ExMaze

LAB 9

SECTION X

SUBMITTED BY:

NATHAN SHULL

SUBMISSION DATE:

DECEMBER 2, 2016

Problem

The purpose of this lab, the ExMaze, is to create a program, which draws a variety of different mazes, based on what the user inputs for a difficulty. The program then reads in accelerometer values to know if the avatar in the maze should move left, right, or go down. The program should not allow an avatar to move into a space occupied by a wall. The point of this lab it to practice working with two-dimensional arrays and develop skills in handling events in a loop.

Analysis

For part 1, the problem stated that I must create a random maze, start the avatar at the top center of the screen, and make a delay for the avatar to fall down the screen. In the second part, I had to add 3 more elements to the program. First off, I had to read input from the Esplora that would make the character move to the left or right, depending on the accelerometer values. Second, I had to make sure that the avatar did not move into locations occupied by the maze or off of the screen. Finally, the third element of part 2 was to check if the avatar made it to the bottom of the screen, and if so, print "You win!" I also chose to attempt the bonus one, which was to detect when the avatar was in a position where it could not move left, right, or down, and was in a position that would print "You lose."

Design

To create a random maze, I first iterated through a two dimensional array. Within the array, I used the rand( ) function mod 100, so that a higher difficulty would have a greater chance of drawing a WALL, and a lower difficulty would have a higher change of drawing an EMPTY_SPACE. To start a character at the top center of the screen I declared and initialized two variables, c(for columns) and r(for rows). I initialized c at 50 and r at 0, making it start at the top center. To create a delay, I subtracted currentTime from lastMove, and compared it with 0.4, so that every four tenths of a second, the avatar would look to move again. I assigned lastMove to currentTime after it went through each loop.

To read accelerometer data, I made a for loop for i being less than the number of samples, and creating an array to which those values were assigned to a_x. And then to move it left of right, I used a few different if loops. If the space down and to the right was clear, AND if the eslora was close to -1, than the avatar would move down and to the right, and an empty space would be drawn where it previously was. I also wrote similar if statements for if the esplora was tilted to the left, or not even tilted at all. My last if statement was if there was no wall under the character, to add one to row, in order to keep the avatar moving downwards. If the rows got to 81, that means that the avatar reached the bottom, and I printed the you win message.

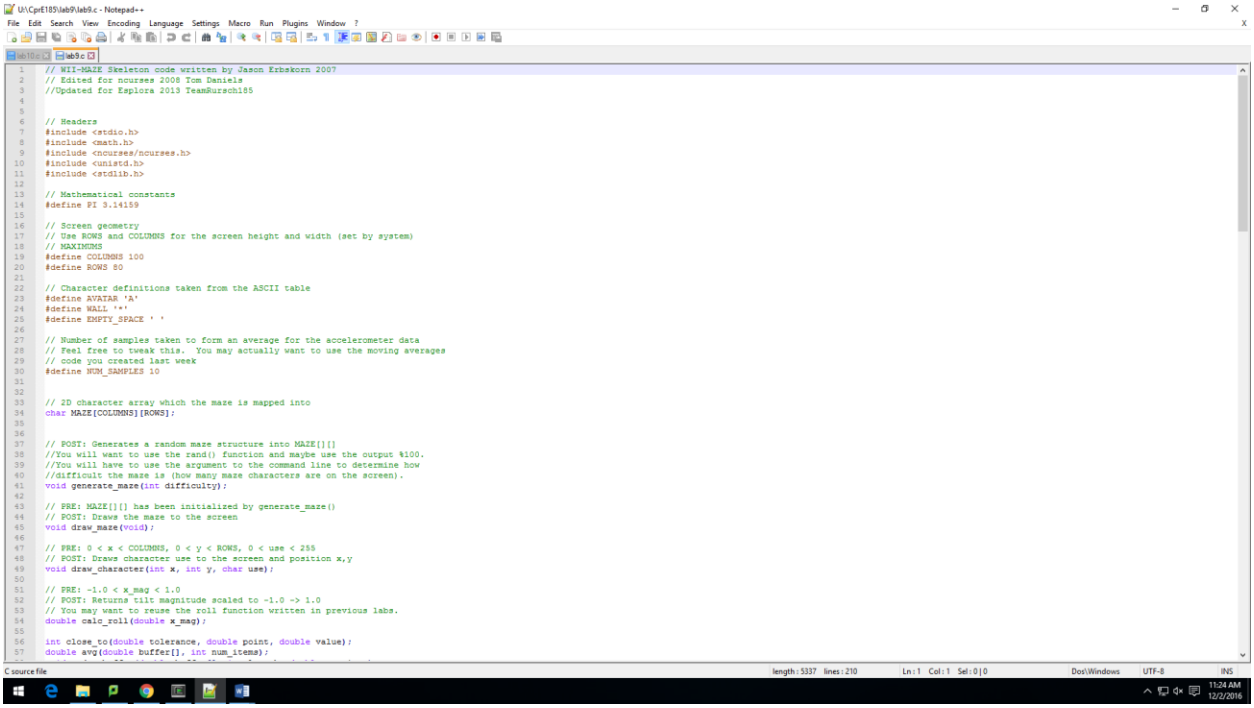
Testing

The biggest issue I had when testing this program was having my avatar go left and right. For a long time, I could only get the avatar to go right, but it would never move to the left. I carefully examined the main function, and couldn't seem to find anything wrong. I then went looking into the functions I was using to move the avatar left, mainly calc_roll. It was here that I

found my problem. I was comparing x_mag to positive 0.3 in both if statements, rather than a -0.3 in the one. After changing the sign, I got a program that was able to move an avatar left and right.

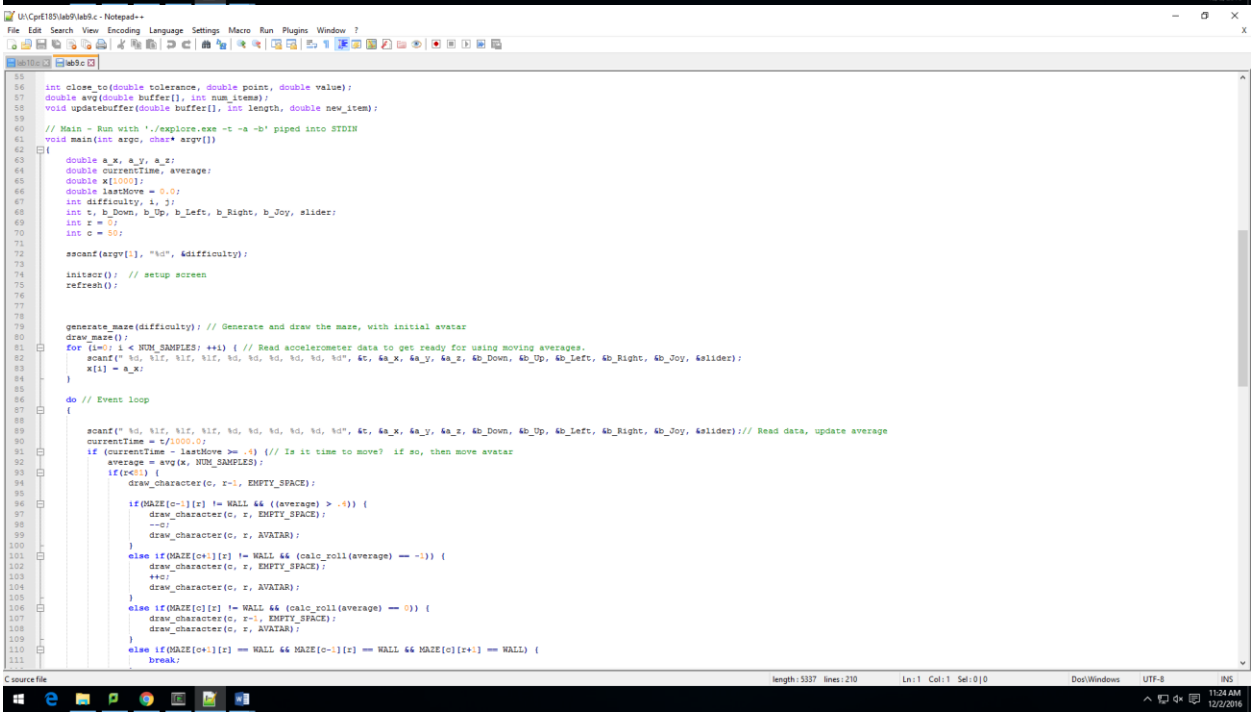
Comments

1. For the safe to go right/left, what is checked is if there is a wall one row down, either to the left or the right, depending on which direction you want to go. For the can I fall, the program checks if there is a wall or empty space at [c][r+1], or one row down.
2. I actually did check to see if a player loses a game. To check this, you create an if statement that checks if there is a wall directly to the left, right, and below the current avatars positon. If ALL three are true, than the if statement breaks, and the program goes to (r != 81), and prints “You lose.”



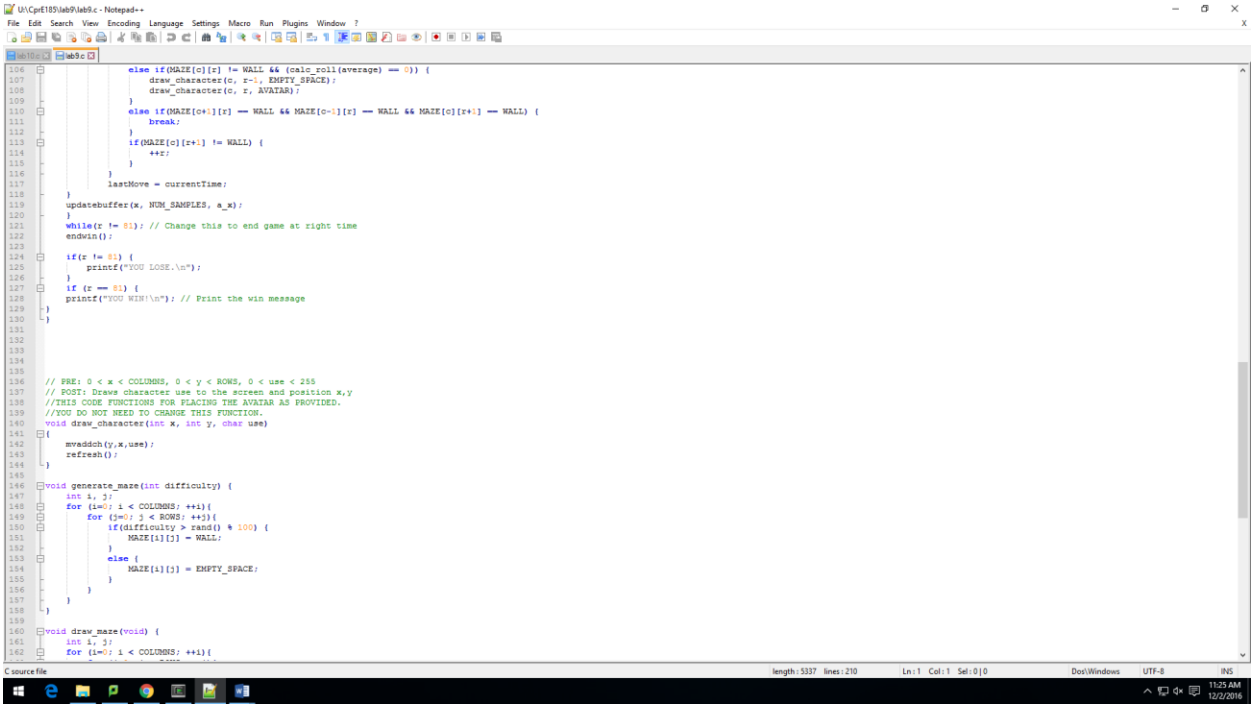
```
// WII-MAZE Skeleton code written by Jason Erbskorn 2007
// Edited for ncurses 2008 Tom Daniels
//Updated for Esplora 2013 TeamRursch185


// Headers
#include <stdio.h>
#include <math.h>
#include <ncurses/ncurses.h>
#include <unistd.h>
#include <stdlib.h>

// Mathematical constants
#define PI 3.14159

// Screen geometry
// Use ROWS and COLUMNS for the screen height and width (set by system)
// MAXIMUMS
#define COLUMNS 100
#define ROWS 80

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

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


// 2D character array which the maze is mapped into
char MAZE[COLUMNS][ROWS];


// 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 < COLUMNS, 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.
double calc_roll(double x_mag);

int close_to(double tolerance, double point, double value);
double avg(double buffer[], int num_items);
```

C source file | length : 5337 lines : 210 | Ln : 1 Col : 1 Sel : 0 | 0 | Dos\Windows | UTF-8 | INS

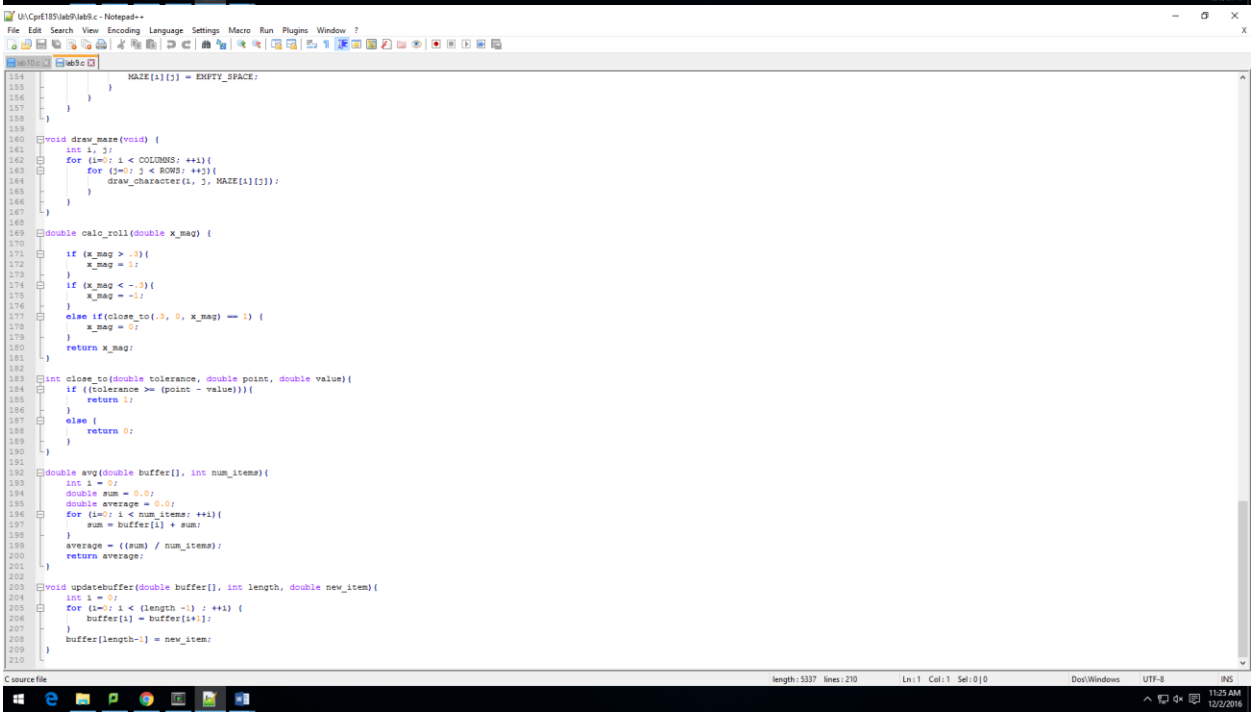


```
int close_to(double tolerance, double point, double value);
double avg(double buffer[], int num_items);
void updatebuffer(double buffer[], int length, double new_item);

// Main - Run with './explore.exe -t -a -b' piped into STDIN
void main(int argc, char* argv[])
{
    double a_x, a_y, a_z;
    double currentTime, average;
    double x[1000];
    double lastMove = 0.0;
    int difficulty, i, j;
    int t, b_Down, b_Up, b_Left, b_Right, b_Joy, slider;
    int r = 0;
    int c = 50;

    sscanf(argv[1], "%d", &difficulty);

    initscr();  // setup screen
    refresh();


    generate_maze(difficulty); // Generate and draw the maze, with initial avatar
    draw_maze();
    for (i=0; i < NUM_SAMPLES; ++i) { // Read accelerometer data to get ready for using moving averages.
        scanf(" %d, %lf, %lf, %lf, %d, %d, %d, %d, %d, %d", &t, &a_x, &a_y, &a_z, &b_Down, &b_Up, &b_Left, &b_Right, &b_Joy, &slider);
        x[i] = a_x;
    }

    do // Event loop
    {

        scanf(" %d, %lf, %lf, %lf, %d, %d, %d, %d, %d, %d", &t, &a_x, &a_y, &a_z, &b_Down, &b_Up, &b_Left, &b_Right, &b_Joy, &slider);// Read data, update average
        currentTime = t/1000.0;
        if (currentTime - lastMove >= .4) {// Is it time to move?  if so, then move avatar
            average = avg(x, NUM_SAMPLES);
            if(r<81) {
                draw_character(c, r-1, EMPTY_SPACE);

                if(MAZE[c-1][r] != WALL && ((average) > .4)) {
                    draw_character(c, r, EMPTY_SPACE);
                    --c;
                    draw_character(c, r, AVATAR);
                }
                else if(MAZE[c+1][r] != WALL && (calc_roll(average) == -1)) {
                    draw_character(c, r, EMPTY_SPACE);
                    ++c;
                    draw_character(c, r, AVATAR);
                }
                else if(MAZE[c][r] != WALL && (calc_roll(average) == 0)) {
                    draw_character(c, r-1, EMPTY_SPACE);
                    draw_character(c, r, AVATAR);
                }
                else if(MAZE[c+1][r] == WALL && MAZE[c-1][r] == WALL && MAZE[c][r+1] == WALL) {
                    break;
```

C source file | length : 5337 lines : 210 | Ln : 1 Col : 1 Sel : 0 | 0 | Dos\Windows | UTF-8 | INS



```
                else if(MAZE[c][r] != WALL && (calc_roll(average) == 0)) {
                    draw_character(c, r-1, EMPTY_SPACE);
                    draw_character(c, r, AVATAR);
                }
                else if(MAZE[c+1][r] == WALL && MAZE[c-1][r] == WALL && MAZE[c][r+1] == WALL) {
                    break;
                }
                if(MAZE[c][r+1] != WALL) {
                    ++r;
                }
            }
            lastMove = currentTime;
    }
    updatebuffer(x, NUM_SAMPLES, a_x);
    }
    while(r != 81); // Change this to end game at right time
    endwin();

    if(r != 81) {
        printf("YOU LOSE.\n");
    }
    if (r == 81) {
    printf("YOU WIN!\n"); // Print the win message
}
}




// PRE: 0 < x < COLUMNS, 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.
//YOU DO NOT NEED TO CHANGE THIS FUNCTION.
void draw_character(int x, int y, char use)
{
    mvaddch(y,x,use);
    refresh();
}

void generate_maze(int difficulty) {
    int i, j;
    for (i=0; i < COLUMNS; ++i){
        for (j=0; j < ROWS; ++j){
            if(difficulty > rand() % 100) {
                MAZE[i][j] = WALL;
            }
            else {
                MAZE[i][j] = EMPTY_SPACE;
            }
        }
    }
}

void draw_maze(void) {
    int i, j;
    for (i=0; i < COLUMNS; ++i){
```

```
                MAZE[i][j] = EMPTY_SPACE;
            }
        }
    }
}

void draw_maze(void) {
    int i, j;
    for (i=0; i < COLUMNS; ++i){
        for (j=0; j < ROWS; ++j){
            draw_character(i, j, MAZE[i][j]);
        }
    }
}

double calc_roll(double x_mag) {

    if (x_mag > .3){
        x_mag = 1;
    }
    if (x_mag < -.3){
        x_mag = -1;
    }
    else if(close_to(.3, 0, x_mag) == 1) {
        x_mag = 0;
    }
    return x_mag;
}

int close_to(double tolerance, double point, double value){
    if ((tolerance >= (point - value))){
        return 1;
    }
    else {
        return 0;
    }
}

double avg(double buffer[], int num_items){
    int i = 0;
    double sum = 0.0;
    double average = 0.0;
    for (i=0; i < num_items; ++i){
        sum = buffer[i] + sum;
    }
    average = ((sum) / num_items);
    return average;
}

void updatebuffer(double buffer[], int length, double new_item){
    int i = 0;
    for (i=0; i < (length -1) ; ++i) {
        buffer[i] = buffer[i+1];
    }
    buffer[length-1] = new_item;
}
```