The A-Mazing DS4 Race

LAB # 8

SECTION # 1

Jesus Horacio Soto Gonzalez

SUBMISSION DATE:

11/11/2022

Problem

This was a really complex two-week lab problem that required the use of the DS4 controller and different functions in order to generate a maze. The DS4 controller was then used to navigate an avatar to the bottom of the screen by avoiding the obstacles generated in the maze. Along with the functions to generate the maze we utilized functions to move the character and change the game difficulty.

Analysis

Since this was a two-week lab, it was divided in different parts to make the programming process more understandable.

In part one, we created a function that helped us with the overall motion of our avatar making smoother and easier to visualize.

In part two, we were provided with a function that draws our character and we just had to jump straight into the actual movement of the avatar by using more functions that later will be combined with the maze to adjust the detection of obstacles and limits.

In part three, we start generating and drawing our maze with two other functions.

Finally in part four, we incorporated all the functions together to appropriately combine the movement of our character and the obstacles presented in the maze as well as the limits of our screen. We also included some special situations like getting stuck which means that the user lost the game or reaching the bottom of the screen which means that the user won.

Design

As I mentioned before the design of this program was divided in different parts, the overall skeleton provided included key elements for this game that we just had to manipulate or add in order to create the actual game. We started the design of this program with a function that was supplied in the source code as a prototype: **double m_avg(double buffer[], int avg_size, double new_item);** this function helped with the overall smoothness of the game movements. The definition of this function consists of two for loops with some nested if and else statements in order to update and average some DS4 input values.

A function to draw our character was also provided, and it was arguably the most important function when it comes to the game since is the one that allow us to print our avatar in the desire spot according to the inputs of movement and time. For the movements of the avatar, I utilize a familiar function from previous labs, **calc_roll** which with the help of the **m_average** function and **draw_character** let the user interact with the game. To do this I had to integrate the three functions into a loop that checked the inputs, movement of character, and the obstacles in the maze.

We also had to define two more functions in order to make this program work, **generate_maze** and **draw_maze**. This to functions are also essential for this program. The **generate_maze** function consist of two for loops and a nested if and else statements which randomize the position of the

obstacles in our maze according to the difficulty provided. Since it sets the obstacles in random positions, srand(), time(), and rand() were critical for this function definition. Next, the **draw_maze** function was in charge of actually printing the obstacles in the screen and is defined by two for loops that set the desire obstacles according to the random position of the two-dimension array according to the previous function.

In the main function we start off by scanning the difficulty entered by the user with the **sscanf();** function compatible with ncurses, after that the program also scans some DS4 data for averages to determine the roll of the controller which then tells us if it moves the avatar or not. Then the program generates and draws the maze according to the difficulty entered. The program reads the DS4 data ones again and the loop starts in which the program keeps scanning the controllers' movements and the avatars movements at the same time. The avatar will keep moving down unless the user gets the A to the bottom where the rows end using a while loop that prints YOU WIN once the 80th row is reached (while r < 80) or the character gets stuck and is unable to move the program ends and prints You Lose.

The mentioned loop starts by calculating the appropriate time that dictates the speed the at which character moves through the maze. Then the loop checks for obstacles either below or to the sides depending on the users input of direction. When the loop detects that there is no obstacle in the path that the user signals, it displaces the character one row and/or column erasing the previous character location at the same time. The loop does not finish until the user reaches the bottom of the screen or gets stuck with no way to move either to the sides or below.

Testing

Testing was an essential part of this lab. Every function incorporated or modification in the loop had to be tested and checked in order to verify that everything was working correctly. During testing I had to test every small step such as every movement direction, every character erased. Since at the beginning I had difficulties erasing the previous position of the character, to fixed that I had to make some adjustments in the loop to delete previous positions every time the character moved. Then the movement of the character was extremely fast, and I just needed to adjust the times in order to fix it. It was incorporating everything together and small details that made the testing process of this lab really important.

Comments

This was a rally hard lab but fun at the same time. It definitely utilizes all the knowledge we have gained so far in order to make this work. I also found myself asking for advice to my peers and TAs multiple times.

1. Explain the differences between the raw data and the averaged data in your

graph for part A: The graphs show the immense differences in outputs thanks to the **m_average** function which as explained before helps to provide a smoother feel to the whole game by eliminating

the errors or spikes of data that the raw can provide. By reducing those spikes in the controller data the program is able to analyze the information way better.

2. Explain the delay you used to ensure character movement is not erratic: The delay in character movement was controlled by the scanned time which is then compared to a secondary timer and increased in order to obtain the desire time between each movement of the character.

3. Describe how you checked if the avatar could safely move down, and go

left/right: Depending on the users input of direction the program checks if the space next and/or below the avatar is a wall, if it is not the character can advance in that direction, but if the space is occupied by a wall the program does not let the character advance and has to find another direction to move.

I wrote the code with the help of **the calc_roll** function for user direction with the new average value obtained by the **m_avg** function and allowing the character to move a column and/or row only if there is no wall with some logic operators like && and !=.

```
if(calc_roll(new_x) < -0.5 && MAZE[c + 1][r] != WALL && c < 99){

                    if(MAZE[c + 1][r + 1] != WALL){
                        r++;
                        draw_character(c + 1, r, AVATAR);
                        draw_character(c, r - 1, EMPTY_SPACE);
                        c++;
                    }
                    else{
                        draw_character(c + 1, r, AVATAR);
                        draw_character(c, r, EMPTY_SPACE);
                        c++;
                    }
```

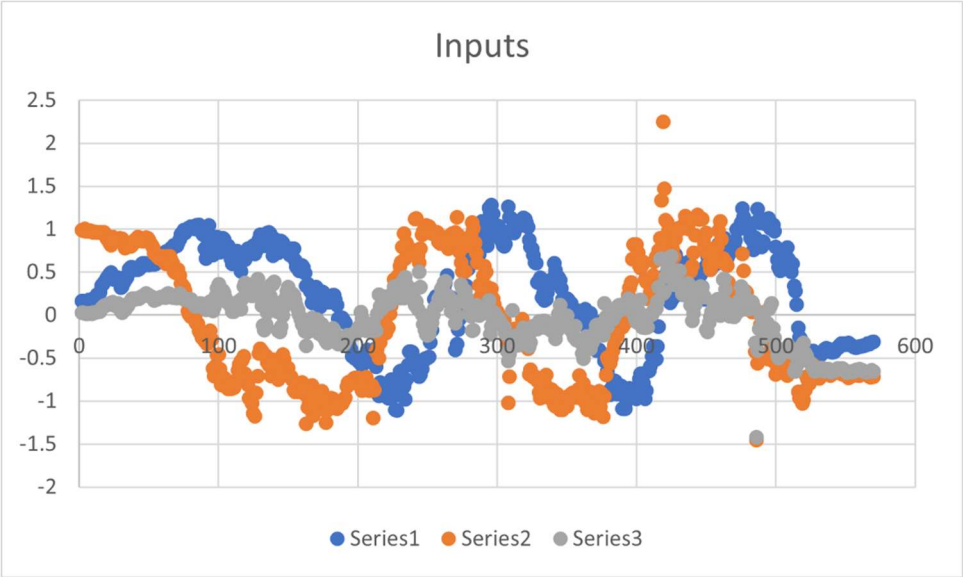
4. Describe what was necessary to check for the player losing the game: In order for the player to lose there are three different positions that need to be checked, ones if the character is unable to move to the right. Second, if the character is unable to move to the left. And third, if the character is unable to go down.

In order to program this, I had to use the logic operator && to ensure that the three positions were included and only if those positions are truly block the player loses.

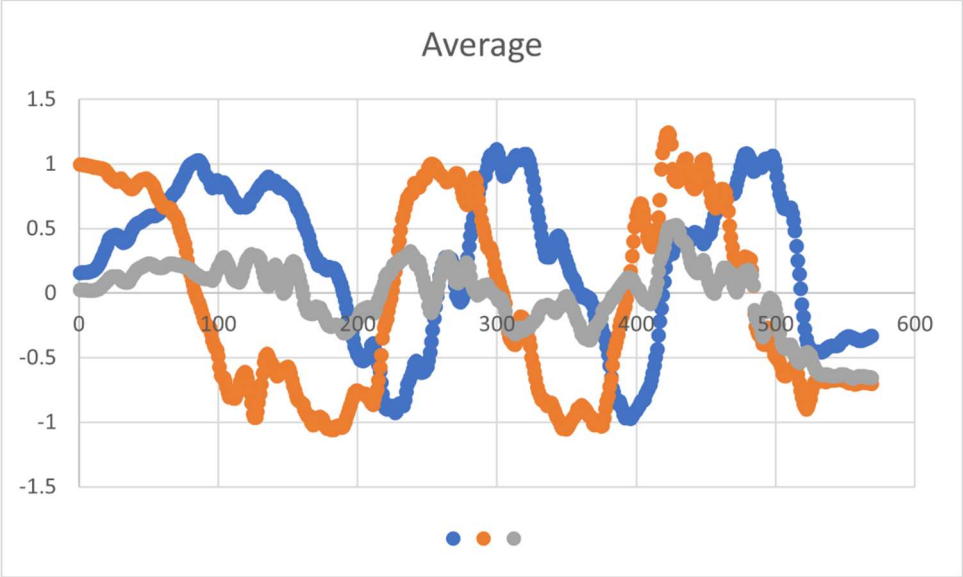
```
else if (MAZE[c + 1][r] == WALL && MAZE[c - 1][ r ] == WALL && MAZE[c][ r + 1] == WALL){
                    endwin();
                    printf("You Lose!\n");
                    exit(1);
                    }
```

Screen Shots

SS #1: Raw Data

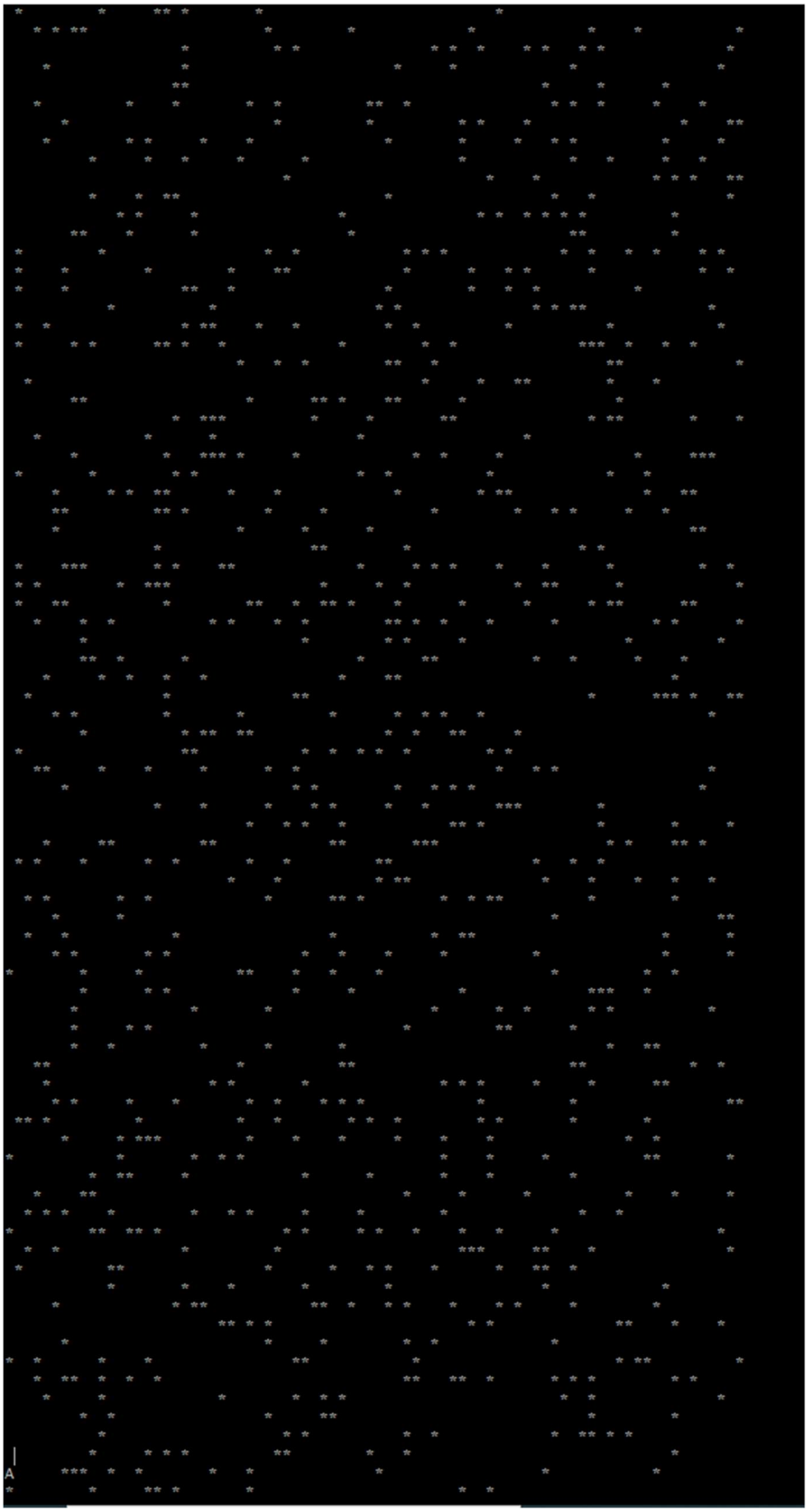


SS #2: Average Data



SS #3: In game 1

SS #4: In game 2



SS #5: Win

```
jhsoto@CO1318-22 /cygdrive/u/fall2022/se185/lab08
$ ./ds4rd-real.exe -d 054c:09cc -D DS4_USB -t -g | ./lab8_2 12
YOU WIN!
```

SS #6: Lose

```
jhsoto@CO1318-22 /cygdrive/u/fall2022/se185/lab08
$ ./ds4rd-real.exe -d 054c:09cc -D DS4_USB -t -g | ./lab8_2 12
You Lose!
```