OOP Fight Arena Game

Project Idea

This will be a two-player based game where the player gets to choose between four types of characters and fight to win by getting opponent's health to zero. Players will have the following attributes: Health, Armour, Mana, and some hidden magical powers (depends on the selected player type). Using different types of attacks, the players will either defend or attack the opponent stats.

Features of the Game

1. Memory Allocation

Player: Polymorphism is applied to create a player from 4 different children classes, and the memory is dynamically allocated.

Stats of the Player: The stats of both players (Health, Armour, Xp) are dynamically allocated to their player class. However, during the duration of the program two copies of the stats are prepared in the ground class that are passed as arguments to perform the attacks, and then are updated in the player using the updateStats behaviour.

Player Name: stored in the player class as stack memory, as it is only called when the leader board is displayed.

Leader board: stack memory stored in the ground class to update the player points after each round has been played. It is initialised as a 2-unit int array to both values at 0. After competition of all the rounds, the ground class calculates from the leader board array, who the winner of the overall game is.

Other: anywhere in the program if any local memory is needed, then it has been called on stack. However, if a memory was dynamically allocated, then has been freed before the termination of the program.

2. Polymorphism & Inheritance

Abstract class of player has been made the parent class from which four different children classes (archer, defender, medic, and mechanic) inherit and overwrite some of the behaviours to show some individual characteristics. The memory is dynamically allocated each time. It has been declared as **Player* p_name = new child_class;**. The parent class has variety of behaviours and stores the name of the

player and an array of stats. All these variables and behaviours are inherited by the children classes, while having individual behaviours.

For an example, if user selected to be a player of type archer, then after each 7 turns, a random number between 1 to 4 will be generated which give either of the four magic powers defined previously. Still the player can get unlucky and roll 3, thereby reducing mana by 5 units.

3. **Attacks**

gunshot('g'): If a player fires a gunshot, then the rival's health gets reduced by 15 points and armour by 5 units. The player receives 20 units (Mana) for firing a gunshot.

sword('s'): To shoot a gunshot, a player needs at least 10 Mana points. Shooting a sword reduces the rival's health by 30 points and armour by 10 points. However, to shoot a sword, it costs the player 10 units of Mana.

Skip('S'): You can skip a move anytime. The player gets 25 points of Mana for skipping.

medical('m'): When your health goes below 20 units, then and then the player can use the medical. It gives the player 15 units of health; however, it costs the player 5 units of armour and mana, respectively.

cannonball('c'): the player needs exactly 100 units of Mana to fire a cannonball. It decreased the rival's health by 40 units and armour by 30 units. However, the player's mana gets down to 0.

Show that your code is inside the SVN system.

- It will be displayed in the code and the terminal window

Be able to build it from your makefile

- Shown in the svn directly and compiled

Be able to describe (and preferably demonstrate) two test situations.

- **Class Attacks:** The class attacks has been thoroughly tested with every possible outputs. The process used for testing this class was **Unit Testing**. The class does not have any variables, however, has 5 different behaviours that performs the attack.

The five types of behaviours tested are gunshot, sword, Skip, medical, and cannonball. Only two arguments of two player arrays of length 3 are passed and checked if the output is equal to the expected output.

- **Class Security:** This class acts as a security class which checks if the player can take the above-mentioned attacks. It has four behaviours and a variable of array that initialises the attacker's stats. The test cases are prepared in a way to test the boundary cases. **Unit Testing** is applied, an output is already generated that checks if the input from the run file matches the expected output.

  1. checkSword(): it checks if the attackers Xp is greater than equal to 10
  2. checkMedical(): it checks if the attackers health is less than 20 units
  3. checkSkip(): it checks if the health of the player is more than 20 units
  4. checkCanon(): it checks if the player has 100 Mana to fire a canonball

Outline your plan for finishing work, with allocated tasks to the group.

- Commenting every file that has been created
- Fixing any bugs in the program
- Modifying the style of the code
- Have clear statement for each class created and its relation to other classes