DS4 Maze

LAB 8

SECTION C

Akash Patel

SUBMISSION DATE: 4/11/2019

4/11/19

Problem

The objective of this lab was to practice top down program design, develop skills in handling events in a loop and work with 2-d arrays.

Analysis

The program first had to scan in all the data from the controller and then assign it to arrays in order to be used in the program. It then entered a while loop which updated each of its functions. After updating the functions, the program would print out each of the new numbers that the array gathered.

Design

The first part of the design was coding in all of the functions. The first function computed the averages for each of the buffers. This was called into the main. It computed the average for each set of numbers after each while loop iteration. The second function was the maxmin function which replaced each new max and min within the array. The last function was updatebuffer which took old items, shifted them over and then out new_item on the right. The second part of the design was to call all of the functions into the main. The main consists of a for loop that scans and redefines the arrays, a while loop that all of the functions are called into and where the output is actually printed, and then a return 0 which makes the button work.

Testing

I had a lot of problems when it came to testing the program. I had to debug everything because the code would not compile. After compiling there were segmentation errors which I had trouble fixing and in the end I was not able to get the program to run how it should.

Comments

This lab was really difficult in my opinion. I tried getting help with Tas but they were also unable to help me fix my code so I was not able to get it tested. I worked with a friend and tried to write the code the best I could but it still wouldn't work.

1.) Raw data is more erratic and are very large variations in numbers, average data tends to be more smooth with smaller variation in numbers
2.) I used calc_roll function and set parameters so the movement wasn't erratic. I was thinking of using buttons to move the character but I didn't think it was allowed.
3.) I used an if statement where moving was equal to TRUE.
4.) To lose it had to hit a wall and the maze. For that I used else if(MAZE[alt_y][alt_x-1]==WALL&&MAZE[alt_y][alt_x-+] == WALL&&MAZE[alt_y+1][alt_x] == WALL) Strcpy(txt, "YOU LOSE!\n")

Break;

Source Code

<Use NPP Exporter to PASTE source code>

```
/*-----------------------------------------------------------------------------
-                                                SE 185 Lab 08
-             Developed for 185-Rursch by T.Tran and K.Wang
-      Name:
-      Section:
-      NetID:
-      Date:
-----------------------------------------------------------------------------*/

/*-----------------------------------------------------------------------------
-                                                Includes
-----------------------------------------------------------------------------*/
#include <stdio.h>

/*-----------------------------------------------------------------------------
-                                                Defines
-----------------------------------------------------------------------------*/
#define MAXPOINTS 10000

/*-----------------------------------------------------------------------------
-                                                Prototypes
-----------------------------------------------------------------------------*/
/* 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
-----------------------------------------------------------------------------*/
int main(int argc, char* argv[]) {

    /* DO NOT CHANGE THIS PART OF THE CODE */
    double x[MAXPOINTS], y[MAXPOINTS], z[MAXPOINTS];
    double new_x, new_y, new_z;
    double avg_x, avg_y, avg_z;
    int lengthofavg = 0;
    if (argc>1) {
        sscanf(argv[1], "%d", &lengthofavg );
        printf("You entered a buffer length of %d\n", lengthofavg);
    }
    else {
        printf("Enter a length on the command line\n");
        return -1;
    }
    if (lengthofavg <1 || lengthofavg >MAXPOINTS) {
        printf("Invalid length\n");
        return -1;
    }
```

```
    for(int i = 0; i < lengthofavg; i++)
    {
        scanf("%lf, %lf, %lf", &new_x, &new_y, &new_z);
        x[i] = new_x;
        y[i] = new_y;
        z[i] = new_z;
    }

    while(1)
    {
        scanf("%lf, %lf, %lf", &new_x, &new_y, &new_z);

        avg_x = m_avg(x, lengthofavg, new_x);
        avg_y = m_avg(y, lengthofavg, new_y);
        avg_z = m_avg(z, lengthofavg, new_z);

        printf("RAW, %lf, %lf, %lf, AVG ,%lf, %lf, %lf\n", new_x, new_y, new_z,
avg_x, avg_y, avg_z);
        fflush(stdout);
    }

}

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];       //swap everything over and replacing the new
item

    buffer[k] = new_item;


    double sum = 0.0;

    for (k = 0; k < avg_size; k++)
        sum += buffer[k];


    return sum/avg_size;
}



/*--------------------------------------------------------------------------
-                                                SE 185 Lab 08
-             Developed for 185-Rursch by T.Tran and K.Wang
-      Name:
-      Section:
-      NetID:
-      Date:
--------------------------------------------------------------------------*/
```

```c
/*---------------------------------------------------------------------------
-                                                                Includes
---------------------------------------------------------------------------*/
#include <stdio.h>
#include <math.h>
#include <ncurses/ncurses.h>
#include <unistd.h>
#include <string.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 ' '
#define WAIT_TIME 300
#define TRUE 1
#define FALSE 0
#define MAXPOINTS 10000
#define BUFF_LENGTH 10
#define UP 0
#define DOWN 1
#define LEFT 2
#define RIGHT 3
#define NONE 4

/*    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];


/*---------------------------------------------------------------------------
-                                                              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[])
{
    /*       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();

    /* WEEK 2 Generate the Maze */
      //generate_maze(difficulty);
      draw_maze();
      y = 0
      x = COLUMNS/2;

    /* Read gyroscope data and fill the buffer before continuing */
      int b[6];
      int t;
      double x, y, z, avg_x, avg_y, avg_z, xmin, xmax, ymin, ymax, zmin, zmax;
    double arr_x[MAXPOINTS], arr_y[MAXPOINTS], arr_z[MAXPOINTS];


    /* Event loop */
      int ay, ax
      char next_char = MAZE[ay][ax];
    int alt_y = ay;
    int alt_x = ax;
    int last_t, first;
    char txt[25];
```

```
    first = TRUE;
    do
    {

        /* Read data, update average */
              scanf("%d, %lf, %lf, %lf, %d, %d, %d, %d, %d, %d", &t, &x, &y, &z,
&b[0], &b[1], &b[2], &b[3], &b[4], &b[5] );
        if(first == TRUE) {
            last_t = t;
            first = FALSE;
        }
        if(t - last_t >= WAIT_TIME) {
            double moving = TRUE;
            last_t = t;
        } else {
            moving = FALSE;
        }
             updatebuffer(arr_x, BUFF_LENGTH, x);
        updatebuffer(arr_y, BUFF_LENGTH, y);
        updatebuffer(arr_z, BUFF_LENGTH, z);

        avg_x = avg(arr_x, BUFF_LENGTH);
        avg_y = avg(arr_y, BUFF_LENGTH);
        avg_z = avg(arr_z, BUFF_LENGTH);

        maxmin(arr_x, BUFF_LENGTH, &xmax, &xmin);
        maxmin(arr_y, BUFF_LENGTH, &ymax, &ymin);
        maxmin(arr_z, BUFF_LENGTH, &zmax, &zmin);

        int move_dir = NONE;
        int valid = TRUE;

             if(close_to(0.5, 1, x) == TRUE && ax > 0) {
               move_dir = LEFT;
            } else if(close_to(0.5, -1, x) == TRUE && ax+1 < COLUMNS) {
               move_dir = RIGHT;
            }


            if(move_dir == LEFT) {
                if(MAZE[ay][ax-1] == '*') {
                    move_dir = NONE;
                }
            } else if(move_dir == RIGHT) {
                if(MAZE[ay][ax+1] == '*') {
                    move_dir = NONE;
                }
            }

        /* Is it time to move?  if so, then move avatar */
             if(moving == TRUE) {
            draw_character(alt_x, alt_y, next_char);
            draw_character(ax, ay, AVATAR);
            //otherwise it's off-screen
```

```
            if(ay+1 < ROWS) {
                alt_y = ay;

                if(MAZE[ay+1][ax] != '*') {
                    ay++;
                }


                    alt_x = ax;
                    if(move_dir == LEFT) {
                        ax--;
                    } else if(move_dir == RIGHT) {
                        ax++;
                    } else {
                        ax = ax;
                    }
                                if (y != alt_y&&x != alt_x){
                                      y=alt_y;
                                }
                                next_char = MAZE[alt_y][alt_x];
                    }
                     if(ay == ROWS-1) {
                strcpy(txt, "YOU WIN!\n");
                break;
            } else if(MAZE[alt_y][alt_x-1] == WALL && MAZE[alt_y][alt_x+1] == WALL &&
MAZE[alt_y+1][alt_x] == WALL) {
                strcpy(txt, "YOU LOSE!\n");
                break;
            } /*else if(scrub_lines(alt_x, alt_y) == FALSE) {
                strcpy(txt, "YOU LOSE!\n");
                break;
            }*/
        }
        refresh();


    } while(1); // Change this to end game at right time

    /* Print the win message */
       printf("%s",txt);


    /* 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();

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

void generate_maze(int difficulty) {
    srand(time(NULL));
    int n, i, h;
```

```
    for(i=0;i<ROWS;i++) {
        for(h=0;h<COLUMNS;h++) {
            n = rand() % 100;
            if(n < difficulty) {
                MAZE[i][h] = WALL;
            } else {
                MAZE[i][h] = EMPTY_SPACE;
            }
        }
    }

}

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

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 x_mag) {
    double val;
```

```
    if(x_mag <= -1) {
        val = asin(-PI/2);
    } else if(x_mag >= 1) {
        val = asin(PI/2);
    } else {
        val = asin(x_mag);
    }

    return val;
}

double avg(double buffer[], int num_items) {
    double avg = 0;
    int i;
    for(i = 0; i < num_items; i+=1) {
        avg = (avg + buffer[i])/2.0;
    }
    return avg;
}

void maxmin(double array[], int num_items, double* max, double* min) {
    int i;
    for(i = 0; i < num_items; i+=1) {
        if(array[i] > *max) {
            *max = array[i];
        }
        if(array[i] < *min) {
            *min = array[i];
        }
    }
}

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

int close_to(double tolerance, double point, double value) {
    int close;

    if(value > point - tolerance && value < point + tolerance) {
        close = TRUE;
    } else {
        close = FALSE;
    }

    return close;
}

int scrub_lines(int posx, int posy) {
    int safe = FALSE;
```

```
    int limr, liml;

    //check for range on current x line
    int i;
    for (i = posx; i < COLUMNS; i++) {
        if(MAZE[posy][i] == WALL) {
            limr = i;
            break;
        }
    }
    for (i = posx; i > 0; i--) {
        if(MAZE[posy][i] == WALL) {
            liml = i;
            break;
        }
    }
    //check for an opening under and in range
    for (i = liml+1; i < limr; i++) {
        if(MAZE[posy+1][i] != WALL) {
            safe = TRUE;
        }
    }
    if (MAZE[posy+1][posx] != WALL) {
        safe = TRUE;
    }

    return safe;
}
```

Screen Shots

<Number the screenshots and paste here. The point of numbering the screenshots is so that you can refer to them during your discussion in the various parts above. Alternatively, you can include the screenshots in-line with the text above as part of your discussion.>