# Assignment_3　勇闯迷宫游戏

## 一、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。

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。

## 二、构建设想：

根据题意可以看出这是一个有关搜索的题目，并且这可以算是一个深度优先搜索的题目，其中DFS中的一个重要的思想就是回溯，利用回溯的话是非常适合解决这类题目的。

**因为地图没法改变，上交的程序中我使用了默认地图，如果想测试不同地图的话请助教修改代码... 不同地图的测试我也在程序运行截图中展示了,谢谢~**

## 三、程序设计：

### 1. 程序重要的变量：

因为这个题目要求的代码量比较少(主要是使用递归解决的问题，代码总是简洁明了的),我就没有使用类来解决这个问题，如下是程序中用到的比较重要的变量的声明：

```
//坐标
struct Position
{
    int x;
    int y;
};

//迷宫
char maze[100][100] =
{
        "#######",
        "#0#000#",
        "#0#0###",
        "#00000#",
        "#0###0#",
        "#0#0#0#",
        "#######",
};

//每次移动的方向
Position direction[4] ={ {0,1},{0,-1},{-1,0},{1,0} };

//出口的位置
Position exitPos = {5,5};

//用来存储每一个经过的坐标
vector<Position> route;

//判断是否到达了出口
int tag = 0;
```

## 2.程序中函数的实现

**(1) 深度优先搜索函数 DFS() :**

这个函数是整个迷宫游戏的核心函数,在函数开始时判断是否发现了出口,如果没有发现就对四个方向进行遍历知道发现出口为止,如下是DFS函数的实现:

```
//递归深度搜索
void dfs(int x,int y)
{
    //发现出口即可退出
    if(x ==  exitPos.x && y == exitPos.y)
    {
        tag = 1;
        route.push_back(exitPos);
        return;
    }

    //每一层都对四个方向遍历
    for(int i = 0; i < 4 ; i++)
    {
        //如果某个方向可以走的话,进入下一层
        if(maze[x + direction[i].x][y + direction[i].y] == '0')
        {
            maze[x][y] = 'F';
            Position cur = {x,y};
            route.push_back(cur);

            dfs(x + direction[i].x,y + direction[i].y);

            //如果找到出口,返回
            if(tag == 1)
                return ;
            //递归返回,消除痕迹
            route.erase(route.end() - 1);
            maze[x][y] = '0';
        }

    }
}
```

**(2) 地图打印函数 printMap() :**

程序开始时对迷宫进行打印,如下是函数的具体实现代码:

```
//打印地图
void printMap()
{
    cout << "迷宫地图:\n\t";

    for(int i = 0; i < 7; i++)
        cout << i << "列" <<"\t\t";

    cout << endl;
    for(int i = 0; i < 7; i++)
    {
        cout << i << "行" <<"\t";
        for(int j = 0 ;  j < 7; j++)
        {
            cout << maze[i][j]  <<"\t\t";
        }
        cout << endl << endl;
    }
}
```

**(3) 路线打印函数 printRoute()：**

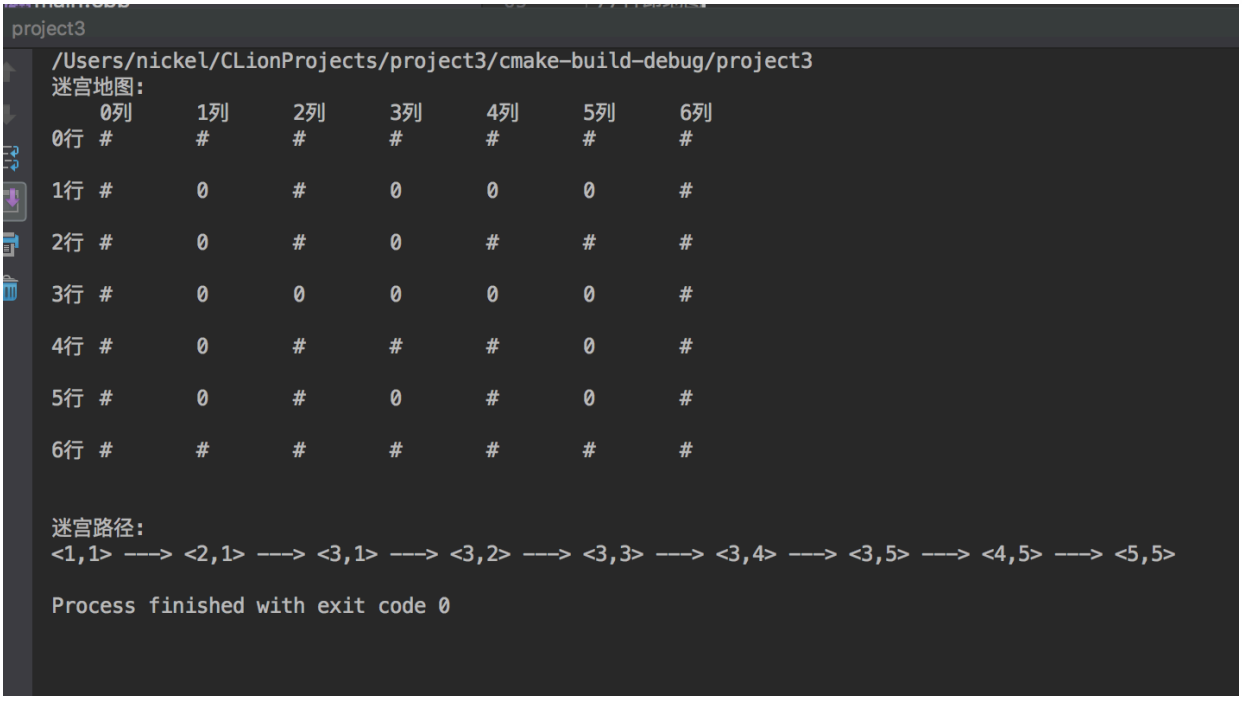
在找到出口之后根据记录的路线打印，如下是函数的具体实现:

```
//打印路线
void printRoute()
{
    cout <<"\n迷宫路径:" << endl;
    for(auto iter = route.begin(); iter != route.end() - 1;iter++)
        cout << "<" << iter->x << "," << iter->y << "> "<< "---> ";
    cout << "<" << (route.end() - 1)->x << "," << (route.end() - 1)->y <<
">"<<endl;
}
```

# 四、程序运行截图：

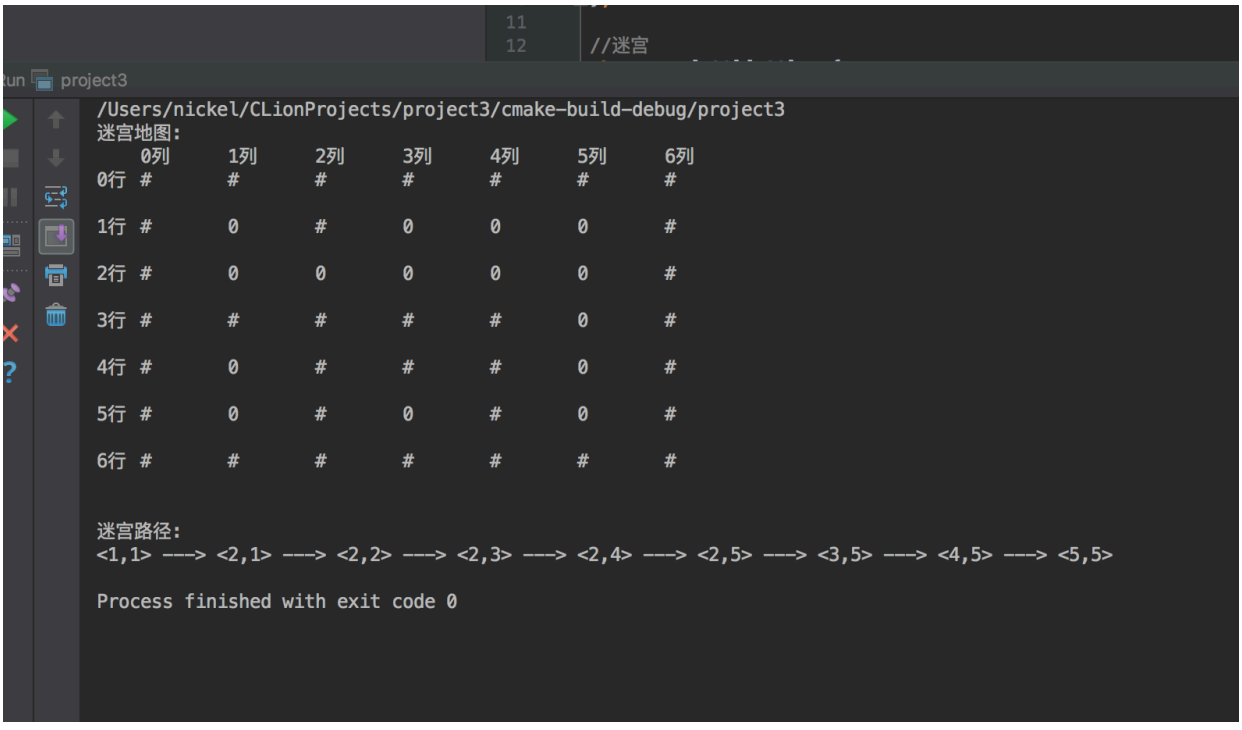
## 1. 截图展示1---默认迷宫：

此地图为题目要求中所给的默认迷宫:



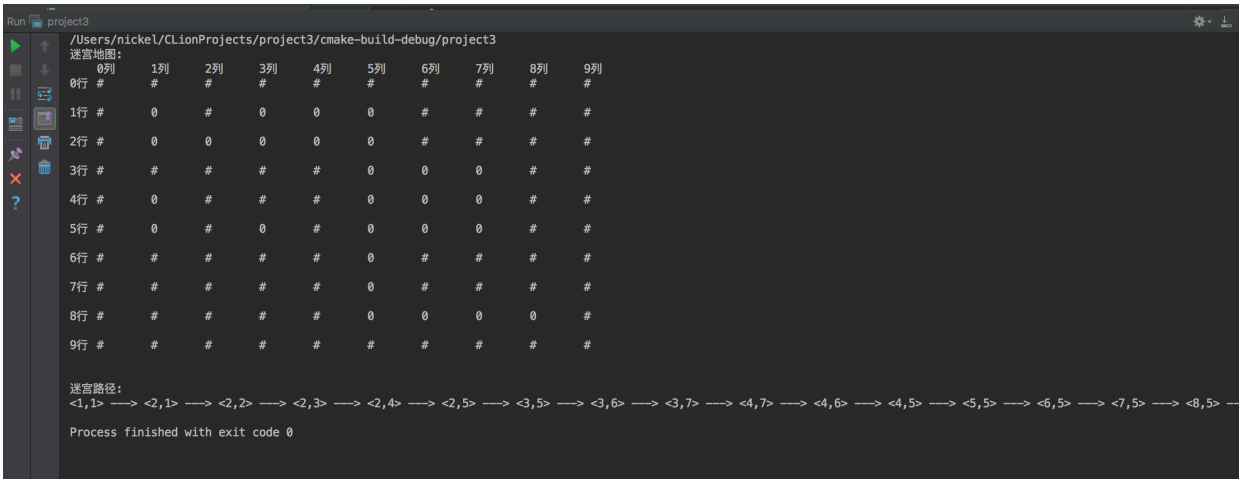

## 2. 截图展示2—自建迷宫：

此地图与默认迷宫的大小相同,改变了一些路线,如图所示:




## 3. 截图展示3—自建迷宫：

这张地图与原来地图的大小不同,换成了10 * 10 的地图,路径结果如图所示:




# 五、程序容错性测试：

## 地图没有出口的情况下测试：

在程序中我设置了tag变量，初始为0，若达到出口点便将tag设置成1,最后加以判断，若没有到达出口(返回到了原点)便提示找不到出口并退出程序.