**项目说明文档**

**数据结构课程设计**

**——勇闯迷宫游戏**

作 者 姓 名 刘淑仪

学 号 2251730

指 导 教 师 张 颖

学 院 专 业 软件学院 软件工程



二〇二三 年 十 月 二十五 日

目录

1 项目分析 1

1.1 项目背景分析 1

1.2 项目功能分析 1

2 项目设计 2

2.1 数据结构设计 2

2.2 类结构设计 2

2.3 成员与操作设计 3

2.4 系统设计 4

3 功能设计与项目实现 4

3.1 输入迷宫高度宽度功能的实现 4

3.1.1 输入迷宫高度宽度功能实现思路 4

3.1.2 输入迷宫高度宽度功能核心代码 4

3.1.3 输入迷宫高度宽度功能实现示例 5

3.2 迷宫生成与输出的实现 5

3.2.1 迷宫生成与输出实现思路 5

3.2.2 迷宫生成与输出核心代码 5

3.2.3 迷宫生成与输出实现示例 7

3.3 深度优先搜索寻路算法的实现 7

3.3.1 深度优先搜索寻路算法实现思路 7

3.3.2 深度优先搜索寻路算法核心代码 8

3.3.3 深度优先搜索寻路算法实现示例 8

3.4 路径坐标打印的实现 8

3.4.1 路径坐标打印实现思路 8

3.4.2 路径坐标打印核心代码 9

3.4.3 路径坐标打印实现示例 9

4 项目测试 10

4.1 功能测试 10

4.1.1 基本功能测试 10

4.1.2 边界条件测试 11

4.2 错误测试 13

4.2.1 输入验证错误 13

4.2.1.1 输入验证错误判断思路 13

4.2.1.3输入示例 13

4.2.2 异常情况处理 14

4.3 Linux环境测试 15

5 集成开发环境与编译运行环境 15

# 项目分析

## 项目背景分析

迷宫项目是一种模拟真实世界挑战的有效方法。它不仅是一种受欢迎的娱乐形式，还在计算机科学教育中发挥着关键作用，特别是在算法设计和数据结构领域。随着计算机科学的发展，探索和解决复杂迷宫问题已成为理解和应用高级算法的重要途径。

在计算机和人工智能技术不断进步的背景下，迷宫项目的开发不仅局限于传统算法的应用。它还为研究和实现更复杂的解决策略。迷宫项目对教育有深远的影响，特别是在逻辑思维、问题解决和编程技能的培养方面。同时，它在实际应用中也具有重要价值，例如在游戏开发、机器人导航和路径规划等领域。

而在最基本的迷宫中，如何灵活使用搜索算法成为我们练习的关键。

## 项目功能分析

1. 迷宫生成算法

算法使用递归的方式来生成迷宫。它从一个起始点开始，随机选择方向，打破墙壁，从而逐步形成迷宫的路径。这个算法的主要功能是创建一个复杂且唯一的迷宫，确保迷宫有足够的探索价值。

1. 寻找最短路径

深度优先搜索（DFS）通过深度优先搜索来探索从迷宫入口到出口的路径。它试图访问未被探索的路径，直到找到出口或所有路径都被探索完毕。这个算法使程序能够找到一条从入口到出口的可行路径，如果存在的话。这是解决迷宫的核心功能，同时也是验证迷宫可解性的关键。

1. 迷宫的展示和路径坐标打印

可视化函数负责将迷宫的当前状态以文本形式显示出来，包括墙壁、路径、起点、终点和已探索的路径。在找到路径后，路径跟踪函数将打印出构成路径的坐标序列。

# 项目设计

## 数据结构设计

迷宫项目中使用的是二维数组maze[MAXSIZE][MAXSIZE]来存储迷宫的布局，其中MAXSIZE定义了迷宫的最大尺寸。二维数组天然适合表现迷宫的格局，其中每个元素代表迷宫的一个单元格，可以是墙（WALL）、路径（PATH）或其他标记。这种数据结构使得在任意位置插入或修改迷宫单元格成为可能，进而有效地实现了迷宫生成和解决算法。

为了追踪迷宫中每个点的访问状态，使用了布尔型数组visited[MAXSIZE][MAXSIZE]。这种结构允许程序快速查询任何给定点是否已经被探索，这对于避免重复路径和寻找最短路径至关重要。

在迷宫中，不同类型的元素（如墙壁、路径等）对应不同的状态和含义。使用枚举类型MazeType对这些不同的元素进行分类，不仅使代码更易于理解和维护，而且在功能上提供了强类型检查，减少了错误的可能性。

迷宫中的每一个位置都可以用一个坐标点来表示。定义了一个结构体Point来存储这些坐标。这种方式不仅使代码更加清晰和组织化，而且为处理位置信息提供了一种灵活而准确的方法。

## 类结构设计

Maze类的设计充分体现了迷宫项目的教育和娱乐目的，同时展示了计算机算法的实际应用。该类通过模块化的方法和高效的数据结构实现了用户友好的交互界面和迷宫的生成及解决功能。这种设计使得迷宫项目不仅易于使用和理解，而且在代码的可维护性、扩展性和效率方面表现出色，为解决复杂问题提供了一个坚实的基础。

Maze类结构设计：用于存储迷宫的状态和实现其基本功能。

在主要cpp文件（假设为MainMazeGame.cpp）中：

创建MazeGame类（如果需要）：用于存储Maze实例和游戏相关的参数作为私有成员，例如迷宫的尺寸或其他游戏设置。公有成员有游戏流程控制函数：用于初始化游戏、开始游戏循环、处理用户输入和展示游戏结果。

## 成员与操作设计

未注释的函数请见下文实现对应功能时代码的函数标签。

·存储枚举类型

enum MazeType {

WALL = 0,

PATH,

START,

END,

WAY,

};

·表示坐标点结构体

struct Point {

int x, y;

Point() : x(0), y(0) {}

Point(int \_x, int \_y) : x(\_x), y(\_y) {}

};

·迷宫的类定义：

class Maze {

private:

int MazeX = MAXSIZE; // 迷宫宽度（列数）

int MazeY = MAXSIZE; // 迷宫高度（行数)

int pathLength = 0;

bool visited[MAXSIZE][MAXSIZE] = { 0 };

MazeType maze[MAXSIZE][MAXSIZE];

Point start, end;

public:

void initMaze();

void menu();

void input(int& inputNum);

void outputMaze();

void generateMazeRecursive(int x, int y);

int findShortestPathDFS(int x, int y);

void printPathCoordinates();

bool isValid(int x, int y) {

return x > 0 && x < MazeX + 2 && y > 0 && y < MazeY + 2;

}

void Exit() {

std::cout << "按任意键继续...\n";

// 清除输入缓冲区

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

// 等待用户按键

std::cin.get();

}

};

## 系统设计

模块化：代码被组织成不同的函数，每个函数负责特定的任务，如初始化、生成迷宫、搜索路径和用户交互，使得代码易于理解和维护。

算法驱动：迷宫的生成和解决过程完全依赖于实现的算法，展示了计算机科学在解决实际问题中的应用。

用户友好性：虽然用户界面简单，但提供了足够的信息和反馈，使用户可以轻松地与系统交互。

可扩展性：系统设计允许在未来对迷宫的功能进行扩展，如添加新的迷宫生成算法、优化路径搜索算法或改进用户界面。

# 功能设计与项目实现

## 输入迷宫高度宽度功能的实现

### 输入迷宫高度宽度功能实现思路

该函数用于处理用户输入的迷宫尺寸。它确保用户输入一个有效的、奇数的尺寸值，且该值不超过预设的最大尺寸（MAXSIZE）。实现思路是创建一个循环来不断请求用户输入，直到输入符合要求（即是一个有效的、在指定范围内的奇数）。它通过多重检查确保输入的有效性，并在发现任何不合规的输入时提供反馈并请求重新输入。

### 输入迷宫高度宽度功能核心代码

void Maze::input(int& inputNum) {

while (1) {

std::cin >> inputNum;

if (std::cin.fail() || inputNum <= 0 || inputNum > MAXSIZE || inputNum % 2 == 0) {

std::cin.clear();

std::cin.ignore(65536, '\n');

std::cout << "输入数据有误，请重新输入\n";

}

else {

if (std::cin.get() == '\n')

break;

else {

std::cin.clear();

std::cin.ignore(65536, '\n');

std::cout << "输入数据有误，请重新输入\n";

}

}

}

### 输入迷宫高度宽度功能实现示例



## 迷宫生成与输出的实现

### 迷宫生成与输出实现思路

四个参数的输入均有Input函数控制，Input函数的形参作为需要输入的参数的引用，通过bool型变量opt控制输入的最大值（除了第一个人数输入的上限为int型上限，后面三个输入都不能超过最大人数）。通过cin输入参数，如果输入错误或输入的数字不在合法范围内，或者一次性输入多个数据，都会报错。直到读取正确的数据且下一个输入的字符为回车才算输入成功。

### 迷宫生成与输出核心代码

void Maze::generateMazeRecursive(int x, int y)

{

// 定义方向数组，顺序随机打乱

int directions[4] = { 1, 2, 3, 4 };

// 随机打乱方向数组

for (int i = 0; i < 4; i++) {

int j = std::rand() % 4;

std::swap(directions[i], directions[j]);

}

for (int i = 0; i < 4; i++) {

int dir = directions[i];

int x\_next = x, y\_next = y;

if (dir == 1)

x\_next = x + 2;

else if (dir == 2)

x\_next = x - 2;

else if (dir == 3)

y\_next = y + 2;

else if (dir == 4)

y\_next = y - 2;

if (x\_next > 0 && x\_next < MazeX + 2 && y\_next > 0 && y\_next < MazeY + 2) {

if (maze[y\_next][x\_next] == WALL) {

maze[y\_next][x\_next] = PATH;

maze[y\_next - (y\_next - y) / 2][x\_next - (x\_next - x) / 2] = PATH;

generateMazeRecursive(x\_next, y\_next);

}

}

}

}

void Maze::outputMaze()

{

maze[end.y][end.x] = END;

maze[1][1] = START;

for (int i = 0; i < MazeY + 2; i++) {

for (int j = 0; j < MazeX + 2; j++) {

if (maze[i][j] == WALL)

std::cout << "■";

else if (maze[i][j] == PATH)

std::cout << " ";

else if (maze[i][j] == START)

std::cout << "始";

else if (maze[i][j] == END)

std::cout << "终";

else if (maze[i][j] == WAY)

std::cout << "·";

}

std::cout << "\n";

}

}

### 迷宫生成与输出实现示例



## 深度优先搜索寻路算法的实现

### 3.3.1 深度优先搜索寻路算法实现思路

检查当前位置是否是迷宫的终点（end.x 和 end.y）。如果是，返回true，表示找到了一条路径。使用一个循环来探索当前位置的所有可能方向（上下左右）。这是通过修改当前坐标x和y来实现的，使用预定义的数组dx和dy来表示方向的变化。

对于每个新的位置（nx, ny），使用isValid函数检查该位置是否在迷宫内并且是一条未被走过的路径（即maze[ny][nx] == PATH）。

如果新位置有效，先将当前位置标记为路径的一部分（maze[y][x] = WAY）。然后递归调用findShortestPathDFS来探索从新位置到终点的路径。

如果从新位置开始的递归调用找到了一条路径，那么当前位置是正确的路径的一部分，函数返回true。如果从当前位置出发无法找到一条通往终点的路径（即递归调用返回false），需要进行回溯。这是通过将当前位置重新标记为PATH（即未走过的状态）来实现的。如果所有方向都探索完毕且没有找到路径，则函数返回false。

它通过递归地探索所有可能的路径，并在找到通往终点的路径时立即返回成功。它利用了递归的自然回溯机制来撤销错误的路径选择。这种算法适用于在迷宫这类问题中寻找路径，尽管它不一定总能找到最短的路径，但它能有效地找到一条通往终点的路径。

### 3.3.2 深度优先搜索寻路算法核心代码

int Maze::findShortestPathDFS(int x, int y) {

if (x == end.x && y == end.y) {

return 1; // 找到终点，返回1表示成功

}

for (int i = 0; i < 4; i++) {

int nx = x + dx[i];

int ny = y + dy[i];

if (isValid(nx, ny) && maze[ny][nx] == PATH) {

maze[y][x] = WAY; // Mark the current point as part of the path

if (findShortestPathDFS(nx, ny)) {

return true;

}

maze[y][x] = PATH; // Unmark the current point if no path is found

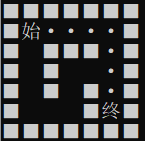
}

}

return false;

}

### 3.3.3 深度优先搜索寻路算法实现示例



## 路径坐标打印的实现

### 3.4.1 路径坐标打印实现思路

输出提示信息，表示开始打印路径坐标。定义一个布尔变量flag，用于控制循环的结束。

使用两层嵌套循环（外层循环遍历行y，内层循环遍历列x），从(1, 1)开始，到(MazeY + 1, MazeX + 1)结束，遍历迷宫的每个格子。在每个格子中，检查是否该格子是路径的一部分，即maze[y][x] == WAY。

如果找到路径上的一个点，打印该点的坐标，格式为(x, y)。检查当前点是否紧邻终点（即是否是路径上的最后一个点）。这通过比较当前点的坐标与终点坐标来实现。如果是，则设置flag为0，并退出内层循环。

在每个路径点后面打印一个箭头->，表示路径的方向和连接。这个箭头在路径的最后一个点之前出现。如果flag被设置为0（意味着已经到达或超过了路径的终点），则退出外层循环。打印一个换行符，表示路径坐标打印完毕。

### 3.4.2 路径坐标打印核心代码

void Maze::printPathCoordinates() {

std::cout << "路径坐标：" << std::endl;

bool flag = 1;

for (int y = 1; y < MazeY + 2; y++) {

for (int x = 1; x < MazeX + 2; x++) {

if (maze[y][x] == WAY) {

std::cout << " (" << x << ", " << y << ") ";

if ((x == end.x && y == end.y - 1) || (x == end.x - 1 && y == end.y)) {

std::cout << " ";

flag = 0;

break;

}

else

std::cout << "->";

}

}

if (!flag)

break;

}

std::cout << std::endl;

}

### 3.4.3 路径坐标打印实现示例



# 项目测试

## 功能测试

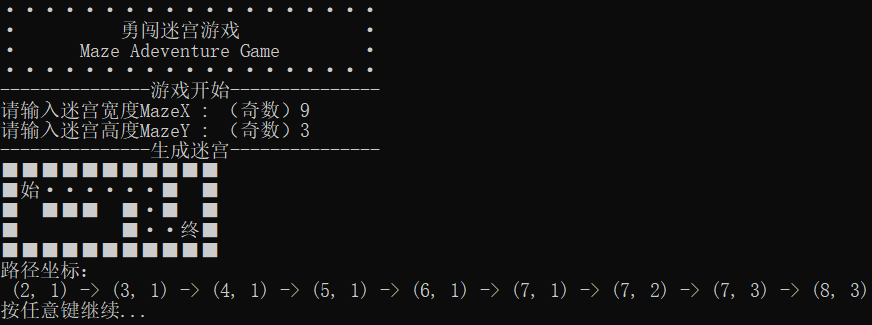
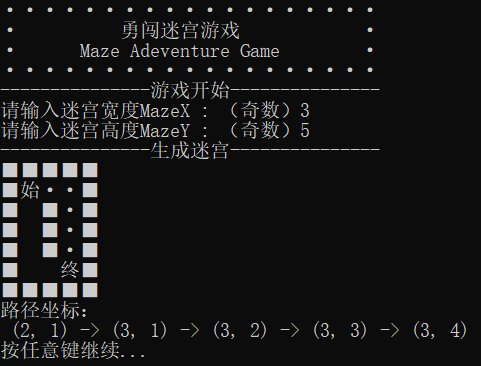
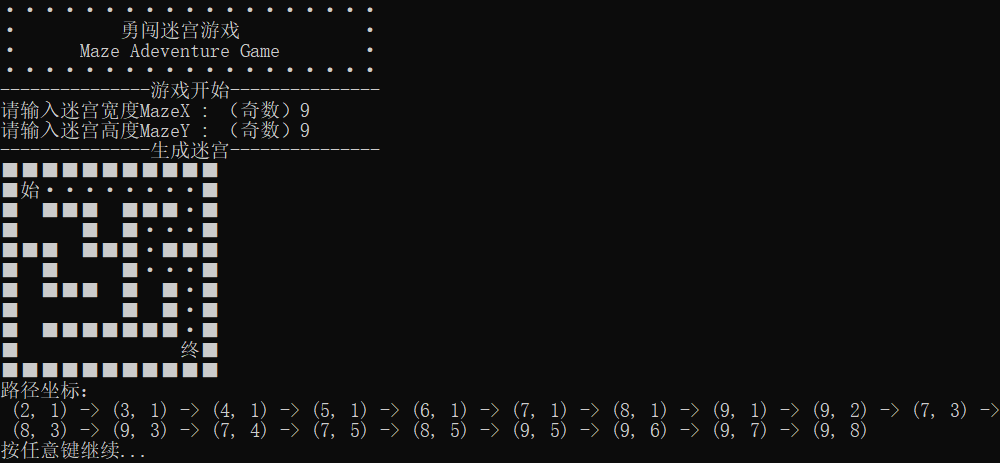
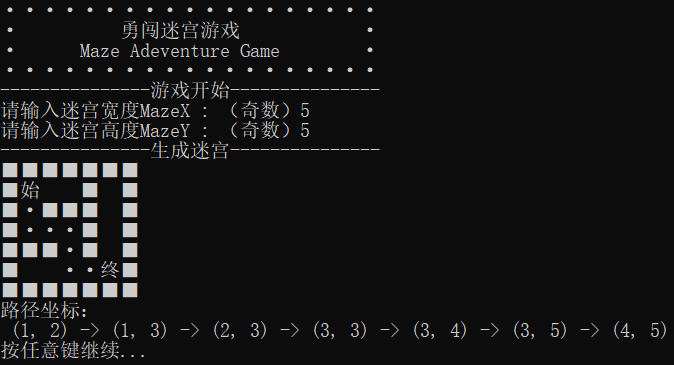
### 基本功能测试

**测试用例：**3 3/5 5/9 9/3 5/9 3

**用例说明：**测试不同大小的迷宫能否正确生成并找到正确的路径

**预期结果：**程序应该正常生成迷宫并找到路径

**实验结果：**



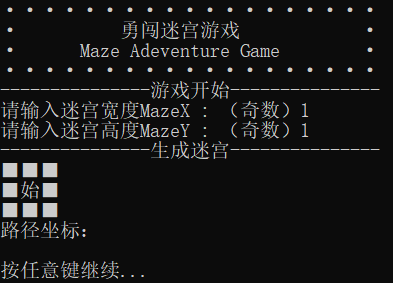
### 4.1.2 边界条件测试

**测试用例：**1 1/MAX\_SIZE MAX\_SIZE/1 MAX\_SIZE

**用例说明：**进行边界测试.

**预期结果：**程序应该在极端情况下也能正确运行，不崩溃

**实验结果：**

**[注：由于屏幕缓冲区的问题，截图没有显示完整，但能过够正确生成迷宫并找到合适路径]**

## 错误测试

### 输入验证错误

#### 4.2.1.1 输入验证错误判断思路

输入非法值（如负数、非数字字符、超出范围的数字）来测试程序对错误输入的处理能力。检查程序是否能妥善处理并提示用户重新输入。

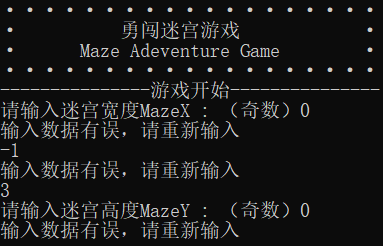
#### 4.2.1.3输入示例

**测试用例：**0 -1/3 0

**用例说明：**测试非正数

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**

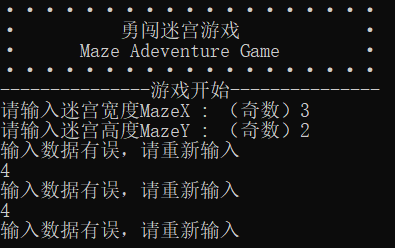


**测试用例：**2 3/4 4

**用例说明：**测试偶数

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**

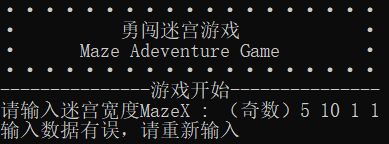


**测试用例：**5 10 1 1；

**用例说明：**测试多个输入

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**

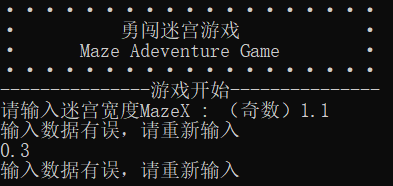


**测试用例：**1.1；

**用例说明：**测试小数；

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**



### 异常情况处理

**测试用例：**abc/ @#￥ / 五

**用例说明：**测试非法输入（如字母）

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**

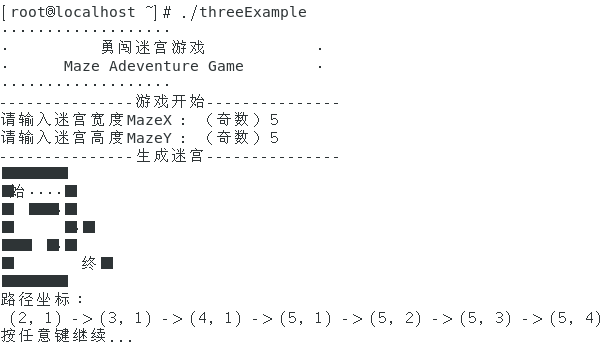


## Linux环境测试

**编译命令：** g++ '/mnt/hgfs/ShareFolder/ 03\_maze\_adventure\_game.cpp' -o threeExample

**运行命令：** ./ threeExample

**实验结果：**



1. **集成开发环境与编译运行环境**

Windows系统：Windows 11 x64

Windows集成开发环境：Microsoft Visual Studio 2022 (Release模式)

Windows编译运行环境：本项目适用于x86架构和x64架构

Linux系统：CentOS 7 x64