迷宮求解程式：詳細結構與流程

Abstract

本文檔詳細解析了一個用 C++ 編寫的迷宮求解程式，涵蓋所有檔案、類別結構、方法功能，並對關鍵程式碼（特別是 main.cpp）提供逐行解釋。該程式支援自動尋路（使用廣度優先搜尋，BFS）和手動玩家導航，能從文字檔案讀取、顯示和儲存迷宮佈局。本版本保留了前一版本的詳細解釋，同時加入總結性說明，並對 main.cpp 進行功能分段，提升可讀性。

1 引言

迷宮求解程式是一個基於控制台的 C++ 應用程式，允許使用者通過手動輸入或自動演算法解決迷宮問題。程式由 Maze、Move 和 PathFinder 類別組成，並由 main.cpp 協調執行。本文檔將每個類別的結構和方法詳細說明，並對 main.cpp 分段解析。

2 程式結構

程式分為以下主要檔案:

- **標頭檔案**: maze.h、move.h、pathfinder.h，定義類別介面。
- **實現檔案**: maze.cpp、move.cpp、pathfinder.cpp，包含方法實現。
- **主程式**: main.cpp，處理使用者交互和程式流程。
- **迷宮檔案**: mission1.txt、mission2.txt、mission3.txt，定義迷宮佈局。

3 類別與檔案詳細說明

3.1 maze.h 和 maze.cpp

3.1.1 maze.h

maze.h 定義了 Maze 類別，負責管理迷宮的資料結構和狀態，包括常量、座標結構和公開方法。

- **常量**:
  - WALL = '/': 表示牆。

  - `PATH = '-'`: 表示可行走路徑。
  - `WENTPATH = '#'`: 表示走過的路徑。
  - `PLAYER = '0'`: 表示玩家位置。
  - `GOAL = '1'`: 表示目標位置。

- **結構**:
  - `Position`: 包含 `int x, y`，表示迷宮中的座標。

- **公開方法**:
  - `Maze()`: 建構函數，初始化迷宮物件。
  - `readMaze(const std::string& filename)`: 從檔案讀取迷宮。
  - `saveMaze(const std::string& filename, bool restoreOriginal)`: 儲存迷宮。
  - `displayMaze()`: 顯示迷宮。
  - `restoreOriginalMaze()`: 恢復原始迷宮。
  - Getter 和 Setter 方法: 存取迷宮屬性。

- **私有屬性**:
  - `maze_`: 當前迷宮的二維向量。
  - `originalMaze_`: 原始迷宮的二維向量。
  - `player_`: 玩家位置。
  - `goal_`: 目標位置。
  - `positionHistory_`: 位置歷史堆疊。
  - `moveHistory_`: 移動方向堆疊。
  - `rows_, cols_`: 迷宮尺寸。
  - `isGameOver_`: 是否到達目標。

3.1.2 maze.cpp

`maze.cpp` 實現了 `Maze` 類別的所有方法，負責迷宮的初始化、檔案讀寫、顯示和狀態管理。以下是每個方法的詳細解釋:

- **`Maze::Maze()`**:

```
Maze::Maze() : rows_(0), cols_(0), isGameOver_(false) {}

```

  初始化迷宮物件，將尺寸設為 0，遊戲狀態設為未結束。
  - `rows_` 和 `cols_`: 設為 0，表示迷宮尚未載入。
  - `isGameOver_`: 設為 `false`，遊戲開始時未到達目標。

- **`readMaze`**:

```
bool Maze::readMaze(const std::string& filename) {
    std::ifstream file(filename);
    if (!file) {
        std::cout << "Error opening file!" << std::endl;
        return false;
    }
    file >> rows_ >> cols_;
    maze_.resize(rows_, std::vector<char>(cols_));
    originalMaze_.resize(rows_, std::vector<char>(cols_));
    for (int i = 0; i < rows_; ++i) {
        for (int j = 0; j < cols_; ++j) {
            file >> maze_[i][j];
            originalMaze_[i][j] = maze_[i][j];
            if (maze_[i][j] == PLAYER) {
                player_ = {i, j};
            } else if (maze_[i][j] == GOAL) {
                goal_ = {i, j};
            }
        }
    }
    file.close();
    return true;
}

```

從指定檔案讀取迷宮佈局，設置尺寸並儲存玩家和目標位置。

- `std::ifstream file(filename)`：開啟檔案，若失敗則返回 `false`。
- `file >> rows_ >> cols_`：讀取迷宮行數和列數。
- `maze_.resize(...)`：動態調整 `maze_` 的大小。
- `for` 迴圈：逐格讀取字符，同時記錄 `player_` 和 `goal_`。

- **saveMaze**：

```
void Maze::saveMaze(const std::string& filename, bool
   restoreOriginal) {
    std::ofstream file(filename);
    file << rows_ << " " << cols_ << std::endl;
    for (int i = 0; i < rows_; ++i) {
        for (int j = 0; j < cols_; ++j) {
            file << (restoreOriginal ? originalMaze_[i][j] :
   maze_[i][j]);
        }
        file << std::endl;
    }
    file.close();
}

```

將迷宮儲存到檔案，可選擇儲存當前或原始狀態。

- `restoreOriginal ? originalMaze_[i][j] : maze_[i][j]`：條件運算子決定寫入哪個迷宮。
- `file << std::endl`：每行結束後換行。

- **displayMaze**：

```
void Maze::displayMaze() const {
    for (int i = 0; i < rows_; ++i) {
        for (int j = 0; j < cols_; ++j) {
            std::cout << maze_[i][j];
        }
        std::cout << std::endl;
    }
}

```

在控制台顯示當前迷宮佈局。

- `const`：方法不修改物件狀態。
- 雙重 `for` 迴圈：逐行逐列輸出字符。

- **restoreOriginalMaze**：

```
void Maze::restoreOriginalMaze() {
    maze_ = originalMaze_;
    for (int i = 0; i < rows_; ++i) {
        for (int j = 0; j < cols_; ++j) {
            if (originalMaze_[i][j] == PLAYER) {
                player_ = {i, j};
            }
        }
    }
    isGameOver_ = false;
    while (!positionHistory_.empty()) positionHistory_.pop();
    while (!moveHistory_.empty()) moveHistory_.pop();
}

```

將迷宮恢復到初始狀態，清空歷史紀錄。

- `maze_ = originalMaze_`：複製原始迷宮。
- `while (!positionHistory_.empty())`：清空堆疊，移除所有歷史資料。

3.2 move.h 和 move.cpp

3.2.1 move.h

`move.h` 定義了 `Move` 類別，處理玩家的移動邏輯。

- **公開方法**：
  - `Move(Maze& maze)`：建構函數，綁定迷宮。

- `isValidMove(int x, int y)`：檢查移動是否有效。
- `isOppositeMove(char current, char last)`：檢查是否為相反移動。
- `movePlayer(char direction)`：執行移動。
- `undoMove()`：撤銷移動。

- **私有屬性**：
  - `maze_`：對迷宮的引用。

3.2.2 move.cpp

`move.cpp` 實現了移動相關的功能，包括驗證、執行和撤銷移動。

- **Move::Move**：

```
Move::Move(Maze& maze) : maze_(maze) {}

```

初始化 `Move` 物件，綁定迷宮引用。

- `maze_`：通過引用綁定，避免複製迷宮。

- **isValidMove**：

```
bool Move::isValidMove(int x, int y) const {
    return x >= 0 && x < maze_.getRows() && y >= 0 && y < maze_.
   getCols() && maze_.getMaze()[x][y] != Maze::WALL;
}

```

驗證目標座標是否在範圍內且不是牆。

- `maze_.getMaze()[x][y] != Maze::WALL`：確保不是障礙物。

- **isOppositeMove**：

```
bool Move::isOppositeMove(char current, char last) const {
    return (current == 'w' && last == 's') || (current == 's' &&
    last == 'w') ||
           (current == 'a' && last == 'd') || (current == 'd' &&
    last == 'a');
}

```

檢查當前移動是否與上一次相反。

- `return (...)`：列舉所有相反方向對（如上下、左右）。

- **movePlayer**：

```
bool Move::movePlayer(char direction) {
    Maze::Position newPos = maze_.getPlayer();
    if (direction == 'w') newPos.x--;
    else if (direction == 's') newPos.x++;
    else if (direction == 'a') newPos.y--;
```

```
    else if (direction == 'd') newPos.y++;
    if (maze_.getMaze()[newPos.x][newPos.y] == Maze::GOAL) {
        maze_.setIsGameOver(true);
    }
    if (isValidMove(newPos.x, newPos.y)) {
        maze_.getPositionHistory().push(maze_.getPlayer());
        maze_.getMoveHistory().push(direction);
        maze_.setMazeCell(maze_.getPlayer().x, maze_.getPlayer()
   .y, Maze::WENTPATH);
        maze_.setPlayer(newPos);
        maze_.setMazeCell(newPos.x, newPos.y, maze_.
   getIsGameOver() ? Maze::GOAL : Maze::PLAYER);
        return true;
    }
    return false;
}

```

根據方向移動玩家，更新迷宮狀態並記錄歷史。

- newPos：計算新位置。
- if (isValidMove(...))：若有效，更新歷史和迷宮格子。

- **undoMove**：

```
void Move::undoMove() {
    if (maze_.getPositionHistory().empty()) {
        std::cout << "There are no undoable moves!" << std::endl
   ;
        std::this_thread::sleep_for(std::chrono::seconds(1));
        return;
    }
    maze_.setMazeCell(maze_.getPlayer().x, maze_.getPlayer().y,
   Maze::PATH);
    maze_.setPlayer(maze_.getPositionHistory().top());
    maze_.getPositionHistory().pop();
    maze_.getMoveHistory().pop();
    maze_.setMazeCell(maze_.getPlayer().x, maze_.getPlayer().y,
   Maze::PLAYER);
}

```

撤銷上一次移動，恢復前一狀態。

- if (maze_.getPositionHistory().empty())：檢查是否有可撤銷的移動。
- pop()：移除堆疊頂部資料。

3.3 pathfinder.h 和 pathfinder.cpp

3.3.1 pathfinder.h

pathfinder.h 定義了 PathFinder 類別，負責自動尋找最短路徑。

- **公開方法**:
  - `PathFinder(Maze& maze)`：建構函數。
  - `findShortestPath()`：返回最短路徑堆疊。
- **私有屬性**:
  - `maze_`：迷宮引用。

3.3.2 pathfinder.cpp

pathfinder.cpp 使用廣度優先搜尋（BFS）實現最短路徑查找。

- **findShortestPath**:

```
std::stack<Maze::Position> PathFinder::findShortestPath() {
    std::vector<std::vector<bool>> visited(maze_.getRows(), std::vector<bool>(maze_.getCols(), false));
    std::vector<std::vector<Maze::Position>> parent(maze_.getRows(), std::vector<Maze::Position>(maze_.getCols(), {-1, -1}));
    std::queue<Maze::Position> q;
    Maze::Position player = maze_.getPlayer();
    visited[player.x][player.y] = true;
    q.push(player);
    while (!q.empty()) {
        Maze::Position current = q.front();
        q.pop();
        if (current.x == maze_.getGoal().x && current.y == maze_.getGoal().y) {
            maze_.setIsGameOver(true);
            break;
        }
        std::vector<Maze::Position> directions = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
        for (const auto& dir : directions) {
            int newX = current.x + dir.x;
            int newY = current.y + dir.y;
            if (newX >= 0 && newX < maze_.getRows() && newY >= 0 && newY < maze_.getCols() &&
                maze_.getMaze()[newX][newY] != Maze::WALL && !visited[newX][newY]) {
                visited[newX][newY] = true;
                parent[newX][newY] = current;
                q.push({newX, newY});
            }
        }
    }
    if (maze_.getIsGameOver()) {
        std::stack<Maze::Position> path;
        Maze::Position current = maze_.getGoal();
        while (current.x != -1 && current.y != -1) {
```

```
            path.push(current);
            current = parent[current.x][current.y];
        }
        return path;
    } else {
        std::stack<Maze::Position> emptyStack;
        return emptyStack;
    }
}

```

使用 BFS 尋找從玩家到目標的最短路徑，返回路徑堆疊。

- `visited`：記錄已訪問節點。
- `parent`：儲存路徑的父節點。
- `while (!q.empty())`：BFS 主迴圈，探索所有可能路徑。
- `path.push(current)`：回溯構建路徑。

3.4 main.cpp

`main.cpp` 是程式入口，負責使用者交互和流程控制。以下按功能分段解釋：

3.4.1 輔助函數

- **`waiting3s`**：

```
void waiting3s(string message) {
    for (int i = 0; i <= 3; ++i) {
        cout << "\r" << message;
        for (int j = 0; j < i; ++j) {
            cout << ".";
        }
        cout.flush();
        this_thread::sleep_for(chrono::seconds(1));
    }
    cout << endl;
}

```

顯示 3 秒動態等待訊息。

- `–°`：回到行首，實現動畫效果。

- **`waitingCountDown`**：

```
void waitingCountDown(string message1, int seconds, string
   message2) {
    for (int i = 0; i <= seconds; ++i) {
        cout << "\r" << message1 << seconds - i << message2;
        for (int j = 0; j < i; ++j) {
            cout << ".";
        }
```

```
        cout.flush();
        this_thread::sleep_for(chrono::seconds(1));
    }
    cout << endl;
}

```

顯示自訂秒數的倒數動畫。

3.4.2 初始化

```
cout << "Enter the name of the maze map document (remember to add \".txt\"): ";
string filename;
cin >> filename;
Maze maze;
if (!maze.readMaze(filename)) {
    cout << "Failed to read maze file." << endl;
    cout << "-----------------------------------------------" << endl;
    waitingCountDown("The program will exit within ", 3, " seconds");
    return 1;
}
Move move(maze);
PathFinder pathFinder(maze);
```

提示輸入檔案名稱，初始化迷宮和相關物件，若失敗則退出。

- `if (!maze.readMaze(filename))`：讀取失敗時顯示錯誤並倒數退出。

3.4.3 主迴圈

```
while (true) {
    maze.restoreOriginalMaze();
    cout << "0 : Player, 1 : Goal" << endl;
    maze.displayMaze();
    cout << "Choose mode (1: Auto, 2: Manual, 3: Exit): ";
    int choosedMode;
    cin >> choosedMode;
    if (choosedMode == 1) { /* 自動模式 */ }
    else if (choosedMode == 2) { /* 手動模式 */ }
    else if (choosedMode == 3) { break; }
    else {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "Invalid choice! Try again." << endl;
        this_thread::sleep_for(chrono::seconds(1));
        cout << "\033[2J\033[1;1H" << flush;
        continue;
    }
}
```

顯示迷宮並提供模式選擇，根據輸入進入不同分支。

- `cin.clear()`：處理無效輸入。
- `33[2J33[1;1H`：清空控制台。

3.4.4 自動模式

```
if (choosedMode == 1) {
    cout << "-----------------------------------------------" << endl;
    waiting3s("Computing shortest path");
    stack<Maze::Position> path = pathFinder.findShortestPath();
    int steps = path.size() - 1;
    if (path.empty()) {
        cout << "No path found!" << endl;
        break;
    } else {
        while (!path.empty()) {
            cout << "\033[2J\033[1;1H" << flush;
            cout << "0 : Player, 1 : Goal" << endl;
            Maze::Position next = path.top();
            path.pop();
            maze.setMazeCell(maze.getPlayer().x, maze.getPlayer().y, Maze::WENTPATH);
            maze.setPlayer(next);
            maze.setMazeCell(next.x, next.y, ((path.size() == 0) ? Maze::GOAL : Maze::PLAYER));
            maze.displayMaze();
            cout << "Step: " << (steps - path.size()) << "/" << steps << endl;
            this_thread::sleep_for(chrono::milliseconds(300));
        }
        cout << "Reached the goal!" << endl;
        cout << "Total steps: " << steps << endl;
    }
    maze.saveMaze(filename, true);
    cout << "-----------------------------------------------" << endl;
    waitingCountDown("The program will return to the main screen in ", 3, " seconds");
    cout << "\033[2J\033[1;1H" << flush;
    continue;
}
```

計算並動畫顯示最短路徑，成功後返回選單。

- `while (!path.empty())`：逐步顯示路徑。
- `steps - path.size()`：計算當前步數。

3.4.5 手動模式

```
else if (choosedMode == 2) {
    while (true) {
        cout << "\033[2J\033[1;1H" << flush;
        cout << "0 : Player, 1 : Goal" << endl;
        maze.displayMaze();
        cout << "Step: " << maze.getPositionHistory().size() << endl;
        cout << "Use W(Up), S(Down), A(Left), D(Right) to move, U to undo, Q to exit: ";
        char input;
        cin >> input;
        input = tolower(input);
        if (input != 'w' && input != 's' && input != 'a' && input != 'd' && input != 'u' && input != 'q') {
            cout << "Invalid input! Please use W, S, A, D, U, or Q." << endl;
            this_thread::sleep_for(chrono::seconds(1));
            continue;
        }
        bool reachGoal = false;
        if (input == 'q') { /* 退出 */ }
        else if (input == 'u') { /* 撤銷 */ }
        else if (!maze.getMoveHistory().empty() && move.isOppositeMove(input, maze.getMoveHistory().top())) { /* 相反移動 */ }
        else if (input == 'w' || input == 's' || input == 'a' || input == 'd') { /* 移動 */ }
        if (reachGoal) { /* 到達目標 */ }
    }
    maze.saveMaze(filename, true);
    cout << "-----------------------------------------------" << endl;
    waitingCountDown("The program will return to the main screen in ", 3, " seconds");
    cout << "\033[2J\033[1;1H" << flush;
    continue;
}
```

處理玩家輸入，更新迷宮並提供撤銷功能。

- `tolower(input)`：統一輸入為小寫。
- `reachGoal`：標記是否到達目標。

3.4.6 程式結束

```
maze.saveMaze(filename, true);
cout << "-----------------------------------------------" << endl;
waitingCountDown("The program will exit within ", 3, " seconds");
return 0;
```

儲存迷宮並退出程式。

4 結論

本程式通過模組化設計實現了迷宮求解功能，`main.cpp` 的分段解釋提升了可讀性，詳細的類別說明則有助於理解實現細節。