



Network Programming Final - 遊戲商城系統

HW3 : Game Store System

作業目標

本作業的核心目標，是實作一個整合遊戲大廳（Lobby）與遊戲商城（Store）的平台。你將在這次作業中整合先前 HW1 / HW2 的成果，打造一個「可實際 Demo、可多人操作」的完整遊戲體驗，同時提供一個給開發者使用的上架平台。

在本次作業中，你們需要完成以下幾項核心需求：

1. 建立開發者平台

建立一個讓開發者可以管理自己遊戲的系統。開發者可以在這個平台上：

1. 上傳遊戲
 - a. 雙人 CLI 遊戲
 - b. 雙人 GUI 遊戲
 - c. 多人以上之小遊戲
2. 更新遊戲（版本更新）
3. 下架遊戲

為了讓遊戲被上傳後可以在大廳端穩定啟動，你需要設計一套統一的遊戲規格（例如檔案結構、啟動方式、必要的介面等），讓不同開發者寫出的遊戲都能被同一套平台正確管理與啟動。

2. 優化遊戲大廳與商城平台

你需要提供一個介面，讓玩家可以：

1. 瀏覽目前大廳狀態

- 玩家列表
- 房間列表
- 上架遊戲列表

2. 建立房間並遊玩遊戲

3. 瀏覽商城中的遊戲

- a. 檢視遊戲詳細資訊（例如：名稱、作者、版本、簡介等）
- b. 為遊戲評分與撰寫評論

整個大廳與商城的體驗應該讓使用者能「看得懂現在有哪些遊戲、有哪些房間、有哪些玩家正在做什麼」，並且可以順利完成選擇遊戲 → 建立房間 → 進入遊戲的流程。

3. 支援玩家自由遊玩與更新遊戲版本

玩家在遊玩遊戲之前，會先從商城下載最新版本的遊戲。系統需要支援：

1. 遊玩前下載對應遊戲，或自動更新至該遊戲的最新版本
2. 在遊玩時，自動啟動對應的 game client，並連線至 server 端已啟動的 game server
3. 遊玩結束後，自動釋放相關資源，並回到房間或大廳等待狀態

完整的 Use Case 與操作流程，會在下方「作業詳細需求」中補充與細化。

系統架構與 Demo 說明

Client 端互動方式：Menu-driven Interface

本次作業要求 Client 端採用選單式互動介面（Menu-driven Interface）。

也就是說，使用者應該可以透過一層一層的選單來完成所有操作，而不是仰賴複雜的指令列參數或隱藏指令。

這樣的設計有幾個目的：

- 讓使用者（包含 Demo 時的助教）在**不需要記住指令**的情況下，也能順利完成所有流程。
- 降低學習成本，讓系統的行為可以「看畫面就懂」，而不是「靠你在旁邊口頭解釋」。
- 更容易引導使用者依照正確的順序操作，例如上架 → 瀏覽 → 建立房間 → 遊玩等。

系統在每個步驟都應該提供清楚的選項編號，引導使用者輸入對應數字來執行操作。

範例互動流程：

==== 商城主選單 ===

1. 瀏覽遊戲
2. 安裝遊戲
3. 啟動遊戲
4. 作者模式
5. 離開

請選擇功能 (1-5):

本次作業 Demo 全程中，**只允許在一開始啟動 Server 與 Client 的時候，使用指令手動啟動程式**。

若在 Demo 過程中，任何一項系統功能需要透過「額外輸入 Shell 指令或額外參數」才能完成，該功能將視為**未完成此評分項目**。



我們鼓勵同學盡可能以 GUI 介面實作完整功能（在 AI 工具的幫助下，這件事情應該會變得相對容易）。

在評分時，我們會特別關注：

- 各種狀態切換是否處理正確
- 介面在不同狀態下是否有清楚回饋
- 畫面更新與渲染是否一致且不會讓人迷失

在設計 Menu 時，請注意以下原則：

- 建議**每一層選單最多提供約 5 個功能項目**。

- 若你把所有功能都塞在同一層 Menu，例如讓使用者在單一畫面就要在 10 個以上的操作中做選擇，會嚴重降低可用性。
- 這樣的設計會被視為 UX 介面設計不足，在互動性相關的項目中可能會被扣分。

UX 建議（從 Demo 操作者角度設計）

在設計整個系統時，請優先站在「實際 Demo 時負責操作的助教」的角度思考。

在正式 Demo 時，我們會儘量讓助教自行操作，而不是由學生全程帶著流程。所以理想狀況是：

助教只看你的 README 與畫面中的說明，就能在幾乎不需要你口頭解說的情況下，完成：

開發遊戲 → 上架遊戲 → 建立房間 → 下載並啟動遊戲 → 遊玩 → 結束並回到大廳 的完整流程。

你可以從以下幾個面向檢查自己的 UX 設計品質：

1. 是否提供快速啟動方式

- 是否提供 啟動腳本、Makefile、Docker 設定或 virtualenv 等機制？
- 助教是否能依照 README 的指示，一次性啟動 Developer Client / Lobby Client，而不是手動下許多零散指令？

2. 畫面文字與資訊呈現是否清楚

- 是否在畫面上提供必要、但不冗長的操作說明？
- 是否使用清晰的排版（例如表格、列表）顯示玩家列表、房間列表、遊戲列表？
- 當操作成功或失敗時，是否有明確的提示訊息（例如：建立房間成功 / 遊戲下載失敗 / 版本已更新等）？

3. User Flow 是否連貫而非碎片化

- 盡量避免讓使用者必須「先在 A 畫面抄資料，再自己輸入到 B 畫面」。
- 例如原本可能的流程是：
 1. 列出遊戲列表
 2. 手動記住遊戲編號或名稱
 3. 再到另一個選單輸入編號建立房間

這種流程不僅容易出錯，也不直覺。

- 一個較佳的 UX 設計可以是：
 1. 使用者選擇「遊玩遊戲」
 2. 系統自動列出目前可遊玩遊戲列表
 3. 使用者在同一個 flow 內直接選擇要玩的遊戲
 4. 系統自動建立房間並顯示結果

這樣的設計不只是 UX 的加分，同時也有助於降低非同步系統中因資料過舊導致的錯誤（例如 30 秒前看到的列表，現在已經不適用）。

4. 輸入體驗是否友善

- 儘量以編號選項取代手打文字或複雜參數，例如：
 - 輸入「1」代表進入商城
 - 輸入「2」代表建立房間
- 當使用者輸入錯誤（例如輸入不存在的選項編號）時，系統是否：
 - 純出清楚的錯誤訊息
 - 提示合法輸入範圍
 - 重新要求輸入，而不是直接崩潰或結束程式

5. Input 與背景事件的協調

- 當 Client 正在等待使用者輸入時，如果有來自 Server 的通知（例如房間狀態更新、玩家加入或離開），你的系統是否有合理的處理策略？
- 例如：
 - 是否會適度中斷當前輸入並重新顯示狀態？
 - 是否有獨立的 thread 處理 input，並妥善避免阻塞導致 UI 沒有反應？

Demo 環境要求與三種角色

本次作業的核心之一，是「網路程式在不同角色間的互動」與「程式碼上傳 / 下載 / 版本管理」。

在實際執行與 Demo 時，我們會從三個角色的視角來看系統：

1. Server 端

- Data / DB Server

- Game / Developer / Lobby Server

2. Developer (遊戲開發者)

3. Player (一般玩家)

實務上，這三種角色會對應到**三份彼此獨立的 Codebase**。其中，Server 端依照 HW2 的要求，必須部署在系計提供的 Linux 環境上執行。

以下分別說明三種 Codebase 的預期結構與職責。

Developer : 遊戲開發端 Codebase

Developer 端代表遊戲開發者的工作環境，負責實作與測試遊戲，並透過 Developer Client 與後端 Server 溝通，完成上架與更新。

你可以參考以下目錄結構進行設計（實際命名與細節可自行調整，但精神需類似）：

```
|── create_game_template.py  
|── developer_client.py  
|── games      # 開發中遊戲（僅供開發者本地測試）  
|   |── agario  
|   |── my_game  
|   |── snake_tutorial  
|── template    # 遊戲範本與腳手架
```



`template` 資料夾與 `create_game_template.py` 腳本，可以提供一套你自訂的「遊戲規格與骨架」，讓開發者透過腳本一鍵建立符合你平台規則的新遊戲專案。

請注意：

- `games` 中的內容只用於開發者本地開發與測試。
- 玩家不應該直接接觸到這裡面的程式碼，也不應直接從這裡執行遊戲。
- 開發者完成開發後，應透過 Developer Client 將遊戲上傳到 Server 端。

Player : 一般玩家端 Codebase

Player 端代表一般玩家在自己電腦上的遊玩環境。玩家透過 Lobby Client 連線到 Lobby Server，瀏覽可遊玩遊戲、下載、啟動並加入房間。

你可以參考以下目錄結構：

```
└── create_game_template.py  
└── lobby_client.py  
└── downloads      # 玩家下載的遊戲  
    ├── Player1  
    │   ├── agario  
    │   ├── my_game  
    │   └── snake_tutorial  
    ├── Player2  
    └── Player3  
└── template
```

- `downloads/` 底下的遊戲，必須是透過 Server 下載取得。
- 玩家不可以任意修改下載下來的遊戲內容。
- 每一位玩家的下載內容需要獨立存放（如上例中的 `Player1`、`Player2` 等）。



為什麼每位玩家需要有獨立的下載資料夾？

在真實情境中，每位玩家都會在各自的電腦上操作。為了在同一台 Demo 機器上模擬多位玩家的獨立行為，我們以不同資料夾區分各玩家的下載內容。這樣在版本更新與下載時間不同的情況下，就能自然產生：

- 有人已更新到最新版本
- 有人還停留在舊版本

這會是我們在 Demo 時會特別關注的測試情境之一。

Server 端 Codebase

Server 端依照 HW2 的要求，需部署於系計提供的 Linux 環境上運行。

你可以依照系統設計拆分為多個服務模組，以下是常見的職責分類：

- **Data / DB Server**
 - 負責資料儲存與管理，可沿用 HW2 的資料庫架構，或自行擴充。
 - 唯一明確要求：**Server 重新啟動後，資料不得遺失。**
- **Developer Server**

- 提供給 Developer Client 使用。
 - 處理遊戲上傳、更新、下架等請求。
 - 負責將上傳後的遊戲檔案放置在統一管理的「上架遊戲空間」。
- **Lobby Server**
 - 提供給 Lobby Client 使用。
 - 維護大廳狀態：玩家、房間、每個房間綁定的遊戲版本等。
 - 協助建立房間、加入房間，以及在需要時啟動對應的 game server。
- **Uploaded Game (上架遊戲檔案存放區)**
 - 用來存放開發者已上架的遊戲檔案。
 - 應視為「真實可供玩家下載的遊戲來源」，不得任意修改其中內容。
 - 你需要設計一套機制，讓 Lobby Server 能夠根據這裡的內容：
 - 提供玩家下載適當版本的遊戲
 - 在需要時啟動對應版本的 game server

整理一下，目前一款遊戲的檔案可能會同時存在三個地方：

1. **Developer 端 (開發中遊戲)**
 - 開發者用來編輯與本地測試的版本。
2. **Developer Server (上架版本)**
 - 作為「商城後端」實際提供給玩家下載的版本。
3. **Lobby Client (玩家下載版本)**
 - 玩家裝置上實際被啟動與遊玩的版本。

由於上傳、更新與下載時間不一致，這三者之間可能存在不同版本。

如何清楚地管理這些版本關係，並在玩家啟動遊戲時給出合理的行為，是本次作業的重要挑戰之一。

Demo 方式與事前準備

在進入實作細節前，先說明 Demo 的流程與事前準備，方便你在開發過程中就朝這個目標規劃。

在正式 Demo 之前，你需要先完成：

1. **將程式碼推上 GitHub**

- 包含 Developer 端、Player 端與 Server 端的程式碼。
- 資料夾結構與 README 需清楚易懂。
- Demo 前將連結繳交到 E3 作業區上面

2. 部署 Server 端程式

- 依照 HW2 的環境要求，在系計提供的 Linux 機器上部署並測試。
- 在 Demo 前清空原有的資料（你可以寫腳本開放測試資料的清空和注入）
- 確認 Server 端在沒有你人工干預的情況下，可以長時間穩定運作。

在 Demo 時，整體流程預期如下：

1. 助教會將你提供的 Client 端程式碼從 GitHub **pull** 到助教本地電腦。
2. 你提交的程式碼中，必須包含一份**完整且清楚的 README**，內容至少要說明：
 - 如何啟動 Developer Client
 - 如何啟動 Lobby Client
 - 如何設定或修改連線資訊（例如 IP、Port）
3. 助教會依照 README 的指示，實際啟動程式並操作整個流程。

在設計專案時，請特別考慮：

- 不同助教的電腦環境可能略有差異
- 若部署或安裝過程過於複雜，Demo 時間可能會大幅延誤

因此建議你：

- 使用 `Makefile`、`啟動腳本`、`Docker`、`python venv` 等方式，讓助教可以用儘量少的指令完成環境安裝與啟動。
- 把所有啟動流程盡可能腳本化與自動化，減少手動輸入的步驟。

在 Demo 時，除了基本功能展示外，助教也可能會詢問你：

- 系統架構設計與模組切分的原因
- 通訊協定、封包格式與錯誤處理策略
- 遊戲版本管理與檔案同步的設計考量

請在 Demo 前先做好準備，以免浪費彼此的時間。

作業詳細需求

在 Developer / Player 的各種 Use Case 之前，帳號系統需遵守以下簡單規則：

- **帳號類型分開管理**：Developer 帳號與 Player 帳號分別獨立，不共用同一個帳號來源。
- **簡單帳號密碼註冊**：
 - 使用者以「帳號名稱 + 密碼」註冊。
 - 同一類型帳號中，帳號名稱不得重複；若欲註冊帳號已存在，系統應提示「帳號已被使用」，並要求改用其他名稱。
- **登入驗證**：
 - 登入時需檢查帳號是否存在、密碼是否正確。
 - 若帳號或密碼錯誤，系統應提示「帳號或密碼錯誤」，並允許重新輸入。
- **避免重複登入**：
 - 同一帳號同一時間只允許一個有效登入 Session。
 - 若同一帳號在另一個 Client 再次登入，系統可以選擇：
 - 拒絕新登入並提示「帳號已在其他裝置登入」，或
 - 讓新的登入覆蓋舊的 Session，並讓舊的登入自動失效（擇一實作即可）。

上述規則適用於 Developer Client 與 Lobby Client 的所有 Use Case。

開發者 (Developer)

開發者端需要使用**開發者帳號登入**後才能操作，且此帳號身分與一般 Player 帳號互相獨立。

在所有與遊戲管理相關的操作（上架 / 更新 / 下架）中，開發者**只能操作與自己帳號相關的遊戲**（例如自己建立或被指派管理的遊戲），不得直接修改其他開發者的作品。

以下是幾個推薦的 Use Case 範例，作為你設計與實作 Developer 端功能的參考。

實作時不強制要求流程一模一樣，但應達成相同的使用目標。

Use Case D1：開發者上架一款新遊戲

- **目標**

讓開發者可以把本地已完成的遊戲，上架到平台，讓玩家之後可以在商城中看到並下載遊玩。

- **前置條件**

- 開發者已在本地完成至少一款可執行的遊戲版本。
- 開發者端程式（Developer Client）已能成功連線到後端（Developer Server）。
- 開發者已經在系統中有可辨識的身分（例如帳號、名稱或其他識別方式），並已成功登入。

- **步驟（實作時可調整具體互動細節）**

1. 開發者啟動 Developer Client，登入並進入「開發者主選單」。
2. 開發者在選單中選擇「上架新遊戲」或功能名稱相近的選項。
3. 系統引導開發者輸入或選擇必要資訊，例如：
 - 遊戲名稱
 - 簡短介紹或描述
 - 遊戲類型（CLI / GUI / 人數上限等）
 - 要上傳的遊戲位置（或由系統提供可選的清單）
 - 遊戲相關的設定檔（例如 config 檔），可以選擇手動輸入設定，也可以直接由配置檔讀取，確保上架內容本身就包含啟動與連線所需的資訊
4. 開發者確認送出上架申請。
5. 系統與 Server 端互動，完成遊戲資訊與檔案的上傳。
6. 上傳完成後，系統顯示上架成功的回饋訊息，並能讓開發者看見這款遊戲已出現在「我的遊戲列表」中。

- **錯誤處理情境（範例）**

以下為建議情境，實作時可依自己設計調整：

- 若必填欄位未填（例如遊戲名稱或類型），系統應提示缺少哪一項資料，並要求補齊後才能送出。
- 若選取的遊戲路徑不存在或檔案無法讀取，系統應顯示清楚的錯誤訊息，而不是直接關閉程式。
- 若 config 檔缺少必要欄位（例如啟動指令或連線埠），系統可以：
 - 提示缺少的欄位名稱，並允許使用者改用手動輸入補上，或

- 拒絕上架並說明原因。
- 若 Server 端暫時無法連線或上傳失敗，系統應提示「上架失敗」與簡要原因（例如連線逾時），並提供重試或取消的選項。
- **預期結果**
 - 開發者可以清楚知道這款遊戲是否已成功上架，若失敗也能知道原因並有機會修正後重試。
 - 在商城 / 大廳相關介面中，可以看到這款新遊戲的基本資訊。
 - 之後玩家能透過 Player 端流程，下載並遊玩這款遊戲。

Use Case D2：開發者更新已上架遊戲版本

- **目標**

當開發者修正 Bug 或增加新功能時，可以更新已上架的遊戲版本，而不需要重新建立新的遊戲條目。
- **前置條件**
 - Game Store 中已存在一款由此開發者上架的遊戲。
 - 開發者在本地已準備好新的遊戲版本。
 - 開發者端程式可以辨識「哪一款遊戲要被更新」，並確認該遊戲屬於目前登入的開發者帳號。
- **步驟**
 1. 開發者啟動 Developer Client，登入並進入「我的遊戲」或類似功能。
 2. 系統列出開發者已上架的遊戲清單（包含目前版本資訊）。
 3. 開發者選擇其中一款要更新的遊戲。
 4. 系統引導開發者指定新的版本（例如：輸入版本號、選擇新檔案來源、撰寫更新說明等）。
 5. 開發者確認更新。
 6. 系統將新的遊戲內容與版本資訊送到 Server 端，並在完成後給出明確的成功訊息。
- **錯誤處理情境（範例）**
 - 若開發者嘗試更新不屬於自己帳號的遊戲，系統應拒絕操作並提示「無權限更新此遊戲」。

- 若指定的新版本檔案不存在或讀取失敗，系統應顯示錯誤並要求重新選擇。
- 若版本號格式不合法（例如空白或重複使用舊版號且未被允許），系統應提示錯誤並要求重新輸入。
- 若在更新過程中與 Server 連線中斷，系統應回報「更新失敗」，並清楚區分：
 - 伺服器端實際是否已更新成功（若可以查詢），或
 - 需要使用者重新嘗試更新。

- **預期結果**

- 這款遊戲在後台被標記為最新版本。
- 之後玩家在下載或更新這款遊戲時，會以新版本為主。
- 若有玩家仍持有舊版本，系統可以在之後的設計中，透過更新或提示方式引導他們升級（具體方式請自行決定）。

Use Case D3：開發者下架一款遊戲

- **目標**

讓開發者可以從商城中暫時或永久移除一款遊戲，避免玩家繼續下載或建立新房間。

- **前置條件**

- 系統中已存在由該開發者上架的遊戲。
- 開發者有權限操作該遊戲（例如不是其他作者的作品）。

- **步驟**

1. 開發者啟動 Developer Client，登入並進入「我的遊戲」清單。
2. 系統列出該開發者目前已上架的遊戲。
3. 開發者選擇要下架的遊戲，並在選單中選擇「下架」或相似功能。
4. 系統提示下架後的影響（例如：無法再被新玩家下載或建立新房間）。
5. 開發者確認下架。
6. 系統將下架請求送至 Server，並在完成後給出明確的回饋。

- **錯誤處理情境（範例）**

- 若開發者嘗試下架不屬於自己帳號的遊戲，系統應拒絕並提示「無權限下架此遊戲」。
- 若該遊戲目前仍有進行中的房間或遊戲局，系統可以：
 - 阻止下架並提示需先關閉相關房間，或
 - 允許標記為「不再接受新房間」，但保留現有房間（具體策略由各組自行設計）。
- 若與 Server 沟通時發生錯誤（例如連線失敗），系統應提示「下架失敗」並保留原狀，不應出現前後端狀態不一致而讓遊戲「半消失」。

- **預期結果**

- 該遊戲不再出現在一般玩家可見的「可下載 / 可遊玩」列表中。
 - 已經下載或正在遊玩的玩家行為，由各組自行設計（例如：允許舊玩家繼續游玩、或在下次啟動時給予提示等）。
-

玩家 (Player)

玩家端需要具備**玩家帳號的註冊與登入機制**，並與開發者帳號分開管理。登入後的玩家身分，會影響其能看到與能操作的資訊，例如：

- 僅能使用自己的玩家身分加入房間、建立房間、發送聊天訊息等
- 評分與留言必須能追溯到實際玩家帳號
- 不同權限的玩家（例如一般玩家、管理員）可以在系統中被區分，實作時可以自行設計等級與對應行為

以下是幾個推薦的 Use Case 範例，作為你設計與實作 Player 端功能的參考。

實作時不要求逐步完全一樣，但應達成類似的玩家體驗。

Use Case P1：玩家瀏覽遊戲商城與詳細資訊

- **目標**

讓玩家可以在一個清楚的介面中，看到目前有哪些遊戲可以遊玩，並能查看每款遊戲的基本說明與評價。

- **前置條件**

- 玩家已經擁有玩家帳號，並成功登入 Lobby Client。
- 至少有一款遊戲已在平台上架。

- Player 端程式（Lobby Client）已能成功連線到 Lobby / Store 相關服務。

- **步驟**

1. 玩家啟動 Lobby Client，進入「商城」或「遊戲列表」相關選單。
2. 系統顯示目前可供下載或遊玩的遊戲列表（例如以表格或列表呈現）。
3. 玩家從列表中選擇一款遊戲。
4. 系統顯示該遊戲的詳細資訊，例如：
 - 遊戲名稱與作者
 - 遊戲簡介
 - 支援人數、遊戲類型
 - 評分與部分玩家評論（若有）
5. 玩家可以選擇返回列表、或進一步進行下載 / 建立房間等後續操作。

- **錯誤處理情境（範例）**

- 若目前沒有任何上架遊戲，系統應清楚顯示「目前沒有可遊玩的遊戲」，而不是呈現空白畫面。
- 若在載入遊戲列表時與 Server 連線失敗，系統應提示「列表載入失敗」並提供重新整理或稍後重試的選項。
- 若某些遊戲資訊缺漏（例如缺少簡介），系統可以：
 - 顯示預設文案（例如「尚未提供簡介」），而不是中斷整個列表的顯示。

- **預期結果**

- 玩家能夠清楚理解每款遊戲的基本特性與玩法概念。
- 玩家在決定要下載或遊玩前，不需要查閱外部文件或口頭說明。

Use Case P2：玩家下載並更新遊戲版本

- **目標**

讓玩家可以在開始遊玩前，一鍵下載或更新到某款遊戲的最新可用版本。

- **前置條件**

- 該遊戲已在平台上架，且有至少一個可下載版本。
- 玩家端具備基本儲存空間與權限，能在本地建立下載目錄。

- **步驟**

1. 玩家在 Lobby Client 的選單中，選擇「下載遊戲」或同類功能。

2. 系統列出目前可供下載的遊戲。

3. 玩家選擇其中一款遊戲。

4. 若玩家尚未下載過該遊戲，系統執行「首次下載」流程。

若玩家已有舊版本，系統可以：

- 呈現目前本地版本與伺服器版本的差異，或

- 直接提示「有新版本，是否更新」。

5. 玩家確認下載或更新。

6. 系統向 Server 索取檔案，並在下載完成後給出成功提示。

- **錯誤處理情境（範例）**

- 若下載過程中連線中斷或檔案損毀，系統應標記該下載為失敗，不應讓半套檔案被誤當成可用版本，並提供重新下載選項。

- 若伺服器回報該版本已被下架或不存在，系統應提示玩家「此版本已不可下載」，並返回遊戲列表。

- **預期結果**

- 對玩家来说，「我要玩某款遊戲」之前，不需要自己去找檔案路徑，只要透過一個明確的操作就能取得最新版本。

- 系統能清楚辨識玩家目前持有的版本狀態，避免玩家在不知情下長期停留在舊版本。

Use Case P3：玩家建立房間並啟動遊戲

- **目標**

讓玩家可以透過簡單的操作，完成「選擇遊戲 → 建立房間 → 啟動遊戲並連線到對應 Server」的流程。

- **前置條件**

- Player 端可以連線到 Lobby Server。

- 玩家欲遊玩的遊戲已經在本地完成下載，或系統能在流程中自動協助下載。

- **步驟**

1. 玩家在 Lobby Client 中選擇「建立房間」或「開始遊戲」相關功能。
 2. 系統自動列出目前可用的遊戲列表（或玩家先選遊戲，再選擇建立房間）。
 3. 玩家選擇一款遊戲。
 4. 系統檢查玩家是否已有對應版本：
 - 若尚未下載，提示玩家下載並完成後再繼續。
 - 若有舊版本，可視設計提示更新。
 5. 條件滿足後，系統向 Lobby Server 發出建立房間請求。
 6. 房間建立成功後，系統顯示房間資訊（例如房號、目前人數、遊戲名稱等）。
 7. 根據系統設計，玩家端啟動對應的 game client，並連線到相應的 game server。
- **錯誤處理情境（範例）**
 - 若在建立房間時，該遊戲已被開發者下架或標記為不再接受新房間，系統應拒絕建立並提示原因。
 - 若 Lobby Server 回報目前人數已達上限，系統應告知玩家無法建立新房間，並可建議加入其他現有房間（若有）。
 - 若啟動 game client 或連線到 game server 失敗（例如連線逾時、port 不通），系統應：
 - 顯示錯誤原因（簡短即可），並
 - 將玩家狀態退回到合理位置（例如回到房間或大廳），避免卡在「半進入遊戲」的狀態。

- **預期結果**
 - 玩家可以在不需要理解底層連線細節的情況下，順利從大廳進入遊戲。
 - 房間狀態在大廳中有清楚反映，例如可以看到新建立的房間、目前等待中的玩家等。

Use Case P4：玩家對遊戲進行評分與留言

- **目標**

提供玩家在遊戲結束後，能對遊戲進行簡單的評分與文字回饋，讓其他玩家與開發者能參考。
- **前置條件**

- 玩家曾經成功啟動並結束某款遊戲的遊玩流程。
 - 系統有儲存基本遊戲紀錄或能辨認「哪些玩家玩過這款遊戲」。
- **步驟**
 1. 玩家在 Lobby Client 中，進入「我的紀錄」或在遊戲詳細資訊畫面中，選擇「評分與評論」相關功能。
 2. 系統確認玩家是否具備評分資格（例如真的玩過這款遊戲）。
 3. 玩家輸入評分（例如 1–5 分）與可選擇的評論文字。
 4. 玩家送出評價。
 5. 系統將評分與留言傳送至 Server，並在成功儲存後顯示回饋。
 - **錯誤處理情境（範例）**
 - 若玩家尚未實際遊玩該遊戲，系統應拒絕建立評價，避免出現「未玩先評」的情況。
 - 若評分超出允許範圍（例如輸入 0 或 10），系統應提示合法範圍並要求重新輸入。
 - 若評論內容過長或包含無法處理的字元，系統可以：
 - 限制字數並提示玩家縮短內容，或
 - 提供明確的失敗原因而不是直接丟失輸入內容。
 - 若在送出評價時與 Server 連線失敗，系統應保留玩家剛剛輸入的內容（例如暫存在記憶體或檔案中），並允許玩家稍後重試送出，而不是整段內容直接消失。
 - **預期結果**
 - 未來在查看遊戲詳細資訊時，可以看到來自多位玩家的平均評分與部分評論。
 - 開發者可以參考這些留言，作為後續更新與調整的依據（實際如何呈現請自行設計）。

Plugin 相關 Use Case（加分用）

以下 Use Case 為 Plugin（例如房間內群組聊天）相關情境，主要用於加分評分。

Use Case PL1：玩家查看可用 Plugin 清單

- **目標**

讓玩家在 Lobby 中可以看到目前有哪些可用的 Plugin (例如「房間內群組聊天 Plugin」)，但就算完全不安裝也不會影響核心流程。

- **前置條件**

- 玩家已登入 Lobby Client。
- 系統端已定義至少一個 Plugin 資訊 (名稱、簡介、版本等)。

- **步驟**

1. 玩家在 Lobby 主選單中選擇「Plugin / 擴充功能」選項。
2. 系統顯示目前可用 Plugin 清單 (例如：[Room Chat Plugin](#))，並展示基本資訊：
 - Plugin 名稱
 - 簡短介紹 (例如「在房間內提供群組聊天視窗」)
 - 狀態 (未安裝 / 已安裝 / 可更新)

- **預期結果**

- 玩家可以清楚知道有哪些 Plugin 可以選擇。
- 若完全不理會 Plugin 清單，仍可以正常使用大廳與遊戲的所有核心功能。

Use Case PL2：玩家安裝或移除 Plugin

- **目標**

讓玩家可以選擇性地安裝或移除某個 Plugin，且安裝／移除過程不會破壞主系統。

- **前置條件**

- 已通過 PL1，可以看到 Plugin 清單。

- **步驟**

1. 玩家在 Plugin 清單中選擇某個 Plugin (例如 [Room Chat Plugin](#))。
2. 系統顯示該 Plugin 的詳細資訊與目前狀態 (未安裝 / 已安裝)。
3. 若尚未安裝：
 - 玩家選擇「安裝」。
 - 系統下載並啟用 Plugin，完成後顯示安裝成功。
4. 若已安裝：

- 玩家可以選擇「移除」。
 - 系統停用並移除該 Plugin，完成後顯示移除成功。
- **錯誤處理情境（範例）**
 - 下載或載入 Plugin 失敗時：
 - 顯示「安裝失敗」，並保持主系統功能不受影響。
 - 移除 Plugin 時：
 - 僅移除 Plugin 功能，不影響玩家原本的大廳與遊戲流程。
 - **預期結果**
 - 玩家可以自由決定是否安裝某個 Plugin。
 - 無論安裝或移除結果如何，核心大廳與遊戲行為保持正常。

Use Case PL3：已安裝 Plugin 的玩家在房間中使用 Plugin 功能

- **目標**

讓已安裝 Plugin 的玩家在遊戲房間中看到並使用額外功能，例如群組聊天視窗。
- **前置條件**
 - 玩家已安裝指定 Plugin (例如 Room Chat Plugin)。
 - 玩家已建立或加入一個遊戲房間。
- **步驟**
 1. 玩家進入某個遊戲房間。
 2. 系統偵測到該玩家有安裝聊天 Plugin：
 - 在房間畫面中顯示額外的「聊天區塊」或 UI。
 3. 玩家可以在聊天區塊輸入訊息並送出。
 4. 房間中其他也安裝此 Plugin 的玩家，可以看到並回覆這些訊息。
- **預期結果**
 - 已安裝 Plugin 的玩家，可以在房間中正常使用群組聊天或其他擴充功能。
 - Plugin 的 UI 與功能不會干擾原本遊戲操作（開始遊戲、準備、離開房間等）。

Use Case PL4：未安裝 Plugin 的玩家仍可正常遊玩

- **目標**

確保未安裝 Plugin 的玩家**完全不會被拖累**：可以正常進出大廳與房間、正常遊玩遊戲，只是看不到或無法使用 Plugin 功能。

- **前置條件**

- 至少有一位玩家安裝了某個 Plugin。
- 有另一位玩家**沒有安裝該 Plugin**。
- 兩位玩家都進入同一個遊戲房間，並準備遊玩同一場遊戲。

- **步驟**

1. 玩家 A (有 Plugin) 與玩家 B (無 Plugin) 一起進入同一房間。
2. 對玩家 A 來說：
 - 房間畫面中會出現 Plugin 提供的額外 UI (例如群組聊天視窗)。
3. 對玩家 B 來說：
 - 房間畫面中**不會**因為 Plugin 而報錯或壞掉，可以照常看到原本的房間資訊與按鈕。
 - 可以正常遊玩該遊戲，只是看不到 Plugin 功能，或只看到簡單提示 (例如「此功能需要安裝 Plugin」)。
4. 遊戲從開始到結束整個流程正常完成。

- **預期結果**

- 未安裝 Plugin 的玩家不會在任何畫面遇到 crash / error。
- 有無安裝 Plugin 的玩家可以在同一房間中一起順利遊玩。
- 整體流程 (Lobby → 房間 → 遊戲 → 結束) 在 Plugin 存在或缺席的情況下都能正常運作。

評分機制

配分說明

本作業總分：100 分，另外有最多 20 分加分項目。

1. 系統功能完整度 (Use Case，共 55 分)

依照下列 Use Case 完成度給分：

Use Case	描述	分數
----------	----	----

D1	開發者上架一款新遊戲	10 分
D2	開發者更新已上架遊戲版本	10 分
D3	開發者下架一款遊戲	5 分
P1	玩家瀏覽遊戲商城與詳細資訊	5 分
P2	玩家下載並更新遊戲版本	10 分
P3	玩家建立房間並啟動遊戲	10 分
P4	玩家對遊戲進行評分與留言	5 分

2. 遊戲實作程度 (15 分，闖關制)

依照實際完成的遊戲型態與驗收結果給分，採「闖關制」，每一關 5 分，最高 15 分：

- **關卡 A：雙人 CLI 遊戲 (5 分)**
 - 至少支援 2 位玩家對戰
 - 可以從「開始遊戲 → 進行對戰 → 結束並看到結果」完整跑完一局
 - 過程中不會在一般情況下當機或卡死
- **關卡 B：GUI 介面 (+5 分)**
 - 有基本圖形介面（視窗、按鈕或畫面顯示，不限框架）
 - 雙人對戰的主要操作可以透過 GUI 完成（不需要再打指令）
 - 遊戲狀態會在 GUI 上正確更新（例如棋盤、分數、回合結果等）
- **關卡 C：多人 GUI 同局遊玩 (+5 分)**
 - 同一場遊戲中，實際能有 **3 位（含）以上** 玩家一起遊玩
 - 多位玩家的連線與狀態更新邏輯正確（例如輪流／同步機制合理）
 - Demo 時，助教可以實際操作至少 3 個玩家帳號跑完一局

若某關卡僅部分達成（例如 GUI 有但無法完整對戰），該關卡分數可視實際情況給 0 ~ 3 分。

3. 系統架構與正確性 (5 分)

- 三種角色與模組切分是否清楚 (Server / Developer / Player)
- 版本管理與檔案來源區分是否合理 (Developer 端 / Server 上架版本 / Player 下載版本)

- 資料在 Server 重啟後是否仍能正確維持

4. 使用者體驗與介面設計（5分）

- Menu 結構是否清楚、層級合理（不會塞爆在同一層）
- Demo 流程是否直覺，助教大致依照 README 就能自己從「上架 → 遊玩」跑完
- 錯誤訊息是否清楚（輸入錯誤、連線失敗、版本衝突等不會只顯示 error code）

5. 程式碼品質與文件（5分）

- 程式碼結構與命名是否有基本可讀性，不是完全 Spaghetti
- README 是否足夠讓助教在新環境啟動 Developer / Lobby Client 並完成基本操作

6. 口頭問答與系統理解（15分）

助教會在 Demo 現場出一些題目，請同學現場說明或實作，例如：

- 說明你們的**系統架構**與模組切分方式（Server / Developer / Player 如何溝通）
 - 如果要新增某個功能（例如新的遊戲類型、額外的資料欄位），你會怎麼改？
 - 某些 Bug 或極端情況（例如 Lobby 掛掉、某個 game server 卡住）在你們的架構下要怎麼處理？
- 也有可能會請你**現場修改程式碼或設定**，所以請務必了解自己的程式架構，並在開發時保留一定的擴充性。

加分項目：Plugin 功能（最多 20 分）

- 實作一個可供玩家下載的 **Plugin** 系統（不需要透過 Developer 上架流程），詳細請參考 Usecase PL1~4。
- 核心要求：
 - **不論玩家有沒有安裝 Plugin，都不應破壞整體流程**（大廳、房間、遊戲都能正常使用）。
 - 沒有下載 Plugin 的玩家，系統應有合理預設行為（例如單純看不到某些功能），而不是當機。
 - 有下載 Plugin 的玩家，可以體驗到**額外功能**，例如：
 - 房間內群組聊天

- 額外的遊戲資訊顯示
- 額外的快捷操作等等（由各組自行設計）



不一定要實作，範例的群組聊天，可以是額外的遊戲設定、UI主題等等，Plugin 內容不限制

TA 的話 - 關於 AI

• 關於 AI 輔助工具

本次作業非常歡迎你們使用 AI 輔助你們進行作業，甚至我們可能會現場請你輸入 prompt 來去修改你的程式碼，因此你更應該了解整個專案的架構與實作方式，多考量垂直與橫向擴充。為了避免你們什麼都沒練習到就教了一個作業，如果無法順利回答助教現場要求的問題，那就算你有完成對應的 UseCase 我們也不予給分。

• 關於加分項目

原則上是給做完的同學進行一些小挑戰，來檢驗你專案實作的精熟度，並不是代表你可以跳過助教口試，或是跳過原本的 UseCase，因此如果你沒有完成以下幾個 UseCase，代表你沒有完成這份作業的核心需求，則加分項目也不予計分



這幾個 UseCase 就基本上就是本次作業最基本的 Flow

- D1：開發者上架一款新遊戲 **(10 分)**
- D2：開發者更新已上架遊戲版本 **(10 分)**
- P2：玩家下載並更新遊戲版本 **(10 分)**
- P3：玩家建立房間並啟動遊戲 **(10 分)**

• 小碎嘴

希望大家是能帶東西走的，雖然 HW1 2 作業已經讓你們拿了足夠通過課程的分數，但還是希望大家能夠認真撰寫此次作業，不要想辦法去湊分數，這樣是對你們自己的學習負責，也不會辜負我們花時間努力出的作業。

作業問題相關詢問

沒有固定 TA 時間，可以利用禮拜一的時間去做詢問。如要請寄信到
quan787887@gmail.com，郵件範例格式如下（為按照格式寄信不會去回信）

| NP_HW3問題詢問_<學號>_<姓名>