - d. Strength points
     - i. 12
   - e. Special
     - i. None
4. Blue men subclass
   - a. Attack

        i. Two rand() from 1-10
    b. Defense
        i. Three rand() from 1-6
    c. Armor
        i. 3
    d. Strength points
        i. 12
    e. Special
        i. Mob
            1. Adjust Defense based on strength points
                a. If strength is > 8, defense is 3d6
                b. Else if strength is > 4 && strength < 8, defense is 2d6
                c. Else if strength is < 4, defense is 1d6

5. Medusa subclass
    a. Attack
        i. Two rand() from 1-6
    b. Defense
        i. One rand() from 1-6
    c. Armor
        i. 3
    d. Strength points
        i. 8
    e. Special
        i. Glare
            1. If attack rand() == 12
                a. Damage is == Enemy strength points
                    i. Enemy health is taken to zero

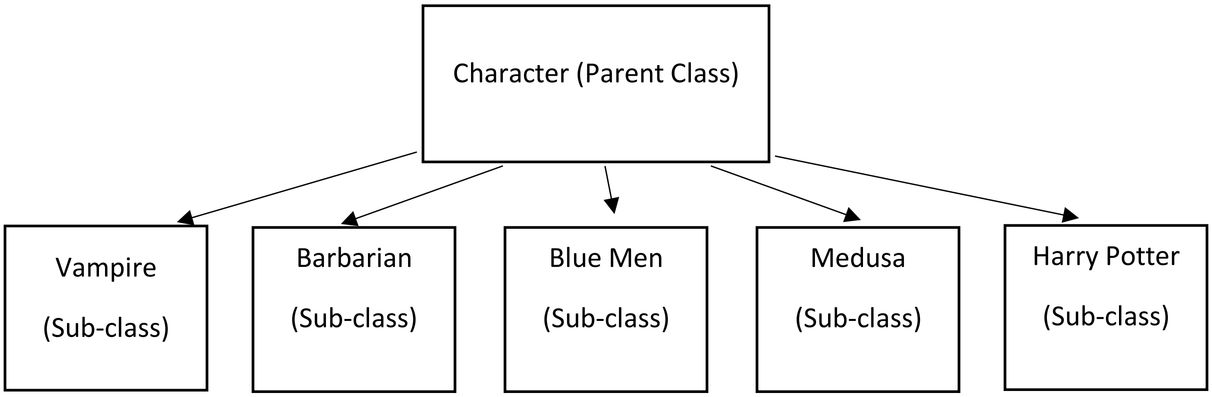
6. Harry Potter
    a. Attack
        i. Two rand() from 1-6
    b. Defense
        ii. Two rand() from 1-6
    c. Armor
        iii. 0
    d. Strength points
        iv. 10
    e. LifeCounter
        a. Set to 1, decremented if Hogwarts is used
    f. Special
        v. Hogwarts
            1. If strength points hit 0 and LifeCounter != 0;
                a. Strength == 20
                b. LifeCounter = 0

Game class

Project 4 Oliver Solorzano
CS 162

1. Ask user to choose how many fighters they want for both teams
   a. Have linked list for both lineups(one for each team) and losers
      i. Loop through character choice (limit with user input)
         1. For each character
            a. Type
            b. Name
2. Rounds loop
   a. Fighters are the head of lineup list
   b. If one character's strength points < 0, exit loop, end battle, else keep looping
      i. Winner goes to back of lineup
      ii. Loser goes to losers list
      iii. Winner gets health restored based on rand() from 1-10, converted to percentage
      iv. Next battle begins with characters at top of lineup
      v. Team points are given: winner +2, loser-
      vi. Game ends when either team has no more characters in lineup
   c. Display Function (inside rounds loop)
      i. Call get functions for each character
         1. Character type
         2. Armor pts
         3. Strength pts
         4. Attack roll value
         5. Defense roll value
         6. Total damage
         7. Defenders strength points after damage
3. After battle ends
   a. Print the result of the game
      i. Show final score for each team
         1. Display winning team for tournament
   b. Ask user if they want to see the loser pile
      i. Print from top to bottom (last defeated at top)
   c. Ask user if they want to keep playing

Class Hierarchy



Project 4

Oliver Solorzano
CS 162

Test Table

Test Case	Input	Expected	Observed
User enters option to play again or quit	Enters “1” or “2”	If “1” plays again If “2” exits program	“1” played again “2” exited program
User enters invalid character selection	Enters value != int, or not within range	Tells user entry is invalid, tries again	Tells user entry is invalid, tries again
Character stats are appropriately displayed	Play round	Health displayed before is changed after damage taken	Health displayed before was changed after damage taken
Charm ability is appropriately executed	Play vampire class	When charm is cast, opponent doesn’t attack	When charm was cast, opponent didn’t attack
Glare ability is appropriately executed	Play Medusa class	When glare is activated, damage done automatically kills opponent	When glare was activated, damage done killed opponent instantly
If player dies after first attack, round ends	Play characters until first attack kills opponent	After attacker kills opponent, round ends, doesn’t get turn	After attacker killed opponent, round ended, didn’t get turn
If Harry Potter dies, comes back once, with 20 strength	Play Harry Potter class	Harry Potter comes back once hp is zero	Harry Potter came back ONCE with 20 hp
If Blue men strength is diminished, defense goes down	Play Blue men class	Blue men class, roll smaller numbers below 8 and 4 strength pts	Blue men class, rolled smaller numbers below 8 and 4 strength pts
Lineups are in the same order they were entered in	Name each fighter from 1-6 Print teams to console	Teams printed in order	Teams printed in order
Winner goes to back of lineup	Track 1st winner to end of game	Winner will go to back of lineup, lineup will proceed	Winner cycled as expected
Loser gets put on top of loser container, removed from lineup	Track 1st loser to end of game, print loser list	1st loser should be at bottom of printed list	1st loser at bottom of printed list
Recovery function works as it should	Track character that took damage to next fight	Character that takes damage, regains 50% (rounded) strength points	Character that took 3 damage regained 1 strength point Character that took 4 damage regained 2 strength points

Project 4 Oliver Solorzano
CS 162

Reflection

Having recently written the base for this program and having coded abstract data structures, I felt confident coming into this assignment. I completely underestimated the changes I would have to implement into what I had for Project 3. Other than the implementation of the recovery function, the rest was unforgivingly buggy. My first hurdle was figuring out what data structure to use, which wasn't straightforward after realizing I needed a structure that I could add nodes to in both the front and back. Neither of the lists that I had coded in the previous weeks had functionality for both. I settled on the circular linked list structure and decided to modify the code so I could add nodes to the back.

Perhaps the hardest part of this project was moving the winners and losers after each round. For the winners, because they were staying in the same list, I decided to swap around the address of the character objects the fighter pointers were holding. For the losers, it wasn't so simple. I needed a way to migrate the character pointers from one list to another. In my first attempt, I decided to try to migrate the pointer addresses from the lineup to the losers list. First I created a new queue node structure in the losers list, then I set the fighter pointer in the new node to hold the same address as the old node in the lineup. I then deleted the node in the old list. This wasn't working, as my printQueue function wouldn't print and my removeFront function would crash, citing a read access error. There was something about the losers list that was different from the lineups. After many unsuccessful attempts to get it to work, I decided to just create new character objects and have the fighter pointers in the losers queue nodes point to them. I took the name/type characteristics of the losers and constructed new ones in the losers queue. The fact that newly constructed characters would have their stats reset wasn't of any consequence for what the guidelines demanded. All I need was the name and type of the losing fighters, and to display them in LIFO order. By doing this my loser queue had the same exact setup as the lineups, therefore it worked flawlessly with the print and remove functions. While I hated to have allocate new memory for such a menial task, it was the only path that worked and I was running out of time. I ensured that the program didn't leak memory and I called it a day.