

**HUBEI UNIVERSITY OF AUTOMOTIVE TECHNOLOGY**



**数据结构**

**课程设计报告**

|  |  |
| --- | --- |
| 课设题目： | 迷宫与栈问题 |
| 专 业： | 计算机科学与技术 |
| 班 级： | 222 |
| 姓 名： | 章崇文 |
| 完成日期： | 2024.2.26-2024.3.1 |
| 指导教师： | 袁科 |

目录

[一.题目的内容及要求 3](#_Toc170394975)

[二.需求分析 4](#_Toc170394976)

[2.1原始项目需求分析 4](#_Toc170394977)

[2.2扩展需求分析 4](#_Toc170394978)

[三.概要设计 5](#_Toc170394979)

[3.1数据结构选择与存储方式 5](#_Toc170394980)

[3.2核心操作与逻辑 6](#_Toc170394981)

[3.3设计的操作与原因 7](#_Toc170394982)

[3.4操作之间的调用关系 7](#_Toc170394983)

[四、详细设计 8](#_Toc170394984)

[4.1数据结构定义 8](#_Toc170394985)

[4.2算法描述与操作 8](#_Toc170394986)

[4.3操作间的调用关系 9](#_Toc170394987)

[五、源代码 10](#_Toc170394988)

[Castar.h： 10](#_Toc170394989)

[Constants： 16](#_Toc170394990)

[Widet.h 16](#_Toc170394991)

[Castar.cpp 20](#_Toc170394992)

[Widget.cpp 23](#_Toc170394993)

[Main.cpp 38](#_Toc170394994)

[六.运行结果及分析 39](#_Toc170394995)

[6.1基础界面展示 39](#_Toc170394996)

[功能展示 39](#_Toc170394997)

[七、收获及体会，总结 43](#_Toc170394998)

一.题目的内容及要求

**【问题描述】**

以一个mXn的矩阵表示迷宫，0和1分别表示迷宫中的通路和障碍。设计一个程序，对任意设定的迷宫，求出一条从入口到出口的通路，或得出没有通路的结论。

**【任务要求】**

1. 首先实现一个以链表作存储结构的栈类型，然后编写一个求解迷宫的非递归程序。求得的通路以三元组（i，j，d）的形式输出。其中：（i，j）指示迷宫中的一个坐标，d表示走到下一坐标的方向。如，对于下列数据的迷宫，输出一条通路为：（1，1，1），（1，2，2），（2，2，2），（3，2，3），（3，1，2），…。
2. 编写递归形式的算法，求得迷宫中所有可能的通路。
3. 以方阵形式输出迷宫及其通路。

**【测试数据】**

迷宫的测试数据如下：左上角（0，1）为入口，右下角（8，9）为出口。



二.需求分析

2.1原始项目需求分析

1. 迷宫模型定义：首先，系统需处理一个以m×n矩阵表示的迷宫，其中0表示可通过的通路，1表示无法通行的障碍。此模型简单而直观，易于理解和编程实现。
2. 路径求解核心功能：
   * 非递归栈方法：实现一个基于链表的栈数据结构，用于存储探索过程中的坐标信息。非递归算法利用栈来模拟递归调用，逐层深入迷宫，回溯时则弹出栈顶元素，直到找到出口或探索完所有可达路径。
   * 方向编码：每一步探索动作以三元组（i, j, d）记录，明确指出当前位置（i, j）及前往下一个位置的方向（d），便于理解和复现探索路径。
3. 递归算法实现：设计递归算法以探索迷宫的所有可能通路。递归方法天然符合迷宫探索的“分而治之”策略，通过不断细分探索区域直至达到出口或遇到障碍，递归返回时逐步拼接成完整的通路。
4. 结果输出：要求程序不仅能判断是否存在通路，还要能够输出至少一条通路的具体步骤，或是在递归情况下输出所有可能的通路，为用户提供清晰的解决方案验证。

2.2扩展需求分析

在此基础上，项目进行了以下扩展：

1. 图形用户界面集成：采用Qt框架开发的界面，不仅增强了用户体验，还使得迷宫构建与探索过程变得直观易操作。用户交互设计允许直接点击设置障碍、起点和终点，大大简化了迷宫配置流程，增加了使用的灵活性和趣味性。
2. 算法多样性：除了基础要求中的栈方法和递归算法，还集成了深度优先搜索(DFS)、广度优先搜索(BFS)以及A\*搜索算法，满足不同场景下的路径搜索需求。这些算法的实现丰富了系统的功能，提升了问题解决的效率和质量。
3. 动态展示与反馈：通过方阵形式的动态迷宫展示，用户可以直观看到迷宫结构、当前探索路径及最终解决方案。高亮显示的路径不仅便于理解算法工作原理，也增加了程序的观赏性和教学价值。
4. 性能与实用性考量：在扩展功能的同时，注重算法的效率与实用性，确保在处理不同规模迷宫时均能保持良好的响应时间。特别是对于A\*算法，通过精心设计启发式函数，平衡探索成本与路径长度，实现高效寻路。

三.概要设计

3.1数据结构选择与存储方式

1. Maze Representation: 使用std::vector<std::vector<int>>来表示迷宫，其中每个元素int代表迷宫的一个单元格，值为0表示通路，1表示障碍。这种二维动态数组的选择是因为它能够高效地存储和访问迷宫数据，且易于理解和实现。
2. Path Storage: 对于路径的存储，采用了std::vector<std::pair<int, int>>来表示每条路径上的坐标序列。每一对(int, int)代表路径上的一个坐标点。在A\*算法中，则使用了vector<APoint\*>来存储最终路径，其中APoint是一个自定义类，封装了坐标信息和相关属性。这样的设计使得路径的管理和遍历更为灵活。
3. Stack and Queue: 在非递归和BFS算法实现中，分别对应深度优先和广度优先搜索中的节点访问顺序。这些标准库容器提供了高效的入栈、出栈和入队、出队操作，适合迷宫探索过程中的路径探索和回溯。
4. APoint类：定义了迷宫中的单个点，包含坐标(x, y)，以及类型AType（如起始点StartPoint、终点EndPoint、路径点Path、已访问点Visited、墙壁Wall等）。此外，还包括了启发式成本h\_cost、从起点到该点的实际成本g\_cost，以及综合成本f\_cost（由g\_cost + h\_cost计算得出）。
5. 邻接点探索：使用双向循环遍历(i, j)，其中i和j取值为{-1, 0, 1}，排除原点自身，以检查上下左右及对角线邻居。
6. **优先级队列：**但在A\*算法的核心循环中，通过每次循环选择f\_cost最小的点进行处理，实质上模拟了优先级队列的行为，以确保总是探索最有希望的路径。

3.2核心操作与逻辑

* 启发式成本计算：通过heuristicCost方法采用曼哈顿距离作为启发式函数，计算两点间的直线距离，作为路径估计成本。
* 点类型管理：setPointType操作允许更改点的类型，但不覆盖起始点和终点，以保护这些关键点的特性。这在路径回溯和可视化更新时非常重要。
* A 寻路算法实现\*：
  + 初始化：创建开放集（待探索点集合）和封闭集（已探索点集合），并将起始点加入开放集。
  + 主循环：在开放集非空时持续循环，每次迭代选择具有最低f\_cost的点作为当前点进行探索。
  + 目标检测：若当前点等于终点，则通过回溯构造路径，更新点类型，并发出可视化更新信号，最后返回路径起点。
  + 邻居探索：对当前点的八个潜在邻居进行遍历，计算移动成本，检查是否为墙壁或已关闭的点，如果邻居不在开放集中，则添加之；如果在开放集中且新的路径成本更低，则更新邻居的父节点、成本，并调整其在开放集中的位置（隐式实现）。
  + 回溯与标记：一旦找到路径，通过回溯从终点到起点，标记路径上的点为Path类型，并恢复起点和终点的类型。
  + 无路径情况：如果开放集变空而未找到路径，则返回nullptr，表示没有可行路径。

3.3设计的操作与原因

* findWay: 在CAstar类中，findWay方法负责执行A搜索算法，通过计算启发式函数估计与实际代价，寻找从起点到终点的最短路径。A算法因其平衡了探索的广度和深度，特别适合于寻求最优解。
* heuristicCost: 提供了一个启发式函数的实现，用于估算从当前点到目标点的代价，这对于A\*算法的高效寻路至关重要。
* setPointType: 用于设置迷宫中某个点的类型，比如障碍或特殊标记，这在动态更新迷宫状态或标记已探索区域时非常有用。
* backtrack: 在MazeSolver类中，backtrack方法是递归实现的核心，用于通过回溯法探索迷宫的所有可能路径。递归调用减少了代码的复杂度，使路径探索逻辑更为清晰。

3.4操作之间的调用关系

* 主流程调用: 用户通过UI操作初始化迷宫（使用std::vector<std::vector<int>>表示），设定起点和终点坐标。之后，可以选择不同的算法接口来求解路径。
* A\*算法流程: 调用CAstar::findWay开始A\*搜索，内部会调用heuristicCost计算启发式代价，同时维护一个开放列表（通常以优先队列形式实现，尽管代码中未直接展现）和一个关闭列表来管理探索过程。
* 递归求解所有路径: 当调用MazeSolver::findPaths时，会触发递归函数backtrack的执行。backtrack负责逐层深入迷宫，每当到达一个新位置时，都会尝试向四个方向继续探索，并在成功到达终点时保存当前路径。通过递归回溯，收集所有可能的路径。
* 可视化更新: 在探索过程中，每当路径有更新（不论是单一最短路径还是所有可能路径），都可以通过信号槽机制触发visualizationUpdated信号，进而通知UI组件更新视图，展示迷宫与路径的最新状态。

四、详细设计

（包括数据结构的类型定义，每个操作的算法描述）

4.1数据结构定义

1. **APoint类**:
   * **成员变量**:
     + int x, y: 表示点的坐标。
     + AType type: 点的类型，如起始点、终点、路径点、墙壁等。
     + double g\_cost, h\_cost, f\_cost: 分别表示从起点到当前点的实际代价、启发式估算代价、总代价。
     + APoint\* parent: 指向父节点，用于路径回溯。
   * **方法**:
     + heuristicCost: 计算与目标点的曼哈顿距离作为启发式成本。
     + setPointType: 设置点的类型，保护起始点和终点不被覆盖。
2. **CAstar类**:
   * 包含A\*寻路算法的主要实现，包括findWay, heuristicCost, 和setPointType方法。
3. **MazeSolver类**（假设存在，根据您的描述）:
   * 虽然直接代码未给出，但根据上下文推断，可能包含递归或非递归算法来寻找所有可能路径。

4.2算法描述与操作

1. \**A* 寻路算法 (findWay)\*\*:
   * **初始化**: 创建开放集（openSet）和封闭集（closedSet），并将起始点加入开放集。
   * **循环探索**:
     + 选择开放集中f\_cost最小的点作为当前点。
     + 若当前点为终点，则回溯构建路径并返回。
     + 将当前点移出开放集，加入封闭集。
     + 遍历当前点的邻居，计算移动成本，检查是否可通行且未访问过。
       - 对新邻居，计算其g\_cost、h\_cost、f\_cost，并设置父节点。
       - 若邻居不在开放集中，则加入；若已存在但新的路径更优，则更新相关信息。
       - 探索过程中，标记访问过的点为已访问（Visited）类型。
   * 若开放集为空，表示无路径可走，返回nullptr。
2. **启发式成本计算 (heuristicCost)**:
   * 计算两个点间曼哈顿距离，即横纵坐标差的绝对值之和，作为预估成本。
3. **点类型管理 (setPointType)**:
   * 改变点的类型，但确保起始点和终点类型不被覆盖，以维持寻路逻辑的完整性。

4.3操作间的调用关系

* **UI与算法交互**:
  + 用户通过UI设置迷宫、起点、终点后，触发A\*寻路算法(findWay)。
* *A 寻路过程*\*:
  + 初始化时，调用heuristicCost计算起始点的启发式成本。
  + 在循环中，持续调用heuristicCost计算每个邻居的启发式成本。
  + 对于每个考虑的邻居，可能会调用setPointType更新其类型。
  + 找到路径后，通过回溯更新路径点的类型，并最终发出visualizationUpdated信号通知UI更新。
* **可视化与反馈**:
  + UI接收到visualizationUpdated信号后，根据最新的点类型信息更新视图，展示迷宫状态和路径。

五、源代码

Castar.h：

/\*

F:路径评分 = g+h

G:走一格格子的花销

H:当前格子到目标格子的估算花销

上下左右走一格花销为10，斜着走一格花销为14，以方便计算

即格子宽高为10 对角线为14

\*/

#ifndef CAstar\_H

#define CAstar\_H

#include "constants.h"

#include <QObject>

#include <QLabel>

#include <QPushButton>

#include <QPainter>

#include <QPaintEvent>

#include <QWidget>

#include <QDebug>

#include <qstring.h>

#include <vector>

#include <queue>

#include <stack>

#include <iostream>

#include <vector>

#include <QTextEdit>

#include <memory>

#include <set>

using namespace std;

class APoint;

class CAstar;

class BFS;

enum class AType {

Unknow,

Wall,

StartPoint,

EndPoint,

Path,

Visited

};

class APoint {

public:

~APoint();

int x;

int y;

AType type;

double current\_cost;

double f\_cost;

double g\_cost;

double h\_cost;

APoint\* parent;

bool operator==(const APoint& po) const {

return (x == po.x && y == po.y);

}

APoint() : x(0), y(0), type(AType::Unknow), g\_cost(0.0), f\_cost(0.0), parent(nullptr) {}

APoint(int x, int y) : x(x), y(y), type(AType::Unknow), g\_cost(0.0), f\_cost(0.0), parent(nullptr) {}

};

class CAstar : public QObject {

Q\_OBJECT

signals:

void visualizationUpdated();

public:

APoint\* findWay(APoint\* beginPoint, APoint\* endPoint, vector<vector<APoint\*>>& allPoints);

double heuristicCost(const APoint\* a, const APoint\* b);

void setPointType(APoint\* point, AType type);

vector<APoint\*> path;

};

class MazeSolver {

public:

std::vector<std::vector<std::pair<int, int>>> findPaths(const std::vector<std::vector<int>>& maze,

const std::pair<int, int>& start,

const std::pair<int, int>& end)

{

int rows = maze.size();

int cols = maze[0].size();

std::vector<std::vector<bool>> visited(rows, std::vector<bool>(cols, false));

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

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

path.push\_back(start);

backtrack(maze, start.first, start.second, end, path, paths, visited);

return paths;

}

private:

void backtrack(const std::vector<std::vector<int>>& maze, int row, int col,

const std::pair<int, int>& end, std::vector<std::pair<int, int>>& path,

std::vector<std::vector<std::pair<int, int>>>& paths,

std::vector<std::vector<bool>>& visited)

{

if (row == end.first && col == end.second)

{

paths.push\_back(path);

return;

}

visited[row][col] = true;

// 上

if (row > 0 && maze[row - 1][col] == 0 && !visited[row - 1][col])

{

path.push\_back(std::make\_pair(row - 1, col));

backtrack(maze, row - 1, col, end, path, paths, visited);

path.pop\_back();

}

// 下

if (row < maze.size() - 1 && maze[row + 1][col] == 0 && !visited[row + 1][col])

{

path.push\_back(std::make\_pair(row + 1, col));

backtrack(maze, row + 1, col, end, path, paths, visited);

path.pop\_back();

}

// 左

if (col > 0 && maze[row][col - 1] == 0 && !visited[row][col - 1])

{

path.push\_back(std::make\_pair(row, col - 1));

backtrack(maze, row, col - 1, end, path, paths, visited);

path.pop\_back();

}

// 右

if (col < maze[0].size() - 1 && maze[row][col + 1] == 0 && !visited[row][col + 1])

{

path.push\_back(std::make\_pair(row, col + 1));

backtrack(maze, row, col + 1, end, path, paths, visited);

path.pop\_back();

}

visited[row][col] = false;

}

};

#endif // CAstar\_H

Constants：

#ifndef CONSTANTS\_H

#define CONSTANTS\_H

const int MAX\_X = 10;

const int MAX\_Y = 10;

const int gridSize = 30; // 每个格子的大小

const int startX = 50; // 网格划线的起点横坐标

const int startY = 50; // 网格划线的起点纵坐标

#endif // CONSTANTS\_H

Widet.h

#ifndef WIDGET\_H

#define WIDGET\_H

#include <QObject>

#include <QVBoxLayout>

#include <QWidget>

#include <QLabel>

#include <QPushButton>

#include <QPainter>

#include <QPaintEvent>

#include <QDebug>

#include <QIcon>

#include <qstring.h>

#include <QPixmap>

#include <QComboBox>

#include <QVBoxLayout>

#include <QApplication>

#include <QComboBox>

#include <QMessageBox>

#include <QTextEdit>

#include <fstream>

#include <sstream>

#include <streambuf>

#include <iostream>

#include "constants.h"

#include "castar.h"

QT\_BEGIN\_NAMESPACE

namespace Ui {

class Widget;

}

QT\_END\_NAMESPACE

class Widget : public QWidget

{

Q\_OBJECT

public:

Widget(QWidget \*parent = nullptr);

~Widget();

vector<vector<APoint\*>> Mainmap;

// vector<vector<APoint\*>> AStarmap;

// vector<vector<APoint\*>> DFSmap;

// vector<vector<APoint\*>> BFSmap;

QPoint startPoint = QPoint(-1, -1); //初始 起点位置

QPoint endPoint = QPoint(-1, -1); // 初始 终点坐标

private:

Ui::Widget \*ui;

QPushButton\* AStarButton;

QPushButton\* DFSButton;

QPushButton\* BFSButton;

QPushButton\* clearButton;

QPushButton\* loadButton;

QPushButton\* saveButton;

QPushButton\* label\_tips;

QTextEdit \* textEdit;

QTimer\* updateTimer; // 添加 updateTimer 成员变量

bool isLeftButtonPressed = false; // 左键是否被按下

// 初始化迷宫

std::vector<std::vector<APoint\*>> initializeMaze(int width, int height);

// std::vector<std::vector<std::unique\_ptr<APoint>>> initializeMaze(int width, int height);

signals:

void updateTextEdit(const QString &text);

public:

void paintEvent(QPaintEvent\* event) override;

void mousePressEvent(QMouseEvent\* event) override;

void mouseMoveEvent(QMouseEvent\* event) override;

public slots:

void updateVisualizationSlot();

void handleUpdateTextEdit(const QString &text);

void AStar();

void BFS();

void DFS();

void clearMap();

void loadMap();

void saveMap();

void showTipsDialog();

};

#endif // WIDGET\_H

Castar.cpp

// 上下左右走一格花销为10，斜着走一格花销为14，以方便计算

// 即格子宽高为10 对角线为14

#include "castar.h"

double CAstar::heuristicCost(const APoint\* a, const APoint\* b) {

return abs(a->x - b->x) + abs(a->y - b->y);

}

void CAstar::setPointType(APoint\* point, AType type) {

if (point->type == AType::StartPoint || point->type == AType::EndPoint) {

return; // Don't override start and end points

}

point->type = type;

}

APoint\* CAstar::findWay(APoint\* beginPoint, APoint\* endPoint, vector<vector<APoint\*>>& allPoints) {

vector<APoint\*> openSet;

vector<APoint\*> closedSet;

openSet.push\_back(beginPoint);

while (!openSet.empty()) {

APoint\* current = openSet[0];

for (auto point : openSet) {

if (point->f\_cost < current->f\_cost) {

current = point;

}

}

if (\*current == \*endPoint) {

APoint\* pathEnd = current;

// 回溯路径并标记

while (pathEnd != nullptr) {

// 恢复起点和终点的原始类型

beginPoint->type = AType::StartPoint;

endPoint->type = AType::EndPoint;

setPointType(pathEnd, AType::Path);

pathEnd = pathEnd->parent;

}

emit visualizationUpdated();

return current; // 返回路径的起点

}

openSet.erase(remove(openSet.begin(), openSet.end(), current), openSet.end());

closedSet.push\_back(current);

for (int i = -1; i <= 1; ++i) {

for (int j = -1; j <= 1; ++j) {

if (i == 0 && j == 0) continue;

int neighborX = current->x + i;

int neighborY = current->y + j;

if (neighborX >= 0 && neighborX < allPoints.size() && neighborY >= 0 && neighborY < allPoints[0].size()) {

APoint\* neighbor = allPoints[neighborX][neighborY];

double cost = 1.0; // 默认移动代价为1

if ((i == 0 && abs(j) == 1) || (abs(i) == 1 && j == 0)) {

// 如果是上下左右移动，则移动代价为1

cost = 1.0;

} else if (abs(i) == 1 && abs(j) == 1) {

// 如果是对角线移动，则移动代价为1.4

cost = 14;

}

if (neighbor->type != AType::Wall && find(closedSet.begin(), closedSet.end(), neighbor) == closedSet.end()) {

double tentative\_g\_cost = current->g\_cost + cost;

if (find(openSet.begin(), openSet.end(), neighbor) == openSet.end()) {

openSet.push\_back(neighbor);

} else if (tentative\_g\_cost >= neighbor->g\_cost) {

continue;

}

neighbor->parent = current;

neighbor->g\_cost = tentative\_g\_cost;

neighbor->h\_cost = heuristicCost(neighbor, endPoint);

neighbor->f\_cost = neighbor->g\_cost + neighbor->h\_cost;

// Set neighbor's type to Visited

setPointType(neighbor, AType::Visited);

}

}

}

}

}

return nullptr; // 如果找不到路径，则返回空指针

}

Widget.cpp

#include "widget.h"

#include "./ui\_widget.h"

Widget::Widget(QWidget \*parent)

: QWidget(parent), ui(new Ui::Widget)

{

Mainmap = initializeMaze(MAX\_X, MAX\_Y);

ui->setupUi(this);

// 设置窗口的固定大小

setFixedSize(800, 600);

// 设置窗口标题

setWindowTitle("HUAT 数据结构课设 课题一 By Kerwin Zhang");

// 设置窗口图标

QPixmap pixmap("D:\\coding\\Study\_code\\Qt\_Project\\MazeProject\\81A5F206980373F0A3F35A7F230\_C305AE68\_27744.png");

QIcon icon(pixmap);

setWindowIcon(icon);

setMouseTracking(true);

// 创建并连接按钮

AStarButton = new QPushButton("A\*");

BFSButton = new QPushButton("BFS");

DFSButton = new QPushButton("DFS");

clearButton = new QPushButton("清理地图");

loadButton = new QPushButton("加载地图");

saveButton = new QPushButton("保存地图");

label\_tips = new QPushButton("提示");

textEdit = new QTextEdit(this);

// 设置按钮的大小

const QSize buttonSize(120, 40);

AStarButton->setFixedSize(buttonSize);

BFSButton->setFixedSize(buttonSize);

DFSButton->setFixedSize(buttonSize);

clearButton->setFixedSize(buttonSize);

loadButton->setFixedSize(buttonSize);

saveButton->setFixedSize(buttonSize);

label\_tips->setFixedSize(buttonSize);

textEdit->setPlaceholderText("等待你的选择");

textEdit->setReadOnly(true);

textEdit->move(550,50);

textEdit->resize(180, 300);

// 创建垂直布局

QVBoxLayout \*layout = new QVBoxLayout(this);

layout->setSpacing(10);

layout->setAlignment(Qt::AlignTop);

// 添加按钮和标签到布局

layout->addWidget(AStarButton);

layout->addWidget(BFSButton);

layout->addWidget(DFSButton);

layout->addWidget(clearButton);

layout->addWidget(loadButton);

layout->addWidget(saveButton);

layout->addWidget(label\_tips);

// 设置布局的边距

layout->setContentsMargins(400, 50, 0, 0);

// 设置布局到窗口

setLayout(layout);

// 连接按钮点击事件到槽函数

connect(AStarButton, &QPushButton::clicked, this, &Widget::AStar);

connect(BFSButton, &QPushButton::clicked, this, &Widget::BFS);

connect(DFSButton, &QPushButton::clicked, this, &Widget::DFS);

connect(clearButton, &QPushButton::clicked, this, &Widget::clearMap);

connect(loadButton, &QPushButton::clicked, this, &Widget::loadMap);

connect(saveButton, &QPushButton::clicked, this, &Widget::saveMap);

connect(this, &Widget::updateTextEdit, this, &Widget::handleUpdateTextEdit);

// 连接label\_tips按钮点击事件到槽函数

connect(label\_tips, &QPushButton::clicked, this, &Widget::showTipsDialog);

}

Widget::~Widget()

{

// for (int i = 0; i < MAX\_X; i++) {

// for (int j = 0; j < MAX\_Y; j++) {

// delete Mainmap[i][j];

// }

// }

delete ui;

}

// 绘制代码...

void Widget::paintEvent(QPaintEvent\* event) {

QPainter painter(this);

painter.setRenderHint(QPainter::Antialiasing, true);

for (int i = 0; i < Mainmap.size(); i++) {

for (int j = 0; j < Mainmap[i].size(); j++) {

// APoint\* point = Mainmap[i][j];

int x = startX + j \* gridSize;

int y = startY + i \* gridSize;

// point->printAPoint();

if (Mainmap[i][j]->type == AType::Wall) {

painter.fillRect(x, y, gridSize, gridSize, Qt::black); // 黑色用于墙壁，以便在浅色背景上有更好的对比度

} else if (Mainmap[i][j]->type == AType::Unknow) {

painter.fillRect(x, y, gridSize, gridSize, Qt::white); // 白色用于未知区域，保持低调且易于区分

} else if (Mainmap[i][j]->type == AType::StartPoint) {

painter.fillRect(x, y, gridSize, gridSize, Qt::cyan); // 浅蓝色用于起点，清晰且引人注目

} else if (Mainmap[i][j]->type == AType::EndPoint) {

painter.fillRect(x, y, gridSize, gridSize, Qt::red); // 红色用于终点，色彩鲜明，易于识别

} else if (Mainmap[i][j]->type == AType::Path) {

painter.fillRect(x, y, gridSize, gridSize, Qt::green); // 绿色用于路径，自然且直观

} else if (Mainmap[i][j]->type == AType::Visited) {

painter.fillRect(x, y, gridSize, gridSize, Qt::darkGray); // 深灰色用于已访问区域，醒目且不失优雅

}

}

}

// 绘制网格线

painter.setPen(Qt::black);

// 绘制水平线

for (int i = 0; i <= MAX\_X; i++) {

int y = startY + i \* gridSize;

painter.drawLine(startX, y, startX + MAX\_Y \* gridSize, y);

}

// 绘制垂直线

for (int j = 0; j <= MAX\_Y; j++) {

int x = startX + j \* gridSize;

painter.drawLine(x, startY, x, startY +MAX\_X \* gridSize);

}

}

// 鼠标事件处理代码...

void Widget::mousePressEvent(QMouseEvent\* event) {

int x = event->position().x();

int y = event->position().y();

int i = (y - startY) / gridSize;

int j = (x - startX) / gridSize;

if (i >= 0 && i < Mainmap.size() && j >= 0 && j < Mainmap[i].size()) {

if (event->button() == Qt::LeftButton) {

Mainmap[i][j]->type = (Mainmap[i][j]->type == AType::Wall) ? AType::Unknow : AType::Wall;

} else if (event->button() == Qt::RightButton) {

if (startPoint == QPoint(-1, -1) && endPoint == QPoint(-1, -1)) {

startPoint = QPoint(j, i);

Mainmap[i][j]->type = AType::StartPoint;

} else if (startPoint != QPoint(-1, -1) && endPoint == QPoint(-1, -1)) {

endPoint = QPoint(j, i);

Mainmap[i][j]->type = AType::EndPoint;

} else if (startPoint != QPoint(-1, -1) && endPoint != QPoint(-1, -1)) {

Mainmap[startPoint.y()][startPoint.x()]->type = AType::Unknow;

Mainmap[endPoint.y()][endPoint.x()]->type = AType::Unknow;

endPoint = QPoint(-1, -1);

startPoint = QPoint(-1, -1);

}

}

update();

}

}

void Widget::mouseMoveEvent(QMouseEvent\* event) {

int x = event->position().x();

int y = event->position().y();

int i = (y - startY) / gridSize;

int j = (x - startX) / gridSize;

if (i >= 0 && i < Mainmap.size() && j >= 0 && j < Mainmap[i].size()) {

if (event->buttons() & Qt::LeftButton) {

Mainmap[i][j]->type = AType::Wall;

update();

}

}

}

std::vector<std::vector<APoint\*>> Widget::initializeMaze(int width, int height) {

std::vector<std::vector<APoint\*>> maze(width, std::vector<APoint\*>(height));

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

for (int j = 0; j < height; j++) {

APoint\* pt = new APoint();

pt->type = AType::Unknow;

pt->x = i;

pt->y = j;

maze[i][j] = pt;

}

}

return maze;

}

// A\*

void Widget::AStar()

{

emit updateTextEdit("A\*");

if (startPoint == QPoint(-1, -1) || endPoint == QPoint(-1, -1)) {

emit updateTextEdit("请设置起点和终点！");

return;

}

CAstar astar;

APoint\* pathEnd = astar.findWay(Mainmap[startPoint.y()][startPoint.x()], Mainmap[endPoint.y()][endPoint.x()], Mainmap);

// 构建路径字符串

QString pathString = "路径：";

QVector<APoint\*> path; // 存储路径节点

while (pathEnd != nullptr) {

path.push\_back(pathEnd);

pathEnd = pathEnd->parent;

}

// 反转路径

std::reverse(path.begin(), path.end());

// 将路径添加到字符串中

for (int i = 0; i < path.size(); ++i) {

pathString += "(" + QString::number(path[i]->x) + ", " + QString::number(path[i]->y) + ")";

if (i != path.size() - 1) {

pathString += " -> ";

}

}

emit updateTextEdit(pathString);

update();

}

//BFS

void Widget::BFS()

{

emit updateTextEdit("点击BFS");

std::vector<std::vector<int>> maze(MAX\_X, std::vector<int>(MAX\_Y));

for (int i = 0; i < MAX\_X; i++) {

for (int j = 0; j < MAX\_Y; j++) {

if (Mainmap[i][j]->type == AType::Wall)

{

maze[i][j] =1;

}else{

maze[i][j] = 0;

}

}

}

MazeSolver solver;

std::pair<int, int> start = {startPoint.y(), startPoint.x()};

std::pair<int, int> end = {endPoint.y(), endPoint.x()};

std::vector<std::vector<std::pair<int, int>>> paths = solver.findPaths(maze, start, end);

for (size\_t i = 0; i < paths.size(); ++i) {

std::stringstream ss;

for (size\_t j = 0; j < paths[i].size(); ++j) {

ss << "(" << paths[i][j].first << ", " << paths[i][j].second << ")";

if (j < paths[i].size() - 1) {

ss << " -> ";

}

}

std::cout << "Path " << (i + 1) << ": " << ss.str() << std::endl;

// QString str = "Path " + QString::number(i + 1) + ": " + QString::fromStdString(ss.str());

// emit updateTextEdit(str);

}

}

//DFS

void Widget::DFS()

{

emit updateTextEdit("点击DFS");

}

// 清理地图按钮点击事件处理函数

void Widget::clearMap() {

emit updateTextEdit("点击清理地图");

Mainmap = initializeMaze(MAX\_X, MAX\_Y);

update();

}

// 加载地图按钮点击事件处理函数

void Widget::loadMap() {

emit updateTextEdit("点击加载地图");

std::string filePath = "D:\\coding\\Study\_code\\Qt\_Project\\MazeProject\\map.txt";

std::ifstream file(filePath);

if (!file.is\_open()) {

emit updateTextEdit("无法打开地图文件进行加载！");

return;

}

Mainmap = initializeMaze(MAX\_X, MAX\_Y);

std::string line;

while (std::getline(file, line)) {

std::vector<APoint\*> row;

std::istringstream iss(line);

int value;

while (iss >> value) {

AType type = (value == 1) ? AType::Wall : AType::Unknow;

APoint\* point = new APoint();

row.push\_back(point);

}

Mainmap.push\_back(row);

}

file.close();

emit updateTextEdit("地图已成功从" + QString::fromStdString(filePath) + "加载");

update();

}

// 保存地图按钮点击事件处理函数

void Widget::saveMap() {

emit updateTextEdit("点击保存地图");

std::string filePath = "D:\\coding\\Study\_code\\Qt\_Project\\MazeProject\\map.txt";

std::ofstream file(filePath);

if (!file.is\_open()) {

emit updateTextEdit("无法打开文件进行保存！");

return;

}

for (int i = 0; i < Mainmap.size(); ++i) {

for (int j = 0; j < Mainmap[i].size(); ++j) {

if (Mainmap[i][j]->type == AType::Wall) {

file << "1 ";

} else {

file << "0 ";

}

}

file << std::endl;

}

file.close();

emit updateTextEdit("地图已成功保存到" + QString::fromStdString(filePath));

}

// 可视化更新槽函数

void Widget::updateVisualizationSlot() {

emit updateTextEdit("可视化更新");

repaint();

}

void Widget::showTipsDialog() {

QMessageBox::information(this, "使用说明",

"<b>右键单击</b>格子选定<b>起始点</b>，<b>左键单击</b>格子选定<b>墙壁</b>或<b>删除墙壁</b>。"

"<br><br>"

"<b>颜色说明：</b>"

"<ul>"

" <li>蓝色：起始点</li>"

" <li>红色：终点</li>"

" <li>黑色：墙壁</li>"

" <li>灰色：已搜索过的（closed 列表）</li>"

" <li>绿色：当前搜索到的路径</li>"

"</ul>"

"<br>"

"<b>联系方式：</b>"

"zhangjszs@foxmail.com"

"<br>"

"<b>Version 1.0</b>"

);

}

void Widget::handleUpdateTextEdit(const QString &text)

{

textEdit->setText(text);

}

Main.cpp

#include "widget.h"

#include "castar.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

Widget w;

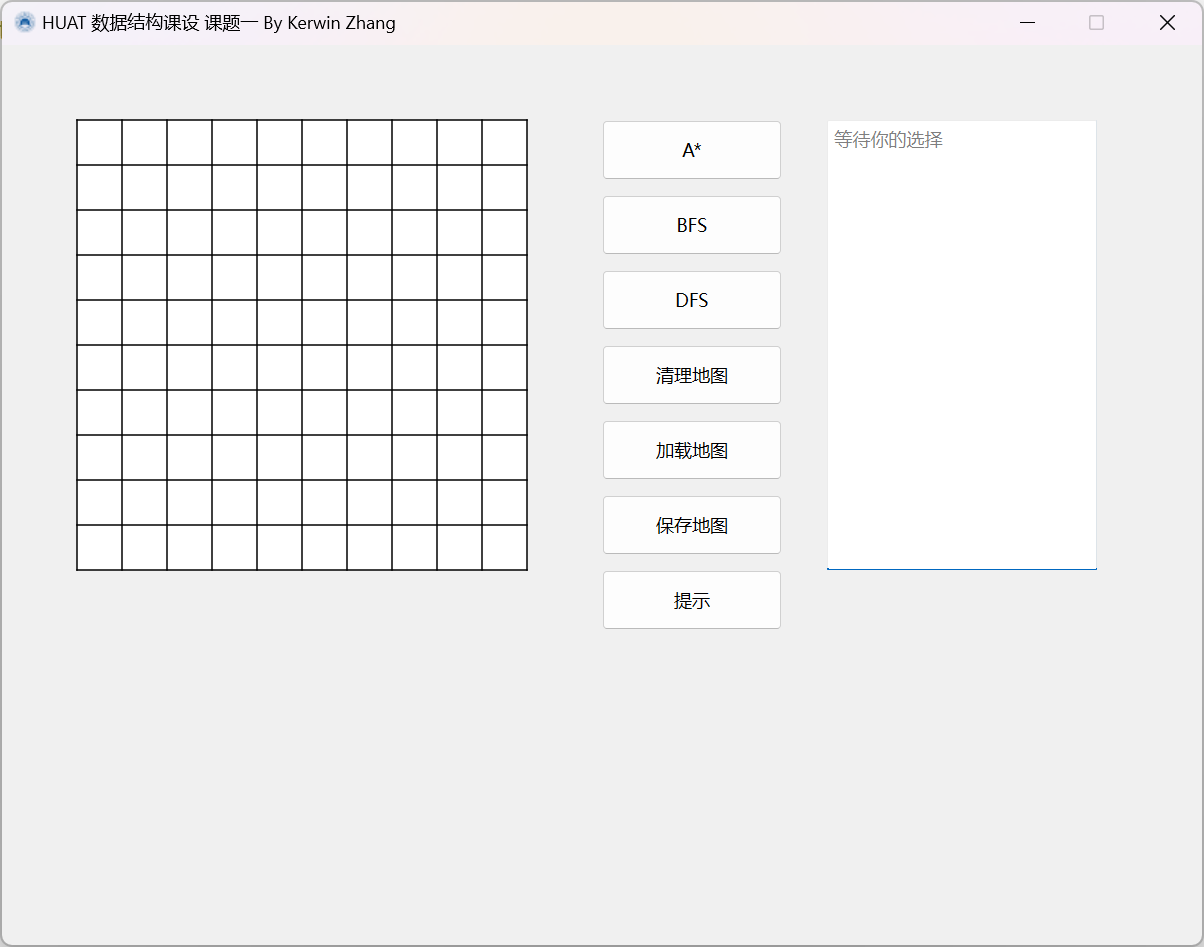
w.show();

return a.exec();

}

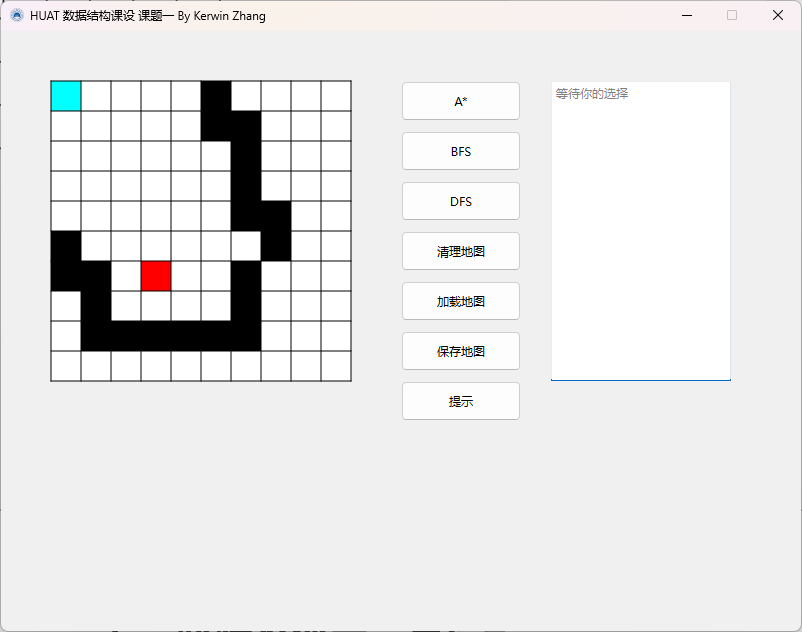
六.运行结果及分析

6.1基础界面展示

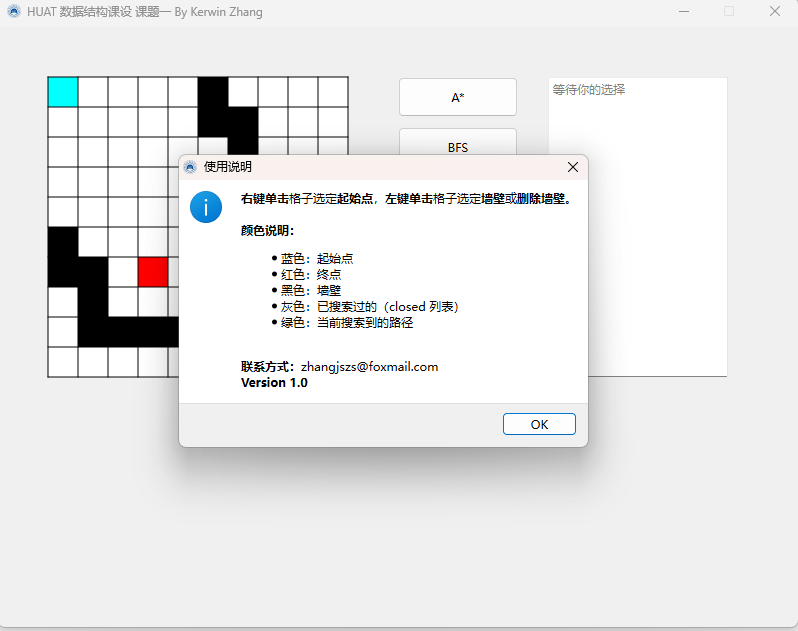


功能展示

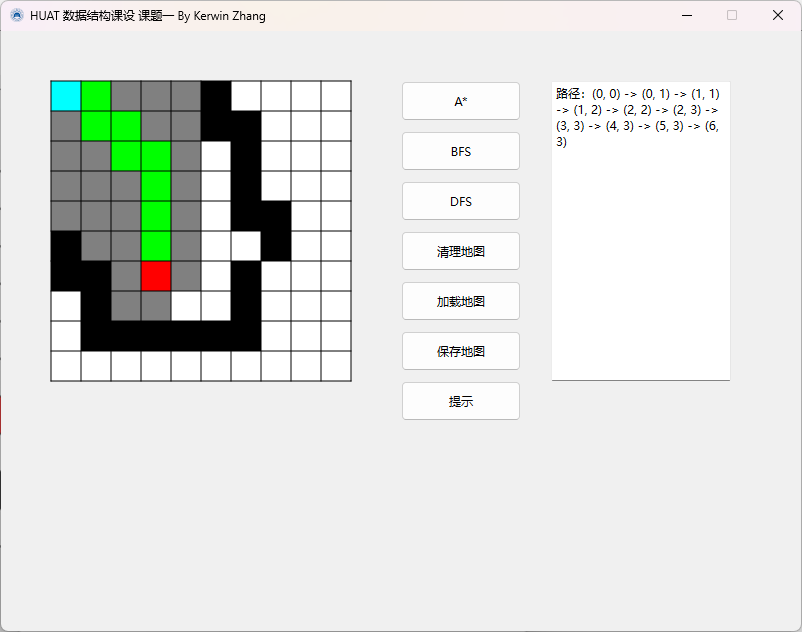
使用左键来进行墙的设置 右键来设置起点和终点



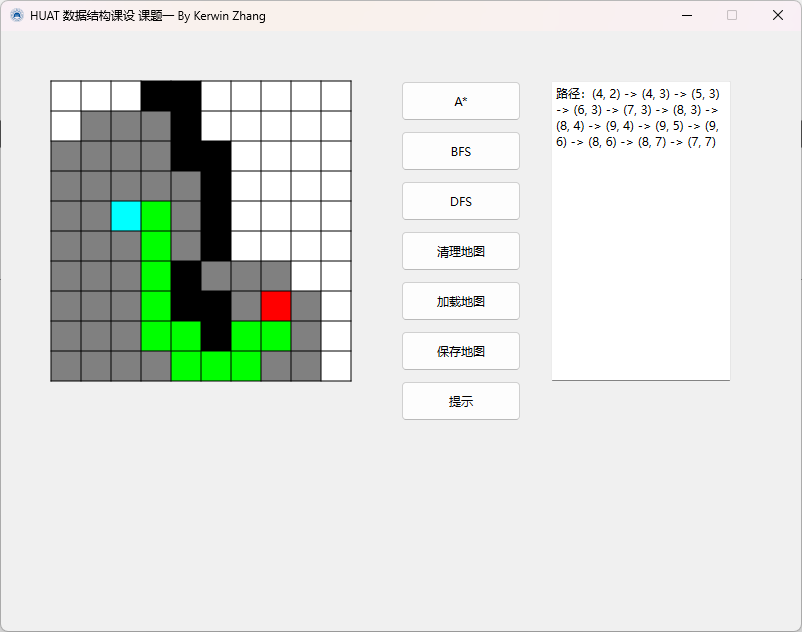
可以点开提示按钮或许基本不信息



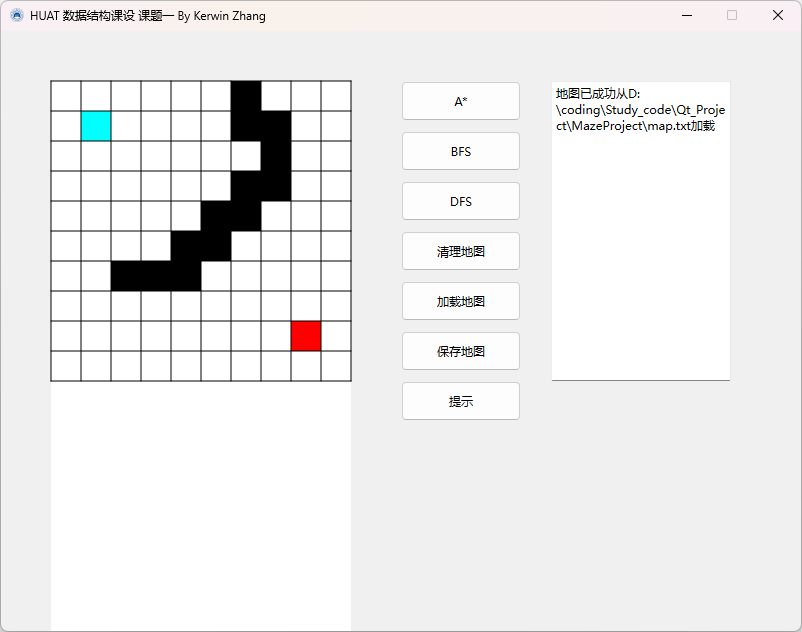
使用A\*搜索

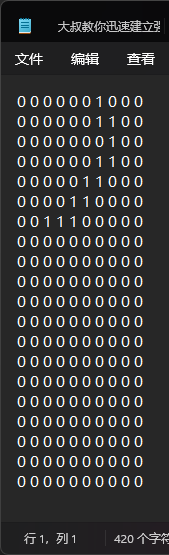
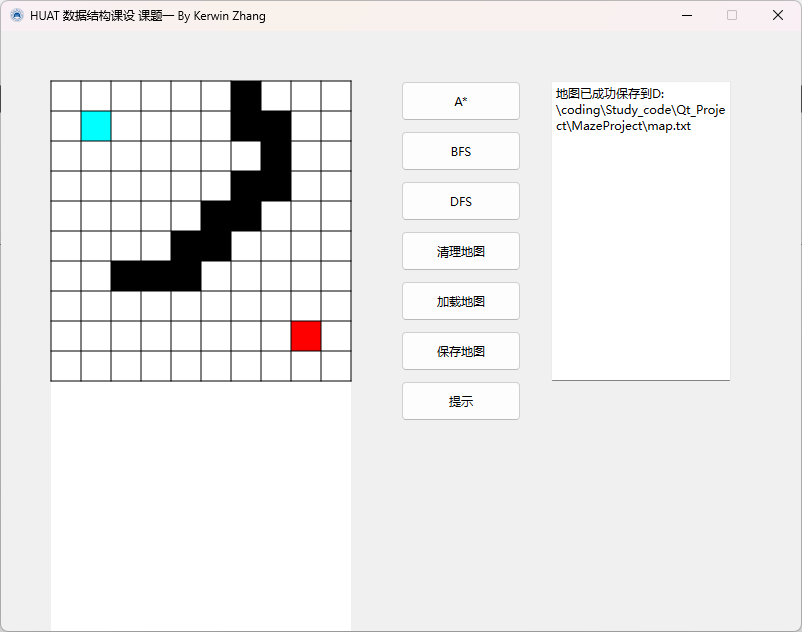


使用BFS搜索



清理地图就不显示了 在每次操作后都会清理地图 加载地图



保存地图

七、收获及体会，总结

过本次数据结构课程设计——基于Qt的迷宫路径探索系统开发，我获得了多方面的宝贵经验和深刻体会，这些不仅巩固了我的理论知识，也极大提升了我的实践技能与问题解决能力。以下是我个人的总结与反思：

理论与实践的融合

理论知识是基础，但将其应用于实际项目中才能真正体会到它的力量。在本次项目中，数据结构与算法不再是抽象的概念，而是变成了迷宫探索中活生生的栈、队列、链表以及A\*等算法的实现。通过亲手实现这些数据结构和算法，我对它们的运行机制、效率特性以及适用场景有了更加深刻的理解。

UI设计与用户体验的重要性

利用Qt框架开发的图形用户界面，让我深刻认识到良好用户界面设计的重要性。直接点击格子设置迷宫、起点和终点的交互方式，极大地提升了用户的参与感和操作的便捷性。同时，动态显示迷宫探索过程和结果，不仅增强了程序的实用性，也使得算法的逻辑变得更加直观易懂，这对于教育和演示来说尤为重要。

算法优化与性能考量

在实现多种寻路算法（DFS、BFS、A\*）的过程中，我深刻体会到算法优化的必要性。A算法通过引入启发式函数，能够在保持高效的同时找到最优路径，这让我认识到在解决实际问题时，选择合适的算法策略是多么重要。同时，我也学到了如何在算法设计阶段就考虑性能因素，如A算法中对移动代价的精细计算，确保了算法在面对大规模迷宫时依然能够快速响应。

代码结构与模块化设计

为了使项目易于维护和扩展，我采用了面向对象的编程思想，设计了清晰的类结构，如CAstar和可能的MazeSolver类。通过合理的类划分和接口设计，我学会了如何有效地组织代码，使其既模块化又具有良好的耦合度。这不仅提升了代码的可读性和可维护性，也为未来添加新功能提供了便利。

问题解决与调试技巧

在开发过程中，我遇到了诸如内存泄漏、逻辑错误等问题，这些问题的解决过程锻炼了我的调试技巧和问题定位能力。我学会了利用调试工具、日志打印以及代码审查等多种手段，快速定位并修复问题，这对我今后解决复杂软件问题有着极大的帮助。

团队合作与沟通

虽然此次项目是个人完成，但反思过程中，我意识到了团队合作的重要性。在实际工作中，有效的沟通和团队协作是项目成功的关键。这次独立完成项目的经验，让我更加明白如何清晰地表达想法、分享进度以及从他人那里获取反馈，这些都是团队合作中不可或缺的技能。

这个迷宫路径探索系统的开发不仅加深了我对数据结构和算法的理解，也让我在软件工程、用户体验设计、问题解决和团队合作等方面获得了宝贵的实践经验。这次经历无疑为我未来的学术研究和职业生涯打下了坚实的基础。