# 義　守　大　學

# 資訊工程學系

# 專題研究報告

## 基於 Python Socket 之多人連線遊戲

專題學生：11303044A 潘致凱
11303032A 謝詠丞
11303035A 饒家豪
11303064A 陳永恩

指導教授：　高典良　　老師

中　華　民　國　一　一　四　年　十　二　月　九　日

# 摘要

本專題旨在實作一款基於TCP/IP 協定的多人連線文字版狼人殺遊戲。透過Python的網路編程，建立一個穩定的主從式（Client-Server）架構，讓多名玩家能透過網路即時互動。系統模擬了完整的桌遊邏輯，包含房間管理、角色分配、日夜交替、技能判定以及勝負結算。此編程重點展示多執行續處理併發連線能力，以及使用Socket傳輸JSON格式資料的通訊協定設計。

# 目錄

# 第一章、 簡介

## 前言

雲端運算不是一項全新的網路技術，它其實算是一種全新的網路應用概念。簡單的說，雲端運算就是將電腦運算和資料儲存的工作，都放到網路上處理，並以動態隨選的方式向使用者提供服務。使用者可以使用各種形式的終端裝置，透過網際網路來取得運算資源服務。所以，雲端運算裡面所指的『雲』就是網際網路；『端』就是泛指連接到網際網路的任何一種終端裝置。雲端運算的概念已經廣泛應用到當代各種網路相關的服務上，提供了 IaaS、 PaaS、SaaS 等類型的服務，使用者可以依照自己的需求，選用這三種雲端服務。

## 1.1 功能特色

1. **多房機制與房間管理：**伺服器支援同時開設多個獨立遊戲房間，玩家可使用 /create (含密碼保護) 或 /join 指令管理加入。
2. **角色與流程：**支援 4-12 人的角色配置，包含狼人、狼王、預言家、女巫、獵人、守衛、白癡等，並嚴格遵循日夜交替的遊戲流程。
3. **即時通訊：**支援存活玩家聊天、狼人專屬夜間頻道，以及 死亡玩家（鬼魂）專屬頻道，確保遊戲資訊隔離。
4. **並發處理與同步：**伺服器利用多執行緒處理各玩家連線，並使用 threading.Lock機制確保在多個玩家同時進行投票或操作時，遊戲數據不會產生競態條件（Race Condition）。
5. **時間控制：**遊戲核心流程設有倒數計時（Timeout），確保玩家未操作時，遊戲能夠自動推進。

## 1.2 系統架構

本系統採用標準的 主從式架構 (Client-Server Architecture)，並透過 Python 的 threading 模組進行並發處理。伺服器端將任務劃分為三個主要執行緒層級：主監聽迴圈、處理 I/O 的 client_thread，以及控制遊戲狀態的 start_werewolf_game 執行緒。

1. **伺服器端（Server.py）：**

   [1] **主執行緒：**負責Socket監聽工作。

   [2] **Client_theard：**處裡每位用富端連線、指令接收、狀態寫入。

[3] **Start_werewolf_game**：獨立遊戲邏輯執行緒，控制單個房間的日夜流程遊戲狀態機制。

2. **客戶端（Client.py）**：

[1] **Send_message**：負責讀取使用者輸入並發送

[2] **Recv_message**：負責及時接收並發送伺服器廣播。

## 1.3 協定設計

本專案使用 JSON 格式作為通訊協定，所有數據均採用 UTF-8 編碼，並以換行符號 \n 作為封包邊界，確保 TCP 數據流在收發時能被準確分割。

| 方向 | 說明 | 程式碼應用 |
|---|---|---|
| Client → Server | 登入請求：發送暱稱 | Client.py 登入迴圈 |
| Server → Client | 登入回應：驗證結果 | Server.py 登入檢查 |
| 雙向 | 訊息傳輸：聊天與遊戲指令 | 核心通訊程式 |

所有指令和聊天內容都包含在傳輸方向為雙向的 Message 欄位之中，伺服器端透過解析該欄位的字串前綴來區分指令與聊天。

### 1. JSON 通訊類型、功能與位元組分析

本系統定義了三種核心 type，雖然 JSON 結構輕量，但其位元組開銷（Byte Overhead）對即時性有影響。

| JSON Type | 功能名稱 | 核心欄位 | 平均位元組開銷 |
|---|---|---|---|
| 1 | 登入請求 | type, nickname | 約 50 ~ 100 bytes |
| 2 | 登入回應 | type, error (可選) | 約 30 ~ 150 bytes |
| 3 | 訊息傳輸 | type, nickname, message | 約 50 ~ 200 bytes |

### 2. JSON 封包內容與結構範例

所有類型 3 訊息都包含在 message 欄位中。伺服器透過解析此欄位的內容來執行動作：

[1] **登入請求**： {"type": 1, "nickname": "玩家A"}\n

[2] **遊戲指令**： {"type": 3, "nickname": "狼人", "message": "殺 "}\n

[3] **系統廣播**： {"type": 3, "nickname": "系統", "message": "天黑請閉眼，狼人開始行動"}\n

# 第二章、程式碼邏輯分析

## 2.1 客戶端（Client.py）邏輯分析

Client.py 專注於 I/O 處理、連線穩定與使用者體驗。

**1. 初始化與登入**

[1] **連線建立：**

設定 TCP Socket（socket.AF_INET, socket.SOCK_STREAM），並使用 SO_REUSEADDR 優化。若連線失敗則終止程式。

[2] **檔案緩衝：**

使用 f = sock.mokefile（encoding = 'utf-8'）將 Socket 轉換為檔案物件，便於使用 f.readline ( ) 讀取以換行符號為邊界的 JSON 封包。

[3] **登入流程：**

循環要求輸入暱稱，發送 JSON 請求。

等待 Server 回應，若回應包含 error 欄位，則要求重新輸入暱稱。

**2. Send_message ( ) 執行緒（輸入處理）：**

此執行緒負責讀取鍵盤輸入，是單向發送邏輯。

[1] **指令識別：**

檢查使用者輸入是否為本地指令（/create、/join、/leave 等）。

[2] **本地狀態更新：**

若為 /leave，除發送 /leave 訊息給 Server 以外，還會將客戶端本地的 room_name 變數設定為 None，作為客戶端防呆機制。

[3] **封包發送：**

所有訊息（包含指令）都封裝為 JSON 後，使用 sock.sendoll ( )傳送。

[4] **錯誤處理：**

捕捉 ConnectionReseError 和 BrokenPipeErro，判定為伺服器斷線並結束執行檔。

**3. Recv_message ( ) 執行緒（輸出處理）：**

此執行緒負責接收並顯示 Server 訊息。

[1] **接收與解析：**

循環使用 f.readline ( ) 讀取，並用 json.loads ( ) 解析為字典物件。

[2] **訊息顯示：**

將 Server 回傳訊息格式化為 [ 時間 ] [ 暱稱 ]：[ 訊息 ]顯示。

[3] **自動狀態同步：**

特別處裡 nickname == '系統' 的訊息。

如果訊息包含「加入房間」或「離開房間」的提示，此函數會自動更新全域變數 room_name，確保 send_message 執行緒能正確判斷玩家是否在房間內，以進行指令防呆。

## 2.2 伺服器端（Server.py）邏輯分析

server.py 是整個系統的核心，其複雜度體現在數據結構和執行緒同步上。

**1. 核心數據架構：**

[1] **client_list：**包含所有連線 Socket 與暱稱的列表。

[2] **rooms：**字典型資料庫。Key 為房名，Value 是一個包含以下重要元素的字典：

- **members:** 房間內玩家詳細資訊（含 nickname, socket, alive, game_role, is_idiot 等）。
- **state:** 房間狀態 (waiting / playing)。
- **game:** 儲存本局遊戲的暫存數據（如 day_votes, wolf_target, phase）。
- **lock:** threading.Lock ( ) 物件，用於保護 game 字典與 members 列表在多執行緒寫入時的同步安全。

**2. 多執行緒同步：threading.Lock() 證據與數據保全：**

在多執行緒環境中，多個 client_thread 可能同時嘗試對 rooms[room_name]['game'] 進行寫入操作（如多人同時投票）。為了避免 競態條件 (Race Condition)，我們使用了 互斥鎖 (Mutex) 機制。

[1] **Lock 證據：**在 server.py 的房間創建函數中，為每個新房間實例化

一個獨立的鎖：

**第 520 行：** rooms[r_name] = {'password': r_pass, 'host': nickname, 'members': [me], 'state': 'waiting', 'lock': threading.Lock(), 'game': {}}

[2] **保全處理與程式碼實證：**

所有涉及修改 day_votes 或 wolf_target 等共享數據的操作，都必須被 with rooms[room_name]['lock']: 語句包圍。這保證了寫入操作的原子性 (Atomicity)。

- **保全目標：**防止多個玩家的投票數據互相覆蓋。
- **程式碼實證：**

  **第650行：**

  with rooms[room_name]['lock']: game['day_votes'][nickname] = target

**3. 輔助功能與廣播機制：**

[1] **json_msg ( )：**將發送者和內容快速封裝成 JSON 格式。

[2] **send_private_msg ( )：**用於發送身分卡、預言家結果等私密資訊。

[3] **broadcast_room ( )：**

- 房間內所有存活玩家廣播。
- 夜間訊息隔離：若 phase 處於夜間階段，會根據玩家身分決定其訊息可見範圍（如：狼人發言只有狼人能收到）。

[4] **broadcast_ghost_room ( )：**

專門向所有 alive: False 的玩家廣播（鬼魂頻道）。

[5] **check_game_over ( )：**

判定好人或狼人數量是否滿足勝利條件。

**4. client_thread ( ) (連線與指令處理)：**

每個連線的客戶端都會啟動一個 client_thread：

[1] **登入：**處理登入請求，檢查暱稱重複性，成功後加入 client_list。

[2] **指令路由：**

- **大廳指令：**處理 /create, /join (檢查密碼、滿房)。
- **遊戲開始：**房主發送 /start 時，檢查人數（4-12 人），呼叫

assign_roles()，並啟動遊戲執行緒 threading.Thread(target=start_werewolf_game, ...)。

- **遊戲動作：**

  **嚴格執行三重驗證：**存活驗證 (alive)、階段驗證 (phase 是否對應)、身分驗證 (game_role 是否擁有權限)。

  **同步鎖定：**通過驗證後，所有涉及修改 rooms 內共享數據的操作都必須在 with rooms[room_name]['lock']: 的保護下進行。

**5. 遊戲主流程執行緒 (start_werewolf_game)狀態機核心：**

此函式是遊戲的核心控制單元，以 狀態機 模式運作，控制日夜流程：

[1] **狀態循環：**進入 while game_running 迴圈，依序切換 phase：wolf → guard → seer → witch → day。

[2] **等待機制 (wait_for_action)：**

- **設定階段時間（Timeout）。**
- **提早結束：**函式內建邏輯會檢查共享數據（例如狼人是否全部投票完畢）。若數據已齊全，立即 return，不等待倒數結束

[3] **夜間結算 (Night Resolution)：**

- **複雜邏輯判定：**結合 wolf_target、guard_target、witch_action 進行結算。
- **衝突處理：**若狼人目標被守衛守護 且 被女巫使用解藥，判定為「同守同救」，即平安；若僅被解藥救，則 wolf_target 設為 None。
- **將最終死者標記 alive: False。**

[4] **白天結算 (Day Resolution)：**

- **投票統計：**使用 collections.Counter 統計 day_votes。
- **平票處理：**平票則無人死亡。
- **白癡機制：**若被處決者為「白癡」，則將其 alive 設為 True，但標記 is_idiot: True（失去投票權）。
- **報復機制：**若死者為「狼王」或「獵人」，流程暫停，進入短暫的 revenge 階段，等待玩家開槍指令。

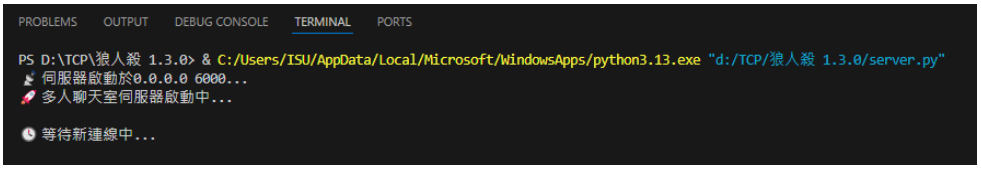
# 第三章、安裝與執行結果

## 3.1 環境需求

1. **Python 3.13**
2. **標準函式庫**：socket, threading, json, time, platform, random, collections。

## 3.2 執行步驟與結果

1. **啟動伺服器 (server.py)：**




2. **啟動客戶端且輸入暱稱 (Client.py)：**



3. **輸入操作指令 (於 Client 端輸入)：**

```
/create
請輸入房間名稱：test
請輸入房間密碼：123
[16:40:09] [系統]：房間 test 建立成功，你是房主 👑
```

```
/who
[16:42:23] [系統]：房間成員:
BOB1 (房主) 👑
BOB5
BOB6
BOB2
BOB7
BOB3
BOB4
BOB8
```

## 4. 遊戲過程片段：

```
/start
[16:42:54] [系統]：🎮 遊戲開始！
[16:42:54] [系統]：======== 遊戲開始 ========
您分配到的職業是：【**女巫**】
[16:42:54] [系統]：✅ 角色已分配完畢，共 8 人，準備進入第一夜。
[16:42:54] [系統]：
========== 第 1 夜 ==========
[16:42:55] [系統]：🌙 夜晚降臨，所有玩家請閉眼 😴
[16:42:56] [系統]：🐺 狼人請睜眼 👀
[16:43:39] [系統]：🐺 狼人請閉眼 😴
[16:43:39] [系統]：🛡️守衛請睜眼 👀
[16:44:00] [系統]：🛡️守衛請閉眼 😴
[16:44:00] [系統]：🔮 預言家請睜眼 👀
[16:44:15] [系統]：🔮 預言家請閉眼 😴
[16:44:15] [系統]：💃♀️女巫請睜眼 👀
[16:44:15] [系統]：💃♀️請選擇操作
==========
本晚狼人欲殺害：**BOB5**。
解藥：✅ 有 | 毒藥：✅ 有
存活名單：BOB1, BOB5, BOB6, BOB2, BOB7, BOB3, BOB4, BOB8
==========
指令：毒藥 <玩家名> / 解藥 <玩家名> / 不使用
不使用
[16:44:25] [系統]：已選擇：不使用
毒藥 BOB4
[16:44:36] [系統]：已選擇：毒藥 BOB4
[16:44:36] [系統]：💃♀️女巫請閉眼 😴
[16:44:36] [系統]：（女巫使用了毒藥）
[16:44:37] [系統]：🌞 天亮了，昨晚死亡的是：BOB4, BOB5
[16:44:37] [系統]：🗣️存活玩家：BOB1, BOB6, BOB2, BOB7, BOB3, BOB8
[16:44:37] [系統]：🌞請討論並投票。指令：`投票 <玩家名>` 或 `投票 棄票`
[16:45:18] [BOB2]：投 BOB4
[16:45:24] [BOB2]：投 BOB7
投票 BOB7[16:45:38] [BOB7]：投 BOB7

[16:45:38] [系統]：✅ 你投給了：BOB7
[16:46:08] [系統]：📦投票結束，正在計票...
[16:46:09] [系統]：BOB6 投給了 BOB1
BOB2 投給了 BOB7
BOB1 投給了 BOB7
BOB3 投給了 BOB7
BOB7 投給了 BOB7
BOB8 投給了 BOB7
[16:46:09] [系統]：⚖️經過多數決投票，**BOB7** 被處決了。
[16:46:09] [系統]：即將進入下一夜...
[16:46:12] [系統]：
```


========== 第 2 夜 ==========
[16:46:13] [系統]：🌙 夜晚降臨，所有玩家請閉眼 😴
[16:46:14] [系統]：🐺 狼人請睜眼 👀
[16:46:33] [系統]：🐺 狼人請閉眼 😴
[16:46:33] [系統]：🧙‍♀️女巫請睜眼 👀
[16:46:33] [系統]：🧙‍♀️請選擇操作
==========
本晚狼人欲殺害：**BOB3**。
解藥：✅ 有 | 毒藥：❌ 無
存活名單：BOB1, BOB6, BOB2, BOB3, BOB8
==========
指令：毒藥 <玩家名> / 解藥 <玩家名> / 不使用
[16:47:27] [系統]：🔔 **請注意！** 剩餘 5 秒！
[16:47:28] [系統]：🔔 **請注意！** 剩餘 4 秒！
[16:47:29] [系統]：🔔 **請注意！** 剩餘 3 秒！
[16:47:30] [系統]：🔔 **請注意！** 剩餘 2 秒！
[16:47:31] [系統]：🔔 **請注意！** 剩餘 1 秒！
[16:47:33] [系統]：❌ **時間到！** 行動結束。
[16:47:33] [系統]：🧙‍♀️女巫請閉眼 😴
[16:47:33] [系統]：🏹 獵人 BOB3 死亡！請獵人開槍。
[16:47:34] [系統]：🌞 天亮了，昨晚死亡的是：BOB3
[16:47:34] [系統]：🏆 遊戲結束：狼人陣營數量等於或大於好人陣營，狼人陣營獲勝！

# 第四章、未來改進方向

## 4.1 圖形化介面 (GUI)：

改用 PyQt 或 Tkinter 模組，以圖形化取代 CLI 介面，提供更豐富的使用者體驗。

## 4.2 數據持久化：

引入 SQLite 或 MySQL 資料庫，取代記憶體中的 rooms 字典，以保存歷史戰績、玩家設定等。。

## 4.3 網路穩定性與容錯：

- 實作 心跳包 (Heartbeat) 機制，更精準地判斷連線狀態。
- 增加斷線重連機制，允許遊戲中的玩家在短時間內恢復連線而不丟失狀態。

## 4.4 安全性增強：

- 對關鍵通訊內容進行 加密，如使用 SSL/TLS 協定。
- 增加對惡意輸入（如過長字串、格式錯誤的 JSON）的防護，防止服務器資源耗盡或崩潰。

# 參考文獻


1. **Source Code**：server.py - Python Server Implementation with Multi-threading and Game State Machine.
2. **Source Code:**：Client.py - Python Client Implementation with Dual-threading I/O.
3. **Python Software Foundation.** "socket — Low-level networking interface." Python 3 Documentation.
4. **Python Software Foundation.** "threading — Thread-based parallelism." Python 3 Documentation.