David Ramirez

Project 4 Reflection Document

Design of classes and functions and how they are used

Base class:

Creature class (abstract class)

Protected: (data members)

String name

- String that holds Creature derived class object name

String customName

- String that holds Creature derived class user defined name

String attStr

- String that holds the attack strength description of Creature derived class object

String defStr

- String that holds the defense strength description of Creature derived class object

String special

- String that holds the special ability description of Creature derived class object

int armor

- Int that holds the armor points of the Creature derived class object

Int strength

- Int that holds the strength points of the Creature derived class object

Int baseStrength

- Int that holds starting strength points of Creature derived class object
- Used to determine how many strength points lost after fight

Public: (member functions)

Creature() (constructor)

- Creates an “empty” creature class object

Virtual int attack (int) = 0

- Pure virtual function to be overridden in derived class

Virtual int defend (int,int) = 0

- Pure virtual function to be overridden in derived class

Virtual int damage (int,int)

- Used to simulate damage done to creature by subtracting armor points from damage done, then subtracting that difference from creature strength points

Virtual “get” functions for all data members

- Return either string or int (depending on corresponding data member data type
- getStrength(), getArmor(), getAttack(), getDefense(), getName(), getCustomName(), getSpecial()

Virtual “set” function for customName data member

- sets the customName of a Creature derived class object to a user defined string

<u>Derived classes:</u>

Barbarian class

Private: (data members)

Same as base class

Public:

Barbarian () (constructor)

- sets all data member values to represent Barbarian class

Int attack (int)

- used to simulate Barbarian attack by rolling 2 six sided dice (generate random numbers between 1 and 6)

Int defend(int, int)

- used to simulate Barbarian defend by rolling 2 six sided die (generate random numbers between 1 and 6)

All other functions are used from base class

Vampire class

Private: (data members)

Same as base class

Public:

Vampire() (constructor)

- sets all data member values to represent Vampire class

Int attack(int)

- used to simulate Vampire attack by rolling 1 twelve sided die (generate random number between 1 and 12)

Int defend(int, int)

- used to simulate Vampire defend by rolling 1 six sided die (generate random number between 1 and 6)
- Vampire special ability “Charm” is implemented in this function
  - 50% percent chance of using ability each turn
  - If used, opponent does no damage
  - Trumps any opponent attack special ability

All other functions are used from base class

Medusa class

Private: (data members)

Same as base class

Public:

Medusa() (constructor)

- sets all data member values to represent Medusa class

Int attack(int)

- used to simulate Medusa’s attack by rolling 2 six sided die (generate random numbers between 1 and 6)
- special ability “Glare” is implemented in this function
  - if both random numbers generated are 6, Glare is used
  - if used, Medusa does 100 damage, representing Medusa turning the opponent into stone
    - this is enough damage to defeat any opponent, except when an opponent uses a special ability
  - is trumped by any opponent defense special ability

Int defend(int, int)

- used to simulate Vampire defend by rolling 1 six sided die (generate random number between 1 and 6)

All other functions are used from base class

HarryPotter class

Private: (data members)

Int hog

- used to keep track of the number of times Harry Potter's special ability "Hogwarts" is used

All other data members are the same same as base class

Public:

HarryPotter() (constructor)

- sets all data member values to represent HarryPotter class

Int attack(int)

- used to simulate Harry Potter's attack by rolling 2 six sided die (generate random numbers between 1 and 6)

Int defend(int, int)

- used to simulate Vampire defend by rolling 1 six sided die (generate random number between 1 and 6)

void damage(int, int)

- Used to simulate damage done to Harry Potter by subtracting armor points from damage done, then subtracting that difference from Harry Potter's strength points
- Harry Potter's special ability "Hogwarts" is implemented in this function
  - Hogwarts is only used 1 time per fight
  - Hogwarts is used when Harry Potter's strength points go below 1
  - Hogwarts revives Harry Potter and sets Harry Potter's strength points to 20
  - Data member "hog" is incremented to 1 after the first use of Hogwarts, ensuring that it cannot be used again in the fight

All other functions are used from base class

BlueMen class

Private: (data members)

Same as base class

Public:

BlueMen() (constructor)

- sets all data member values to represent BlueMen class

Int attack(int)

- used to simulate BlueMen attack by rolling 2 ten sided dice (generate random numbers between 1 and 10)

Int defend(int, int)

- used to simulate BlueMen defend by rolling 3 six sided die (generate randoms number between 1 and 6)
- BlueMen special ability "Mob" is implemented in this function
  - Unlike other Creature derived class's special abilities, special harms rather than helps BlueMen
  - When BlueMen strength points is under 9, defense dice roll goes down by 1 die in total (ie. from 3 six sided dice in the beginning, to 2 six sided dice)
  - When BlueMen strength points is under 6, defense dice roll goes down by 2 dice in total (ie. from 3 six sided dice in the beginning, to 1 six sided die)

All other functions are used from base class

<u>Tournament class:</u>

Private: (data members)

Int queueNum

- Holds the number of QueueNodes in Player 1 team's queue

Int queueNum2

- Holds the number of QueueNodes in Player 2 team's queue

Int score1

- Holds the score of Player 1's team (+1 for a win, +0 for a loss)

Int score2

- Holds the score of Player2's team (+1 for a win, +0 for a loss)

Queue queue1

- Holds the QueueNodes that hold a pointer to a Creature derived object

- Represents Player 1's team

Queue queue2

- Holds the QueueNodes that hold a pointer to a Creature derived object
- Represents Player 2's team

Queue queueL

- Holds the QueueNodes that hold a pointer to a Creature derived object
- Represents the Creature derived objects that lost in the tournament

Public: (Member Functions)

Tournament() (constructor)

- Creates a Tournament class object with all data members set to 0

Void attackSeq()

- Used by start() to simulate fight between 2 Creature derived class objects
- The first Creature derived class object in each Player's team queue is used
- Outputs intro for each fighter (Creature derived class object)
- Randomizes which fighter will attack first
- Uses loop of fighter1.attack(), fighter2.defend(), fighter2.damage() functions
    - Loop is reversed after each loop
- Outputs round fight commentary and summary
- Outputs final fight summary
- Places losing Creature derived object in losers queue and removes from Player team queue
- Places winning Creature derived object at back of Player team queue
- Regains some strength points for winning Creature derived object

Void queueMake()

- Uses the menu2() function to get user choice on how large each player's team should be
- Uses the chooseFighter() function to have user choose which Creature derived objects should be used for which team
- Uses the setCustomName() member function of the Creature derived object to name each Creature derived object

Void start()

- Starts and carries out the tournament for as long as each Player's team queue has at least 1 QueueNode holding a Creature derived object
- Uses attackSeq() function until a player has no Creature derived objects left in team

Void print()

- Displays the contents of each queue that Tournament holds

- Used only for testing the program

Creature* choosefFighter(int)

- Takes user input for what Creature derived object to create in integer form and dynamically allocates new Creature derived object corresponding to that integer
- Returns pointer to dynamically allocated Creature derived object

Void printLoss()

- Prompts the user for option to display list of Creature derived objects that lost the tournament (most recent loss first)
- Displays loser list if user chooses
- Delete all dynamically allocated memory in Tournament class

Void ending()

- Displays the results of the tournament
    - Which player won
    - Pointer per player team

Structure:

QueueNode

- Used by Queue class to make a queue
- Holds a pointer to the next QueueNode in the list, a pointer to the previous QueueNode in the list, and a pointer to a Creature class object

Functions:

Main()

- Uses Tournament() class member functions to facilitate a fighting tournament for as long as user does not choose to quit

Int menu()

- Used by makeQueue() function to retrieve and return user input in integer form based on displayed list of fighters
- Uses inputVal() function to validate user input as an integer and within bounds

Int menu2()

- Used by makeQueue() function to retrieve and return user input in integer form based on the amount of fighters per player team

Int menu3()

- Used by printLoss() function to retrieve and return user input in integer form of whether the user would like the list of Creature derived objects that lost in the tournament to be displayed

Int menu4()

- Used by main() function to retrieve and return user input in integer form of whether the user would like to start a new tournament or end the program

Bool inputVal(string)

- Used by menu() functions to validate input as integers and within bounds
- Parameter is the user input in string form
- Returns true if input valid

Testing plan

Since the Creature and Creature derived classes used for simulating a fight are the same from Project 3 and the testing plan for Project 3 already tested these classes for correct win percentage, these classes will not be tested for correct win percentage. The following tests are instead intended to assess the correct functionality of the queues and the correct functionality of the Tournament class. The input validation has been tested for data type and boundaries but is not included in the following table for readability reasons.

Test	Input Values	Expected Outcome	Observed Outcome
General Tests			
Determine if the correct number of fights is occurring per tournament	3 fighters for team 1 (any), 3 fighters for team 2 (any)	total points matches the total number of fighters in the loser queue and total number of fights	total points matches the total number of fighters in the loser queue and total number of fights
Determine if the correct number of fights is occurring per tournament when one team has less fighters	5 fighters for team 1 (any), 2 fighters for team 2 (any)	total points, total number of fighters in the loser queue, and total number of fights all match	total points, total number of fighters in the loser queue, and total number of fights all match
Determine if the correct fighter is regaining some strength points after a fight	1 BlueMen fighter for team 1, 3 Barbarian fighters for team 2	After every fight, a message should be displayed stating that the BlueMen for player 1 has gained some strength points	After every fight, a message displayed stating that the BlueMen for player 1 has gained some strength points
Determine if the fighter is regaining the correct amount of strength points after a fight	1 BlueMen fighter for team 1, 3 Barbarian fighters for team 2	For each new fight after fight 1, the BlueMen fighter should have at least 1 more strength point (but no more than the strength that was lost) compared to the end of the previous fight (this is shown in the intro of the fight)	For each new fight after fight 1, the BlueMen fighter had at least 1 more strength point (but no more than the strength that was lost) compared to the end of the previous fight (this is shown in the intro of the fight)
Determine if the correct fighters are being placed in the losers queue	3 fighters for team 1 (any), 3 fighters for team 2 (any)	Each fighter in the loser list displayed at the end of the tournament should correspond to a fighter that lost a fight during the tournament	Each fighter in the loser list displayed at the end of the tournament corresponded to a fighter that lost a fight during the tournament
Determine if the order of the loser list is correct	3 fighters for team 1 (any), 3 fighters for team 2 (any)	The first fighter in the loser list should have lost most recently, the second fighter should	The first fighter in the loser list lost most recently, the second fighter lost second most recently, and so on

		have lost second most recently, and so on	
Determine that no draws will occur	10 fighters for team 1 (any; random each time), 10 fighters for team 2 (any; random each time), done 3 times	There should be no draws in a fight or in a whole tournament	There are no draws in any fight or in any whole tournament
Determine that the correct number of points is being accumulated in a tournament	3 fighters for team 1 (any), 3 fighters for team 2 (any)	Total points per team should match total wins that team has	Total points per team matches total wins that team has
Determine if correct winner is being declared after tournament	3 fighters for team 1 (any), 3 fighters for team 2 (any)	Declared winner should be the player with the most points	Declared winner is the player with the most points
<u>Tests with each fighter instance</u>			
Determine that tournament program runs correctly with Medusa fighter	1 Medusa fighter for team 1, 2 BlueMen fighters for team 2	The Medusa creature should lose the tournament and be displayed in loser list, player 2 should be declared winner	The Medusa creature lost the tournament and was displayed in loser list, player 2 was declared winner
Determine that tournament program runs correctly with Barbarian fighter	1 Barbarian fighter for team 1, 2 BlueMen fighters for team 2	The Barbarian creature should lose the tournament and be displayed in loser list, player 2 should be declared winner	The Barbarian creature lost the tournament and was displayed in loser list, player 2 was declared winner
Determine that tournament program runs correctly with Vampire fighter	1 Vampire fighter for team 1, 2 BlueMen fighters for team 2	The Vampire creature should lose the tournament and be displayed in loser list, player 2 should be declared winner	The Vampire creature lost the tournament and was displayed in loser list, player 2 was declared winner
Determine that tournament program runs correctly with Harry Potter fighter	1 Harry Potter fighter for team 1, 2 BlueMen fighters for team 2	The Harry Potter creature should lose the tournament and be displayed in loser list, player 2 should be declared winner	The Harry Potter creature lost the tournament and was displayed in loser list, player 2 was declared winner
Determine that tournament program runs	1 BlueMen fighter for team 1, 2	The BlueMen creature should win the tournament, both	The BlueMen creature won the tournament, both Barbarian creatures were displayed in the

correctly with Blue Men fighter	Barbarian fighters for team 2	Barbarian fighters should be displayed in the loser list, and player 1 should be declared winner	loser list, and player 1 was declared winner

Reflection Statement

As Project 4 is a continuation of Project 3, I had originally planned Project 4 to not have a Tournament class, and instead use a series of independent functions (like the chooseFighter() and attackSeq() functions were used in Project 3). However, as the parameter list for each of these functions kept getting longer, it made more sense to group them under a Tournament class. This way the parameters only needed to be entered once and then the Tournament class could store them as data members for all the Tournament member functions to use.

For the list of fighters for player 1's team, the list of fighters for player 2's team, and the list of fighters that lost their fight in the tournament I decided to use the Queue class created in lab 7. 3 Queue class objects (named queue1, queue2, and queueL) are created and held as data members by the Tournament class that represent these 3 lists. The QueueNode struct is used by each Queue class object to create a doubly linked circular queue. Each QueueNode holds a pointer to the next QueueNode and a pointer to the previous QueueNode (as in lab 7). Each QueueNode also holds a pointer to a Creature object (this allows each QueueNode to be a placeholder for the creature in the queue). I had originally attempted to create the queues with each node being a Creature, and adding pointers to Creature to the Creature data members. However, I kept receiving segmentation fault errors and eventually made QueueNode the elements of each queue, with QueueNode having a pointer to Creature.

Because the player 1 team queue and the player 2 team queue both behave the same as the queue created for lab 7, I was able to use the addBack() and getFront() Queue class member functions already made for most of the functionality of these 2 queues. The getBack() function had to be created for the Queue class in order to access the last fighter in the queue to regain strength after a fight. Since the queue that holds the losing Creature class derived objects has to be added to from the front, I created a addFront() function for the Queue class. I also added a specific printLoss() function that is meant for printing the queue of Creature class objects that lost (this is because it displays wording that states that the contents displayed are from the loser's list

The program keeps the score of each player in the tournament with the score1 and score2 Tournament data members. When a fighter wins, that fighter's team gains 1 point. When a fighter loses,

that fighter's team gains 0 points. The score is added after each fight and each player's team total score is displayed after each fight, as well as after the entire tournament. At the end of the tournament, the team with the most points is declared the winner. Since the fights between the Creature derived objects never end in a tie, the tournament similarly never ends in a tie.

The regainStrength() function has been added to the Creature class in order to have a Creature object regain strength points after a fight (if that Creature derived object has won the fight). The Creature derived object that loses the fight does not regain any strength. The baseStrength() data member is used by the regainStrength() function to determine how much strength has been lost during the fight. Then, a pseudo random number is generated between 1 and the amount of strength lost, and is added to the objects strength. This way, the Creature derived object will never have strength above the starting strength. This also means that unless the winning Creature derived object did not take any damage in the fight, it will gain at least 1 strength point back.