DS4 Maze

LAB #8
SECTION 5

SUBMITTED BY:

Jackson Collalti

SUBMISSION DATE:

11/30/2022

Problem

With the introduction of the Ncurse Library and arrays, create a random maze. The maze will have an avatar starting at the top middle of the 72x100 layout. Walls will be represented by *'s, and Avatar will be 'A,' then empty spaces will be left blank. The maze must also have difficulty inputting in the command line. After a certain amount of time, the avatar will fall one line down. Part two of the lab will add horizontal movement using the tilt of the DS4. The avatar must not go offscreen. If the avatar reaches the bottom, print "you win."

Analysis

The program must have difficulty as an input that will decide how many walls are generated. The avatar can't travel through walls and must start top middle of the maze. The avatar can only move through blank spaces that are within the screen bound. After a certain time frame, the avatar will move down. Depending on the tilt (input of gx) the avatar will also move left or right. If the avatar manages to reach the bottom of the screen without getting stuck, the program should end and print you win. Output in this program will be updating the location of the avatar and printing a message saying stuck or you win.

Design

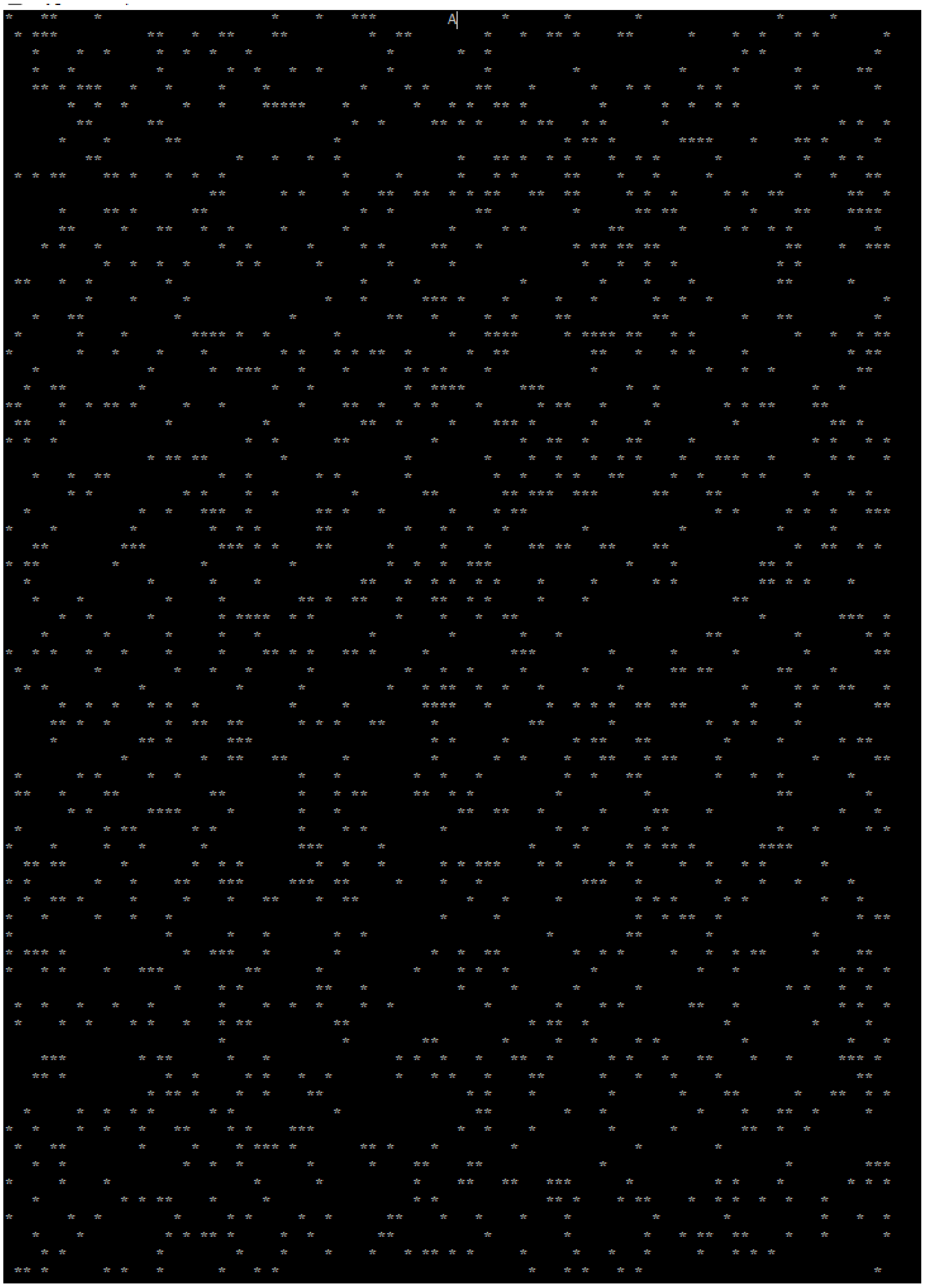
The first thing is to generate the maze. The difficulty will be the percentage of spaces with walls. For example, a difficulty of 25 will mean that about a quarter of the total spaces will be walls. To both generate and draw the maze, I'll need a nested loop that goes through columns and rows. Each position in the 2d array will either be assigned a * or ' '. For the avatar placement, I will need the avatar to start at numRows / 2. For movement down, I'll need a condition that is used to keep track of the time and will activate after a set time. After the maze and down movement is functional, I'll add a function that checks if the avatar can move to a location by checking to see if the desired position is a wall or not. To add horizontal movement, I'll need to read the gx values and create tolerance values. In order to print you win, I'll need to have the loop that runs all of the code for the tasks above. The condition will end when the avatar reaches the last row. After this condition is met, the program will print the winning message.

Testing

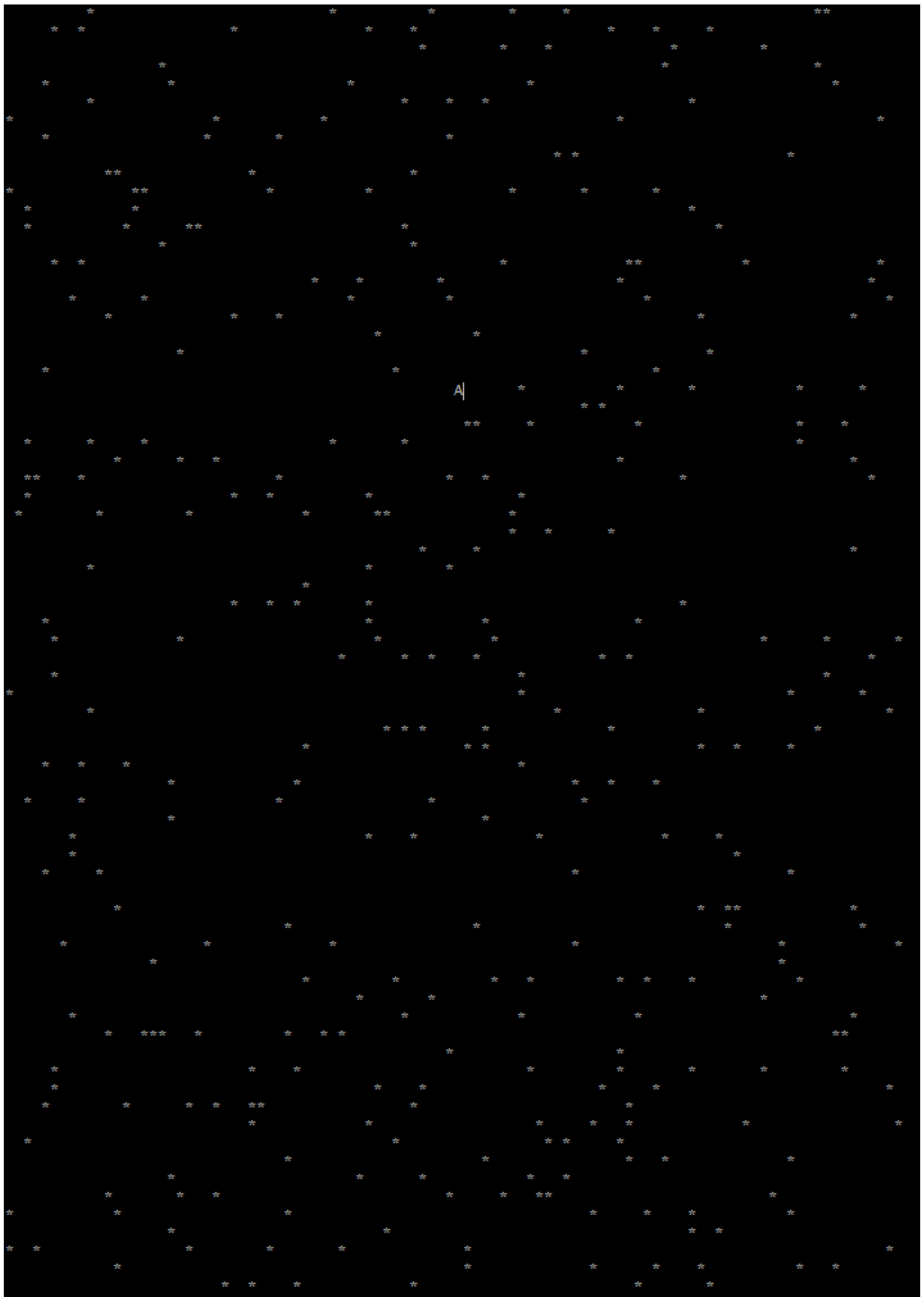
For testing the generate and draw maze functions, I plan on using different difficulties and comparing them to what I was expecting. I plan on testing using values 0-100 and two values over 100 to ensure that no errors occur in a generation. To test verticle movement is simple either it moves or doesn't move. To test horizontal movement, I'll need to test what gx values I decide on fall straight down, and what values are left and right. This will be up to what I feel is usable.

To test out a win and stuck messages, I'll need to test to see if they come up when put into a relevant situation.

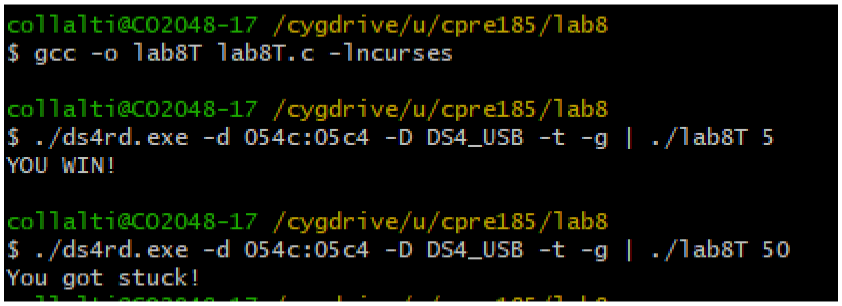
Maze Generated with a difficulty of 50



Maze Generated with a difficulty of 5 in progress.



Print messages of win and lose.



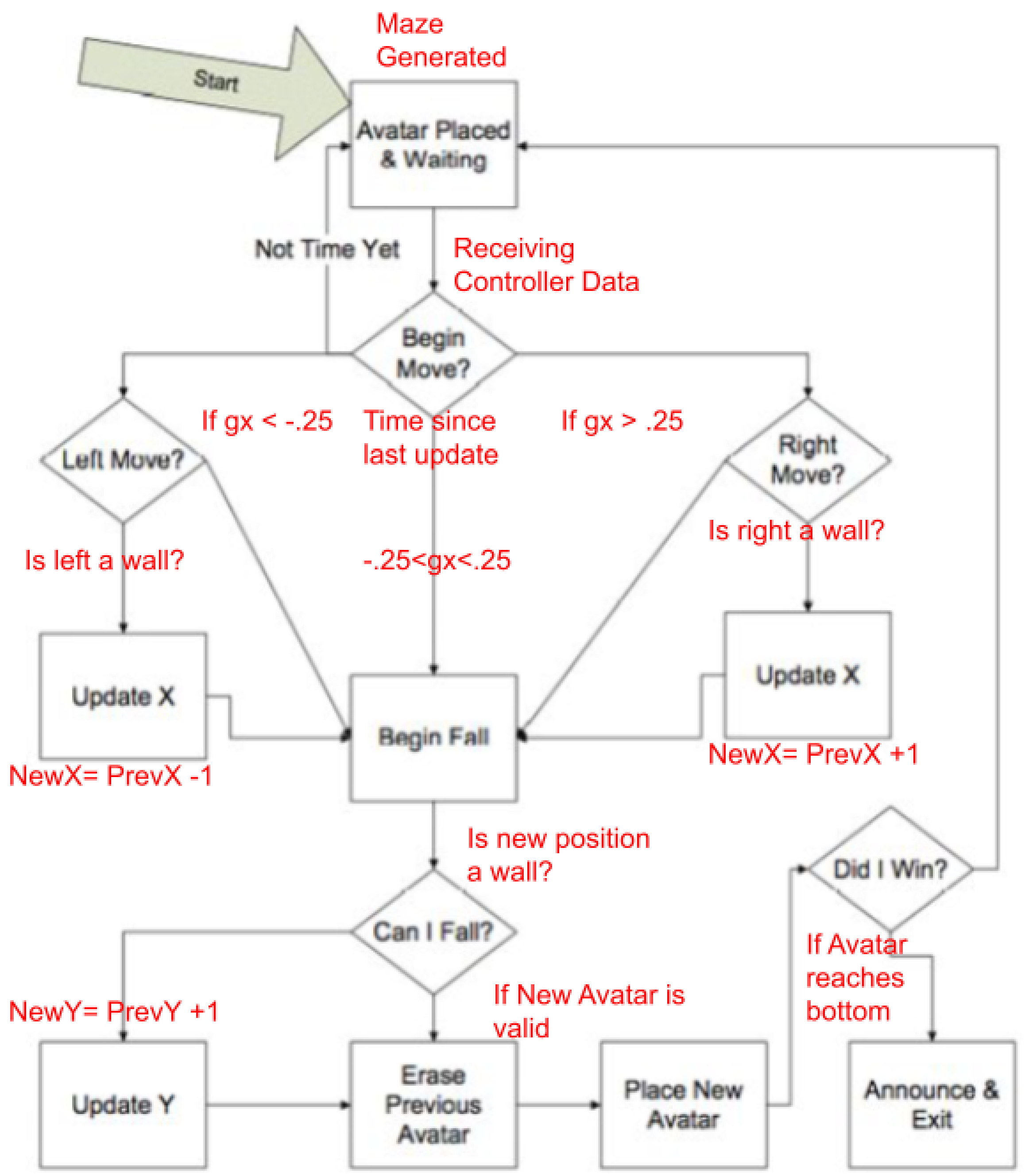
```
collalti@CO2048-17 /cygdrive/u/cpre185/lab8
$ gcc -o lab8T lab8T.c -lncurses

collalti@CO2048-17 /cygdrive/u/cpre185/lab8
$ ./ds4rd.exe -d 054c:05c4 -D DS4_USB -t -g | ./lab8T 5
YOU WIN!

collalti@CO2048-17 /cygdrive/u/cpre185/lab8
$ ./ds4rd.exe -d 054c:05c4 -D DS4_USB -t -g | ./lab8T 50
You got stuck!
```

Comments

The key thing I learned is improving my ability to work with 2d arrays. Originally I had issues with creating the down movement because logically, I saw down as a negative value, but in the array, that's positive. This was an easy thing to fix just changing Y - - to Y++.To generate the maze was some trial and error without any long-lasting problems. Things I'd change are limited, but if I were to do this lab again, I'd likely try to understand more about the Ncurse library.

1. In the "Safe to Go RIGHT/LEFT" conditions and the "Can I fall" conditions, document what is checked and how.
For the left and right, I need to make sure that the desired position isn't a wall and is within the window. My conditions for the right movement were that the new spot isn't a wall and is less than the max columns. For left movement, I need the same wall condition, but this time, the array value must be greater than 0 to avoid going off-screen. For can fall, the is just checking to see if the space below is or isn't a wall.

2. Describe what would be necessary to check for the player losing the game and how you would add it to the state machine.
To check if a player is losing the game is to see if the player has a point where they can't move in any direction (stuck). For the state machine, I would mean that the avatar can't move left and right are both and can fall fails. That means it's impossible for the player to move in any direction, so they can't reach the bottom. Thus they lose.



```
//implementation page
// WII-MAZE Skeleton code written by Jason Erbskorn 2007
// Edited for ncurses 2008 Tom Daniels
// Updated for Esplora 2013 TeamRursch185
// Updated for DualShock 4 2016 Rursch
```

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

// Mathematical constants
#define PI 3.14159

// Screen geometry
// Use ROWS and COLS for the screen height and width (set by system)
// MAXIMUMS
#define NUMCOLS 100
#define NUMROWS 72

// Character definitions taken from the ASCII table
#define AVATAR 'A'
#define WALL '*'
#define EMPTY_SPACE ' '

// 2D character array which the maze is mapped into
char MAZE[NUMROWS][NUMCOLS];

// 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 < COLS, 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 < x_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.
float calc_roll(float x_mag);

int checkStuck(int x, int y);

// Main - Run with './ds4rd.exe -t -g -b' piped into STDIN
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();

	// Generate and draw the maze, with initial avatar
	srand(time(NULL));
	generate_maze(difficulty);
	draw_maze();

	// Read gyroscope data to get ready for using moving averages.
	int t;
	int last = 0;
	double gx, gy, gz;
	int avaX = NUMCOLS / 2;
	int avaY = 0;
	draw_character(avaX, avaY, AVATAR);

	// Event loop
	do
	{
		// Read data, update average
		scanf(" %d, %lf, %lf, %lf", &t, &gx, &gy, &gz);

		// Is it time to move?  if so, then move avatar
		//Begin move?
		if (t - last > 500)
		{
			//Erase old avatar
			draw_character(avaX, avaY, EMPTY_SPACE);
```

```
        //Move left?
        if (0.25 < gx)
        {
                if (avaX > 0 && MAZE[avaY][avaX - 1] != WALL)
                {
                        //Update X
                        avaX--;
                }
        }
        else if (-.25 > gx) //Move right?
        {
                if (avaX < NUMCOLS && MAZE[avaY][avaX + 1] != WALL)
                {
                        //Update X
                        avaX++;
                }
        }

        //Begin fall
        //Can I fall?
        if (MAZE[avaY + 1][avaX] == EMPTY_SPACE)
        {
                //Update Y
                avaY++;
        }

        else if (checkStuck(avaX, avaY))
        {
                endwin();
                printf("You got stuck!");
                return 0;
        }


        //Place new avatar
        draw_character(avaX, avaY, AVATAR);
        last = t;
}
```

```
		//Did I win?
	} while (avaY < NUMROWS - 1); // Change this to end game at right time

	//Announance and exit
	// Print the win message
	endwin();

	printf("YOU WIN!\n");
	return 0;
}



// PRE: 0 < x < COLS, 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 CHANGE THIS FUNCTION.<<<<
void draw_character(int x, int y, char use)
{
	mvaddch(y, x, use);
	refresh();
}

void generate_maze(int difficulty)
{
	int i;
	int j;
	for (i = 0; i < NUMROWS; i++)
	{
		for (j = 0; j < NUMCOLS; j++)
		{
			int r = rand() % 100;

			if (r < difficulty)
			{
				MAZE[i][j] = WALL;
			}
			else
			{
```

```
                        MAZE[i][j] = EMPTY_SPACE;
                    }
                }
            }
        }

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

        refresh();
}

int checkStuck(int x, int y)
{
        int i = x;

        while (i >= 0 && MAZE[y][i] != WALL)
        {
                if (MAZE[y + 1][i] == EMPTY_SPACE)
                {
                        return 0;
                }
                i--;
        }

        i = x;

        while (i < NUMCOLS && MAZE[y][i] != WALL)
        {
                if (MAZE[y + 1][i] == EMPTY_SPACE)
                {
```

```
                return 0;
            }
            i++;
        }

        return 1;
}
```