Network Programming - HW1 Two-Player Online Game Report

1. System Architecture （Lobby、P2P、DB）

Lobby Server（TCP）：

負責註冊/登入、線上名單與回報（REGISTER / LOGIN / REPORT / PLAYERS / LOGOUT）。與玩家之間採 newline-delimited JSON 傳輸(用換行符 `\n` 分隔)。內部連到 SQLite 寫入 users 與 user_state（帳密、login_count、xp、coins、online、last_seen）

DataBase

用SQLite提供帳密驗證、登入/登出狀態更新、心跳落盤與整理線上列表（超時則離線）
主要有：
`users(username, password_hash, salt, created_at)` 用於帳密驗證；
`user_state(username, login_count, online, last_seen)` 用於玩家狀態/心跳。
提供註冊、驗證、登入/登出標記、心跳寫回與線上名單整理（會把太久未心跳者標記為離線）
詳細說明如下：

- **username (PRIMARY KEY)：** 使用者識別鍵；其餘狀態都以它為索引。

- **login_count (INTEGER, default 0)：** 成功登入一次就 +1。登入成功回覆 LOGIN_SUCCESS 時，會把這個值放在 profile.login_count 回給客戶端。

- **online (INTEGER, default 0)：**
  mark_login() 或 report() 會把它設為 1；mark_logout() 會設為 0。
  另外在 list_online(stale_sec=30) 會做過期檢測：若 last_seen 距今超過 30 秒，就把該玩家 online 設回 0（避免沒正常登出卻斷線的人一直占在線名單）。
  主要用途：

  1. 快速判斷某使用者是否在線（is_online() 用它）
  2. 防止重複登入（登入時先檢查 online=1 就回 LOGIN_DUPLICATE）
  3. 產出 PLAYERS 線上名單。

- **last_seen (TEXT ISO8601, default '')**
  每次 mark_login()、report()、mark_logout() 都會更新成當下 UTC 時間字串。
  主要用途：

1. 作為 list_online() 的過期判斷基準（>30 秒視為離線並清旗標），
2. 在 PLAYERS 清單裡顯示玩家「最後一次動作」時間，方便 UI 呈現「剛剛 / 幾秒前 / 幾分鐘前」。

P2P

先用 UDP 做「發現與邀請」：DISCOVER / DISCOVER_ACK / INVITE / INVITE_REPLY / GAME_TCP；
受邀一方同意後，由主機方用 TCP 開一個臨時（或指定）埠，透過 UDP 把 GAME_TCP{host,port} 告訴對方，再建立 TCP 對戰連線。

2. Communication Details

序列化與訊框分隔

- **傳輸格式**：所有 TCP 訊息皆為 **JSON**
- **封包邊界**：使用 **newline-delimited JSON**（每個 JSON 後面接 `\n`）

> 範例
> `{"type":"LOGIN","payload":{"username":"Alice","password":"***"}}\n`

玩家 ↔ Lobby（TCP）

請求／回應一覽

方向	訊息	說明	回應
Client → Lobby	`REGISTER{username,password}`	註冊帳號	`REGISTER_OK` / `REGISTER_TAKEN`
Client → Lobby	`LOGIN{username,password}`	登入、標記上線	`LOGIN_SUCCESS{profile{login_count,xp,coins}}` / `LOGIN_FAIL` / `LOGIN_DUPLICATE`
Client → Lobby	`REPORT{username,stats{xp,coins}}`	心跳與累積資料回報（≈10s 一次）	`REPORT_OK`
Client → Lobby	`PLAYERS{}`	查線上名單（回傳前會清除逾時心跳者）	`PLAYERS_OK{online:[{username,last_seen,xp,coins}]}`
Client →	`LOGOUT{username}`	登出、標記離線	`LOGOUT_OK`

Lobby			

JSON 範例

```
{"type":"LOGIN","payload":{"username":"Alice","password":"p@ss"}}
{"type":"LOGIN_SUCCESS","payload":{"profile":{"login_count":12,"xp":30,"coins":5

{"type":"REPORT","payload":{"username":"Alice","stats":{"xp":31,"coins":6}}}
{"type":"REPORT_OK"}

{"type":"PLAYERS"}
{"type":"PLAYERS_OK","payload":{"online":[
  {"username":"Alice","last_seen":"2025-10-12T04:12:00Z","xp":31,"coins":6},
  {"username":"Bob","last_seen":"2025-10-12T04:11:45Z","xp":12,"coins":1}
]}}

{"type":"LOGOUT","payload":{"username":"Alice"}}
{"type":"LOGOUT_OK"}
```

玩家 ↔ 玩家（UDP + TCP）

A. 發現／邀請（UDP）

- `DISCOVER` → `DISCOVER_ACK{player,udp_port}`
- `INVITE{from}` → `INVITE_REPLY{accept:true|false}`

B. 交換 TCP 端點（UDP）

- 主機建立遊戲 TCP 監聽後，用 `GAME_TCP{host,port}` 告知對手

C. 對戰（TCP；newline-delimited JSON）

- 啟始：`WELCOME{mark:'X'|'O', first:'X'|'O', rule:'recycle-3'}`
- 回合：`YOUR_TURN` →（對手出手）`MOVE{pos}` →（主機判定&同步）`STATE{board,turn,last,recycled}`
- 結束：`GAME_OVER{winner:'X'|'O'}`
- 例外：`ERROR{reason}`

JSON 範例

```
{"type":"WELCOME","payload":{"mark":"X","first":"X","rule":"recycle-3"}}
{"type":"YOUR_TURN"}
{"type":"MOVE","payload":{"pos":6}}
{"type":"STATE","payload":{
  "board":"X O   X   O",
  "turn":"O",
  "last":{"player":"X","pos":6},
  "recycled":{"player":"X","pos":0}
```

```
}}
{"type":"GAME_OVER","payload":{"winner":"O"}}
```

3. Game Play（規則、流程、結束）

3.1 規則（Recycling Tic-Tac-Toe 3×3）

- 棋盤 3×3，標記 `'X'` 與 `'O'`，**X 先手**
- **回收規則**：同一玩家在盤面上**最多 3 枚**；當該玩家下第 4 手（或之後）時：
  1. 先把新子放上去
  2. **自動移除**該玩家**最早放的那一枚**（FIFO）
- **勝利**：每手落子與回收完成後檢查三連線（直／橫／斜），達成即勝

3.2 回合流程

1. 主機送 `WELCOME`（指派 `mark` 與 `first`）
2. 輪到某方 → 主機送 `YOUR_TURN`
3. 該方回 `MOVE{pos}`（`pos` ∈ 0..8 且位置為空）
4. 主機更新盤面、套用回收（若需要）、判勝，並廣播 `STATE{...}`
5. 若有人勝出 → 廣播 `GAME_OVER{winner}`，關閉連線；否則換手

3.3 錯誤情況（示例）

- 非當前玩家送出 `MOVE` → `ERROR{"reason":"not_your_turn"}`
- 非法位置／位置非空 → `ERROR{"reason":"invalid_move"}`
- 連線中斷 → 由主機決策判負或結束（依課規／實作策略）

常用指令

0) 伺服器端啟動 Lobby

```
python3 lobby_server.py --host 0.0.0.0 --port 17000 --db lobby.sqlite --verbose
```

開兩個終端機（或兩台機器）：

1) A 端：註冊 + 登入 + 等待邀請（UDP）

```
python3 player.py --username Alice --lobby linux1.cs.nycu.edu.tw:17000 --password 123 register
```

```
python3 player.py --username Alice --lobby linux1.cs.nycu.edu.tw:17000 --password 123 wait --udp-port 18001 --auto-accept
```

2) B 端：註冊 + 登入 + 邀請 A（透過 UDP）

```
python3 player.py --username Bob --lobby linux1.cs.nycu.edu.tw:17000 --password 123 register
```

```
python3 player.py --username Bob --lobby linux1.cs.nycu.edu.tw:17000 --password 123 invite --target linux1.cs.nycu.edu.tw:18001 --tcp-bind-host 0.0.0.0 --tcp-port 19001
```

B：登入後掃描/邀請

```
python3 player.py --username Bob --lobby linux1.cs.nycu.edu.tw:17000 --password 123 scan --hosts linux1.cs.nycu.edu.tw,linux2.cs.nycu.edu.tw --ports 18001-18005 --timeout 0.2
```

```
python3 player.py --username Bob --lobby linux1.cs.nycu.edu.tw:17000 --password 123 invite --target linux1.cs.nycu.edu.tw:18001 --tcp-bind-host 0.0.0.0 --tcp-port 19001
```

P2P 對戰流程 B：自動配對連續開打

```
python3 player.py --username Bob --lobby linux1.cs.nycu.edu.tw:17000 --password 123 match --hosts linux1.cs.nycu.edu.tw,linux2.cs.nycu.edu.tw,linux3.cs.nycu.edu.tw,linux4.cs.nycu.edu.tw --ports 18001-18005 --timeout 0.2 --tcp-bind-host 0.0.0.0 --tcp-port 19001
```