Lab 8

LAB 8

SECTION 4

Bilal Hodzic

11/12/21

11/12/21

Problem:

Create a moving average function to smooth input data from the controller

Analysis:

Makes ds4 data less erratic

Design:

A function was created that moves all values in array 1 to the left then adds on the new value and then averages the array and outputs the average. This creates smooth movement.

Testing:

Tested the function

Comments:

No comments

Source code:

```
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)
{
                                                    for (int i = avg_size; i > 0; i--){

                                                        buffer[i] = buffer[i-1];
                                                    }
                                                buffer[0] = new_item;
                                                    double sum = 0;
                                                for (int i = 0; i < avg_size; i++){

                                                        sum += buffer[i];
                                                    }
    return sum / avg_size;
}
```

Output:
Outputted to file in zip
Problem:
Using moving average create a character that moves down the screen and through a maze. The character should win when it reaches the bottom and lose when it is unable to move anymore. The character should not be able to move through walls of the maze.
Design:
First all functions were created to do as outlined in the code. These functions were then tested to ensure that they work. Main method code was written to check if the character was intersecting a wall and to move the character depending on the controller tilt as well as a time condition. The win and lose conditions were done at the end.
Testing:
Tested all functions individually and tested main code as more things were implemented.
Comments:
Fun lab
Source code:

```
#include <stdio.h>
#include <math.h>
#include <ncurses/ncurses.h>
```

```
#include <unistd.h>
#include <stdlib.h>
#include <time.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 1


/*----------------------------------------------------------------------------
-




                                                                    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[])
{
                                                                srand(time(NULL));

  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();

/* WEEK 2 Generate the Maze */
                                                                    int diff;
                                                                diff =  atoi(argv[1]);
                                                            generate_maze(diff);
                                                                draw_maze();
/*Read gyroscope data and fill the buffer before continuing */
                                                                int t, q , w, e, r;
                                                                double x, y, z;
                                                            int maxTime = 700;
                                                                int tempT = 700;
                                                                int yPos = 0;
                                                                int xPos = 50;
                                                            double mAvg = 0;
                                                        double xArray[NUM_SAMPLES];
                                                                int slow = 0;
                                                                int oldy =0;
                                                                int oldx = 50;
                                                            int winCon = 0;
                                                                int loseT = 0;
                                                    for (int i = 0; i< NUM_SAMPLES; i++){

                                        scanf(" %d, %lf, %lf, %lf, %d, %d, %d, %d", &t, &x, &y, &z, &q, &w, &e, &r);

                                                                xArray[i] = x;
                                                                    }

/* Event loop */
do
{

   /* Read data, update average */

                                        scanf(" %d, %lf, %lf, %lf, %d, %d, %d, %d", &t, &x, &y, &z, &q, &w, &e, &r);

                                                                loseT = t;

                                                        mAvg = m_avg(xArray, NUM_SAMPLES, x);

                                                                if (mAvg > .2){


                                                                slow++;


                                                                if (slow ==10){



                                                    draw_character(xPos, yPos, EMPTY_SPACE);
```

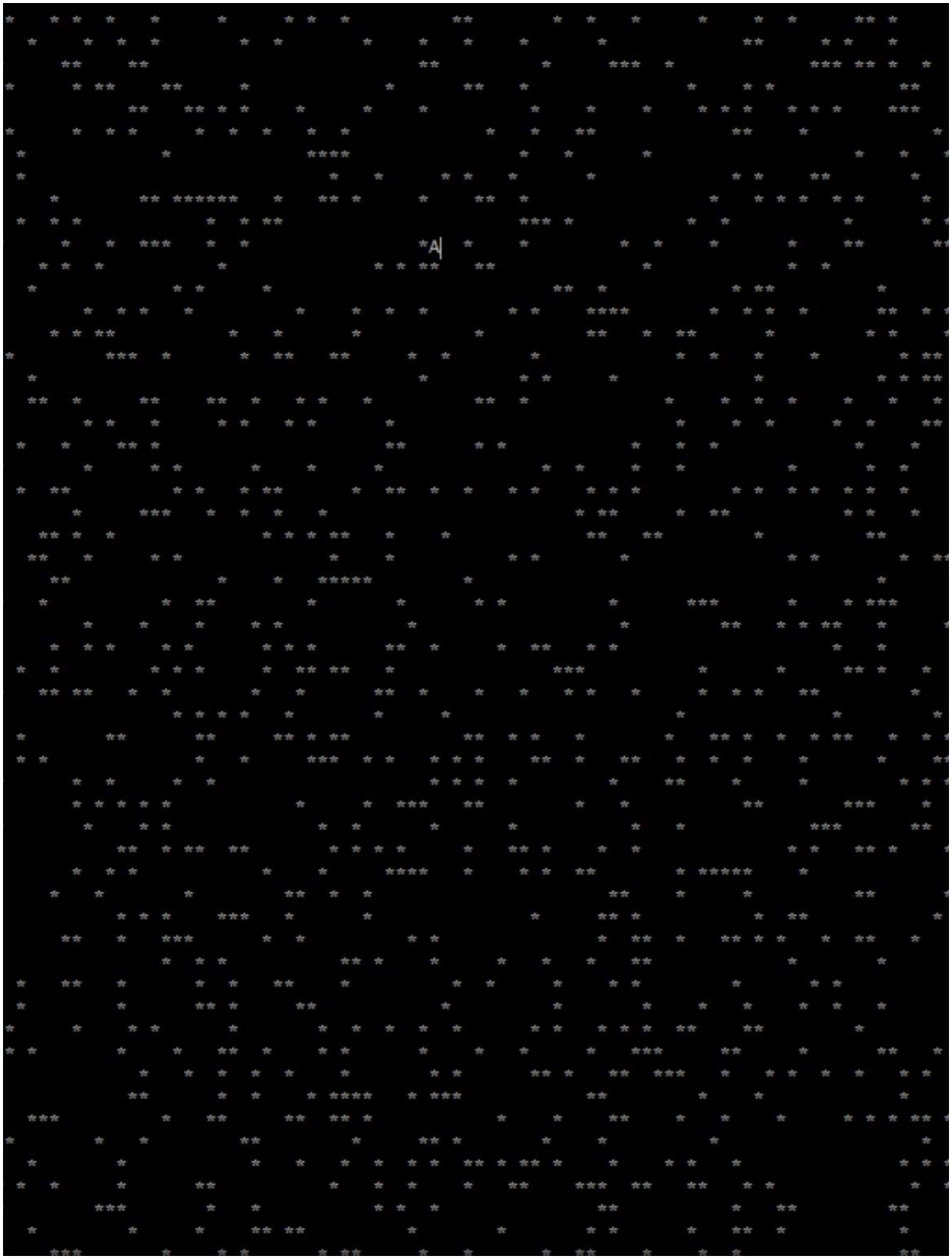
```
                xPos--;

            if (MAZE[xPos][yPos] == WALL){

                xPos = oldx;

            }

            oldx = xPos;

            slow = 0;

        }
    }
    if (mAvg < -.2){

        slow++;

        if (slow ==10){

draw_character(xPos, yPos, EMPTY_SPACE);

            xPos++;

            if (MAZE[xPos][yPos] == WALL){

                xPos = oldx;
```

```
}
oldx = xPos;
slow = 0;
}
}
draw_character(xPos, yPos, AVATAR);
/* Is it time to move?  if so, then move avatar */
if (t >= tempT + maxTime){
draw_character(xPos, yPos, EMPTY_SPACE);
yPos++;
if (MAZE[xPos][yPos] == WALL){
yPos = oldy;
}
tempT = t;
oldy = yPos;
}
if (MAZE[xPos + 1][yPos] == WALL && MAZE[xPos - 1][yPos] == WALL && MAZE[xPos][yPos + 1] == WALL){
winCon = 1;
while (t< loseT + 1000){
```

```
                                        scanf(" %d, %lf, %lf, %lf, %d, %d, %d, %d", &t, &x, &y, &z, &q, &w, &e, &r);


                                                                draw_character(xPos, yPos, AVATAR);

                                                                                }

                                                                                break;

                                                                                }



    } while(yPos < ROWS); // Change this to end game at right time

    /* Print the win message */


    /* 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();
                                                                        if(winCon == 0){

                                                                    printf("You win!\n");
                                                                        }else{

                                                                    printf("You lose");
                                                                        }


  }
}

double m_avg(double buffer[], int avg_size, double new_item)
{
                                                        for (int i = avg_size; i > 0; i--){

                                                                    buffer[i] = buffer[i-1];
                                                                        }
                                                                buffer[0] = new_item;
                                                                    double sum = 0;
                                                        for (int i = 0; i < avg_size; i++){

                                                                    sum += buffer[i];
                                                                        }
    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();
}
void generate_maze(int difficulty){
                                                                    for (int i = 0; i < ROWS; i++){

                                                                  for (int j = 0; j < COLUMNS; j++){

                                                                       int random = rand()%100;

                                                                       if (random < difficulty){


                                                                          MAZE[j][i] = WALL;

                                                                       }else{


                                                                          MAZE[j][i] = EMPTY_SPACE;


                                                                       }

                                                                    }
                                                                    }
}
void draw_maze(void){
                                                                    for(int i = 0; i < ROWS; i++){

                                                                  for (int j = 0; j < COLUMNS; j++){


                                                                     draw_character(j , i, MAZE[j][i]);

                                                                    }
                                                                    }
}
```

Output screenshot:



Questions:

1. Raw data is choppy and changes a large amount very frequently while smooth data does not change much and moves much smoother.
2. Character moves down at a set rate. Left and right movement must be above a certain tilt as well as having to be above that tilt for a certain amount of iterations.
3. Checked if the avatar could move by comparing the new x value and y value to the maze array and checking to see if there was a wall present. If there was the y value would return back to the old value.
4. Checked if the character x value plus one and minus one was a wall as well as checking if the y value plus one was a wall. If all are true the game ends.