網路程式設計 - 作業二報告

Tetris 多人對戰遊戲系統

學號: 112550134

姓名: 賴儁樞

日期: 2025 年 11 月 5 日

系統架構

本專案採用 **三層式 Client-Server 架構**，分為以下三個主要服務：

1. Database Server (Port 12000)

- **功能**: 統一管理所有資料庫操作
- **實作**: SQLite + Socket API 包裝
- **職責**:
    - 用戶註冊、登入驗證
    - 房間資料管理
    - 遊戲歷史紀錄
    - 邀請系統

2. Lobby Server (Port 13000)

- **功能**: 大廳管理與遊戲房間協調
- **實作**: TCP Server + 動態 Game Server 生成
- **職責**:

- 用戶連線管理
- 房間創建、加入、離開
- 玩家邀請系統
- 動態生成 Game Server 子進程
- 觀戰模式協調

3. Game Server (Port 10000-20000, 動態分配)

- **功能**: 遊戲邏輯處理與狀態同步
- **實作**: TCP Server + 權威式伺服器架構
- **職責**:
  - 遊戲狀態管理
  - 輸入處理與驗證
  - 定時廣播狀態快照 (100ms)
  - Shared 7-bag 方塊生成
  - 觀眾模式支援

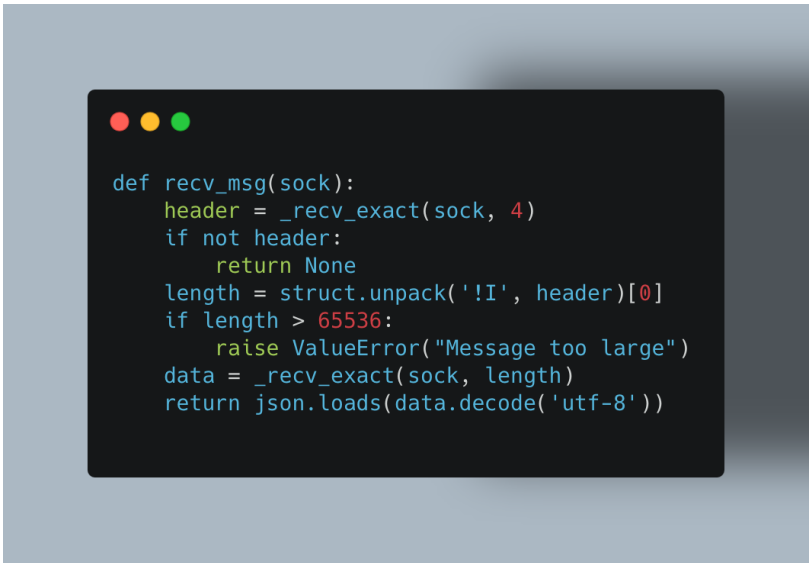
網路協定設計

通訊協定: Length-Prefixed Framing Protocol

發送訊息 (utils.py):

```python
def send_msg(sock, obj):
    data = json.dumps(obj, ensure_ascii=False).encode('utf-8')
    length = len(data)
    if length > 65536:  # 限制 64KB
        raise ValueError("Message too large")
    header = struct.pack('!I', length)  # Big-endian unsigned int
    sock.sendall(header + data)
```

接收訊息 (utils.py):



```python
def recv_msg(sock):
    header = _recv_exact(sock, 4)
    if not header:
        return None
    length = struct.unpack('!I', header)[0]
    if length > 65536:
        raise ValueError("Message too large")
    data = _recv_exact(sock, length)
    return json.loads(data.decode('utf-8'))
```

封包格式定義

1. 玩家連線到 Game Server

Client → Game Server: HELLO

```json
{
  "type": "HELLO",
  "version": 1,
  "roomId": 123,
  "userId": 17,
  "roomToken": "abc123"
}
```

欄位說明:

- type: 訊息類型，固定為 "HELLO"
- version: 協定版本號，目前為 1
- roomId: 房間 ID，由 Lobby Server 分配
- userId: 用戶 ID，來自資料庫
- roomToken: 房間驗證 token，防止未授權加入

Game Server → Client: WELCOME

```
{
  "type": "WELCOME",
  "role": "P1",
  "seed": 987654321,
  "bagRule": "7bag",
  "gravityPlan": {
    "mode": "fixed",
    "dropMs": 500
  }
}
```

欄位說明:

- type: 訊息類型，固定為 "WELCOME"
- role: 玩家角色，"P1" 或 "P2" 或 "SPECTATOR"
- seed: 隨機種子，用於初始化 7-bag 演算法
- bagRule: 方塊生成規則，固定為 "7bag"
- gravityPlan: 重力設定
    - mode: 重力模式，"fixed" 表示固定速度
    - dropMs: 方塊自動下落間隔（毫秒）

---

2. 遊戲進行中的通訊

Client → Game Server: INPUT

```
{
  "type": "INPUT",
  "userId": 17,
  "seq": 102,
  "ts": 1234567890,
  "action": "CW"
}
```

欄位說明:

- type: 訊息類型，固定為 "INPUT"
- userId: 用戶 ID
- seq: 輸入序列號，用於去重與順序保證
- ts: 時間戳記（毫秒），客戶端發送時間
- action: 操作指令
    - "CW": 順時針旋轉 / "CCW": 逆時針旋轉
    - "LEFT": 左移 / "RIGHT": 右移
    - "SOFT_DROP": 軟降
    - "HARD_DROP": 硬降
    - "HOLD": 暫存方塊

設計理念:

- Client 只發送輸入，不處理邏輯
- Server 驗證輸入合法性後更新遊戲狀態
- 避免客戶端作弊或狀態不同步

Game Server → Clients: SNAPSHOT

欄位說明:

- type: 訊息類型，固定為 "SNAPSHOT"
- tick: 遊戲刻度（tick），每次廣播遞增
- userId: 此快照對應的玩家 ID
- boardRLE: 遊戲面板壓縮字串
    - 格式: "cell_value:count,cell_value:count,..."
    - 例如: "0:200" 表示 200 個空格
    - 減少網路傳輸量 (200×10 格 → 壓縮為短字串)
- active: 當前活動方塊
    - shape: 方塊形狀 ("I", "O", "T", "S", "Z", "L", "J")
    - x, y: 方塊位置（左上角座標）
    - rot: 旋轉狀態 (0, 1, 2, 3)
- hold: 暫存方塊，null 表示無暫存
- next: 下一個方塊序列（預覽用）
- score: 當前分數
- lines: 已消除行數
- level: 當前等級
- at: 快照生成時間（毫秒）

廣播頻率: 每 100ms 廣播一次 (定義於 game_server.py 的 SNAPSHOT_INTERVAL = 0.10)

廣播範圍:

- 每個玩家收到自己和對手的 SNAPSHOT
- 觀眾收到所有玩家的 SNAPSHOT

---

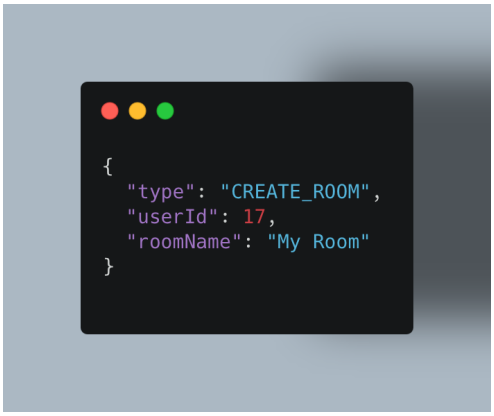
3. Lobby Server 通訊協定

Client → Lobby Server

註冊新用戶:

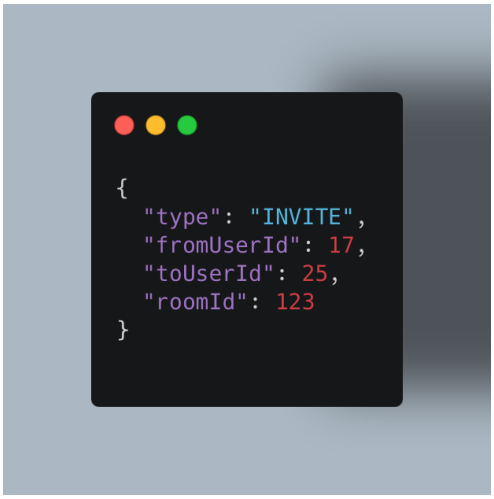
登入:

創建房間:



```
{
  "type": "CREATE_ROOM",
  "userId": 17,
  "roomName": "My Room"
}
```

邀請玩家:



```
{
  "type": "INVITE",
  "fromUserId": 17,
  "toUserId": 25,
  "roomId": 123
}
```

接受邀請:

```
{
  "type": "ACCEPT_INVITE",
  "userId": 25,
  "invitationId": 456
}
```

開始遊戲:

觀戰模式:

Lobby Server → Client

遊戲伺服器資訊:

房間狀態推送:

```
{
  "type": "PUSH_ROOM_INFO",
  "room": {
    "roomId": 123,
    "roomName": "My Room",
    "status": "PLAYING",
    "owner": 17,
    "players": [17, 25]
  }
}
```

邀請通知:

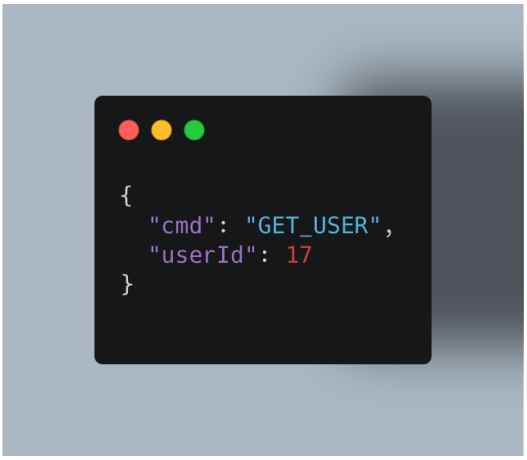
```
{
  "type": "PUSH_INVITATIONS",
  "invitations": [
    {
      "invitationId": 456,
      "fromUserId": 17,
      "fromUsername": "player123",
      "roomId": 123,
      "roomName": "My Room"
    }
  ]
}
```

---

4. Database Server Socket API

Client → DB Server

查詢用戶資訊:



```
{
  "cmd": "GET_USER",
  "userId": 17
}
```

新增遊戲紀錄:

```
{
  "cmd": "ADD_GAMELOG",
  "roomId": 123,
  "winner": 17,
  "player1": 17,
  "player2": 25,
  "player1Score": 15600,
  "player2Score": 12400,
  "player1Lines": 14,
  "player2Lines": 9
}
```

DB Server → Client

查詢結果:

```
{
  "ok": true,
  "data": {
    "userId": 17,
    "username": "player123",
    "totalGames": 42,
    "wins": 25,
    "losses": 17
  }
}
```

錯誤回應:

```
{
  "ok": false,
  "error": "User not found"
}
```

遊戲玩法設計

基本規則

本遊戲為 **Tetris 多人對戰系統**，遵循標準 Tetris Guideline 規則：

遊戲面板

- **尺寸**: 10 寬 × 20 高
- **方塊種類**: 7 種標準 Tetromino (I, O, T, S, Z, J, L)
- **方塊生成**: Shared 7-bag 演算法（確保公平性）

操作方式

操作	按鍵	說明
左移	←/A	方塊向左移動一格
右移	→/D	方塊向右移動一格
軟降	↓/S	方塊加速下落
硬降	Space	方塊瞬間落地並鎖定
順時針旋轉	↑/W	順時針旋轉 90°
逆時針旋轉	Z	逆時針旋轉 90°
暫存	C/Shift	將當前方塊存入 Hold 欄位

旋轉系統

實作 Wall Kick 機制：

- 當旋轉位置被阻擋時，嘗試偏移 (-1, 0, +1, -2, +2) 格
- 找到第一個可行位置後執行旋轉
- 若所有位置都失敗，則取消旋轉

實作 (game_server.py 第 76-80 行):

```
def rotate_kick(board, state):
    new = (state['rot'] + 1) % 4
    for dx in [0, -1, 1, -2, 2]:
        if not collide(board, state['shape'], new, state['x'] + dx, state['y']):
            state['rot'] = new; state['x'] += dx; return
```

計分方式

消行計分

採用標準 Tetris 計分規則：

消除行數	得分	名稱
1 行	100	Single
2 行	300	Double
3 行	500	Triple
4 行	800	Tetris

實作 (game_server.py 第 21 行):

SCORES = {0:0, 1:100, 2:300, 3:500, 4:800}

計分邏輯

當方塊鎖定 (Hard Drop 或自然落地) 時:

1. 檢查所有完整行
2. 移除完整行，上方方塊下落
3. 根據消除行數加分
4. 累計總消行數

實作 (game_server.py 第 82-87 行):

```
def hard_drop(state):
    # ... 方塊落地邏輯 ...
    state['board'], cleared = lock_piece(state['board'], ...)
    state['score'] += SCORES.get(cleared, 0)  # 加分
    state['lines'] += cleared  # 累計消行
```

重力系統

自動下落

- **初始速度**: 1000ms (每秒下落 1 格)
- **實作方式**: Game Server 在 game_loop 中定時檢查
- **加速機制**: 可透過 TEMPO 訊息調整速度

實作 (game_server.py 第 18 行):

GRAVITY_MS = 1000 # 初始重力間隔

Lock Delay

- **延遲時間**: 500ms
- **目的**: 給予玩家最後調整方塊位置的機會
- **觸發條件**: 方塊接觸地面或其他方塊

實作 (game_server.py 第 19 行):

LOCK_DELAY_MS = 500 # 鎖定延遲

結束條件

Game Over 判定

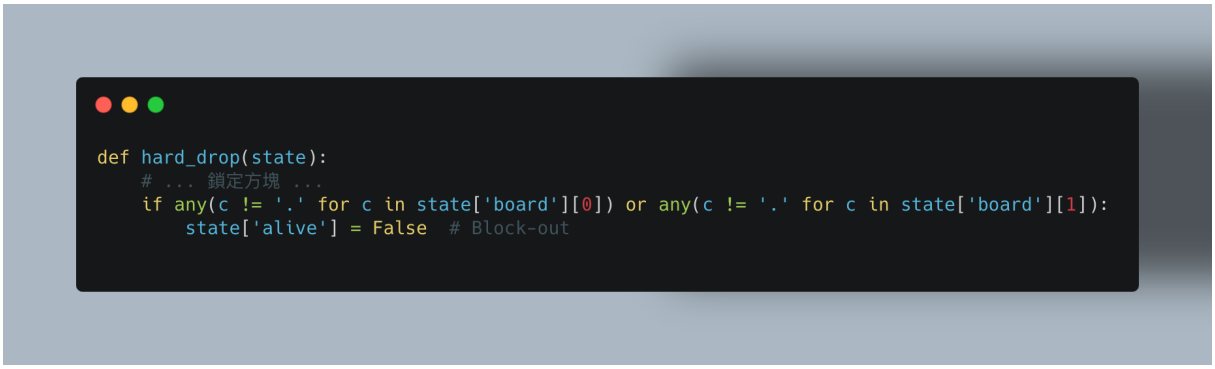
當以下任一情況發生時，玩家 Game Over：

1. **Top-out (頂部溢出)**
    - 新方塊生成時，位置已被佔據
    - 檢查時機：每次 spawn() 生成新方塊時
2. **Block-out (方塊鎖定在頂部)**
    - 方塊鎖定後，頂部 2 行 (隱藏區) 內有方塊
    - 檢查時機：Hard Drop 或自然鎖定後

實作 (game_server.py 第 69-75 行):

```python
def spawn(state, check_topout=True):
    # ... 生成方塊 ...
    if check_topout:
        for (cx, cy) in cells(state['shape'], state['rot'], state['x'], state['y']):
            if 0 <= cy < BOARD_H and 0 <= cx < BOARD_W:
                if state['board'][cy][cx] != '.':
                    state['alive'] = False  # Top-out
                    return
```

實作 (game_server.py 第 88-91 行):



```python
def hard_drop(state):
    # ... 鎖定方塊 ...
    if any(c != '.' for c in state['board'][0]) or any(c != '.' for c in state['board'][1]):
        state['alive'] = False  # Block-out
```

勝負判定

對戰模式:

- 勝利條件: 對手 Game Over 時，存活玩家獲勝
- 失敗條件: 自己 Game Over
- 平局: 雙方同時 Game Over（少見）

遊戲結束流程:

1. Game Server 偵測到玩家 alive = False
2. 廣播 GAME_OVER 訊息給所有客戶端

3. 統計雙方分數、消行數
4. 呼叫 Lobby Server，寫入遊戲紀錄到資料庫
5. 玩家返回房間，可選擇再次開始

實作 (game_server.py 第 195-214 行):

遊戲模式

本系統支援兩種遊戲模式:

1. Timed Mode (計時模式)

- **時間限制**: 固定時長
- **勝利條件**: 時間到時，分數較高者獲勝

2. Survival Mode (生存模式)

- **無時間限制**: 持續對戰直到分出勝負
- **勝利條件**: 對手 Game Over

實作 (game_server.py 第 11 行):

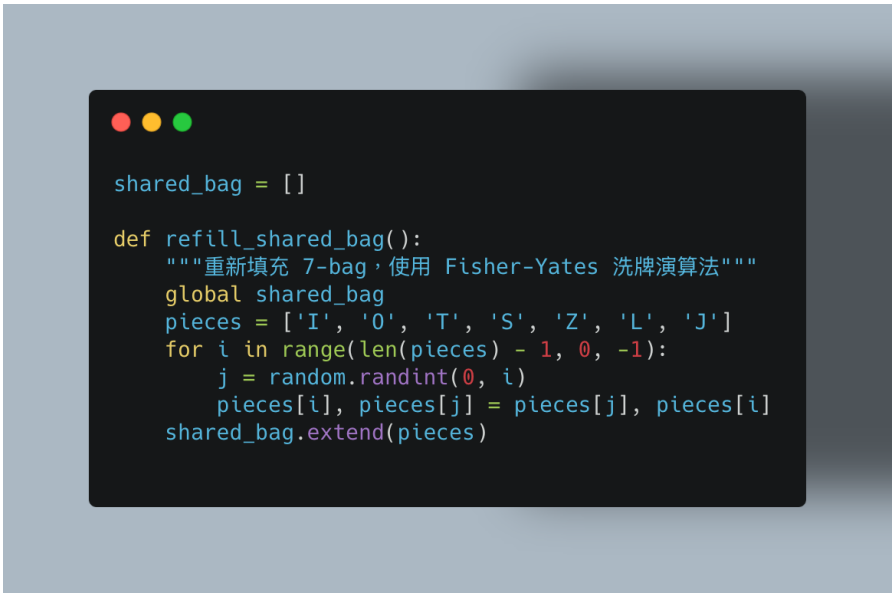
```
MODE = sys.argv[3]  # "timed" or "survival"
DURATION = int(sys.argv[4]) # 遊戲時長 (秒)
```

遊戲邏輯實作

1. Shared 7-bag 演算法

設計目標: 確保兩位玩家拿到完全相同的方塊序列

實作 (game_server.py):



```python
shared_bag = []

def refill_shared_bag():
    """重新填充 7-bag，使用 Fisher-Yates 洗牌演算法"""
    global shared_bag
    pieces = ['I', 'O', 'T', 'S', 'Z', 'L', 'J']
    for i in range(len(pieces) - 1, 0, -1):
        j = random.randint(0, i)
        pieces[i], pieces[j] = pieces[j], pieces[i]
    shared_bag.extend(pieces)
```

```python
def spawn(board):
    """從 shared bag 中取出方塊生成"""
    global shared_bag
    if not shared_bag:
        refill_shared_bag()
    shape = shared_bag.pop(0)  # FIFO
    # ... 生成新方塊
```

流程:

1. 遊戲開始時，使用固定 seed 初始化 random
2. 第一次呼叫 spawn() 時，shared_bag 為空，觸發 refill_shared_bag()
3. Fisher-Yates 洗牌產生隨機順序的 7 個方塊
4. 兩位玩家依序從同一個 shared_bag 取方塊
5. 當 bag 空了，再次重新填充

保證公平性: 兩位玩家在相同時間點會拿到相同方塊

---

2. 輸入處理與狀態同步

處理流程 (game_server.py):

```
def handle_input(userId, action):
    """處理玩家輸入"""
    with state_lock:
        board = boards[userId]

        if action == "CW":
            rotate_clockwise(board)
        elif action == "LEFT":
            move_left(board)
        elif action == "HARD_DROP":
            hard_drop(board)
            if not spawn(board):
                end_game(userId)  # Game Over
        # ... 其他動作
```

```python
def game_loop():
    """主遊戲循環，每 100ms 廣播一次"""
    while not stop_event.is_set():
        time.sleep(SNAPSHOT_INTERVAL)  # 0.1 秒

        with state_lock:
            # 處理重力（自動下落）
            for userId in boards:
                if should_drop(userId):
                    move_down(boards[userId])

            # 廣播狀態快照給所有連線
            for sock in all_sockets:
                for userId in boards:
                    snapshot = build_snapshot(userId)
                    send_msg(sock, snapshot)
```

關鍵設計:

- Client 只發送輸入事件，不計算遊戲邏輯
- Server 驗證並執行所有動作
- 定時廣播狀態快照給所有客戶端
- 使用 鎖 (state_lock) 保證多執行緒安全

---

3. 觀戰模式實作

實作細節 (game_server.py):

```python
def handle_client(conn, addr):
    """處理玩家或觀眾連線"""
    msg = recv_msg(conn)
    if msg['type'] != 'HELLO':
        return

    role = "SPECTATOR" if is_spectator(msg['roomToken']) else "P1 or P2"

    if role == "SPECTATOR":
        spectators.append(conn)
        send_msg(conn, {"type": "WELCOME", "role": "SPECTATOR"})
        # 觀眾不參與遊戲邏輯，只接收快照
    else:
        # 正常玩家，初始化遊戲狀態
        boards[msg['userId']] = create_board()
```

```python
def game_loop():
    """廣播給玩家和觀眾"""
    while not stop_event.is_set():
        time.sleep(0.1)

        # 建立所有玩家的快照
        snapshots = [build_snapshot(uid) for uid in boards]

        # 發送給玩家
        for sock in player_sockets:
            for snapshot in snapshots:
                send_msg(sock, snapshot)

        # 同樣發送給觀眾
        for sock in spectators:
            for snapshot in snapshots:
                send_msg(sock, snapshot)
```

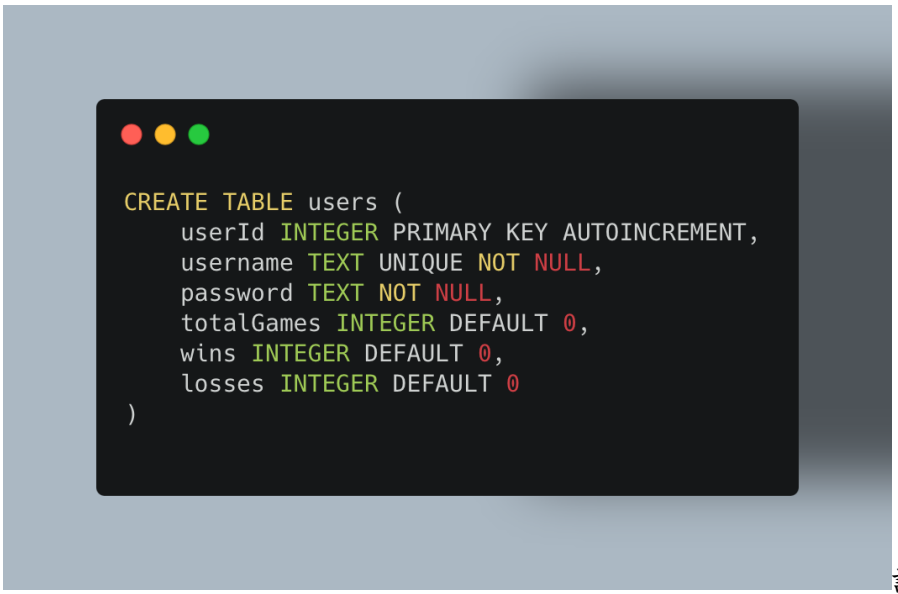
觀眾特性:

- 只讀模式: 不接收 INPUT，只接收 SNAPSHOT
- 同步即時: 與玩家同步接收狀態更新
- 可重複觀看: 離開後可再次加入同一場遊戲

資料庫設計

Schema 定義 (db_server.py)

1. users 表

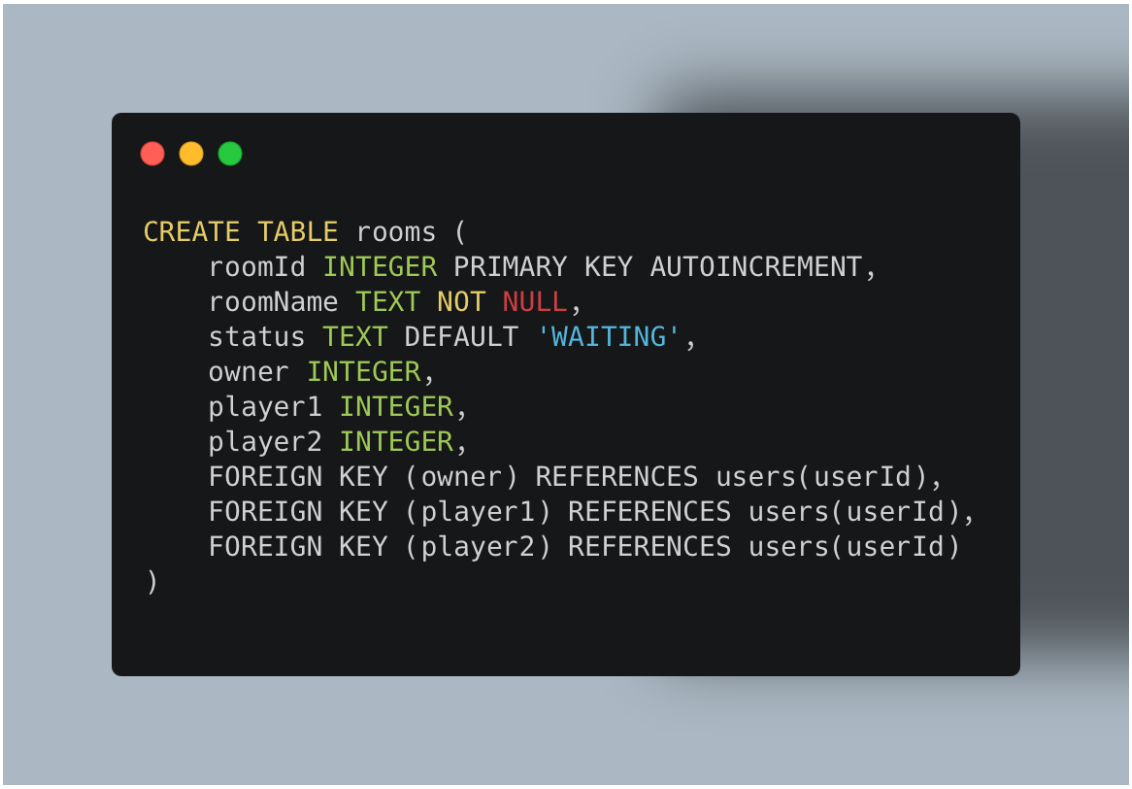


說明:

- userId: 主鍵，自動遞增
- username: 用戶名稱，唯一索引
- password: 密碼（明文儲存，實際應用應 hash）
- totalGames, wins, losses: 統計資料

---

2. rooms 表



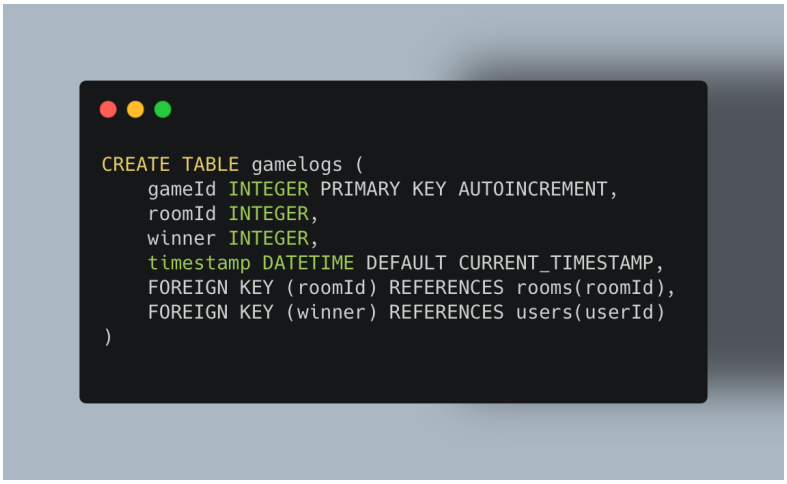
```
CREATE TABLE rooms (
    roomId INTEGER PRIMARY KEY AUTOINCREMENT,
    roomName TEXT NOT NULL,
    status TEXT DEFAULT 'WAITING',
    owner INTEGER,
    player1 INTEGER,
    player2 INTEGER,
    FOREIGN KEY (owner) REFERENCES users(userId),
    FOREIGN KEY (player1) REFERENCES users(userId),
    FOREIGN KEY (player2) REFERENCES users(userId)
)
```

說明:

- roomId: 房間 ID
- status: 房間狀態
    - "WAITING": 等待中
    - "PLAYING": 遊戲中
    - "FINISHED": 已結束
- owner: 房主 userId
- player1, player2: 玩家 userId

---

3. gamelogs 表



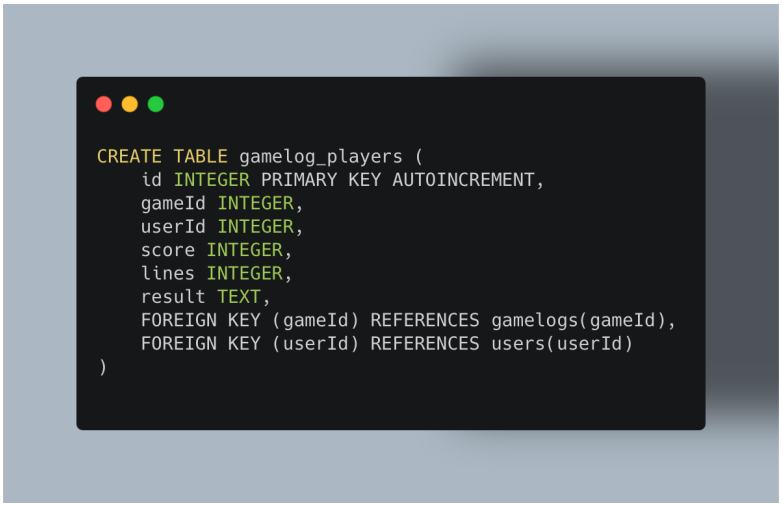
```sql
CREATE TABLE gamelogs (
    gameId INTEGER PRIMARY KEY AUTOINCREMENT,
    roomId INTEGER,
    winner INTEGER,
    timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (roomId) REFERENCES rooms(roomId),
    FOREIGN KEY (winner) REFERENCES users(userId)
)
```

說明:

- gameId: 遊戲記錄 ID
- roomId: 對應房間
- winner: 勝利者 userId
- timestamp: 遊戲結束時間

---

4. gamelog_players 表



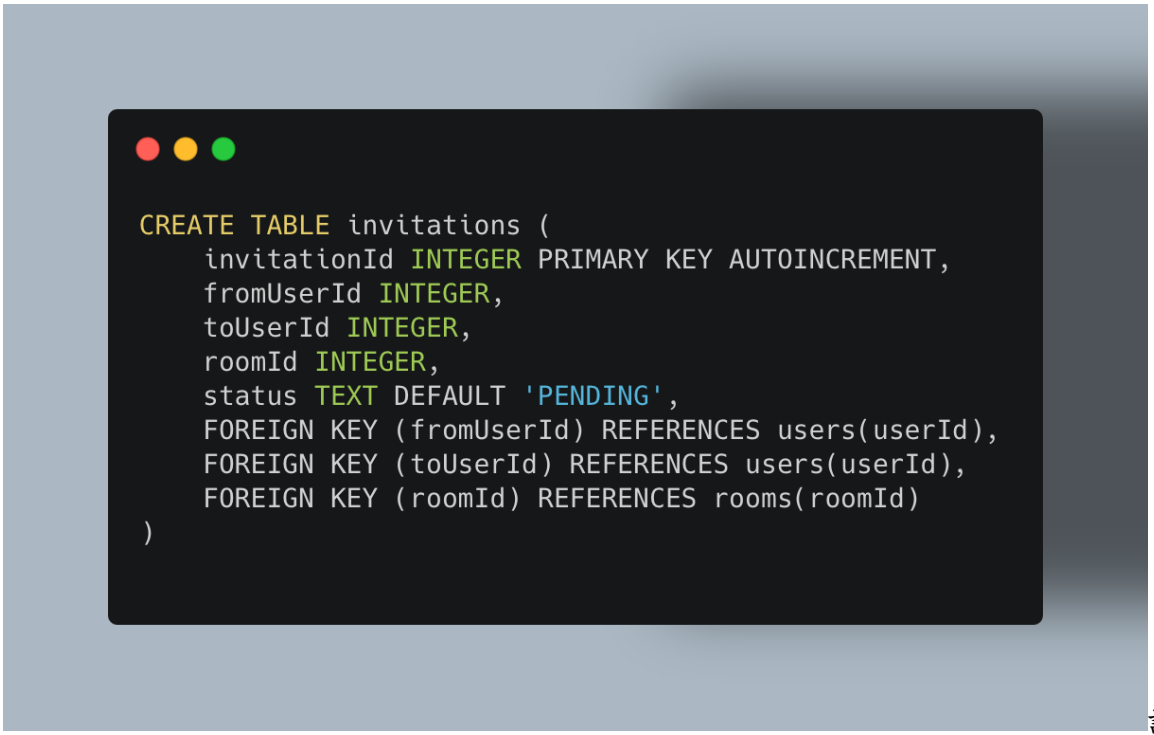
```sql
CREATE TABLE gamelog_players (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    gameId INTEGER,
    userId INTEGER,
    score INTEGER,
    lines INTEGER,
    result TEXT,
    FOREIGN KEY (gameId) REFERENCES gamelogs(gameId),
    FOREIGN KEY (userId) REFERENCES users(userId)
)
```

說明:

- 儲存每位玩家在該局的詳細資料
- result: "WIN" 或 "LOSE"
- 一場遊戲有兩筆紀錄（雙方玩家）

---

5. invitations 表



```
CREATE TABLE invitations (
    invitationId INTEGER PRIMARY KEY AUTOINCREMENT,
    fromUserId INTEGER,
    toUserId INTEGER,
    roomId INTEGER,
    status TEXT DEFAULT 'PENDING',
    FOREIGN KEY (fromUserId) REFERENCES users(userId),
    FOREIGN KEY (toUserId) REFERENCES users(userId),
    FOREIGN KEY (roomId) REFERENCES rooms(roomId)
)
```

說明:

- status: "PENDING", "ACCEPTED", "DECLINED"
- 用於實作玩家邀請系統

Database Server Socket API 設計

優點:

- 統一資料庫存取介面
- 避免多進程同時寫入 SQLite 造成 lock
- 方便橫向擴展（可替換為 PostgreSQL 等）

API 範例:

```python
# Lobby Server 呼叫 DB Server
def db_call(cmd, **params):
    """向 DB Server 發送請求並等待回應"""
    req = {"cmd": cmd, **params}
    send_msg(db_sock, req)
    resp = recv_msg(db_sock)
    if not resp.get("ok"):
        raise Exception(resp.get("error"))
    return resp.get("data")

# 查詢用戶
user = db_call("GET_USER", userId=17)

# 新增遊戲記錄
db_call("ADD_GAMELOG", roomId=123, winner=17,
        player1=17, player2=25,
        player1Score=15600, player2Score=12400,
        player1Lines=14, player2Lines=9)
```

依賴套件

pygame>=2.5.0