Assignment 4(Reflection Document)

Woohyuk Yang

yangwo@oregonstate.edu

1. Requirements:

The requirement is to make a creature combat tournament program. Starting with the program made for assignment 3 to build each creatures. The user should be able to type how many fighters to fight for each team. The users(the first and the second) should be able to supply the line ups for each team. In the end the user should be able to choose whether he/she wants to see the list of the final order of all fights.

All kinds of creatures should inherit from the Creature class. The user should be able to choose the creature. Each object should be instantiated and put into each team's line up. Polymorphism should be used to control each instantiated objects.

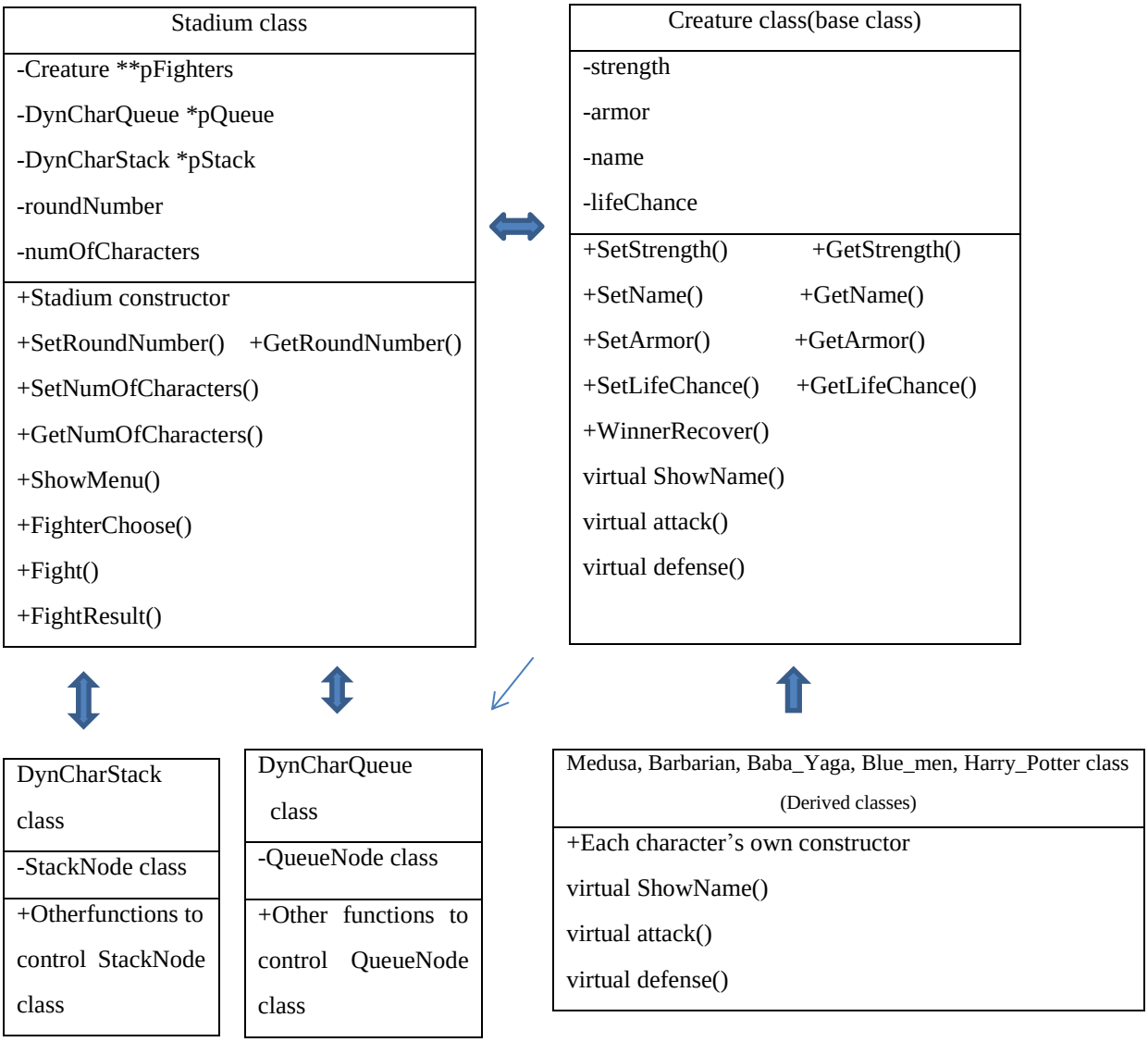
All creatures line up first as the user decided. The winner of each round goes to the back of the lineup while the loser gets piled on the stack. Loser who lost earlier should be located at more bottom place in the stack than a recent loser.

Structures made in module 2 (stack-like, and queue-like) should be used to store line ups for each team and for the stack which will store the losers.

There should also be a function which restore some percentage of the damage of the winners when the winners get sent to the back of the line. Any special ability which were gained during the fight should be reset also.

In the end, there should be some display of three finishers and which team won the tournament.

2. Class Design



Creature class has some features like strength Points, armor, name, and lifeChance. So other derived classes can inherit those features. This base class also contains some functions as member.

The class has accessor and mutation functions for strength and name variables. And ShowName() function which shows the name of each creatures.(access function for variable name) To use polymorphic features of C++ language, attack() function and defense() function were designed to be virtual functions. So even from derived classes like Medusa, Barbarian, Baba_Yaga, Blue_men, Harry_Potter classes can attack or defense in their own ways as the program is required.

When each objects get instantiated, they should be stored in queue-like structure. So each creatures get controlled with the functions controlling the queue-like structure in the program.

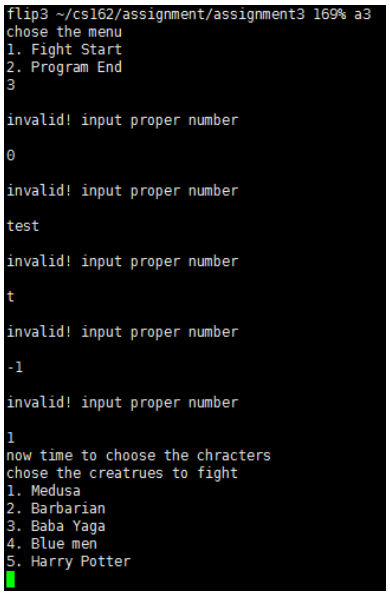
Stadium class is the class which allows two chosen creatures fight each other.

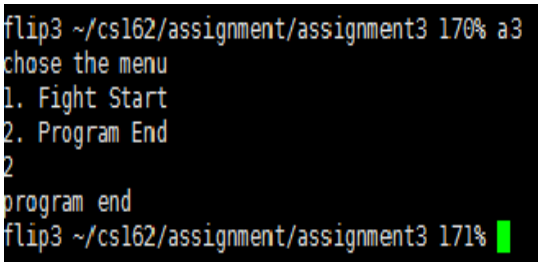
They are composed of functional functions deriving fight simulation. The first one is ShowMenu() function. This function lets user to start the fight or end the program. This will display the first show menu for the user. The second one is FighterChoose() function. When the user decides to do Fight simulation then he/she has to decide how many creatures get stationed for each team. And after that, first player chooses the fighters as many as the user decided before in an order and when the first player is done with that process, then the second player chooses the fighters in an order as many as the first player chose before.

Fight function makes each creatures fight until one of each gamer's queue is empty or the game gets too brutal(game seems that it will not end). FightResult function decides who did win or lose or tie. Based on the result the winner goes back to the queue and the loser goes to the stack. When the winner goes back to the queue then the function WinnerRecover makes the winner recovered and reset for the next fight.

3.Test Plan

When the program starts the user will see some menus he/she can choose the number. The user is only supposed to input '1' or '2' because '1' is for staring fight and '2' is to end the program. they type characters or invalid number then the program could crash. So I put validating process. As results below, the validating process worked fine and once I input '1' then I can choose the characters and if I input '2' then I can end the game.





```
flip3 ~/csl62/assignment/assignment4_1 183% a4
chose the menu
1. Fight Start
2. Program End
1
now enter the number of fighters
3
```

When user firstly choose to 1.Fight Start then the user can input how many fighters to choose for each team.

```
team 1 queue
q = 1 : strength = 10
  inside print queu
q = 2 : strength = 12
  inside print queu
q = 3 : strength = 12
  inside print queu
team 2 queue
q = 2 : strength = 12
  inside print queu
q = 3 : strength = 12
  inside print queu
q = 4 : strength = 12
  inside print queu
```

In the case when the user put '3' for each team, the user could type 3 characters for each team and 3 creatures for each team got stored in the queue like above picture.

I tested every cases when it comes to fight. There are 15 cases and I made it in a table. Some areas are black colored because there is same test plan for that.

VS.	Medusa	Barbarian	Baba Yaga	Blue men	Harry Potter
Medusa	success	success	success	success	success
Barbarian		success	success	success	success
Baba Yaga			success	success	success
Blue men				success	success
Harry Potter					success

```
Round 1 Start


stregnth points before round 1
Medusa has 10 strength points left
Medusa has 10 strength points left

 medusa attack 5
Medusa defense 3
Medusa damage is 0

 medusa attack 7
Medusa defense 5
Medusa damage is 0


Round 2 Start


stregnth points before round 2
Medusa has 10 strength points left
Medusa has 10 strength points left

 medusa attack 10
Medusa defense 4
Medusa damage is 3

 medusa attack 5
Medusa defense 1
Medusa damage is 1


Round 3 Start


stregnth points before round 3
Medusa has 9 strength points left
Medusa has 7 strength points left
```

The picture on the left is when I tested my first test case

Medusa vs. Medusa.

Medusa defense is randomly number of sum of dice rolled. And damage is the actual damage concerned with armor(each creature's own feature.)

I differentiated defense and armor to show that the Fight() function does good process with attacking and defensing and renewing Strength Points for each creature after each round.

I checked each cases do randomly result attack points and defense points, and calculate those well with armor and left Strength Points inside the function.

I had to check each character's special features. For medusa's glare case, I put integer '12' directly to

the variable representing the attack point and tested whether it works and it worked well. For Baba yaga's soul, I checked the value is well calculated and I also tested if fighter's strength Points go over "certain point" then it makes the match draw. And if the round number goes over " certain number" then it makes the match draw also. I set certain number like 10 instead of 100 only for the test. The matter is the process works well not the number itself. Because there is no change of function itself based on number, so if it works with small number then it would work well with bigger number. For blue men's Mob feature, I put some "cout" statement inside each if statement to check on which condition the function works with. I could check it works fine following its strength points also. Harry's Hogwarts feature could be checked when I first set 0 to harry's strength point so harry potter revives again and used GetStrength function to see if harry's strength points has been assigned properly. I used the same fuctions for attack and defense for each creatures so I could make it sure again that those fight mechanism works well still with my new other functions.

For FightResult() function, I checked whether each cases work properly with GetStrength() function. Threre are 6 cases here. The first is when both creatures dies, the second is the first creature dies and loose, the third is the second creature dies and loose, the fourth is both do not die, the fifth is strength points of both creatures go over certain number(100 in this program) so it makes the game draw. And the last one is the round number goes over certain number(100 in this program) so it makes the game draw. For the last two cases, I set smaller numbers to each to verify it works properly more easily but still accurately. The result was all worked successfully. This time I made each cases return different values so it got easier to differentiate which situation is going on now.

```
Do you want to see the result of the game?
1. Yes
2. No
```

When the tournament ends, the user can decide to see the result or not. If he/she chooses no then the program will end. If he/she chooses 1.Yes then the user will be able to see the result.

```
Do you want to see the result of the game?
1. Yes
2. No
1
the first and second and third finishers are
this queue is empty
the number of creatures stored in team 1  = 0
the number of creatures stored in team 2  = 3

team 1 queue
this queue is empty
this queue is empty
this queue is empty

team 2 queue
q = 2 : strength = 20
  inside print queu
q = 3 : strength = 12
  inside print queu
q = 4 : strength = 22
  inside print queu
sorted queue = 22
sorted queue = 20
sorted queue = 12
this queue is empty
this queue is empty
player 2's team has won! Congrats!
loser piled stack
#1 = Baba Yaga : strength : -1
#2 = Barbarian : strength : -1
#3 = Medusa : strength : -1
```

you can see which team did win over and which creatures are left on which team's queue(line up) and which creatures are piled in the stack.

3. Reflection

I changed my design doing my implementation.

At first, I had hard time controlling queue which is based on doubly linked list. I lacked the experience to implemented doubly linked list and used it with different types of variables. The implementation looked well but sometimes memory leaks happened so the result was having a coredump with memory segmentation fault. When memory allocation is necessary, it is important to deallocate the memory appropriately.

I made each winners for the each round gets twice of their remained strength points for the next fight. Before they get sent to the back of the queue like structure I applied WinnerRecover member function to each winner, so winners could benefit from that member function.

Most of creatures do not have any specific abilities when it comes to strength points, so those creatures were easy to deal with. The problem was when Baba yaba and the other Baba yaba met in the other team. When they fought they gained strength points whenever they attack so the fight goes permanently most of time. So I wrote a code to send baba yabas to the stack when they meet in the fight.

I made the tournament to go till one of team gets exhausted and there is no more team member left to fight. So in that case the other team(having at least one character to fight in the queue) wins over. When there is no character left in the both teams then that tournament is tie.

Printing the stack, I printed recently lost loser first and the loser who had lost earlier later. So recent loser will be #1 and #last will be the loser who lost long before.

After every match, the user can choose whether to watch the result of the game or not, when the user click ‘2’ then the user can not watch the result while clicking ‘1’ and being able to watch the result of the tournament like which team won over and which creatures are piled in the stack.