CS200 Introduction to Programming
Spring 2022
Programming Assignment 4

Guidelines

Rules:

1. The object is not simply to get the job done, but to get it done in the way that is asked for.
2. Any cheating case will be reported to the Disciplinary Committee without any delay.

Coding Conventions:

- Constants are “ALLCAPS” or “ALL_CAPS”.
- Variables are “allsmall” or “all_small”.
- All function names must be “firstWordSmallAllOtherWordsCamelCase”.
- All class names must be “CamelCaseWords”.
- All curly brackets defining a block must be vertically aligned.
- Use tabs where needed instead of spaces.
- File naming: RollNumber_PA4.zip

Course Learning Objective (CLOs):

1. CLO1: Enabling Knowledge:
    a. (C1) use object-oriented programming models: abstract data types, encapsulation, inheritance, and polymorphism to code algorithmic solutions using standard coding conventions.
    b. (C1) use of fundamental features of an object-oriented language like C++.
2. CLO2: Critical Thinking and Analysis:
    a. (C4) analyze the requirements for solving simple algorithmic problems.
3. CLO3: Problem Solving:
    a. (C6) design algorithms and implement program code in an object-oriented programming language such as C++ to solve simple algorithmic computing problems, based on the analysis of the requirements.
    b. (C5) evaluate the correctness of the proposed solution.
4. CLO4: Communication:
    a. (C2) explain key concepts of algorithmic design in written form.
5. CLO5: Responsibility:
    a. (C3) apply relevant conventions, standards, and ethical considerations to writing computer programs.

Task : A Little Advanced Turn-based Combat Game Marks:100

Your task for this assignment is to create a turn-based combat game based on all the concepts you have learnt so far. So you will create a Pokemon-esque game with multiple characters, movesets. In this assignment, the details of class implementations will be slightly more open ended and you will also get marks for good Class structure and design based on what you have learnt throughout this course. You will be required to design a number of classes, the names of which are given below.

1. **Game** class: to manage flow of the game.
2. **Player** class: to allow all actions a player can do.
   Character class (a class to represent game characters)
3. **Move** class: This is an abstract class to represent moves which characters make.
4. 4-10 classes derived from the Move classes. These are moves which characters can directly use.
5. **Item** interface class: Items are things that can affect a player in some way. These are single use items. This class has a private variable **name** and a single PVF use() and it will be defined in all derived classes.
6. 3-5 derived classes from **Item**. These are different items. They can either restore health, boost defense or boost attack. (change the defense or attack booster variable in character). Note that attack and defense boosters will only be active for one move for ease.

Further Details are given below:

Game class:

The gameplay will go as follows:

1. A game screen is displayed at the start.
2. A player has three options in a turn
   a. Attack
   b. Switch character
   c. Use Item (skips the player's turn)
3. Switch character will switch the current character based on user choice.
4. Use Item will apply some **Item** to a living character.
5. The player will be shown a list of available moves.
6. The player will choose the move and the move will affect the other character in some way. The details of the moves will be given below.

The game class will be required to do three things:

1. Display the menu screen.

2. Start the game. This would involve displaying the relevant character selection options for both players, and starting the game. You can create a static array in the game class to save a list of all characters.
3. Implement a turn function based on the functionality given above.
4. Figure out the winner of the game.

Additional details:

1. There will be 2 players.
2. There are 2 types of games. 1v1 and 3v3. In 3v3 games for example, each player will have 3 characters.
3. Moves incur damage and the game ends when all of the characters belonging to a player die. That is, their health becomes 0.

The player class

The player class will hold all the **characters and items in arrays** (details given later). It will have a variable int **active_character** to indicate the current character. The character array will be implemented in the following way.

1. If a character dies, it's status will be changed to dead.
2. The active character will change to the next character. If the last character dies, then the active character will shift to the first character.
3. If a player chooses "swap character" then the active character will be changed to the character chosen if it is a valid option.

The items array will be a **fixed** size array (size = 3) of single-use items. It will hold objects of the Item class
(discussed below).
The method **useItem** will take as input an index of the item, and apply that item to the active character. The effects of the items are described in the **Item** section.

The Character class

The character class will be a class with private variables **String name, String type, String weak_type, double max_health, double current_health and String status**(alive or dead). It will also hold a private array, **moves** of type **Move**. Details of the **Move** class will be given in a later subsection. Moreover, it will have two variables, **attack_booster** and **defence_booster.** These are of type double and will be used as multipliers. For example if a character is hit with an attack, the attack will be divided by the defence_booster. It is 1.0 by default but can increase based on use of items. Similarly if a character attacks their opponent, the total damage will be multiplied by the attack_booster. The Character class will also have a function move(). This will take in an index to the move array and a reference to another character object. The function will apply the move at the index to the referenced character. The type variable will store the character type. These can be (fire, water, earth, lightning, wind). This relates to the **move** type as well. A character will have a weak_type as well. And the character will incur double damage from attacks of that type.

The Move class + derived classes

Move will be an abstract class. It will have three private variables: **type** (see Character sub-section), **name,** and **String use_type**. It will also have the PVF **use**(). 'Use' will be implemented in the derived classes. And will detail the effects of the move. Note that effects of the move can involve different things. It can be a normal move which incurs damage. It could also be a move which decreases the other character's defence_booster. E.t.c. The implementation in the derived classes is up to you. **Use_type** can be 'once', or 'infinite'. Also be sure to implement the weak against case as described in the character class above.

Item class + derived classes

An Item is a game object which is held by a player and can be used on a character. It is an interface class. And will only have one PVF **use**(). The use function needs to be overridden in the derived classes. The function can be used to restore health, boost defense or boost attack. (change the defense or attack booster variable in character). Note that attack and defense boosters will only be active for one move for ease. You can choose different values for health boosts e.t.c in different derived items. Again, the implementation of the use function in the derived classes is open but needs to do some of the things mentioned above.

Note: you must handle all exceptions in calculations and inputs. These involve invalid operations such as dividing by zero. Or actions which are not allowed, such as invalid inputs.
Note: You must not use inline functions.
Note: Try to make the game playable with appropriate menus, prompts and formatting.

Marks Distribution

1. Game class made with correct variables [25]

   Tasks:

- Correct constructors/setters/getters (5)
- implementation of attack switch character and use item (10)
- Correct display of options and menu (5)
- Correct implementation of game types 1v1 and 3v3 (5)

2. Correct Player and Character Class functions [25]

   Tasks:

- Correct setters constructors (5)

- Correct implementation of character attack function (10)
- Correct implementation of attack and defense boosters + weak_type (5)
- Correct class Design (5)

3. Move Class functions [15]
4. Item Interface class [10]
5. Correct Game Flow output [10]
6. Error handling and exceptions [10]
7. Coding Conventions and Comments [5]

Note: After implementing the Game write a short explanation about the Game and moves of the games in comments of your code.

Bonus Part Marks:10

The game you have made until now is probably text based. However, if you want to go one step further and make it more playable, You could create a sort of UI on the terminal. This could be a screen which shows player names, current characters, a health bar, items, e.t.c. It could also show very basic character design, as well as messages like. ‘X used scratch, It’s super effective” (not plagiarized). Menu and Input prompts could be shown on the lower part of the screen. Anyway, this is just a suggestion, make anything you want and if it’s cool enough you will get bonus marks :). **Also don’t focus on this before completing the rest of the assignment. Functionality before Appearance.**

NOTE: Total marks remain 100 even including the bonus marks.

****

Good Luck !

****