遊戲專案：玩家管理系統與簡單射擊遊戲

專案介紹

這個專案是一個簡單的控制台遊戲，包含玩家管理系統和遊戲功能。玩家管理系統允許玩家註冊、查詢、打印排名和刪除玩家；遊戲部分則包含了基本的遊戲邏輯，包括玩家移動、敵人移動和子彈處理。

使用的課程主題

1. **內存動態分配：**
    - 使用 `std::vector` 動態管理敵人和子彈的位置。
    - 使用 `std::map` 動態管理玩家及其得分。
2. **文件處理：**
    - 使用文件存儲和加載玩家信息，確保玩家信息在程序重新啟動後仍然存在。
3. **數據結構：**
    - 使用 `std::vector` 和 `std::map` 管理遊戲和玩家數據。
    - 使用 `std::pair` 存儲敵人和子彈的位置。
4. **指針應用：**
    - 使用引用傳遞 `PlayerManager` 對象，以便在不同函數之間共享數據。

各功能介紹

PlayerManager 類

成員函數

- `addPlayer(const std::string& playerID)`：添加新玩家。
- `deletePlayer(const std::string& playerID)`：刪除玩家。
- `searchPlayer(const std::string& playerID)`：查詢玩家。
- `printScoreRanking()`：打印玩家得分排名。
- `savePlayers()`：將玩家信息保存到文件中。
- `loadPlayers()`：從文件加載玩家信息。
- `addPlayerScore(const std::string& playerID, int score)`：增加玩家得分。
- `setCurrentPlayerID(const std::string& playerID)`：設置當前玩家ID。
- `getCurrentPlayerID()`：獲取當前玩家ID。

Game 類

成員函數

- `setup()`：初始化遊戲狀態。
- `nextLevel()`：進入下一關。

- `draw()`：繪製遊戲畫面。
- `input()`：處理玩家輸入。
- `logic()`：處理遊戲邏輯。
- `start()`：開始遊戲，主遊戲循環。

主控代碼（`main.cpp`）

主控函數

- `main()`：程序入口，負責初始化 `PlayerManager` 並啟動主菜單。
- `mainMenu(PlayerManager& playerManager)`：主菜單函數，處理玩家管理和遊戲選擇。

遊戲說明與操作步驟

遊戲說明

本遊戲為簡單的射擊遊戲，玩家需要操控角色射擊敵人，並躲避敵人的攻擊。遊戲中，玩家只有一點血量，被敵人的子彈擊中一次即會 Game Over。遊戲包含多個關卡，每一關卡敵人的數量會逐漸增加，玩家需要不斷提升自己的技術來面對更高難度的挑戰。

操作步驟

1. **輸入玩家 ID**
    - 開啟遊戲後，玩家需要先輸入一個 ID 來記錄自己的遊戲進度與分數。
2. **主選單操作**
    - 輸入 ID 後，玩家可以在主選單中選擇以下操作：
        1. **Enter Player ID**：輸入玩家 ID 來開始遊戲。
        2. **Search Player ID**：查詢玩家 ID 是否已經存在。
        3. **Print Score Ranking**：打印玩家的分數排行榜。
        4. **Delete Player ID**：刪除已存在的玩家 ID。
        5. **Start the Game**：開始遊戲。
        6. **Exit**：退出遊戲。
3. **遊戲操作**
    - 使用鍵盤左右方向鍵 (← 和 →) 控制角色移動。
    - 使用空格鍵 (`Space`) 發射子彈攻擊敵人。
4. **遊戲目標**
    - 擊敗所有敵人並避開敵人的子彈。每當擊中敵人時，玩家會獲得分數。當所有敵人被擊敗後，進入下一關卡，敵人的數量會增加，難度也會提高。
    - 若玩家被敵人的子彈擊中，遊戲結束，並顯示玩家的最終分數和等級。

完整代碼

`main.cpp`

```cpp
#include <iostream> // 包含iostream標頭檔，用於輸入輸出
#include "player.h" // 包含player.h標頭檔，定義PlayerManager類
#include "game.h" // 包含game.h標頭檔，定義Game類

using namespace std; // 使用標準命名空間

// 定義主菜單函數，用於玩家管理和啟動遊戲
void mainMenu(PlayerManager& playerManager) {
    int choice; // 用於存儲用戶選擇
    do {
        // 顯示菜單選項
        cout << "\nPlayer Management System\n";
        cout << "1. Enter Player ID\n";
        cout << "2. Search Player ID\n";
        cout << "3. Print Score Ranking\n";
        cout << "4. Delete Player ID\n";
        cout << "5. Start the Game\n";
        cout << "6. Exit\n";
        cout << "Please choose an operation: ";
        std::cout << "\n";  // 換行

        cin >> choice; // 獲取用戶輸入
        switch (choice) { // 根據用戶輸入執行對應操作
            case 1: // 選項1：輸入玩家ID
                {
                    string id; // 用於存儲玩家ID
                    cout << "Enter Player ID: ";
                    cin >> id; // 獲取玩家ID
                    playerManager.addPlayer(id); // 添加玩家
                    playerManager.setCurrentPlayerID(id); // 設置當前玩家ID
                }
                break;
            case 2: // 選項2：搜索玩家ID
                {
                    string id; // 用於存儲要搜索的玩家ID
                    cout << "Enter Player ID to search: ";
                    cin >> id; // 獲取要搜索的玩家ID
                    playerManager.searchPlayer(id); // 搜索玩家
                }
                break;
            case 3: // 選項3：打印分數排名
                playerManager.printScoreRanking(); // 打印分數排名
                break;
            case 4: // 選項4：刪除玩家ID
                {
                    string id; // 用於存儲要刪除的玩家ID
                    cout << "Enter Player ID to delete: ";
                    cin >> id; // 獲取要刪除的玩家ID
                    playerManager.deletePlayer(id); // 刪除玩家
                }
                break;
            case 5: // 選項5：開始遊戲
                {
```

```
                    if (playerManager.getCurrentPlayerID().empty()) { // 如果沒有設置當前玩家ID
                        cout << "Please enter a Player ID first.\n"; // 提示用戶先輸入玩家ID
                    } else {
                        Game game(playerManager); // 創建Game對象
                        game.start(); // 開始遊戲
                        // 遊戲結束後顯示菜單選項
                        cout << "\nPlayer Management System\n";
                        cout << "1. Enter Player ID\n";
                        cout << "2. Search Player ID\n";
                        cout << "3. Print Score Ranking\n";
                        cout << "4. Delete Player ID\n";
                        cout << "5. Start the Game\n";
                        cout << "6. Exit\n";
                        cout << "Please choose an operation: ";
                    }
                }
                break;
            case 6: // 選項6：退出
                playerManager.savePlayers(); // 保存玩家數據
                break;
            default: // 非法選項
                cout << "Invalid choice. Please try again.\n"; // 提示用戶重新選擇
        }
    } while (choice != 6); // 當用戶選擇退出時，退出循環
}

int main() {
    PlayerManager playerManager; // 創建PlayerManager對象
    playerManager.loadPlayers(); // 加載玩家數據
    mainMenu(playerManager); // 顯示主菜單
    return 0; // 返回0，表示程序正常結束
}
```

player.h

```
#ifndef PLAYER_H // 防止重複包含標頭檔案
#define PLAYER_H // 定義PLAYER_H宏

#include <string> // 包含string標頭檔，用於字符串操作
#include <unordered_map> // 包含unordered_map標頭檔，用於哈希映射
#include <fstream> // 包含fstream標頭檔，用於文件操作
#include <vector> // 包含vector標頭檔，用於動態數組
#include <algorithm> // 包含algorithm標頭檔，用於標準算法
#include <iostream> // 包含iostream標頭檔，用於輸入輸出
#include <utility> // 包含utility標頭檔，用於std::pair

class PlayerManager {
private:
```

```
    std::unordered_map<std::string, int> playerScores; // 存儲玩家ID和分數的哈希映射
    std::string currentPlayerID; // 當前玩家的ID

public:
    void addPlayer(const std::string& id); // 添加玩家ID
    void deletePlayer(const std::string& id); // 刪除玩家ID
    void searchPlayer(const std::string& id); // 搜索玩家ID
    void printScoreRanking(); // 打印分數排名
    void savePlayers(); // 保存玩家數據到文件
    void loadPlayers(); // 從文件加載玩家數據
    void setCurrentPlayerID(const std::string& id); // 設置當前玩家ID
    std::string getCurrentPlayerID(); // 獲取當前玩家ID
    void addPlayerScore(const std::string& id, int score); // 添加玩家分數
};

#endif // 結束PLAYER_H的條件編譯
```

player.cpp

```
#include "player.h" // 包含player.h標頭檔案，定義PlayerManager類
#include <iostream> // 包含iostream標頭檔，用於輸入輸出
#include <algorithm> // 包含algorithm標頭檔，用於標準算法
#include <vector> // 包含vector標頭檔，用於動態數組
#include <fstream> // 包含fstream標頭檔，用於文件輸入輸出

// 添加一個新的玩家ID
void PlayerManager::addPlayer(const std::string& id) {
    if (playerScores.find(id) == playerScores.end()) { // 檢查玩家ID是否已存在
        playerScores[id] = 0; // 如果不存在，添加玩家並設置初始分數為0
        std::cout << "Player " << id << " added.\n"; // 輸出玩家已添加的信息
    } else {
        std::cout << "Player " << id << " already exists.\n"; // 如果玩家已存在，輸
出提示信息
    }
}

// 刪除指定的玩家ID
void PlayerManager::deletePlayer(const std::string& id) {
    if (playerScores.erase(id)) { // 刪除玩家，如果成功返回true
        std::cout << "Player " << id << " deleted.\n"; // 輸出玩家已刪除的信息
    } else {
        std::cout << "Player " << id << " not found.\n"; // 如果玩家未找到，輸出提示
信息
    }
}

// 搜索並顯示指定的玩家ID及其分數
void PlayerManager::searchPlayer(const std::string& id) {
    auto it = playerScores.find(id); // 查找玩家ID
    if (it != playerScores.end()) { // 如果找到玩家
```

```
        std::cout << "Player ID: " << it->first << ", Score: " << it->second <<
"\n"; // 顯示玩家ID和分數
    } else {
        std::cout << "Player " << id << " not found.\n"; // 如果未找到玩家，輸出提示
信息
    }
}

// 打印所有玩家的分數排名
void PlayerManager::printScoreRanking() {
    std::vector<std::pair<std::string, int>> ranking(playerScores.begin(),
playerScores.end()); // 創建一個包含所有玩家ID和分數的向量
    std::sort(ranking.begin(), ranking.end(), [](const std::pair<std::string,
int>& a, const std::pair<std::string, int>& b) {
        return a.second > b.second; // 按照分數從高到低排序
    });

    for (const auto& it : ranking) { // 遍歷排序後的向量
        std::cout << "Player ID: " << it.first << ", Score: " << it.second <<
"\n"; // 顯示玩家ID和分數
    }
}

// 保存所有玩家數據到文件
void PlayerManager::savePlayers() {
    std::ofstream ofs("players.dat"); // 打開文件進行寫操作
    for (const auto& pair : playerScores) { // 遍歷所有玩家
        ofs << pair.first << " " << pair.second << "\n"; // 將玩家ID和分數寫入文件
    }
}

// 從文件中加載玩家數據
void PlayerManager::loadPlayers() {
    std::ifstream ifs("players.dat"); // 打開文件進行讀操作
    std::string id; // 用於存儲讀取的玩家ID
    int score; // 用於存儲讀取的分數
    while (ifs >> id >> score) { // 逐行讀取玩家ID和分數
        playerScores[id] = score; // 將讀取的玩家數據存入map
    }
}

// 設置當前玩家ID
void PlayerManager::setCurrentPlayerID(const std::string& id) {
    currentPlayerID = id; // 設置當前玩家ID
}

// 獲取當前玩家ID
std::string PlayerManager::getCurrentPlayerID() {
    return currentPlayerID; // 返回當前玩家ID
}

// 為指定玩家添加分數
void PlayerManager::addPlayerScore(const std::string& id, int score) {
    if (playerScores.find(id) != playerScores.end()) { // 檢查玩家是否存在
```

```
            playerScores[id] += score; // 添加分數
        }
    }
```

game.h

```
#ifndef GAME_H // 防止重複包含標頭檔案
#define GAME_H // 定義標頭檔案宏

#include <iostream> // 包含iostream標頭檔，用於輸出
#include <vector> // 包含vector標頭檔，用於動態數組
#include <utility> // 包含utility標頭檔，用於std::pair
#include <ctime> // 包含ctime標頭檔，用於時間相關函數
#include <conio.h> // 包含conio.h，用於非阻塞式鍵盤輸入
#include <windows.h> // 包含windows.h，用於Windows API
#include <sstream> // 包含sstream標頭檔，用於字符串流

class PlayerManager; // 前向宣告PlayerManager類

class Game {
public:
    Game(PlayerManager& pm); // Game類的構造函數
    void setup(); // 設置遊戲
    void nextLevel(); // 進入下一級
    void draw(); // 繪製遊戲界面
    void input(); // 處理玩家輸入
    void logic(); // 更新遊戲邏輯
    void start(); // 開始遊戲
private:
    const int width = 30; // 遊戲區域寬度
    const int height = 20; // 遊戲區域高度
    char player = '^'; // 玩家符號
    char enemy = '@'; // 敵人符號
    char bullet = '|'; // 子彈符號
    char enemyBullet = '!'; // 敵方子彈符號
    char border = '#'; // 邊界符號
    int playerX; // 玩家位置
    std::vector<std::pair<int, int>> enemies; // 敵人位置列表
    std::vector<std::pair<int, int>> bullets; // 子彈位置列表
    std::vector<std::pair<int, int>> enemyBullets; // 敵方子彈位置列表
    bool gameOver; // 遊戲結束標誌
    int score; // 分數
    int level; // 等級
    PlayerManager& playerManager; // PlayerManager的引用
};

#endif // 結束GAME_H的條件編譯
```

game.cpp

```
#include "game.h" // 包含game.h頭文件，定義Game類
#include "player.h" // 包含player.h頭文件，定義PlayerManager類
#include <iostream> // 包含iostream標頭檔，用於輸出
#include <conio.h> // 包含conio.h，用於非阻塞式鍵盤輸入
#include <windows.h> // 包含windows.h，用於Windows API
#include <vector> // 包含vector標頭檔，用於動態數組
#include <sstream> // 包含sstream標頭檔，用於字符串流
#include <algorithm> // 包含algorithm標頭檔，用於標準算法
#include <cstdlib> // 包含cstdlib標頭檔，用於標準庫函數
#include <ctime> // 包含ctime標頭檔，用於時間相關函數
using namespace std; // 使用標準命名空間

// Game類的構造函數，初始化PlayerManager的引用
Game::Game(PlayerManager& pm) : playerManager(pm) {}

// setup()函數，用於初始化遊戲狀態
void Game::setup() {
    srand(time(0)); // 設置隨機數生成器的種子
    gameOver = false; // 初始化遊戲結束標誌
    playerX = width / 2; // 將玩家位置設置在屏幕中間
    score = 0; // 初始化分數
    level = 1; // 初始化等級
    enemies.clear(); // 清空敵人列表
    bullets.clear(); // 清空子彈列表
    enemyBullets.clear(); // 清空敵方子彈列表

    // 初始化敵人位置
    for (int i = 0; i < 5; ++i) {
        for (int j = 0; j < width; j += 5) {
            enemies.push_back({i, j}); // 添加敵人到敵人列表
        }
    }

    // 隱藏控制台光標
    CONSOLE_CURSOR_INFO cursorInfo;
    GetConsoleCursorInfo(GetStdHandle(STD_OUTPUT_HANDLE), &cursorInfo); // 獲取控制台光標信息
    cursorInfo.bVisible = false; // 設置光標不可見
    SetConsoleCursorInfo(GetStdHandle(STD_OUTPUT_HANDLE), &cursorInfo); // 更新控制台光標信息
}

// nextLevel()函數，用於進入下一級
void Game::nextLevel() {
    level++; // 增加等級
    enemies.clear(); // 清空敵人列表
    bullets.clear(); // 清空子彈列表
    enemyBullets.clear(); // 清空敵方子彈列表

    // 初始化新的敵人位置，根據等級增加敵人數量
```

```
    for (int i = 0; i < 5 + level; ++i) {
        for (int j = 0; j < width; j += 5) {
            enemies.push_back({i, j}); // 添加敵人到敵人列表
        }
    }
}

// draw()函數，用於繪製遊戲界面
void Game::draw() {
    ostringstream oss; // 創建字符串流對象

    // 繪製頂部邊界
    for (int i = 0; i < width + 2; i++) oss << border;
    oss << endl;

    // 繪製遊戲區域
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            if (j == 0) oss << border; // 繪製左邊邊界

            bool drawn = false; // 用於標記當前位置是否已繪製

            // 繪製敵人
            for (const auto &enemyPos : enemies) {
                if (enemyPos.first == i && enemyPos.second == j) {
                    oss << enemy; // 如果當前位置有敵人，繪製敵人
                    drawn = true; // 標記已繪製
                    break; // 跳出循環
                }
            }

            // 如果當前位置沒有敵人，繪製子彈
            if (!drawn) {
                for (const auto &bulletPos : bullets) {
                    if (bulletPos.first == i && bulletPos.second == j) {
                        oss << bullet; // 如果當前位置有子彈，繪製子彈
                        drawn = true; // 標記已繪製
                        break; // 跳出循環
                    }
                }
            }

            // 如果當前位置沒有敵人和子彈，繪製敵方子彈
            if (!drawn) {
                for (const auto &enemyBulletPos : enemyBullets) {
                    if (enemyBulletPos.first == i && enemyBulletPos.second == j) {
                        oss << enemyBullet; // 如果當前位置有敵方子彈，繪製敵方子彈
                        drawn = true; // 標記已繪製
                        break; // 跳出循環
                    }
                }
            }

            // 如果當前位置沒有敵人、子彈和敵方子彈，繪製玩家
```

```
            if (!drawn && i == height - 1 && j == playerX) {
                oss << player; // 如果當前位置是玩家，繪製玩家
                drawn = true; // 標記已繪製
            }

            // 如果當前位置什麼都沒有，繪製空格
            if (!drawn) oss << ' ';

            // 繪製右邊邊界
            if (j == width - 1) oss << border;
        }
        oss << endl; // 換行
    }

    // 繪製底部邊界
    for (int i = 0; i < width + 2; i++) oss << border;
    oss << endl;

    // 繪製分數和等級信息
    oss << "Score: " << score << "  Level: " << level << endl;

    // 將光標位置設置為(0, 0)，以便更新整個控制台內容
    COORD coord = {0, 0};
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
    cout << oss.str(); // 輸出字符串流中的內容
}

// input()函數，用於處理玩家輸入
void Game::input() {
    if (_kbhit()) { // 如果有鍵盤輸入
        char key = _getch(); // 獲取按鍵
        if (key == -32) { // 如果是特殊按鍵，繼續讀取
            key = _getch();
        }
        switch (key) {
            case 75: // 左箭頭鍵
                if (playerX > 0) playerX -= 1; // 移動玩家到左邊
                break;
            case 77: // 右箭頭鍵
                if (playerX < width - 1) playerX += 1; // 移動玩家到右邊
                break;
            case 32: // 空格鍵
                bullets.push_back({height - 2, playerX}); // 發射子彈
                break;
        }
    }
}

// logic()函數，用於更新遊戲邏輯
void Game::logic() {
    // 更新子彈位置
    for (auto &bulletPos : bullets) {
        bulletPos.first--; // 子彈向上移動
    }
```

```
    // 刪除屏幕外的子彈
    bullets.erase(remove_if(bullets.begin(), bullets.end(),
        [this](const std::pair<int, int> &bulletPos) {
            return bulletPos.first < 0; // 如果子彈超出屏幕上邊界，刪除
        }),
        bullets.end());

    // 檢測子彈與敵人的碰撞
    for (auto bulletIt = bullets.begin(); bulletIt != bullets.end();) {
        bool hit = false; // 紀錄是否有擊中敵人
        for (auto enemyIt = enemies.begin(); enemyIt != enemies.end();) {
            if (bulletIt->first == enemyIt->first && bulletIt->second == enemyIt-
>second) {
                score++; // 增加分數
                enemyIt = enemies.erase(enemyIt); // 刪除被擊中的敵人
                hit = true; // 標記為擊中
                break; // 跳出內循環
            } else {
                ++enemyIt; // 否則，繼續檢查下一個敵人
            }
        }
        if (hit) {
            bulletIt = bullets.erase(bulletIt); // 刪除擊中敵人的子彈
        } else {
            ++bulletIt; // 否則，檢查下一顆子彈
        }
    }

    // 更新敵方子彈位置
    for (auto &enemyBulletPos : enemyBullets) {
        enemyBulletPos.first++; // 敵方子彈向下移動
    }

    // 刪除屏幕外的敵方子彈
    enemyBullets.erase(remove_if(enemyBullets.begin(), enemyBullets.end(),
        [this](const std::pair<int, int> &enemyBulletPos) {
            return enemyBulletPos.first >= height; // 如果敵方子彈超出屏幕下邊界，刪
除
        }),
        enemyBullets.end());

    // 檢測敵方子彈與玩家的碰撞
    for (const auto &enemyBulletPos : enemyBullets) {
        if (enemyBulletPos.first == height - 1 && enemyBulletPos.second ==
playerX) {
            gameOver = true; // 如果敵方子彈擊中玩家，遊戲結束
        }
    }

    // 根據等級隨機生成敵方子彈
    if (rand() % 10 < level) {
        if (!enemies.empty()) { // 如果還有敵人
            int randomEnemy = rand() % enemies.size(); // 隨機選擇一個敵人
```

```
            enemyBullets.push_back({enemies[randomEnemy].first + 1,
enemies[randomEnemy].second}); // 從選擇的敵人位置發射子彈
        }
    }

    // 如果所有敵人都被消滅且遊戲未結束，進入下一級
    if (enemies.empty() && !gameOver) {
        nextLevel(); // 進入下一級
    }
}

// start()函數，用於開始遊戲
void Game::start() {
    setup(); // 設置遊戲
    while (!gameOver) { // 當遊戲未結束
        draw(); // 繪製遊戲界面
        input(); // 處理玩家輸入
        logic(); // 更新遊戲邏輯
        Sleep(100); // 遊戲速度控制
    }

    // 遊戲結束，顯示分數和等級
    cout << "Game Over! Your score is: " << score << "  Level: " << level << endl;

    // 添加玩家分數到PlayerManager
    playerManager.addPlayerScore(playerManager.getCurrentPlayerID(), score);

    // 恢復控制台光標顯示
    CONSOLE_CURSOR_INFO cursorInfo;
    GetConsoleCursorInfo(GetStdHandle(STD_OUTPUT_HANDLE), &cursorInfo);
    cursorInfo.bVisible = true;
    SetConsoleCursorInfo(GetStdHandle(STD_OUTPUT_HANDLE), &cursorInfo);
}
```