****

项目说明文档

**数据结构课程设计**

**——勇闯迷宫游戏**

**培养单位：软件学院**

**本 科 生：蓝 笙 聆**

**学 号：1951096**

**指导老师：张 颖**

二○二○年十二月

目录

[第1章 功能分析 1](#_Toc59827254)

[第2章 设计 2](#_Toc59827255)

[2.1 数据结构设计 2](#_Toc59827256)

[2.2 类结构设计 2](#_Toc59827257)

[2.3 成员与操作设计 2](#_Toc59827258)

[2.4 系统设计 3](#_Toc59827259)

[第3章 实现 4](#_Toc59827260)

[3.1 DFS搜索功能的实现 4](#_Toc59827261)

[3.1.1 DFS搜索功能流程图 4](#_Toc59827262)

[3.1.2 DFS搜索功能核心代码 5](#_Toc59827263)

[3.1.3 DFS搜索功能截屏示例 6](#_Toc59827264)

[3.2 总体系统的实现 7](#_Toc59827265)

[3.2.1 总体系统流程图 7](#_Toc59827266)

[3.2.2 总体系统代码实现 7](#_Toc59827267)

[3.2.3 总体功能截屏示例 9](#_Toc59827268)

[第4章 测试 10](#_Toc59827269)

[4.1 功能测试 10](#_Toc59827270)

[4.1.1 DFS搜索功能测试 10](#_Toc59827271)

[4.2 边界测试 10](#_Toc59827272)

[4.2.1 多路径测试 10](#_Toc59827273)

[4.3 出错测试 11](#_Toc59827274)

[4.3.1 输入不存在的地图 11](#_Toc59827275)

# 第1章 功能分析

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。

说明：将迷宫写死在代码当中，使用一定算法返回迷宫从起点走到终点的路径。

输出说明：输出迷宫路线以及坐标。

# 第2章 设计

## 2.1 数据结构设计

如上所述，创建Maze类，使用二维vector存储迷宫一个迷宫，使用vector<pair<int, int>>存储路径。若有多个迷宫则可以使用三位vector进行存储。

## 2.2 类结构设计

为了保存地图，使用二维vector；而为了保存多个地图，使用三维vector。为了保存路径，使用vector<pair<int, int>>；而为了保存多个路径，使用vector<vector<pair<int, int>>>。

## 2.3 成员与操作设计

**迷宫类（Maze）**

class Maze {

   private:

    int \_step[4][2] = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};  //枚举的每一个方向

    std::vector<std::vector<std::vector<int>>> \_maps;  //保存多个地图

    std::vector<std::vector<std::pair<int, int>>> \_dir\_all;  //保存多个路径

    std::vector<std::pair<int, int>> \_dirv;                  //暂存路径

    int \_map = 0;      //选择的第几个地图

    int \_row = 0;      //行

    int \_col = 0;      //列

    int \_start\_x = 1;  //起点横坐标

    int \_start\_y = 1;  //起点纵坐标

    int \_end\_x = 0;    //终点横坐标

    int \_end\_y = 0;    //终点纵坐标

   public:

    Maze();

    std::vector<std::vector<int>> Copy();       //拷贝一份棋盘

    void Dfs(int now\_x, int now\_y, int depth);  //深度优先搜索主体

    void PrintAns(std::ostream& os);            //输出路径与地图

    void PrintMaps(std::ostream& os);           //输出所有的原始地图

    void SelectMap(int id);                     //选择地图

    int GetStartX() const { return this->\_start\_x; }    //返回起点横坐标

    int GetStartY() const { return this->\_start\_y; }    //返回起点纵坐标

    int GetMaps() const { return this->\_maps.size(); }  //返回地图总数

};

## 2.4 系统设计

系统首先调用opening ()函数实现对屏幕的初始化并选择地图，使用solution ()函数开始执行搜索，使用ending()函数输出地图结果。

# 第3章 实现

## 3.1 DFS搜索功能的实现

### 3.1.1 DFS搜索功能流程图

### 3.1.2 DFS搜索功能核心代码

void Maze::Dfs(int now\_x, int now\_y, int depth) {

    if (now\_x == \_end\_x && now\_y == \_end\_y) {  //如果到达终点

        std::vector<std::pair<int, int>> ans;

        for (int i = 0; i < depth; i++) {

            ans.push\_back(\_dirv[i]);

        }

        \_dir\_all.push\_back(ans);        //将路径保存并结束该层搜索

        return;

    }

    \_maps[\_map][now\_x][now\_y] = 1;             //标记该点已被访问

    for (int i = 0; i < 4; i++) {

        int next\_x = now\_x + \_step[i][0];

        int next\_y = now\_y + \_step[i][1];  //枚举4个方向

        if (\_maps[\_map][next\_x][next\_y] != 0)

            continue;                  //若该点不可到达则下一个方向

        if (\_dirv.size() < depth + 1)  //将该点坐标暂存

            \_dirv.push\_back({next\_x, next\_y});

        else

            \_dirv[depth] = {next\_x, next\_y};

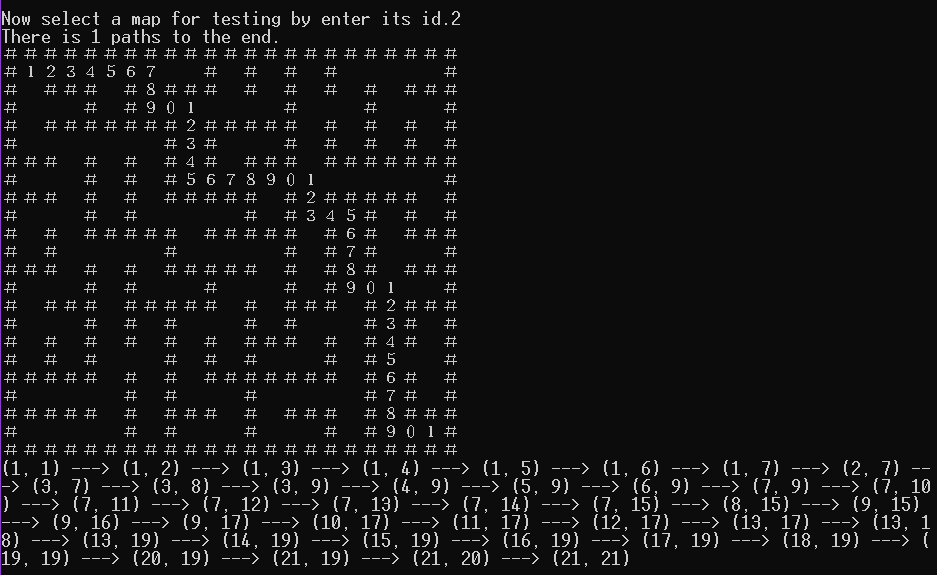
        Dfs(next\_x, next\_y, depth + 1);  //传入该点坐标递归进行搜索

    }

    \_maps[\_map][now\_x][now\_y] = 0;  //取消该点标记

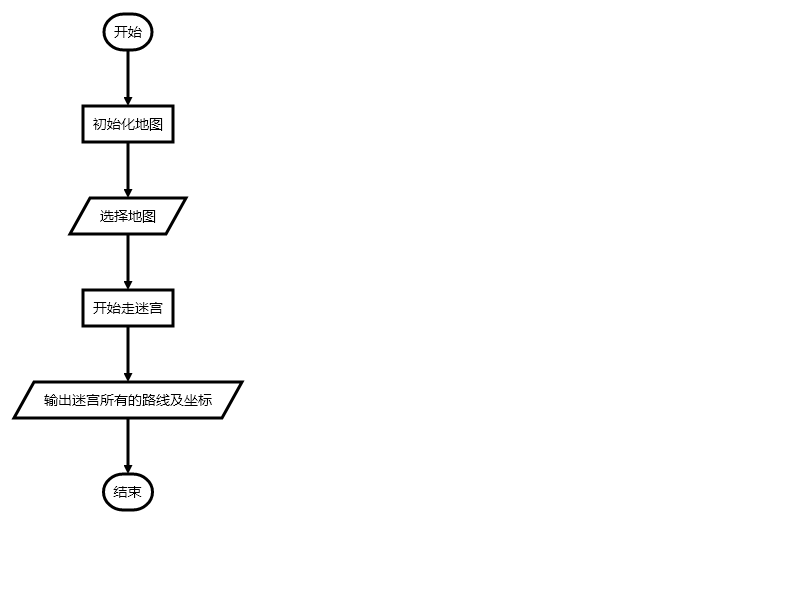
}

### 3.1.3 DFS搜索功能截屏示例



## 3.2 总体系统的实现

### 3.2.1 总体系统流程图



### 3.2.2 总体系统代码实现

void opening(Maze& m) {

    std::cout << "Maze Game" << std::endl

              << "Automatically generate mazes first." << std::endl

              << "Here are " << m.GetMaps() << " maps to choose from"

              << std::endl;

    m.PrintMaps(std::cout);

    std::cout << std::endl;

    std::cout << "Now select a map for testing by enter its id.";

    int temp = 0;

    while (true) {

        std::cin >> temp;

        try {

            m.SelectMap(temp - 1);

            break;

        } catch (const char\* e) {

            std::cerr << e << std::endl;

        }

    }

    return;

}

void solution(Maze& m) { m.Dfs(m.GetStartX(), m.GetStartY(), 1); }

void ending(Maze& m) {

    m.PrintAns(std::cout);  //对地图和走过的路径进行打印

}

bool loop(){

    Maze m;

    opening(m);

    solution(m);

    ending(m);

    std::cout << "Again? (y for yes, n for no)";

    char c;

    std::cin >> c;

    if (c == 'y')

        return true;

    else

        return false;

}

int main() {

    while (loop())

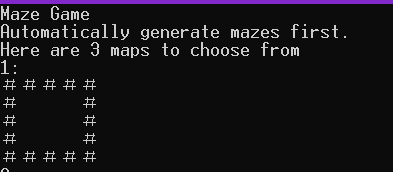
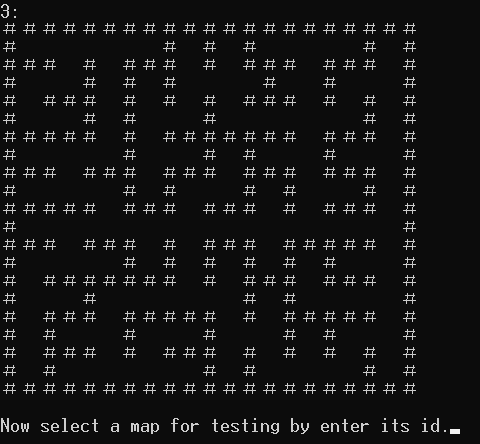
        ;

    std::cout << "Thanks for using it. See you next time! " << std::endl;

    return 0;

}

### 3.2.3 总体功能截屏示例



# 第4章 测试

## 4.1 功能测试

### 4.1.1 DFS搜索功能测试

**测试用例**：

使用地图2

**预期结果**：

正常运行

**实验结果**



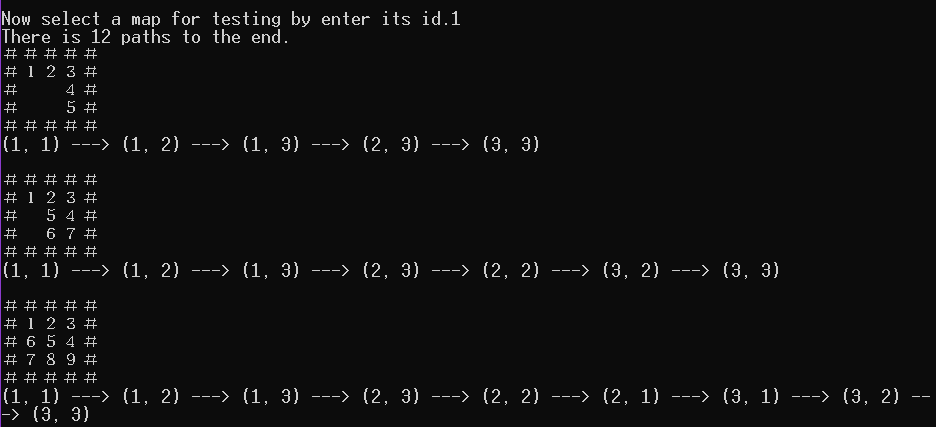
## 4.2 边界测试

### 4.2.1 多路径测试

**测试用例：**使用地图1

**预期结果：**应有12条路线。

**实验结果：**



（后略）

## 4.3 出错测试

### 4.3.1 输入不存在的地图

**测试用例：**选择除了提供的1、2、3以外的地图。

**预期结果：**抛出异常，程序要求重新输入，不崩溃。

**实验结果：**

