Lab 6 [Enemy Generation]

In this workshop, we will continue our development of a text based game "Code Quest!". This workshop focuses on input buffer clearing and the use of C library functions.

LEARNING OUTCOMES

Upon successful completion of this workshop, you will be able to

- Validate and clear the input buffer of garbage
- Use library functions to perform random number generation

PART 1:

SPECIFICATIONS

Write a program that acts as an enemy generator for CodeQuest! This program generates unique random enemy stats for each enemy encounter. This program also features more advanced input verifications that will be used to further develop the game. This program will require the following three functions:

`void SeedRandom()`: This function seeds the C random number generator with the current time of the system (to ensure unique numbers between program executions). This function receives no parameters and returns nothing.

`float GetRandom(float low, float high):` This function returns a randomly generated value that is between two values low and high. This function accepts two floats (high and low range) as parameters and returns a float.

`void ClearInputBuffer():` This function continuously consumes characters in the input buffer until it consumes a newline character. This function is used when the user has entered an invalid input and the buffer must be cleared before normal operations can be resumed. This function receives no parameters and returns nothing.

Once the above three functions are implemented, follow the following steps to write a program that randomly generates a collection of enemy stats and displays them to the user.

1. When the program starts, it calls the **`SeedRandom`** Function to prepare the generator.
2. The program then calls the **`GetRandom`** function to generate four random ratio values for each of the following stats: StrengthRatio, SpeedRatio, DefenseRatio, and IntelligenceRatio. The ratio values must be between 0.15 and 0.5.

3. The program then calls the `GetRandom` function to generate a random value between 0.5 and 2.0 as the HP ratio.
4. Finally, the program will randomly determine the level of the enemy between 3 and 20.
5. To determine the final stats of the enemy, multiply each stat ratio with the squared value of the level, truncate the final value, and store the result. The HP stat should simply be calculated by multiplying the HP ratio with the level of the enemy, truncating the final value.
   For example, given *Level = 12* and *Strength Ratio = 0.3, Strength stat = 0.3 * 12 * 12 = 43*).
6. Display the finalized stats to the user and prompt the user if they would like to generate another enemy (1 - Yes, 2 - No). If the user enters anything but 1 or 2, clear the input buffer and prompt them again.

The output of a typical run-through of your program should look like this (user's input highlighted in green). All values are randomly generated thus outputs will not match exactly:

```
Enemy Generator

 Level        - 3
 HP           - 1
 Strength     - 1
 Speed        - 2
 Defence      - 4
 Intelligence - 3

 Generate Another? 1

 Level        - 20
 HP           - 40
 Strength     - 200
 Speed        - 160
 Defence      - 120
 Intelligence - 60

 Generate Another? RB23214L
 Invalid Input 4
 Invalid Input 1

 Level        - 12
 HP           - 14
 Strength     - 28
 Speed        - 64
 Defence      - 21
 Intelligence - 72

 Generate Another? 2
```

If your program's output exactly matches the output shown above, given the provided inputs then your lab is complete and ready to be submitted (read below).

PART 2:

BUILDING THE GAME (BONUS)

If you have completed all previous game labs, you may complete this section in order to merge your current lab with you previous labs.

Copy the functions (and prototypes) into your w6 source code file. Delete the display and looping code from this lab and copy the stat generation component of this lab into the battle sequence function from the previous lab. Remove the hard-coded enemy stats and replace them with this code so that enemies will be randomly generated each time the battle sequence function is called. This adds dynamic enemies so you are not battling the same enemies over and over again!

Alter the battle sequence function so that the enemy will randomly choose between using physical attacks and magical attacks (add the required calculations for enemy magic attacks based on the equation used for the player's magic attacks). Further alter the code so that instead of the player always attacking first, the combatant with the higher speed attacks first. This will require a minor bit of reorganizing your code!

Finally, alter the Get Valid Input function so that it clears the input buffer if invalid input is entered by the user.

SUBMISSION

Upload your solution according to your instructor’s guidelines.