Tongji University

Data Structure Course Curriculum Design Document

Task3: Brave Maze Game

Author:
嘉杰 李
Jiajie Li

Supervisor:
颖 张
Prof. Ying Zhang

A companion use documentation for a program submitted in fulfillment of the requirements for the Data Structure curriculum design

in the

School of Software Engineering, Tongji

December 27, 2019

"Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism."

Dave Barry

Contents

Chapter 1

Analysis

1.1 Background Analysis

The solution of the maze-type pathfinding problem has similar applications in many real-world problems. Usually, we use the backtracking method to solve this type of problem. The background of the ontology is that the maze has only two doors, one is called the entrance and the other is called the exit. A knight rides into the maze from the entrance. The maze sets many obstacles. The knight needs to find a way in the maze to reach the exit.

1.2 Functional Analysis

In this problem, mazes are arranged at corresponding positions in sequence, where "#" represents a wall and "0" represents an open space. The player starts from (1, 1) and finds a feasible path to (5, 5). After arranging your own maze, the program runs and redisplays the maze. The feasible path is marked with "x", and a feasible path is displayed below the maze.

Chapter 2

Design

2.1 Data structure design

As described in the functional analysis above, the backtracking method is selected to solve this maze problem. This method enumerates and tests the candidate solutions of the problem one by one in a certain order. When it is found that the current candidate solution cannot be the solution, it is discarded and the next candidate solution is selected. If all the other requirements have been met except that the problem size requirements have not been met, then the current candidate solution size will be expanded to continue the trial. If the current candidate solution meets all requirements including the problem size, this candidate solution becomes a solution to the problem. In this course design, I used a recursive method for heuristics in the backtracking method.

For this purpose, a two-dimensional array of type char maze [m] [p] is used to represent the maze. When maze [i] [j] == '0', it means that the position is a path and can pass, and in other cases it is Obstacles cannot pass.

In order to prevent re-traveling, another mark matrix m [m] [p] is set, all elements of which are initialized to 0. Once walking to a certain position of the maze (i, j), mark[i][j] is set to 1. The next time this position can not be left.

2.2 Design of Direction Table

Because the knight can walk in four directions: up, down, left, and right, a table of four directions is defined as follows:

```
    int dir[4][2]={{1,0},{0,1},{-1,0},{0,-1}};
}
```

2.3 System design

The system first requires the user to manually enter the initial situation of the maze. "#" Represents the wall and "0" represents the open space. Then, based on

the user's input, the default starting position of the maze is (1, 1) and the exit is (5, 5). A feasible path is generated and output. The output map is marked with a feasible path with "x".

Chapter 3

Implementation

3.1 Recursive Maze Function

3.1.1 Recursive Maze core code

```
    int dfs(int x, int y){
    if (x==1 && y==1) {
        p_tag=1;
        cout << endl << "迷宫地图: " << endl << endl << "    ";
        for (int i=0;i<n;++i) cout << "c"<< i << "  ";
        cout << endl << endl;
        for (int i=0;i<n;++i){
            cout << "r" << i << "   ";
            for (int j=0;j<m;++j)
                if (tag[i][j]) cout << t[i][j] << "   ";
                else cout<<"x   ";
            cout << endl << endl;
        }
        cout << endl << "迷宫路径: " << endl << "(1,1)";
        return 0;
    }

    for (int i=0;i<4;++i)
        if (tag[x+dir[i][0]][y+dir[i][1]] && t[x+dir[i][0]][y+dir[i][1]]=='0') {
            tag[x+dir[i][0]][y+dir[i][1]]=0;
            dfs(x+dir[i][0],y+dir[i][1]);
            tag[x+dir[i][0]][y+dir[i][1]]=1;
            if (p_tag) {
                cout << " ---> (" << x << "," << y << ")";
                return 1;
            }
    }
    return 0;
}
```

3.1.2 Recursive Maze flowchart

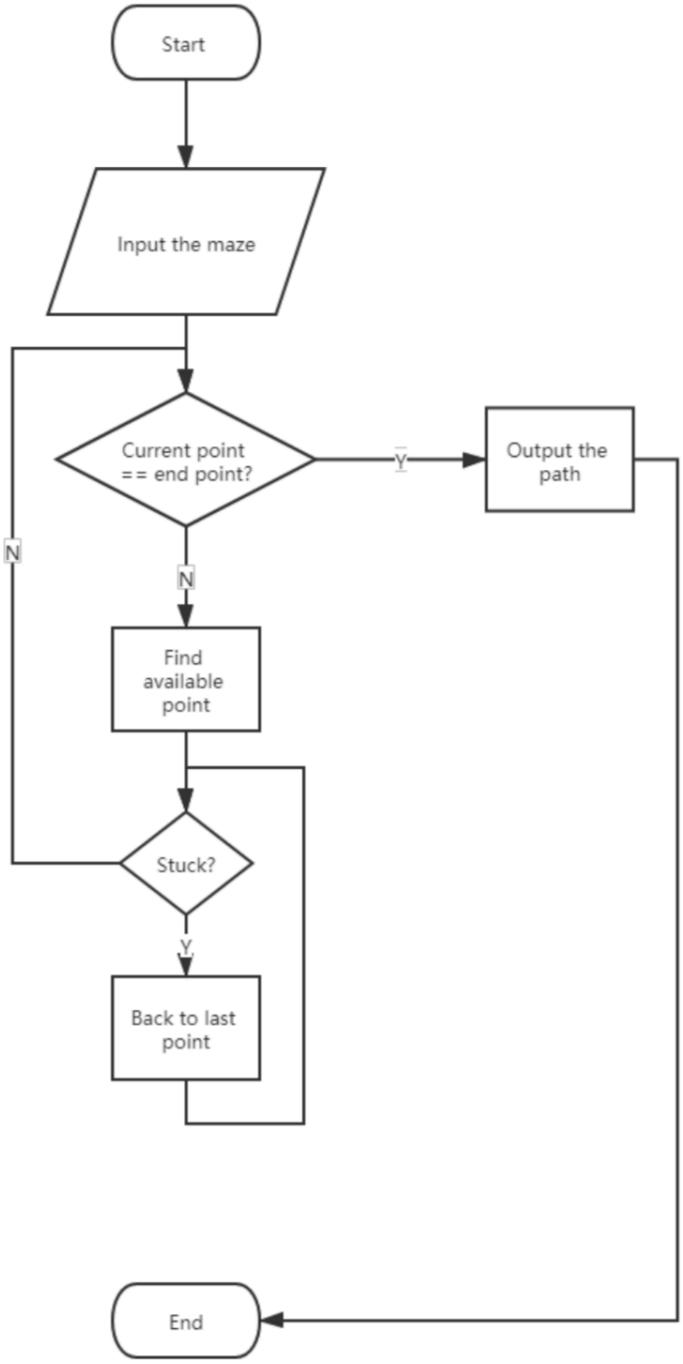

Figure 3.1: Recursive Maze flowchart

Chapter 4

Test

4.1 Function Tests

4.1.1 Seek Solution

Test Case: None

Expected Result:

	c0	c1	c2	c3	c4	c5	c6
r0	#	#	#	#	#	#	#
r1	#	x	#	0	0	0	#
r2	#	x	#	0	#	#	#
r3	#	x	x	x	#	0	#
r4	#	0	#	x	x	x	#
r5	#	0	#	0	#	x	#
r6	#	#	#	#	#	#	#

(1,1) —> (2,1) —> (3,1) —> (3,2) —> (3,3) —> (4,3) —> (4,4) —> (4,5) —> (5,5)

Program Result:

Figure 4.1: Screenshots

Appendix A

Frequently Asked Questions

A.1 Why this document looks so sparse?

This document is built using LaTeX and is arranged in book format. There may be blank pages before and after each chapter.

A.2 Why is there header files in this project?

I wrote basic stl header files, such as vector.h, etc. Except for the header files I wrote, the only header files used in all projects are iostream and string.h