数据结构课程设计

项目说明文档

勇闯迷宫游戏

|  |  |
| --- | --- |
| 作者姓名： | 杨烜赫 |
| 学 号： | 2252709 |
| 指导教师： | 张 颖 |
| 学院专业： | 软件学院 软件工程 |



同济大学

Tongji University

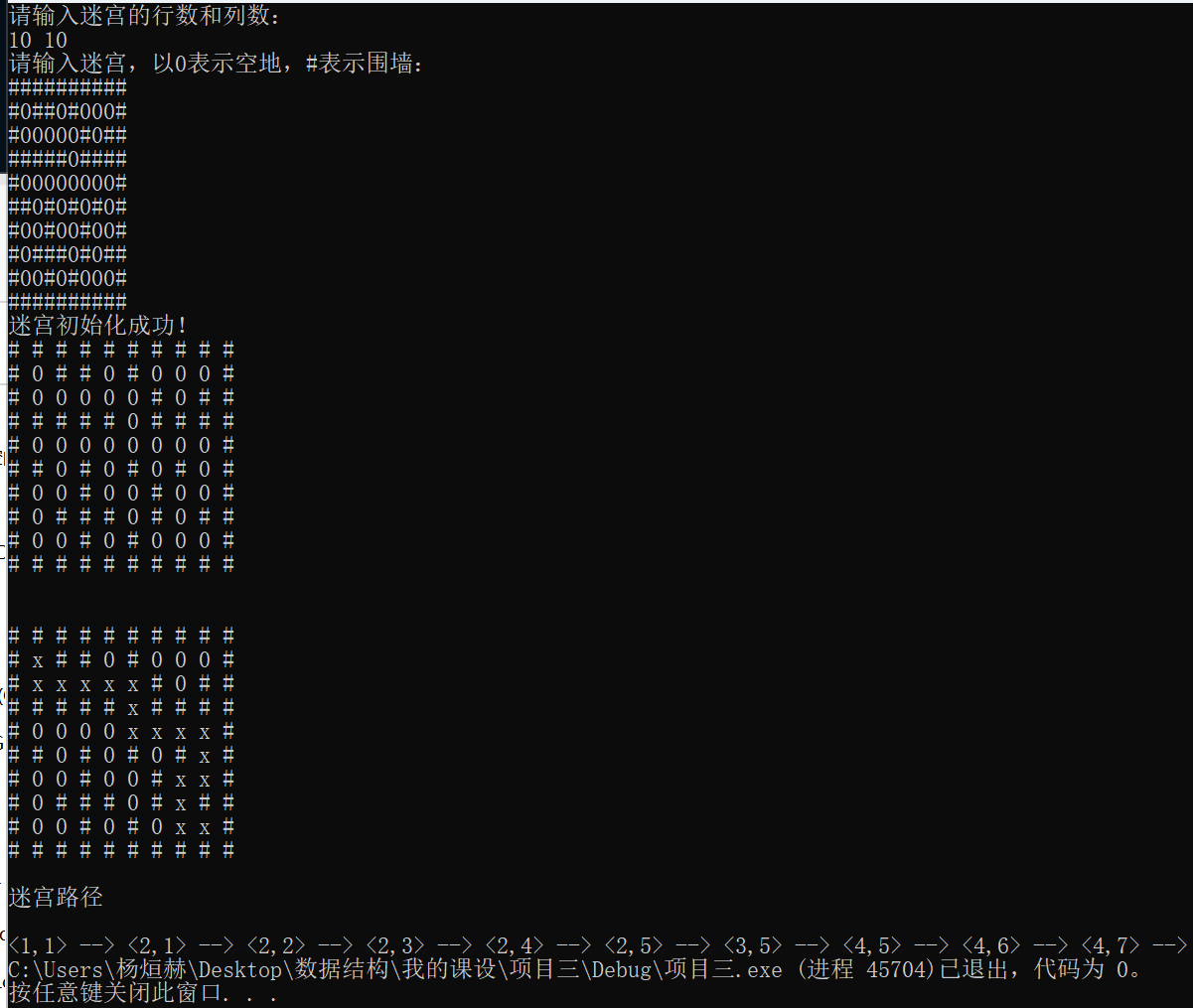
1. **题目简介**

**1.实验背景**

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。

**2.实验功能**

迷宫问题的求解过程可以采用回溯法即在一定的约束条件下试探地搜索前进，若前进中受阻，则及时回头纠正错误另择通路继续搜索的方法。从入口出发，按某一方向向前探索，若能走通，即某处可达，则到达新点，否则探索下一个方向；若所有的方向均没有通路，则沿原路返回前一点，换下一个方向再继续试探，直到所有可能的道路都探索到，或找到一条通路，或无路可走又返回入口点。在求解过程中，为了保证在达到某一个点后不能向前继续行走时，能正确返回前一个以便从下一个方向向前试探，则需要在试探过程中保存所能够达到的每个点的下标以及该点前进的方向，当找到出口时试探过程就结束了。

**3.项目示例**

1. **项目设计**

## 1.数据结构设计

在这个迷宫游项目中，数据结构设计关键在于两个方面：迷宫地图的存储和寻路路径的记录。

**迷宫地图**：使用一个二维字符数组 char\*\* map 来存储迷宫的布局，其中 '0' 表示可通行的空地，'#' 表示不可通行的围墙。

**寻路节点**：定义了一个结构体 linknode，用于表示迷宫中的每个单元格。每个节点包含其在迷宫中的坐标 (x, y)，一个标记 (mark) 表示该节点是否已被访问过，以及一个指向下一个节点的指针 next。

**路径记录**：通过 linknode 的链表 game\_linklist 来记录寻路过程中的路径。链表的每个节点表示路径上的一个位置。

## 2.类结构设计

本项目包含三个主要的类：

maze 类：用于表示迷宫本身。它包含迷宫的尺寸 (row 和 col)，迷宫地图的存储，以及相关的初始化和验证方法。

linknode 结构体：用于表示迷宫中的单个单元格或节点。

game\_linklist 类：用于管理寻路过程中的路径。它包含指向链表头的指针和相关的链表操作方法。

game\_function 类：负责寻路逻辑和路径显示。它使用 game\_linklist 来存储和处理找到的路径。

## 3.成员与操作设计

**maze 类**

成员变量：包括迷宫的行数 (row)、列数 (col)、可通行路径数量 (road\_num)、迷宫地图 (map) 和解决方案 (solve)。

**成员函数：**

构造函数 (maze())：初始化迷宫参数。

init\_maze()：输入迷宫尺寸和布局，创建迷宫地图。

check\_map()：验证迷宫的有效性，如围墙和起始点的合法性。

**game\_linklist 类**

成员变量：包括一个指向链表头的指针 (head)。

**成员函数：**

构造函数和析构函数：创建和销毁链表。

Insert(int i, linknode& e)：在链表的第 i 个位置插入一个新节点。

Getlink(int i)：获取链表中第 i 个节点的地址。

Length()：返回链表的长度。

ListDisplay()：显示链表中所有节点的坐标。

**game\_function 类**

成员变量：包括一个 game\_linklist 实例 (way) 用于存储路径，以及一个 linknode 数组 (node) 用于存储迷宫中的每个单元格。

**成员函数：**

构造函数 (game\_function())：初始化迷宫中的每个节点。

init\_find\_way()：初始化寻路函数。

find\_way()：实现寻路算法，找到从起点到终点的路径。

display\_way()：输出找到的路径。

## 4.算法设计

本题为经典的深度优先搜索例题。以下为深度优先搜索介绍：

深度优先搜索算法（Depth First Search , DFS）的思想即回溯法（Backtrack），即在一定的约束条件下试探地搜索前进，若前进中受阻，则及时回头纠正错误另择通路继续搜索的方法。从入口出发，按某一方向向前探索，若能走通，即某处可达，则到达新点，否则探索下一个方向；若所有的方向均没有通路，则沿原路返回前一点，换下一个方向再继续试探，直到所有可能的道路都探索到，或找到一条通路，或无路可走又返回入口点。

在本题的求解过程中，我们通过深度优先搜索的方式寻找可达路径。当我们处在某一个点时，我们会先沿着一个方向一直向下搜索，当在某个点搜索失败后，会回溯到上一层的点，换方向继续搜索。为了保证实现这个功能时，能正确返回前一个点以便向下一个方向试探，则需要在试探过程中保存所能够达到的每个点的下标以及该点前进的方向，当找到出口时试探过程就结束了。此时，从路径的一端回溯，并记录下回溯时经过的点，一直到另外一端，便生成了所求的路径信息。

1. **项目实现**

**3.1. 迷宫初始化**

第一部分通过game\_function类实现了对迷宫的初始化

1. //迷宫初始化
2. **class** game\_function;
3. **class** maze {
4. **friend** **class** game\_function;
5. **public**:
6. maze();
7. **bool** init\_maze();
8. **bool** check\_map();
9. **private**:
10. **int** row;
11. **int** col;
12. **int** road\_num;
13. **char**\*\* map;
14. **char**\*\* solve;
16. }Maze;
17. maze::maze()
18. {
19. row = 0;
20. col = 0;
21. road\_num = 0;
22. map = nullptr;
23. solve = nullptr;
24. }
25. **bool** maze::init\_maze() {
26. **while** (**true**) {
27. cout << "请输入迷宫的行数和列数：" << endl;
28. cin >> row >> col;
29. **if** (row <= 0 || col <= 0 || !cin.good()) {
30. cout << "输入行列不合法，请重新输入" << endl;
31. cin.clear();
32. cin.ignore(1024, '\n');
33. **continue**;
34. }
35. map = **new** **char**\* [row];
36. **for** (**int** i = 0; i < row; i++) {
37. map[i] = **new** **char**[col];
38. }
39. cout << "请输入迷宫，以0表示空地，#表示围墙：" << endl;
40. **for** (**int** i = 0; i < row; i++) {
41. **for** (**int** j = 0; j < col; j++) {
42. cin >> map[i][j];
43. }
44. }
45. // 检查迷宫有效性
46. **if** (!check\_map()) {
47. **for** (**int** i = 0; i < row; i++) {
48. **delete**[] map[i];
49. }
50. **delete**[] map;
51. cin.clear();
52. cin.ignore(1024, '\n');
53. **continue**;
54. }
56. cout << "迷宫初始化成功！" << endl;
57. // 打印迷宫
58. **for** (**int** i = 0; i < row; i++) {
59. **for** (**int** j = 0; j < col; j++) {
60. cout << map[i][j] << " ";
61. }
62. cout << endl;
63. }
64. **break**;
65. }
66. **return** **true**;
67. }
68. **bool** maze::check\_map() {
69. // 检查周围一圈是否为围墙
70. **for** (**int** i = 0; i < row; i++) {
71. **if** (map[i][0] != '#' || map[i][col - 1] != '#') **return** **false**;
72. }
73. **for** (**int** j = 0; j < col; j++) {
74. **if** (map[0][j] != '#' || map[row - 1][j] != '#') **return** **false**;
75. }
77. // 检查起始点是否为空地
78. **if** (map[1][1] != '0' || map[row - 2][col - 2] != '0') {
79. cout << "迷宫起始或终点坐标不合法，请重新输入" << endl;
80. **return** **false**;
81. }
82. **return** **true**;
83. }

**maze 类**

功能：maze 类负责迷宫的初始化和验证，确保迷宫地图的合法性。

**maze::init\_maze**功能：初始化迷宫地图。

过程：

用户输入迷宫的行数和列数。

检查输入是否合法（行数和列数必须大于0）。

动态创建二维字符数组来存储迷宫地图。

用户输入迷宫地图，其中 '0' 表示可通行的空地，'#' 表示不可通行的围墙。

调用 check\_map 方法验证迷宫的合法性。

输入验证：如果用户输入的行列数不合法，程序会提示错误信息，并要求用户重新输入。

迷宫地图输入：用户逐行输入迷宫地图，程序存储这些信息。

错误处理：如果迷宫地图不符合规定的格式，如围墙不完整或起始终点不是空地，程序提示错误并重新开始初始化过程。

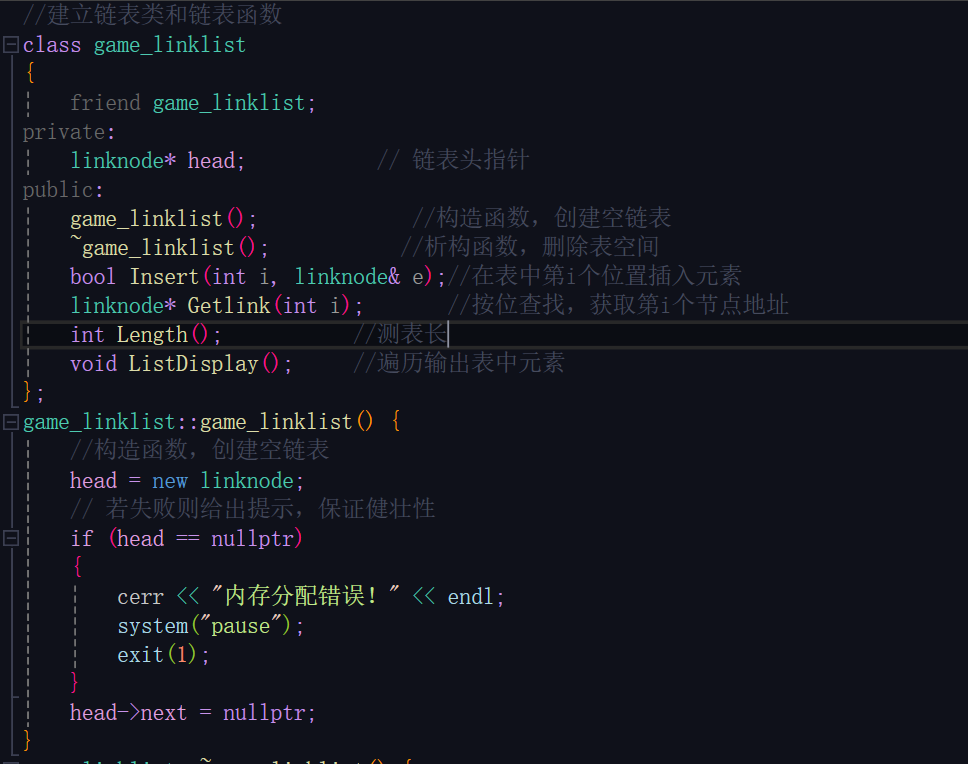
**maze::check\_map**

功能：检查迷宫地图的合法性。

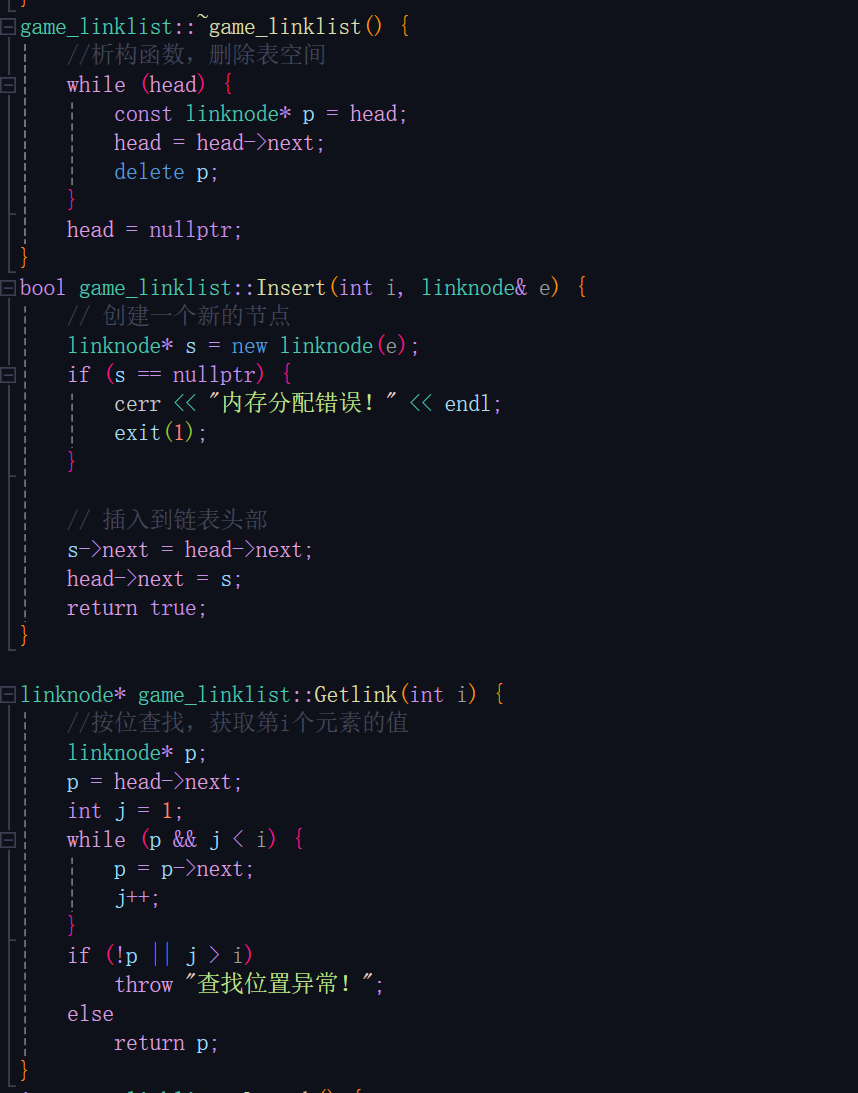
过程：

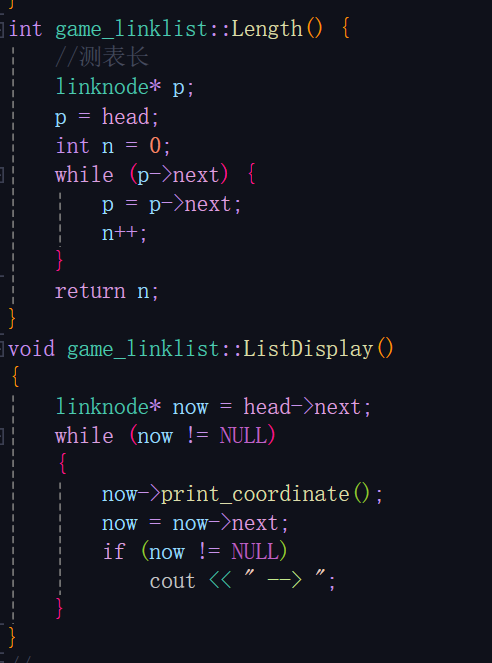
检查迷宫的四周是否完全由围墙包围。

确保迷宫的起点和终点（左上角和右下角的相邻位置）为可通行的空地。

****错误处理：如果地图不符合要求，返回 false 并提示用户重新输入。

**3.2. 路径链表建立实现**

****

****

**实现逻辑**

game\_linklist 类是一个用于管理迷宫游戏中路径节点的链表。它允许在链表中插入新节点、获取特定位置的节点、测量链表长度，并遍历输出链表中的所有元素。

**功能**

构造函数 (game\_linklist())：创建一个空链表。初始化链表的头指针 head 指向一个新的 linknode。如果内存分配失败，则抛出错误并终止程序。

析构函数 (~game\_linklist())：删除链表，释放所有节点占用的内存。遍历链表，逐个删除每个节点。

插入节点 (Insert(int i, linknode& e))在链表的头部插入一个新的 linknode。

如果内存分配失败，则抛出错误并终止程序。

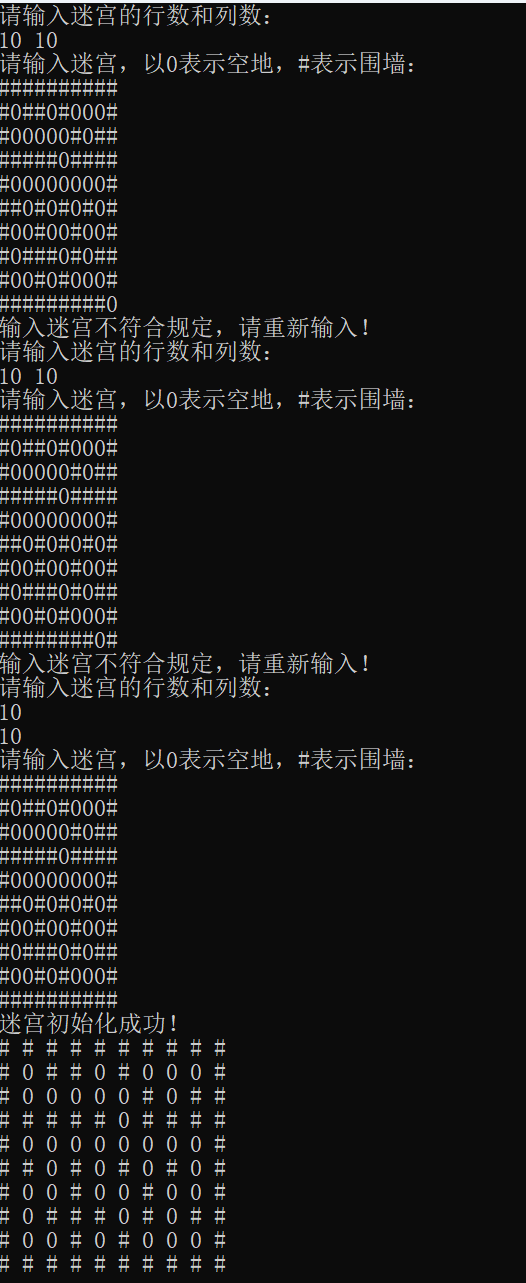
按位查找 (Getlink(int i))获取链表中第 i 个位置的节点，如果指定位置超出链表范围，抛出异常。

测量链表长度 (Length())

返回链表中节点的数量。输出链表中的元素 (ListDisplay())遍历链表，输出每个节点的坐标。节点之间用箭头 " --> " 表示连接关系。

game\_linklist 类为迷宫游戏提供了一种有效的方式来管理寻路过程中的路径。通过链表结构，它能够灵活地处理路径节点的添加和遍历操作。此外，类中的错误处理确保了程序在内存分配失败等异常情况下的稳定运行。

**3.2. 游戏部分核心算法实现**

1. //建立游戏功能实现类
2. **class** game\_function
3. {
4. **friend** **class** maze;
5. **private**:
6. game\_linklist way;
7. linknode node[X\_MAX+1][Y\_MAX+1];//位置节点
9. **public**:
10. game\_function();//初始化节点坐标
11. **void** init\_find\_way();//初始化寻路函数
12. **bool** find\_way(**int** now\_x, **int** now\_y, **int** end\_x, **int** end\_y,**char**\*\* map,**int** row,**int** col);//寻路函数
13. **void** display\_way();//输出路径
14. **const** **int** dx[4] = { 0,0,-1,1 };//定义棋子移动双数组
15. **const** **int** dy[4] = { -1,1,0,0 };
16. }game;
17. game\_function::game\_function()
18. {
19. **for** (**int** i=0;i<X\_MAX;i++)
20. {
21. **for**(**int** j=0;j<Y\_MAX;j++)
22. {
23. node[i][j].x = i;
24. node[i][j].y = j;
25. }
26. }
27. }
28. **void** game\_function::init\_find\_way()
29. {
30. find\_way(1, 1, Maze.row - 2, Maze.col - 2, Maze.map, Maze.row, Maze.col);
32. }
33. **bool** game\_function::find\_way(**int** now\_x, **int** now\_y, **int** end\_x, **int** end\_y,**char**\*\* map,**int** row,**int** col)
34. {
35. **if**(now\_x<0||now\_x>row||now\_y<0||now\_y>col|| node[now\_x][now\_y].mark ||map[now\_x][now\_y] == '#')
36. {
37. **return**  **false**;//到达障碍物或者出界
38. }
39. **else**
40. {
41. node[now\_x][now\_y].mark = **true**;
42. }
43. //终点处理
44. **if**(now\_x==end\_x&&now\_y==end\_y)
45. {
46. way.Insert(way.Length(), node[now\_x][now\_y]);
47. map[now\_x][now\_y] = 'x';
48. **return** **true**;
49. }
50. //四周搜索
51. **for**(**int** i=0;i<4;i++)
52. {
53. **if**(find\_way(now\_x+dx[i],now\_y+dy[i],end\_x,end\_y,map,row,col))
54. {
55. map[now\_x][now\_y] = 'x';
56. way.Insert(way.Length(), node[now\_x][now\_y]);
57. **return** **true**;
58. }
60. }
61. **return** **false**;
63. }
64. **void** game\_function::display\_way()
65. {
66. cout << endl << endl;
67. **for** (**int** i = 0; i <Maze. row; i++)
68. {
69. **for** (**int** j = 0; j < Maze.col; j++)
70. {
71. cout <<Maze. map[i][j] << " ";
72. }
73. cout << endl;
74. }
75. cout << endl;
76. cout << "迷宫路径" << endl;
77. cout << endl;
78. way.ListDisplay();
79. ****
81. }

game\_function 类作为迷宫游戏的核心，集成了寻路算法和路径展示的关键功能。在其构造函数中，它初始化了一个二维数组 node，代表迷宫中的每个位置节点，为每个节点赋予了对应的行列坐标。主要功能之一是 find\_way 方法，它采用递归方式来探索迷宫，寻找从起点到终点的可行路径。此方法在遇到障碍物或出界时返回失败，否则标记当前节点并继续探索四周。一旦找到终点，它就沿途标记路径并通过 game\_linklist 类的 Insert 方法将路径节点逆序存储。类中还包括 display\_way 方法，负责输出标记过的迷宫地图和找到的路径。整体而言，game\_function 类的设计兼顾了寻路算法的逻辑严谨性和用户交互的直观性，同时在内存和异常处理上表现出了足够的健全性，确保了程序在探索复杂迷宫时的稳定运行。

**健全性测试：**

输入验证：在 maze::init\_maze 方法中，程序要求用户输入迷宫的行数和列数，并检查这些输入是否合法。如果输入的行列数小于或等于0，或者输入流状态不正确（!cin.good()），程序会提示错误信息，并要求用户重新输入。这通过清空输入缓冲区（cin.clear()）和忽略之前的错误输入（cin.ignore(1024, '\n')）来实现。

内存分配：在创建迷宫地图时，程序动态分配内存来存储迷宫的布局。如果内存分配失败（new char\*[row] 返回 nullptr），则会抛出错误并终止程序。这是通过检查 new 运算符的返回值来实现的。

迷宫合法性检查：在迷宫初始化后，check\_map 方法验证迷宫地图的合法性。它检查迷宫的四周是否完全由围墙包围，并确认起始和终点位置为可通行的空地。如果检查失败，程序会提示错误并重新开始迷宫的初始化过程。

递归调用的边界检查：在 game\_function::find\_way 方法中，程序在递归调用前检查当前节点是否超出迷宫边界或遇到障碍物。这防止了程序进入无效区域或无限递归。

链表操作的健全性：在 game\_linklist 类中，链表操作（如插入节点）前检查是否成功分配了新节点的内存。如果内存分配失败，程序会抛出错误并退出。

资源管理和清理：在 game\_linklist 的析构函数中，程序逐个删除链表中的节点，并最终将头指针设置为 nullptr，这样做确保了程序结束时不会留下内存泄漏。

****

**四、程序创新点**

本程序在迷宫探索游戏的实现上展现了显著的创新性。首先，它采用了动态内存分配来创建可变尺寸的迷宫地图，这不仅增加了游戏的灵活性，还提升了用户交互体验。其次，程序在寻找迷宫出路时采用递归算法，有效地模拟了探索过程，并能实时标记路径，这增强了游戏的可视性和教育性。此外，程序中对于链表的使用在路径记录上显示出了优越性，使路径的动态记录和回溯变得简单而直观。最后，综合考虑了错误处理和用户输入验证，确保了程序的健壮性和稳定性，使其适合于教学和娱乐。

**五、实验小结**

本次实验通过构建一个迷宫探索游戏，有效地结合了数据结构的理论知识和实际编程技能。实验过程中，深入理解了递归算法的应用、链表的动态操作、以及动态内存管理的重要性。特别是在错误处理和用户交互方面的实践，极大地提升了编程的综合能力。实验不仅加深了对数据结构和算法的理解，而且锻炼了解决实际问题的能力。通过这次实践活动，能更好地将理论知识应用于复杂问题的解决中，为未来的软件开发和算法设计打下坚实的基础。