Lab 2: Solving Simple Problems in C

LAB 8

SECTION C

James Mechikoff

SUBMISSION DATE: 11/12/2018

Date

11/11/2018

Problem

The purpose of this lab was to showcase our knowledge of arrays, functions, and pointers to produce a program that creates a randomly generated maze with varying difficulty, generates a character that can move through this maze, allows control of this character as they fall at a constant rate, generate that movement through a moving average function using the gyroscope values of the controller and finally fail when you are stuck.

Analysis

This lab was broken down into two parts over four weeks. This helped break the lab down into multiple parts that could be done over the four weeks. First things that needed to be created was the moving average and the function to get this working. After this was done the next portion was generating a maze. After this we needed to generate the character and get it moving. Finally, it was adding finishing touches like failing if you get stuck, winning when you make it to the bottom, adding outer boundaries and removing the snail trail on your character.

Design

Breaking this portion down, the first part was a moving average based on gyroscope values from the controller. This was done through a function that has a 5-length array that had constant shifting values and taking the average of those 5. Next after this was completed, was implementing maze generation. This was done through a random number that was generated based on difficulty input given by the user. This number determined the amount of empty spaces and wall spaces in row, executing for the number of columns. After this the character was made and given the ability to move. This produced a trail of left-over characters which were deleted with a simple action every time you successfully moved. There was also the matter of losing if you had three walls directly adjacent to you. The player also needed to win when they made it to the bottom.

Testing

The first portion of this program was getting the moving average and ensuring that it properly calculated everything. This involved a lot of resetting, rethinking, and reconsidering how the logic of the program worked. After that it was setting up the Maze generation properly, which taught me the difference between Rows and Columns well. The final part was making sure the character moved properly, failed/won properly, and could not leave the area. That portion was a lot of save, compile, run and read errors or rethink how something needs to be run.

Comments

I loved this lab because it really reminded me of a lot of the games I am playing. These are games like Dwarf Fortress, Brogue, Cataclysm Dark Days Ahead, because of the ASCII and the movement of the character feeling very Stop go. It gives me an idea of how to actually create these or mod these and it also showed us some things we don't understand that makes me want to learn it and try to apply it to potential projects in the future.

Questions

Part 1

1. Explain the differences between the raw data and the averaged data in your graph for part A.

   The RAW data was the data prior to it being ran through and shifted through the formula for the averaged data.

2. Explain the delay you used to ensure character movement is not erratic.

   I just added a basic delay like we have used before.

Part 2

1. Describe how you checked if the avatar could safely move down, and go left/right.

   I just checked if it was blocked on all three sides which would prevent movement then fail out, otherwise you were free to move.

2. Describe what was necessary to check for the player losing the game.

   A wall on the left, right, and bottom.

Source Code

<Use NPP Exporter to PASTE source code>

```
/*-------------------------------------------------------------------------
--
```

```
-                                          SE 185 Lab 08
-             Developed for 185-Rursch by T.Tran and K.Wang
-     Name:       James Mechikoff
-     Section:    C
-     NetID:      jamesm47
-     Date:       11/12/2018
--------------------------------------------------------------------------------
*/

/*-------------------------------------------------------------------------------
--
-                                                 Includes
-------------------------------------------------------------------------------
*/


#include <stdio.h>
#include <math.h>
#include <ncurses/ncurses.h>
#include <unistd.h>
#include <stdlib.h>
#include <time.h>
/*-------------------------------------------------------------------------------
--
-                                                 Defines
-------------------------------------------------------------------------------
*/
/* Mathmatical constants */
#define PI 3.14159
// Screen geometry
// Use ROWS and COLS for the screen height and width (set by system)
// MAXIMUMS
#define NUMCOLS 100
#define NUMROWS 80

// Character definitions taken from the ASCII table
#define AVATAR 'A'
#define WALL '*'
#define EMPTYSPACE ' '

// Number of samples taken to form an average for the gyroscope data
// Feel free to tweak this. you may actually want to use the moving averages
// code you created last week
#define NUM_SAMPLES 10

/*-------------------------------------------------------------------------------
--
-                                                 Static Data
-------------------------------------------------------------------------------
*/
/* 2D character array which the maze is mapped into */
char MAZE[NUMROWS][NUMCOLS];

/*-------------------------------------------------------------------------------
--
-                                                 Prototypes
```

```
------------------------------------------------------------------------------
*/
// POST: Generates a random maze structure into MAZE[][]
//you will want to use the rand() function and maybe use the output %100.
//you will have to use the argument to the command line to determine how
//difficult the maze is (how many maze characters are on the screen).
void generate_maze(int difficulty);

// PRE: MAZE[][] has been initialized by generate_maze()
// POST: Draws the maze to the screen
void draw_maze(void);

// PRE: 0 < x < COLS, 0 < y < ROWS, 0 < use < 255
// POST: Draws character use to the screen and position x,y
void draw_character(int x, int y, char use);

// PRE: -1.0 < x_mag < 1.0
// POST: Returns tilt magnitude scaled to -1.0 -> 1.0
// you may want to reuse the roll function written in previous labs.
float calc_roll(float x_mag);

/*------------------------------------------------------------------------------
--
-                                                    Implementation
------------------------------------------------------------------------------
*/
/*    Main - Run with './ds4rd.exe -t -g -b' piped into STDIN */
int main(int argc, char* argv[])
{
      double gx, gy, gz;
      int difficulty = 0;
      int y_location = 0;
      int x = 50;
      int result = 1;
      int time;
      int t2;
      if (argc > 1) {
      sscanf(argv[1], "%d", &difficulty );
      }

      // setup screen
      initscr();
      refresh();
            // From 0 to 100

      // Generate and draw the maze, with initial avatar
      generate_maze(difficulty);
      draw_maze();

      scanf("%d, %lf, %lf, %lf,", &time, &gx, &gy, &gz);

      // Read gyroscope data to get ready for using moving averages.
      draw_character(50, 0, AVATAR);
      // Event loop

      do
      {
```

```
            // Read data, update average

            scanf("%d, %lf, %lf, %lf", &time, &gx, &gy, &gz);
            // Is it time to move? if so, then move avatar
            if(time - t2 > 200){
                  t2 = time;

                  if(calc_roll(gx) < -0.5 && MAZE[y_location][x + 1] != WALL
&& x < 99){
                        if(MAZE[y_location + 1][x + 1] != WALL){
                              y_location++;
                              draw_character(x + 1, y_location, AVATAR);
                              draw_character(x, y_location - 1, EMPTYSPACE);
                              x++;
                        }

                        else{
                              draw_character(x + 1, y_location, AVATAR);
                              draw_character(x, y_location, EMPTYSPACE);
                              x++;
                        }
                  }

                  else if(calc_roll(gx) > 0.5 && MAZE[y_location][x - 1] !=
WALL && x > 0){

                        if(MAZE[y_location + 1][x - 1] != WALL){
                              y_location++;
                              draw_character(x - 1, y_location, AVATAR);
                              draw_character(x, y_location - 1, EMPTYSPACE);
                              x--;
                        }

                        else{
                              draw_character(x - 1, y_location, AVATAR);
                              draw_character(x, y_location, EMPTYSPACE);
                              x--;
                        }
                  }

                  //
                  else if(MAZE[y_location + 1][x] == WALL &&
MAZE[y_location][x - 1] == WALL && MAZE[y_location][x + 1] == WALL){
                        result = 0;
                        break;
                  }

                  else if(MAZE[x][y_location + 1] != WALL){
                        y_location++;
                        draw_character(x, y_location, AVATAR);
                        draw_character(x, y_location - 1, EMPTYSPACE);
                  }

            }
      }

      while(y_location <= 80); // Change this to end game at right time
```

```
            // Print the win message
            endwin();

            if(result == 1){
                  printf("YOU WIN!\n");
            }
            else{
                  printf("YOU LOSE!\n");
            }
}

// PRE: 0 < x < COLS, 0 < y < ROWS, 0 < use < 255
// POST: Draws character use to the screen and position x,y
//THIS CODE FUNCTIONS FOR PLACING THE AVATAR AS PROVIDED.
//
//          >>>>DO NOT CHANGE THIS FUNCTION.<<<<

void draw_character(int x, int y, char use)
{
      mvaddch(y,x,use);
      refresh();
}

// POST: Generates a random maze structure into MAZE[][]
//you will want to use the rand() function and maybe use the output %100.
//you will have to use the argument to the command line to determine how
//difficult the maze is (how many maze characters are on the screen).
void generate_maze(int difficulty){
      int i = 0;
      int j = 0;

      for(i = 0; i < NUMCOLS; i++){
            for(j = 0; j < NUMROWS; j++){
                  if(rand() % 100 >= difficulty){
                        MAZE[i][j] = ' ';
                  }

                  else{
                        MAZE[i][j] = '*';
                  }
            }
      }
}

// PRE: MAZE[][] has been initialized by generate_maze()
// POST: Draws the maze to the screen
void draw_maze(){
      int i = 0;
      int j = 0;

      for(i = 0; i < NUMCOLS; i++){
            for(j = 0; j < NUMROWS; j++){
                  draw_character(i, j, MAZE[i][j]);
            }
      }
}
```

```
// PRE: -1.0 < x_mag < 1.0
// POST: Returns tilt magnitude scaled to -1.0 -> 1.0
// you may want to reuse the roll function written in previous labs.
float calc_roll(float x_mag){
      if(x_mag <= 1.0 && x_mag >= -1.0){
            x_mag = (asin(x_mag) * (PI/2));
      }
      return x_mag;
}
```

Screen Shots