Sean Poston

CS351

Project 2: A Maze Game

Video Report:

https://youtu.be/HVQyD4IUz-c

Maze Game Program:

https://repl.it/@seanposton4/CS351Project2Maze

It should be noted repl sometimes has issues printing the final stats of move count and run time, but it works fine in NetBeans. The maze is still playable on repl.

Maze Array Generator:

https://repl.it/@seanposton4/Maze-Array-Generator

Abstract

This project is for CS351. The goal of this project was to create a maze game where the user can use the keyboard to control a player character and solve a maze. The project came with specifications such that the maze must be well designed, it must have two entrances and two exits, a player that can be controlled by user input, and the program must count time and moves made by the player. The project is also supposed to be accompanied by a video report on YouTube and this written report.

Introduction

This project was more of a challenge than originally speculated. Most of the difficulty came from find a way to create the actual maze without inputting all 2,500 characters that would encompass a 50 x 50 maze. Originally, the plan was to use Depth-First Search (DFS) in order to make a program with a self-generating maze. However, this proved to be too difficult. The solution to this problem is in the next section “Maze Design.”

After creating the maze, the next step was to figure out a way to create two entrances and two exits. The entrances proved easier than the exits, which was unexpected. The entrances use a simple user input of 1 or 2 at the start of the program to determine at which door to start. After the user traverses the maze, there are two exits to choose from. Both will exit the program as complete and move on to display the time taken and the move count.

The player, represented by ‘o,’ was the first part of the project that was created. It was simple following the logic of use WASD to move, but only allow movement on a certain character. The program uses ‘.’ as the “path character,” where the player can move along, and ‘#’ as the “wall character,” where the player is blocked.

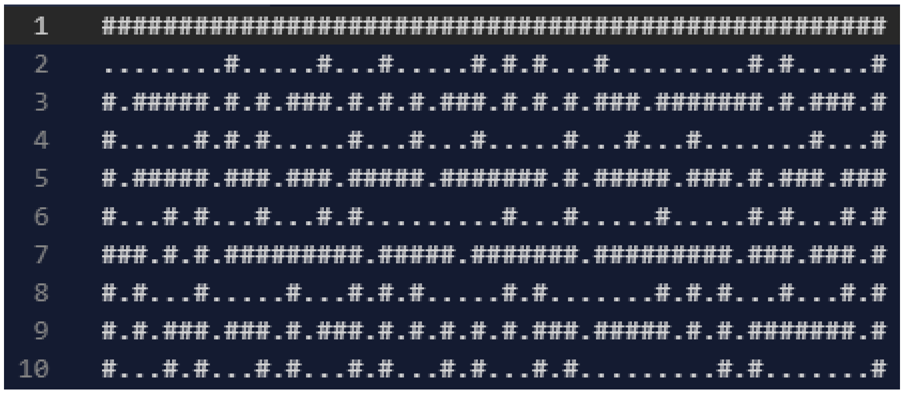
The program runtime and move count were more difficult than anticipated. The move count ticker was easy. Inside each move command (WASD), the moveCount variable was placed with an incrementing operation on it. This means that every time the player could legally move (onto the ‘.’ characters), the moveCount variable would increment by one. The real-world runtime counter was way beyond anything we’ve learned in this course and will be discussed in more detail in the “Time and Step Count” section.

Finally, pointers were what enabled this program to work. Although there weren’t any pointers explicitly declared (e.g. int * var), the use of a 2D array allows the maze to be stored. The array “maze” is declared and initialized inside main. This array name is then used as a pointer for the maze characters in the function mazePlayer. Inside this function, the characters of “maze” are accessed through the addresses pointed to by the array indexes.

Maze Design

The design of the maze was the most difficult part, as was mentioned above. This is due to my refusal to input 2,500 (plus extra characters required to enter it into an array would put the total around 10,000 characters) characters by hand. As was quickly learned, using DFS to generate a maze was extremely difficult. After a few hours of researching this, it was scrapped because the only tutorials of how to do it were in object-oriented languages.

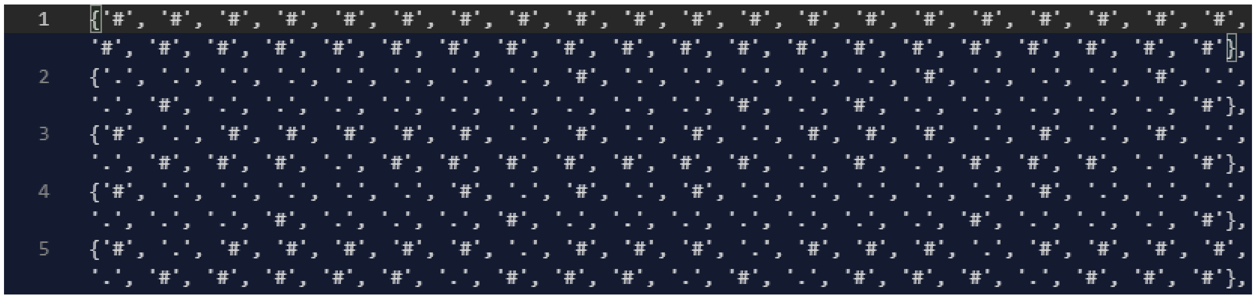
Finally, the idea of using a predesigned maze generator [1] seemed the most reasonable. However, this didn't come without its problems, and it certainly wasn't a free pass for the maze. Using the maze generator output a maze in the form:



```
###################################################
........#.....#...#.....#.#.#...#.........#.#.....#
#.#####.#.#.###.#.#.#.###.#.#.#.###.#######.#.###.#
#.....#.#.#.....#...#...#.....#...#...#.......#...#
#.#####.###.###.#####.#######.#.#####.###.#.###.###
#...#.#...#...#.#.........#...#.....#.....#.#...#.#
###.#.#.#########.#####.#######.#########.###.###.#
#.#...#.....#...#.#.#.....#.#.......#.#.#...#...#.#
#.#.###.###.#.###.#.#.#.#.#.###.#####.#.#.#######.#
#...#.#...#.#...#.#...#.#...#.#.........#.#.......#
```

Figure 1 – Raw Maze Input

All the characters were right by each other, which really only saved 2,500 character inputs. To take this a step further, I wrote a program that read from a text file (Figure 1), and would output it in a form that was able to be placed into an array

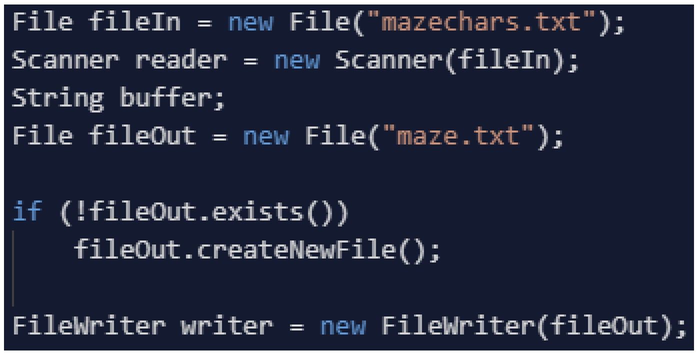


```
{'#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#',
'#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#'},
{'.', '.', '.', '.', '.', '.', '.', '.', '#', '.', '.', '.', '.', '.', '#', '.', '.', '.', '#', '.',
'.', '#', '.', '.', '.', '.', '.', '.', '.', '.', '.', '#', '.', '#', '.', '.', '.', '.', '.', '#'},
{'#', '.', '#', '#', '#', '#', '#', '.', '#', '.', '#', '.', '#', '#', '#', '.', '#', '.', '#', '.',
'.', '#', '#', '#', '.', '#', '#', '#', '#', '#', '#', '#', '.', '#', '.', '#', '#', '#', '.', '#'},
{'#', '.', '.', '.', '.', '.', '#', '.', '#', '.', '#', '.', '.', '.', '.', '.', '#', '.', '.', '.',
'.', '.', '.', '#', '.', '.', '.', '#', '.', '.', '.', '.', '.', '.', '.', '#', '.', '.', '.', '#'},
{'#', '.', '#', '#', '#', '#', '#', '.', '#', '#', '#', '.', '#', '#', '#', '.', '#', '#', '#', '#',
'.', '#', '#', '#', '#', '#', '.', '#', '#', '#', '.', '#', '.', '#', '#', '#', '.', '#', '#', '#'},
```

Figure 2 – Array-Formatted Input

The full source code for this can be found in the Maze Array Generator Repl (https://repl.it/@seanposton4/Maze-Array-Generator).

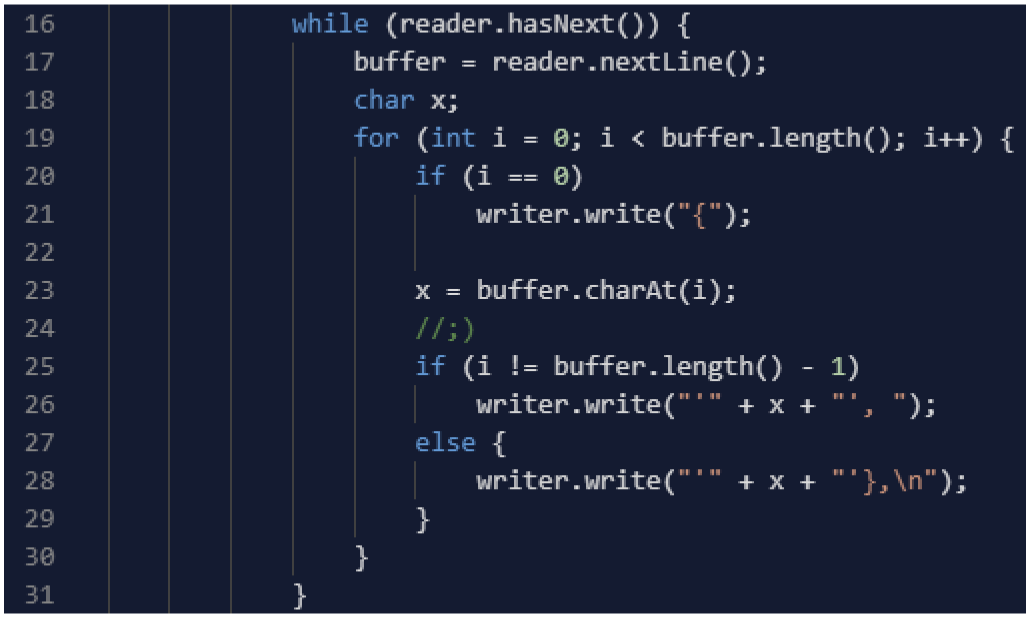
The program reads from the input file (Figure 1) using a Scanner and prints to an output file (Figure 2) using a FileWriter object.



```java
File fileIn = new File("mazechars.txt");
Scanner reader = new Scanner(fileIn);
String buffer;
File fileOut = new File("maze.txt");

if (!fileOut.exists())
    fileOut.createNewFile();

FileWriter writer = new FileWriter(fileOut);
```

Figure 3 – Maze Array Generator I/O

Then the program reads a line of the input file (Figure 1), stores that line into a buffer, and uses a for loop to cycle through each character of the line, adding the required characters to be input into an array.

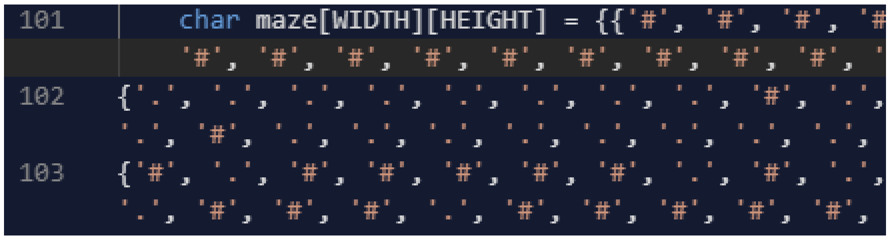


```java
while (reader.hasNext()) {
    buffer = reader.nextLine();
    char x;
    for (int i = 0; i < buffer.length(); i++) {
        if (i == 0)
            writer.write("{");

        x = buffer.charAt(i);
        //;)
        if (i != buffer.length() - 1)
            writer.write("'" + x + "', ");
        else {
            writer.write("'" + x + "'},\n");
        }
    }
}
```

Figure 4 – Maze Array Generator Output Block

There are a few different if statements. First when i = 0, it starts the line with a brace to start that row of the array. Then the proper character is saved into x. In the next if-else block, the program will print the character surrounded by single quotes and followed by a comma, as is required for characters in an array. When the end of the line is reached, it will print a closing brace to signify the closing of the row and a new line to start the next (this is mostly for readability).

Then, it was the simple task of copy-pasting the contents of Figure 2 into the array to create the maze:



```
    char maze[WIDTH][HEIGHT] = {{'#', '#', '#', '#
    '#', '#', '#', '#', '#', '#', '#', '#', '#', '
{'.', '.', '.', '.', '.', '.', '.', '.', '#', '.',
'.', '#', '.', '.', '.', '.', '.', '.', '.', '.',
{'#', '.', '#', '#', '#', '#', '#', '.', '#', '.',
'.', '#', '#', '#', '.', '#', '#', '#', '#', '#',
```

Figure 5 – Maze Game Maze Array

Doors

The doors consist of two entrances and two exits.

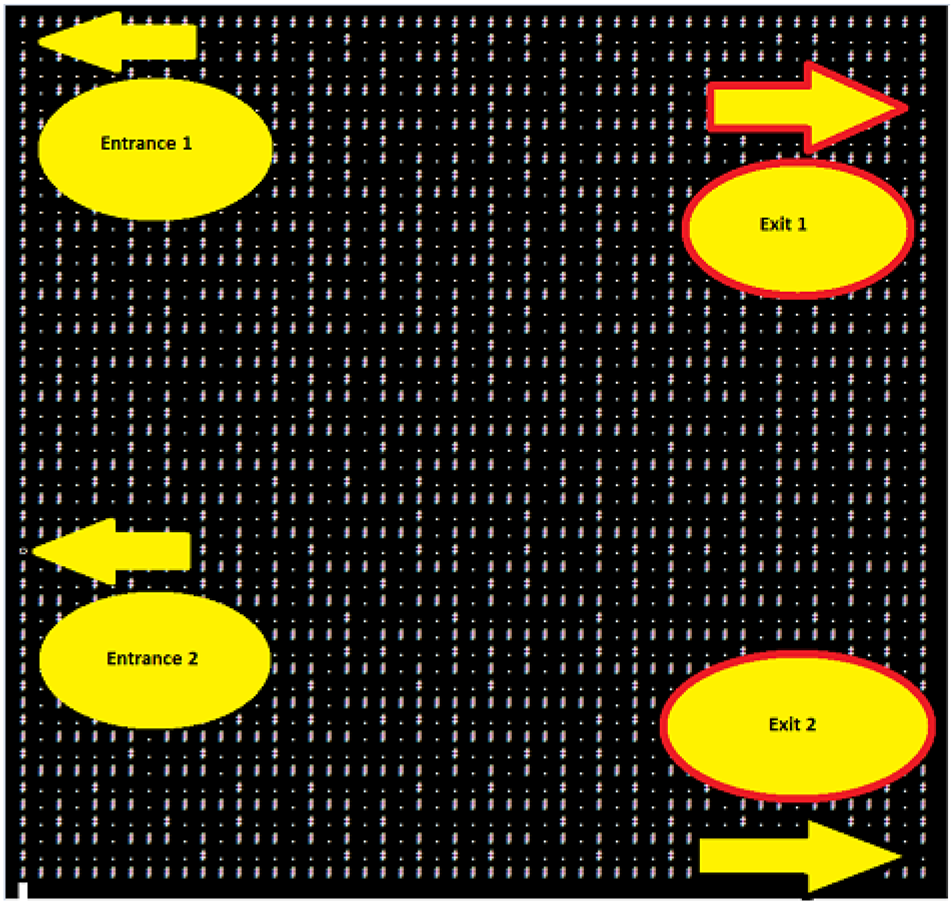
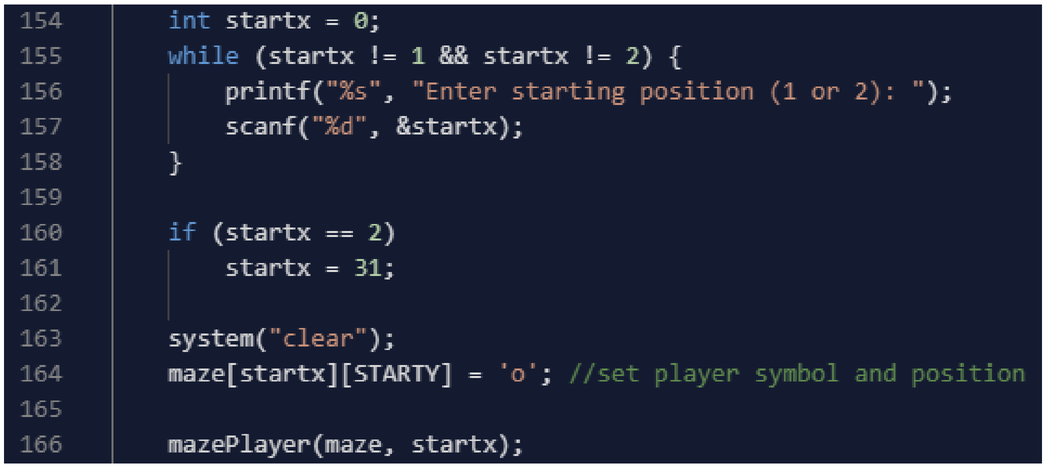


Figure 6 – Entrances and Exits

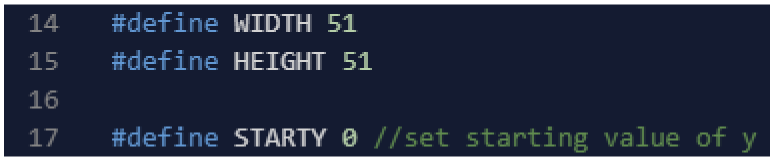
The exits are open for the user to choose whichever they prefer. When the program starts, the user is prompted to enter either 1 or 2 to choose which entrance they prefer.



```
    int startx = 0;
    while (startx != 1 && startx != 2) {
        printf("%s", "Enter starting position (1 or 2): ");
        scanf("%d", &startx);
    }

    if (startx == 2)
        startx = 31;

    system("clear");
    maze[startx][STARTY] = 'o'; //set player symbol and position

    mazePlayer(maze, startx);
```

Figure 7 – Player Entrance Choice

The while loop keeps the user from entering anything other than a 1 or 2. Since Entrance 1 is on row 1, the value of *startx* doesn't need to be changed there. However, Entrance 2 is on row 31 so if the user inputs 2, the program changes the value of *startx* to the proper row of 31. The column is set to 0 as a constant because both starting options are in the same column.



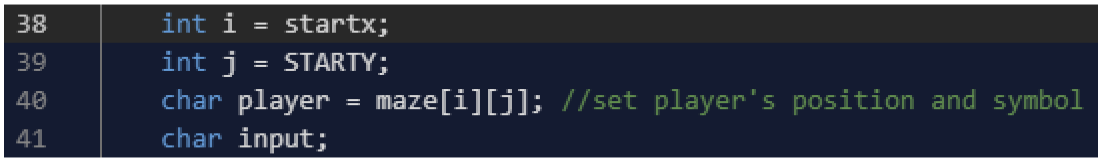
```
#define WIDTH 51
#define HEIGHT 51

#define STARTY 0 //set starting value of y
```

Figure 8 – Setting Y Starting Constant

Figure 7, line 164 is where the player is set into the starting position, and line 166 is where the maze game starts.

Player

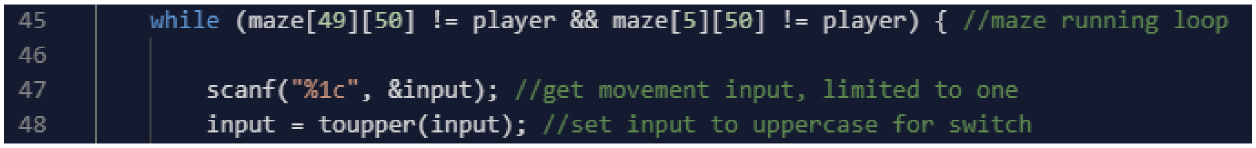
The player brings the whole program together. It allows the maze to be useful and the user to have something to control. To accomplish this, the char *player* is set to be the same as it was in Figure 7, line 164. Then the user's input variable is initialized.



```c
    int i = startx;
    int j = STARTY;
    char player = maze[i][j]; //set player's position and symbol
    char input;
```

Figure 9 – Set Player and Input

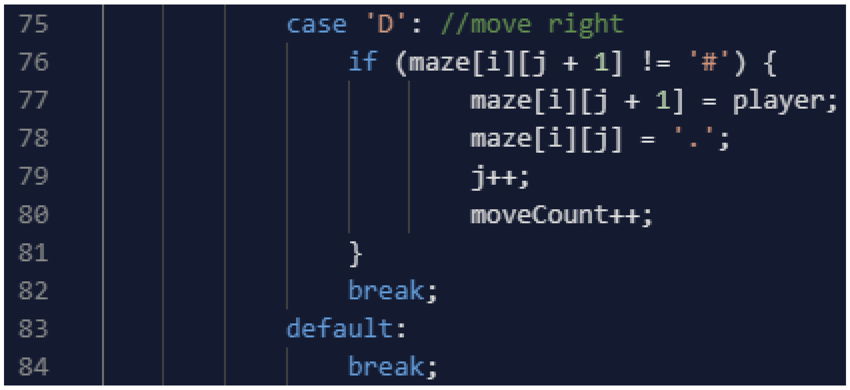
The movement of the player is the most important part of it. The goal was to allow freedom of movement along the track of the maze. The player can move anywhere as long as it's not trying to move into a wall. To accomplish this, the program gets a user input, and only moves if it's W, A, S, or D (capitalization doesn't matter as the input is run through the <ctype.h> library's toupper(char) function). This continues as long as the player isn't at either of the exits.



```c
    while (maze[49][50] != player && maze[5][50] != player) { //maze running loop

        scanf("%1c", &input); //get movement input, limited to one
        input = toupper(input); //set input to uppercase for switch
```

Figure 10 – User Input

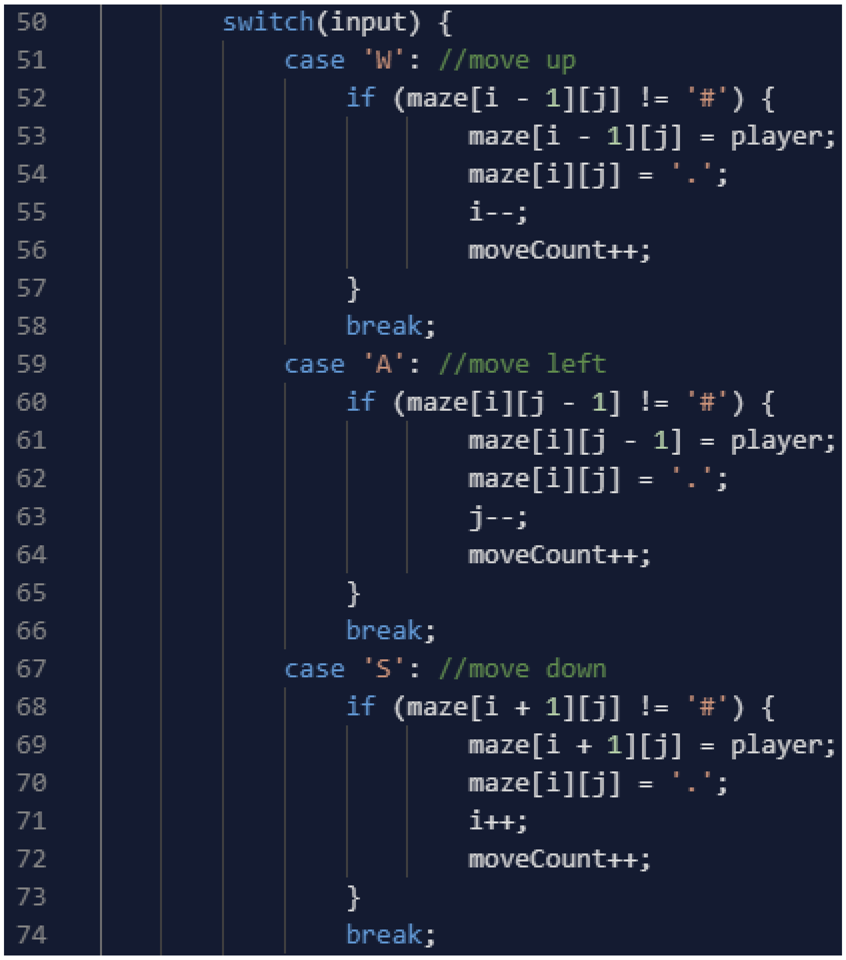
If the input doesn't fall within WASD, then the default case in a switch catches it, and it simply restarts the loop asking for input again. Assuming proper input, the switch catches the input and moves the player to the desired place in the array, assuming there is no wall there. For example, if the user wants to move to the right, then they enter 'D'. If the space to the right of the player isn't a wall, then the program will change that space to the player character, 'o', and replace the previous space with a path character, '.'.



```c
            case 'D': //move right
                if (maze[i][j + 1] != '#') {
                        maze[i][j + 1] = player;
                        maze[i][j] = '.';
                        j++;
                        moveCount++;
                }
                break;
            default:
                break;
```

Figure 11 – User Right Move and Default Case

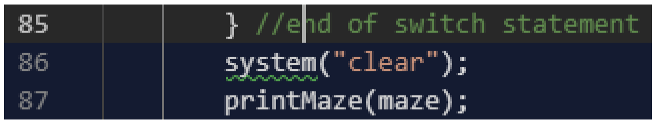
There are, of course, the three other directions to move as well.



```
        switch(input) {
            case 'W': //move up
                if (maze[i - 1][j] != '#') {
                        maze[i - 1][j] = player;
                        maze[i][j] = '.';
                        i--;
                        moveCount++;
                }
                break;
            case 'A': //move left
                if (maze[i][j - 1] != '#') {
                        maze[i][j - 1] = player;
                        maze[i][j] = '.';
                        j--;
                        moveCount++;
                }
                break;
            case 'S': //move down
                if (maze[i + 1][j] != '#') {
                        maze[i + 1][j] = player;
                        maze[i][j] = '.';
                        i++;
                        moveCount++;
                }
                break;
```

Figure 12 – Movement Switch

The printing of the maze is also based on user input. Every time the user inputs something, the program clears the terminal and prints the new array. Clearing the terminal is the program's way of keeping the screen readable and uncrowded [2].



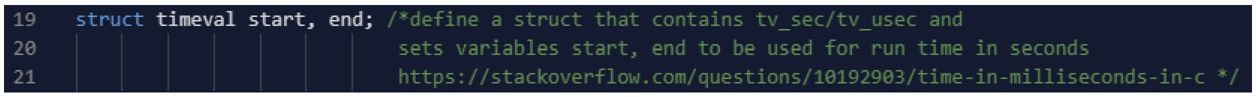
```
        } //end of switch statement
        system("clear");
        printMaze(maze);
```

Figure 13 – Terminal Printing

The while loop (Figure 10, line 45) encompasses the user input and this switch statement, so the program will continue until the player moves onto an "exit space." In programming terms, whenever the exit space of maze[49][50] or maze[5][50] become 'o' due to user movement onto them, the program will end and display the time and movement statistics.

Time and Step Count

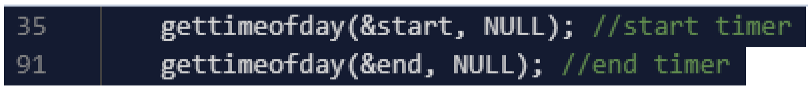
As stated in the introduction, the real-time counter was extremely difficult to figure out and eventually had to be copied from StackOverflow [3], as stated in the program as well.



```c
struct timeval start, end; /*define a struct that contains tv_sec/tv_usec and
                            sets variables start, end to be used for run time in seconds
                            https://stackoverflow.com/questions/10192903/time-in-milliseconds-in-c */
```

Figure 14 – Declaring Structures start and end

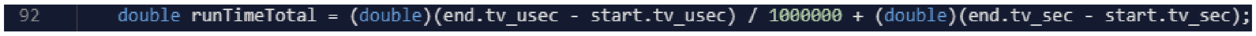
Figure 14 is creating two *timeval* structs, as defined in the <sys/time.h> library [4]. The *timeval* struct has two variables in it, *tv_sec* and *tv_usec*. The former is used for storing full seconds, and the latter is used for storing microseconds, which are one millionth of a second. The *gettimeofday* function is the classic time function that returns milliseconds since the Epoch [5]. This is used and stored into the *timeval* structs of *start* and *end*.



```c
    gettimeofday(&start, NULL); //start timer
    gettimeofday(&end, NULL); //end timer
```

Figure 15 – start and end Storing Milliseconds since Epoch

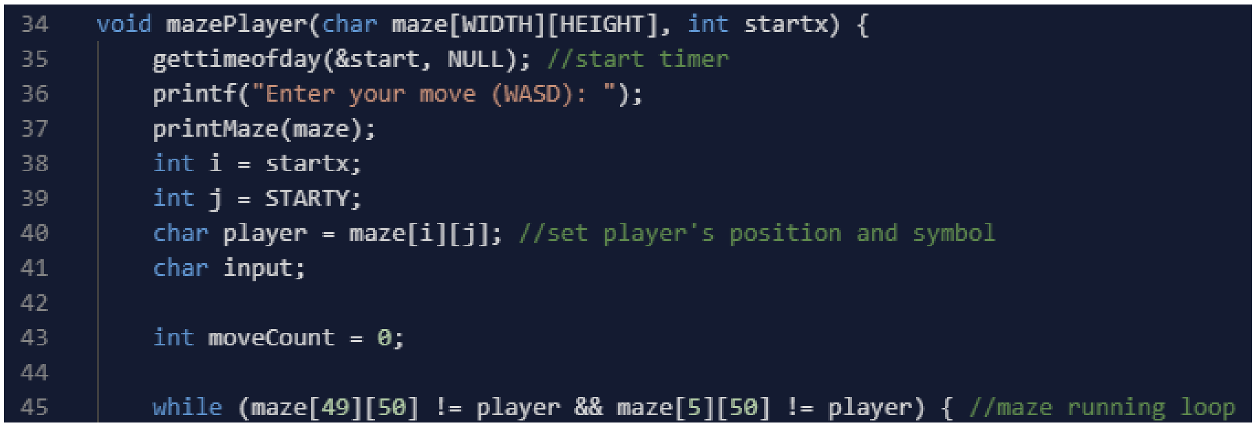
Then the math is done to turn usec into seconds by dividing by one million, and it's added to the time in seconds of the run time of the program. Arguably the usec could be left off at the cost of a small amount of precision for this program.



```c
    double runTimeTotal = (double)(end.tv_usec - start.tv_usec) / 1000000 + (double)(end.tv_sec - start.tv_sec);
```

Figure 16 – Turning start and end Into Seconds Run

For move count, an int type *moveCount* is declared before the start of the maze loop.

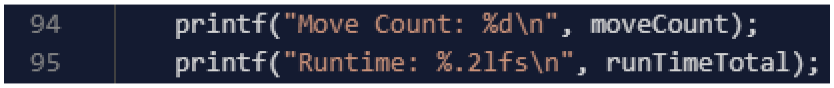


```c
void mazePlayer(char maze[WIDTH][HEIGHT], int startx) {
    gettimeofday(&start, NULL); //start timer
    printf("Enter your move (WASD): ");
    printMaze(maze);
    int i = startx;
    int j = STARTY;
    char player = maze[i][j]; //set player's position and symbol
    char input;

    int moveCount = 0;

    while (maze[49][50] != player && maze[5][50] != player) { //maze running loop
```

Figure 17 – Declaring moveCount at the Beginning of mazePlayer

Referring to Figure 12, lines 56, 64, and 72, and Figure 11, line 80, each time the player moves, it increments *moveCount*. It's set up so if the user inputs a move that isn't allowed (e.g. into a wall, invalid character), then moveCount is not incremented. It only adds when a valid move is made.

Finally, both variables *runTimeTotal* and *moveCount* are printed when the player reaches the end of the maze.



```
    printf("Move Count: %d\n", moveCount);
    printf("Runtime: %.2lfs\n", runTimeTotal);
```

Figure 18 – Print Final Stats After User Completes the Game

Pointers

This section will be kept succinct as the only two instances of pointers are sending the maze's address to the *mazePlayer* function and referencing the array from there.

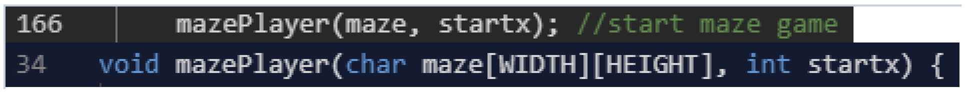

```
        mazePlayer(maze, startx); //start maze game
void mazePlayer(char maze[WIDTH][HEIGHT], int startx) {
```

Figure 19 – Sending the Maze Array from Main to the mazePlayer Function

While this is a simple operation, it is the only thing that allows this program to work. If an array name couldn't be used as a pointer, then it would be much more difficult to send the data to a function.

As a different way of doing things, in the function, the maze array is referenced in moving the player around (Figures 11 & 12). It would be an interesting change (or maybe even improvement) to move the player around by changing the addresses rather than going through the array. For example, instead of doing what is in Figure 11, the program could increment the player's address by one to move the player to the right. The opposite could be done to move the player to the left. In order to move up and down, the player would have to be moved by fifty addresses. This is an idea of using pointers for effectively and cutting down on lines of code.

Conclusion

This project was a challenge and a learning experience. Between figuring out a creative solution to input such a huge maze into the program, getting two entrances and exits to work, creating a functioning, playable character, getting a real-world run time and a move count, and using pointers, this program was very educational. When thrown into the deep end, one either sinks or learns how to swim. Being forged in the fires of adversity are the best way to learn programming, so projects like these are always the best way to learn.

Member

Sean Poston

References

[1] Maze Generator https://www.dcode.fr/maze-generator

[2] Console Clear https://www.geeksforgeeks.org/clear-console-c-language/

[3] Real Time Function https://stackoverflow.com/questions/10192903/time-in-milliseconds-in-c

[4] C <sys/time.h> Library https://pubs.opengroup.org/onlinepubs/007908799/xsh/systime.h.html

[5] C gettimeofday() function
https://pubs.opengroup.org/onlinepubs/009695399/functions/gettimeofday.html

Source Code

Maze Game: https://repl.it/@seanposton4/CS351Project2Maze

```c
/*
Author: Sean Poston
Date: 3/17/2020
Purpose: To create a maze game for a player to solve.
Include things like two exits and entrances, and runtime and movecount.
CS351 PROJECT 2
*/

#include <stdio.h>
#include <sys/time.h> //for timeval timer (lines 19, 36, 92, 93)
#include <ctype.h> //for toupper function (line 49)


#define WIDTH 51
#define HEIGHT 51

#define STARTY 0 //set starting value of y

struct timeval start, end; /*define a struct that contains tv_sec/tv_usec and
                            sets variables start, end to be used for run time in
seconds
                            https://stackoverflow.com/questions/10192903/time-in-
milliseconds-in-c */


void printMaze(char maze[WIDTH][HEIGHT]) { //prints the maze
    for (int i = 0; i < WIDTH; i++) {
        puts("");
        for (int j = 0; j < HEIGHT; j++) {
            printf("%c ", maze[i][j]);
        }
    }
    puts("");
}

void mazePlayer(char maze[WIDTH][HEIGHT], int startx) {
    gettimeofday(&start, NULL); //start timer
    printf("Enter your move (WASD): ");
    printMaze(maze);
    int i = startx;
    int j = STARTY;
    char player = maze[i][j]; //set player's position and symbol
    char input;

    int moveCount = 0;

    while (maze[49][50] != player && maze[5][50] != player) { //maze running loop

        scanf("%1c", &input); //get movement input, limited to one
        input = toupper(input); //set input to uppercase for switch

        switch(input) {
            case 'W': //move up
                if (maze[i - 1][j] != '#') {
                        maze[i - 1][j] = player;
                        maze[i][j] = '.';
                        i--;
                        moveCount++;
                }
                break;
            case 'A': //move left
                if (maze[i][j - 1] != '#') {
                        maze[i][j - 1] = player;
                        maze[i][j] = '.';
                        j--;
                        moveCount++;
                }
                break;
            case 'S': //move down
                if (maze[i + 1][j] != '#') {
                        maze[i + 1][j] = player;
                        maze[i][j] = '.';
                        i++;
                        moveCount++;
                }
                break;
            case 'D': //move right
                if (maze[i][j + 1] != '#') {
                        maze[i][j + 1] = player;
                        maze[i][j] = '.';
                        j++;
                        moveCount++;
                }
                break;
            default:
                break;
        }
        system("clear");
        printMaze(maze);
    }
    system("clear");

    gettimeofday(&end, NULL); //end timer
    double runTimeTotal = (double)(end.tv_usec - start.tv_usec) / 1000000 +
(double)(end.tv_sec - start.tv_sec); //use tv_usec for microseconds and add to
tv_sec for full seconds

    printf("Move Count: %d\n", moveCount);
    printf("Runtime: %.2lfs\n", runTimeTotal);

} //end mazePlayer function

int main(void) {

    char maze[WIDTH][HEIGHT] = {{'#', '#', '#', '#', '#', '#', '#', '#', '#',
'#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#',
'#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#',
'#', '#', '#', '#', '#', '#', '#', '#', '#', '#'},
{'.', '.', '.', '.', '.', '.', '.', '.', '#', '.', '.', '.', '.', '.', '#', '.',
'.', '.', '#', '.', '.', '.', '.', '.', '#', '.', '#', '.', '#', '.', '.', '.',
'#', '.', '.', '.', '.', '.', '.', '.', '.', '.', '#', '.', '#', '.', '.', '.',
'.', '.', '#'},
{'#', '.', '#', '#', '#', '#', '#', '.', '#', '.', '#', '.', '#', '#', '#', '.',
'#', '.', '#', '.', '#', '.', '#', '#', '#', '.', '#', '.', '#', '.', '#', '.',
'#', '#', '#', '.', '#', '#', '#', '#', '#', '#', '#', '.', '#', '.', '#', '#',
'#', '.', '#'},
{'#', '.', '.', '.', '.', '.', '#', '.', '#', '.', '#', '.', '.', '.', '.', '.',
'#', '.', '.', '.', '#', '.', '.', '.', '#', '.', '.', '.', '.', '.', '#', '.',
'.', '.', '#', '.', '.', '.', '#', '.', '.', '.', '.', '.', '.', '.', '#', '.',
'.', '.', '#'},
{'#', '.', '#', '#', '#', '#', '#', '.', '#', '#', '#', '.', '#', '#', '#', '.',
'#', '#', '#', '#', '#', '.', '#', '#', '#', '#', '#', '#', '#', '.', '#', '.',
'#', '#', '#', '#', '#', '.', '#', '#', '#', '.', '#', '.', '#', '#', '#', '.',
'#', '#', '#'},
{'#', '.', '.', '.', '#', '.', '#', '.', '.', '.', '#', '.', '.', '.', '#', '.',
'#', '.', '.', '.', '.', '.', '.', '.', '.', '.', '#', '.', '.', '.', '#', '.',
'.', '.', '.', '.', '#', '.', '.', '.', '.', '.', '#', '.', '#', '.', '.', '.',
'#', '.', '.'},
{'#', '#', '#', '.', '#', '.', '#', '.', '#', '#', '#', '#', '#', '#', '#', '#',
'#', '.', '#', '#', '#', '#', '#', '.', '#', '#', '#', '#', '#', '#', '#', '.',
'#', '#', '#', '#', '#', '#', '#', '#', '#', '.', '#', '#', '#', '.', '#', '#',
'#', '.', '#'},
{'#', '.', '#', '.', '.', '.', '#', '.', '.', '.', '.', '.', '#', '.', '.', '.',
'#', '.', '#', '.', '#', '.', '.', '.', '.', '.', '#', '.', '#', '.', '.', '.',
'.', '.', '.', '.', '#', '.', '#', '.', '#', '.', '.', '.', '#', '.', '.', '.',
'#', '.', '#'},
{'#', '.', '#', '.', '#', '#', '#', '.', '#', '#', '#', '.', '#', '.', '#', '#',
'#', '.', '#', '.', '#', '.', '#', '.', '#', '.', '#', '.', '#', '#', '#', '.',
'#', '#', '#', '#', '#', '.', '#', '.', '#', '.', '#', '#', '#', '#', '#', '#',
'#', '.', '#'},
{'#', '.', '.', '.', '#', '.', '#', '.', '.', '.', '#', '.', '#', '.', '.', '.',
'#', '.', '#', '.', '.', '.', '#', '.', '#', '.', '.', '.', '#', '.', '#', '.',
'.', '.', '.', '.', '.', '.', '.', '.', '#', '.', '#', '.', '.', '.', '.', '.',
'.', '.', '#'},
{'#', '.', '#', '#', '#', '.', '#', '.', '#', '.', '#', '.', '#', '.', '#', '#',
'#', '.', '#', '#', '#', '.', '#', '#', '#', '#', '#', '.', '#', '.', '#', '#',
'#', '#', '#', '.', '#', '#', '#', '#', '#', '.', '#', '.', '#', '.', '#', '#',
'#', '#', '#'},
{'#', '.', '.', '.', '#', '.', '#', '.', '#', '.', '#', '.', '.', '.', '.', '.',
'.', '.', '.', '.', '#', '.', '#', '.', '.', '.', '#', '.', '.', '.', '#', '.',
'#', '.', '.', '.', '#', '.', '#', '.', '.', '.', '#', '.', '#', '.', '#', '.',
'.', '.', '#'},
{'#', '.', '#', '#', '#', '.', '#', '.', '#', '.', '#', '.', '#', '.', '#', '#',
'#', '.', '#', '.', '#', '#', '#', '.', '#', '#', '#', '.', '#', '.', '#', '.',
'#', '#', '#', '.', '#', '.', '#', '.', '#', '.', '#', '.', '#', '#', '#', '#',
'#', '.', '#'},
{'#', '.', '#', '.', '.', '.', '#', '.', '#', '.', '#', '.', '#', '.', '.', '.',
'#', '.', '#', '.', '#', '.', '#', '.', '.', '.', '#', '.', '#', '.', '.', '.',
'#', '.', '.', '.', '.', '.', '.', '.', '#', '.', '#', '.', '.', '.', '#', '.',
'#', '.', '#'},
{'#', '.', '#', '.', '#', '.', '#', '#', '#', '#', '#', '#', '#', '#', '#', '.',
'#', '#', '#', '.', '#', '.', '#', '.', '#', '#', '#', '.', '#', '.', '#', '.',
'#', '.', '#', '#', '#', '.', '#', '#', '#', '#', '#', '#', '#', '.', '#', '.',
'#', '.', '#'},
{'#', '.', '#', '.', '#', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.',
'#', '.', '.', '.', '#', '.', '#', '.', '.', '.', '#', '.', '#', '.', '#', '.',
'#', '.', '#', '.', '#', '.', '.', '.', '.', '.', '#', '.', '.', '.', '.', '.',
'.', '.', '#'},
{'#', '#', '#', '#', '#', '.', '#', '.', '#', '.', '#', '#', '#', '#', '#', '.',
'#', '#', '#', '.', '#', '.', '#', '.', '#', '.', '#', '#', '#', '#', '#', '.',
'#', '.', '#', '.', '#', '#', '#', '.', '#', '.', '#', '.', '#', '#', '#', '#',
'#', '#', '#'},
{'#', '.', '.', '.', '.', '.', '#', '.', '#', '.', '.', '.', '.', '.', '#', '.',
'#', '.', '.', '.', '.', '.', '#', '.', '#', '.', '#', '.', '#', '.', '.', '.',
'.', '.', '.', '.', '#', '.', '#', '.', '#', '.', '#', '.', '#', '.', '.', '.',
'.', '.', '#'},
{'#', '.', '#', '#', '#', '#', '#', '#', '#', '#', '#', '.', '#', '#', '#', '#',
'#', '#', '#', '.', '#', '#', '#', '#', '#', '.', '#', '.', '#', '.', '#', '.',
'#', '#', '#', '#', '#', '.', '#', '.', '#', '.', '#', '.', '#', '#', '#', '.',
'#', '#', '#'},
{'#', '.', '.', '.', '.', '.', '.', '.', '#', '.', '.', '.', '.', '.', '#', '.',
'.', '.', '.', '.', '.', '.', '.', '.', '#', '.', '#', '.', '.', '.', '#', '.',
'.', '.', '#', '.', '.', '.', '#', '.', '#', '.', '.', '.', '.', '.', '.', '.',
'.', '.', '#'},
{'#', '.', '#', '.', '#', '#', '#', '#', '#', '#', '#', '.', '#', '#', '#', '#',
'#', '.', '#', '#', '#', '.', '#', '#', '#', '.', '#', '#', '#', '#', '#', '.',
'#', '.', '#', '.', '#', '#', '#', '#', '#', '.', '#', '.', '#', '#', '#', '#',
'#', '#', '#'},
{'#', '.', '#', '.', '#', '.', '.', '.', '.', '.', '.', '.', '.', '.', '#', '.',
'#', '.', '#', '.', '#', '.', '.', '.', '#', '.', '#', '.', '.', '.', '#', '.',
'#', '.', '.', '.', '#', '.', '#', '.', '.', '.', '#', '.', '.', '.', '#', '.',
'#', '.', '#'},
{'#', '#', '#', '#', '#', '.', '#', '.', '#', '.', '#', '#', '#', '#', '#', '.',
'#', '#', '#', '.', '#', '#', '#', '.', '#', '.', '#', '.', '#', '#', '#', '.',
'#', '.', '#', '#', '#', '.', '#', '#', '#', '.', '#', '.', '#', '#', '#', '.',
'#', '.', '#'},
{'#', '.', '.', '.', '#', '.', '#', '.', '#', '.', '.', '.', '.', '.', '.', '.',
'#', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '#', '.',
'#', '.', '#', '.', '.', '.', '.', '.', '.', '.', '#', '.', '#', '.', '#', '.',
'.', '.', '#'},
{'#', '.', '#', '.', '#', '.', '#', '#', '#', '.', '#', '#', '#', '.', '#', '#',
'#', '#', '#', '.', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#',
'#', '#', '#', '.', '#', '#', '#', '.', '#', '#', '#', '.', '#', '.', '#', '#',
'#', '.', '#'},
{'#', '.', '#', '.', '.', '.', '#', '.', '#', '.', '.', '.', '#', '.', '#', '.',
'.', '.', '.', '.', '#', '.', '.', '.', '#', '.', '.', '.', '#', '.', '.', '.',
'.', '.', '.', '.', '#', '.', '.', '.', '.', '.', '#', '.', '#', '.', '#', '.',
'.', '.', '#'},
{'#', '#', '#', '.', '#', '#', '#', '.', '#', '#', '#', '#', '#', '#', '#', '#',
'#', '.', '#', '.', '#', '.', '#', '#', '#', '#', '#', '.', '#', '.', '#', '.',
'#', '.', '#', '#', '#', '#', '#', '#', '#', '.', '#', '#', '#', '.', '#', '#',
'#', '.', '#'},
{'#', '.', '.', '.', '#', '.', '#', '.', '#', '.', '.', '.', '.', '.', '#', '.',
'.', '.', '#', '.', '.', '.', '#', '.', '.', '.', '.', '.', '.', '.', '#', '.',
'#', '.', '#', '.', '#', '.', '.', '.', '#', '.', '#', '.', '.', '.', '.', '.',
'#', '.', '#'},
{'#', '#', '#', '.', '#', '.', '#', '.', '#', '#', '#', '#', '#', '.', '#', '#',
'#', '.', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '.', '#', '#',
'#', '.', '#', '.', '#', '.', '#', '#', '#', '.', '#', '#', '#', '.', '#', '#',
'#', '.', '#'},
{'#', '.', '.', '.', '.', '.', '.', '.', '.', '.', '#', '.', '.', '.', '#', '.',
'.', '.', '.', '.', '.', '.', '#', '.', '.', '.', '.', '.', '#', '.', '.', '.',
'#', '.', '.', '.', '.', '.', '.', '.', '#', '.', '#', '.', '.', '.', '#', '.',
'#', '.', '#'},
{'#', '#', '#', '#', '#', '#', '#', '.', '#', '.', '#', '.', '#', '.', '#', '#',
'#', '.', '#', '#', '#', '#', '#', '.', '#', '#', '#', '.', '#', '#', '#', '#',
'#', '#', '#', '#', '#', '#', '#', '.', '#', '.', '#', '.', '#', '.', '#', '.',
'#', '.', '#'},
{'.', '.', '.', '.', '#', '.', '.', '.', '#', '.', '#', '.', '#', '.', '.', '.',
'#', '.', '.', '.', '.', '.', '#', '.', '#', '.', '.', '.', '#', '.', '.', '.',
'#', '.', '#', '.', '#', '.', '#', '.', '#', '.', '#', '.', '#', '.', '.', '.',
'.', '.', '#'},
{'#', '.', '#', '#', '#', '.', '#', '#', '#', '#', '#', '#', '#', '.', '#', '.',
'#', '#', '#', '#', '#', '.', '#', '#', '#', '.', '#', '#', '#', '#', '#', '.',
'#', '.', '#', '.', '#', '.', '#', '.', '#', '#', '#', '#', '#', '.', '#', '#',
'#', '#', '#'},
{'#', '.', '.', '.', '#', '.', '#', '.', '.', '.', '#', '.', '#', '.', '#', '.',
'#', '.', '.', '.', '#', '.', '.', '.', '.', '.', '#', '.', '#', '.', '.', '.',
'.', '.', '#', '.', '#', '.', '.', '.', '#', '.', '.', '.', '.', '.', '#', '.',
'.', '.', '#'},
{'#', '#', '#', '.', '#', '#', '#', '.', '#', '#', '#', '.', '#', '#', '#', '.',
'#', '#', '#', '.', '#', '.', '#', '#', '#', '.', '#', '.', '#', '#', '#', '#',
'#', '.', '#', '.', '#', '.', '#', '#', '#', '#', '#', '.', '#', '.', '#', '.',
'#', '#', '#'},
{'#', '.', '#', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.',
'.', '.', '#', '.', '.', '.', '.', '.', '#', '.', '.', '.', '.', '.', '.', '.',
'#', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '#', '.', '.', '.',
'#', '.', '#'},
{'#', '.', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '.', '#', '#',
'#', '#', '#', '.', '#', '#', '#', '.', '#', '#', '#', '#', '#', '#', '#', '#',
'#', '.', '#', '#', '#', '#', '#', '#', '#', '.', '#', '#', '#', '#', '#', '.',
'#', '.', '#'},
{'#', '.', '#', '.', '#', '.', '.', '.', '#', '.', '#', '.', '.', '.', '.', '.',
'.', '.', '#', '.', '#', '.', '.', '.', '#', '.', '#', '.', '#', '.', '.', '.',
'#', '.', '#', '.', '.', '.', '.', '.', '#', '.', '.', '.', '.', '.', '#', '.',
'#', '.', '#'},
{'#', '.', '#', '.', '#', '#', '#', '.', '#', '.', '#', '#', '#', '.', '#', '#',
'#', '.', '#', '#', '#', '.', '#', '.', '#', '.', '#', '.', '#', '.', '#', '#',
'#', '#', '#', '#', '#', '#', '#', '.', '#', '#', '#', '.', '#', '.', '#', '#',
'#', '.', '#'},
{'#', '.', '#', '.', '.', '.', '.', '.', '#', '.', '#', '.', '.', '.', '.', '.',
'#', '.', '.', '.', '.', '.', '#', '.', '.', '.', '#', '.', '.', '.', '.', '.',
'.', '.', '#', '.', '.', '.', '#', '.', '.', '.', '#', '.', '#', '.', '.', '.',
'.', '.', '#'},
{'#', '.', '#', '.', '#', '.', '#', '.', '#', '.', '#', '.', '#', '.', '#', '#',
'#', '.', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#',
'#', '.', '#', '#', '#', '.', '#', '#', '#', '.', '#', '.', '#', '#', '#', '#',
'#', '#', '#'},
{'#', '.', '.', '.', '#', '.', '#', '.', '.', '.', '.', '.', '#', '.', '#', '.',
'#', '.', '.', '.', '#', '.', '#', '.', '#', '.', '.', '.', '.', '.', '.', '.',
'#', '.', '#', '.', '#', '.', '.', '.', '.', '.', '.', '.', '#', '.', '#', '.',
'.', '.', '#'},
{'#', '.', '#', '#', '#', '#', '#', '.', '#', '#', '#', '#', '#', '#', '#', '.',
'#', '#', '#', '.', '#', '.', '#', '.', '#', '#', '#', '#', '#', '#', '#', '.',
'#', '.', '#', '.', '#', '.', '#', '#', '#', '#', '#', '.', '#', '.', '#', '#',
'#', '.', '#'},
{'#', '.', '.', '.', '.', '.', '#', '.', '.', '.', '#', '.', '.', '.', '#', '.',
'.', '.', '.', '.', '.', '.', '.', '.', '#', '.', '.', '.', '#', '.', '#', '.',
'#', '.', '.', '.', '.', '.', '#', '.', '.', '.', '#', '.', '.', '.', '.', '.',
'.', '.', '#'},
{'#', '#', '#', '#', '#', '#', '#', '.', '#', '#', '#', '.', '#', '#', '#', '#',
'#', '#', '#', '#', '#', '#', '#', '.', '#', '.', '#', '.', '#', '.', '#', '.',
'#', '#', '#', '.', '#', '#', '#', '#', '#', '.', '#', '#', '#', '.', '#', '#',
'#', '#', '#'},
{'#', '.', '.', '.', '#', '.', '.', '.', '.', '.', '.', '.', '#', '.', '.', '.',
'#', '.', '.', '.', '.', '.', '#', '.', '.', '.', '#', '.', '.', '.', '#', '.',
'#', '.', '#', '.', '.', '.', '#', '.', '.', '.', '#', '.', '.', '.', '.', '.',
'#', '.', '#'},
{'#', '.', '#', '.', '#', '#', '#', '.', '#', '#', '#', '#', '#', '#', '#', '.',
'#', '#', '#', '.', '#', '.', '#', '.', '#', '#', '#', '#', '#', '#', '#', '.',
'#', '.', '#', '.', '#', '#', '#', '.', '#', '#', '#', '.', '#', '#', '#', '#',
'#', '.', '#'},
{'#', '.', '#', '.', '#', '.', '.', '.', '#', '.', '.', '.', '#', '.', '.', '.',
'#', '.', '#', '.', '#', '.', '.', '.', '.', '.', '.', '.', '.', '.', '#', '.',
'.', '.', '#', '.', '#', '.', '.', '.', '#', '.', '.', '.', '.', '.', '#', '.',
'#', '.', '#'},
{'#', '.', '#', '#', '#', '.', '#', '#', '#', '.', '#', '.', '#', '#', '#', '.',
'#', '.', '#', '.', '#', '.', '#', '.', '#', '.', '#', '#', '#', '.', '#', '.',
'#', '.', '#', '#', '#', '#', '#', '.', '#', '.', '#', '.', '#', '.', '#', '.',
'#', '.', '#'},
{'#', '.', '.', '.', '.', '.', '.', '.', '.', '.', '#', '.', '.', '.', '.', '.',
'.', '.', '#', '.', '#', '.', '#', '.', '#', '.', '#', '.', '.', '.', '.', '.',
'#', '.', '.', '.', '#', '.', '.', '.', '.', '.', '#', '.', '#', '.', '.', '.',
'.', '.', '.'},
{'#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#',
'#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#',
'#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#',
'#', '#', '#'}
};
```

```c
    int startx = 0;
    while (startx != 1 && startx != 2) {
        printf("%s", "Enter starting position (1 or 2): ");
        scanf("%d", &startx);
    }

    if (startx == 2)
        startx = 31;

    system("clear");
    maze[startx][STARTY] = 'o'; //set player symbol and position

    mazePlayer(maze, startx);

    return 0;
}
```

Maze Array Generator (in Java): https://repl.it/@seanposton4/Maze-Array-Generator

```java
import java.io.*;
import java.util.*;

public class Main {
    public static void main(String[] args){
        try {
            File fileIn = new File("mazechars.txt");
            Scanner reader = new Scanner(fileIn);
            String buffer;
            File fileOut = new File("maze.txt");

            if (!fileOut.exists())
                fileOut.createNewFile();

            FileWriter writer = new FileWriter(fileOut);
            while (reader.hasNext()) {
                buffer = reader.nextLine();
                char x;
                for (int i = 0; i < buffer.length(); i++) {
                    if (i == 0)
                        writer.write("{");

                    x = buffer.charAt(i);
                    //;)
                    if (i != buffer.length() - 1)
                        writer.write("'" + x + "', ");
                    else {
                        writer.write("'" + x + "'},\n");
                    }
                }
            }
            writer.close();
            reader.close();
        }
        catch (IOException e) { System.out.println(e);}
    }
}
```