# 迷宮推箱子遊戲專案更新說明文件

## 目錄



## 1. 引言

本文件旨在詳細說明迷宮遊戲專案從先前 HW3 版本演進至今的各項重要更新。主要的變革是引入了「推箱子」的核心玩法，取代了單純的迷宮尋路。這一轉變涉及到遊戲邏輯、資料結構以及使用者互動方式的多方面調整。本文件將逐一解析這些變更，特別是 `move.cpp` 中關於移動和復原邏輯的重大修改。

# 2. 遊戲運作原理

## 2.1 基本目標

玩家 (以 '0' 表示) 的目標是將箱子 (以 '1' 表示) 推到迷宮中的目標位置 (以 '2' 表示)。

## 2.2 控制方式

遊戲透過鍵盤進行控制：

- **W**: 向上移動
- **S**: 向下移動
- **A**: 向左移動
- **D**: 向右移動
- **U**: 復原上一步移動
- **Q**: 退出遊戲

## 2.3 核心機制

- **玩家移動**: 玩家可以在迷宮的通路 ('-') 上自由移動。
- **推動箱子**:
    - 當玩家嘗試移動到箱子所在的位置時，如果箱子的前方（即玩家推動的方向）是通路或目標點，則箱子會被推動一格。
    - 如果箱子前方是牆壁 ('/') 或迷宮邊界，則玩家和箱子都無法移動。
- **牆壁與邊界**: 玩家和箱子都不能穿過牆壁或移出迷宮邊界。
- **復原 (Undo)**: 玩家可以透過 'U' 鍵撤銷上一步的移動，包括玩家的移動以及箱子可能發生的位移。歷史紀錄會追蹤玩家位置、箱子位置以及箱子是否被移動。
- **卡死判斷**: `main.cpp` 中新增了對箱子是否陷入特定死角（例如兩面都是牆的角落）的初步判斷邏輯。如果箱子卡死：
    - 若無移動歷史（遊戲剛開始或無法復原），則提示迷宮無法解決。
    - 若有移動歷史，則會自動執行一次復原操作。
- **遊戲結束**: 當箱子被推到目標位置時，遊戲達成目標。

---

# 3. 程式碼結構與各檔案功能

## 3.1 `main.cpp`

- **作用**: 程式主入口點，負責整個遊戲的流程控制。
- **功能**:
    - 提示使用者輸入迷宮地圖檔案名稱。
    - 讀取並初始化迷宮。

- 提供模式選擇：自動模式（目前未完成）、手動模式、退出。
- 在手動模式下，包含主遊戲迴圈：
  - 顯示迷宮狀態、玩家 ('0') 與目標 ('1'，此處應為箱子 '1'，目標 '2') 的圖例。
  - 顯示當前步數。
  - 實作箱子卡死判斷邏輯。
  - 接收並處理使用者輸入 (W, S, A, D, U, Q)。
  - 呼叫 `Move` 物件的方法來執行玩家移動或復原操作。
  - 判斷遊戲是否達成目標。
  - 處理遊戲結束或返回主選單的流程。
- 程式結束前儲存迷宮狀態（可選擇是否恢復為原始迷宮）。
- 包含輔助函式如 `waiting3s` 和 `waitingCountDown`，用於改善使用者體驗。

### 3.2 `maze.h`

- **作用**: `Maze` 類別的宣告（介面）。
- **功能**:
  - 定義迷宮中各種元素的符號常量 (WALL, PATH, PLAYER, BOX, GOAL)。
  - 定義 `Position` 結構體，用於表示座標。
  - 宣告 `Maze` 類別的成員變數，包括：
    - `maze_`：目前的迷宮佈局 (二維 `char` 向量)。
    - `originalMaze_`：原始迷宮佈局，用於重置。
    - `player_`, `box_`, `goal_`：分別儲存玩家、箱子和目標的位置。
    - `playerPositionHistory_`, `boxPositionHistory_`, `boxIsMovedHistory_`：用於儲存移動歷史的堆疊，以支援復原功能。
    - `rows_`, `cols_`：迷宮的尺寸。
    - `isGameOver_`：遊戲是否結束的標記。
  - 宣告 `Maze` 類別的成員函式，包括：
    - 建構函式 `Maze()`。
    - `readMaze()`：從檔案讀取迷宮。
    - `saveMaze()`：將迷宮儲存到檔案。
    - `displayMaze()`：在控制台顯示迷宮。
    - `restoreOriginalMaze()`：將迷宮恢復到初始狀態。
    - 各種 `get` 和 `set` 函式，用於存取和修改迷宮的狀態與屬性 (例如 `getPlayer()`, `setBox()`, `getMaze()`, `getBoxPositionHistory()`)。

### 3.3 `maze.cpp`

- **作用**: `Maze` 類別的實作。
- **功能**:

- `Maze::Maze()`：建構函式，初始化行數、列數和遊戲結束狀態。
- `Maze::readMaze(const std::string& filename)`：
  - 開啟並讀取指定的迷宮檔案。
  - 讀取迷宮的行數和列數。
  - 調整 `maze_` 和 `originalMaze_` 的大小。
  - 逐字元讀取迷宮佈局，並儲存到 `maze_` 和 `originalMaze_`。
  - 在讀取過程中，記錄玩家（`player_`）、箱子（`box_`）和目標（`goal_`）的初始位置。
- `Maze::saveMaze(const std::string& filename, bool restoreOriginal)`：
  - 開啟檔案並將迷宮資料寫入。
  - 根據 `restoreOriginal` 參數決定儲存目前迷宮狀態還是原始迷宮狀態。
- `Maze::displayMaze() const`：
  - 迭代遍歷 `maze_` 並將每個字元輸出到控制台，以顯示目前迷宮樣貌。
- `Maze::restoreOriginalMaze()`：
  - 將 `maze_` 恢復為 `originalMaze_` 的內容。
  - 重新從 `originalMaze_` 中找到並設定玩家和箱子的初始位置。
  - 重置 `isGameOver_` 為 `false`。
  - 清空所有歷史紀錄堆疊（`playerPositionHistory_`, `boxPositionHistory_`, `boxIsMovedHistory_`）。

### 3.4 move.h

- **作用**: `Move` 類別的宣告（介面）。
- **功能**:
  - 包含對 `maze.h` 的引用，因為 `Move` 類別需要操作 `Maze` 物件。
  - 宣告 `Move` 類別，其成員變數 `maze_` 是一個對 `Maze` 物件的引用。
  - 宣告 `Move` 類別的成員函式:
    - `Move(Maze& maze)`：建構函式，接收一個 `Maze` 物件的引用。
    - `isValidMove(int x, int y) const`：檢查指定座標是否為迷宮內的有效移動位置（非牆壁、未越界）。
    - `movePlayer(char direction)`：處理玩家的移動請求，包含推箱子邏輯。
    - `undoMove()`：執行復原上一步操作的邏輯。

### 3.5 move.cpp

- **作用**: `Move` 類別的實作，包含遊戲中最核心的移動和推箱子邏輯。
- **功能**:
  - `Move::Move(Maze& maze) : maze_(maze)`：建構函式，初始化對 `Maze` 物件的引用。

- `Move::isValidMove(int x, int y) const`：
  - 判斷座標 `(x, y)` 是否在迷宮的行列範圍內。
  - 判斷該座標在 `maze_.getMaze()` 中是否不為牆壁 ( `Maze::WALL` )。
- `Move::movePlayer(char direction)`：詳細邏輯見 4.4.2 `movePlayer()` 和 `undoMove()` 邏輯的重大修改。
- `Move::undoMove()`：詳細邏輯見 4.4.2 `movePlayer()` 和 `undoMove()` 邏輯的重大修改。

---

# 4. 與 HW3 版本之主要差異

以下將根據您提供的更動提示，詳細說明各項變更的內容與原因。

## 4.1 `main.cpp` 的變更

- **1.1. 移除 `#include "pathfinder.h"` 和 `PathFinder pathFinder(maze);`**
  - **原因**: 由於自動尋路模式 (auto mode) 的功能尚未完成或已決定暫不實作。
  - **影響**: 程式不再包含或試圖使用 `PathFinder` 類別，減少了不必要的程式碼和依賴。
- **1.2. 更改 `choosedMode == 1` (自動模式) 的程式碼**
  - **變更**: 當使用者選擇模式 1 (自動模式) 時，程式現在會顯示「此功能尚未完成，3 秒後返回主畫面」的訊息，然後等待 3 秒並清屏返回模式選擇介面。
  - **原因**: 自動模式功能未實作，提供使用者明確提示。
  - **影響**: 避免了先前版本中可能存在的未完成功能的錯誤或未知行為。
- **1.3. 更改「箱子卡死」(原提示為 "Detected opposite move") 的邏輯**
  - **變更**: 新增了偵測箱子是否被困在角落的邏輯。具體判斷條件是檢查箱子 ( `maze.getBox()` ) 的上/下與左/右是否同時為牆壁。

cpp

```cpp
// 位於 main.cpp 的箱子卡死判斷片段
int x = maze.getBox().x;
int y = maze.getBox().y;
auto& grid = maze.getMaze();
if ((grid[x - 1][y] == Maze::WALL && grid[x][y - 1] == Maze::WALL) ||
    (grid[x - 1][y] == Maze::WALL && grid[x][y + 1] == Maze::WALL) ||
    (grid[x + 1][y] == Maze::WALL && grid[x][y - 1] == Maze::WALL) ||
    (grid[x + 1][y] == Maze::WALL && grid[x][y + 1] == Maze::WALL)) {
    if (maze.getPlayerPositionHistory().size() == 0) { // 如果沒有歷史紀錄
        cout << "The box is stuck! The maze cannot be solved." << endl;
        break; // 跳出當前遊戲迴圈
    } else { // 如果有歷史紀錄
        waiting3s("The box is stuck! Auto undoing");
        move.undoMove(); // 自動執行一次復原
        continue; // 繼續遊戲迴圈的下一次迭代
    }
}
```

- **原因**: 提供更智能的遊戲輔助。當箱子陷入特定死角時，如果無法復原，則提示遊戲無法解決；如果可以復原，則自動幫助玩家撤銷一步，避免玩家需要手動退出或卡在無法解決的局面。
- **影響**: 提升了遊戲的容錯性和使用者體驗。需要注意的是，此判斷僅涵蓋了箱子在角落的情況，並非所有箱子卡死的狀況。

## 4.2 `maze.cpp` 的變更

- **2.1. `readMaze()` 函數**
  - **2.1.1. 新增 `box_` (箱子) 位置的設定**
    - **變更**: 在 `readMaze()` 函數中，當讀取迷宮檔案的字元時，如果遇到代表箱子的符號 ( `Maze::BOX` )，則將其座標記錄到 `box_` 成員變數中。

      cpp

      ```cpp
      // 位於 maze.cpp 的 readMaze() 相關片段
      // ...
      else if (maze_[i][j] == BOX) { // BOX 是 Maze::BOX
          box_ = {i, j};
      }
      // ...

      ```

    - **原因**: 為了正確初始化箱子在迷宮中的起始位置。
    - **影響**: 遊戲開始時能正確載入箱子的位置。
- **2.2. `restoreOriginalMaze()` 函數**
  - **2.2.1. 新增 `box_` (箱子) 位置的重置**
    - **變更**: 在 `restoreOriginalMaze()` 中，遍歷 `originalMaze_` ，當找到代表箱子的符號時，更新 `box_` 的位置為其在原始迷宮中的位置。

      cpp

      ```cpp
      // 位於 maze.cpp 的 restoreOriginalMaze() 相關片段
      // ...
      else if (originalMaze_[i][j] == BOX) { // BOX 是 Maze::BOX
          box_ = {i, j};
      }
      // ...

      ```

    - **原因**: 確保在重置迷宮時，箱子也能恢復到其初始設定的位置。
    - **影響**: 遊戲重新開始或重置時，箱子狀態正確。
  - **2.2.2. 移除** `while (!moveHistory_.empty()) moveHistory_.pop();`
    - **原因**: `moveHistory_` 成員變數已從 `Maze` 類別中移除 (詳見 4.3.2.2)。
    - **影響**: 程式碼更簡潔，移除了對不存在成員的操作。
  - **2.2.3. 新增** `while (!boxPositionHistory_.empty()) boxPositionHistory_.pop();` **和** `while (!boxIsMovedHistory_.empty())`

```
boxIsMovedHistory_.pop();
```

- **變更**: 在 `restoreOriginalMaze()` 中，新增了清空 `boxPositionHistory_` (箱子位置歷史) 和 `boxIsMovedHistory_` (箱子是否移動歷史) 堆疊的程式碼。
- **原因**: 配合新的復原機制，在重置迷宮時，必須清除所有與箱子移動相關的歷史紀錄，以確保下一次遊戲的歷史紀錄是乾淨的。
- **影響**: 保證了遊戲重置功能的完整性和正確性。

## 4.3 `maze.h` 的變更

- **3.1. Public 成員**
  - **3.1.1. 移除 `WENTPATH` 部分，新增 `BOX`**
    - **變更**: `Maze` 類別的靜態常量中，代表箱子的符號從可能存在的 `WENTPATH` (或其他) 更改為 `BOX = '1'`。原提示中 "Removed the "WENTPATH" part" 指的是可能移除了之前用於標記已走過路徑的符號，而 "added "BOX"" 指的是明確定義了箱子的符號。
    - **原因**: `WENTPATH` (如果之前用於此目的) 的概念不適用於推箱子遊戲中箱子的角色。需要一個明確的符號來代表箱子本身。
    - **影響**: 迷宮中箱子的表示更加清晰和標準化。
  - **3.1.2. 新增 `getBox()` 和 `setBox()`**
    - **變更**: 新增了 `getBox()` 成員函式以獲取箱子的當前位置，以及 `setBox()` 成員函式以設定箱子的位置。

      cpp

      ```cpp
      // 位於 maze.h
      Position getBox() const { return box_; }
      void setBox(const Position& pos) { box_ = pos; }
      ```

    - **原因**: 為了讓外部類別 (主要是 `Move` 類別) 能夠存取和修改箱子的狀態。
    - **影響**: 提供了必要的介面來操作箱子，是實現推箱子功能的基礎。
  - **3.1.3. 移除 `getMoveHistory()` (不再需要)，新增 `getBoxPositionHistory()` 和 `getBoxIsMovedHistory()` (用於復原移動)**
    - **變更**:
      - 移除了可能存在的 `getMoveHistory()` 函式。
      - 新增了 `getBoxPositionHistory()` 函式，返回對 `boxPositionHistory_` 堆疊的引用。
      - 新增了 `getBoxIsMovedHistory()` 函式，返回對 `boxIsMovedHistory_` 堆疊的引用。

      cpp

      ```cpp
      // 位於 maze.h
      std::stack<Position>& getBoxPositionHistory() { return boxPositionHistory_; }
      ```

```
std::stack<bool>& getBoxIsMovedHistory() { return boxIsMovedHistory_; }
// playerPositionHistory_ 的 get 函數也存在
std::stack<Position>& getPlayerPositionHistory() { return playerPositionHistory_;
```

        - **原因**:
            - 舊的 `moveHistory_` (如果指的是單純的玩家移動方向或位置歷史) 可能不足以處理包含箱子移動的復原邏輯。
            - 新的復原機制需要分別追蹤玩家的位置歷史、箱子的位置歷史以及每一步中箱子是否確實被移動了。
        - **影響**: 為 `Move` 類別中的 `undoMove()` 函式提供了必要的歷史數據。
- **3.2. Private 成員**
    - **3.2.1. 新增** `box_`
        - **變更**: 在 `Maze` 類別的私有成員中新增了 `Position box_;` 。
        - **原因**: 用於儲存箱子在迷宮中的當前座標。
        - **影響**: `Maze` 類別現在可以直接管理箱子的狀態。
    - **3.2.2. 移除** `moveHistory_` **(不再需要)，新增** `boxPositionHistory_` **和** `boxIsMovedHistory_` **(用於復原移動)**
        - **變更**:
            - 移除了私有成員 `moveHistory_` (假設其類型和用途不再適合新邏輯)。
            - 新增了 `std::stack<Position> boxPositionHistory_;` 用於儲存箱子在每次移動前的歷史位置。
            - 新增了 `std::stack<bool> boxIsMovedHistory_;` 用於儲存每次移動時箱子是否被推動的布林值。
            - (註: `playerPositionHistory_` 仍然保留，用於儲存玩家的歷史位置。)
        - **原因**: 理由同 4.3.1.3。這是實現複雜復原功能的後端資料結構支持。
        - **影響**: `Maze` 類別內部現在維護了更詳細的狀態歷史，專門用於推箱子情境下的復原。

## 4.4 `move.cpp` 的變更 (核心變更)

這是專案中最重要的變更，因為它直接關係到推箱子玩法的實現。

- **4.1. 移除** `isOppositeMove()`
    - **原因**: `isOppositeMove()` 函數 (如果其功能是檢測玩家是否在原地來回移動或類似的簡單移動模式) 在推箱子遊戲中可能不再適用或其功能已被新的遊戲邏輯 (如箱子卡死判斷) 所間接取代。推箱子遊戲的有效移動判斷更為複雜。
    - **影響**: 簡化了 `Move` 類別，移除了不再需要的邏輯。
- **4.4.2.** `movePlayer()` **和** `undoMove()` **邏輯的重大修改**

`Move::movePlayer(char direction)` 的新邏輯

此函數現在需要處理玩家移動以及可能伴隨的箱子移動。

1. **計算預期新位置**:
    - 根據輸入的 `direction` (w, a, s, d)，計算玩家的預期新位置 `newPlayerPos`。
    - 同時，暫時將箱子的預期新位置 `newBoxPos` 設為箱子當前位置 `maze_.getBox()`。
    - `isBoxMoved` 標記初始化為 `false`。
2. **檢查是否推動箱子**:
    - 判斷玩家的 `newPlayerPos` 是否與當前箱子 `maze_.getMaze()[newPlayerPos.x][newPlayerPos.y] == Maze::BOX` 的位置重疊。
    - 如果是，則表示玩家試圖推動箱子。此時，根據 `direction` 更新箱子的預期新位置 `newBoxPos` (即箱子在推動方向上再前進一格)，並將 `isBoxMoved` 設為 `true`。
3. **檢查箱子是否到達目標**:
    - 如果箱子被推動 ( `isBoxMoved` 為 true)，檢查箱子的 `newBoxPos` 是否為目標點 `Maze::GOAL`。如果是，則設定 `maze_.setIsGameOver(true)`。
4. **有效性驗證**:
    - 呼叫 `isValidMove(newPlayerPos.x, newPlayerPos.y)` 檢查玩家的預期新位置是否有效 (非牆、未越界)。
    - 呼叫 `isValidMove(newBoxPos.x, newBoxPos.y)` 檢查箱子的預期新位置是否有效 (**即使箱子未被推動，此檢查也會進行，但此時 `newBoxPos` 等於箱子原始位置，所以通常會通過。**關鍵在於箱子被推動時，其新位置的有效性)。
5. **執行移動與更新狀態 (如果有效)**:
    - 如果玩家的移動 和 箱子 (若被推動) 的移動都有效:
      a. **記錄歷史**:
      * 將箱子當前是否被移動的狀態 `isBoxMoved` 存入 `maze_.getBoxIsMovedHistory()` 堆疊。
      * 將玩家當前位置 `maze_.getPlayer()` 存入 `maze_.getPlayerPositionHistory()` 堆疊。
      * 將箱子當前位置 `maze_.getBox()` 存入 `maze_.getBoxPositionHistory()` 堆疊。
      b. **更新迷宮地圖 ( `maze_` )**:
      * 將玩家的舊位置 `maze_.getPlayer()` 在地圖上更新為通路 `Maze::PATH` (或如果舊位置是目標點，則保留為 `Maze::GOAL`，這點程式碼中是 `(maze_.getPlayer().x == maze_.getGoal().x && maze_.getPlayer().y == maze_.getGoal().y) ? Maze::GOAL : Maze::PATH)`)。
      * 將玩家的新位置 `newPlayerPos` 在地圖上更新為玩家符號 `Maze::PLAYER`。
      c. **更新物件實際位置**:
      * 設定玩家的新位置: `maze_.setPlayer(newPlayerPos)`。
      d. **如果箱子被推動 ( `isBoxMoved` )**:
      * 將箱子的新位置 `newBoxPos` 在地圖上更新為箱子符號 `Maze::BOX` (或者如果

是目標點，則保持 `Maze::GOAL` ，程式碼為 `maze_.getIsGameOver() ? Maze::GOAL : Maze::BOX` )。

* 設定箱子的新位置： `maze_.setBox(newBoxPos)` 。

e. 返回 `true` 表示移動成功。

6. **無效移動**: 如果任一位置無效，則不執行任何操作，返回 `false` 。

cpp

```cpp
// move.cpp 中的 movePlayer 核心片段
bool Move::movePlayer(char direction) {
    Maze::Position newPlayerPos = maze_.getPlayer();
    Maze::Position newBoxPos = maze_.getBox(); // 初始化為當前箱子位置
    bool isBoxMoved = false;

    // 根據方向計算 newPlayerPos，如果撞到箱子，則計算 newBoxPos 並設定 isBoxMoved
    if (direction == 'w') { /* ... */ }
    else if (direction == 's') { /* ... */ }
    // ... (a, d 類似)

    // 檢查箱子是否到達目標
    if (maze_.getMaze()[newBoxPos.x][newBoxPos.y] == Maze::GOAL) { // 注意: 這裡應該判斷 isBo
        maze_.setIsGameOver(true);
    }

    // 驗證移動
    if (isValidMove(newPlayerPos.x, newPlayerPos.y) && isValidMove(newBoxPos.x, newBoxPos.y))
        // 記錄歷史
        maze_.getBoxIsMovedHistory().push(isBoxMoved);
        maze_.getPlayerPositionHistory().push(maze_.getPlayer());
        maze_.getBoxPositionHistory().push(maze_.getBox());

        // 更新地圖與物件位置
        maze_.setMazeCell(maze_.getPlayer().x, maze_.getPlayer().y,
                          (maze_.getPlayer().x == maze_.getGoal().x && maze_.getPlayer().y ==
        maze_.setMazeCell(newPlayerPos.x, newPlayerPos.y, Maze::PLAYER);
        maze_.setPlayer(newPlayerPos);
        if (isBoxMoved) {
            maze_.setMazeCell(newBoxPos.x, newBoxPos.y, maze_.getIsGameOver() ? Maze::GOAL :
            maze_.setBox(newBoxPos);
        }
        return true;
    }
    return false;
}

```

`Move::undoMove()` 的新邏輯

此函數用於撤銷上一步的 `movePlayer` 操作。

1. **檢查是否有可復原的步驟**:
    - 如果 `maze_.getPlayerPositionHistory()` 為空，表示沒有歷史紀錄，則提示無法復原並返回。
2. **獲取上一步的狀態**:
    - `bool lastMoveMovedBox = maze_.getBoxIsMovedHistory().top();` (獲取上一步箱子是否移動)

- `Maze::Position prevPlayerPos = maze_.getPlayerPositionHistory().top();` (獲取玩家前一位置)
- `Maze::Position prevBoxPos = maze_.getBoxPositionHistory().top();` (獲取箱子前一位置)

3. **恢復迷宮地圖 (`maze_`)**:
   - **如果上一步箱子被移動 (`lastMoveMovedBox == true`)**:
     - 將箱子當前在地圖上的位置 `maze_.getBox()` 設定為通路 `Maze::PATH`。
     - 將玩家當前在地圖上的位置 `maze_.getPlayer()` (即箱子被推之前的舊位置) 設定回箱子 `Maze::BOX`。
   - **如果上一步箱子未被移動 (`lastMoveMovedBox == false`)**:
     - 將玩家當前在地圖上的位置 `maze_.getPlayer()` 設定為通路 `Maze::PATH` (或如果該位置是目標點，則為 `Maze::GOAL`)。
4. **恢復物件實際位置**:
   - 設定玩家位置為 `prevPlayerPos`: `maze_.setPlayer(prevPlayerPos);`
   - 設定箱子位置為 `prevBoxPos`: `maze_.setBox(prevBoxPos);`
5. **移除已復原的歷史紀錄**:
   - `maze_.getPlayerPositionHistory().pop();`
   - `maze_.getBoxPositionHistory().pop();`
   - `maze_.getBoxIsMovedHistory().pop();`
6. **更新地圖上恢復後玩家的位置**:
   - 將恢復後的玩家位置 `maze_.getPlayer()` (即 `prevPlayerPos`) 在地圖上標記為玩家 `Maze::PLAYER`。
   - (注意：如果 `isGameOver_` 狀態因復原而改變，例如復原了一個使箱子到達目標的移動，目前的 `undoMove` 邏輯沒有直接重置 `isGameOver_`。遊戲狀態的正確性可能依賴於下一次 `movePlayer` 或遊戲迴圈的檢查。)

cpp

```cpp
// move.cpp 中的 undoMove 核心片段
void Move::undoMove() {
    if (maze_.getPlayerPositionHistory().empty()) { /* ... 無法復原 ... */ return; }

    bool lastMoveMovedBox = maze_.getBoxIsMovedHistory().top(); // 先獲取再 pop
    Maze::Position prevPlayerPos = maze_.getPlayerPositionHistory().top();
    Maze::Position prevBoxPos = maze_.getBoxPositionHistory().top();

    // 清理當前狀態，準備恢復
    if (lastMoveMovedBox) {
        maze_.setMazeCell(maze_.getBox().x, maze_.getBox().y, Maze::PATH);       // 清除當前
        maze_.setMazeCell(maze_.getPlayer().x, maze_.getPlayer().y, Maze::BOX); // 玩家現處
    } else {
        maze_.setMazeCell(maze_.getPlayer().x, maze_.getPlayer().y,
                          (maze_.getPlayer().x == maze_.getGoal().x && maze_.getPlayer().y =
    }

```

```
    // 恢復歷史狀態
    maze_.setPlayer(prevPlayerPos);
    maze_.setBox(prevBoxPos);
    maze_.getPlayerPositionHistory().pop();
    maze_.getBoxPositionHistory().pop();
    maze_.getBoxIsMovedHistory().pop();

    // 在新恢復的玩家位置上標記玩家
    maze_.setMazeCell(maze_.getPlayer().x, maze_.getPlayer().y, Maze::PLAYER);
    // 如果箱子之前在目標點上，undo後箱子不在目標點，isGameOver_ 也應該更新
    // 目前版本 maze_.setIsGameOver(false) 在 undoMove 中沒有明確處理。
    // 但 restoreOriginalMaze() 會重置 isGameOver_。
    // 如果箱子推到終點後再undo，然後再推一個非箱子物體到終點，isGameOver仍為true，
    // 這可能是一個潛在的小問題，但推箱子遊戲中通常箱子到終點就結束了。
}
```

### 4.5 `move.h` 的變更

- **5.1. 移除** `isOppositeMove()`
  - **變更**: 從 `Move` 類別的宣告中移除了 `isOppositeMove()` 函數的宣告。
  - **原因**: 與 `move.cpp` 中的移除相對應，該函數已不再使用。
  - **影響**: `Move` 類別的介面更加精簡。

### 4.6 `pathfinder.cpp` 的移除

- **原因**: 自動模式 (auto mode) 尚未完成，因此實現路徑尋找功能的 `pathfinder.cpp` 檔案被整個移除。
- **影響**: 專案不再包含自動尋路演算法的實作，減少了專案的複雜度和檔案數量。

### 4.7 `pathfinder.h` 的移除

- **原因**: 同 4.6，由於自動模式未完成，其對應的頭檔案 `pathfinder.h` 也被移除。
- **影響**: 專案依賴減少。

---

## 5. 結論

相較於 HW3 版本，此專案經歷了重大的功能轉變，從一個迷宮尋路遊戲演化為一個推箱子益智遊戲。核心的變更圍繞著引入「箱子」這一可互動元素，並重新設計了玩家移動（`movePlayer`）和撤銷移動（`undoMove`）的邏輯，以支援推動箱子及相應的狀態回溯。
主要的技術調整包括：

- 在 `Maze` 類別中新增對箱子位置（`box_`）及相關歷史紀錄（`boxPositionHistory_`，`boxIsMovedHistory_`）的管理。
- 大幅修改 `Move` 類別中的 `movePlayer` 函數，使其能夠處理玩家與箱子的互動，包括推動、邊界檢測、目標達成判斷。
- 相應地重寫 `undoMove` 函數，使其能夠正確恢復玩家和箱子的前一狀態。
- 移除了與舊版尋路功能相關的 `PathFinder` 模組。

- 在 `main.cpp` 中加入了初步的箱子卡死判斷與自動復原機制。

這些變更使得遊戲的核心玩法煥然一新，並引入了更複雜的狀態管理和邏輯判斷。雖然自動模式等功能尚待完善，但目前的手動推箱子模式已具備了核心的可玩性。