项目说明文档

数据结构课程设计

——**勇闯迷宫游戏**

作 者 姓 名： 陈翔飞

学 号： 1851756

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 3](#_Toc27433907)

[1.1 项目要求分析 3](#_Toc27433908)

[1.2 功能分析 3](#_Toc27433909)

[2 设计 3](#_Toc27433910)

[2.1 数据结构设计 3](#_Toc27433911)

[2.2 类结构设计 3](#_Toc27433912)

[2.3 成员与操作设计 3](#_Toc27433913)

[2.4 系统设计 7](#_Toc27433914)

[3 实现 7](#_Toc27433915)

[3.1 迷宫展示的实现 7](#_Toc27433916)

[3.1.1迷宫展示核心代码 7](#_Toc27433917)

[3.1.2迷宫展示功能截屏展示 7](#_Toc27433918)

[3.2路径求解的实现 8](#_Toc27433919)

[3.2.1路径求解流程图 8](#_Toc27433920)

[3.2.2 路径求解核心代码 10](#_Toc27433921)

[3.3 路径展示功能的实现 11](#_Toc27433922)

[3.3.1 路径展示功能核心代码 11](#_Toc27433923)

[3.2.2 路径展示功能截屏示例 12](#_Toc27433924)

[3.3 总体系统的实现 14](#_Toc27433925)

[3.3.1 总体系统流程图 14](#_Toc27433926)

[3.3.2 总体系统核心代码 15](#_Toc27433927)

[3.3.3 总体系统截屏示例 16](#_Toc27433928)

# 1 分析

## 项目要求分析

在一个迷宫中有一个入口，有一个出口，从出口开始有很多个可以通过的位置，也有很多障碍， 现在需要从中找出一条通路能够从入口到达出口，走出迷宫。

## 1.2 功能分析

已经有一个地图、起点和终点的位置，需要从一个地图中找到一条从出口到入口的通路，这需要程序能保存路径和进行迷宫搜索。

# 2 设计

## 2.1 数据结构设计

该项目处于方便考虑，已经内置了一张迷宫地图，故迷宫只需要用固定大小的二维字符数组存储即可。其次需要能进行迷宫的搜索工作，本项目采取的是宽度优先搜索（DFS）,不需要额外的数据结构。最后还需要保存路径，由于路径长度无法在求解之前确定，需要动态增加长度，故采用Vector存储。

## 2.2 类结构设计

为了保证设计的数据结构的泛用性，本项目选择将Vector类模板类。为了让模板类能实际应用于本项目，还设计了Point类来储存迷宫中的位置信息。最后将所有功能需求和迷宫打包成一个类Maze(具体在下文介绍)。

## 2.3 成员与操作设计

**向量类（Vector）**

**类定义：**

1. **template**<**typename** ElementType> **class** Vector
2. {
3. **public**:
4. ~Vector<ElementType>();
5. Vector<ElementType>() = **default**;
6. Vector<ElementType>(**const** Vector<ElementType> & v);
7. Vector<ElementType>& operator = (**const** Vector<ElementType>&v);
8. **void** PushBack(**const** ElementType& t);
9. **void** PopBack();
10. **void** Clear();
11. **int** GetSize() **const**;
12. **void** ReSize(**int** NewSize);
13. ElementType& operator[](**int** Index) **const**;
14. **bool** Empty()**const**;
15. **const** ElementType& Back() **const**;
16. **private**:
17. **void** Extend();
18. **int** Size = 0;
19. **int** Capacity = 0;
20. ElementType\* Array = nullptr;
21. };

**私有成员：**

int Size;//Vector中实际储存的元素数量

int Capacity;//Vector已经申请的空间

ElementType\* Array;//储存的数据的起始地址

**私有操作：**

void Extend();

//扩容函数，当容量不足时调用

**公有操作：**

~Vector<ElementType>();

//析构函数，通过调用Clear()来释放内存

Vector<ElementType>() = default;

//默认构造函数

Vector<ElementType>(const Vector<ElementType> & v);

//拷贝构造函数

Vector<ElementType>& operator = (const Vector<ElementType>&v);

//重载=运算符，使该类支持赋值运算

void PushBack(const ElementType& t);

//向Vector末尾添加一个元素

void PopBack();

//删除末尾的元素

void Clear();

//清空Vector，释放内存

int GetSize() const;

//返回储存元素的数量

void ReSize(int NewSize);

//重设Vector的大小

ElementType& operator[](int Index) const;

//重载[]运算符，使Vector可以像数组一样使用

bool Empty()const;

//判断Vector是否为空

const ElementType& Back() const;

//返回末尾的元素

**点类（Point）**

**类定义：**

1. **struct** Point
2. {
3. **int** x = 0;
4. **int** y = 0;
5. Point(**int** x=0, **int** y=0) :x(x), y(y) {};
6. Point& operator =(**const** Point& p)
7. {
8. x = p.x;
9. y = p.y;
10. **return** \***this**;
11. }
12. };

**公有成员：**

int x;//点的横坐标

int y;//点的纵坐标

**公有操作：**

Point(int x=0, int y=0) :x(x), y(y) {};

//带默认参数的构造函数

Point& operator =(const Point& p)

//重载=运算符，使该类支持赋值运算

**迷宫类（Maze）**

**类定义：**

1. **class** Maze
2. {
3. **public**:
4. **void** ShowMap();
5. **void** FindRoad();
6. **void** ShowPath();
7. **private**:

10. **void** InitVis();
11. **bool** DFS(**const** Point& p);
12. **bool** Vis[100][100] = { 0 };
13. **char** MazeMap[100][100] =
14. {
15. {"##########################"},
16. {"#O# ##  #    # #   ##  # #" },
17. {"# #    #  ##     #   #   #"},
18. {"#  # #  ##  ### # ##  # ##" },
19. {"## #  #   #   # # #  #  ##" },
20. {"#  ##  ##   # # #  # ##  #" },
21. {"# #   # ## ##   ## #   # #" },
22. {"#  # #    #  # # #  ## # #" },
23. {"# #  # ##  #  ##   #   # #" },
24. {"#   #    #   #   #  # #  #" },
25. {"# ## ## #X# #  ## # #  # #" },
26. {"##    #   #  # #  #  #   #"},
27. {"#  ##  # #  ##   # #  ## #" },
28. {"# #  #  #  #  # #    #   #" },
29. {"# ##  #   #  #   ## #  ###" },
30. {"#   #  ##  # # ##    # # #" },
31. {"# #  #    #    ## ## #   #" },
32. {"#  #  ####  ###    # ### #" },
33. {"### # #    #    ###    # #" },
34. {"#     # ### # ##    # #  #" },
35. {"# #####   # #   # ##  # ##" },
36. {"# #    ##    ##  ##  #   #" },
37. {"#   ##  ## ##  #   # # # #" },
38. {"# ### #       # ##  #   ##" },
39. {"#      # ## #     #   #  #" },
40. {"##########################" },
41. };
42. **const int** dx[4] = { 0,0,1,-1 };
43. **const int** dy[4] = { 1,-1,0,0 };
44. **int** MapSize = 26;
45. Point Start = Point(1, 1);
46. Vector<Point> PointPath;
47. };

**私有成员：**

bool Vis[100][100];//用于记录迷宫中的点是否访问过

char MazeMap[100][100];//储存迷宫地图

const int dx[4] = { 0,0,1,-1 };//遍历迷宫时，x坐标变化量

const int dy[4] = { 1,-1,0,0 };//遍历迷宫时，y坐标变化量

int MapSize = 26;//迷宫的边长

Point Start = Point(1, 1);//起点

Vector<Point> PointPath;//路径，初始为空

**私有操作：**

void InitVis();

//初始化Vis数组

bool DFS(const Point& p);

//深度搜索函数，从p点作为起点继续搜索迷宫

**公有操作：**

void ShowMap();

//展示迷宫

void FindRoad();

//求解起点到终点的路径

void ShowPath();

//展示路径

## 2.4 系统设计

 打开系统后，系统会向用户展示迷宫，'#'表示障碍，空格表示可行，’O’表示起点，’X’表示终点。然后系统会进行迷宫路径的求解，然后向用户展示可视化的路径，然后从起点向用户展示路径上每一个点的坐标。

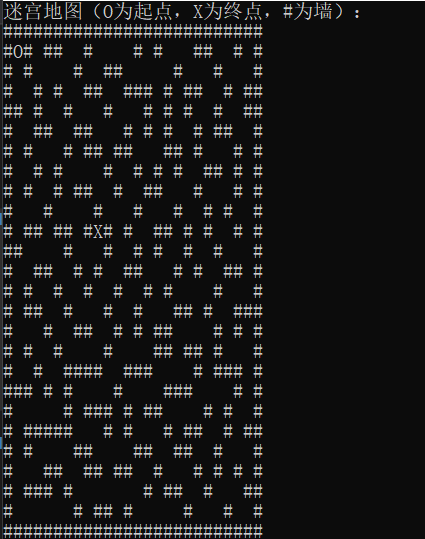
# 3 实现

## 3.1 迷宫展示的实现

### 3.1.1迷宫展示核心代码

1. **inline** **void** Maze::ShowMap()
2. {
3. **int** i;
4. cout << "迷宫地图（O为起点，X为终点，#为墙）：" << endl;
5. **for** (i = 0; i < MapSize; ++i)
6. {
7. cout << MazeMap[i] << endl;
8. }
9. }

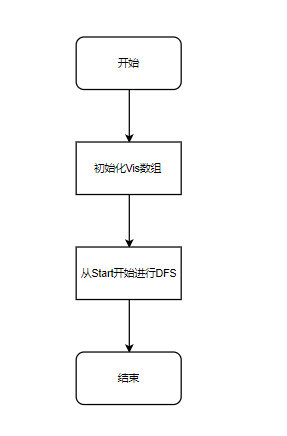
### 3.1.2迷宫展示功能截屏展示



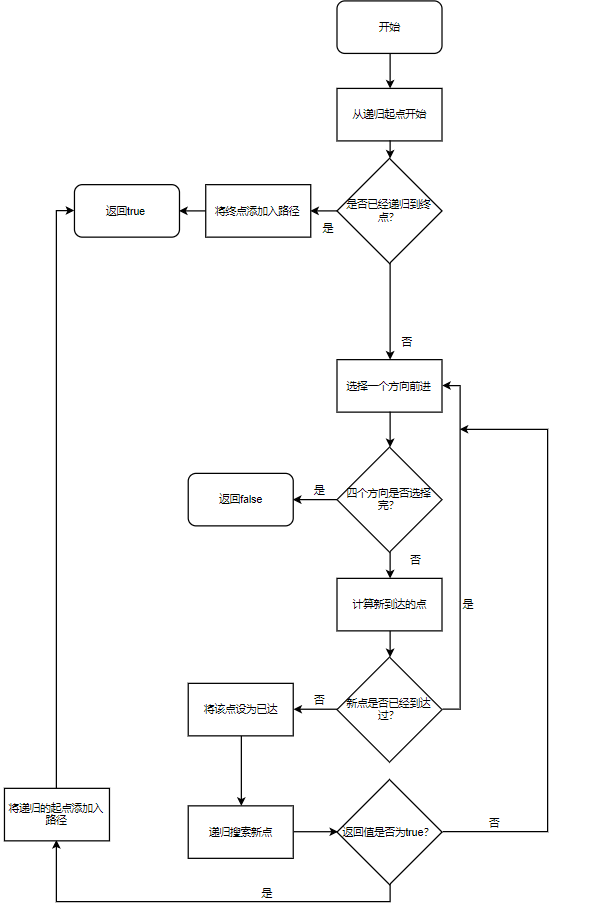
## 3.2路径求解的实现

### 3.2.1路径求解流程图

#### 3.2.1.1 路径求解入口流程图



#### 3.2.1.2 深度优先搜索（DFS）流程图



### 3.2.2 路径求解核心代码

**用户界面接口：**

1. **inline** **void** Maze::FindRoad()
2. {
3. PointPath.Clear();
4. InitVis();
5. DFS(Start);
6. }

**DFS内部实现：**

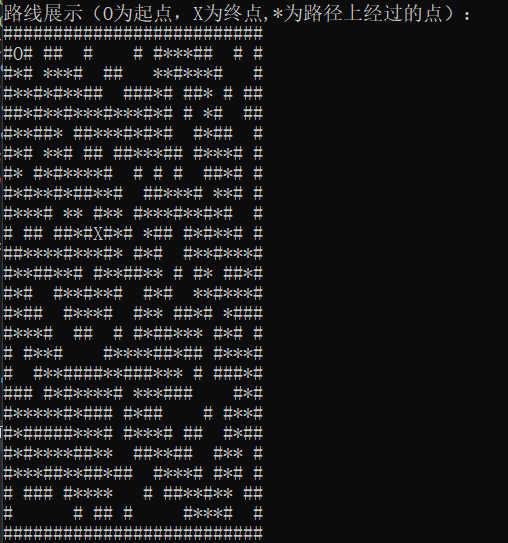
1. **inline** **bool** Maze::DFS(**const** Point & p)
2. {
3. Vis[p.x][p.y] = **true**;
4. **if** (MazeMap[p.x][p.y] == 'X')
5. {
6. PointPath.PushBack(p);
7. **return** **true**;
8. }
9. **int** i;
10. **for** (i = 0; i < 4; ++i)
11. {
12. **int** nx = p.x + dx[i];
13. **int** ny = p.y + dy[i];
14. **if** (Vis[nx][ny] == **true** || MazeMap[nx][ny] == '#')
15. **continue**;
16. **if** (DFS(Point(nx, ny)) == **true**)
17. {
18. PointPath.PushBack(p);
19. **return** **true**;
20. }
21. }
22. **return** **false**;
23. }

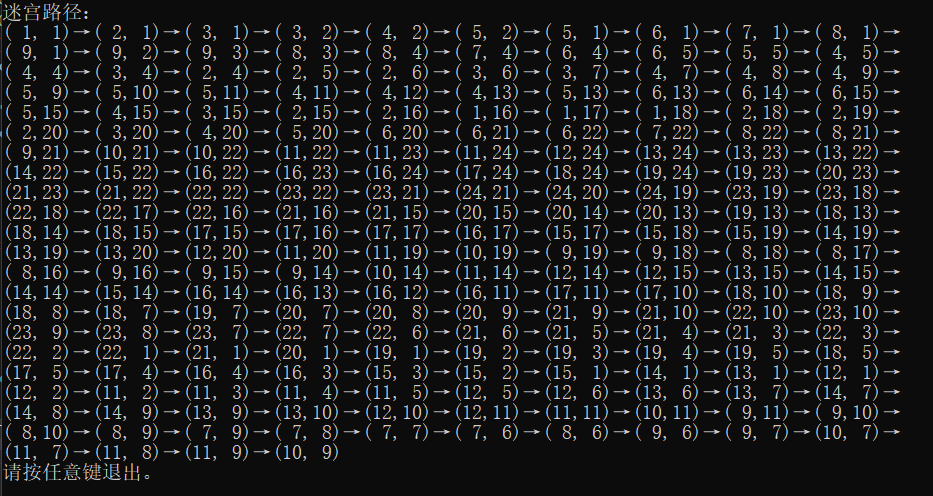
## 3.3 路径展示功能的实现

### 3.3.1 路径展示功能核心代码

1. **inline** **void** Maze::ShowPath()
2. {
3. cout << "路线展示（O为起点，X为终点,\*为路径上经过的点）：" << endl;
4. **int** i;
5. **for** (i = PointPath.GetSize() - 2; i > 0; --i)
6. {
7. **int** x = PointPath[i].x, y = PointPath[i].y;
8. MazeMap[x][y] = '\*';
9. }
10. **for** (i = 0; i < MapSize; ++i)
11. {
12. cout << MazeMap[i] << endl;
13. }
14. **for** (i = PointPath.GetSize() - 2; i > 0; --i)
15. {
16. **int** x = PointPath[i].x, y = PointPath[i].y;
17. MazeMap[x][y] = ' ';
18. }
19. **int** cnt = 0;
20. **for** (i = PointPath.GetSize() - 1; i >= 0; --i)
21. {
22. **int** x = PointPath[i].x, y = PointPath[i].y;
23. **if** (cnt % 10 == 0 && cnt != 0) cout << endl;
24. **if** (cnt == 0)   cout << "迷宫路径：" << endl;
25. cout << "(";
26. **if** (x < 10) cout << " ";
27. cout << x;
28. cout << ",";
29. **if** (y < 10) cout << " ";
30. cout << y;
31. cout << ")";
32. **if** (i != 0)
33. {
34. cout << "→";
35. }
36. cnt++;
37. }
38. }

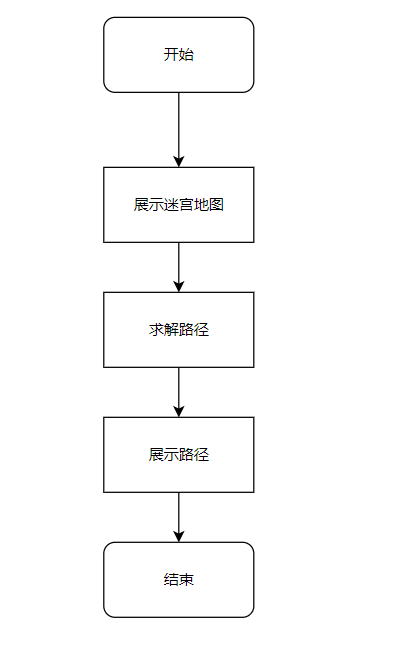
### 3.2.2 路径展示功能截屏示例





## 3.3 总体系统的实现

### 3.3.1 总体系统流程图



### 3.3.2 总体系统核心代码

1. **int** main(**void**)
2. {
3. Maze MyMaze;
4. MyMaze.ShowMap();
5. MyMaze.FindRoad();
6. MyMaze.ShowPath();
7. cout << endl;
8. cout << "请按任意键退出。" ;
9. getchar();
10. **return** 0;
11. }

### 3.3.3 总体系统截屏示例

