Maze Lab

LAB # 8
SECTION # 2

Jaden Burke

SUBMISSION DATE: 10/6/2022

DATE: 10/6/2022

Problem

I must design a program in C that will randomly generate a maze and then move an avatar through said maze using user input and a moving average. The maze will have walls that should stop the user avatar and there will be a win and lose condition that depends on if that avatar made it to the bottom of the screen or got stuck before it could. Lastly, the user should be able to read a user input to determine the difficulty of the maze.

Analysis

- The first part of the lab is the moving average. This can be accomplished pretty simply with a for loop and a few extra statements.
- The next part is rudimentary movement. This can be done by taking user input and inputting it into the already written moving average function and depending on the input, the avatar will move left, right, or down. The user input will be the x axis of the controller as the idea behind the movement of the avatar is very similar to the last lab. The frequency of the movement is done through using a second time variable.
- The next challenge of the lab is generating the maze. This can be done pretty easily with a nested for loop to just fill the 2D array. The function will take the user input for difficulty in the command line as part of the input, soI can use the rand function to decide whether a space is a wall or not. The drawing part is as simple as just drawing the matrix value with the indexes as the x and y.
- The second to last problem is effectively collision. The avatar needs to know whether or not if where it is trying to move is either a wall or out of bounds. This can be done pretty easily using the addition of more variables and some if statements in the move part of the main function.
- Lastly is the win and lose condition. This is super easy in that winning is just if the y value of the avatar gets above 80. Losing is a little more difficult, however this can be done by just checking the left.right, and below for if all of them are a wall or not.

Design

- The moving average can be completed mainly through the use of a for loop. Every iteration will add the value of the current index to a holder variable and then set the index to the next index. Since this requires the for loop to end one iteration early so that it doesn’t go out of bounds for the array, once the loop ends I can just add the new value that is part of the input to the end of the array and to the holder variable. Finally I just return the average by dividing the holder by the length of the array.
- The rudimentary movement is very simple since all I need to do is put the x gyroscope value from the controller and using the average that the function written above returns decide what movement to do. I chose a pretty arbitrary value of .8 for this and depending on if it was above .8 or below -.8 it would either move left or right. If the x was neither of those things it would move down. The

frequency of the movement is pretty easy. I just need a second holder time value that starts at 0 and is only redefined inside the if statement after its decided its going to try and move. I just check to see if the current time - the past time is greater than a certain value, 500 in this case, and then continue to let it try and move or not.

- The maze generation is again pretty simple. Through the use of a nested for loop I can run through all the rows and columns of the maze matrix. Within each individual cell I check whether a randomly generated number mod 100 is below the given difficulty value. If it is it becomes a wall and if it isnt it becomes empty space. The drawing of the maze is super easy because as stated above in the analysis its just drawing whatever is in a given cell of the matrix at with its row as the y and column as the x.
- Collision is next up and again not super difficult. It can be done by making a set of variables futureX and futureY that will be incremented before the actual x and y and used to check if the movement that is being attempted is allowed or not. This is done by checking to see if whatever is at the next spot is an empty space and is not out of bounds. If it is move, otherwise stay where the avatar is.
- The win condition is the easier of the two here and is as simple as checking to see if the y of the avatar is higher than in this case 80 and then break out of the loop. The loss is a little more challenging but it just take an if statement to see if there is a wall to the left, right, and below of the avatar and break out of the loop. I then have a variable that is set as 1 if there was a win and 0 if a loss and depending on what it is once the while loop has been broke I print the correct message.

Testing

For the most part everything ran as intended, however I ran into some minor issues. I really did not understand very well how using the values in argc worked but after being helped by TA's I figured it out. I only had one issue other than that. I got the x and y mixed up in the maze draw function because I put it in row column format even though it shouldve been the other way around due to the draw function taking x then y as input. This caused me to be confused for a great while because everything looked sound but it was a pretty simple error.

Comments

Question 1 for part A: The difference between the raw data and the average is the the average is a better and more consistent value. Due to it being kind of difficult to hold the controller perfectly steady, the moving average is a better way to read values since it is taking the average of the raw data values over time.

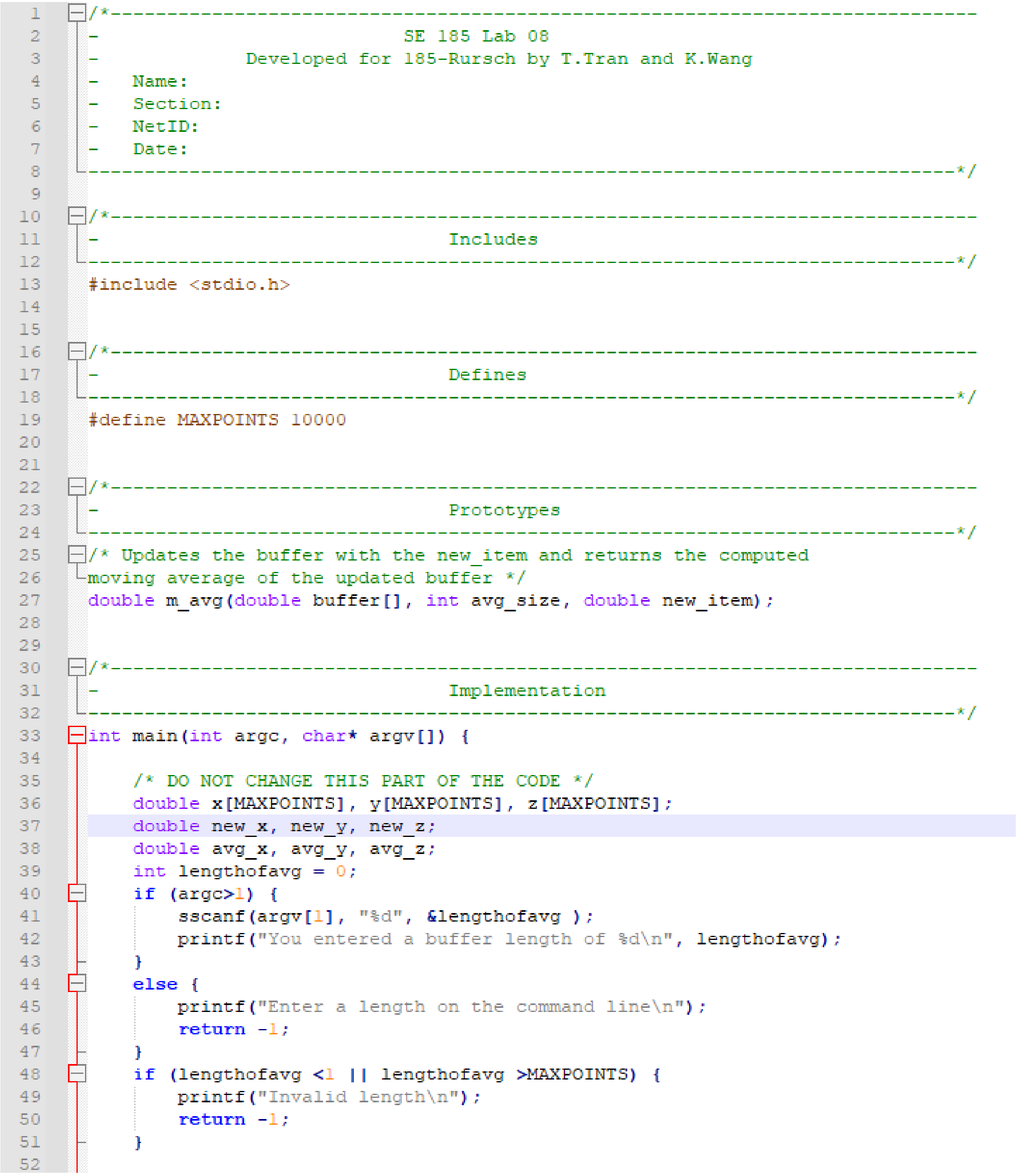
Question 2 for part A: I used the same method as I have in past lab and just had a holder for the time a draw happened and not redrawing unless the difference between current time and that last draw time is great enough.

Question 1 for part B: As stated above I used variables that were incremented accordingly depending on the attempted movement and used the new values to check if the avatar is allowed to move or not. This was done by checking if the future position was both in empty space and not out of bounds and allowing movement if both conditions were true

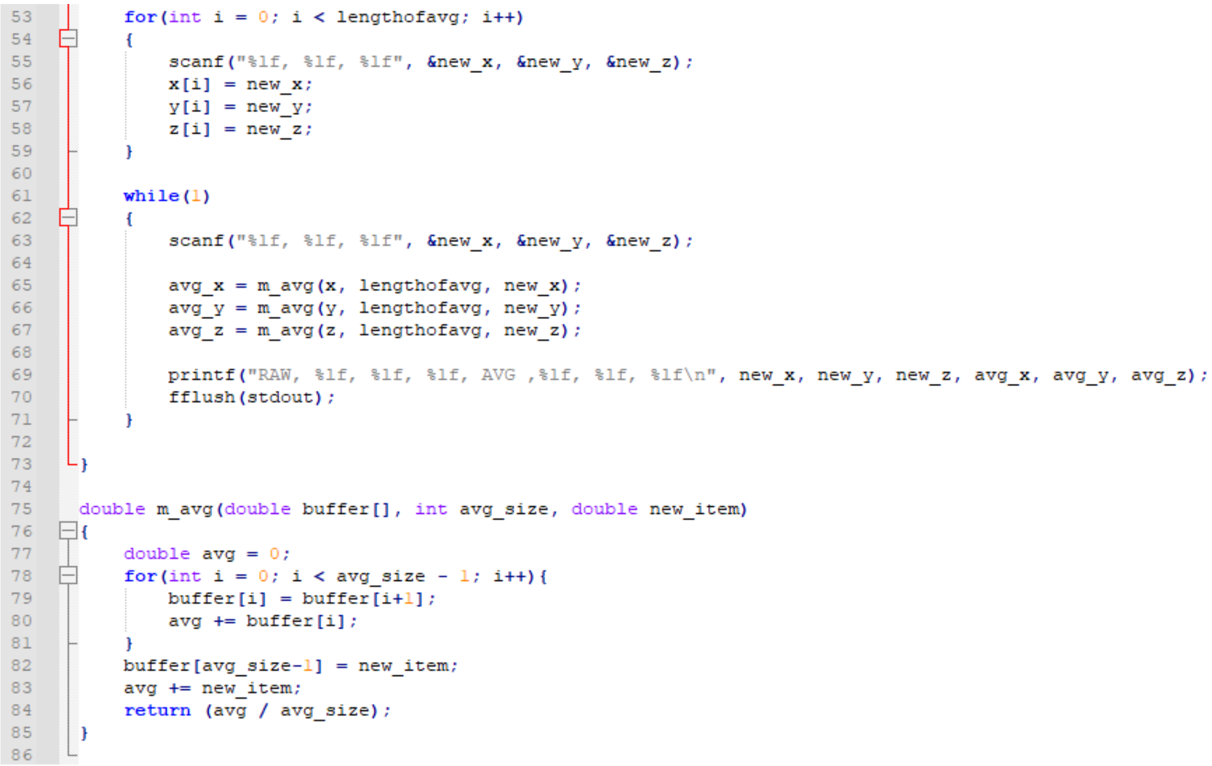
Question 2 for part B:I just used an if statement with two && inside of it to check if left right and bottom were all walls.

Screen Shots

Source code 8.1

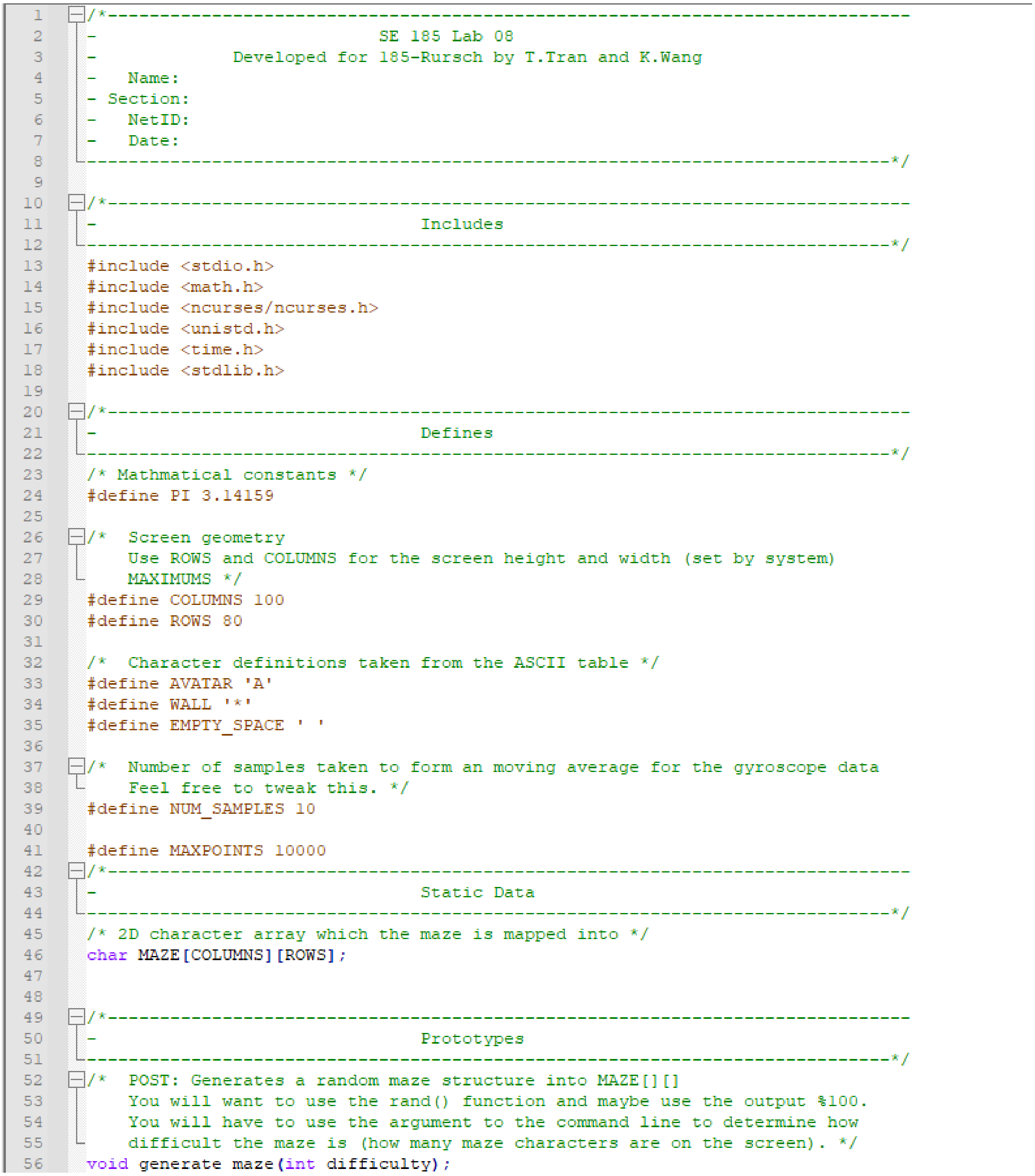


```c
/*-----------------------------------------------------------------------------
-                           SE 185 Lab 08
-             Developed for 185-Rursch by T.Tran and K.Wang
-   Name:
-   Section:
-   NetID:
-   Date:
-----------------------------------------------------------------------------*/

/*-----------------------------------------------------------------------------
-                               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;
    }

```

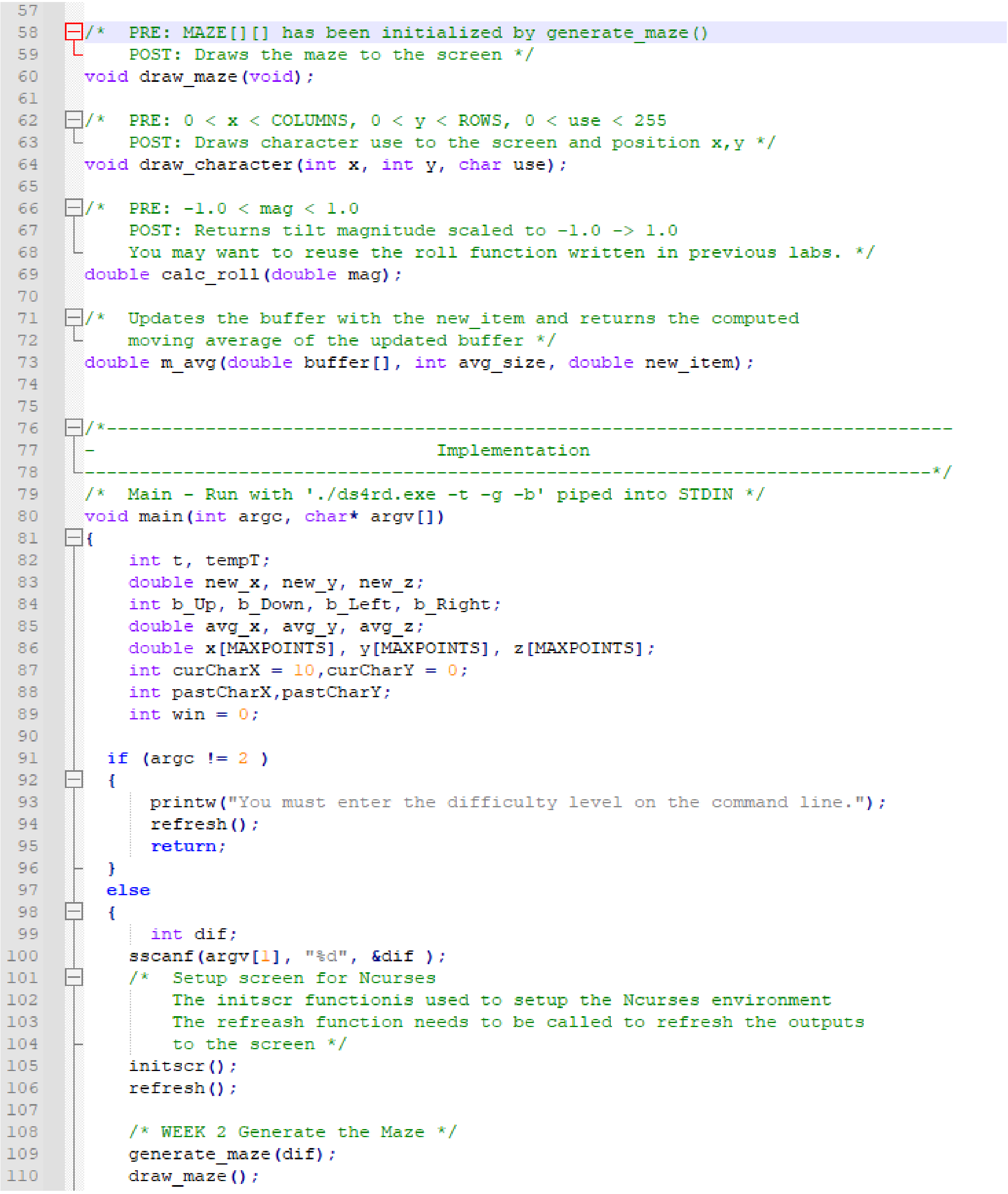


```c
    for(int i = 0; i < lengthofavg; i++)
    {
        scanf("%lf, %lf, %lf", &new_x, &new_y, &new_z);
        x[i] = new_x;
        y[i] = new_y;
        z[i] = new_z;
    }

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

```

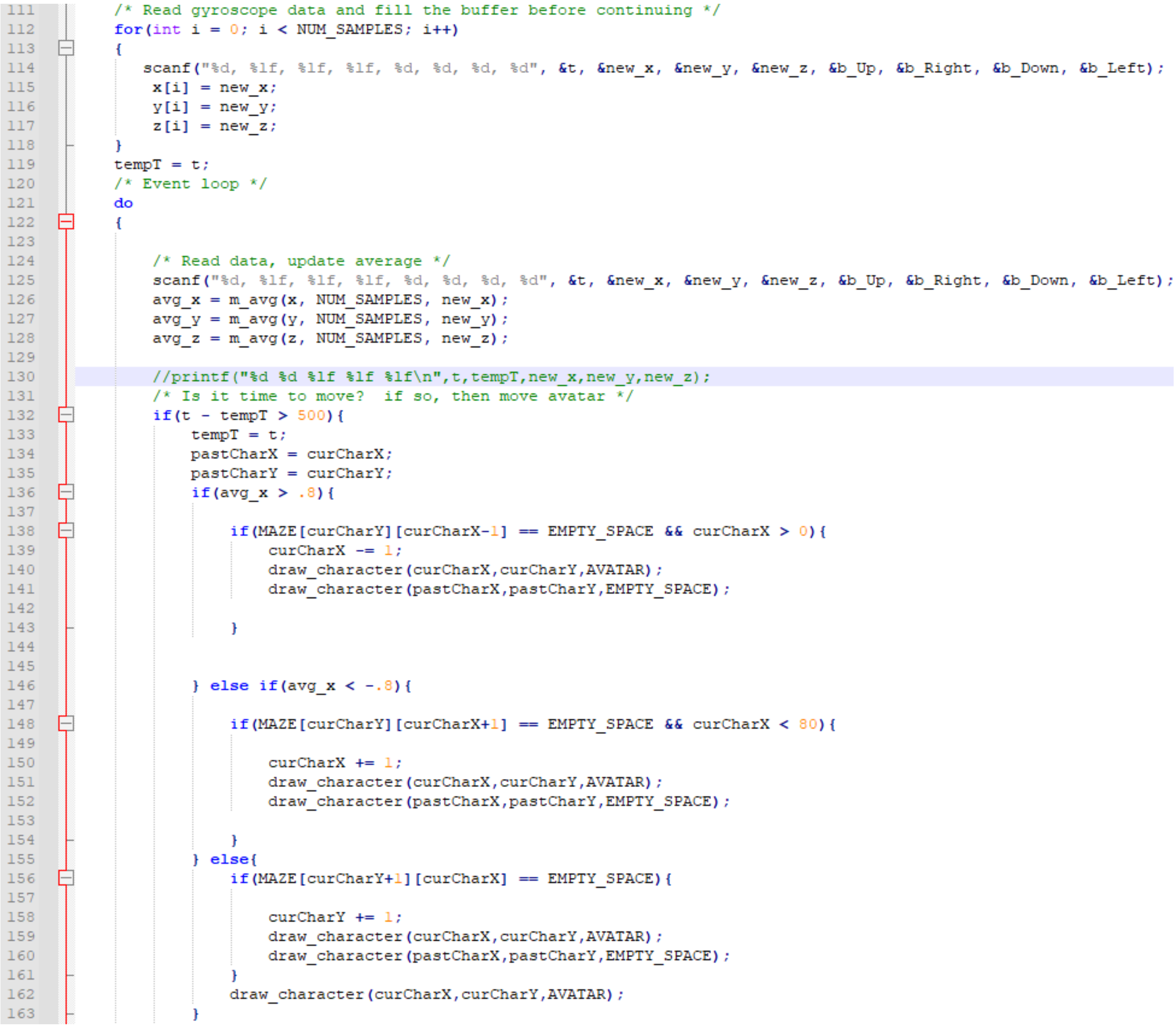
Source code for 8.2



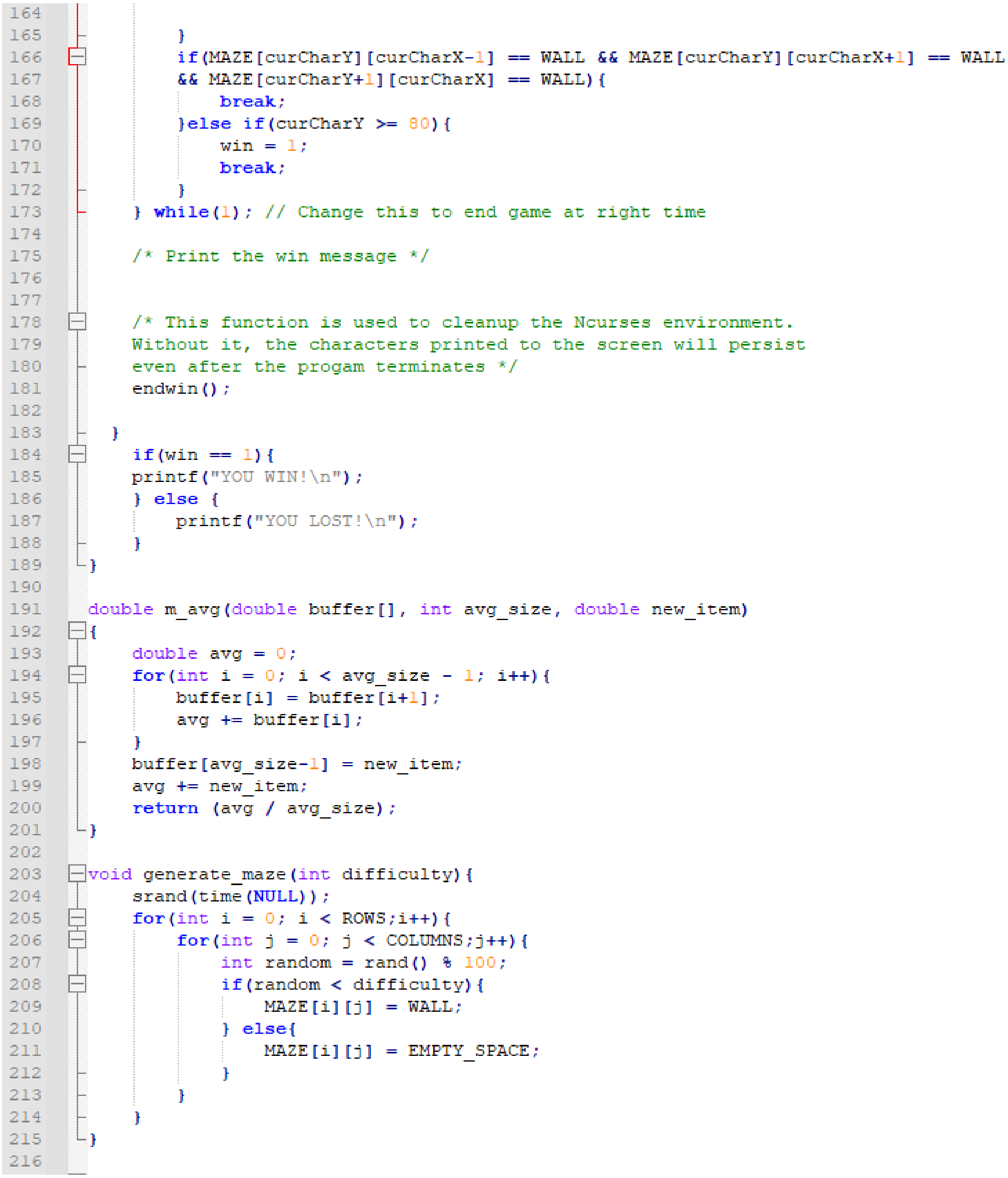
```
/*------------------------------------------------------------------------------
-                           SE 185 Lab 08
-             Developed for 185-Rursch by T.Tran and K.Wang
-   Name:
- Section:
-   NetID:
-   Date:
-----------------------------------------------------------------------------*/

/*------------------------------------------------------------------------------
-                               Includes
-----------------------------------------------------------------------------*/
#include <stdio.h>
#include <math.h>
#include <ncurses/ncurses.h>
#include <unistd.h>
#include <time.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

#define MAXPOINTS 10000
/*------------------------------------------------------------------------------
-                               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);
```



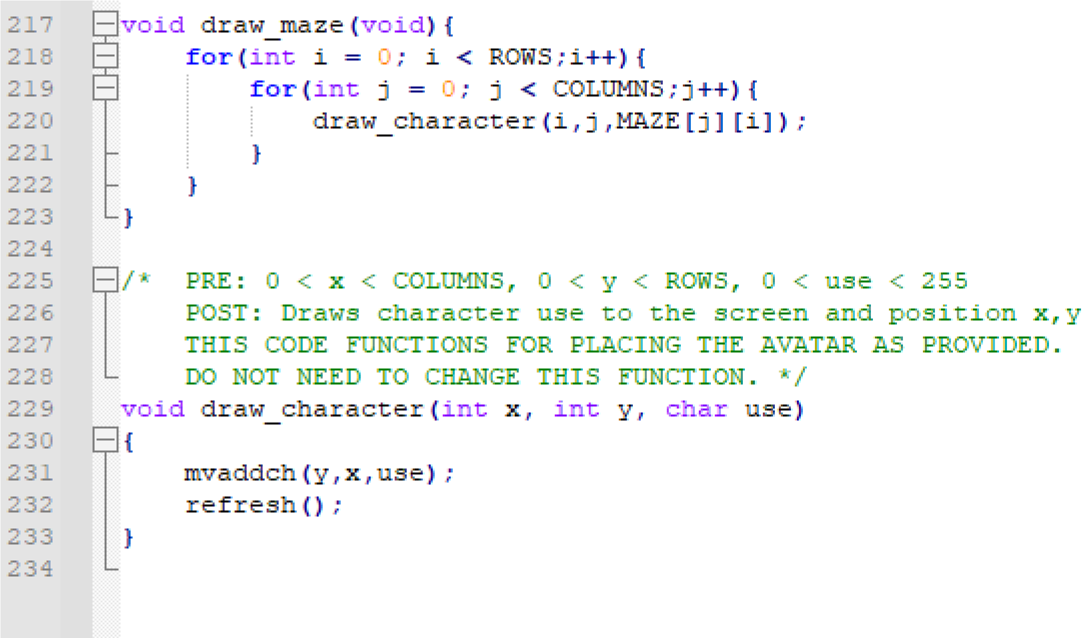
```
/*  PRE: MAZE[][] has been initialized by generate_maze()
    POST: Draws the maze to the screen */
void draw_maze(void);

/*  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[])
{
    int t, tempT;
    double new_x, new_y, new_z;
    int b_Up, b_Down, b_Left, b_Right;
    double avg_x, avg_y, avg_z;
    double x[MAXPOINTS], y[MAXPOINTS], z[MAXPOINTS];
    int curCharX = 10,curCharY = 0;
    int pastCharX,pastCharY;
    int win = 0;

  if (argc != 2 )
  {
      printw("You must enter the difficulty level on the command line.");
      refresh();
      return;
  }
  else
  {
      int dif;
    sscanf(argv[1], "%d", &dif );
    /*  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(dif);
    draw_maze();
```



```
/* Read gyroscope data and fill the buffer before continuing */
for(int i = 0; i < NUM_SAMPLES; i++)
{
   scanf("%d, %lf, %lf, %lf, %d, %d, %d, %d", &t, &new_x, &new_y, &new_z, &b_Up, &b_Right, &b_Down, &b_Left);
    x[i] = new_x;
    y[i] = new_y;
    z[i] = new_z;
}
tempT = t;
/* Event loop */
do
{

    /* Read data, update average */
    scanf("%d, %lf, %lf, %lf, %d, %d, %d, %d", &t, &new_x, &new_y, &new_z, &b_Up, &b_Right, &b_Down, &b_Left);
    avg_x = m_avg(x, NUM_SAMPLES, new_x);
    avg_y = m_avg(y, NUM_SAMPLES, new_y);
    avg_z = m_avg(z, NUM_SAMPLES, new_z);

    //printf("%d %d %lf %lf %lf\n",t,tempT,new_x,new_y,new_z);
    /* Is it time to move?  if so, then move avatar */
    if(t - tempT > 500){
        tempT = t;
        pastCharX = curCharX;
        pastCharY = curCharY;
        if(avg_x > .8){

            if(MAZE[curCharY][curCharX-1] == EMPTY_SPACE && curCharX > 0){
                curCharX -= 1;
                draw_character(curCharX,curCharY,AVATAR);
                draw_character(pastCharX,pastCharY,EMPTY_SPACE);

            }


        } else if(avg_x < -.8){

            if(MAZE[curCharY][curCharX+1] == EMPTY_SPACE && curCharX < 80){

                curCharX += 1;
                draw_character(curCharX,curCharY,AVATAR);
                draw_character(pastCharX,pastCharY,EMPTY_SPACE);

            }
        } else{
            if(MAZE[curCharY+1][curCharX] == EMPTY_SPACE){

                curCharY += 1;
                draw_character(curCharX,curCharY,AVATAR);
                draw_character(pastCharX,pastCharY,EMPTY_SPACE);
            }
            draw_character(curCharX,curCharY,AVATAR);
        }
```



```c

        }
        if(MAZE[curCharY][curCharX-1] == WALL && MAZE[curCharY][curCharX+1] == WALL
        && MAZE[curCharY+1][curCharX] == WALL){
            break;
        }else if(curCharY >= 80){
            win = 1;
            break;
        }
    } while(1); // 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(win == 1){
    printf("YOU WIN!\n");
    } else {
        printf("YOU LOST!\n");
    }
}

double m_avg(double buffer[], int avg_size, double new_item)
{
    double avg = 0;
    for(int i = 0; i < avg_size - 1; i++){
        buffer[i] = buffer[i+1];
        avg += buffer[i];
    }
    buffer[avg_size-1] = new_item;
    avg += new_item;
    return (avg / avg_size);
}

void generate_maze(int difficulty){
    srand(time(NULL));
    for(int i = 0; i < ROWS;i++){
        for(int j = 0; j < COLUMNS;j++){
            int random = rand() % 100;
            if(random < difficulty){
                MAZE[i][j] = WALL;
            } else{
                MAZE[i][j] = EMPTY_SPACE;
            }
        }
    }
}

```



```c
void draw_maze(void){
    for(int i = 0; i < ROWS;i++){
        for(int j = 0; j < COLUMNS;j++){
            draw_character(i,j,MAZE[j][i]);
        }
    }
}

/*  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();
}
```