项目说明文档
——勇闯迷宫游戏

姓名：吴桐欣
学号：1652677
同济大学 软件学院 软件工程专业

目录

1.项目概述

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。
项目功能要求：
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。

1.1.1 功能分析

本项目要求实现功能
（1） 回溯法
（2） 输出走迷宫的路径

1.2 文件目录

（1）P03_1652677_吴桐欣_说明文档.docx（本文档）
（2）P03_1652677_吴桐欣.exe（可执行文件）
（3）P03_1652677_吴桐欣.cpp（源文件）
（4）P03_1652677_吴桐欣.h（头文件）
（5）迷宫输入样例.txt（测试输入样例）

1.3 操作指南

本程序要求用户自行输入迷宫地图。
运行程序后，将获得程序提示“输入迷宫地图的行数和列数：”和“请输入空位置的坐标“行 列”，一个坐标一行；若已输入完毕，行、列输入为-1”
用户需按要求输入行数，列数和坐标数。
随后再随系统提示“请输入迷宫入口的坐标(行 列)：请输入迷宫出口的坐标(行 列)：”分别输入迷宫入口和出口的坐标。
至此完成迷宫地图的绘制，即可获得迷宫的解法。
[输入样例]
输入迷宫地图的行数和列数：2 2
请输入空位置的坐标“行 列”，一个坐标一行；若已输入完毕，行、列输入为-1
1 0
-1 -1
请输入迷宫入口的坐标(行 列)：0 1
请输入迷宫出口的坐标(行 列)：1 1
[输出样例]
迷宫地图：
0 1

0 # S
1 0 E

迷宫路径:
(0,1)->(1,1)

1.4 注意事项

(1) 用户不得输入除数字以外的字符
(2) 用户所输入的坐标不能大于等于行数或列数

2.思路与设计

2.1 基本思路

每走一步，就对该位置的四个方向进行扫描，寻找可走的下一位置。如果没有，则说明此路不通，需要返回上一岔口；如果有一个，则走到该位置上；如果有不止一个，则此位置为一个岔口，需要将此位置的数据压入栈内，再任选一个可走的位置往下走。此题用到回溯法，用栈对已走过的位置进行存储。回溯时，弹出栈顶的数据，返回上个岔口。

2.2 设计

2.2.1 数据结构

主要用了两个数据结构来存储信息。一个是 path，即走过的路径，存储了走过的每一个位置，用于最终输出迷宫的解决路径。一个是 cross，即十字路口，这些位置都有不止一个可能走向。回溯时只需回到岔口即可，不必倒走回去对每一位置检查一遍。但是两个数据结构的元素是不同的，path 只用存储位置信息，而 cross 不仅要存储岔口的位置，还要存储岔口的所有可能走向。

2.2.2 成员与成员函数

为了存储数据直观，用 Point 结构将棋盘上每一个位置的行(x)、列(y)数都放在一个 Point 中，使一个迷宫中的位置只对应一个 Point。重载操作符是为了使用方便。path 内存储的元素就是 Point 结构

```
struct Point{
    int x;
    int y;
    //重载操作符
    Point& operator = (const Point& rhs){
        x = rhs.x;
        y = rhs.y;
        return *this;
    };
    bool operator != (const Point& rhs){
        if (x==rhs.x && y==rhs.y){return false;}
        else return true;
    };
    bool operator == (const Point& rhs){
        if (x==rhs.x && y==rhs.y){return true;}
        else return false;
    };
};
```

step 结构是为了能够在每一步上存储位置和可能的走向，将位置和可能的走向紧密结合。cross 中存储的元素就是 step 结构的。

```
struct step{
    Point pos;//现在的位置
    vector<Point> choice;//可能的下一位置
    //重载操作符
    step& operator = (const step& rhs){
        pos = rhs.pos;
        choice = rhs.choice;
        return *this;
    };
};
```

solut 类是将整个解题程序整合在一起，方便获取数据和调用函数。

```
class solut{
public:
    int* _track;//记录走过的地方
    char* _map;//迷宫地图
    stack<step> _cross;//经过的岔口
    stack<Point> _path;//走过的路径

    solut(int* track=NULL, char* map=NULL){_track=track;_map=map;};
    ~solut(){};
    bool findWay(step* cur, int col, int row); //寻找某一位置的下一可能位置
};
```

3.具体实现

3.1 用户自定义迷宫地图

核心代码

```
int x,y;
    cout<<"请输入空位置的坐标“行 列”，一个坐标一行；若已输入完毕，
行、列输入为-1"<<endl;
    while (true) {
        cin>>x>>y;
        map[x][y]='0';
        if (x==-1 && y==-1) {
            break;
        }
    }
```

说明

用一个存储 char 类型的二维数组来表示迷宫地图，其内所有元素初始化为‘#’，表示墙壁。用户输入空位置的坐标，则将相应位置的元素值更改为‘0’，表示通路可走。

3.2 寻找某一位置的下一可能位置

---

findWay(step* cur, int col, int row)函数

函数代码

```
bool solut::findWay(step* cur, int col, int row){
    int x=cur->pos.x,y=cur->pos.y;
    //上侧
    if (_map[up(x, y, col)] == '0' && _track[up(x, y, col)] == 0 &&
within(row, x-1)) {
        cur->choice.push_back({x-1,y});
    } else if(_map[up(x, y, col)] == 'S'){
        cur->choice.clear();
        cur->choice.push_back({x-1,y});
        return true;
    }
    //下侧
    if (_map[dw(x, y, col)] == '0' && _track[dw(x, y, col)] == 0 &&
within(row, x+1)) {
        cur->choice.push_back({x+1,y});
    } else if(_map[dw(x, y, col)] == 'S'){
        cur->choice.clear();
        cur->choice.push_back({x+1,y});
        return true;
    }
    //左侧
    if (_map[le(x, y, col)] == '0' && _track[le(x, y, col)] == 0 &&
within(col, y-1)) {
        cur->choice.push_back({x,y-1});
    } else if(_map[le(x, y, col)] == 'S'){
        cur->choice.clear();
        cur->choice.push_back({x,y-1});
        return true;
    }
    //右侧
    if (_map[ri(x, y, col)] == '0' && _track[ri(x, y, col)] == 0 &&
within(col, y+1)) {
        cur->choice.push_back({x,y+1});
    } else if(_map[ri(x, y, col)] == 'S'){
        cur->choice.clear();
        cur->choice.push_back({x,y+1});
        return true;
    }
    return false;
}
```

说明

如果下一位置没走过（track 中的值为 0 而不是 1），并且是通路（map 中符号为 0 而不是#），并且位于迷宫地图范围内（0～row/col），则认为该位置有效，存进 choice 中。

宏定义

以上函数代码中应用了宏定义，定义式如下。

```
#define ctr(x,y,n) x*n+y
#define up(x,y,n) (x-1)*n+y
#define dw(x,y,n) (x+1)*n+y
#define le(x,y,n) x*n+y-1
#define ri(x,y,n) x*n+y+1
#define within(border,x) (x<border?1:0)&&(x>=0)
```

3.3 回溯法解出迷宫路径

核心代码

```
//寻找通路
    solut solut1(*track, *map);
    step* s = new step;
    s->pos = exitP;
    track[exitP.x][exitP.y] = 1;
    solut1._path.push(s->pos);

    while (!solut1.findWay(s, col, row)) {
        //若无路可走
        while (s->choice.size() == 0) {
            //返回上一岔口
            if (solut1._cross.empty()) {
                cout<<"无解！"<<endl;
                exit(0);
            } else {
                s->pos = solut1._cross.top().pos;
                s->choice = solut1._cross.top().choice;
            }
            //清除此段路径痕迹，包括path和track
            while (s->pos != solut1._path.top()) {
                track[solut1._path.top().x][solut1._path.top().y]=0;
                solut1._path.pop();
            }
            //删除这一选项
            s->choice.pop_back();
            solut1._cross.top().choice.pop_back();
            //若删除后只剩一条去路，不再是岔口
            if (s->choice.size()==1) {solut1._cross.pop();}
            //若删除后没有去路，则此路不通，继续循环返回上一岔口
        }
        //若此处为岔口
        if(s->choice.size() > 1){
            //标记岔口
            solut1._cross.push(*s);
        }
        //更新s，往下走
        s->pos = s->choice.back();
        s->choice.clear();
        //标记痕迹
        solut1._path.push(s->pos);
        track[s->pos.x][s->pos.y] = 1;

    }
    solut1._path.push(start);
```

说明

因为最后输出的路径是从入口到出口，但栈是后进先出的结构，所以我们解题时是从出口找到入口，则逆输出时可以得到从入口到出口的路径。
s 代表着当前行进到的位置。每更新一次 s，意味着走到一个新的位置，调用 findWay 函数寻找新位置的下一个可能位置。findWay 函数具有返回值，若 s 四周的位置中存在入口，则说明解谜成功，返回 bool 值为 true，否则为 false。当 findWay 的返回值为 true 时，停止 while 循环。
最后一次在扫描发现入口后推出循环，此时入口位置没被压入 path 栈中，因此循环结束后要补上入口的位置坐标。

3.4 输出迷宫路径

核心代码

```
//输出结果
    cout<<"迷宫路径: "<<endl;
    for (; ; ) {
        cout<<solut1._path.top()<<"->";
        solut1._path.pop();
        if (solut1._path.size()==1) {
            cout<<solut1._path.top()<<endl;
            solut1._path.pop();
            break;
        }
    }
```

说明

依次弹出记录路径的 path 栈内的元素，并输出，得到结果。

4.测试

4.1 只有一种解法的地图

```
迷宫地图：
   0  1  2  3  4  5  6  7  8  9
0  #  S  0  #  0  0  0  #  0  #
1  #  #  0  #  0  #  #  #  0  #
2  #  #  0  0  0  0  0  0  0  #
3  #  #  #  #  0  #  #  #  0  #
4  #  #  0  0  0  #  #  #  0  0
5  0  0  #  0  #  0  0  0  #  0
6  #  0  0  0  0  0  #  0  #  0
7  #  #  #  0  #  #  0  0  #  #
8  #  #  #  0  0  0  #  0  0  #
9  #  0  0  0  #  #  #  #  E  #

迷宫路径：
(0,1)->(0,2)->(1,2)->(2,2)->(2,3)->(2,4)->(3,4)->(4,4)->(4,3)->(5,3)
->(6,3)->(6,4)->(6,5)->(5,5)->(5,6)->(5,7)->(6,7)->(7,7)->(8,7)->(8,
8)->(9,8)

--------------------------------
```

4.2 有多种解法的地图

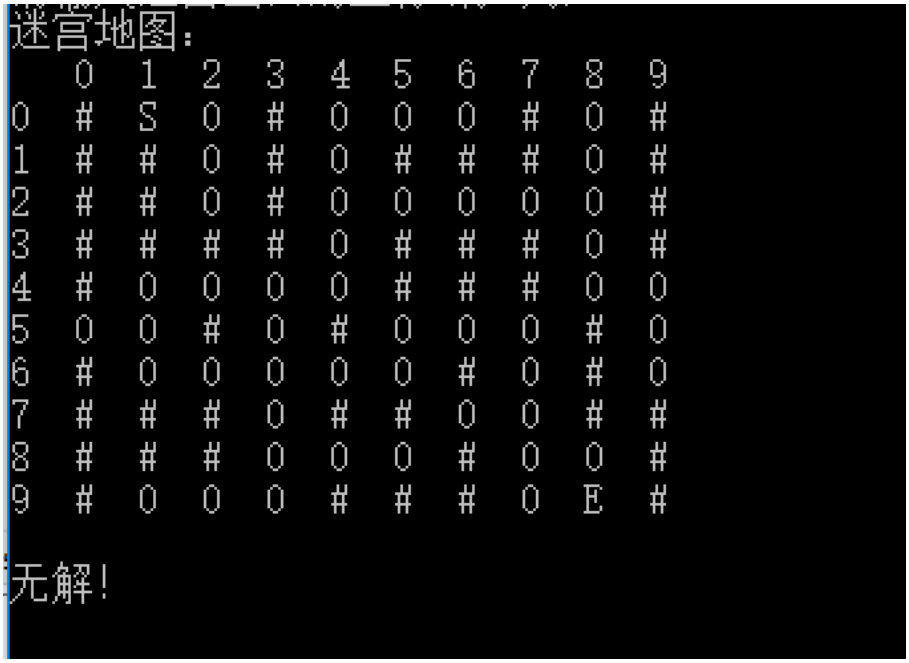
程序只会输出最先找到的一个解法，不一定是最短路径。

```
迷宫地图：
   0  1  2  3  4  5  6  7  8  9
0  #  S  0  #  0  0  0  #  0  #
1  #  #  0  #  0  #  #  #  0  #
2  #  #  0  0  0  0  0  0  0  #
3  #  #  #  #  0  #  #  #  0  #
4  #  0  0  0  0  #  #  #  0  0
5  0  0  #  0  #  0  0  0  #  0
6  #  0  0  0  0  0  #  0  #  0
7  #  #  #  0  #  #  0  0  #  #
8  #  #  #  0  0  0  #  0  0  #
9  #  0  0  0  #  #  #  0  E  #

迷宫路径：
(0,1)->(0,2)->(1,2)->(2,2)->(2,3)->(2,4)->(3,4)->(4,4)->(4,3)->(4,2)->(4,1)
->(5,1)->(6,1)->(6,2)->(6,3)->(6,4)->(6,5)->(5,5)->(5,6)->(5,7)->(6,7)->(7,
7)->(8,7)->(9,7)->(9,8)

--------------------------------
```

4.3 无解的地图


迷宫地图:
0 1 2 3 4 5 6 7 8 9
0 # S 0 # 0 0 0 # 0 #
1 # # 0 # 0 # # # 0 #
2 # # 0 # 0 0 0 0 0 #
3 # # # # 0 # # # 0 #
4 # 0 0 0 0 # # # 0 0
5 0 0 # 0 # 0 0 0 # 0
6 # 0 0 0 0 0 # 0 # 0
7 # # # 0 # # 0 0 # #
8 # # # 0 0 0 # 0 0 #
9 # 0 0 0 # # # 0 E #
无解!