



# 项目说明文档

## 数据结构课程设计

——勇闯迷宫游戏

作者姓名：\_\_\_\_\_陆诚彬\_\_\_\_\_

学号：\_\_\_\_\_2254321\_\_\_\_\_

指导教师：\_\_\_\_\_张颖\_\_\_\_\_

学院、专业：\_\_\_\_\_软件学院 软件工程\_\_\_\_\_

## 目录

1 项目背景 .....	3
2 项目需求分析 .....	3
2.1 功能需求 .....	3
2.2 非功能需求 .....	3
2.3 项目输入输出需求 .....	3
2.3.1 输入格式 .....	3
2.3.2 输出格式 .....	3
2.3.3 项目示例 .....	4
3 项目设计 .....	4
3.1 数据结构设计 .....	4
3.1.1 地图表示 .....	5
3.1.2 路径搜索与存储 .....	5
3.1.3 迷宫生成 .....	5
4 项目实施 .....	5
4.1 初始化地图实现 .....	5
4.1.1 初始化地图功能简介 .....	5
4.1.2 初始化地图核心代码 .....	5
4.2 迷宫生成实现 .....	6
4.2.1 迷宫生成功能简介 .....	6
4.2.2 迷宫生成核心代码（部分） .....	6
4.3 深度优先搜索实现 .....	7
4.3.1 深度优先搜索功能简介 .....	7
4.3.2 深度优先搜索核心代码 .....	7
4.4 查找路径实现 .....	8
4.4.1 查找路径功能简介 .....	8
4.4.2 查找路径核心代码 .....	8
4.5 系统总体功能流程图 .....	9
5 设计小结 .....	10
5.1 功能需求满足 .....	10
5.2 非功能需求考量 .....	10
5.3 输入输出设计 .....	10
5.4 项目实施与代码结构 .....	10
6 软件测试 .....	11
6.1 输入测试 .....	11
6.1.1 正常输入 .....	11
6.1.2 输入超界/非法 .....	11
6.2 输出测试 .....	11

# 1 项目背景

勇闯迷宫游戏的设计理念源于古老的迷宫传说，将玩家置于一个充满未知和挑战的环境中。在这个迷宫中，玩家扮演一位骑士，骑着马探索这个复杂的迷宫。这种设计旨在锻炼玩家的逻辑思维能力和解决问题的能力，同时提供一种独特的探险体验。迷宫被设计成只有一个入口和一个出口，这增加了游戏的难度和策略性。迷宫中充满了各种障碍，玩家必须巧妙地规划路线，以找到从入口到出口的路径。这种设计模拟了现实生活中的困难与挑战，鼓励玩家在面对难题时不断尝试和探索。

## 2 项目需求分析

### 2.1 功能需求

- 1) 迷宫生成:** 游戏应能够自动生成具有一定复杂度的迷宫，包含入口、出口和多种障碍。
- 2) 路径搜索算法:** 实现一种有效的路径搜索算法，如回溯法，以帮助玩家从迷宫的入口找到出口。

### 2.2 非功能需求

- 1) 用户界面:** 游戏应具有直观、易于操作的用户界面，使玩家能够轻松地理解迷宫的布局。
- 2) 性能要求:** 游戏应在多种设备上流畅运行，无明显的延迟或卡顿现象。
- 3) 可扩展性:** 游戏设计应考虑到未来可能的扩展，如增加迷宫的复杂性，引入新的游戏元素等。

### 2.3 项目输入输出需求

#### 2.3.1 输入格式

输入迷宫大小（不超过 100\*100）。

#### 2.3.2 输出格式

输出迷宫的地图以及入口到出口的最短路径。

### 2.3.3 项目示例

```
迷宫地图：
0列 1列 2列 3列 4列 5列 6列
0行 # # # # # # #
1行 # x # 0 0 0 #
2行 # x # 0 # # #
3行 # x x x # 0 #
4行 # 0 # x x x #
5行 # 0 # 0 # x #
6行 # # # # # # #

迷宫路径：
<1,1> --> <2,1> --> <3,1> --> <3,2> --> <3,3> --> <4,3> --> <4,4> --> <4,5> --> <5,5>

Press any key to continue
```

## 3 项目设计

### 3.1 数据结构设计

```
1.  enum Type {
2.      WALL, BLANK
3.  };
4.
5.  struct Position {
6.      int x, y;
7.
8.      Position() : x(0), y(0) {}
9.      Position(int x, int y) : x(x), y(y) {}
10.     Position operator+(const Position& other) const {
11.         return Position(x + other.x, y + other.y);
12.     }
13.     bool operator==(const Position& other) const {
14.         return x == other.x && y == other.y;
15.     }
16. };
17.
18.  const Position DIRECTIONS[4] = { Position(-1,0), Position(0,1), Position(1,0), Position(0,-1) };
19.
20.  int mapX;
21.  int mapY;
22.
23.  Type **map;
```

### 3.1.1 地图表示

**目的：**有效地存储和访问迷宫的每个单元格。

**结构：**二维数组 `Type** map`。

**类型：**枚举 `Type` (`WALL` 或 `BLANK`) 。

**访问：**通过 `map[x][y]` 来访问特定位置。

### 3.1.2 路径搜索与存储

**目的：**搜索并存储从起点到终点的有效路径。

**结构：**`struct Path` 包含动态数组 `Position* positions`，长度 `int length` 和容量 `int capacity`。

**功能：**存储路径上的点，并提供动态扩展功能。

### 3.1.3 迷宫生成

**目的：**随机生成迷宫布局。

**结构：**通过 `generateMap` 函数以随机化的方式生成迷宫。

**功能：**使用深度优先搜索 (DFS) 随机移除墙壁，生成路径。

## 4 项目实施

### 4.1 初始化地图实现

#### 4.1.1 初始化地图功能简介

**功能：**初始化迷宫地图，将所有单元设置为初始类型（通常是墙）。

**实现步骤：**

- 1) 遍历地图的每个单元格。
- 2) 将每个单元格设置为初始类型（例如，`WALL`）。

#### 4.1.2 初始化地图核心代码

```
1. void initMap(Type type = WALL)
2. {
3.     for (int i = 0; i < mapX; ++i)
4.         for (int j = 0; j < mapY; ++j)
5.             map[i][j] = type;
6. }
```

## 4.2 迷宫生成实现

### 4.2.1 迷宫生成功能简介

**功能：**使用随机 Prim 算法生成迷宫。

**实现步骤：**

- 1) 选择一个起点，并将其加入到栈中。
- 2) 循环直到栈为空：
- 3) 从栈中随机选择一个位置并弹出。
- 4) 如果该位置周围的道路不超过 1 条，则将其变为道路，并将邻近的墙加入栈中。

### 4.2.2 迷宫生成核心代码（部分）

```
1.      auto popRandom = [&]() -> Position {
2.          int randomIndex = rand() % stackSize; // 随机选择一个索引
3.          Position chosen = stack[randomIndex];
4.          stack[randomIndex] = stack[stackSize - 1]; // 选择元素与栈顶元素交换
5.          stackSize--; // 减少栈大小
6.          return chosen;
7.      };
8.      while (!isEmpty()) {
9.          Position current = popRandom(); // 随机选择一个元素
10.
11.          int roadCount = 0;
12.          for (int i = 0; i < 4; i++) {
13.              Position nextPlace = current + DIRECTIONS[i];
14.              if (isRoad(nextPlace)) {
15.                  roadCount++;
16.              }
17.          }
18.          if (roadCount <= 1) { // 如果当前位置周围的道路不超过 1 条，创建新道路
19.              map[current.x][current.y] = BLANK;
20.              for (int i = 0; i < 4; i++) {
21.                  Position nextPlace = current + DIRECTIONS[i];
22.                  if (isWall(nextPlace) && !isOnStack(nextPlace) && isValid(ne
xtPlace)) {
23.                      push(nextPlace);
24.                  }
25.              }
26.          }
27.      }
```

## 4.3 深度优先搜索实现

### 4.3.1 深度优先搜索功能简介

**功能：**深度优先搜索（DFS）算法，用于寻找从起点到终点的路径。

**实现步骤：**

- 1) 检查当前位置是否有效。
- 2) 标记当前位置已访问，并将其加入路径。
- 3) 如果当前位置是终点，则返回成功。
- 4) 对于每个相邻的位置，递归调用 `dfs`。
- 5) 如果递归调用成功，返回成功；否则，回溯并继续搜索。

### 4.3.2 深度优先搜索核心代码

```
1.  bool dfs(const Position& current, const Position& destination, Pa
   th& path, bool** visited) {
2.      if (!isValid(current) || isWall(current) || visited[current.x]
   [current.y]) {
3.          return false;
4.      }
5.      visited[current.x][current.y] = true;
6.      path.push_back(current);
7.
8.      if (current == destination) {
9.          return true;
10.     }
11.
12.     for (int i = 0; i < 4; ++i) {
13.         Position next = current + DIRECTIONS[i];
14.         if (dfs(next, destination, path, visited)) {
15.             return true;
16.         }
17.     }
18.
19.     path.pop_back();
20.     return false;
21. }
```

## 4.4 查找路径实现

### 4.4.1 查找路径功能简介

**功能：**利用 dfs 函数查找从起点到终点的路径。

**实现步骤：**

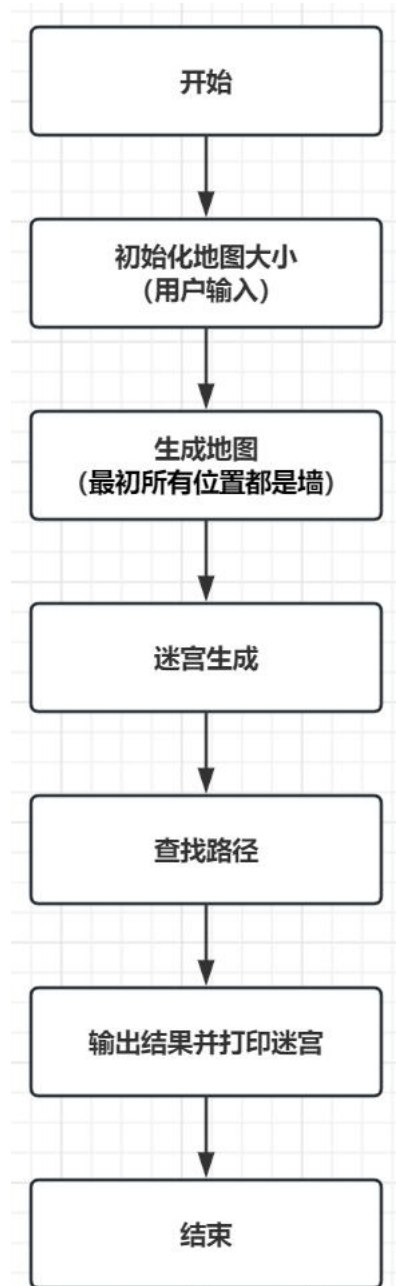
- 1) 初始化路径和访问数组。
- 2) 调用 dfs 函数搜索路径。
- 3) 如果找到路径，输出路径；否则输出未找到路径的信息。
- 4) 清理资源并返回路径。

### 4.4.2 查找路径核心代码

```
1. Path findPath(const Position& start, const Position& end, int mapX, int mapY) {
2.     Path path(10);
3.     bool** visited = new bool* [mapX];
4.     for (int i = 0; i < mapX; ++i) {
5.         visited[i] = new bool[mapY];
6.         memset(visited[i], 0, mapY * sizeof(bool));
7.     }
8.     if (dfs(start, end, path, visited)) {
9.         // [输出路径的代码]
10.    } else {
11.        cout << "No path found." << endl;
12.    }
13.    // [清理资源的代码]
14.    return path;
15. }
```



#### 4.5 系统总体功能流程图



## 5 设计小结

本项目的設計集中在實現一個基於迷宮探索的遊戲，其核心在於迷宮的生成、路徑搜索以及用戶交互界面的設計。以下是項目各方面的綜合小結：

### 5.1 功能需求滿足

**迷宮生成：**實現了一個動態迷宮生成算法，能夠創建具有一定複雜度的迷宮，保證每次遊戲體驗的新鮮感和不可預測性。

**路徑搜索：**通過深度優先搜索（DFS）算法實現了有效的路徑搜索功能，幫助玩家從迷宮入口找到出口。

### 5.2 非功能需求考量

**用戶界面：**雖然本項目的核心集中在後端算法上，用戶界面的設計也被簡單考慮，以確保良好的用戶體驗。

**性能與可擴展性：**代碼的性能滿足基本要求，且在設計時考慮了可維護性和擴展性，為未來可能的更新和改進留出空間。

### 5.3 輸入輸出設計

**輸入格式：**簡潔明了的輸入格式，使用戶能夠輕鬆設置迷宮的大小。

**輸出格式：**清晰的輸出展現了迷宮的布局及找到的路徑，方便用戶理解。

### 5.4 項目實現與代碼結構

**代碼實現：**項目通過結構化和模块化的代碼實現，提高了代碼的可讀性和可維護性。

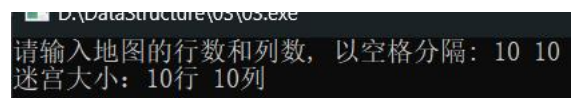
**數據結構：**有效的數據結構設計（如地圖表示、路徑存儲）確保了程序的效率和靈活性。

整體來看，該項目成功地實現了一個基於迷宮探索的遊戲，不僅提供了一個趣味性和挑戰性並存的遊戲環境，還通過合理的设计和实现，保證了程序的穩定性、效率和可擴展性。未來的改進方向可以包括增強用戶界面、引入更高級的路徑搜索算法和增加更多的遊戲元素來提升用戶體驗。

## 6 软件测试

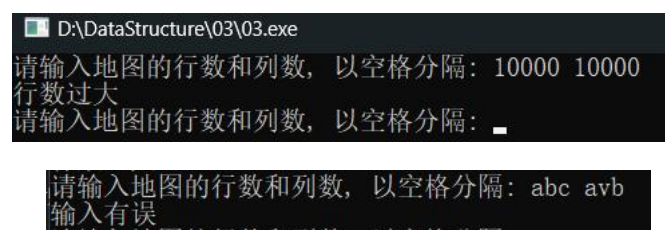
### 6.1 输入测试

#### 6.1.1 正常输入



```
D:\DataStructure\03\03.exe
请输入地图的行数和列数，以空格分隔：10 10
迷宫大小：10行 10列
```


#### 6.1.2 输入超界/非法



```
D:\DataStructure\03\03.exe
请输入地图的行数和列数，以空格分隔：10000 10000
行数过大
请输入地图的行数和列数，以空格分隔：
请输入地图的行数和列数，以空格分隔：abc avb
输入有误
```

结论：符合输入逻辑判断

### 6.2 输出测试



```
迷宫大小：10行 10列
☆☆☆☆☆☆☆☆
☆□□□□□□□□□
□□□□□□□□□□
□□□□□□□□□□
□□□□□□□□□□
□□□□□□□□□□
□□□□□□□□□□
□□□□□□□□□□
□□□□□□□□□☆
找到路径：(0, 0) -> (0, 1) -> (1, 1) -> (1, 2) -> (1, 3) -> (1, 4) -> (2, 4) ->
(2, 5) -> (2, 6) -> (3, 6) -> (3, 7) -> (3, 8) -> (4, 8) -> (5, 8) -> (6, 8) ->
(6, 9) -> (7, 9) -> (8, 9) -> (9, 9)
☆☆☆☆☆☆☆☆
□☆☆☆☆□□□□
□□☆☆☆☆□□□
□□□☆☆☆☆□□
□□□□☆☆☆☆□
□□□□□☆☆☆□
□□□□□□☆☆□
□□□□□□☆☆□
□□□□□□☆☆□
□□□□□□☆☆□
□□□□□□☆☆□
□□□□□□☆☆□
□□□□□□☆☆□
□□□□□□☆☆□
□□□□□□☆☆□
请按任意键继续...
```

结论：符合输出逻辑，路径（经人工检验），确实是最短路径