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`

```
#include <iostream>
#include "player.h"
#include "game.h"

using namespace std;

void mainMenu(PlayerManager& playerManager) {
    int choice;
    do {
        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";  // 添加空行

        cin >> choice;
        switch (choice) {
            case 1:
                {
                    string id;
                    cout << "Enter Player ID: ";
                    cin >> id;
                    playerManager.addPlayer(id);
                    playerManager.setCurrentPlayerID(id);
                }
                break;
            case 2:
                {
                    string id;
                    cout << "Enter Player ID to search: ";
                    cin >> id;
                    playerManager.searchPlayer(id);
                }
                break;
            case 3:
                playerManager.printScoreRanking();
                break;
            case 4:
                {
                    string id;
                    cout << "Enter Player ID to delete: ";
                    cin >> id;
                    playerManager.deletePlayer(id);
                }
                break;
            case 5:
                {
                    if (playerManager.getCurrentPlayerID().empty()) {
                        cout << "Please enter a Player ID first.\n";
```

```
                    } else {
                        Game game(playerManager);
                        game.start();
                        // 在遊戲結束後重新打印選單
                        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";  // 添加空行
                    }
                }
                break;
            case 6:
                playerManager.savePlayers();
                break;
            default:
                cout << "Invalid choice. Please try again.\n";
        }
    } while (choice != 6);
}

int main() {
    PlayerManager playerManager;
    playerManager.loadPlayers();
    mainMenu(playerManager);
    return 0;
}
```

player.h

```
#ifndef PLAYER_H
#define PLAYER_H

#include <iostream>
#include <fstream>
#include <map>
#include <string>

class PlayerManager {
public:
    void addPlayer(const std::string& playerID);
    void deletePlayer(const std::string& playerID);
    void searchPlayer(const std::string& playerID);
    void printScoreRanking();
    void savePlayers();
    void loadPlayers();
    void addPlayerScore(const std::string& playerID, int score);
```

```
    void setCurrentPlayerID(const std::string& playerID);
    std::string getCurrentPlayerID();

private:
    std::map<std::string, int> playerScores;
    std::string currentPlayerID;
};

#endif
```

player.cpp

```
#include "player.h"

void PlayerManager::addPlayer(const std::string& playerID) {
    if (playerScores.find(playerID) == playerScores.end()) {
        playerScores[playerID] = 0;
        std::cout << "Player " << playerID << " added.\n";
    } else {
        std::cout << "Player " << playerID << " already exists.\n";
    }
}

void PlayerManager::deletePlayer(const std::string& playerID) {
    auto it = playerScores.find(playerID);
    if (it != playerScores.end()) {
        playerScores.erase(it);
        std::cout << "Player " << playerID << " deleted.\n";
    } else {
        std::cout << "Player " << playerID << " not found.\n";
    }
}

void PlayerManager::searchPlayer(const std::string& playerID) {
    auto it = playerScores.find(playerID);
    if (it != playerScores.end()) {
        std::cout << "Player ID: " << playerID << ", Score: " << it->second <<
"\n";
    } else {
        std::cout << "Player " << playerID << " not found.\n";
    }
}

void PlayerManager::printScoreRanking() {
    std::vector<std::pair<std::string, int>> ranking(playerScores.begin(),
playerScores.end());
    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;
    });
    for (const auto& it : ranking) {
```

```
            std::cout << "Player ID: " << it.first << ", Score: " << it.second <<
"\n";
        }
    }

    void PlayerManager::savePlayers() {
        std::ofstream file("players.txt");
        for (const auto& it : playerScores) {
            file << it.first << " " << it.second << "\n";
        }
    }

    void PlayerManager::loadPlayers() {
        std::ifstream file("players.txt");
        if (file.is_open()) {
            std::string playerID;
            int score;
            while (file >> playerID >> score) {
                playerScores[playerID] = score;
            }
        }
    }

    void PlayerManager::addPlayerScore(const std::string& playerID, int score) {
        playerScores[playerID] += score;
    }

    void PlayerManager::setCurrentPlayerID(const std::string& playerID) {
        currentPlayerID = playerID;
    }

    std::string PlayerManager::getCurrentPlayerID() {
        return currentPlayerID;
    }
```

game.h

```
    #ifndef GAME_H
    #define GAME_H

    #include <iostream>
    #include <vector>
    #include <utility>
    #include "player.h"

    class Game {
    public:
        Game(PlayerManager& pm);
        void setup();
        void nextLevel();
        void draw();
```

```
    void input();
    void logic();
    void start();

private:
    int playerX;
    std::vector<std::pair<int, int>> enemies;
    std::vector<std::pair<int, int>> bullets;
    std::vector<std::pair<int, int>> enemyBullets;
    bool gameOver;
    int score;
    int level;
    PlayerManager& playerManager;


    const int width = 30;
    const int height = 20;
    const char player = '^';
    const char enemy = '@';
    const char bullet = '|';
    const char enemyBullet = '!';
    const char border = '#';
};

#endif
```

game.cpp

```
#include "game.h"
#include <conio.h>
#include <windows.h>
#include <sstream>
#include <algorithm>
#include <cstdlib>
#include <ctime>

Game::Game(PlayerManager& pm) : playerManager(pm) {}

void Game::setup() {
    srand(time(0));
    gameOver = false;
    playerX = width / 2;
    score = 0;
    level = 1;
    enemies.clear();
    bullets.clear();
    enemyBullets.clear();

    // 初始化敌人
    for (int i = 0; i < 5; ++i) {
        for (int j = 0; j < width; j += 5) {
```

```
            enemies.push_back({i, j});
        }
    }

    // 設置控制台游標隱藏
    CONSOLE_CURSOR_INFO cursorInfo;
    GetConsoleCursorInfo(GetStdHandle(STD_OUTPUT_HANDLE), &cursorInfo);
    cursorInfo.bVisible = false; // 設置游標不可見
    SetConsoleCursorInfo(GetStdHandle(STD_OUTPUT_HANDLE), &cursorInfo);
}

void Game::nextLevel() {
    level++;
    enemies.clear();
    bullets.clear();
    enemyBullets.clear();

    // 增加敵人的數量
    for (int i = 0; i < 5 + level; ++i) {
        for (int j = 0; j < width; j += 5) {
            enemies.push_back({i, j});
        }
    }
}

void Game::draw() {
    std::ostringstream oss;

    for (int i = 0; i < width + 2; i++) oss << border;
    oss << std::endl;

    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            if (j == 0) oss << border;

            bool drawn = false;

            // 畫敵人
            for (const auto &enemyPos : enemies) {
                if (enemyPos.first == i && enemyPos.second == j) {
                    oss << enemy;
                    drawn = true;
                    break;
                }
            }

            // 畫玩家子彈
            if (!drawn) {
                for (const auto &bulletPos : bullets) {
                    if (bulletPos.first == i && bulletPos.second == j) {
                        oss << bullet;
                        drawn = true;
                        break;
                    }
```

```
                }
            }

            // 畫敵人子彈
            if (!drawn) {
                for (const auto &enemyBulletPos : enemyBullets) {
                    if (enemyBulletPos.first == i && enemyBulletPos.second == j) {
                        oss << enemyBullet;
                        drawn = true;
                        break;
                    }
                }
            }

            // 畫玩家
            if (!drawn && i == height - 1 && j == playerX) {
                oss << player;
                drawn = true;
            }

            if (!drawn) oss << ' ';

            if (j == width - 1) oss << border;
        }
        oss << std::endl;
    }

    for (int i = 0; i < width + 2; i++) oss << border;
    oss << std::endl;

    oss << "Score: " << score << "  Level: " << level << std::endl;

    // 移動光標到頂端並輸出畫面
    COORD coord = {0, 0};
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
    std::cout << oss.str();
}

void Game::input() {
    if (_kbhit()) {
        char key = _getch();
        if (key == -32) { // special key (like arrow keys)
            key = _getch(); // get the actual key code
        }
        switch (key) {
            case 75: // 左箭頭
                if (playerX > 0) playerX -= 1; // 每次按鍵移動一個位置
                break;
            case 77: // 右箭頭
                if (playerX < width - 1) playerX += 1; // 每次按鍵移動一個位置
                break;
            case 32: // 空格鍵發射子彈
                bullets.push_back({height - 2, playerX});
                break;
```

```
        }
    }
}

void Game::logic() {
    // 更新玩家子彈位置
    for (auto &bulletPos : bullets) {
        bulletPos.first--;
    }

    // 移除越界的玩家子彈
    bullets.erase(std::remove_if(bullets.begin(), bullets.end(),
                                 [this](const std::pair<int, int> &bulletPos) {
                                     return bulletPos.first < 0;
                                 }),
                  bullets.end());

    // 檢查玩家子彈是否擊中敵人
    for (auto bulletIt = bullets.begin(); bulletIt != bullets.end();) {
        bool hit = false;
        for (auto enemyIt = enemies.begin(); enemyIt != enemies.end();) {
            if (bulletIt->first == enemyIt->first && bulletIt->second == enemyIt-
>second) {
                score++;
                enemyIt = enemies.erase(enemyIt);
                hit = true;
                break;
            } else {
                ++enemyIt;
            }
        }
        if (hit) {
            bulletIt = bullets.erase(bulletIt);
        } else {
            ++bulletIt;
        }
    }

    // 更新敵人子彈位置
    for (auto &enemyBulletPos : enemyBullets) {
        enemyBulletPos.first++;
    }

    // 移除越界的敵人子彈
    enemyBullets.erase(std::remove_if(enemyBullets.begin(), enemyBullets.end(),
                                      [this](const std::pair<int, int> &enemyBulletPos)
{
                                          return enemyBulletPos.first >= height;
                                      }),
                       enemyBullets.end());

    // 檢查敵人子彈是否擊中玩家
    for (const auto &enemyBulletPos : enemyBullets) {
        if (enemyBulletPos.first == height - 1 && enemyBulletPos.second ==
```

```
playerX) {
            gameOver = true;
        }
    }

    // 隨機生成敵人子彈
    if (rand() % 10 < level) { // 隨機概率隨等級增加
        if (!enemies.empty()) {
            int randomEnemy = rand() % enemies.size();
            enemyBullets.push_back({enemies[randomEnemy].first + 1,
enemies[randomEnemy].second});
        }
    }

    // 如果所有敵人都被擊敗，則進入下一關
    if (enemies.empty() && !gameOver) {
        nextLevel();
    }
}

void Game::start() {
    setup();
    while (!gameOver) {
        draw();
        input();
        logic();
        Sleep(100); // 調整FPS和遊戲速度
    }

    std::cout << "Game Over! Your score is: " << score << "  Level: " << level <<
std::endl;

    // 恢復控制台游標可見
    CONSOLE_CURSOR_INFO cursorInfo;
    GetConsoleCursorInfo(GetStdHandle(STD_OUTPUT_HANDLE), &cursorInfo);
    cursorInfo.bVisible = true;
    SetConsoleCursorInfo(GetStdHandle(STD_OUTPUT_HANDLE), &cursorInfo);

    playerManager.addPlayerScore(playerManager.getCurrentPlayerID(), score);
}
```