项目说明文档

数据结构课程设计

——勇闯迷宫游戏

作 者 姓 名： 伊啸

学 号： 1551534

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

目 录

[1 分析 1](#_Toc59978701)

[1.1 功能分析 1](#_Toc59978702)

[2 设计 1](#_Toc59978703)

[2.1 数据结构设计 1](#_Toc59978704)

[2.2 系统设计 2](#_Toc59978705)

[3 实现 2](#_Toc59978706)

[3.1 初始化功能的实现 2](#_Toc59978707)

[3.1.1 初始化功能核心代码 2](#_Toc59978708)

[3.1.2 初始化功能截屏示例 3](#_Toc59978709)

[3.2 搜索通路功能的实现 3](#_Toc59978710)

[3.2.1 搜索通路功能流程图 3](#_Toc59978711)

[3.2.2 搜索路径功能核心代码 5](#_Toc59978712)

[3.3 总体系统的实现 7](#_Toc59978713)

[3.3.1 总体系统流程图 7](#_Toc59978714)

[3.3.2 总体系统核心代码 9](#_Toc59978715)

[3.3.3 总体系统截屏示例 9](#_Toc59978717)

[4 测试 11](#_Toc59978718)

[4.1 功能测试 11](#_Toc59978719)

[4.2 出错测试 12](#_Toc59978720)

[4.2.1 size为不合理的整数 12](#_Toc59978721)

[4.2.2 size不为整数 12](#_Toc59978722)

# 1 分析

## 1.1 功能分析

迷宫只有两个门，一个门叫入口，另一个门叫出口。一个骑士骑马从入口进入迷宫，迷宫设置很多障碍，骑士需要在迷宫中寻找通路以到达出口。迷宫问题的求解过程可以采用回溯法即在一定的约束条件下试探地搜索前进，若前进中受阻，则及时回头纠正错误另择通路继续搜索的方法。从入口出发，按某一方向向前探索，若能走通，即某处可达，则到达新点，否则探索下一个方向；若所有的方向均没有通路，则沿原路返回前一点，换下一个方向再继续试探，直到所有可能的道路都探索到，或找到一条通路，或无路可走又返回入口点。在求解过程中，为了保证在达到某一个点后不能向前继续行走时，能正确返回前一个以便从下一个方向向前试探，则需要在试探过程中保存所能够达到的每个点的下标以及该点前进的方向，当找到出口时试探过程就结束了。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，该系统要求使用回溯法，要方便取任意一个迷宫点的坐标，因此自然想到用数组存储迷宫。因为该项目并不大，只要求使用回溯法从入口到出口，通过递归即可实现，因此我并未设计类结构。

设计了一个\_move[4]数组，表示移动方向

struct offsets {

int a;

int b;

};

offsets \_move[4] = { {-1,0} ,{0, -1},{1,0}, {0,1} };

//在上下左右四个方向的移动

## 2.2 系统设计

系统首先实现对屏幕的初始化，在屏幕上输出迷宫游戏的规则，简单来说，迷宫为正方形，用1表示障碍物，0表示可通行，最外面一圈是围栏，坐标（1,0）为入口，相应的右下角为出口。由用户输入迷宫的size（4-13），太小的话输出没什么意义，太大的话输出长度太长。迷宫是随机生成的，除去最外一层围栏，障碍物和通路的数量大概是1:2，即0的数量大概是1数量的两倍，这样是通行的可能性大一些，但又不至于障碍物太少。输入size后随机生成迷宫，再自动判断是否有通路从入口到出口，有就将通路路径输出，没有则输出迷宫中无通路。

# 3 实现

## 3.1 初始化功能的实现

### 3.1.1 初始化功能核心代码

srand((unsigned int)time(NULL));

for (int i = 0; i < maxsize; i++) {

for (int j = 0; j < maxsize; j++) {

//实现通路与障碍物之比为2:1

int target = rand() % 3;

if (target == 0)

Maze[i][j] = 1;

else

Maze[i][j] = 0;

}

}

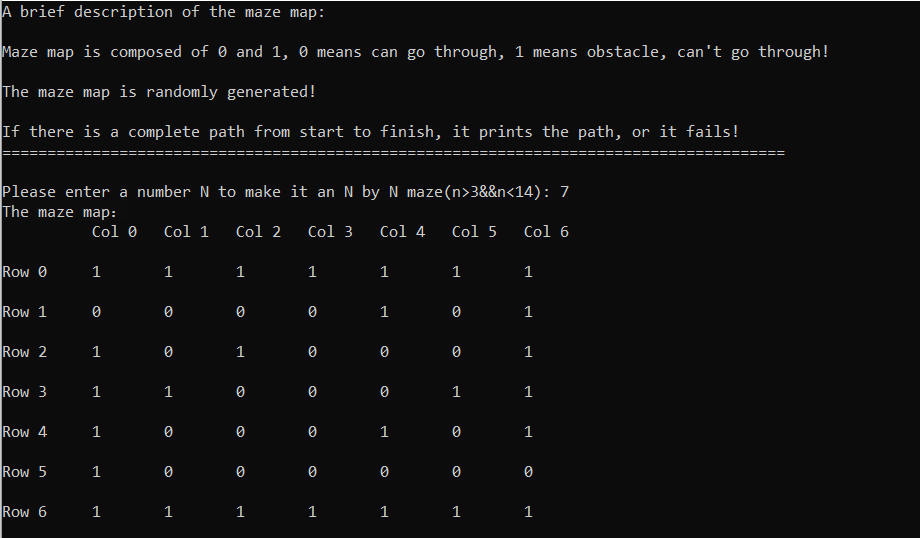
Maze[1][0] = Maze[1][1] = 0;

//避免在入口处就被障碍物封住

Maze[maxsize - 2][maxsize - 2] = Maze[maxsize - 2][maxsize - 1] = 0;

//避免在出口处被障碍物封住

### 3.1.2 初始化功能截屏示例



## 3.2 搜索通路功能的实现

### 3.2.1 搜索通路功能流程图

因为要正向输出路径而且用递归，因此从出口开始搜索，直到找到入口



### 3.2.2 搜索路径功能核心代码

int g, h;

if (x == 1 && y == 0) //到达入口

return 1;

for (int i = 0; i < 4; i++) {

//在四个方向上循环

g = x + \_move[i].a;

h = y + \_move[i].b;

if (maze[g][h] == 0 && mark[g][h] == 0) {

mark[g][h] = 1; //标记走过的路径

if (Path(size, maze, mark, g, h)) {

cout << "<" << g << "," << h << "> ---> ";

return 1;

}

}

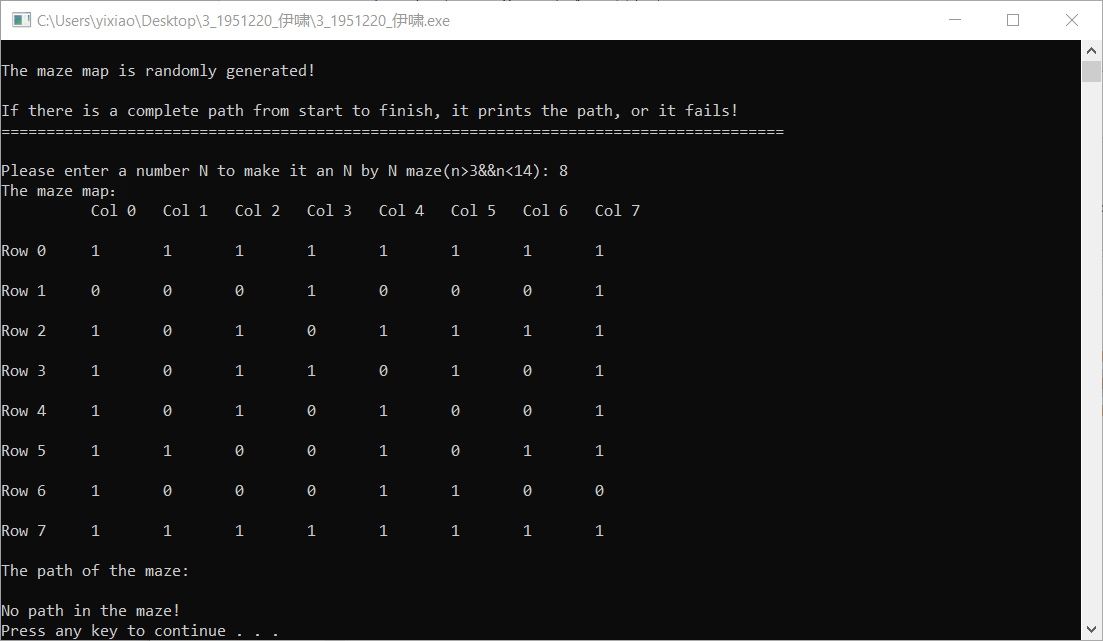
}

if (x == size-2 && y == size-2) //如果回到出口

cout << "No path in the maze!" << endl;

return 0;

3.2.3 搜索路径功能截屏示例



## 3.3 总体系统的实现

### 3.3.1 总体系统流程图



### 3.3.2 总体系统核心代码

cout << "Please enter a number N to make it an N by N maze(n>3&&n<14): ";

while (1) {

//判断输入的是否为正整数

cin >> maxsize;

if (cin.good() == false) {

while (getchar() != '\n')

;

cin.clear();

cout << "The data you entered is not a positive integer. Please re-enter(n>3&&n<14): ";

}

else

break;

### }

Initial(maxsize, Maze);

cout << "The path of the maze:\n" << endl;

Path(maxsize, Maze, mark, maxsize - 2, maxsize - 1);

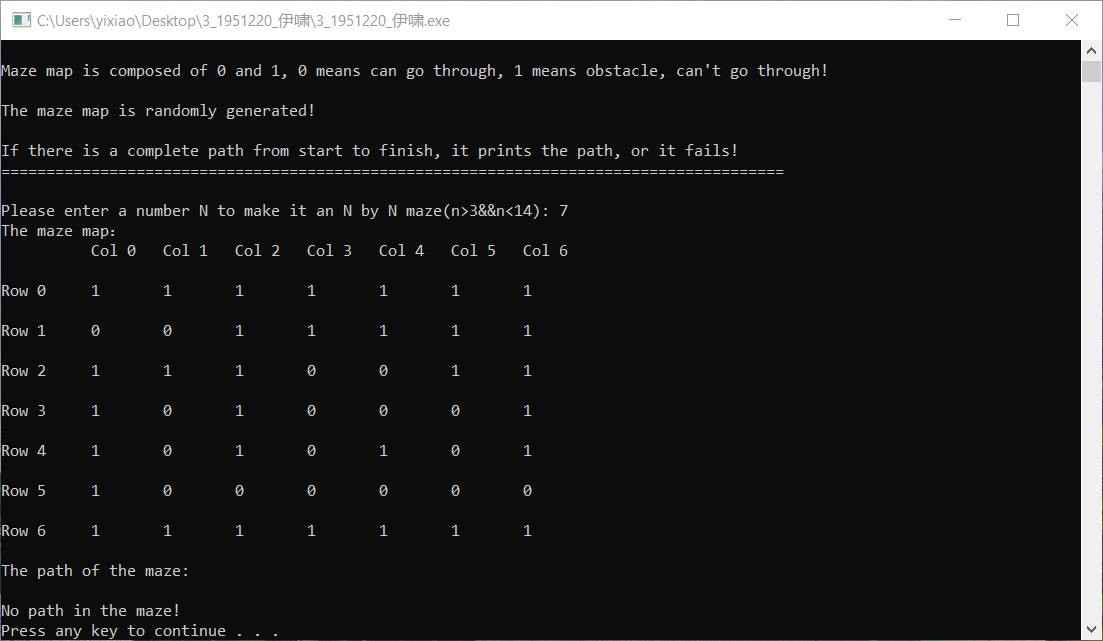
// 对于正向输出路径，从末尾开始，一直到开始

if(mark[1][1]) //如果迷宫能走通，输出出口位置

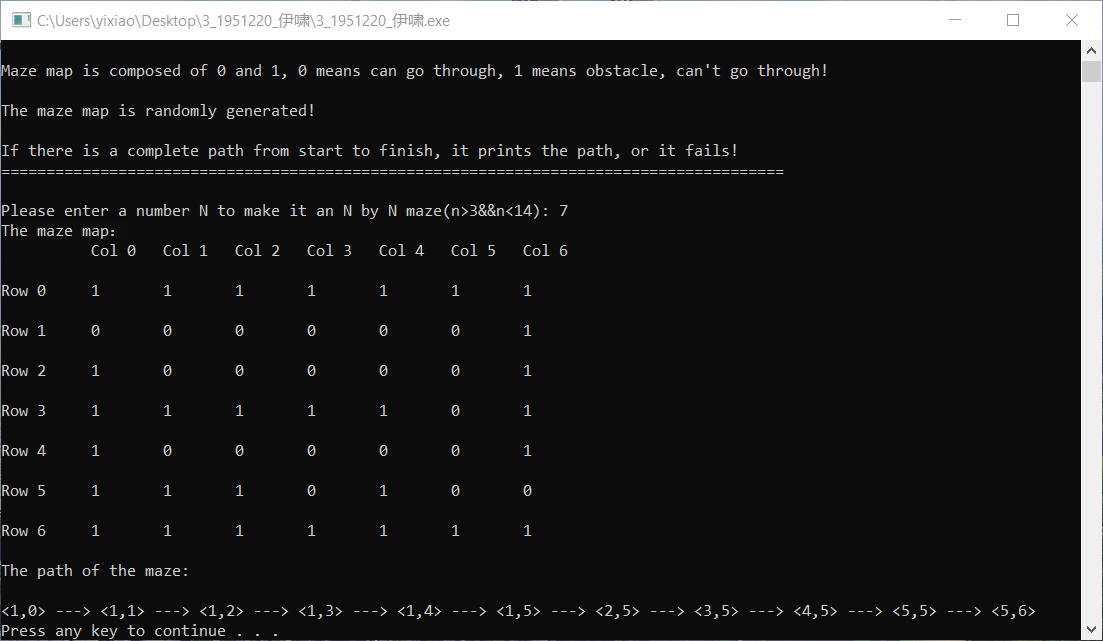
cout << "<" << maxsize - 2 << "," << maxsize - 1 << ">" << endl;

### 3.3.3 总体系统截屏示例

无通路：



有通路：



# 4 测试

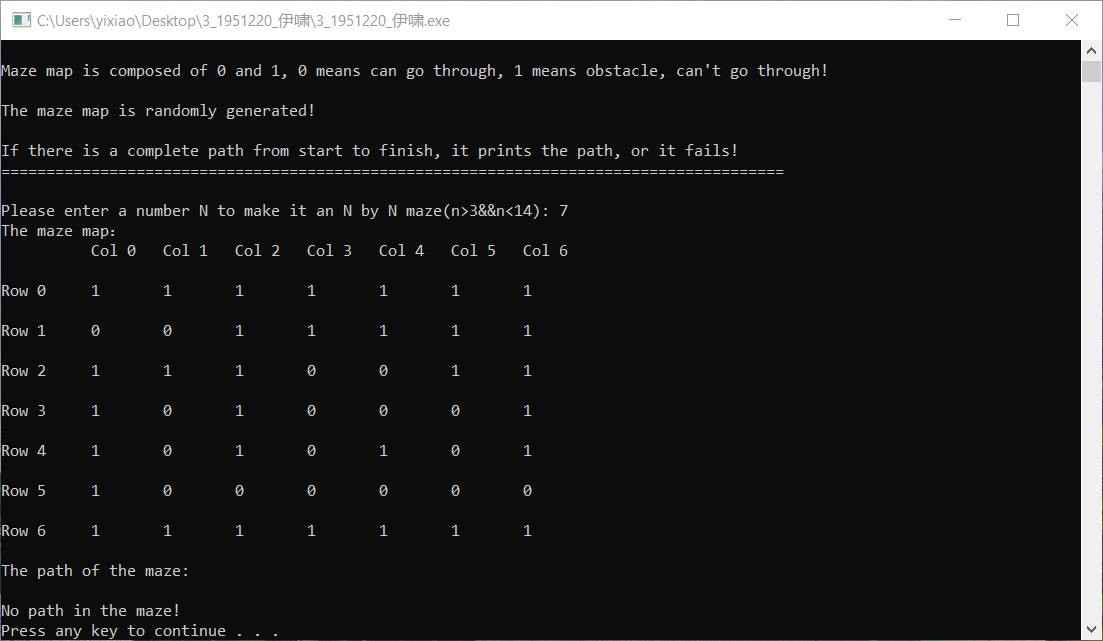
## 4.1 功能测试

**测试用例：**输入迷宫大小为7

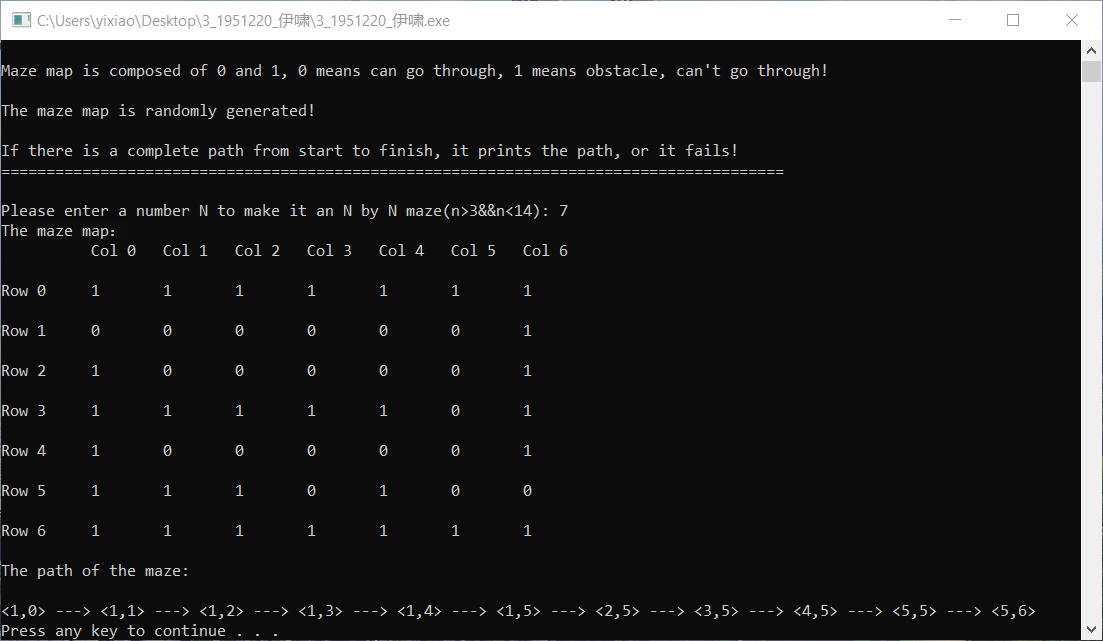
**预期结果：**可能存在通路，也可能没有

**实验结果：**

无通路：



有通路：

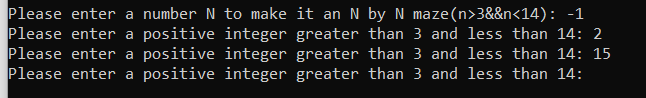


## 4.2 出错测试

### 4.2.1 size为不合理的整数

**测试用例：输入负数或不合理的正整数**

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

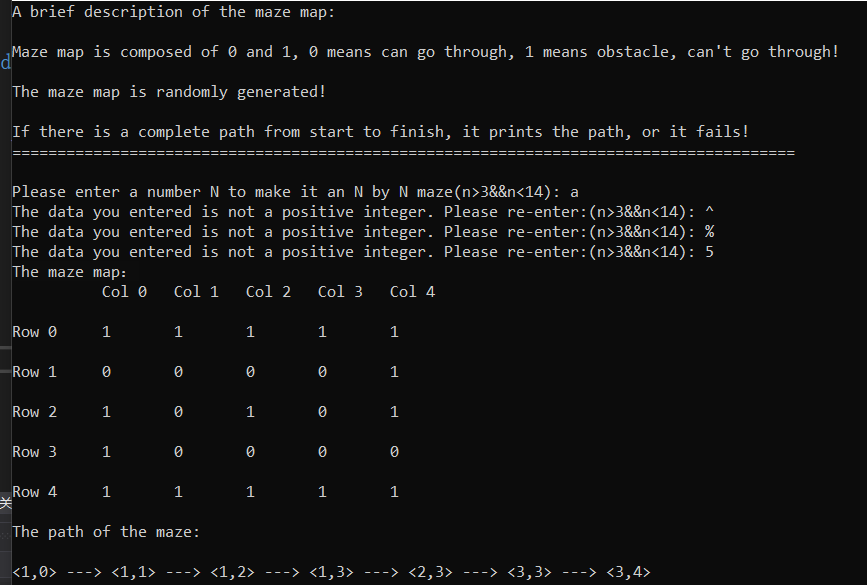
**实验结果：**

### 4.2.2 size不为整数

**测试用例：**输入其他字符

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**

****