LAB REPORT

LAB 9 PARTS 1&2
SECTION C

SUBMITTED BY:

SCOTT VLASIC

SUBMISSION DATE:

APRIL 21, 2016

Problem:

The problem in this lab was that I was trying to create a real time game with the esplora. I had to create a random maze of characters using the variables COLS and ROWS as well as set my avatar at the top of the screen. Then, every so often a the character would fall based on a delay created by me. This was the main goal of part 1, to get the character to move from the top of the screen to the bottom but also stop if it hit one of the maze characters.

For part 2, I had to make it so that when the esplora was tilted to the right, the avatar would move to the right and when the avatar was tilted to the left, the avatar would tilt to the left. The avatar would win if it made it to the bottom of the screen without getting stuck and would not be able to move where there was a maze object or be able to move off the screen.

Analysis:

To begin this lab I started by opening lab5.c on Notepad++ so I could edit the code and being solving the problem. This lab was a lot like past labs since I had to look at the skeleton code I was provided with and work from there. For example, in draw_character(int x, int y, char use), I read the skeleton code and it said I didn't have to modify anything for the function. In the main function it also listed all the tasks that needed to be done in comments, such as checking if it was time to move and where to read the data and update the average. So, much like the previous labs, I used the commented skeleton code to get myself started and went from there.

Design:

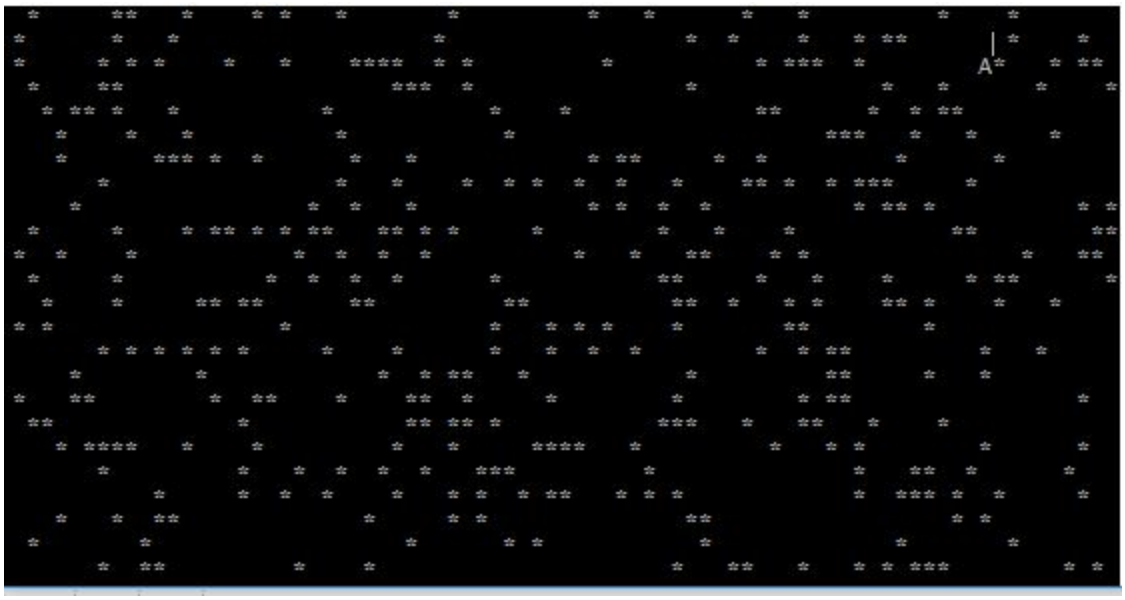
To begin, I started by writing the generate_maze function that would be used for the avatar to move throughout. To generate the maze, I wrote a nested for() loop that went through the number of columns(COLNUM), then the number of rows(ROWNUM). From there I randomized the maze by writing if(rand() %100 <= difficulty){ MAZE[i][j] = EMPTY_SPACE. This said that is the random number % 100 was less than or equal to the difficulty, it would print an empty space. Else the maze would print a WALL character. Then, to draw the maze to the screen, I wrote another nested for loop that went through the number of columns and rows and resulted in draw_character(i,j,MAZE[i][j]);. Finally, I wrote a function called calc_roll(double x_mag) that took the acceleration of x and checked to see if it was greater than 1 and sets x_mag equal to 1. If it's not greater than 1, it sets x_mag equal to -1. The return of this function was the arcsin(x_mag). Moving on to the main function, I began by creating my variables for the problem. I created three variables for the accelerations(ax, ay, and az), as well as variables for the buttons/slider on the esplora (b1,b2,b3,b4,b5,b6). There were also variables for the time, initial time, the position in terms of x, the position in terms of y, the movement that was going to be made, the difficulty, and if the player was a winner or not. To set up the screen, I was given the terms initscr(); and refresh();. Initscr() initialized the screen and refresh() refreshes the screen. Then, to generate

the maze, I called the generate_maze(difficulty) function that created a maze based on the difficulty. I also called draw_maze(); which would draw the maze generated above as well as draw_character(posx, posy, AVATAR) function to create the avatar on the screen. I had to initialize the posx= 45 and posy = 0 so that the avatar would start at the top of the screen and on the middle of the screen. After this, I scanned the esplora for data using a scanf() function to get ready for using moving averages.

I then created a do while loop so as the event loop for which the avatar would move. I started off by scanning in more data and thus updating the average. Then I moved on to the problem of if it was time to move the avatar or not. I wrote the if statement if(time>inittime+1000), which checked if it was time for the avatar to move or not and made the delay at which the avatar would fall. Then, within that if statement there was an if() that checked it the position on the maze was a wall or not and if so it moved the posy++, thus moving the avatar down. I then calculated the sin of the tilt in the direction x by setting arcsin = calc_roll (ax) and calling the calc_roll function for this. Then, I checked to see if the avatar should move left. To do this I wrote an if() that said if the arcsin < -PI/6 && posx > 0 && MAZE[posx+1][posy] != WALL, then the avatar would move left. To see if it moved right, I wrote and if() that said if arcsin>PI && posx< COLNUM-1 && MAZE[posx-1][posy] != WALL, then the avatar would move right. If none of these conditions were true, then the avatar could not move at all and the loop would break. All of this was executed while (posy!= ROWNUM) since the posy had to be less than the number of rows so that the game would fit on the screen. If the character reached the bottom of the screen, it would print the endwin(); which wa the winner's message. I added another if() statement to see if the avatar won saying that if the winner variable == FALSE, the it would print "YOU LOSE" and if not it would print "YOU WIN".

Testing:

To test this code I went through many different trials and errors. The first came with making the esplora move to the left or right, respectively. I had the if() statements mixed up and thus when I moved the epslora left the avatar would move right and when I moved the esplora to the right, it would move to the left. I also tested different difficulty values seeing how a maze with 5 differed in difficulty to one with 90. I also forgot multiple times when compiling my code to use the -lncurses command and my code wasn't working. Some of my output for my code can be seen below.




```
svlasic@co2018-17 /cygdrive/u/cpre185/lab9
$ ./explore.exe -p COM3 -t -a -b | ./lab9.exe 60
YOU LOSE!

svlasic@co2018-17 /cygdrive/u/cpre185/lab9
$ ./explore.exe -p COM3 -t -a -b | ./lab9.exe 10
YOU LOSE!
```

Comments:

I found this lab to be the most challenging one I had done so far. Making the avatar move different directions was hard but the output for this lab was definitely the most interesting and fun of the other labs. In the safe to go left/right part of the main function, I looked to see if the acceleration value was greater than PI/6 and that the position of x was greater than the number of columns -1 for the movement to the right. I also had to make sure the space it was moving to was not a WALL piece. For the left I had to see if the acceleration value was less than -PI/6 and that the position of x was greater than 0. Again, I had to make sure the space it was going to wasn't a WALL piece. To see if it could fall, I had to make sure the position of y wasn't going to fall onto a WALL piece and if not it was allowed to fall down a peg. To see if the player is losing the game, I would implement a nested for loop that would check how many moves a player can make given their previous move. For example, if a player makes a move down and to the left it would print that the user has 2 moves left and also where those moves are. Ultimately, the user would be able to see how many moves they have and have a sense if they're losing if their number of moves left goes down.

Implementation:

```
// 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 COLS for the screen height and width (set by system)
// MAXIMUMS
#define COLNUM 100
#define ROWNUM 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
#define TRUE 1
#define FALSE 0


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


// 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 < y_mag < 1.0
// POST: Returns tilt magnitude scaled to -1.0 -> 1.0
// You may want to reuse the pitch function written in previous labs.
float calc_pitch(float y_mag);

void updatebuffer(double buffer[], int length, double new_item);

double calc_roll(double x_mag);

// Main - Run with './explore.exe -t -a -b' piped into STDIN
void main(int argc, char* argv[])
{
   double ax, ay, az; // Values for the accelerations
   int b1, b2, b3, b4, b5, b6; // Variables for the buttons, slider, etc.
   int time = 0;
   int inittime;
   int posx = 45;
   int posy = 0;
   int move;
   int difficulty = 20;
   double arcsin = 0.0;
   int winner = TRUE;

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

```
   if (argc>1) {
      sscanf(argv[1], "%d", &difficulty );
      printf("You entered a buffer length of %d\n", difficulty);
   }
   else {
      printf("Enter a length on the command line\n");
      return;
   }

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

   // Read accelerometer data to get ready for using moving averages.
   scanf("%d, %lf, %lf, %lf, %d, %d, %d, %d, %d, %d", &time, &ax, &ay, &az, &b1, &b2,
&b3, &b4, &b5, &b6);
   inittime = time;
   //updatebuffer(double MAZE[], int length, double new_item)


   // Event loop
   do
   {

      // Read data, update average
      scanf("%d, %lf, %lf, %lf, %d, %d, %d, %d, %d, %d", &time, &ax, &ay, &az, &b1, &b2,
&b3, &b4, &b5, &b6);

      // Is it time to move?  if so, then move avatar
      if(time > inittime + 100)
      {
            //Checks if Avatar should move down
            if(MAZE[posx][posy + 1] != WALL)
            {
               posy++;
            }
```

```
            draw_character(posx, posy, AVATAR);
            inittime = time;

            draw_character(posx, posy - 1 , ' ');

            // Calculate the sin  of the tilt in the x direction
            arcsin = calc_roll(ax);

            //Check if the avatar should move left
            if(arcsin < -PI / 6 && posx > 0 && MAZE[posx + 1][posy] != WALL)
            {
               draw_character(posx, posy, ' ');
               posx++;
               draw_character(posx, posy, AVATAR);




            }
            //Check if the avatar should move right
            else if(arcsin > PI / 6 && posx < COLNUM-1 && MAZE[posx - 1][posy] != WALL)
            {
               draw_character(posx, posy, ' ');
               posx--;
               draw_character(posx, posy, AVATAR);

            }
            //Check if the avatar can even move. If not break the loop
            if(MAZE[posx + 1][posy] == WALL && MAZE[posx - 1][posy] == WALL &&
MAZE[posx][posy + 1] == WALL)
            {
               winner = FALSE;
               break;
            }
      }
   } while(posy != ROWNUM-1); // Change this to end game at right time

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

```
    //Check if the avatar won
    if(winner == FALSE)
    {
        printf("YOU LOSE!\n");
    }
    else
    {
        printf("YOU WIN!\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.
//YOU DO NOT NEED TO 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)
{
    for(int i = 0; i < COLNUM; i++)
    {
        for(int j = 0; j < ROWNUM; j++)
        {
            if(rand() % 100 <= difficulty)
            {
                MAZE[i][j] = EMPTY_SPACE;
            }
            else
            {
```

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

// PRE: MAZE[][] has been initialized by generate_maze()
// POST: Draws the maze to the screen
void draw_maze(void)
{
   for(int i = 0; i < COLNUM; i++)
   {
      for(int j = 0; j < ROWNUM; j++)
      {
         draw_character(i, j, MAZE[i][j]);
      }
   }
}

// Takes the accelaration of x and checks if its greater 1 and sets it to 1, if its less than 1. Then returns the arcsin of
// the acceleration of x
double calc_roll(double x_mag)
{
   if(x_mag > 1.0)
      x_mag = 1.0;
   if(x_mag < -1.0)
      x_mag = -1.0;
   return asin(x_mag);
}
```