CSE 1321L: Programming and Problem Solving I Lab

Assignment 4

Loops

Assignment Outcomes

By completing this assignment, students will be able to:

1) Design programs that leverage loops to solve problems
2) Apply concepts from earlier in the semester, including conditional statements
3) Generate and use random numbers
4) Solve problems of increasing complexity

Program 1: Stay on the Screen! Animation in video games is just like animation in movies – it's drawn image by image (called "frames"). Before the game can draw a frame, it needs to update the position of the objects based on their velocities (among other things). To do that is relatively simple: add the velocity to the position of the object each frame.

For this program, imagine we want to track an object and detect if it goes off the left or right side of the screen (that is, it's X position is less than 0 and greater than the width of the screen, say, 100). Write a program that asks the user for the starting X and Y position of the object as well as the starting X and Y velocity, then prints out its position each frame until the object moves off of the screen. Design (pseudocode) and implement (source code) for this program.

Sample run 1:

```
Enter the starting X position: 50
Enter the starting Y position: 50
Enter the starting X velocity: 4.7
Enter the starting Y velocity: 2
X:50    Y:50
X:54.7  Y:52
X:59.4  Y:54
X:64.1  Y:56
X:68.8  Y:58
X:73.5  Y:60
X:78.2  Y:62
X:82.9  Y:64
X:87.6  Y:66
X:92.3  Y:68
X:97    Y:70
X:101.7 Y:72
```

Sample run 2:

```
Enter the starting X position: 20
Enter the starting Y position: 45
Enter the starting X velocity: -3.7
Enter the starting Y velocity: 11.2
X:20    Y:45
X:16.3  Y:56.2
X:12.6  Y:67.4
X:8.9   Y:78.6
X:5.2   Y:89.8
X:1.5   Y:101
X:-2.2  Y:112.2
```

Program 2: Buh-RING IT! For this assignment, you're going to simulate a text-based Role-Playing Game (RPG). Design (pseudocode) and implement (source) for a program that reads in 1) the hero's Hit Points (HP – or health), 2) the maximum damage the hero does per attack, 3) the monster's HP and 4) the maximum monster's damage per attack. When the player attacks, it will pick a random number between 0 and up to the maximum damage the player does, and then subtract that from the monster. The same thing happens when the monster attacks the hero, but damage is to the hero. The program should display rounds and the HP of the hero and monster each round. If the hero or monster dies, it should print that this happened and should NOT continue (i.e. no extra text). To learn how to create random numbers, see the appendix.

Sample run 1:

```
Enter the hero's starting hit points: 50
Enter the damage the hero's weapon does per strike: 20
Enter the monster's starting hit points: 40
Enter the monster's damage per strike: 15
====== ROUND 1 ======
Hero attacks for: 10
Monster has 30 HP left
Monster attacks you for: 1
You have 49 HP left
====== ROUND 2 ======
Hero attacks for: 18
Monster has 12 HP left
Monster attacks you for: 7
You have 42 HP left
====== ROUND 3 ======
Hero attacks for: 0
Monster has 12 HP left
Monster attacks you for: 14
You have 28 HP left
====== ROUND 4 ======
Hero attacks for: 18
Monster has -6 HP left
The monster dies and you earn 5 XP
Battle ends...
```

Sample run 2:

```
Enter the hero's starting hit points: 50
Enter the damage the hero's weapon does per strike: 10
Enter the monster's starting hit points: 40
Enter the monster's damage per strike: 20
====== ROUND 1 ======
Hero attacks for: 1
Monster has 39 HP left
Monster attacks you for: 6
You have 44 HP left
====== ROUND 2 ======
Hero attacks for: 5
Monster has 34 HP left
Monster attacks you for: 1
You have 43 HP left
====== ROUND 3 ======
Hero attacks for: 8
Monster has 26 HP left
Monster attacks you for: 8
You have 35 HP left
====== ROUND 4 ======
Hero attacks for: 4
Monster has 22 HP left
Monster attacks you for: 5
```

```
You have 30 HP left
====== ROUND 5 ======
Hero attacks for: 7
Monster has 15 HP left
Monster attacks you for: 1
You have 29 HP left
====== ROUND 6 ======
Hero attacks for: 7
Monster has 8 HP left
Monster attacks you for: 9
You have 20 HP left
====== ROUND 7 ======
Hero attacks for: 0
Monster has 8 HP left
Monster attacks you for: 14
You have 6 HP left
====== ROUND 8 ======
Hero attacks for: 4
Monster has 4 HP left
Monster attacks you for: 11
You have -5 HP left
You are killed by the monster and lose 10 gold.
Battle ends...
```

Program 3: Give a baby $5,000! Did you know that, over the last century, the stock market has returned an average of 10%? You may not care, but you'd better pay attention to this one. If you were to give a newborn baby $5000, put that money in the stock market and NOT add any additional money per year, that money would grow to over $2.9 million by the time that baby is ready for retirement (67 years)! Don't believe us? Check out the compound interest calculator from MoneyChimp and plug in the numbers!

To keep things simple, we'll calculate interest in a simple way. You take the original amount (called the principle) and add back in a percentage rate of growth (called the interest rate) at the end of the year. For example, if we had $1,000 as our principle and had a 10% rate of growth, the next year we would have $1,100. The year after that, we would have $1,210 (or $1,100 plus 10% of $1,100). However, we usually add in additional money each year which, for simplicity, is included before calculating the interest.

Your task is to design (pseudocode) and implement (source) for a program that 1) reads in the principle, additional annual money, years to grow, and interest rate from the user, and 2) print out how much money they have each year. Task 3: think about when you earn the most money!

Lesson learned: whether it's your code or your money, save early and save often…

```
Sample run 1:
Enter the principle: 2000
Enter the annual addition: 300
Enter the number of years to grow: 10
Enter the interest rate as a percentage: 10
Year 0: $2000
Year 1: $2530
Year 2: $3113
Year 3: $3754.3
Year 4: $4459.73
Year 5: $5235.7
Year 6: $6089.27
Year 7: $7028.2
Year 8: $8061.02
Year 9: $9197.12
Year 10: $10446.8
```

```
Sample run 2 (yeah, that's $9.4MM):
Enter the principle: 5000
Enter the annual addition: 1000
Enter the number of years to grow: 67
Enter the interest rate as a percentage: 10
Year 0: $5000
Year 1: $6600
Year 2: $8360
Year 3: $10296
Year 4: $12425.6
Year 5: $14768.2
.
.
Year 59: $4.41782e+06
Year 60: $4.86071e+06
Year 61: $5.34788e+06
Year 62: $5.88376e+06
Year 63: $6.47324e+06
Year 64: $7.12167e+06
Year 65: $7.83493e+06
Year 66: $8.61952e+06
Year 67: $9.48258e+06
```

Submission:

Part 1: Pseudocode:

1. Review the assignment submission requirements and grading guidelines.
2. Upload the pseudocode files (Word doc or PDF) to the assignment submission folder in Gradescope.
3. The files must be uploaded to Gradescope by the due date.
4. The Pseudocode must be complete and following the standards listed on the FYE website.

Part 2: Source Code:

1. Review the assignment submission requirements and grading guidelines on the FYE website.
2. Upload the source code files to the assignment submission folder in Gradescope.
3. The files must be uploaded to Gradescope by the due date.

Appendix

Random numbers: for most languages, you have a special function you can call to get a random number. Often, this returns a value between 0.0-1.0 that you then have to multiply by the range you want. For example, if you multiply a number between 0.0-1.0 by 30, you get a result between 0.0-30.0. However, because Java and C# are Object-Oriented Programming languages, there are special objects you have work with, each of which have multiple (useful) methods.

C++ is funky. You call rand() and get a large number between 0-RAND_MAX, but every time you run the program, you get the same number! So, we have to "seed" the random number generator with something that is different each time we run our program (in this case, the time since January 1st, 1970). To get a number between 0.1 and 1.0, you have to divide that number by RAND_MAX.

In Java, it looks like (import java.util.Random;) :

```
Random generator = new Random();
float randNum = generator.nextFloat();
int randNum2 = generator.nextInt(10);
System.out.println (randNum + " " + randNum2);
```

While in C#, it looks like:

```
Random generator = new Random();
int randNum = generator.Next(1, 10); // 1-10
float randNum2 = (float)generator.NextDouble()*30; // 0.0-30.0
Console.WriteLine(randNum+" "+randNum2);
```

In C++ (and EVERYONE SHOULD READ THIS):

```
#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;
int main() {
  // Here, we'll get the same number each time we run the program
  // A big number between 0 and RAND_MAX.  You should seed it first!!!
  // Note: this is just to show you get the same number each time.
  int number = rand();
```

```
  cout << number << endl;


  // Give the random number generator a "seed"
  // based on time.  Now, we get different random numbers
  srand(time(NULL));
  number = rand();
  cout << number << endl;

  // To get a value between 0 and "n", mod by (n+1)
  // Here, we get a number between 0-9
  int newValue = number % 10;
  cout << newValue << endl;

  // To get a value between 0.0 - 1.0, we have to divide
  // through by RAND_MAX.  We have to type cast to a double
  // because dividing an int by an int returns an in
  double newValue2 = number/(double)RAND_MAX;
  cout << newValue2 << endl;
}
```