**ExMaze**

**LAB #9**

**SECTION AA**

**SUBMITTED BY:**

**ANDREW HANCOCK**

**SUBMISSION DATE:**

**12/4/2016**

**Lab 9 part 1**

**// 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>**

**// 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){**

**for(int i = 0;i < COLUMNS;i++){**

**for(int j = 0;j < ROWS;j++){**

**int AppearanceChance = (rand() % 100);**

**if(AppearanceChance > 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 < COLUMNS;i++){**

**for(int j = 0;j < ROWS;j++){**

**draw\_character(i, j, MAZE[i][j]);**

**}**

**}**

**}**

**// 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 < 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 x\_mag){**

**double pitch = asin(x\_mag);**

**return pitch;**

**}**

**// Main - Run with './explore.exe -t -a -b' piped into STDIN**

**void main(int argc, char\* argv[])**

**{**

**int t, i = 0,L = 0,xCoord,WL = 0;**

**double x, y, z, pitch;**

**int difficulty = atoi(argv[1]);**

**xCoord = (COLUMNS/2);**

**// setup screen**

**initscr();**

**refresh();**

**// Generate and draw the maze, with initial avatar**

**generate\_maze(difficulty);**

**draw\_maze();**

**draw\_character((COLUMNS/2), 0, AVATAR);**

**// Read accelerometer data to get ready for using moving averages.**

**scanf("%d, %lf, %lf, %lf", &t, &x, &y, &z);**

**// Event loop**

**do**

**{**

**scanf("%d, %lf, %lf, %lf", &t, &x, &y, &z);**

**// Read data, update average**

**pitch = calc\_pitch(x);//calculate pitch**

**// Is it time to move? if so, then move avatar**

**if((i % 50) == 0){**

**if(pitch < -0.5){//Right move**

**if((MAZE[xCoord + 1][L] == EMPTY\_SPACE) && (xCoord < COLUMNS)){//is the space clear and is there a wall**

**xCoord = xCoord + 1;//update x coord**

**}**

**}else if(pitch > 0.5){//Left move**

**if((MAZE[xCoord - 1][L] == EMPTY\_SPACE) && (xCoord > 0)){// is the space clear and is there a wall**

**xCoord = xCoord - 1;//update x coord**

**}**

**}**

**draw\_character(xCoord, L, AVATAR); //Place New Avatar**

**if(((MAZE[xCoord + 1][L]) == WALL) && ((MAZE[xCoord - 1][L]) == WALL) && ((MAZE[xCoord][L+1]) == WALL)){//is the next space a bucket**

**WL = 1;//sets win variable equal to 1 meaning a loss**

**break;//break game loop**

**}**

**if(MAZE[xCoord][L + 1] == EMPTY\_SPACE){//is the next space below clear**

**L++;//add one row**

**}**

**}**

**i++; //timing for the loop to execute**

**} while(L != 81); // checks to see if its at the bottom**

**// Print the win message**

**endwin();**

**if(WL == 0){//check to see if its a win**

**printf("YOU WIN!\n");//announce a win**

**}else if(WL == 1){//check to see if its a loss**

**printf("YOU LOSE");//announce a loss**

**}**

**}**

**// 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();**

**}**

**Questions and Experiments**

1. In the safe-to-go left and right conditions the spot that the avatar will be going to checked for two things: to make sure it’s clear and to make sure there isn’t a wall. In the safe to fall condition only one thing is checked: that the spot below is clear.
2. One could check to see if the player is losing the game by calculating the if the player is going to fall into a bucket that is bigger than one space and the avatar is farther away because then the avatar might not have enough spaces to fall off to the side of it. You could also check to see if there is a path to the end. If not, then the player is losing anyway.

**Problem**

The problem is that a program needs to be made to create a maze that is 80 rows and 100 columns. The program needs to be input a percentage of spaces that will be covered. The empty spaces will be empty and the filled spaces will have stars. There will be an avatar that will start at the top and will fall slow enough so that the player can dodge and maneuver the stars. The program will take Arduino accelerometer input. Based off of the calculated pitch, the program will either move left, right, or straight. Throughout the entire game loop, the program will need to check to make sure that the spot the character wants to move into is clear.

**Analysis**

The input will be an argument in the command line. The input will be a percentage between 0 and 100. This will determine the amount of the spaces that are covered in stars which are the spaces that the player won’t be able to go. The second input will be the Arduino esplora accelerometer data. This will help determine whether the player wants to go right, left, or straight. The program will use this data to calculate pitch through a formula in the code. The output will be on a screen that will constantly update with the new location of the avatar and whether it has won or lost. There are conditions in the code that will check if the player gets stuck, hits a wall, or has gotten to the bottom and won.

**Design**

My program uses four functions not including main. The first one is called generate\_maze. This takes the difficulty and uses rand to decide the correct amount and correct placement of the stars in the maze. It uses two for loops to cycle through all of the spaces in the maze array. The second function is draw\_maze. This function uses the same two for loops to cycle through every spot in the array. For every spot, it calls the draw\_character function and sends the x and y value and the star or space at that x and y value. The third function is the calc\_pitch function. This function takes the x magnitude from the accelerometer and uses the arcsine function to calculate the roll in radians. This data is used to decide, with a tolerance, whether the player is going left, right, or straight. The last function is the draw\_character function. This takes the inputs: x, y, and the character to use. This function draws whatever character on the game board at the location it was sent, then it refreshes the game board with the updates it made. My main loop is made up of two main parts: the initial code and the game loop. In the initial code my program gets the difficulty, generates the maze, draws the maze, and draws the avatar. It also scans one line of data in from the esplora as well as declaring all of the variables. The main loop continually scans in esplora data, calculates the pitch in radians, and executes an if statement every 50 iterations of the overall do while loop. The if loop that is executed checks the pitch and if it is less than -0.5 check the correct spaces and then updates the maze if it’s clear to move. This internal if loop also checks to see if the player is in a bucket and then sets the win/loss variable to the correct number. It also adds a row. My do while loop finally breaks after the rows reach 81 which would be the end of the board. Then an if statement prints whether the player won or loss.

**Testing**

I tested and compiled this program throughout the different stages of development. Once I got the program to generate and draw the game board I tested it with different difficulties. Once I got the entire program written I made sure that the player wouldn’t go through the wall, didn’t hit stars, and would lose if it was in a bucket.

**Comments**

I learned a lot about lncurses this lab and how powerful of a tool it is. I realized that writing all of that code to check the conditions can get really confusing if I don’t keep track of what is going on at the time or where the avatar is. I messed this up once because I didn’t realize that the character hadn’t moved before a condition was checked but I caught it and fixed it after it wasn’t working correctly.