项目说明文档

数据结构课程设计

——勇闯迷宫游戏

作 者 姓 名：翟晨昊

学　　号：1952216

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

1.1 背景分析

相信每个人都或多或少的接触过迷宫类型的游戏，在迷宫里有着两个门，分别为入口和出口，人物需要从入口出发，避开障碍，分辨岔路口，最后找到一条路能够通往出口。本来就来设计一个程序来复刻这一类经典的迷宫游戏。

1.2 功能分析

首先一个迷宫游戏必须要有地图，这里为了让游戏更有可玩性，设置了一张25*25大小的地图，内部设有一个出口和一个入口。其次，要能够使游戏人物寻找通路以到达出口。最后，游戏还要能保存人物最终通过迷宫的路径并显示。

综上所述，一个迷宫游戏需要有建立地图，人物寻找通路，输出通过路径并显示的功能。

2 设计

2.1 数据结构设计（DFS 示意）

如上功能分析所述，地图已经固定，使用一个二维数组存储即可。而路径由于在进行迷宫搜索时会动态变化，无法确定长度，因此决定使用 Vector 向量来保存路径。迷宫最重要的是搜索通路功能，图形搜索算法最广泛使用的是深度优先搜索（DFS）与广度优先搜索（BFS），本游戏最终采用了深度优先搜索（DFS）的数据结构。

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。

2.2 类结构设计

首先有一个保存迷宫路径的 Vector 类，其次，为了更方便管理，将迷宫地图、迷宫搜索任意时刻的状态与搜索时需要的操作整合进一个迷宫类（Maze 类）中。最后设计了一个坐标点类（Point 类）来表示迷宫内的点。为了使 Vector 向量更具泛用性，将 Vector 设计为模板类。

2.3 成员与操作设计

向量类（Vector）：

```
template <typename ElementType> class Vector {
public:
    ~Vector();
    Vector();
    ElementType& operator[](const int x);
    const ElementType& operator[](const int x)const;
    Vector<ElementType>(const Vector<ElementType>& rhs);
    Vector<ElementType>& operator=(const Vector<ElementType>& rhs);
    bool isFull();
    bool isEmpty();
    void pushBack(const ElementType& temp);
    void popBack();
    void clear();
    int getSize();
    void reSize(int newSize);
private:
    void extendSize();
```

```
    int size;
    int maxSize;
    ElementType* myVector;
};
```

私有成员：

int size;//Vector 中已经存储的元素数量
int maxSize;//Vector 中已经申请的空间大小，元素数量如果超过要再次申请
ElementType* myVector;//存储的元素序列的起始地址

私有操作：

void extendSize();
//在 Vector 申请的空间已经被占满时再次申请空间，每次扩充至 maxsize*2+1
是因为刚开始的 maxsize 为 0

公有操作：

Vector();
//构造函数，初始化指针并将 size 与 maxSize 置为 0

~Vector();
//析构函数，调用 clear()函数来删除元素，释放内存

ElementType& operator[](const int x);
//重载[]运算符，返回 ElementType&类型

const ElementType& operator[](const int x)const;
//重载[]运算符，返回 const ElementType&类型

Vector<ElementType>(const Vector<ElementType>& rhs);
//复制构造函数，将一个 Vector 复制给另一个 Vector

Vector<ElementType>& operator=(const Vector<ElementType>& rhs);
//重载=运算符，可以将一个 Vector 赋给另一个 Vector

bool isFull();
//判断是否 Vector 中申请的内存已经被占满

bool isEmpty();
//判断 Vector 是否为空

void pushBack(const ElementType& temp);
//在 Vector 末尾添加一个元素

void popBack();

```
//删除 Vector 最末尾的元素

void clear();
//删除 Vector 中的所有元素并释放内存

int getSize();
//返回 Vector 已经存储的元素数量

void reSize(int newSize);
//重设 Vector 的大小，若比之前小，则会抛弃多余的元素
```

坐标点类（Point）：

```
class Point {
public:
    Point(int inputX, int inputY) :x(inputX), y(inputY) {}
    Point() : x(0), y(0) {}
    friend ostream& operator<<(ostream& os, const Point& temp);
    friend class Maze;
private:
    int x;
    int y;
};
```

私有成员：

```
int x;//坐标点的横坐标
int y;//坐标点的纵坐标
```

公有操作：

```
Point(int inputX, int inputY);
//含参构造函数

Point();
//无参构造函数，将 x，y 初始化为 0，0

friend ostream& operator<<(ostream& os, const Point& temp);
//重载<<运算符并声明为友元，使该类可以进行<<运算

friend class Maze;
//将迷宫类声明为友元，使得搜索路径时可以直接访问某点坐标
```

迷宫类（Maze）：

```
class Maze {
```

```
public:
    Maze(int size):mazeSize(size){}
    void printMap();
    bool DFS(Point &curPoint);
    void outputPath();
    void findPath();
private:
    char mazeMap[30][30] = {
        {"##########################"},
        {"#S0######00#0000######000#"},
        {"#0000##0####000000000000#"},
        {"####0##0##0#0##0#######0#"},
        {"#0000#0##00#0###0000###0#"},
        {"#0##00##000000###0000000#"},
        {"#0##00000##0#0###0######"},
        {"#0######000#####00000#0###"},
        {"#0000000####00####000000#"},
        {"###0######0##0#0#0#0##0##"},
        {"###0####00##00000000##0##"},
        {"#0##00000###0####0#000###"},
        {"#0##########0#00##0######"},
        {"#0#0000000000####00000###"},
        {"#0#0##############0#0#0###"},
        {"###0000000000000#0#0#0###"},
        {"################0#0###000#"},
        {"#00#0000000####000#####0#"},
        {"#00#0#####000#######0000#"},
        {"#00#0##000##0000000000####"},
        {"###000##0#############000#"},
        {"##00#0##0##0000##000#####"},
        {"##0000000000##0##0#00000#"},
        {"#000#0#00#####0000#####E#"},
        {"##########################"},
    };
    int knightMove[4][2] = { {1, 0}, {0, 1}, {-1, 0}, {0, -1} };
    Vector<Point> path;
    int hasPassed[25][25] = { 0 };
    int mazeSize;
    int curSize = 0;
    Point start = Point(1, 1);
};
```

私有成员：

```
char mazeMap[30][30];
```

```
//初始化迷宫地图，其中 S 代表起点，E 代表终点，0 代表可以走的点，#代表不可以走的点
int knightMove[4][2] = { {1, 0}, {0, 1}, {-1, 0}, {0, -1} };
//深度优先搜索（DFS）时需要的四个坐标变化量
Vector<Point> path;//保存走过的路径
int hasPassed[25][25];//记录某一点是否已经走过
int mazeSize;//迷宫的一边长度
int curSize;//当前路径中的坐标个数
Point start;//起点坐标
```

公有操作：

```
Maze(int size);
//含参构造函数，初始化迷宫边长

void printMap();
//打印并展示当前迷宫

bool DFS(Point &curPoint);
//深度优先搜素（DFS）函数

void outputPath();
//输出通路路径

void findPath();
//搜索通路
```

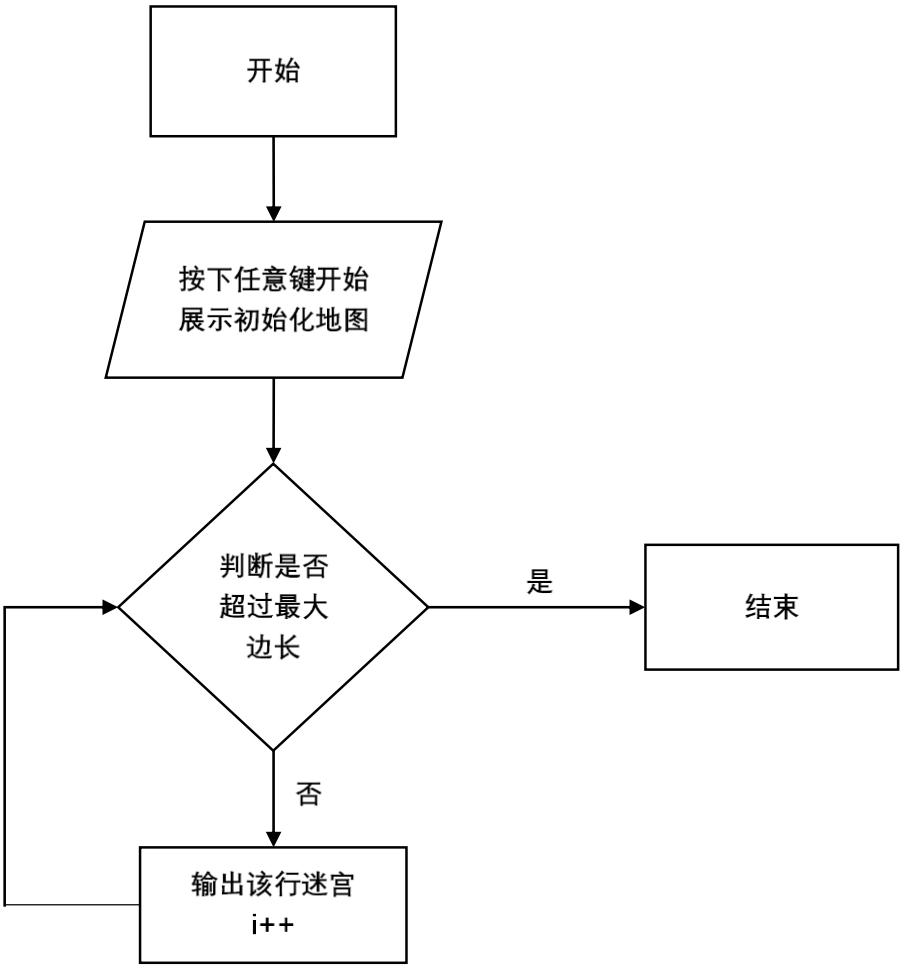
2.4 系统设计

开始游戏后，首先按下任意键来显示初始化的地图，其中 S 代表起点，E 代表终点，0 代表可以走的点，#代表不可以走的点。随后游戏会搜索通路路径，完毕后用户再次按下任意键，如果没有通路则会输出提示，如果有通路，则会输出在迷宫中的通路，以及从起点到终点，路径中每一个点的坐标。

3 实现

3.1 初始化迷宫功能的实现

3.1.1 初始化迷宫功能流程图

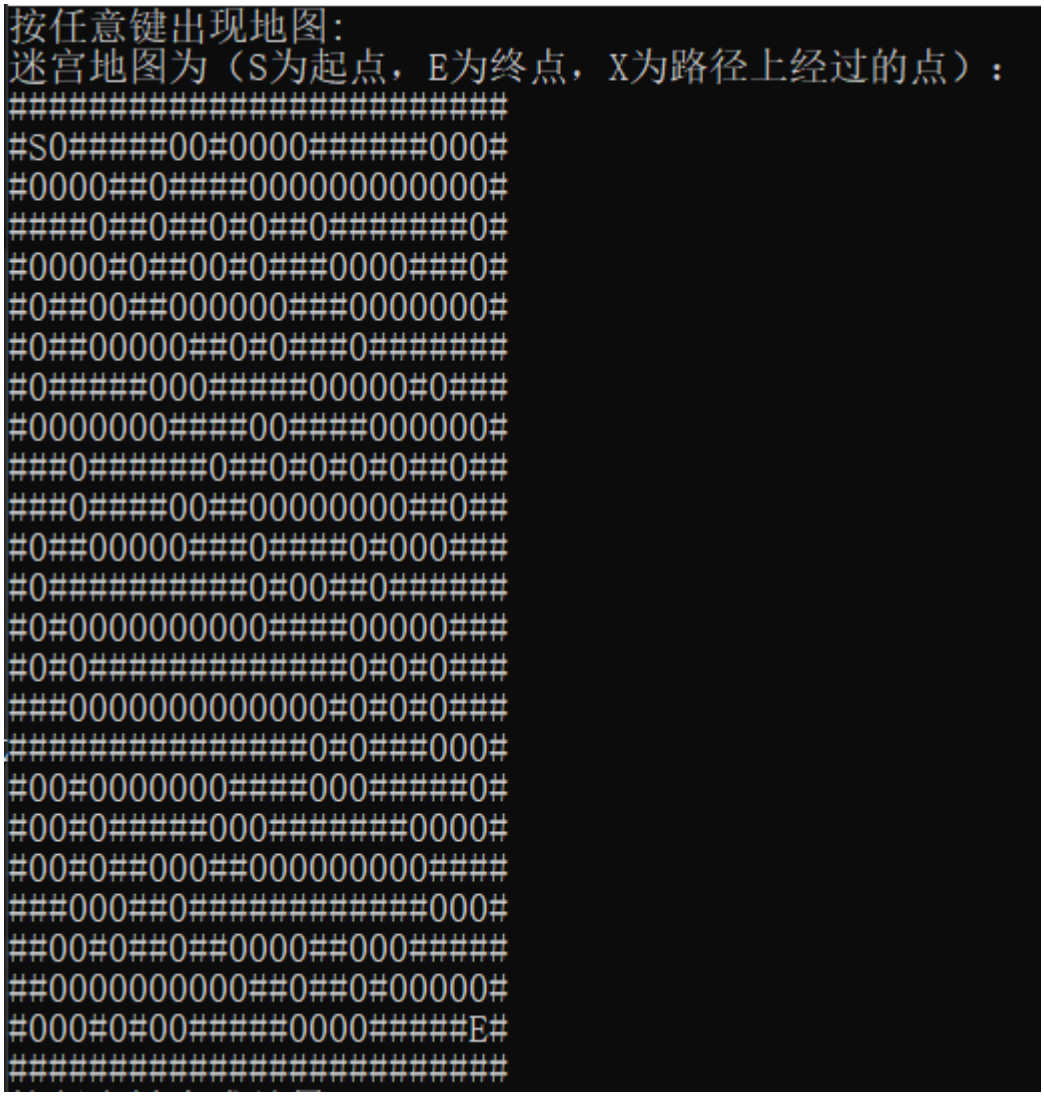


3.1.2 初始化迷宫功能核心代码

Maze 类中：

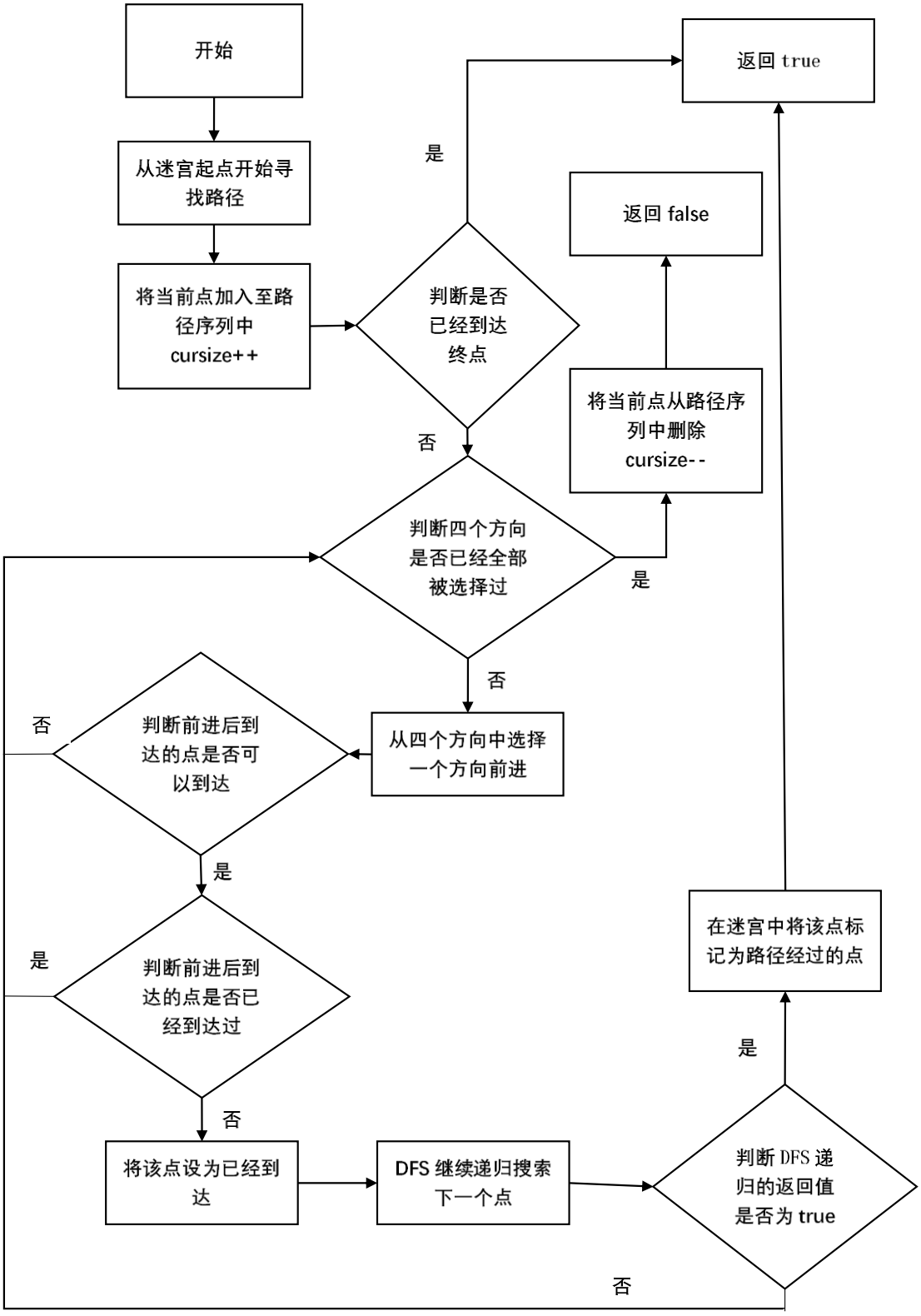
```
void Maze::printMap()
{
    cout << "迷宫地图为（S 为起点，E 为终点，X 为路径上经过的点）：" << endl;
    for (int i = 0; i < mazeSize; i++)
    {
        cout << mazeMap[i] << endl;
    }
}
```

3.1.3 初始化迷宫功能截屏示例



3.2 寻找路径功能的实现

3.2.1 寻找路径功能流程图

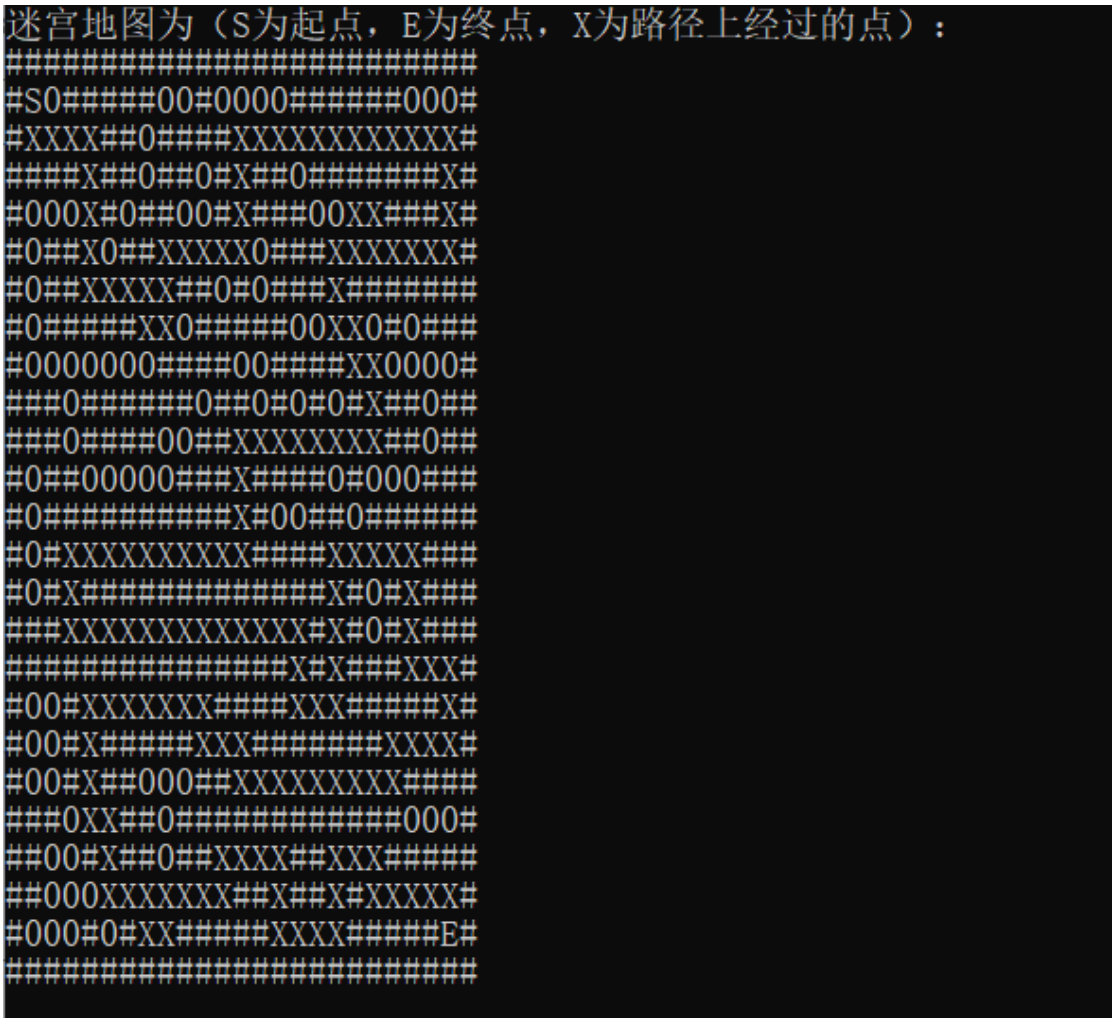


3.2.2 寻找路径功能核心代码

Maze 类中：

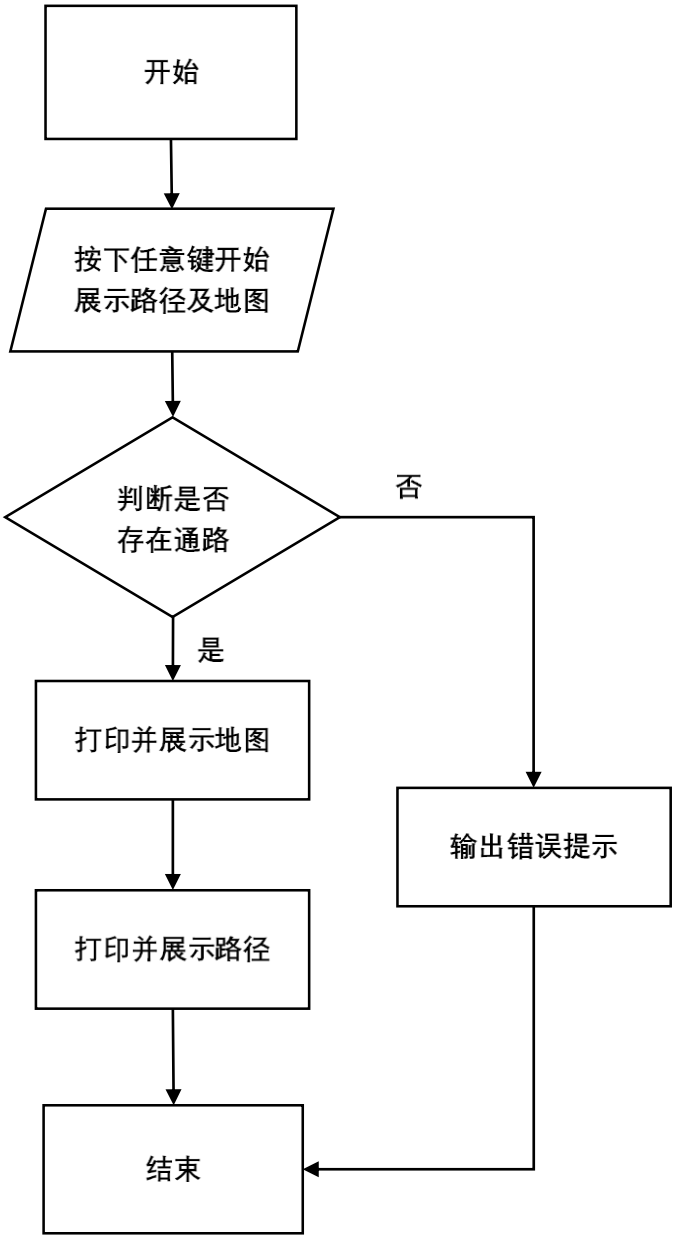
```
void Maze::findPath()
{
    DFS(start);
}

bool Maze::DFS(Point &curPoint)
{
    Point temp;
    path.pushBack(curPoint);
    curSize++;
    if (mazeMap[curPoint.x][curPoint.y] == 'E')
    {
        return true;
    }
    for (int i = 0; i < 4; i++)
    {
        temp.x = curPoint.x + knightMove[i][0];
        temp.y = curPoint.y + knightMove[i][1];
        if (mazeMap[temp.x][temp.y] != '#' && hasPassed[temp.x][temp.y]
 == 0)
        {
            hasPassed[temp.x][temp.y] = 1;
            if (DFS(temp))
            {
                mazeMap[curPoint.x][curPoint.y] = 'X';
                return true;
            }
        }
    }
    curSize--;
    path.popBack();
    return false;
}
```

3.2.3 寻找路径功能截屏示例



3.3 路径展示功能的实现

3.3.1 路径展示功能流程图

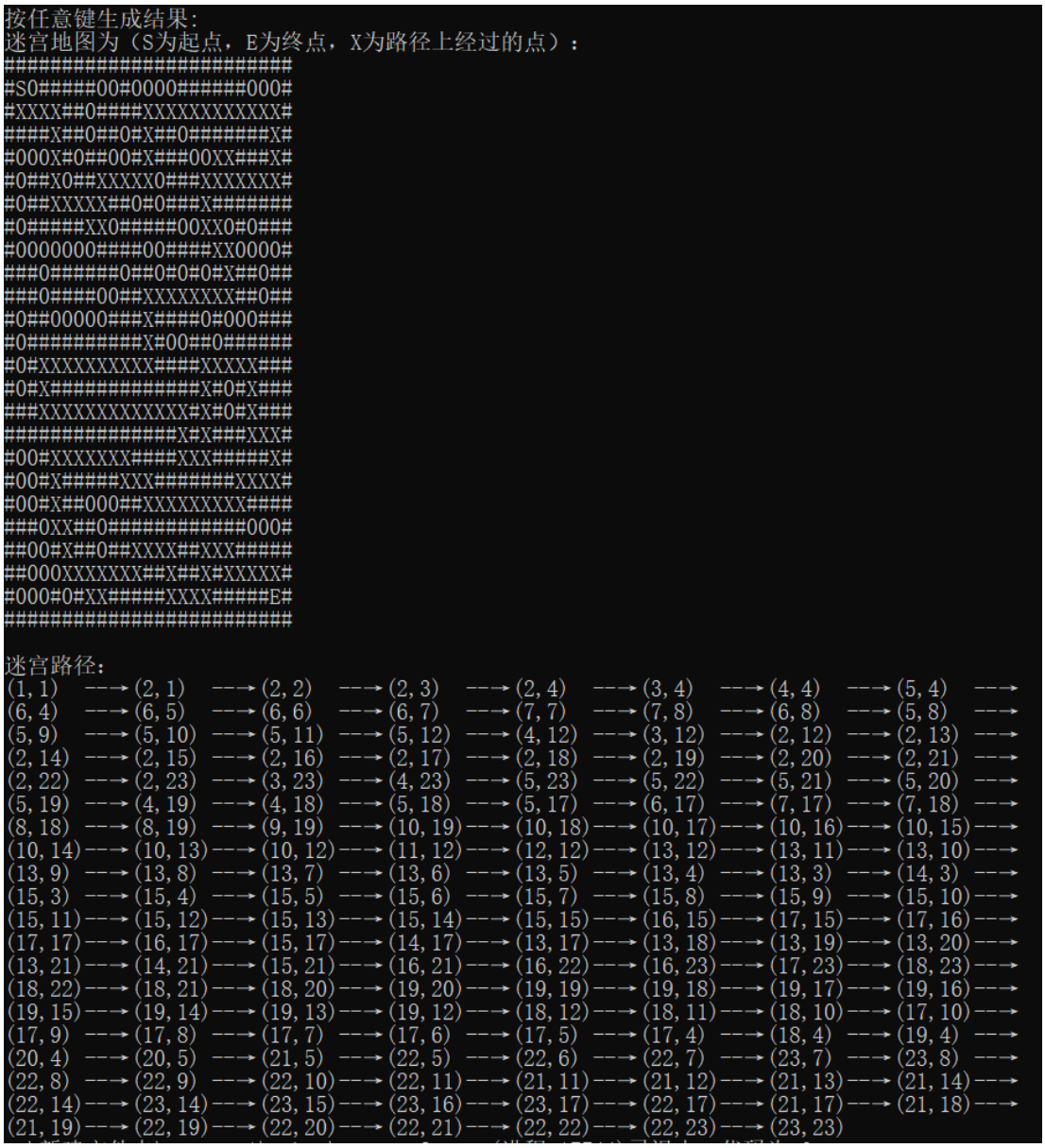


3.3.2 路径展示功能核心代码

Maze 类中：

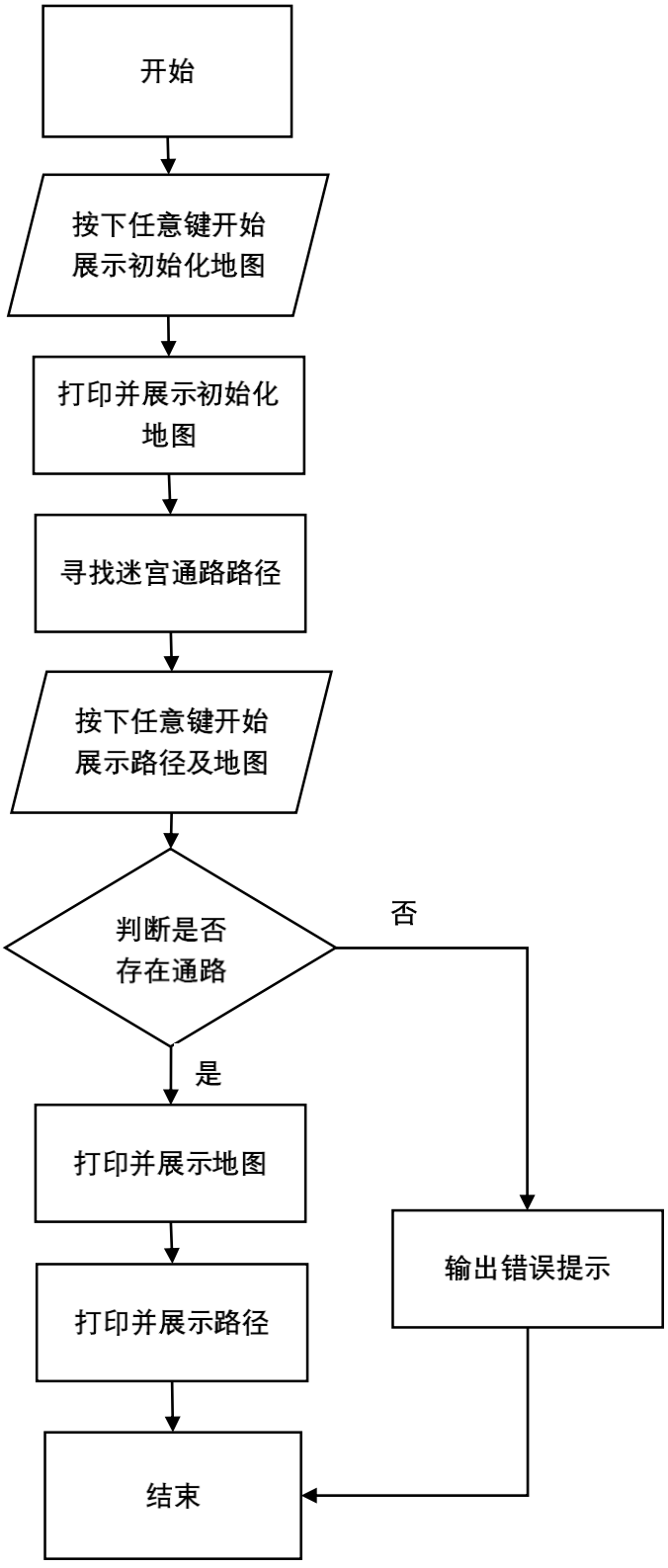
```
void Maze::outputPath()
{
    if (curSize == 0)
    {
        cout << "迷宫的入口与出口不通!" << endl;
        return;
    }
    mazeMap[start.x][start.y] = 'S';
    printMap();
    cout << endl;
    cout << "迷宫路径: " << endl;
    for (int i = 0; i < curSize; i++)
    {
        cout << path[i];
        if (path[i].x < 10)
        {
            cout << ' ';
        }
        if (path[i].y < 10)
        {
            cout << ' ';
        }
        if (i != curSize - 1)
        {
            cout << "-->";
        }
        if ((i + 1) % 8 == 0)
        {
            cout << '\n';
        }
    }
}
```

3.3.3 路径展示功能截屏示例



3.4 总体功能的实现

3.4.1 总体功能流程图



3.4.2 总体功能核心代码

```
int main()
{
    string input;
    Maze map(25);
    cout << "按任意键出现地图:";
    getline(cin, input);
    map.printMap();
    map.findPath();
    cout << "按任意键生成结果:";
    getline(cin, input);
    map.outputPath();
    return 0;
}
```

3.4.3 总体功能截屏示例

```
按任意键出现地图:
迷宫地图为（S为起点，E为终点，X为路径上经过的点）：
##########################
#S0#####00#0000######000#
#0000##0####000000000000#
####0##0##0#0##0#######0#
#0000#0##00#0###0000###0#
#0##00##000000###0000000#
#0##00000##0#0###0#######
#0#####000#####00000#0###
#0000000####00####000000#
###0######0##0#0#0#0##0##
###0####00##00000000##0##
#0##00000###0####0#000###
#0##########0#00##0######
#0#0000000000####00000###
#0#0#############0#0#0###
###0000000000000#0#0#0###
###############0#0###000#
#00#0000000####000#####0#
#00#0#####000#######0000#
#00#0##000##000000000####
###000##0############000#
##00#0##0##0000##000#####
##0000000000##0##0#00000#
#000#0#00#####0000#####E#
##########################
按任意键生成结果:
迷宫地图为（S为起点，E为终点，X为路径上经过的点）：
##########################
#S0#####00#0000######000#
#XXXX##0####XXXXXXXXXXX#
####X##0##0#X##0#######X#
#000X#0##00#X###00XX###X#
#0##X0##XXXXX0###XXXXXXX#
#0##XXXXX##0#0###X#######
#0#####XX0#####00XX0#0###
#0000000####00####XX0000#
###0######0##0#0#0#X##0##
###0####00##XXXXXXXX##0##
#0##00000###X####0#000###
#0##########X#00##0######
#0#XXXXXXXXXX####XXXXX###
#0#X#############X#0#X###
###XXXXXXXXXXXXX#X#0#X###
###############X#X###XXX#
#00#XXXXXXX####XXX#####X#
#00#X#####XXX#######XXXX#
#00#X##000##XXXXXXXXX####
###0XX##0############000#
##00#X##0##XXXX##XXX#####
##000XXXXXXX##X##X#XXXXX#
#000#0#XX#####XXXX#####E#
##########################

迷宫路径:
(1,1) --->(2,1) --->(2,2) --->(2,3) --->(2,4) --->(3,4) --->(4,4) --->(5,4) --->
(6,4) --->(6,5) --->(6,6) --->(6,7) --->(7,7) --->(7,8) --->(6,8) --->(5,8) --->
(5,9) --->(5,10) --->(5,11) --->(5,12) --->(4,12) --->(3,12) --->(2,12) --->(2,13) --->
(2,14) --->(2,15) --->(2,16) --->(2,17) --->(2,18) --->(2,19) --->(2,20) --->(2,21) --->
(2,22) --->(2,23) --->(3,23) --->(4,23) --->(5,23) --->(5,22) --->(5,21) --->(5,20) --->
(5,19) --->(4,19) --->(4,18) --->(5,18) --->(5,17) --->(6,17) --->(7,17) --->(7,18) --->
(8,18) --->(8,19) --->(9,19) --->(10,19)--->(10,18)--->(10,17)--->(10,16)--->(10,15)--->
(10,14)--->(10,13)--->(10,12)--->(11,12)--->(12,12)--->(13,12)--->(13,11)--->(13,10)--->
(13,9) --->(13,8) --->(13,7) --->(13,6) --->(13,5) --->(13,4) --->(13,3) --->(14,3) --->
(15,3) --->(15,4) --->(15,5) --->(15,6) --->(15,7) --->(15,8) --->(15,9) --->(15,10)--->
(15,11)--->(15,12)--->(15,13)--->(15,14)--->(15,15)--->(16,15)--->(17,15)--->(17,16)--->
(17,17)--->(16,17)--->(15,17)--->(14,17)--->(13,17)--->(13,18)--->(13,19)--->(13,20)--->
(13,21)--->(14,21)--->(15,21)--->(16,21)--->(16,22)--->(16,23)--->(17,23)--->(18,23)--->
(18,22)--->(18,21)--->(18,20)--->(19,20)--->(19,19)--->(19,18)--->(19,17)--->(19,16)--->
(19,15)--->(19,14)--->(19,13)--->(19,12)--->(18,12)--->(18,11)--->(18,10)--->(17,10)--->
(17,9) --->(17,8) --->(17,7) --->(17,6) --->(17,5) --->(17,4) --->(18,4) --->(19,4) --->
(20,4) --->(20,5) --->(21,5) --->(22,5) --->(22,6) --->(22,7) --->(23,7) --->(23,8) --->
(22,8) --->(22,9) --->(22,10)--->(22,11)--->(21,11)--->(21,12)--->(21,13)--->(21,14)--->
(22,14)--->(23,14)--->(23,15)--->(23,16)--->(23,17)--->(22,17)--->(21,17)--->(21,18)--->
(21,19)--->(22,19)--->(22,20)--->(22,21)--->(22,22)--->(22,23)--->(23,23)
```

4 测试

4.1 功能测试

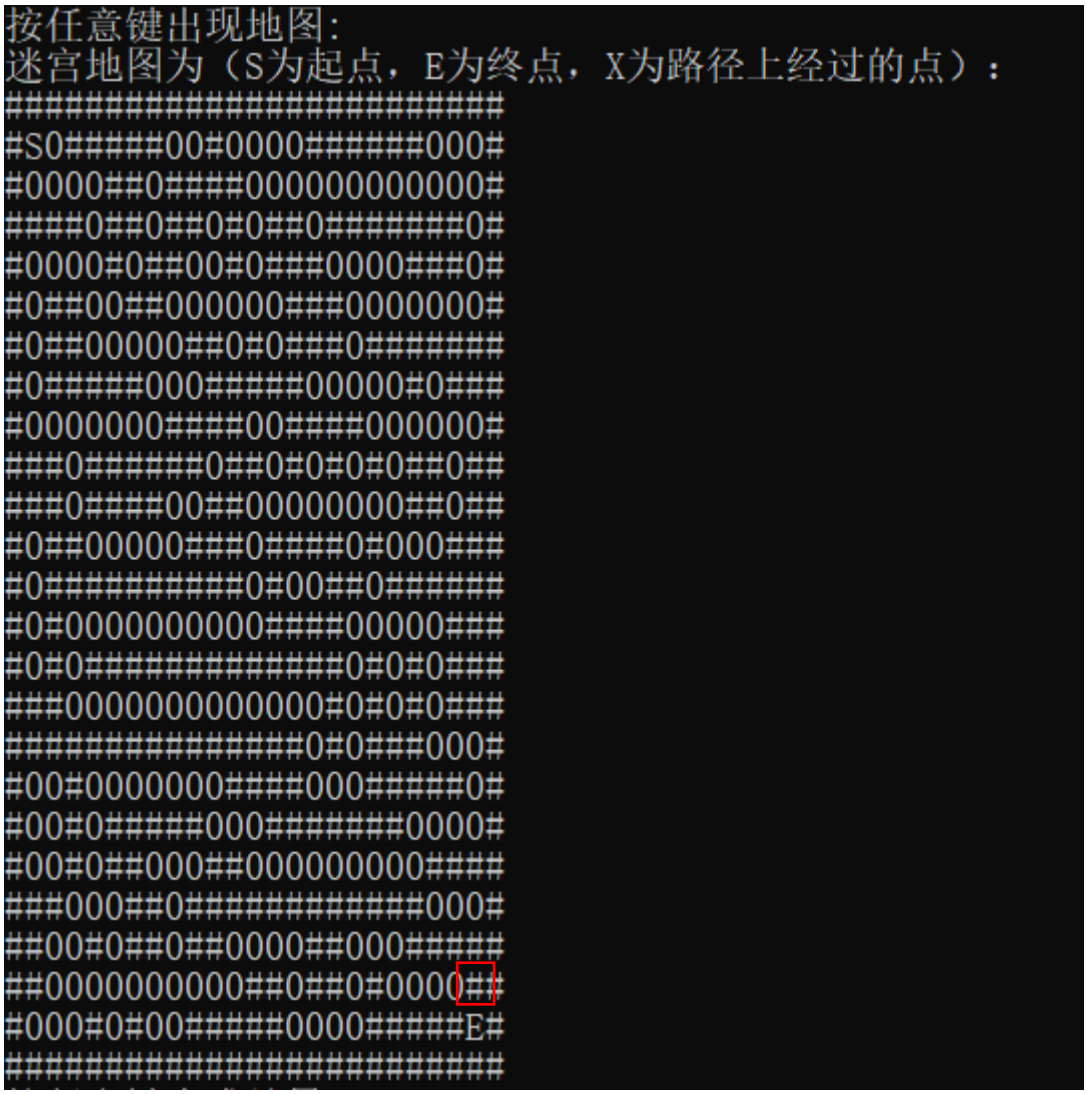
4.1.1 寻路功能测试

测试用例：初始地图
预期结果：正常寻路并给出结果
实验结果：

```
按任意键生成结果:
迷宫地图为（S为起点，E为终点，X为路径上经过的点）：
##########################
#S0#####00#0000######000#
#XXXX##0####XXXXXXXXXXXX#
####X##0##0#X##0#######X#
#000X#0##00#X###00XX###X#
#0##X0##XXXXX0###XXXXXXX#
#0##XXXXX##0#0###X#######
#0#####XX0#####00XX0#0###
#0000000####00####XX0000#
###0######0##0#0#0#X##0##
###0####00##XXXXXXXX##0##
#0##00000###X####0#000###
#0##########X#00##0######
#0#XXXXXXXXX####XXXXX####
#0#X##############X#0#X###
###XXXXXXXXXXXXX#X#0#X###
################X#X###XXX#
#00#XXXXXXX####XXX#####X#
#00#X#####XXX#######XXXX#
#00#X##000##XXXXXXXXX####
###0XX##0############000#
##00#X##0##XXXX##XXX#####
##000XXXXXXX##X##X#XXXXX#
#000#0#XX#####XXXX#####E#
##########################

迷宫路径:
(1,1) --->(2,1) --->(2,2) --->(2,3) --->(2,4) --->(3,4) --->(4,4) --->(5,4) --->
(6,4) --->(6,5) --->(6,6) --->(6,7) --->(7,7) --->(7,8) --->(6,8) --->(5,8) --->
(5,9) --->(5,10) --->(5,11) --->(5,12) --->(4,12) --->(3,12) --->(2,12) --->(2,13) --->
(2,14) --->(2,15) --->(2,16) --->(2,17) --->(2,18) --->(2,19) --->(2,20) --->(2,21) --->
(2,22) --->(2,23) --->(3,23) --->(4,23) --->(5,23) --->(5,22) --->(5,21) --->(5,20) --->
(5,19) --->(4,19) --->(4,18) --->(5,18) --->(5,17) --->(6,17) --->(7,17) --->(7,18) --->
(8,18) --->(8,19) --->(9,19) --->(10,19)--->(10,18)--->(10,17)--->(10,16)--->(10,15)--->
(10,14)--->(10,13)--->(10,12)--->(11,12)--->(12,12)--->(13,12)--->(13,11)--->(13,10)--->
(13,9) --->(13,8) --->(13,7) --->(13,6) --->(13,5) --->(13,4) --->(13,3) --->(14,3) --->
(15,3) --->(15,4) --->(15,5) --->(15,6) --->(15,7) --->(15,8) --->(15,9) --->(15,10)--->
(15,11)--->(15,12)--->(15,13)--->(15,14)--->(15,15)--->(16,15)--->(17,15)--->(17,16)--->
(17,17)--->(16,17)--->(15,17)--->(14,17)--->(13,17)--->(13,18)--->(13,19)--->(13,20)--->
(13,21)--->(14,21)--->(15,21)--->(16,21)--->(16,22)--->(16,23)--->(17,23)--->(18,23)--->
(18,22)--->(18,21)--->(18,20)--->(19,20)--->(19,19)--->(19,18)--->(19,17)--->(19,16)--->
(19,15)--->(19,14)--->(19,13)--->(19,12)--->(18,12)--->(18,11)--->(18,10)--->(17,10)--->
(17,9) --->(17,8) --->(17,7) --->(17,6) --->(17,5) --->(17,4) --->(18,4) --->(19,4) --->
(20,4) --->(20,5) --->(21,5) --->(22,5) --->(22,6) --->(22,7) --->(23,7) --->(23,8) --->
(22,8) --->(22,9) --->(22,10)--->(22,11)--->(21,11)--->(21,12)--->(21,13)--->(21,14)--->
(22,14)--->(23,14)--->(23,15)--->(23,16)--->(23,17)--->(22,17)--->(21,17)--->(21,18)--->
(21,19)--->(22,19)--->(22,20)--->(22,21)--->(22,22)--->(22,23)--->(23,23)
```

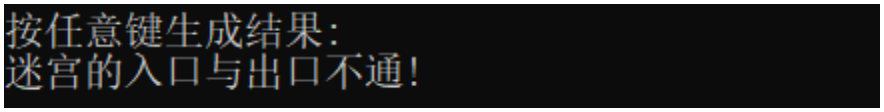
4.1.2 迷宫不通功能测试

测试用例：如图，使得迷宫不通



预期结果：输出迷宫不通的提示，程序不崩溃

实验结果：



4.1.3 按任意键功能测试

测试用例：输入“wasd　输入测试　1234　”
预期结果：程序正常运行不崩溃
实验结果：

```
按任意键出现地图:wasd  输入测试  1234
迷宫地图为（S为起点，E为终点，X为路径上经过的点）：
#########################
#S0#####00#0000######000#
#0000##0####000000000000#
####0##0##0#0##0#######0#
#0000#0##00#0###0000###0#
#0##00##000000###0000000#
#0##00000##0#0###0#######
#0#####000#####00000#0###
#0000000####00####000000#
###0######0##0#0#0#0#0##0#
###0####00##00000000##0##
#0##00000###0####0#000###
#0##########0#00##0######
#0#0000000000####00000###
#0#0##############0#0#0###
###0000000000000#0#0#0###
###############0#0###000#
#00#0000000####000#####0#
#00#0#####000#######0000#
#00#0##000##0000000000####
###000##0############000#
##00#0##0#0000##000#####
##0000000000##0##0#00000#
#000#0#00#####0000#####E#
#########################
按任意键生成结果:wasd  输入测试  1234
迷宫地图为（S为起点，E为终点，X为路径上经过的点）：
#########################
#S0#####00#0000######000#
#XXXX##0####XXXXXXXXXXX#
####X##0##0#X##0#######X#
#000X#0##00#X###00XX###X#
#0##X0##XXXXX0###XXXXXXX#
#0##XXXXX##0#0###X#######
#0#####XX0#####00XX0#0###
#0000000####00####XX0000#
###0######0##0#0#0#X##0##
###0####00##XXXXXXXX##0##
#0##00000###X####0#000###
#0##########X#00##0######
#0#XXXXXXXXX####XXXXX###
#0#X##############X#0#X###
###XXXXXXXXXXXXX#X#0#X###
################X#X###XXX#
#00#XXXXXXX####XXX#####X#
#00#X#####XXX#######XXXX#
#00#X##000##XXXXXXXXX####
###0XX##0############000#
##00#X##0##XXXX##XXX#####
##000XXXXXXX##X##X#XXXXX#
#000#0#XX#####XXXX#####E#
#########################

迷宫路径：
(1,1) --->(2,1) --->(2,2) --->(2,3) --->(2,4) --->(3,4) --->(4,4) --->(5,4) --->
(6,4) --->(6,5) --->(6,6) --->(6,7) --->(7,7) --->(7,8) --->(6,8) --->(5,8) --->
(5,9) --->(5,10) --->(5,11) --->(5,12) --->(4,12) --->(3,12) --->(2,12) --->(2,13) --->
(2,14) --->(2,15) --->(2,16) --->(2,17) --->(2,18) --->(2,19) --->(2,20) --->(2,21) --->
(2,22) --->(2,23) --->(3,23) --->(4,23) --->(5,23) --->(5,22) --->(5,21) --->(5,20) --->
(5,19) --->(4,19) --->(4,18) --->(5,18) --->(5,17) --->(6,17) --->(7,17) --->(7,18) --->
(8,18) --->(8,19) --->(9,19) --->(10,19)--->(10,18)--->(10,17)--->(10,16)--->(10,15)--->
(10,14)--->(10,13)--->(10,12)--->(11,12)--->(12,12)--->(13,12)--->(13,11)--->(13,10)--->
(13,9) --->(13,8) --->(13,7) --->(13,6) --->(13,5) --->(13,4) --->(13,3) --->(14,3) --->
(15,3) --->(15,4) --->(15,5) --->(15,6) --->(15,7) --->(15,8) --->(15,9) --->(15,10)--->
(15,11)--->(15,12)--->(15,13)--->(15,14)--->(15,15)--->(16,15)--->(17,15)--->(17,16)--->
(17,17)--->(16,17)--->(15,17)--->(14,17)--->(13,17)--->(13,18)--->(13,19)--->(13,20)--->
(13,21)--->(14,21)--->(15,21)--->(16,21)--->(16,22)--->(16,23)--->(17,23)--->(18,23)--->
(18,22)--->(18,21)--->(18,20)--->(19,20)--->(19,19)--->(19,18)--->(19,17)--->(19,16)--->
(19,15)--->(19,14)--->(19,13)--->(19,12)--->(18,12)--->(18,11)--->(18,10)--->(17,10)--->
(17,9) --->(17,8) --->(17,7) --->(17,6) --->(17,5) --->(17,4) --->(18,4) --->(19,4) --->
(20,4) --->(20,5) --->(21,5) --->(22,5) --->(22,6) --->(22,7) --->(23,7) --->(23,8) --->
(22,8) --->(22,9) --->(22,10)--->(22,11)--->(21,11)--->(21,12)--->(21,13)--->(21,14)--->
(22,14)--->(23,14)--->(23,15)--->(23,16)--->(23,17)--->(22,17)--->(21,17)--->(21,18)--->
(21,19)--->(22,19)--->(22,20)--->(22,21)--->(22,22)--->(22,23)--->(23,23)
```