项目说明文档

数据结构课程设计

——勇闯迷宫游戏

作 者 姓 名： 何慧琳

学 号： 2152343

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

迷宫游戏是我们最常见的游戏之一，该游戏出现在手机游戏、掌上游戏机和电脑游戏当中，因此迷宫游戏也是一个十分经典的游戏。一般的迷宫游戏比较简单，游戏中随机出现一些障碍，要求玩家能够找出一条从起点到终点的通路。它上手简单，但要掌握其中的奥秘且快速找出一条通路难度却不低。

## 1.2 功能分析

作为一个迷宫游戏，它能够读取地图、打印地图，低要求的迷宫游戏只要求找到通路，但本程序能够查找从入口到出口的最短路径并输出。该版本是一个破解版的勇闯迷宫游戏，程序内部已给出三份地图，在exe中按回车键即可看到原地图、破解后含路径标记的地图以及最短路径的坐标表示。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，该系统要求能够存储、遍历地图，所以定义了一个二维数组类MyMatrix，不同坐标设不同值以标识是障碍物还是可行路；为了寻找通路且是最短通路，采用广度优先搜索，涉及逐层处理，所以定义了一个队列Queue；此外，还有地图的输入和打印工作，于是构建了一个地图类Map，它能够实现迷宫游戏的完整功能。

## 2.2 类结构设计

**二维数组类（MyMatrix）**

描述一个矩阵的参数有行数、列数以及各位置上的元素值，所以它里面包含三个私有成员：行数（int rowSize）、列数（int colSize）、一个二维指针——数组元素存放空间（Type\*\* elements）。既然是二维数组类，为了方便使用，对[]进行了运算符重载从而能直接用[]取相应位置的元素值。

**私有成员：**

Type\*\* elements; //数组元素存放空间

int colSize; //当前数组列的大小

int rowSize; //当前数组宽的大小

**公有操作：**

MyMatrix() :elements(NULL),colSize(INT\_MAX),rowSize(INT\_MAX) {};//无参构造函数

MyMatrix(int rowsz,int colsz,Type \*\*elmts=NULL);//构造函数

MyMatrix(const MyMatrix<Type>& Matrix);//复制构造函数

~MyMatrix();//析构函数

void getMatrix(int rowsz=0,int colsz=0); //建立数组空间同时若声明对象时未初始化rowSIze和//colSize则在此可给它们赋有效值

Type\* operator [ ](int i) const;//取元素值

**队列结点类（QueueNode）**

队列结点要包含自身的值以及能链接到下一个结点，所以它有两个私有成员：结点数据（Type data）和链接指针（QueueNode\* link）

**私有成员及操作：**

Type data;//队列结点数据

QueueNode\* link;//结点链指针

QueueNode() :link(NULL) {};//默认构造函数

QueueNode(Type d, QueueNode\* l = NULL) :data(d), link(l) {}//带参数的构造函数

**队列类（Queue）**

一个队列的组成基本元素就是队列结点，所以为了能访问队列结点中的成员，在队列结点类中将队列类设为了友元。队列类自身则定义了两个私有成员，即队列头指针和尾指针。为了实现队列的插入、删除、取元素值，设有函数：push（插入元素）、pop(删除队头)、top(取队头结点)以及对[]的运算符重载以返回队列中指定位置结点。此外还有队列置空函数和判断是否为空函数。

**私有成员：**

QueueNode<Type>\* front, \* rear;//队列头指针和尾指针

**公共操作：**

Queue() :rear(NULL), front(NULL) {}//队列构造函数

~Queue();//队列析构函数

void push(const Type item);//将data为item的结点加入队列

Type pop();//删除队头结点

QueueNode<Type>\* top();//取出队头结点

QueueNode<Type>\* operator[](int i);//返回第i+1个结点

void MakeEmpty();//将队列置空

int IsEmpty()const { return front == NULL; }//判断队列是否为空

**地图类（Map）**

地图信息主要由一个矩阵来存储，除此之外还包含矩阵的大小参数以及入口坐标和出口坐标，同时还应包含寻找通路时用的队列，所以它一共有8个私有成员。为了实现读取地图打印地图以及找最短路径这三个功能，定义了三个函数：EnMap、print、find\_path。找最短路径时加入队列的应为所经点的坐标，同时为了能够回溯标记出虽短路径以便顺序打印路径设有一个结点坐标结构体coordinate，除包含坐标外还包含经过某点时走过的步数，通过BFS找到终点时即可根据步数递减回溯之前经过的路径从而找出最短路径。

**私有成员：**

int row, col;//行数和列数

int entrance\_x, entrance\_y, exit\_x, exit\_y;//入口和出口的坐标

MyMatrix<char> map\_matrix;//Map的内部矩阵

Queue<coordinate> path;//存储路径的队列

**公有成员及操作：**

Map(int ro, int co, int en\_x, int en\_y, int ex\_x, int ex\_y, MyMatrix<char> matrix);//构造函数

Map(const Map& copymap);//复制构造函数

void EnMap(char val, int y, int x) { map\_matrix[y][x] = val; };//将值val赋给map[y][x]

void print();//打印地图

void find\_path();//找从入口到出口的最短路径

struct coordinate {//坐标

int x, y;//坐标

int step;//经过某点时的步数

coordinate(int x1=0,int y1=0,int step1=0) :x(x1), y(y1),step(step1) {}//构造函数

};

## 2.3 系统设计

系统首先main()函数实现对屏幕的初始化，给出操作提示。读取到用户按下的回车后调用已创建好的三个地图的功能函数从而在屏幕上打印地图及最短通路路径。

# 3 实现

## 找最短路径功能的实现

### 原理流程图

起点入队

队列path非空？

取队头current

current为终点？

n

检查current上下左右的位置是否可走，可走则加入path

y

path队列置空，终点入队，当前step=队列中终点结点的step

n

step>0？

y

结束，path为最短路径的逆序

step--;依次看当前结点上下左右的结点哪一个的step等于当前的step，将该结点加入path，当前结点坐标更新为刚加入的坐标

### 核心代码

coordinate \*current=new coordinate(entrance\_x, entrance\_y,1);

//当前位置指针指向起点位置,起始步数为1

path.push(\*current);//起点位置入栈

int find = 0;//是否找到终点标记

int n = 0;//最终path里的结点个数

int i, j;

MyMatrix<int> mark(row, col);

//创建标记矩阵，标记某位置是否走过，对应位置元素的值为经过该位置时的步数

for (i = 0; i < row; i++)//初始化标记矩阵

for (j = 0; j < col; j++)

mark[i][j] = 0;

while (path.top()!=NULL) {

\*current = path.top()->data;//取队头结点的值（坐标）

int step = current->step;//取当前节点走过的步数

path.pop();//队头结点出队

int x = current->x;

int y = current->y;

mark[y][x] = step;

if (x == exit\_x && y == exit\_y) {//找到终点

find = 1;//更改find标记

path.MakeEmpty();//path队列置空

path.push(\*current);//当前位置结点（终点)入队

n++;//结点个数加1

while (step>0) {//根据步数递减回溯之前经过的路径，将最短路径所经位置存入path

step--;

map\_matrix[y][x] = '\*';//将最短路径所经位置在map矩阵中标记出来

if (step == 1)

break;

if (y>0&&mark[y - 1][x] == step) {

path.push(coordinate(x, y - 1));

y -= 1;

}

else if (y<row&&mark[y + 1][x] == step) {

path.push(coordinate(x, y + 1));

y += 1;

}

else if (x>0&&mark[y][x - 1] == step) {

path.push(coordinate(x - 1, y));

x -= 1;

}

else if (x<col&&mark[y][x + 1] == step) {

path.push(coordinate(x + 1, y));

x += 1;

}

n++;

}

break;//退出循环

}

if (y > 0 && map\_matrix[y - 1][x] != '#'&&mark[y-1][x]==0) {//向上可走

path.push(coordinate(x, y - 1, step + 1));//上面的位置入队

}

if (y < row - 1 && map\_matrix[y + 1][x] != '#' && mark[y + 1][x] == 0) {//向下可走

path.push(coordinate(x, y + 1, step + 1));//下面的位置入队

}

if (x > 0 && map\_matrix[y ][x-1] != '#' && mark[y][x - 1] == 0) {//向左可走

path.push(coordinate(x - 1, y, step + 1));//左边的位置入队

}

if (x < col-1 && map\_matrix[y][x + 1] != '#' && mark[y][x + 1] == 0){//向右可走

path.push(coordinate(x + 1, y, step + 1));//右边的位置入队

}

}

# 4 测试

## 4.1 功能测试

**测试用例**：地图如下，#表示障碍物，0表示可通路

# # # # # # #

# 0 # 0 0 0 #

# 0 # 0 # # #

# 0 0 0 # 0 #

# 0 # 0 0 0 #

# 0 # 0 # 0 #

# # # # # # #

**预期结果**：

# # # # # # #

# 0 # 0 0 0 #

# \* # 0 # # #

# \* \* \* # 0 #

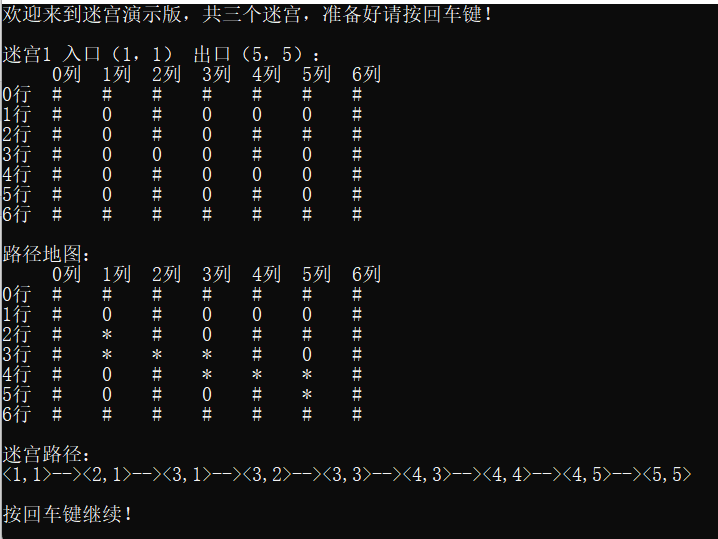
# 0 # \* \* \* #

# 0 # 0 # \* #

# # # # # # #

**<1,1>--><2,1>--><3,1>--><3,2>--><3,3>--><4,3>--><4,4>--><4,5>--><5,5>**

**实验结果**



## 4.2 边界测试

**测试用例**：

0 0 0 # 0

0 # 0 # #

0 # 0 0 0

0 # # # #

0 # 0 0 0

**预期结果**：

起点到终点间不存在通路！

**实验结果**

