DS4 Maze LAB REPORT

LAB #8

SECTION M

SUBMITTED BY:

SHOUNAK LAHIRI

SUBMISSION DATE:

11/15/2018

Part 1:

Problem

The goal for both of the parts of this lab is to create a simple real time maze game that will be controlled by the DualShock 4 controller. For part 1 of the lab the problem consists of generating a random maze that reflects the difficulty level that the user enters. Additionally, this part of the lab should make the avatar move down by 1 line after a certain amount of time, and the program should print that the user has won when the avatar hits the bottom of the screen.

Analysis

When the program is run the avatar should start in the middle of the screen. The difficulty that the user enters should be used in the random generator in some way. The skeleton code that is provided for this lab includes a function called draw_character which takes the values of the location of where you would like to draw the character in the order of x, y. Throughout this lab the mvaddch is used which take the location of a character in the order y, x. When working on the lab keeping track of the coordinates and which order which functions take the coordinates is very important.

Design

The problem presented in this lab can be easily broken down into smaller parts

1. Generate the maze
2. Get the avatar to display in the center of the screen and fall after an appropriate amount of time
3. Display that the user wins after the avatar hits the bottom of the screen

To generate the maze, we needed the user to enter the difficulty and to make use of the gernerate_maze function that was provided in the skeleton code. The first thing we needed to do was to access each value inside the two dimensional array which made up the maze. We accessed each value by using two for loops. After getting to each value we needed to know if that point would be a wall or not, to find this out randomly with the users difficulty in mind we used a random generator. The random generator provided a value between 1 and 100, we used the users difficulty as the condition in an if statement which decided if the point would be a wall. If the value from the random generator was over the users entered difficulty, then the point would not be a wall. This ensured that the maze that was generated was random and it kept the users difficulty in mind.

To print the avatar in the center of the screen, we needed to first, generate the maze, then call the draw_character function and send the half way point on the x axis, and the avatar's character. Before the do loop that was in the skeleton code, we placed a scanf statement and set a variable to the time. After the loop was finished we placed another scanf statement and set another variable to the time. Inside the do loop, we placed an if statement that checked if the endTime – startTime was greater than some value, ensuring that the character would fall after a certain amount of time. Inside the if statement we placed another draw_character call that turned the current position into empty space, and the next line of the same x coordinate into the avatar.

Inside the skeleton code, there was already a printf statement which printed that the user had won the game. To ensure that the printf statement only executed when the avatoar has hit the bottom of the screen, we made the loop run until the avatar hit the bottom of the screen. The while condition checked the y coordinate and made sure it was greater than the max value for the screen.

Testing

To test that the program was working correctly, we change the value of the difficulty to see if the mazes that we generated we randomly creating walls and they were following the difficulty levels that we were entering. Additionally, we let the program run until the avatar hit the bottom of the screen to make sure that the avatar was falling at a constant speed and that the program printed that the user had won after the avatar hit the bottom of the screen.

Comments

I think that the design that I implemented was good because the program worked correctly and it worked in a timely manner. The biggest thing that I learned when working on this lab is to keep track of the coordinates, and the way that functions take in variables.

Part 2:

Problem

In the first part of the lab we were able to get the program to create a random maze based on the user's entered difficulty. We also got the program to start the avatar in the center of the screen and fall down after a delay, then display a win message when the avatar hits the bottom of the screen. For this part of the lab we need to alter the program to include controls for the avatar. The avatar needs to be controlled by the DualShock 4 controller, when it is turned right the avatar should go right and when the controller is turned to the left, the avatar should move to the left. Additionally, the avatar should not be able to go further than the dimensions that were included in the skeleton code.

Analysis

It should be noted that the values for moving right are negative, and the values for moving to the left are positive; gyroscopic values from the controller. The avatar should not be able to move outside of the screen constraints that are listed as constants in the skeleton code, there would need to be some condition that would check to see if the avatar was on the border and not let it continuing moving toward the border. Additionally, it would be important to take a tolerance value into account when trying to figure out if the controller is being tilted.

Design

The problem presented in this lab can easily be broken into smaller parts

1. Make the avatar move left if the controller is tilted left
2. Make the avatar move right if the controller is tilted right
3. Make the avatar not move when on the borders of the screen

To get the program to move the avatar to the left when the controller is tilted to the left, we needed to implement the calc_roll function from a previous lab. Inside of the previously implemented if statement which checked if a certain amount of time had passed, we placed another if then else statement which used the calc_roll function. We knew from previous labs and testing the working version of the program that the value that calc_roll returned when the controller was tilted to the left was positive, which is the opposite of what most people would assume. Using the knowledge that the value was positive, we placed a condition that the value of calc_roll should be greater than .2, which would provided a small tolerance value. Another condition in the same if branch was that the next coordinate, one to the left, was not a wall. This statement got the avatar to move to the left correctly.

Then, we needed the program to move the avatar to the right when the controller was titled to the right. Similar to the process of getting the avatar to move to the left we placed an else if to continue the if branch from the left movement step. We knew from previous labs and testing the working version of the program that the value of calc_roll when the controller was tilted to the right was negative. Inside the else if conditional, we put the condition that the calc_roll value that is returned has to be less than -.2, which accounts for a small tolerance. Additionally, we needed the know that the next coordinate where the avatar would move was not a wall, so placed that condition in the conditional. This got the avatar to move to the right correctly. Another aspect of the program was to make sure that the next position of the avatar on the y-axis was not a wall, for that we placed an if statement after the if branch for the movement, which was responsible for moving the avatar down by 1 if the down by 1 position was not a wall.

The last part of the lab was to make sure that the avatar was not able to move outside of the game area, the screen. To do this we added another condition inside each of the branches of the if statement which was controlling the movement of the avatar. The condition varied for each movement, if the avatar was moving left then the condition stated that if the current x position -1 is greater than zero making sure that there was room to move left, if the avatar was moving to the right then the condition stated that if the current x position +1 is less than 72 making sure there was room to move right. This additional condition ensured that the avatar was restricted to moving only on the game screen.

Testing

To test that the program was working correctly, we played the game a few times. After playing the game a few times we were convinced that the program was working correctly, it was moving the avatar to the left when the controller was tilted to the left and moving the avatar to the right when the controller was tilted to the right. We then needed to see if the avatar was restricted to the game screen, to do this we generated a maze with difficulty of zero, and tilted the controller to the right which eventually showed that there was a border of where the game screen ended and the avatar could not continue to move in that direction. Additionally, we made sure that the win screen was printed after we reached the bottom of the screen.

Comments

I think that the design that I implemented was good because the program worked how it was intended to. The biggest thing that I learned by doing this part of the lab is that you must think about all the cases of what the user might do, and all the ways that they can break your program.

Questions/Experiments

1. In the condition that checked which direction we were moving, we checked that if the calc_roll was above or below some tolerance value, if the next x coordinate (in relation to which direction we were moving) was not a wall, and that we were not outside of the game screen. After we checked that we could move the avatar, we updated the x coordinate. Afterwards, we checked the can we fall condition by checking that the next y coordinate was not a wall, then we updated the screen with the new x and y values.
2. If this was a maze game that the player lost if they hit the walls, we would just need to add else statements to the move conditional and the can I fall conditional. Inside the work part of the else, we could place a break to get out of the loop. Additionally, we could set a flag variable to true. Then outside of the loop, we could create an if statement checking

that the flag variable. If the flag variable is false, then we print the win screen, else if the variable is true then we print a lose screen.

Source Code

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

#define PI 3.14159
#define NUMCOLS 100
#define NUMROWS 72

#define AVATAR 'A'
#define WALL '*'
#define EMPTY_SPACE ' '

char MAZE[NUMROWS][NUMCOLS];

void generate_maze(int difficulty);
void draw_maze(void);
void draw_character(int x, int y, char use);
float calc_roll(float x_mag);

int main(int argc, char* argv[])
{
      if (argc <2) { printf("You forgot the difficulty\n"); return 1;}
      int difficulty = atoi(argv[1]); // get difficulty from first command
line arg
      // setup screen
      initscr();
      refresh();

      int x = 50;
      int y = 0;

      generate_maze(difficulty);
      draw_maze();
      draw_character(x, y, AVATAR);

      int t, b1, b2, b3, b4;
    double ax, ay, az, gx, gy, gz;
      int startTime = 0, endTime = 0, end = 0, flag = 0;

      do
      {

            scanf("%d, %lf, %lf, %lf, %lf, %lf, %lf, %d, %d, %d, %d", &t,
&ax, &ay, &az, &gx, &gy, &gz, &b1, &b2, &b3, &b4 );
```

```
            startTime = t;

            //Moves the AVATAR down each time
            if(startTime - endTime > 500) //Is it time to move?
            {

                  if(calc_roll(gx) > .2 && MAZE[y][x-1] != WALL && x - 1 >=
0) //Is the controller tilted to the left, and the next space not a wall
(can we move left?
                  {
                        MAZE[y][x] = EMPTY_SPACE;
                        x -= 1;
                        draw_character(x,y,AVATAR);
                        draw_character(x + 1, y, EMPTY_SPACE);
                  }
                  else if (calc_roll(gx) < -.2 && MAZE[y][x+1] != WALL && x
+ 1 <= 100)//Is the controller tilted to the right, and the next space not
a wall (can we move right)?
                  {
                        MAZE[y][x] = EMPTY_SPACE;
                        x += 1;
                        draw_character(x,y,AVATAR);
                        draw_character(x - 1, y, EMPTY_SPACE);
                  }

                  if(MAZE[y +1 ] [x] != WALL) //Is the next space not a
wall?
                  {
                        y += 1;
                        draw_character(x , y, AVATAR);
                        draw_character(x, y-1, EMPTY_SPACE);
                  }

                  scanf("%d, %lf, %lf, %lf, %lf, %lf, %lf, %d, %d, %d, %d",
&t, &ax, &ay, &az, &gx, &gy, &gz, &b1, &b2, &b3, &b4 );
                  endTime = t;
            }

      }while(y < NUMROWS); // Are we still inside the screen?

      endwin();
      //Did the user win?
            printf("YOU WIN\n");


}
```

```
void draw_character(int x, int y, char use)
{
      mvaddch(y,x,use);
      refresh();
}


void generate_maze(int difficulty)
{

      int i, j;
      for(i = 0 ; i < NUMROWS; i++)
      {
            for(j = 0; j < NUMCOLS; j++)
            {
                  if(rand() % 100 >= difficulty)
                  {
                        MAZE[i][j] = EMPTY_SPACE;
                  }
                  else
                  {
                        MAZE[i][j] = WALL;
                  }

            }

      }
}//End generate_maze function


void draw_maze()
{

      int i, j;
      for(i = 0; i < NUMROWS; i++)
      {
            for(j = 0; j < NUMCOLS; j++)
            {
                  draw_character(j, i , MAZE[i][j]);
            }
      }

}


float calc_roll(float x_mag)
{

      if(x_mag < -1.0)
```

```
        {
                x_mag = -1.0;
        }
        else if (x_mag > 1.0)
        {
                x_mag = 1.0;
        }
        return asin(x_mag);

}
```