🧾 RPG 遊戲專案技術文件

# 專案簡介

本專案為一款以 C++ 所開發的文字冒險 RPG 遊戲，玩家可透過登入系統建立角色，進入遊戲世界進行商店購物、戰鬥挑戰與任務完成。專案整合多個子系統，包括登入系統、角色系統、道具系統、背包系統、商店系統、戰鬥系統與任務系統，採用物件導向設計以提高擴充性與可維護性。

# 二、系統架構說明

系統主要分為以下模組：

1. 登入系統：負責使用者註冊、登入與玩家資料載入。

2. 角色系統：管理玩家角色屬性與行為。

3. 物品系統： 物品生成、物品管理

3. 背包系統：背包管理、移出玩家物品以及放入。

4. 商店系統：提供購物介面，連結道具系統與玩家背包。

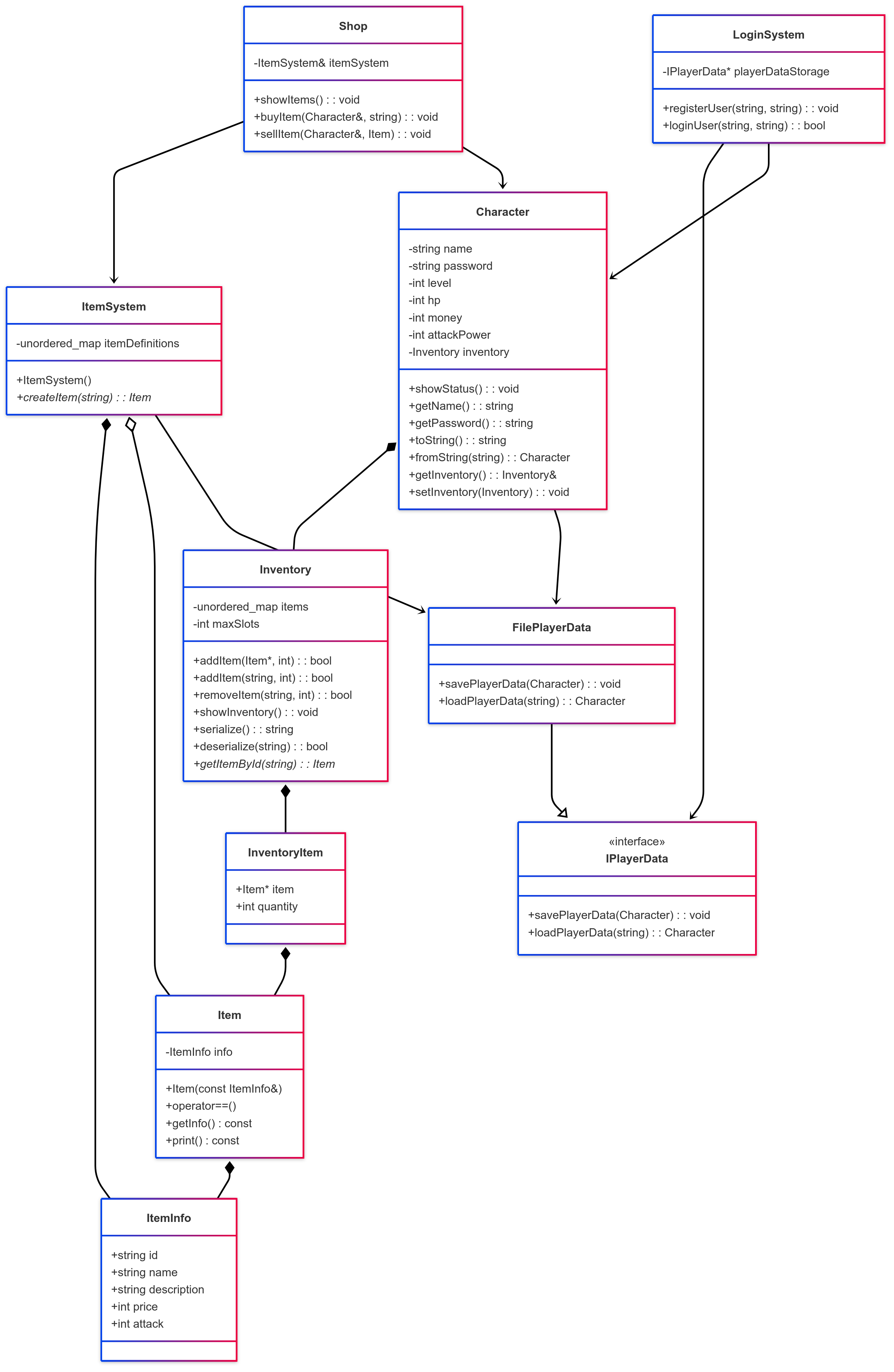
5. 戰鬥系統：與不同怪物進行回合制戰鬥。

6. 任務系統：整合戰鬥與劇情需求。

7. 資料儲存系統：透過檔案系統儲存使用者資訊與角色屬性。

# 2.1 流程圖與互動架構

以下為系統整體模組互動流程圖：



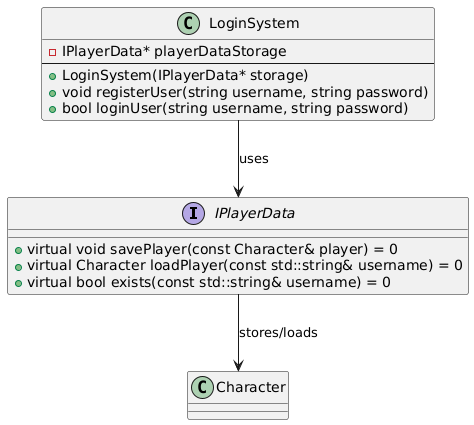
# 2.2 主要功能說明

程式碼撰寫細節跟邏輯在程式碼旁都有註解。

## 2.2.1 登入與註冊系統

**2.2.1.1 功能說明：**登入系統由 LoginSystem 類別負責，提供玩家帳號的註冊與登入功能。使用者可以創建帳號並儲存資料，或使用已註冊的帳號進行登入。

**2.2.1.2 角色類別設計：**



* **登入系統類別 (LoginSystem)：**

LoginSystem 類別負責管理玩家的帳號註冊與登入，並與玩家資料儲存系統（IPlayerData 接口）互動。

* **建構函數：**

初始化登入系統，並接收一個 IPlayerData 的儲存物件指標。這樣，登入系統可以根據玩家資料儲存系統的實現進行玩家資料的存取操作。

* **註冊功能 (registerUser)：**

檢查帳號是否已存在，若已存在則提示用戶帳號已存在；若帳號不存在則創建新玩家資料並儲存。

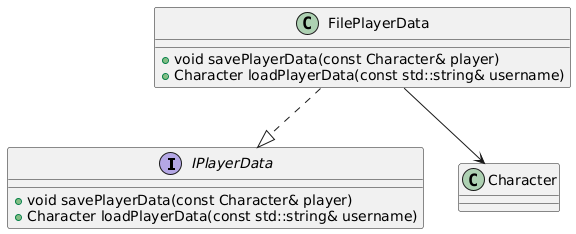
* **登入功能 (loginUser)：**

檢查帳號是否存在，若不存在則提示用戶帳號不存在；若帳號存在則進行密碼比對，若密碼正確則登入成功，否則提示密碼錯誤。

## 2.2.2 檔案系統

**2.2.2.1 功能說明：** 檔案系統由 FilePlayerData 類別負責，提供玩家資料的存取功能。使用者的資料會被儲存至檔案中，並可根據用戶名稱載入相應的資料。

**2.2.2.2 角色類別設計：**



* **檔案儲存類別 (FilePlayerData)：**

FilePlayerData 類別實作了 IPlayerData 介面，負責管理玩家資料的儲存與載入。

* **儲存玩家資料功能 (savePlayerData)：**

根據玩家名稱創建一個以玩家名稱命名的 .txt 檔案，並將玩家資料轉換為字串後寫入該檔案。

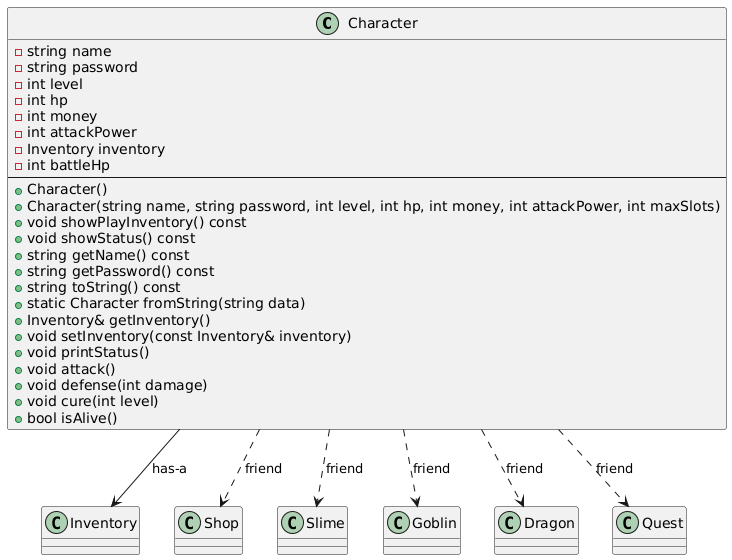
* **載入玩家資料功能 (loadPlayerData)：**

根據玩家名稱讀取對應的 .txt 檔案，並將檔案中的資料讀入並反序列化為 Character 物件，若檔案不存在則回傳一個空的 Character 物件。

## 2.2.3 角色系統

**2.2.3.1 功能說明：**角色系統負責管理角色的基本屬性（如名稱、等級、金錢、攻擊力和血量）以及角色的背包管理與購買行為。玩家可以註冊新帳號、登入、查看角色狀態，並進行物品的互動與管理。

**2.2.3.2 角色類別設計：**



* **角色類別：**

封裝角色的基本屬性（名稱、等級、生命值、金錢、攻擊力），並提供對背包的管理功能。

* **構造函式：**

初始化角色基本屬性，確保角色在遊戲開始時有正確的初始狀態。

* **顯示角色狀態：**

showStatus() 函數顯示角色的名稱、等級、血量、金錢和攻擊力，讓玩家了解角色狀況。

* **背包功能：**

getInventory() 和 setInventory() 函數管理角色背包，讓角色與物品系統互動。

* **序列化 ( toString() )：**

將角色的所有資料（名稱、密碼、等級、血量、金錢、攻擊力）轉換成字串格式，以便儲存或傳輸。背包資料也會附加在字串後面。

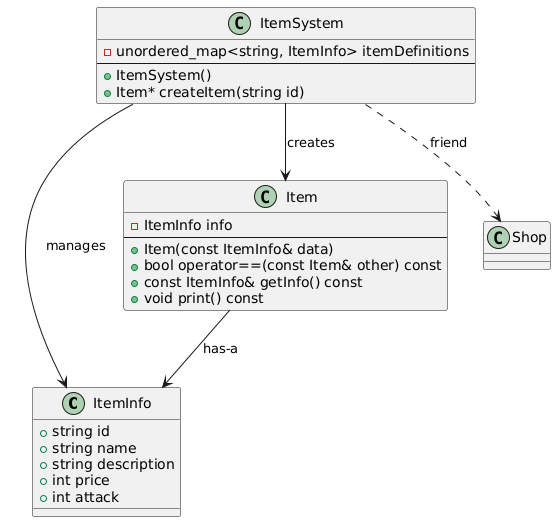
* **反序列化 ( fromString() )：**

從字串中解析出角色資料，並重建角色對象。包括解析名稱、密碼、等級、血量等資料，並使用這些資料建立角色物件，同時從字串中讀取並重建背包內容。

## 2.2.4 物品系統

**2.2.4.1 功能說明：**物品系統由 ItemSystem 控制，負責提供物品的資料與管理功能。透過物品類別 (Item) 和物品管理系統 (ItemSystem)，角色能夠創建、使用和管理物品。

**2.2.4.2 角色類別設計：**



**2.2.4.3物品類別 (Item)**

* **ItemInfo：**這是一個結構，儲存物品的靜態資料，包括物品的 ID、名稱、描述、價格和攻擊力等資訊。
* **Item：**代表遊戲中的單一物品。包含：

**構造函式：**將 ItemInfo 資料初始化為物品對象。

**operator==：**重載比較運算符，判斷兩個物品是否相等，根據物品的 ID、名稱、描述、價格和攻擊力進行比較。

**getInfo()：**返回物品的靜態資料 (ItemInfo)。

**print()：**顯示物品的詳細資訊，包括名稱、價格、描述和攻擊力。

------------------------------------------------------------------------------------------

**2.2.4.4物品管理系統 (ItemSystem)**

* **ItemSystem：**負責管理遊戲中的所有物品資料。包含：

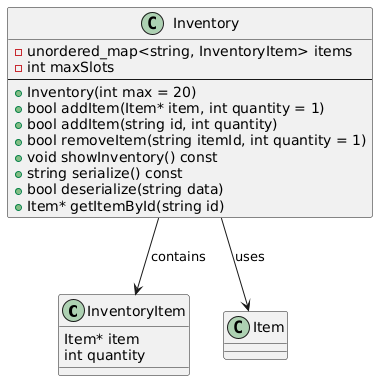
**構造函式：**初始化並載入預設的物品資料（例如鐵劍、鋼斧、龍牙劍等），這些資料存儲在 itemDefinitions 映射中，使用物品 ID 作為鍵。

**createItem()：**根據物品 ID 創建一個物品實例，並返回指向物品對象的指標。如果該 ID 不存在，則返回 nullptr。

## 2.2.5 背包系統

**2.2.5.1 功能說明：** 背包系統由 Inventory 類別負責，管理角色持有物品的新增、移除、顯示與儲存功能，並可與其他系統進行物品互動。

**2.2.5.2 角色類別設計：**

****

**2.2.5.3背包類別 (Inventory)：**

負責管理角色擁有的物品，包含物品的新增、移除、顯示、序列化與還原等功能。透過 unordered\_map 儲存每種物品及其數量，並限制背包最大格數。

* **建構函數：**初始化背包最大容量，預設為 20 格，確保角色在遊戲中有合理的物品持有上限。
* **新增物品功能：addItem()**

支援使用物品指標或物品 ID 來新增物品。若物品已存在則增加數量，否則新增至背包。新增物品時會檢查是否超過最大格數限制。

* **移除物品功能：removeItem()**

可移除指定數量的某種物品，若數量歸零則自動將該物品從背包移除，確保資料乾淨一致。

* **顯示功能：showInventory()**

列出背包中所有物品及其數量，並呼叫每個物品的 print() 函式顯示其詳細資料，提供玩家查看物品資訊。

* **序列化與反序列化：serialize(), deserialize()**

將背包資料轉為字串格式（itemId,quantity），便於儲存與還原。deserialize() 可從儲存的字串中重新建立對應物品並加入背包，支援遊戲進度儲存與載入。

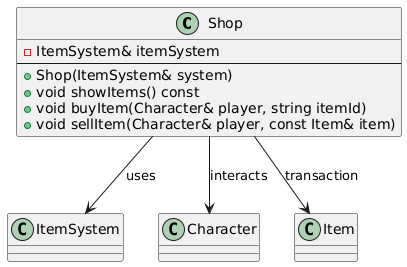
* **查找功能：getItemById()**

透過物品 ID 查詢背包中是否存在對應的物品並回傳指標，支援角色與物品互動的需求（例如裝備或使用道具）。

## 2.2.6 商店系統

**2.2.6.1功能說明：**商店系統由 Shop 類別負責，整合 ItemSystem 與角色系統，提供玩家與遊戲物品進行互動的功能，包括顯示可購買物品清單、處理購買流程並檢查玩家金錢與背包狀況。

**2.2.6.2角色類別設計：**



**2.2.6.3商店類別 (Shop)：**

提供玩家與遊戲中物品進行買賣的互動介面，與 ItemSystem 搭配，顯示商品清單、進行購買與販售功能。

* **建構函數：**

接收 ItemSystem 參考，讓商店能取得所有可販售的物品定義與價格資訊。

* **顯示物品清單：showItems()**

將商店中所有物品依照格式（包含 ID、名稱、描述、攻擊力、價格）顯示於終端機，並處理中文字對齊問題，提升可讀性。

* **購買物品功能：buyItem()**

讓玩家透過輸入物品 ID 購買物品，購買前會驗證金錢是否足夠，購買成功後扣除金錢並將物品加入角色背包中。

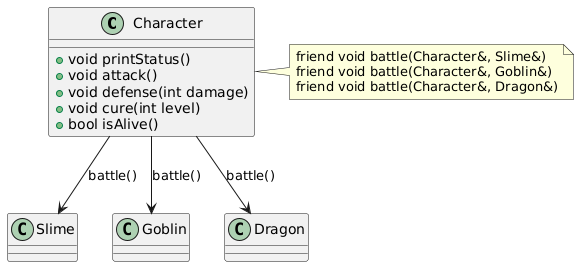
* **販售物品功能：sellItem()**

顯示玩家成功販售物品的訊息，目前為展示用途，未實作實際金錢加值與物品移除邏輯，可於未來擴充。

## 2.2.7 戰鬥系統

怪物以 Monster 為基底類別，Slime、Goblin、Dragon 繼承並實作特殊攻擊與防禦技能。

戰鬥由 Quest 函式主導，隨機選擇怪物並觸發對應 battle 函式。戰鬥採回合制方式進行，並根據傷害與回血邏輯變動血量。



## 2.2.8 任務系統

任務目前整合於戰鬥流程中，可日後擴充為劇情驅動任務或多人合作目標。

# 三、技術決策記錄

# 3.1資料結構與演算法 (Data Structures and Algorithms)

## 3.1.1 【角色系統 Character】

* 採用 `std：：string` 管理玩家名稱與密碼，避免 C-style string 安全問題。

## 3.1.2 【物品系統】

* **ItemInfo 結構體：**封裝物品靜態資料，包括 id, name, description, price, attack 等欄位，便於統一管理與重用。
* **ItemSystem 類別：**採用 std：：unordered\_map<std：：string, ItemInfo> 儲存所有靜態物品定義，以 ID 作為鍵值。

**unordered\_map 底層為 哈希表（Hash Table），查找效率為 O(1)**，非常適合頻繁查詢物品定義的情境。

如：建立物品 ( createItem )、顯示物品詳細資訊、在商店中查找定價與屬性。

## 3.1.3 【背包系統】

* **InventoryItem 結構體：**用來儲存背包內每個物品的實例（Item\* item）使用 Item\* item原因： 有效避免複製物品數據，尤其是在物品的數據量較大或複雜時。
* **Inventory 類別：**利用 std：：unordered\_map<std：：string, InventoryItem> 儲存所有物品的資料，鍵值為物品的 itemId，

使用原因也是頻繁查詢物品定義。

## 3.1.4 【商店系統】

* **ItemSystem&：**使用引用 (ItemSystem&) 是為了避免在 Shop 類別中重複儲存物品資料，並且讓物品資料管理更加集中在 ItemSystem 中。這樣的設計也能減少不必要的記憶體開銷。

## 3.1.5 【登入系統】

* IPlayerData 介面：使用介面 IPlayerData 可以使得 LoginSystem 類別的設計具有良好的擴展性，若未來需要改變資料存儲方式（例如從檔案轉到資料庫），只需實現一個新的 IPlayerData 類別即可。

# 3.2 效能與優化 (Performance and Optimization)

1. 使用 constructor initializer list 初始化成員變數。
2. 函式傳參多用 const std：：string& 或 reference，避免多餘複製。
3. 空間與存儲：使用指向 Item 類別的指標來管理物品，避免每個物品都複製一份資料，達到節省內存的目的。

# 3.3 程式碼約定 ( Code Convention and Coding Style )

專案遵循以下程式碼風格與命名慣例：

1. \*\*命名規則\*\*

- 類別名稱使用 PascalCase，如 `Character`, `Inventory`

- 成員變數使用 camelCase，如 `attackPower`, `battleHp`

- 常數使用全大寫，如 `RESET`, `GOLD`

2. \*\*風格統一\*\*

- 每個函式前皆附加簡要註解，說明用途。

- 使用 `#ifndef` 作為 Header Guard，防止重複定義。

- 盡量避免 `using namespace std;`，提升命名空間清晰度。

- 所有模組分開於 `.h / .cpp` 檔案中，結構清晰。

3. \*\*模組化原則\*\*

- 登入系統、角色系統、物品系統、戰鬥系統皆獨立設計。

- 各模組僅透過必要介面互相溝通，降低耦合。

# 四、開發與測試紀錄

# 4.1 單元測試（Unit Testing）

* **目的：**單元測試旨在對每個單獨的類別（Class）進行測試，確保每個模組或類別的功能符合預期，並且能在單獨運行時正確執行。
* **方法：**每次編寫新類別或修改現有類別時，都會先對其進行編譯並確認無誤。

撰寫簡單的main函數或測試類（Test Class），通過調用每個類別中的方法，測試其是否按照預期工作。

**測試步驟範例：**

* 撰寫一個測試類別（如LoginTest），在該類別中調用LoginSystem中的validateCredentials()方法，測試其在不同情況下（正確密碼、錯誤密碼）是否正確返回結果。
* 測試InventorySystem中加入物品、刪除物品等操作是否能正確執行，並通過輸出確認結果。
* 測試的範圍包括類別中的方法邏輯、邊界條件、錯誤處理等，確保每個類別能夠單獨運行並達到預期功能。

# 4.2 集成測試（Integration Testing）

* **目的：**集成測試的目的是測試系統中不同模組的協同工作，確保各模組能夠互相協 調並完成指定的功能，測試多個模組集成後的整體流程是否能正常運行。
* **方法：**測試系統中多個模組的交互是否正常。例如：登入流程會結合角色系統、檔案系統、登入系統、物品系統、背包系統等模組，一個完整的測試流程會測試這些模組之間的交互，確保整體流程的正確性。

在測試過程中會模擬用戶行為，像是註冊、登入、角色創建、物品管理等，測試每個流程是否按預期執行。

**測試步驟範例：**

* 測試用戶註冊和登入流程：

用戶先註冊一個新帳戶，然後通過登入系統進行登錄。

測試註冊後的用戶資料是否被正確存儲並能夠在登入時正確檢索。

* 測試角色創建和物品系統集成：

在成功登入後，創建一個角色並加入物品。

測試角色是否能夠成功接收物品，並檢查物品是否正確顯示在角色的背包中。

**測試完整流程：**從登入開始，檢查所有模組之間的協同作用，確保整個系統流程從用戶登入到背包管理都能順利進行。

# 4.3 發現的bug或技術問題與解法

1. **資料同步問題：**透過明確的物件生命週期與檔案讀寫封裝確保一致性。

2. **類別耦合問題：**使用友元函式與純虛擬函式設計降低耦合。

3. **命令列操作的使用者體驗：**以顏色與清除畫面命令改善操作感。

# 五、部署與維護規劃

1. **開發平台：** 硬體平台：x86、操作系統：window 10。
2. **開發環境：** Visual Studio Code。
3. **編譯專案：**在專案目錄下執行以下命令來編譯專案：

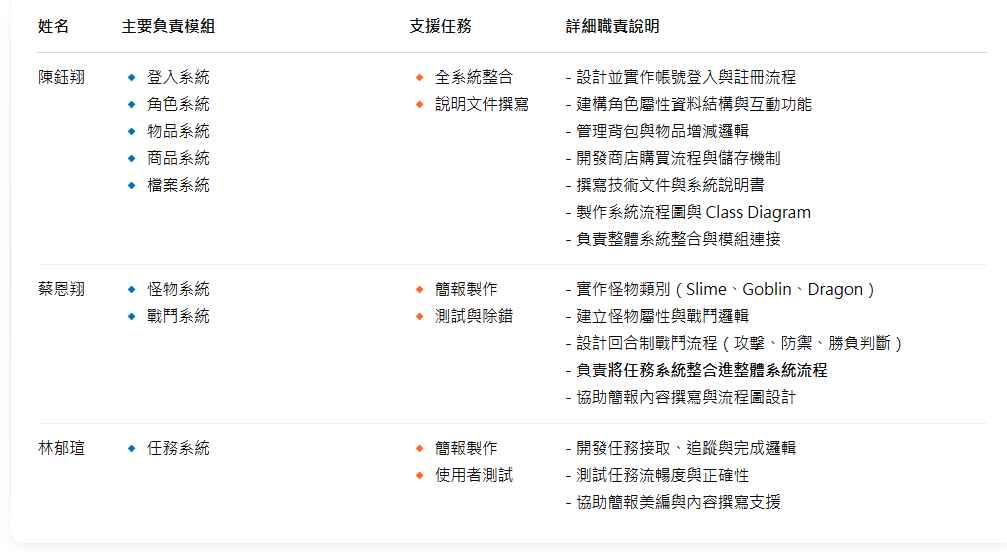
**g++ Character.cpp FilePlayerData.cpp Inventory.cpp ItemSystem.cpp LoginSystem.cpp monster.cpp Quest.cpp Shop.cpp battle.cpp -o RPG\_game**

1. **專案管理：**

Github： [https：//github.com/squirrel1209/Kway\_-.git](https://github.com/squirrel1209/Kway_-.git)



1. **分工：**



# 六、未來擴充建議

實作 **Socket** 程式設計，實現多人連線對戰與互動模式，讓遊戲變成支持協作或競爭的線上遊戲，並支援以下玩法：

1. 競技場模式：即時對戰，獲得排名與獎勳。
2. 合作任務：玩家組隊協力完成劇情任務或挑戰。
3. 交易與社交系統：玩家互相交換物品與裝備。
4. 多人副本：多人合作挑戰強大敵人。