Lab 11: Displaying Multiple Characters and Randomly Generated Characters

Introduction

A key component to video games is the presence of an obstacle or challenge which the player must overcome. The simplest form this challenge can take is the mere presence of an enemy character. When designing a game, care must be taken to ensure that the generation of enemies does not follow a predictable pattern, or the game loses its challenge. This week, you will use the STM32L100RC microcontroller to generate a random pattern of enemy characters that will move across the screen.

Random Number Generation in C

Random number generation in electronics has long been a topic of research and investigation. The standard practice is to use a random number generation function that relies on a seed number. When writing software to implement this, the programmer will usually provide the seed number based on the current time when the program is executed; this allows for theoretically random number generation, since the seed number is different for each execution of the program. However, the STM32L100RC does not support the `time.h` library necessary to use the current time as the seed value.

In order to use the random number generation function of the STM32L100RC, include the `stdlib.h` library and use the following lines of code:

```
srand(NUMBER);        //seed random number generator
random = rand() % N; //generate a random number between 0 and N-1
```

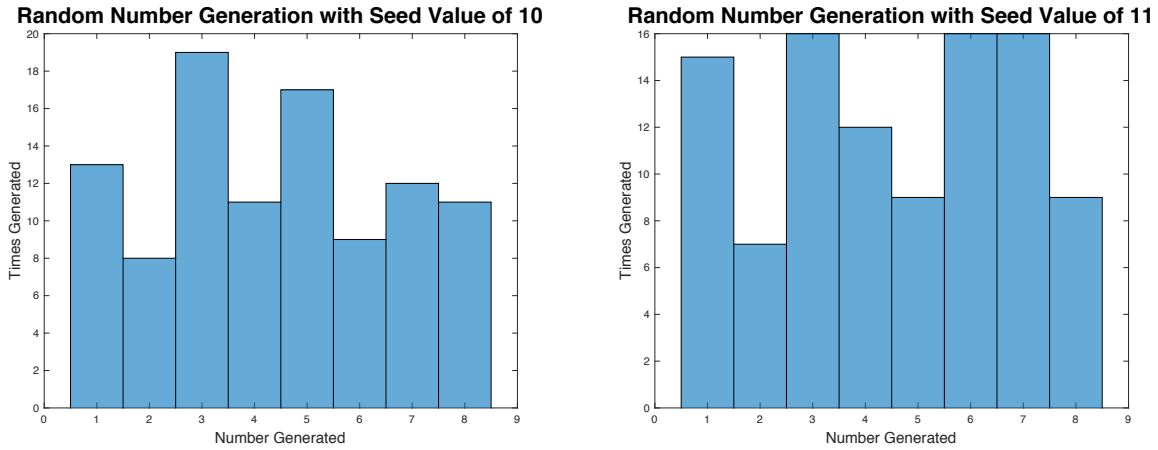

Figure 1: Comparison of 100 Numbers Generated with Different Seed Values

Assignment Overview

This week, you will be responsible for modifying the previous week's code to generate a 'rain' animation in addition to last week's simple moveable character game. Single pixel characters are to spawn in the top row of the LED display matrix in a random column at a constant rate. Additionally, these single pixels should move down at the same rate that new pixels spawn in the top row; this should create a the effect of a 'rain' animation. In addition to this rain animation, the player character from last week's lab must be present and maintain all previous functionalities. When the 'Pause' key is pressed, the rain should pause in place as well as the player controlled character. When the 'Reset' key is pressed, the rain pixels currently on screen should be cleared when the player controlled character is moved to the bottom left corner; this is to give the effect of a true reset.

In addition to the aforementioned functionality, you will be responsible for investigating the distribution of the randomly generated values for different seed numbers. Refer to the materials on the course webpage for instructions on logging data from the debug window.

Pre-Lab Assignment

Modify the code from last week's lab to implement the functionality described above. Give special consideration to the following:

1. How will the location of each 'rain' pixel be stored and displayed?
2. How will the location of all characters be displayed at the same time?
3. How often will the display values need to be updated for there to be no flickering or lagging of displayed characters?

Laboratory Experiments

1. Record all observations in your lab notebook
2. Graph the histograms of the recorded random numbers generated for at least 3 different seed values. Which seems to perform the best? How many data points are needed for the histogram to show a truly random distribution?
3. What would need to be changed to make collisions between the player character and the rain characters cause the screen to reset? What would need to be changed to keep track of how many 'rain drops' are dodged?