**Lab8.Ogbondah**

**LAB 8**

**SECTION D**

**Chimzim Ogbondah**

**SUBMISSION DATE: 11.13.2018**

**DATE: 11.13.2018**

# Problem

1. The purpose of this lab was to be able to manipulate and problem solve with loops, work with 2D arrays, and practice top down program design.

# Analysis

1. The problem for this lab was to finish writing function given in the source code that would generate the maze based on the difficulty entered on the command line. The maze would then be printed to the screen where the user could try and get to the bottom without being blocked off by obstacles. If the user made it to the end “YOU WIN” was to be printed to the terminal, but if the user could no longer advance then the game was terminated and a “YOU LOSE” message was printed.

# Design

1. Our problem was to create a program using the given functions to generate the maze with obstacles, and the number of obstacles was determined by the difficulty. Then print the maze to the terminal so the user could play the game. The user wasn’t allowed to move into spaces that were occupied by the maze difficulty and if the user could no longer moved downwards the game was terminated. If the user could make it to the end a YOU WIN message was printed. I first started by making my function work right, then I moved on to logic inside of the main function.
   1. I first I worked on the calc\_roll function. I took my code from the last lab, but I changed it so it only returned calues between -1 and 1 by capping everything greater than 1 and less than -1 at those values and then returning the value.
   2. Next, I worked on the generatemaze function. I used a loop for this function since the maze is a 2D array. I looped through it and had and if statement using rand() % 100 and if it was less than the entered user difficulty then a WALL would be added, otherwise an EMPTY\_SPACE was added.
   3. For the draw\_maze function I used the same loop but instead I called the draw\_characters function and put to the screen whatever was saved by the generate maze function for the MAZE array.
   4. Inside the main function I scanned in the values from the DS4 and then I filled my state[] array which would be used for the moving average function. I then used the inside my m\_avg() and then called that function inside of my calc\_roll function to ensure the AVATAR didn’t jump all over the place. In my logic the AVATAR was suppose to move down one space in the Y direction every second so I made a while statement to check if one second had passed (time % 1000 > 0) I then made the AVATARS position a space and had if statements to check if the user should move left or right based on the calc\_roll. It would then do the same by taking the current position of the AVATAR on the X axis and make it an EMPTY\_SPACE it would the increment the position based off the ouput value and then draw the AVATAR in the new space. Finally, the Y position would be updated. There was also a check so if the positions of left and right and under the AVATAR were WALL then the game would be eneded.

Using the basic outline above I met the criteria from the lab instructions and the functions instructions which in the main function would allow the user to play this maze game. To ensure my code was right I compiled it many times and looked for wrong outputs.

# Testing

1. To make sure the results of the solution were correct, I compiled to make sure I didn’t get any errors, and then I went back and ensured that I added comment to every line to make the code work, and what each line meant and that it made sense also looking at what the function was supposed to do. I then I ran the program to make sure it worked effectively again.

# Comments

1. I added and && check along with my calc roll. So if it was wanting to move right and the position 1 to the right was not a WALL then it would move
2. I just checked to see if one to the right and one to the left on the x axis were equal to walls and then the one below on the y axis was the same. Then the game would end.
3. The average data is a lot smoother along the curves and in general than the raw data
4. The moving average function, and then I also capped off out puts at -1 and 1 so it made it easier on the moving average.

/\*-----------------------------------------------------------------------------

- SE 185 Lab 08

- Developed for 185-Rursch by T.Tran and K.Wang

- Name: CHimzim Ogbondah

- Section: D

- NetID: ogbondah

- Date: 11.13.2018

-----------------------------------------------------------------------------\*/

/\*-----------------------------------------------------------------------------

- Includes

-----------------------------------------------------------------------------\*/

#include <stdio.h>

#include <math.h>

#include <ncurses/ncurses.h>

#include <unistd.h>

#include <stdlib.h>

/\*-----------------------------------------------------------------------------

- Defines

-----------------------------------------------------------------------------\*/

/\* Mathmatical 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 moving average for the gyroscope data

Feel free to tweak this. \*/

#define NUM\_SAMPLES 10

/\*-----------------------------------------------------------------------------

- Static Data

-----------------------------------------------------------------------------\*/

/\* 2D character array which the maze is mapped into \*/

char MAZE**[**COLUMNS**][**ROWS**];** //use to update maze

/\*-----------------------------------------------------------------------------

- 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 < 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 < 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 mag**);**

/\* Updates the buffer with the new\_item and returns the computed

moving average of the updated buffer \*/

double m\_avg**(**double buffer**[],** int avg\_size**,** double new\_item**);**

/\*-----------------------------------------------------------------------------

- Implementation

-----------------------------------------------------------------------------\*/

/\* Main - Run with './ds4rd.exe -t -g -b' piped into STDIN \*/

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

int skillLevel**;**

/\* Setup screen for Ncurses

The initscr functionis used to setup the Ncurses environment

The refreash function needs to be called to refresh the outputs

to the screen \*/

initscr**();**

refresh**();**

**if** **(**argc **!=** 2 **)** **{**

printw**(**"You must enter the difficulty level on the command line."**);**

refresh**();**

**return;**

**}**

**else** **{**

/\* Setup screen for Ncurses

The initscr functionis used to setup the Ncurses environment

The refreash function needs to be called to refresh the outputs

to the screen \*/

initscr**();**

refresh**();**

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

**while** **(**skillLevel **<** 0 **||** skillLevel **>** 100**)**

**{**

printw**(**"The difficulty level must be between 0 and 100"**);**

refresh**();**

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

**}**

/\* WEEK 2 Generate the Maze \*/

/\* Read gyroscope data and fill the buffer before continuing \*/

generate\_maze**(**skillLevel**);**

int time**,** i**;**

double g\_z**,** g\_x**,** g\_y**,** state**[**NUM\_SAMPLES**];**

**for(**i **=** 0**;** i **<** NUM\_SAMPLES**;** i**++)** **{**

scanf**(**"%d, %lf, %lf, %lf"**,** **&**time**,** **&**g\_x**,** **&**g\_y**,** **&**g\_z**);**

state**[**i**]** **=** g\_x**;**

**}**

int playerPosX **=**40**;**

int playerPosY **=** 0**;**

draw\_character**(**playerPosX**,** playerPosY**,** AVATAR**);**

/\* Event loop \*/

**do** **{**

scanf**(**"%d, %lf, %lf, %lf"**,** **&**time**,** **&**g\_x**,** **&**g\_y**,** **&**g\_z**);**

double movingAvg **=** m\_avg**(**state**,** NUM\_SAMPLES**,** g\_x**);**

/\* Read data, update average \*/

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

**if(**time **%** 200 **<=** 1**)** **{**

draw\_character**(**playerPosX**,** playerPosY**,** EMPTY\_SPACE**);**

**if(**MAZE**[**playerPosX**+**1**][**playerPosY**]** **!=** WALL **&&** calc\_roll**(**movingAvg**)** **>** 0**)** **{** //moves to the right

draw\_character**(**playerPosX**,** playerPosY**,** EMPTY\_SPACE**);** //makes the current postion a space

playerPosX**++;** //increments the x by one

draw\_character**(**playerPosX**,** playerPosY**,** AVATAR**);** //updates new position with the Avatar

**}**

**else** **if(**MAZE**[**playerPosX**-**1**][**playerPosY**]** **!=** WALL **&&** calc\_roll**(**movingAvg**)** **<** 0**)** **{** //moves to the left

draw\_character**(**playerPosX**,** playerPosY**,** EMPTY\_SPACE**);** //current position becomes space

playerPosX**--;** //decrements x by one

draw\_character**(**playerPosX**,** playerPosY**,** AVATAR**);** //updates the new position with the Avatar

**}**

**else** **if(**playerPosX **+** 1 **==** WALL **&&** playerPosX **-** 1 **==** WALL **&&** playerPosY **+** 1 **==** WALL**)** **{** //if all spaces around avatar are walls then end the game

**break;**

**}**

**if(**MAZE**[**playerPosX**][**playerPosY**+**1**]** **!=** WALL**)** **{** //if the space below is empty then move avatar down one

draw\_character**(**playerPosX**,** playerPosY**,** EMPTY\_SPACE**);**

playerPosY**++;**

draw\_character**(**playerPosX**,** playerPosY**,** AVATAR**);**

**}**

**}**

**}while(**playerPosY **!=** 81**);**

**if(**playerPosY **!=** 81**)** **{**

printf**(**"YOU LOSE\n"**);**

**}**

**else** **{**

printf**(**"YOU WIN!\n"**);**

**}**

/\* This function is used to cleanup the Ncurses environment.

Without it, the characters printed to the screen will persist

even after the progam terminates \*/

endwin**();**

**}**

**}**

double m\_avg**(**double buffer**[],** int avg\_size**,** double new\_item**)**

**{**

int k**;**

**for** **(**k **=** 0**;** k **<** avg\_size **-** 1**;** k**++)**

buffer**[**k**]** **=** buffer**[**k**+**1**];**

buffer**[**k**]** **=** new\_item**;**

double sum **=** 0.0**;**

**for** **(**k **=** 0**;** k **<** avg\_size**;** k**++)**

sum **+=** buffer**[**k**];**

**return** sum**/**avg\_size**;**

**}**

/\* 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.

DO NOT NEED TO CHANGE THIS FUNCTION. \*/

void draw\_character**(**int x**,** int y**,** char use**)**

**{**

mvaddch**(**y**,**x**,**use**);**

refresh**();**

**}**

double calc\_roll**(**double mag**)** **{**

**if(**mag **>** 1.0**)** **{** // sets the input to 1 if it is greater than 1

mag **=** 1**;**

**}**

**else** **if(**mag **<** **-**1.0**)** **{** //sets the input to negative one if it is less than -1

**return** mag **=** **-**1**;**

**}**

**else** **if** **(**mag **>** 0.0 **&&** mag **<** 1.0**)** **{** //returns rad times 39 for positive numbers

**return** mag**;**

**}**

**else** **if(**mag **<** 0.0 **&&** mag **>** **-**1.0**)** **{** //returns rad for negative numbers

**return** mag**;**

**}**

**}**

void generate\_maze**(**int difficulty**)** **{**

int i**,** j**;**

**for(**i **=** 0**;** i **<**COLUMNS**;** **++**i**)** **{** //loops throught the column

**for(**j **=** 0**;** j **<** ROWS**;** **++**j**)** **{** //loops through every space under the current column until done

**if(**rand**()** **%** 100 **<** difficulty**)** **{**

draw\_character**(**j**,** i**,** WALL**);** //if the random number is less than the difficulty then a wall is added (draw to screen with function)

**}**

**else** **{**

draw\_character**(**j**,** i**,** EMPTY\_SPACE**);** //otherwise a space is given

**}**

**}**

**}**

**}**

