项目说明文档

数据结构课程设计

——勇闯迷宫游戏

作 者 姓 名：杨鑫

学　　号：1950787

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 -
1.1 背景分析 - 1 -
1.2 功能分析 - 1 -
2 设计 - 1 -
2.1 数据结构设计 - 1 -
2.2 类结构设计 - 1 -
2.3 成员与操作设计 - 2 -
2.4 系统设计 - 3 -
3 实现 - 4 -
3.1 初始化迷宫功能的实现 - 4 -
3.1.1 初始化迷宫功能流程图 - 4 -
3.1.2 初始化迷宫功能核心代码 - 4 -
3.1.3 初始化迷宫功能截屏示例 - 5 -
3.2 寻找路径功能的实现 - 5 -
3.2.1 寻找路径功能流程图 - 5 -
3.2.2 寻找路径功能核心代码 - 5 -
3.2.3 寻找路径功能截屏示例 - 6 -
4 测试 - 7 -
4.1 功能测试 - 7 -
4.1.1 寻路功能测试 - 7 -
4.1.2 按其他键功能测试 - 7 -

1 分析

1.1 背景分析

迷宫类型的游戏的大意是，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。本程序就是基于此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，这里在游戏开始前就设置了一张迷宫地图，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

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，在寻找路径的时候可能会需要不断的添加新节点和删除之前的节点。这里采用回溯法寻找路径，即需要一个栈来保存实时的路径。

所以，本程序主体上使用栈来完成核心操作。其中将其设计为带有前驱和后驱指针的栈，同时栈中保存了头指针和尾指针。这样的目的是为了在之后输出路径时更加的方便。对于传统的栈，如果要输出路径，应当要将栈中元素全部弹出才能找到第一个节点（即起始位置），这时就需要一个额外的数组来存储弹出的元素，同时遍历的时间也会成为两倍。而在本程序中，仅仅是增加了一个后驱指针和一个尾指针，这样的空间开销肯定小于额外开一个数组来存放所有信息带来的开销，同时遍历时也只需要遍历一次，时间不会有额外的开销。

2.2 类结构设计

首先有一个记录坐标的位置信息结构体（Position）。为了实现栈，还得有一个节点类（Node），栈类（Stack）。此外，还有一个迷宫类（Maze），它整合了用户交互和相应的一些操作。

2.3 成员与操作设计

位置信息结构体（Position）：

```
int x;
int y;
Position(int xNew, int yNew) :x(xNew), y(yNew) {} // 带参数的构造函数
Position() :x(0), y(0) {} // 默认构造函数
Position(const Position& P) :x(P.x), y(P.y) {} // 复制构造函数

// 重载 + 运算
Position& operator + (const Position& P);
// 重载 - 运算
Position& operator - (const Position& P);
// 重载 == 运算
bool operator == (const Position& P);
// 重载 << 运算符，实现坐标输出
friend ostream& operator << (ostream& out, const Position& P);
```

节点类（Node）：

```
T data;
Node<T>* link;
Node<T>* pre;
Node() :link(NULL), pre(NULL) {} // 默认构造函数
Node(T& x, Node<T>* l = NULL, Node<T>* p = NULL) :data(x), link(l), pre(p) {} // 带参数的构造函数
friend Stack<T>; // 声明 Stack 为友类
```

栈类（Stack）：

私有成员：

```
1.Node<T>* top; // 头指针
2.Node<T>* bottom; // 尾指针
3.void makeEmpty(); // 私有函数， 清空栈
```

公有操作：

```
Stack() : top(NULL), bottom(NULL) {} // 默认构造函数
~Stack() { makeEmpty(); } // 析构函数
void Push(const T& x); // 入栈
bool Pop(T& x); // 出栈
void getTop(T& x); // 获取栈顶元素
bool IsEmpty() const { return top == NULL; } // 判断栈是否空
int getSize() const; // 获取栈的大小
void Print() const; // 顺序打印路径
```

迷宫类（Maze）：

私有成员：

```
// 初始迷宫图
char map[25][25] = {
    "####################",
    "#S00########0#######",
    "##0#####0###000######",
"##00#################",
    "#00##0000#000000####",
    "#0#000##0#0####0####",
    "#000####000####0####",
    "##0##########00000##",
    "##0##########0##0####",
    "###0#00####0000###00##",
    "#00#0#####0##########",
    "#0000#####0##00000###",
    "#########0000####0###",
    "###0#######0####0000#",
    "##00####000########0#",
    "###0####0##########0#",
    "###0##000#####00000#",
    "################0#####",
    "################00000#",
    "####################"
};
Stack<Position> mazeStack; // 路径栈
```

公有操作：

```
Maze() {} // 默认构造函数
void PrintMap(); // 打印带正确路径的迷宫图
void PrintPath(); // 打印详细路径
void GeneratePath(); // 生成路径
```

2.4 系统设计

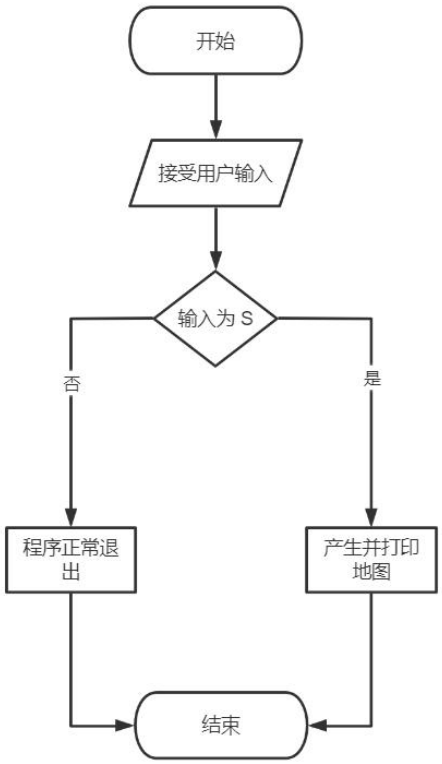
开始游戏后，首先按下 S 键来显示初始化的地图，其中 S 代表起点，0 代表可以走的点，#代表不可以走的点。随后用户再次按下 S 键后会搜索通路路径，并将路径存放于栈中并将其打印出来，向用户展示迷宫的长度以及路径。

程序兼容了 windows 和 LINUX 平台，在双平台下均可以正常运行。

3 实现

3.1 初始化迷宫功能的实现

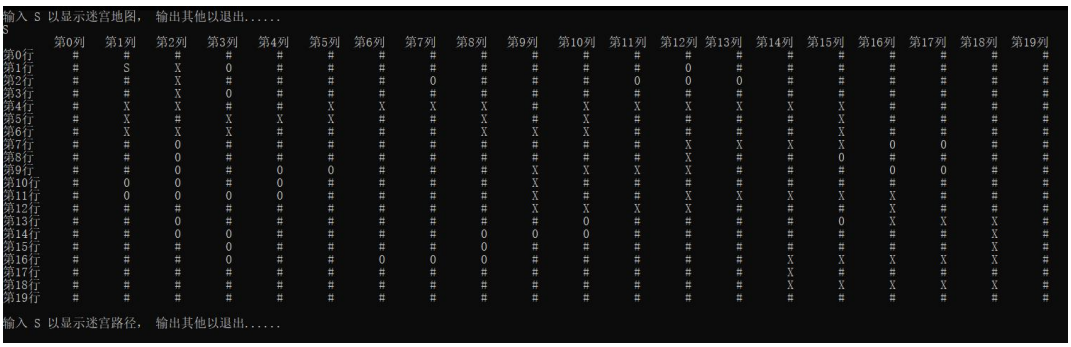
3.1.1 初始化迷宫功能流程图



3.1.2 初始化迷宫功能核心代码

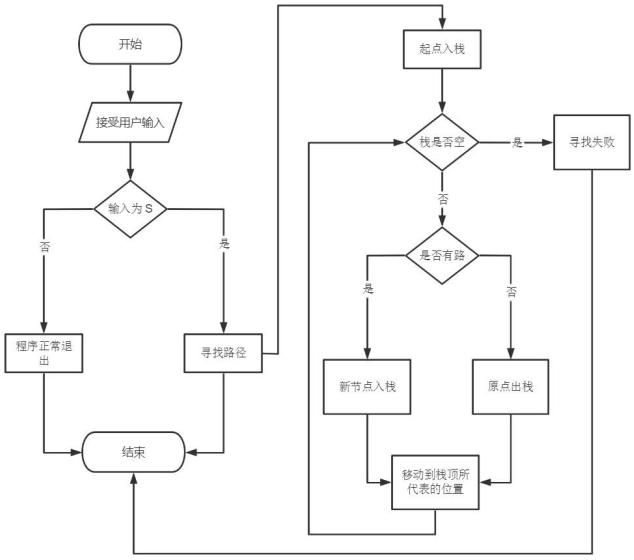
```
cout << "\t";
for (int i = 0; i < 20; i++) cout << "第" << i << "列" << '\t';
// 遍历迷宫图数组并按格式打印
for (int i = 0; i < 20; i++) {
    cout << endl;
    cout << "第" << i << "行" << '\t';
    for (int j = 0; j < 20; j++) {
        cout << setw(4) << map[i][j] << "    ";
    }
}
cout << endl << endl;
```

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

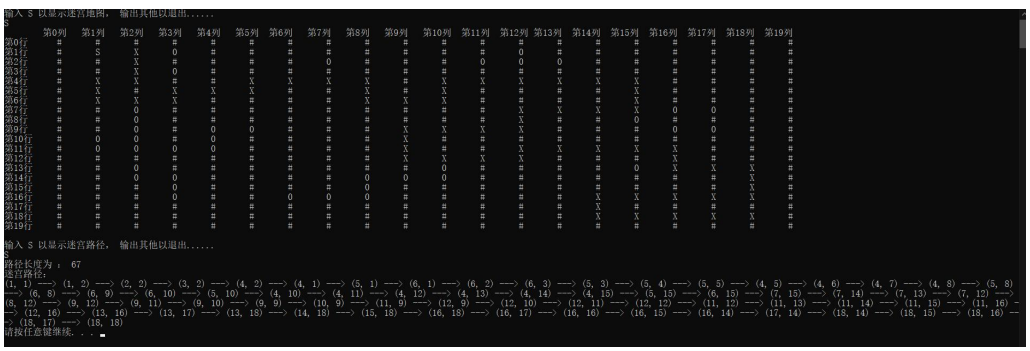
```
1.  // 建立一个方位数组，代表前进的四个方向
2.  Position up(-1, 0), down(1, 0), left(0, -1), right(0, 1);
3.  Position direction[4] = { up, down, left, right };
4.
5.  bool temMap[25][25]; // 标记数组，用于记录某位置是否被遍历过
6.  for (int i = 0; i < 25; i++)
7.      for (int j = 0; j < 25; j++) temMap[i][j] = 1; // 初始化标记数组
8.  Position cur(1, 1), end(18,18); // 记录当前位置和终点位置
```

```
    mazeStack.Push(cur); // 先将起点入栈

    // 当前位置不等于终点（找到出口）且路径栈不为空（无路至终点）时继续往下搜寻
    while (!mazeStack.IsEmpty() && !(cur == end)) {
        bool tag = 0; // 是否能从当前位置往下走的标记
        int i;

        // 四个方向依次搜寻
        for (i = 0; i < 4; i++) {
            cur = cur + direction[i];
            if (map[cur.x][cur.y] == '0' && temMap[cur.x][cur.y]) { // 检测欲移动到的位置是否为障碍物或已经被走过
                mazeStack.Push(cur);
                map[cur.x][cur.y] = 'X'; // 在迷宫图中标记为走过的点（若之后回退，可取消标记）
                temMap[cur.x][cur.y] = 0; // 标记已走过（不会取消标记）
                tag = 1; // 标记为 1
                break;
            }
            cur = cur - direction[i]; // 无路继续往下走
        }
        if (!tag) {
            if (i >= 4) { // 所有方向尝试完均不能走通，回退该点并从路径栈中弹出该点
                map[cur.x][cur.y] = '0';
                mazeStack.Pop(cur);
            }
            mazeStack.getTop(cur); // 尝试栈顶点的其他方向（之前试过的路已被标记而无法再走）
        }
    }
```

3.2.3 寻找路径功能截屏示例

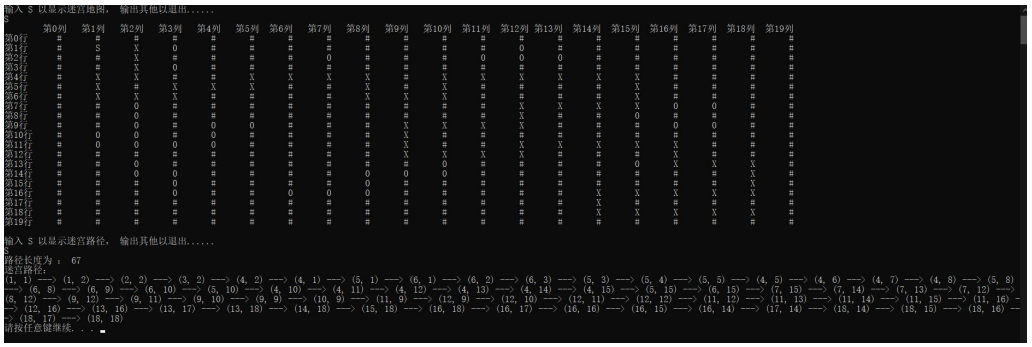


4 测试

4.1 功能测试

4.1.1 寻路功能测试

测试用例：启动程序后按照提示按下两次 S 键
预期结果：正常寻路并给出结果
实验结果：



4.1.2 按其他键功能测试

测试用例：启动程序后按下其他按键
预期结果：程序正常退出，不会产生异常
实验结果：