项目说明文档

数据结构课程设计

——勇闯迷宫游戏

作 者 姓 名： 崔鑫宇

学 号： 1853444

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

# 1 分析

## 1.1 背景分析

一般的迷宫只有两个门，一个门叫入口，另一个门叫出口。现有一个骑士骑着马从入口进入迷宫，而迷宫中有很多障碍（其中的‘#’表示墙壁，‘\*’表示道路），骑士需要在迷宫中寻找通路以到达出口。

## 1.2 功能分析

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。综上所述，一个考试报名系统至少应该具有输入、输出、插入、删除、修改、退出的功能。

综上所述，本程序需要保存一条从入口直通至出口的道路的坐标即该点前进的方向（如果存在），如果一条路无法继续往前行进，本程序还应可以返回上一个可行的路口，以继续向下一路口探索。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，该系统要求大量的存储与删除数据，并且在程序结束的时候，还可以将之前保存的数据全部输出，由此可以选择使用栈——这一数据结构来完成本程序的设计。

## 2.2 类结构设计

本程序中我们需要使用栈，这一数据结构我们的栈应该具有推入元素、删除尾元素以及输出整个数组的功能。

## 2.3 成员与操作设计

**栈类（MyStack）**

public:

int size();//栈的大小

MyStack() = default;

~MyStack() = default;

T top();//返回栈顶元素

void push(T);//向栈尾打入元素

void pop();//弹出栈尾元素

private:

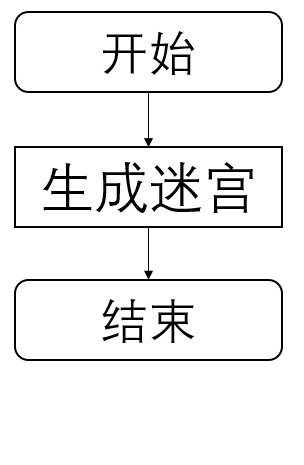
container StackBuilt;//栈的存储容器

## 2.4 系统设计

系统首先调用BuildMaze()函数实现生成一个随机迷宫地图并打印出来，然后进行寻路处理，如果找到路径将路径打印出来，否则提示本迷宫不存在通路，最后输出寻路操作过后的迷宫地图（如果存在通路，将路径用‘0’表示出来）。

# 3 实现

### 3.1 总体系统流程图



### 3.2 总体系统核心代码

①迷宫生成部分

char\*\* Maze = new char\* [Size];

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

Maze[i] = new char[Size];

}

srand((unsigned)time(NULL));

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

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

if ((i == 0) || (i == Size - 1) || (j == 0) || (j == Size - 1)) {

if (i == 1 && j == 0) {//用‘B’表示迷宫起点

Maze[i][j] = 'B';

}

else if (i == Size - 2 && j == Size - 1) {//用‘E’表示迷宫终点

Maze[i][j] = 'E';

}

else

Maze[i][j] = BuiltTool[0];

}

else if ((i == 1 && j == 0) || (i == 1 && j == 1) || (i == Size - 2 && j == Size - 1) || (i == Size - 2 && j == Size - 2)) {

Maze[i][j] = BuiltTool[9];

}

else

Maze[i][j] = BuiltTool[rand() % 10];

}

}

return Maze;

②寻路部分

MyStack<point> s;

point p(1, 1, 2);

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

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

Maze[i][j] = BuiltTool[9];

}

}

Maze[0][1] = 1;

Maze[1][1] = 1;

s.push(p);

int i = 0;

while (s.size()!=0)

{

while (i < 8)

{

int x = p.x + dir(i).x;

int y = p.y + dir(i).y;

if (maze[x][y] == '\*' && Maze[x][y] == '\*') {

move(p, i);

s.push(p);

if (x == Size - 2 && y == Size - 2) {

return s;

}

Maze[x][y] = '#';

i = 0;

continue;

}

++i;

}

s.pop();

if (s.size()==0) {

cerr << "该迷宫入口与出口间没有通路！";

}

else{

p = point(s.top().x, s.top().y, s.top().dir);

i = 0;

}

}

③综合处理部分

char \*\*maze = BuildMaze();

cout << "迷宫地图：" << endl;

MazePrint(maze);//打印随机生成的迷宫的初始状态

cout << endl;

MyStack<point> s = Search(maze);

MyStack<point> out;

cout << endl;

while (s.size() != 0) {

point p = s.top();

out.push(p);

s.pop();

}

while (out.size() != 0) {

point q = out.top();

out.pop();

if (out.size() == 0) {

cout << "(" << q.x << "," << q.y << ")" << endl;

maze[q.x][q.y] = '0';

}

else {

cout << "(" << q.x << "," << q.y << ") ---> ";

maze[q.x][q.y] = '0';

}

}

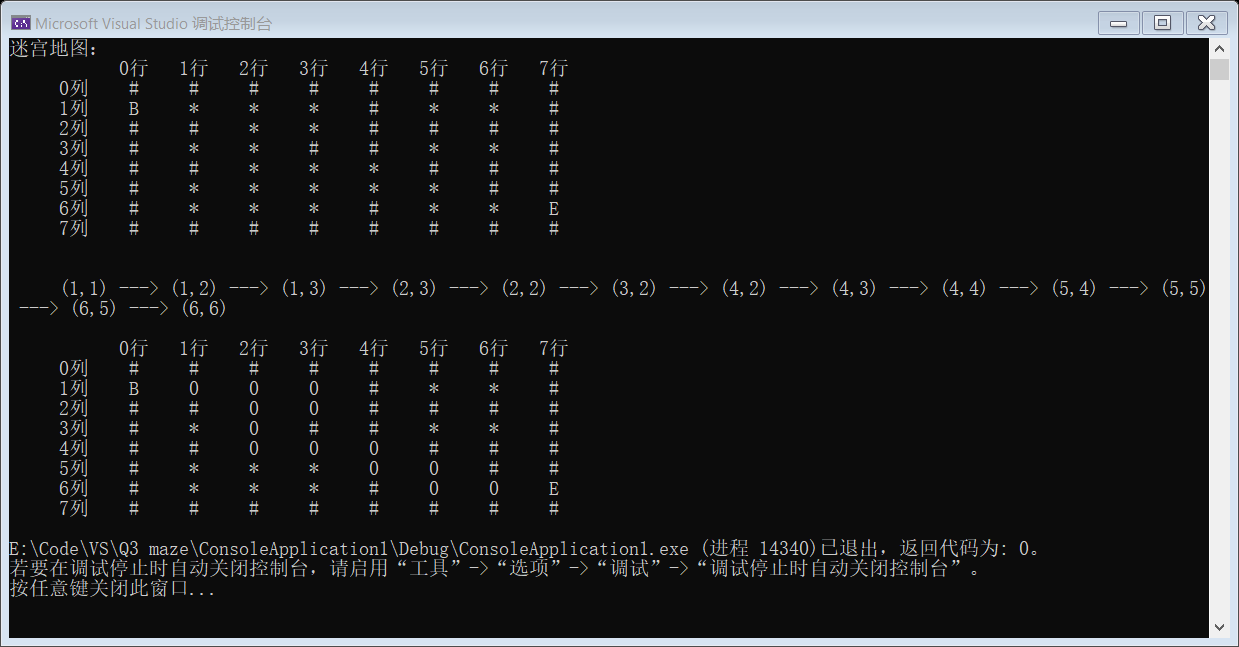
cout << endl;

MazePrint(maze);

//打印描绘出通路的（如存在通路）迷宫的最终状态

return 0;

### 3.3 总体系统截屏示例

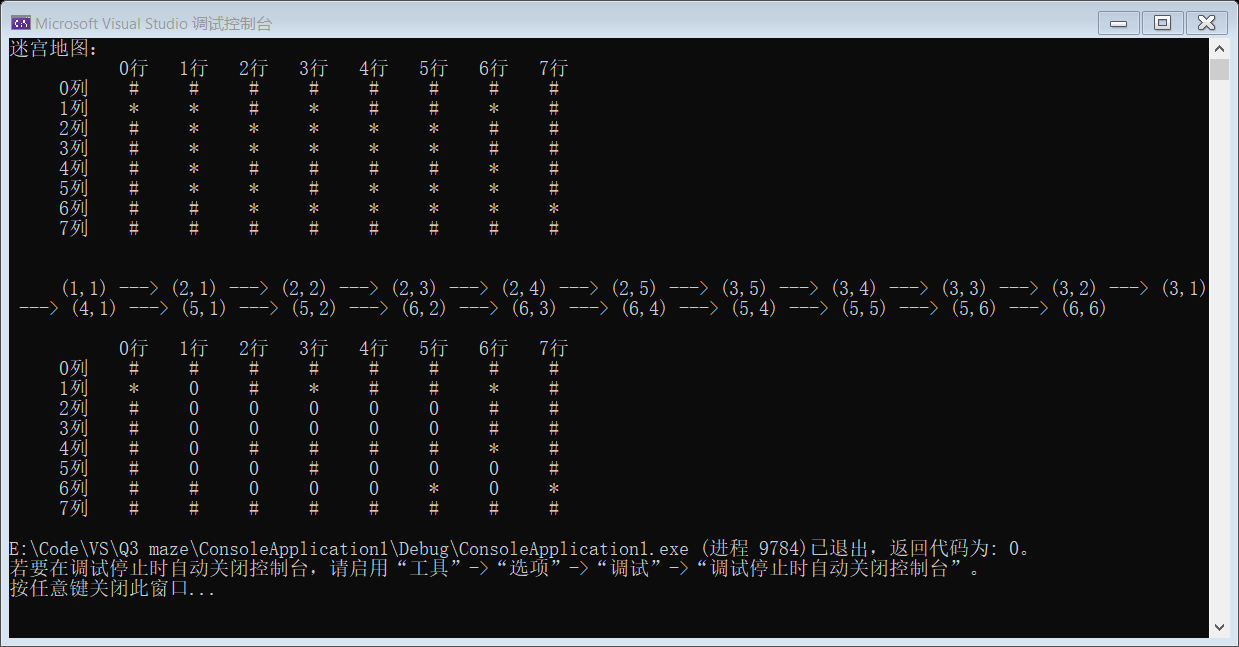


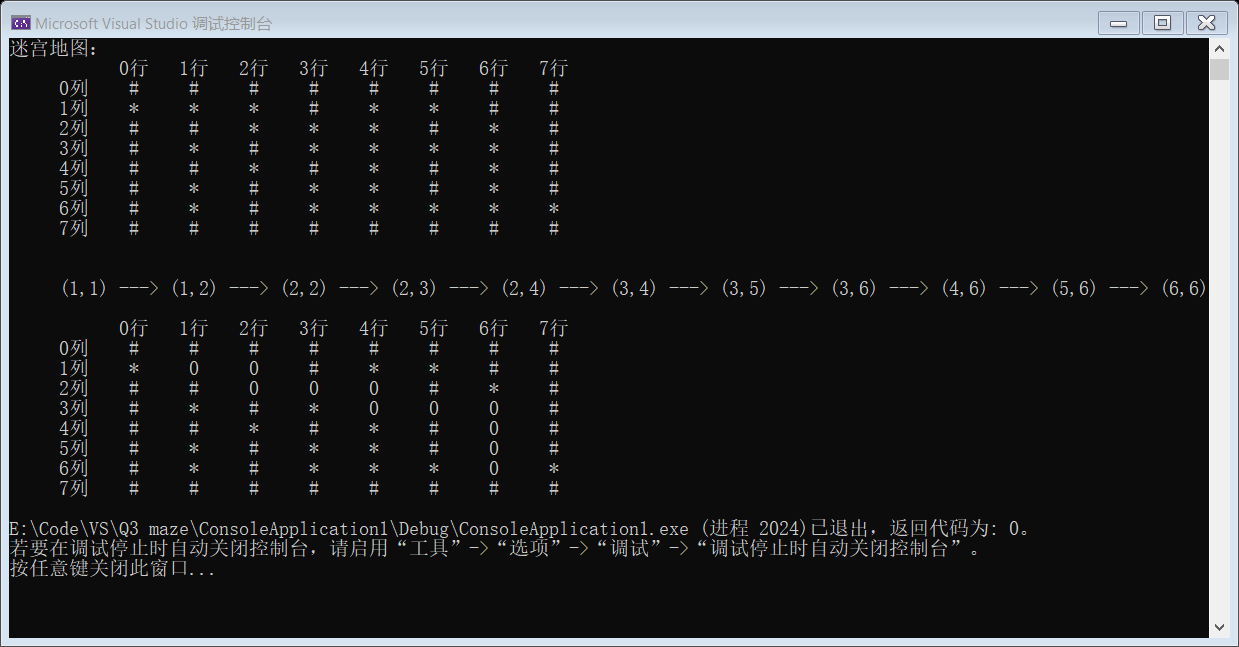
# 4 测试

## 4.1 功能测试

4.1.1 迷宫存在通路

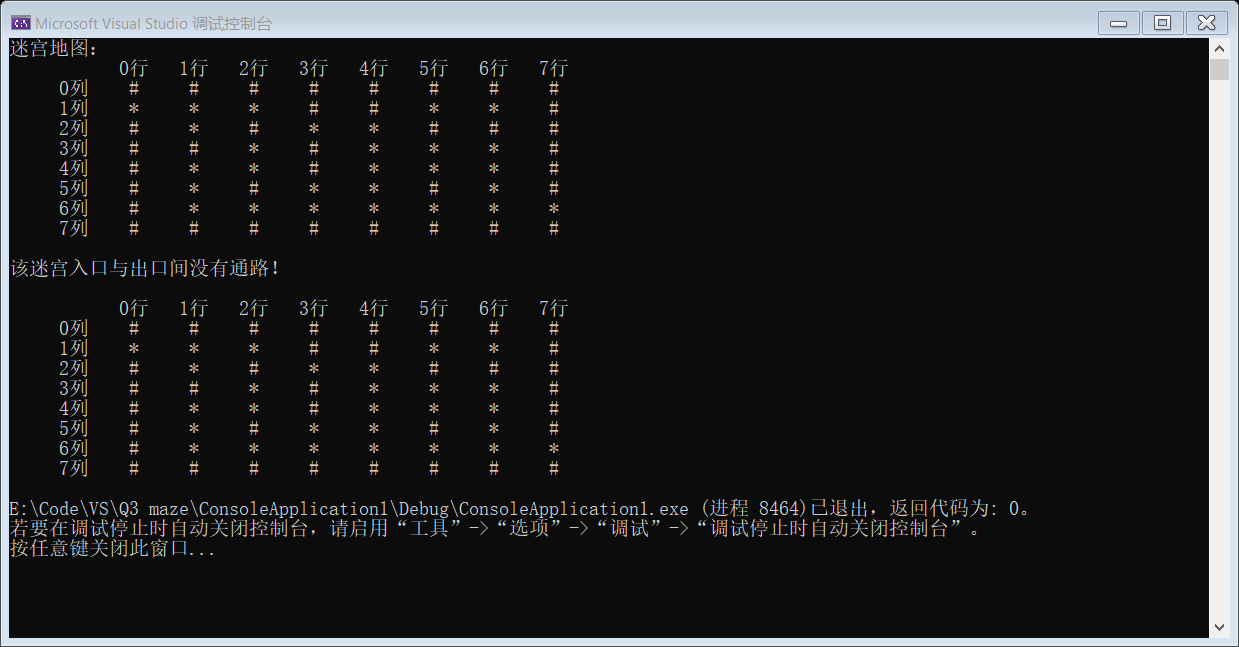
测试结果：

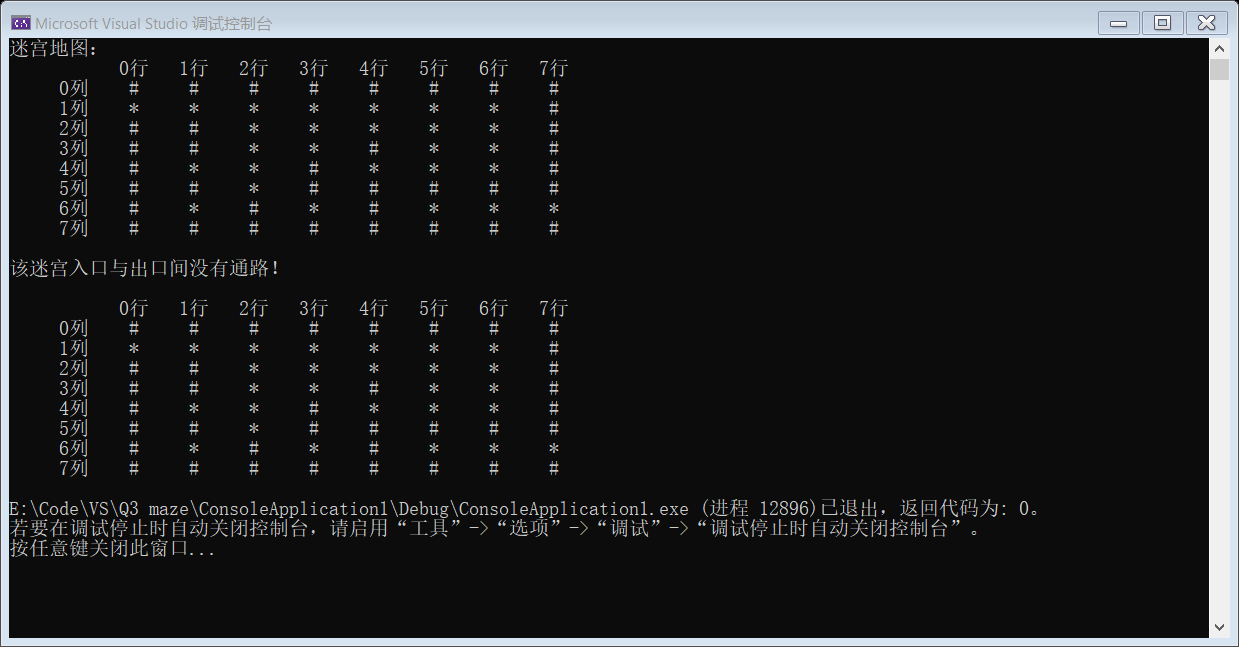




4.1.2 迷宫不存在通路

测试结果：





# 