+

项目说明文档

数据结构课程设计

——勇闯迷宫游戏

作 者 姓 名： 祝新元

学 号： 1751629

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](#_Toc531522808)

[1.1 项目简介 1](#_Toc531522809)

[1.2 功能分析 1](#_Toc531522810)

[2 设计 2](#_Toc531522811)

[2.1 数据结构设计 2](#_Toc531522812)

[2.2 类结构设计 2](#_Toc531522813)

[2.3 成员与操作设计 2](#_Toc531522814)

[2.4 设计 3](#_Toc531522815)

[3 实现 4](#_Toc531522816)

[3.1 地图等的初始化 4](#_Toc531522817)

[3.1.1 地图的初始化 4](#_Toc531522818)

[3.1.2 走过路径的初始化 4](#_Toc531522819)

[3.1.3 方向 4](#_Toc531522820)

[3.2 寻路功能 4](#_Toc531522821)

[3.2.1 寻路功能核心代码 4](#_Toc531522822)

[3.3 程序主体 5](#_Toc531522823)

[3.3.1 输出地图 5](#_Toc531522824)

[3.3.2 寻路并输出 6](#_Toc531522825)

[4 测试 6](#_Toc531522826)

[4.1 结果展示 6](#_Toc531522827)

# 1 分析

## 1.1 项目简介

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。

## 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。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，该项目要求及时纠正错误另外选择道路，所以需要保存之前的路径。同时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，所以我选择栈来保存每一步走的路径。地图采用二维数组，用1 表示墙壁，不能行走；用0表示空地，可以行走，最终出口（5，5）和入口（1，1）都先设置好。

## 2.2 类结构设计

考虑到总的路径不会很多，本项目采用数组（elements）保存栈里面的每一个元素。栈内私有成员包括栈的大小和一个top指针，用来处理溢出等各种情况。另外用一个struct offset来表示方向，保存了一个二维方向的变化。

## 2.3 成员与操作设计

**方向类**

//位置在直角坐标下的偏移

struct offsets

{

int a,b; //a refer to x,b refer to y

};

**栈类（**SeqStack**）**

**私有成员：**

T \*elements; //保存栈内元素的数组

int top; //头指针

int maxSize; //最大大小

**公有操作：**

SeqStack<T>::SeqStack(int sz)

//SeqStack的构造函数，新建一个sz大小的数组

SeqStack<T>::~SeqStack()

//SeqStack的析构函数，删除数组

void SeqStack<T>::overflowProvess()

//SeqStack的溢出函数，如果溢出，则再次开辟maxsize大小的空间

void SeqStack<T>::Push(const T& x)

//值为x的元素入栈，top++

bool SeqStack<T>::Pop(T& x)

//出栈，出栈的值保存在x里面

bool SeqStack<T>::getTop(T& x)const

//得到栈顶元素的值，记录在x里面

bool IsEmpty()const

//栈是否为空

bool IsFull() const

//栈是否为满

int getSize()const

//得到栈的大小

## 2.4 设计

通过默认地图或者手动输入地图的方式来定义地图（全局二维数组Maze），和地图对应给每一个位置赋一个值来标记这个位置是否已经走过或者是无法行走（二维数组mark），用二维数组Move来定义行走的方向。程序首先调用SeekPath函数寻找路径，找到的路径会自动输入到栈里面，只需要把一个一个出栈就能输出所有路径。

# 3 实现

## 3.1 地图等的初始化

### 3.1.1 地图的初始化

int Maze[m + 2][p + 2] = {{1,1,1,1,1,1,1},

{1,0,1,0,0,0,1},

{1,0,1,0,1,1,1},

{1,0,0,0,1,0,1},

{1,0,1,0,0,0,1},

{1,0,1,0,1,0,1},

{1,1,1,1,1,1,1}};

/\*也可以通过输入地图来定义地图

for(i=0;i<m+2;i++)

{

for(j=0;j<p+2;j++)

{

cin>>Maze[i][j];

}

}\*/

### 3.1.2 走过路径的初始化

int mark[m+2][p+2];

for(i =0;i<m+2;i++)

{

for(j=0;j<p+2;j++)

{

mark[i][j]=0;//0表示没有走过

}

}

mark[1][1]=1;

//出发点已经经过

### 3.1.3 方向

offsets Move[8]={{-1,0},{-1,1},{0,1},{1,1},{1,0},{1,-1},{0,-1},{-1,-1}};

//代表八个方向

## 3.2 寻路功能

### 3.2.1 寻路功能核心代码

int SeekPath(int x,int y,SeqStack<offsets> &s)

{

int i,g,h;

if(x==m&&y==p) return 1;

for(i=0;i<8;i++)

{

g=x+Move[i].a;

h=y+Move[i].b;

if(Maze[g][h]==0 &&mark[g][h]==0) //下一位置可以

{

mark[g][h]=1;//标记这个位置走过

if(SeekPath(g,h,s))

{

offsets newoff={g,h};

s.Push(newoff);

//cout<<"("<<g<<","<<h<<") ---> ";

return 1;

}

}

}

if(x==1 &&y==1)cout<<"No path in Maze"<<endl;

return 0;

}

## 3.3 程序主体

### 3.3.1 输出地图

cout << "迷宫地图："<<endl;

for (i = 0; i < m + 2; i++)

{

for (j = 0; j < p + 2; j++)

{

if (Maze[i][j] == 1)

{

cout << '#';

}

else

{

cout << '0';

}

}

cout << endl;

}

cout << endl;

### 3.3.2 寻路并输出

cout << "迷宫路径：" << endl;

mark[1][1]=1;//出发点已经经过

if(SeekPath(1,1,stk)==1)

{

cout<<"("<<1<<","<<1<<")";

}

offsets offout;

while(stk.Pop(offout))

{

cout<< " ---> ("<<offout.a<<","<<offout.b << ')';

}

# 4 测试

## 4.1 结果展示

