Sarah Newport
Assignment 4
2/27/15

Report

Discovering Requirements

This assignment is designed to develop skills in using appropriate data structure containers from the STL and polymorphism. The main focus will be on correctly implementing containers from the STL to hold lineups of character subclasses and characters that have lost a match. To be successful in this program, it will be important to correctly assess the requirements for the program, plan how to best fulfill them, and also plan testing methods. Once the design plan has been finalized and implementation is begun, the focus will shift to constant testing throughout the process. The following items are the assignments requirements.

- Maintain a consistent programming style and documentation(Comments in the code)
- Create tournament, create a lineup of fighters, resolving the round, and putting the two fighters in appropriate containers.
- Prompt the users to fill the two lineups based on the number of fighters entered by the user
- Create and properly use the containers to hold the lineups and loser piles
- Create a heal or recovery function for the winners of each round
- Create and implement a system to determine the final three finishers as well as determine which side won the tournament

Design

In order to correctly implement STL containers, inheritance, and polymorphism, my design must focus on object oriented development. I will be building upon the program structure from the last assignment and will only need to design the pieces of code that will implement the new requirements. The following is my design plan on how to best implement the new requirements.

After prompting the users for the number of fighters and names of their teams, a for loop will ask each player to choose a character until the inputted number of fighters has been met. The characters will be placed in the STL container called a queue. (There will be one queue for each user.) This is due to the queue being a First in, first out mechanic. The first fighter that the user chooses will be the first fighter to participate in the tournament. Both queues will then be passed to the Combat function that will initialize “Character * player” to the character at the front of each queue. I will then pop both queues to remove those characters. At the end of the round, the winner will then call on the recover function to regain some of their lost health. I plan to do a 50/50 chance of recovering half the damage they lost. However, the recover function will for sure heal for at least 1 strength

point as a reward for winning. The winning character will then be put back into the queue. The loser character will be placed into a vector. This will repeat until one team has been completely eliminated. The surviving characters will then be transferred into the loser vectors. This is to be able to iterate though both vectors to sum of the points of each character. The point system will be +2 points for a successful attack, +1 point for a successful defend, and +3 points for winning a match. Once both vectors for each team has been iterated through to sum of the total points and number of wins, the results foreach team will be displayed and the team with the most points will be declared the winner. (This means that a team can be annihilated but still win the tournament based on points.) Both vectors will then be joined and iterated through to find the top 3 characters with the most points. The first, second, and third place characters will then be displayed with the number of points and wins they accumulated.

Implementation:

Please see various source code files.

Testing and debugging

For the majority of the implementation, increment development was used by testing each piece after it was finished by building and running it. The toughest problem that was encountered was iterating through the vectors in determining the top three characters. It would work most of the time, but there seemed to be specific cases (different character match ups) that would cause it to display incorrect results. After trying to discover these specific circumstances, I decided to rewrite a new function to determine the top 3 winners that was much more effective. Another problem that was encountered was that the code compiled and ran well in visual studio but not so much when transferred to the FLIP server. I had to rewrite some small sections of code, but got it working in the end. I used the character fight statistics from the test driver made for the last assignment to test every character match up for the lineups. The results were as expected. (Please see unit testing data at end of the report.)

Reflection:

I learned how to better handle character pointers and passing them to other functions. I really enjoyed implementing this program as the STL containers proved to be so beneficial! During implementation, I tried to use a stack container for the loser piles but decided that vectors would be best since I would be using them to also determine the tournament results. The implementation went well and was true to the design. After the requirements were implemented, I added some extra features such as a huge list of names to be randomly assigned to the characters participating in the tournament. After looking back on my code, I think I need to practice more efficient encapsulation of the different classes. The testing and debugging phase was very involved for this program. I had to add a check that the number of fighters for each team was three or more so that the top three placements had more meaning. I spent lots of time testing as many of the possible lineups of characters. This assignment has definitely helped me better understand STL

containers and polymorphism and am excited to incorporate them into a side project such as an RPG game.

I used the following unit testing data to aid me in testing the different lineups:

```
Goblins vs Goblins
-------------------------
1000 battles each as the attacking player:
Goblin wins: 49.85%
Goblin wins: 50.15%
Barbarians vs Barbarians
-------------------------
1000 battles each as the attacking player:
Barbarian wins: 48.5%
Barbarian wins: 51.5%
Reptile People vs Reptile People
-------------------------
1000 battles each as the attacking player:
Reptilian wins: 49.95%
Reptilian wins: 50.05%
Blue Men vs Blue Men
-------------------------
1000 battles each as the attacking player:
Blue men wins: 50.2%
Blue men wins: 49.8%
The Shadow vs The Shadow
-------------------------
1000 battles each as the attacking player:
Shadow wins: 48.15%
Shadow wins: 51.85%
Goblins vs Barbarians
-------------------------
1000 battles each as the attacking player:
Goblin wins: 24.8%
Barbarian wins: 75.2%
Goblins vs Reptile People
-------------------------
1000 battles each as the attacking player:
Goblin wins: 0%
Reptilian wins: 100%
Goblins vs Blue Men
-------------------------
1000 battles each as the attacking player:
Goblin wins: 0%
Blue men wins: 100%
Goblins vs The Shadow
-------------------------
1000 battles each as the attacking player:
Goblin wins: 4.45%
Shadow wins: 95.55%
Barbarians vs Reptile People
-------------------------
```

1000 battles each as the attacking player:
Barbarian wins: 0%
Reptilian wins: 100%
Barbarians vs Blue Men

1000 battles each as the attacking player:
Barbarian wins: 0%
Blue men wins: 100%
Barbarians vs The Shadow

1000 battles each as the attacking player:
Barbarian wins: 12.1%
Shadow wins: 87.9%
Reptile People vs Blue Men

1000 battles each as the attacking player:
Reptilian wins: 11.4%
Blue men wins: 88.6%
Reptile People vs The Shadow

1000 battles each as the attacking player:
Reptilian wins: 88.7%
Shadow wins: 11.3%
Blue Men vs The Shadow

1000 battles each as the attacking player:
Blue men wins: 93.25%
Shadow wins: 6.75%