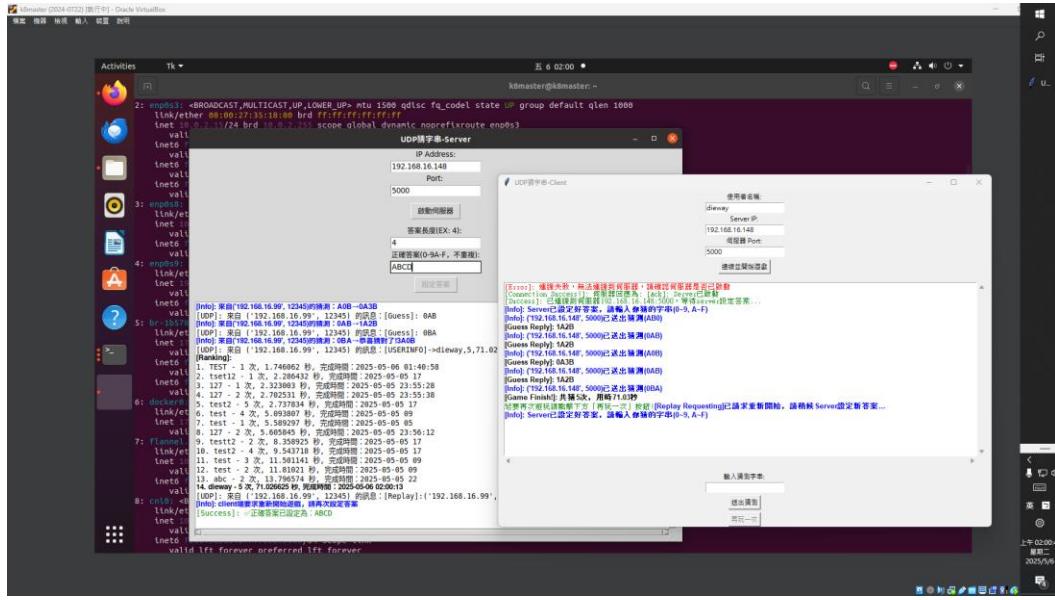


進階電網作業 - UDP 猜字串遊戲說明文件

S1154005-李育林(配分 100%)



1. 程式功能說明

本程式為一個基於 Python Tkinter GUI 與 UDP socket 網路通訊協定的「猜字串」小遊戲，分為 Server 與 Client 端，玩家可透過 GUI 輸入猜測，並即時收到 Server 的回應。基本功能包含：

功能名稱:	功能說明:
Server 設定答案及進行答案比對判斷	Server 端可設定本輪猜字串遊戲的答案長度及答案，並且能夠接收 Client 傳送的“Guess”訊息來判斷幾 A 幾 B 並回傳給 Client 端
Server/Client 處理接收到封包	接收另一端傳送的封包並依據各類封包進行判斷並處理
Client 端 user 及連線 Server 設定功能	Client 端可透過 GUI 介面向決定要發起連線的 Server 和 username
Server 排行榜及排名功能	Server 端可透過存取 <code>rankings.json</code> 檔案永久讀取/儲存排行榜，並且最後猜對遊戲結束時，會依據遊戲時間告知 Client 排名及用時
再次遊玩及結束遊戲功能	當 Client 猜對答案時，可以選擇按下再玩一次或結束遊戲來決定要重置答案再次遊玩或結束

延伸功能包含：

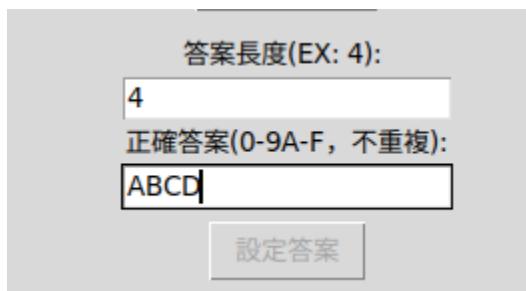
功能名稱:	功能說明:
Tkinter GUI 顯示介面	將原本只有文字輸入介面設計出 GUI 介面，方便視覺化使用
Server 端設定答案及 Client 端猜測字串防呆輸入	遊戲過程會判斷答案輸入，若不符合條件(EX: 答案不符合長度、輸入值為空...)則會回報錯誤提醒使用者。
遊戲流程防呆偵測	Client 啟動時確保和 Server 端的連線可以建立，以及當 Server 端設定好答案時才可進行猜測
超時自動結束(timeout)	Server 端和 Client 端都會有各自的 Timer 計時，若超過一定時間沒有任何動作則會自動 timeout socket 中斷 socket 的監聽
訊息樣式標記	遊戲過程皆會使用各個樣式的訊息輸出，方便使用者閱讀，並且提醒使用者目前進行的遊戲階段

程式功能說明

◆ Server 設定答案及進行答案比對判斷

Server 端的 GUI 中提供兩個欄位供管理者輸入，且皆提供防呆機制：

1. 答案長度 (N)：輸入一個正整數，代表本輪遊戲答案的長度。
2. 正確答案：由使用者自訂輸入一組長度為 N 的字串，必須由 0-9 或 A-F 中的字符組成，且不得重複。



設定好答案後，Client 每次猜字串送出 [Guess]: XXXX 格式的字串，Server 會解析字串，並與目前的正確答案進行比對產生幾 A 幾 B 回傳給 Client。

```
elif msg.startswith("[Guess]:"):
    if not self.answer:
        server_reply = "[Error]: 伺服器尚未設定好答案"
    else:
        guess = msg.split(":")[1].strip()
        server_reply = self.check_client_guess(guess)
        self.modify_output_text(f"[Info]: 來自{addr}的猜測:{guess}→{server_reply}\n", "info")
        self.server_socket.sendto(f"[Guess Reply]: {server_reply}".encode(), self.client_address)
```

```

def check_client_guess(self, guess: str):
    # 驗證 client 猜測的結果
    if len(guess) != self.answer_length:
        return f"[Error]: 格式錯誤，請輸入{self.answer_length}位數字"
    A = sum(a == b for a, b in zip(guess, self.answer))
    B = sum(min(guess.count(x), self.answer.count(x)) for x in set(guess)) - A
    if A == self.answer_length:
        return f"恭喜猜對了! {A}A{B}B"
    return f"{A}A{B}B"

```

▲Server 判斷 Guess 封包與答案進行判斷並回傳封包給 Client

```

def send_guess(self):
    """送出猜測值至伺服器"""
    guess = self.guess_entry.get().strip().upper()
    self.reset_timeout_timer()

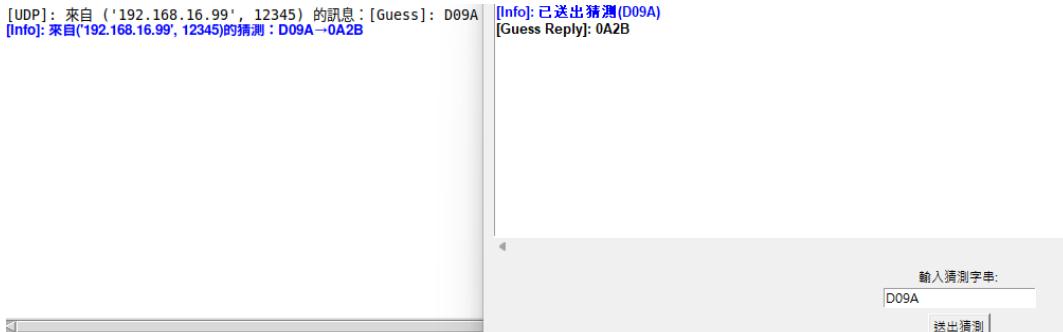
    if not self.ready_to_guess:
        self.modify_output_text("[Waiting...]: 等待Server設定答案\n")
        return

    if any(c not in "0123456789ABCDEF" for c in guess):
        self.modify_output_text("[Error]: 請只輸入0-9或A-F的字元\n", "error")
        return
    if len(set(guess)) != len(guess):
        self.modify_output_text("[Error]: 請確認輸入字元皆不重複\n", "error")
        return
    if len(guess) != self.answer_length:
        self.modify_output_text(f"[Error]: 請輸入{self.answer_length}個字元\n", "error")
        return

    self.guess_count += 1
    try:
        self.socket.sendto(f"[Guess]: {guess}".encode(), self.server_address)
        self.modify_output_text(f"[Info]: 已送出猜測({guess})\n", "info")
    except Exception as e:
        self.modify_output_text(f"[Error]: 傳送失敗({e})\n", "error")

```

▲Client 端按下送出猜測按鈕向 Server 端送出 Guess 封包給 Server 進行判斷



✧ Client 端 user 及連線 Server 設定功能

Client 端可以指定連線 Server 並且設定自己的名字，可以在遊戲結束時進行排名，並且當輸入欄都填好後按下連線按鈕就能開始連線進行遊戲

```

def start_game(self):
    """啟動遊戲流程[1]建立socket[2]發出初始訊息並啟動接收執行緒"""
    ip = self.ip_entry.get()
    port = self.port_entry.get()
    username = self.name_entry.get()

    if not ip or not port or not username:
        self.modify_output_text("[Error]: 請填寫完整IP、Port和Username\n", "error")
        return

    try:
        port = int(port)
    except ValueError:
        self.modify_output_text("[Error]: 請確認填寫之port為整數數字\n", "error")
        return

    self.server_address = (ip, port)
    self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) You, 20 hours ago • 1st main
    self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    self.socket.bind(self.client_address)
    self.socket.settimeout(3)

    if not self.test_server_connection(self.server_address):
        self.modify_output_text("[Error]: 連線失敗，請確認伺服器是否啟動\n", "error")
        self.socket.close()
        return

    self.modify_output_text(f"[Success]: 已連線到伺服器{ip}:{port}[3]，等待Server設定答案...\n", "success")
    self.game_running = True
    self.username = username
    self.guess_count = 0
    self.start_button.config(state="disabled")
    self.receive_thread = threading.Thread(target=self.receive_response, daemon=True)
    self.receive_thread.start()
    self.reset_timeout_timer()

```

▲傳送開始遊戲的封包給 Server 端，且會先確認是否已和 Server 端建立連線



◆ Server/Client 處理接收到封包

在 UDP 協定下，資料是以封包形式非同步傳輸，Server 與 Client 必須正確辨識接收到的每個封包內容，並做出相對應的處理與回應。這是遊戲能正確運作的關鍵。

Server 端會接收到的封包和處理功能：

封包開頭	功能說明
[Connecting]:	表示 Client 初次連線，Server 回

	應確認並儲存對方地址
[Guess]:	Client 傳送猜測答案，Server 比對後回傳幾 A 幾 B 的結果
[USERINFO]:	Client 猜對後傳送使用者資料，Server 記錄進排行榜
[Replay]:	Client 請求再次遊玩，Server 重置遊戲狀態並提示設定新答案
[Timeout]:	Client 通知因閒置過久終止遊戲，Server 被動結束本輪
QUIT	Client 主動結束遊戲，Server 關閉連線與 socket

```

def receive_messages(self):
    # 接收 client 傳送的封包，並進行處理
    while self.socket_running:
        try:
            if self.server_socket:
                data, addr = self.server_socket.recvfrom(1024)
                self.client_address = addr
                self.reset_timeout_timer()
                msg = data.decode()
                self.modify_output_text(f"[UDP]: 來自 {addr} 的訊息 :{msg}\n")

                # 判斷各類封包種類做處理
                if msg.startswith("[Connecting]"):
                    self.modify_output_text(f"[Success]: 收到來自{addr}的連接訊息\n", "success")
                    self.client_address = addr
                    self.game_running = True
                    self.set_answer_button.config(state='normal')
                    self.server_socket.sendto("[Ack]: Server已啟動".encode(), addr)
                elif msg.startswith("[Guess]:"):
                    if not self.answer:
                        server_reply = "[Error]: 伺服器尚未設定好答案"
                    else:
                        guess = msg.split(":")[1].strip()
                        server_reply = self.check_client_guess(guess)
                    self.modify_output_text(f"[Info]: 來自{addr}的猜測 :{guess}+{server_reply}\n", "info")
                    self.server_socket.sendto(f"[Guess Reply]: {server_reply}".encode(), self.client_address)
                elif msg.startswith("[USERINFO]"):
                    data = msg.split(">")[1].split(",")
                    username, guess_count, duration, finish_time_str = data[0], int(data[1]), float(data[2]), data[3]
                    self.add_user_rankings(username, guess_count, duration, finish_time_str)
                    rank = self.get_rank(username, duration, finish_time_str)
                    self.server_socket.sendto(f"[Congratulations]!: {username} ! 你是第 {rank} 名!\n".encode(), self.client_address)
                    self.show_rankings(highlight_username=username, highlight_time=duration, highlight_finish=finish_time_str)
                elif msg.startswith("[Replay]:"):
                    self.modify_output_text(f"[Info]: client端要求重新開始遊戲 [請再次設定答案]\n", "info")
                    self.answer = ""
                    self.answer_length = 0
                    self.set_answer_button.config(state='normal')
                    self.client_address = addr
                elif msg.startswith("[Timeout]:"):
                    self.modify_output_text(f"[{msg}]\n", "error")
                    self.answer = ""
                    self.answer_length = 0
                    self.client_address = None
                    self.socket_running = False
                    self.game_running = False
                    self.set_answer_button.config(state="normal")
                    self.server_socket.close()
                elif msg == "QUIT":
                    self.stop_server()
        except Exception as e:
            print(f"An error occurred: {e}")

```

Client 會接收到的封包和處理功能：

封包開頭	功能說明
------	------

[Ready]:	Server 完成答案設定，Client 開始開放猜測區與啟動計時
[Guess Reply]:	回傳幾 A 幾 B 或猜對提示，更新畫面與紀錄猜測次數
[Congratulations!]:	顯示最終名次結果
[Timeout]:	顯示 timeout 訊息並關閉猜測與按鈕功能

```

def receive_response(self):
    """接收伺服器的所有訊息並處理各種回應"""
    while self.game_running:
        try:
            data, _ = self.socket.recvfrom(1024)
            response = data.decode()
            self.reset_timeout_timer()

            if response.startswith("[Ready]:"):
                self.ready_to_guess = True
                self.answer_length = int(response.split(":")[1].split("[")[-1])
                self.modify_output_text(f"[Info]: Server已設定答案(長度{self.answer_length})，可開始猜測\n", "info")
                self.guess_button.config(state='normal')
                self.start_time = datetime.now()
                self.guess_count = 0
            elif response.startswith("[Guess Reply]:") and "恭喜猜對" in response:
                duration = (datetime.now() - self.start_time).total_seconds()
                finish_time_str = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                self.modify_output_text(f"[Game Finish!]: 共猜{self.guess_count}次，用時{duration:.2f}秒\n", "bold")
                self.modify_output_text("若要再次遊玩請點下方「再玩一次」或按「結束遊戲」離開\n", "success")
                self.guess_button.config(state="disabled")
                self.replay_button.config(state="normal")
                self.quit_button.config(state="normal")
                userinfo_msg = f"[USERINFO] > {self.username},{self.guess_count},{duration},{finish_time_str}"
                self.socket.sendto(userinfo_msg.encode(), self.server_address)
            elif response.startswith("[Guess Reply]"):
                self.modify_output_text(f"\n{response}\n", 'bold')
            elif response.startswith("[Congratulations!]:"):
                txt = response.split(":")[1].strip()
                self.modify_output_text(f"\n{txt}\n", 'bold')
            elif response.startswith("[Timeout]:"):
                self.modify_output_text(f"\n{response}\n", "error")
                self.guess_button.config(state="disabled")
                self.replay_button.config(state="disabled")
                self.quit_button.config(state='disabled')
                self.game_running = False
                self.start_button.config(state="normal")
                self.modify_output_text("[Info]: 客戶端socket已關閉\n", "info")
                self.socket.close()
            except OSError:
                break
            except Exception:
                continue
        
```

❖ Server 排行榜及排名功能

使用 `rankings.json` 檔案作為資料儲存媒介，Server 在一開始執行就會先讀取過去紀錄排行榜，在玩家猜對時會顯示並告訴玩家計時排名。

```

def load_rankings(self):
    # 載入歷史排行榜資料
    if os.path.exists(RANKING_FILE):
        with open(RANKING_FILE, "r") as f:
            try:
                self.rankings = json.load(f)
            except json.JSONDecodeError:
                self.rankings = []
    else:
        self.rankings = []

def save_rankings(self):
    # 儲存排行榜資料至 JSON 檔
    with open(RANKING_FILE, "w") as f:
        json.dump(self.rankings, f, indent=2)

```

▲Server 端 json 排行榜讀取和存檔相關 function

```

def show_rankings(self, highlight_username=None, highlight_time=None, highlight_finish=None):
    # 顯示排行榜內容，並標示本次紀錄
    self.modify_output_text("[Ranking]:\n", "bold")
    for i, rec in enumerate(self.rankings, 1):
        is_highlight = (
            rec["name"] == highlight_username and
            rec["time"] == highlight_time and
            rec["finish_time"] == highlight_finish
        )
        line = f"{i}. {rec['name']} - {rec['guesses']} 次, {rec['time']} 秒, 完成時間: {rec['finish_time']}\n"
        if is_highlight:
            self.output_text.config(state="normal")
            self.modify_output_text(line, "bold")
            self.output_text.config(state="disabled")
        else:
            self.modify_output_text(line)

def add_user_rankings(self, username, guess_count, duration, finish_time_str):
    # 將猜對使用者資料加入排行榜並排序
    self.rankings.append({
        "name": username,
        "guesses": guess_count,
        "time": duration,
        "finish_time": finish_time_str
    })
    self.rankings.sort(key=lambda x: x["time"])
    self.save_rankings()

def get_rank(self, username, time_used, finish_time_str):
    # 查詢該使用者的名次
    for i, rec in enumerate(self.rankings):
        if rec["name"] == username and rec["time"] == time_used and rec["finish_time"] == finish_time_str:
            return i + 1
    return -1

```

▲Server 端在接受到 Client 猜對回傳的玩家遊戲資訊後，會先在處理封包時透過 `add_user_rankings()` 將玩家加入排行榜，並透過 `show_ranking()` 在 Server 端顯示排行榜，最後透過 `get_rank()` 回傳玩家排名資訊回 Client 端

```
[UDP]: 來自 ('192.168.16.99', 12345) 的訊息:[USERINFO]->gamer1,2,5613.816441,2025-05-06 05:52:44
[Ranking]:
1. TEST - 1 次, 1.746062 秒, 完成時間: 2025-05-06 01:40:58
2. tset12 - 1 次, 2.286432 秒, 完成時間: 2025-05-05 17
3. 127 - 1 次, 2.323003 秒, 完成時間: 2025-05-05 23:55:28
4. hello - 1 次, 2.469817 秒, 完成時間: 2025-05-06 02:39:37
5. 127 - 2 次, 2.702531 秒, 完成時間: 2025-05-05 23:55:38
6. test2 - 5 次, 2.737834 秒, 完成時間: 2025-05-05 17
7. hello - 1 次, 2.766797 秒, 完成時間: 2025-05-06 02:35:30
8. dieway - 1 次, 3.807614 秒, 完成時間: 2025-05-06 02:29:48
9. test - 4 次, 5.093807 秒, 完成時間: 2025-05-05 09
10. test - 1 次, 5.589297 秒, 完成時間: 2025-05-05 05
11. 127 - 2 次, 5.605845 秒, 完成時間: 2025-05-05 23:56:12
12. hello - 1 次, 6.734774 秒, 完成時間: 2025-05-06 02:35:22
13. dieway - 1 次, 6.828645 秒, 完成時間: 2025-05-06 02:28:32
14. testt2 - 2 次, 8.358925 秒, 完成時間: 2025-05-05 17
15. test2 - 4 次, 9.543718 秒, 完成時間: 2025-05-05 17
16. test - 3 次, 11.501141 秒, 完成時間: 2025-05-05 09
17. test - 2 次, 11.81021 秒, 完成時間: 2025-05-05 09
18. abc - 2 次, 13.796574 秒, 完成時間: 2025-05-05 22
19. dieway - 6 次, 38.703549 秒, 完成時間: 2025-05-06 02:01:21
20. dieway - 5 次, 71.026625 秒, 完成時間: 2025-05-06 02:00:13
21. gamer1 - 2 次, 5613.816441 秒, 完成時間: 2025-05-06 05:52:44
```

▲Server 端在玩家猜對時會列出目前所有排行榜玩家資訊，包括猜的次數、花費時間、系統時間…

```
[Info]: 已送出猜測(D09A)
[Guess Reply]: 0A2B
[Info]: 已送出猜測(ABCD)
[Game Finish]: 共猜2次, 用時5613.82秒
若要再次遊玩請點下方「再玩一次」或按「結束遊戲」離開
gamer1 ! 你是第 21名!
```

▲Client 端在玩家猜對時會告訴玩家猜的次數和用時，並且告訴玩家位在排行榜第幾名

◆ 再次遊玩及結束遊戲功能

使用者在猜對後可以選擇「再次遊玩」或「結束遊戲」，讓玩家可依照意願決定是否繼續進行下一場遊戲，或結束當前連線與關閉程式。

```
def replay_game(self):
    """重新開始遊戲流程"""
    self.reset_timeout_timer()
    if self.socket:
        self.socket.sendto(f"[Replay]:{self.client_address}".encode(), self.server_address)
        self.replay_button.config(state="disabled")
        self.quit_button.config(state="disabled")
        self.guess_entry.delete(0, tk.END)
        self.modify_output_text("[Replay Requesting]已請求重新開始, 請稍候Server設定新答案...\n", "info")

def quit_game(self):
    """結束遊戲, 傳送QUIT給伺服器並關閉視窗"""
    try:
        quit_message = "QUIT"
        self.socket.sendto(quit_message.encode(), self.server_address)
    except Exception as e:
        print(f"Error sending quit message: {e}")
    finally:
        self.socket.close()
        self.root.destroy()
```

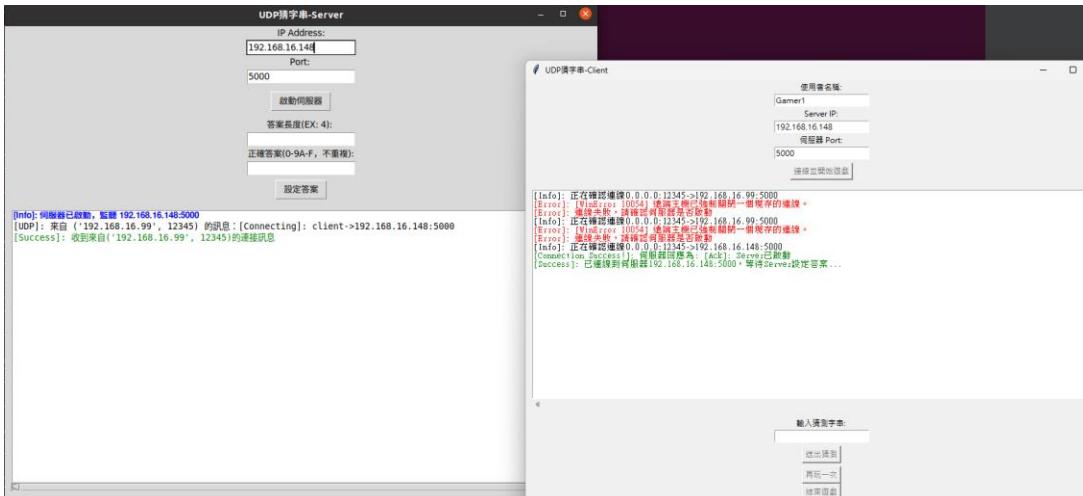
▲Client 端在最後可以選擇再玩一次或結束遊戲，分別會傳送 Replay 和 QUIT 封包給 Server 端進行後續流程處理

◆ Tkinter GUI 顯示介面

依照原本作業要求應該是單純使用 CLI 來進行整個程式，並且 socket 監聽 IP 都要寫死在程式中。由於閱讀不易加上修改麻煩，因此我使用了 Python 的 tkinter 函式庫設計了 GUI 介面讓使用起來更好看也更方便。

```
# ===== IP與Port設定區塊 =====
self.IP_frame = tk.Frame(self.root)
tk.Label(self.IP_frame, text="IP Address:").pack()
self.ip_entry = tk.Entry(self.IP_frame)
self.ip_entry.insert(0, "127.0.0.1")
self.ip_entry.pack()
tk.Label(self.IP_frame, text="Port: ").pack()
self.port_entry = tk.Entry(self.IP_frame)
self.port_entry.insert(0, "5000")
self.port_entry.pack()
self.start_button = tk.Button(self.IP_frame, text="啟動伺服器", command=self.start_server)
self.start_button.pack(pady=10)
```

▲Server 端 tkinter 設計界面片段程式碼，在程式啟動階段就開始使用 tkinter 函式庫內的組件渲染出畫面，由於程式碼篇幅較長且大致相同，因此只節錄片段。



▲左邊為 server 端之 GUI 介面，右邊為 Client 端之 GUI 介面

◆ Server 端設定答案及 Client 端猜測字串防呆輸入

設計當 Server 端在設定答案時若輸入不符合規則或有缺漏值，都會提醒使用者。同樣在 Client 端輸入 username、IP、port 時，若有缺漏值或非法值也都會提醒使用者，並且當在進行字串猜測時若輸入猜測長度錯誤或輸入其他非法值也都會提醒使用者達到防呆作用。

```

def set_answer(self):
    # 設定正確答案邏輯
    self.reset_timeout_timer()
    answer = self.answer_entry.get().upper()
    length = 0

    if not self.game_running:
        self.modify_output_text("[Info]: 目前無已連接之client，請再次確認是否已有連線\n", "info")
        self.set_answer_button.config(state="disabled")
        return

    try:
        length = int(self.answer_len_entry.get())
    except ValueError:
        self.modify_output_text("[Error]: 請確認答案長度為一整數\n", "error")
        return

    allowed_input_char = set("0123456789ABCDEF")
    if len(answer) != length:
        self.modify_output_text(f"[Error]: 答案長度應為 {length} 位\n", "error")
        return
    if any(c not in allowed_input_char for c in answer):
        self.modify_output_text("[Error]: 請確認答案僅包含0~9或A~F\n", "error")
        return
    if len(set(answer)) != length:
        self.modify_output_text("[Error]: 答案中有重複字元\n", "error")
        return

```

▲Server 端設定答案時，都有相對應防呆偵測

```

def send_guess(self):
    """送出猜測值至伺服器"""
    guess = self.guess_entry.get().strip().upper()
    self.reset_timeout_timer()

    if not self.ready_to_guess:
        self.modify_output_text("[Waiting...]: 等待Server設定答案\n")
        return

    if any(c not in "0123456789ABCDEF" for c in guess):
        self.modify_output_text("[Error]: 請只輸入0~9或A~F的字元\n", "error")
        return
    if len(set(guess)) != len(guess):
        self.modify_output_text("[Error]: 請確認輸入字元皆不重複\n", "error")
        return
    if len(guess) != self.answer_length:
        self.modify_output_text(f"[Error]: 請輸入{self.answer_length}個字元\n", "error")
        return

    self.guess_count += 1
    try:
        self.socket.sendto(f"[Guess]: {guess}".encode(), self.server_address)
        self.modify_output_text(f"[Info]: 已送出猜測({guess})\n", "info")
    except Exception as e:
        self.modify_output_text(f"[Error]: 傳送失敗({e})\n", "error")

```

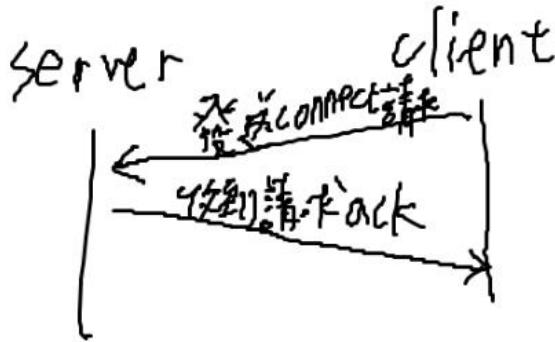
▲Client 端猜答案時，也有相對應防呆偵測



▲上述兩圖分別為 Server 端設定答案及 Client 端猜答案的防呆實際畫面

◇ 遊戲流程防呆偵測

有時候當 client 開始連線時可能還沒有可用 Server 啟動導致後續運作流程錯亂，因此我設計了一個模擬 TCP connect 的步驟，確認當 client 有連上 server 後才開始進行後續 Server 設定答案，Client 猜的遊戲過程。同時也會在不同遊戲過程調整 GUI 介面可使用的按鈕，以防未知的流程錯誤(在還未設定好答案 Server 端就送出猜字串、遊戲進行到一半就重新設定答案…)。

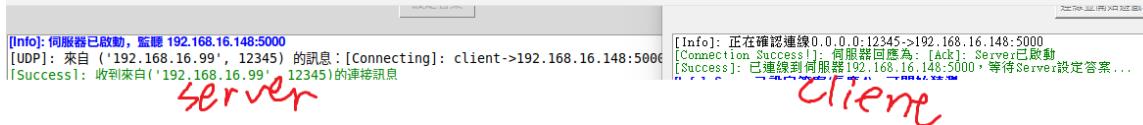


```

def test_server_connection(self, server_address):
    """測試連線用，發出測試封包並回傳是否成功"""
    ip = server_address[0]
    port = server_address[1]
    try:
        self.modify_output_text(f"[Info]: 正在確認連線{self.client_address[0]}:{self.client_address[1]}->{ip}:{port}\n")
        self.socket.sendto(f"[Connecting]: client->{ip}:{port}".encode(), self.server_address)
        data, addr = self.socket.recvfrom(1024)
        self.modify_output_text(f"[Connection Success!]: 伺服器回應為: {data.decode()}\n", "success")
        return True
    except socket.timeout:
        return False
    except Exception as e:
        self.modify_output_text(f"[Error]: {e}\n", "error")
        return False
    finally:
        self.socket.settimeout(None)
if msg.startswith("[Connecting]"):
    self.modify_output_text(f"[Success]: 收到來自{addr}的連接訊息\n", "success")
    self.client_address = addr
    self.game_running = True
    self.set_answer_button.config(state='normal')
    self.server_socket.sendto("[Ack]: Server已啟動".encode(), addr)

```

▲Client 端會先發送 connecting 訊息去確認是否能連接上 Server 端，若 Server 端有成功收到則會回送一個 Ack 告知 Client 端連接成功。同時 Client 端也有設定一個 timeout 來防止一直無法連接程式卡在這動不了。



▲同時也會將連線訊息顯示在 GUI 介面，方便監測

✧ 超時自動結束(timeout)

為了防止玩家或伺服器閒置過久導致遊戲停滯，我使用了 threading 設計了一套 timeout 機制。若 Client 或 Server 任一端在指定時間內未有任何互動，系統會自動結束遊戲，釋放資源並通知對方。

流程：

1. 遊戲開始後，timeout 計時器即啟動。

2. 每次互動（如猜字、重新開始）都會重設計時器。
3. 若超過 120 秒未操作，timeout 執行並結束當前遊戲。
4. Server/Client 會釋放 socket，並提示用戶並無法繼續操作。

```
# timeout計時器相關
self.timeout_timer = None
self.TIMEOUT_DURATION = 120 # 玩家若 120 秒內沒動作就 timeout
self.game_running = False

def reset_timeout_timer(self):
    """重設閒置計時器（用於遊戲自動 timeout）"""
    if self.timeout_timer:
        self.timeout_timer.cancel()
    self.timeout_timer = threading.Timer(self.TIMEOUT_DURATION, self.handle_timeout)
    self.timeout_timer.start()
```

▲使用 threading 來進行 timer 計時，若累積到指定時間(TIMEOUT_DURATION)
就會執行下面 handle_timeout() 的操作

```
def handle_timeout(self):
    """處理 timeout（玩家太久沒動作自動結束）"""
    self.modify_output_text(f"[Timeout]: {self.TIMEOUT_DURATION}秒內無操作，遊戲自動結束\n", "error")
    self.game_running = False
    try:
        self.socket.sendto("[Timeout]: Client已閒置過久，自動中止遊戲".encode(), self.server_address)
    except Exception as e:
        self.modify_output_text(f"[Error]: 傳送timeout通知失敗: {e}\n", "error")
    if self.socket:
        try:
            self.socket.close()
            self.modify_output_text("[Info]: 客戶端socket已關閉\n", "info")
        except Exception as e:
            self.modify_output_text(f"[Error]: 關閉socket時錯誤:{e}\n", "error")
        finally:
            self.socket = None

    # 重設所有狀態
    self.answer_length = 0
    self.guess_count = 0
    self.server_address = None
    self.start_button.config(state="normal")
    self.guess_button.config(state="disabled")
    self.replay_button.config(state="disabled")
    self.quit_button.config(state='disabled')
```

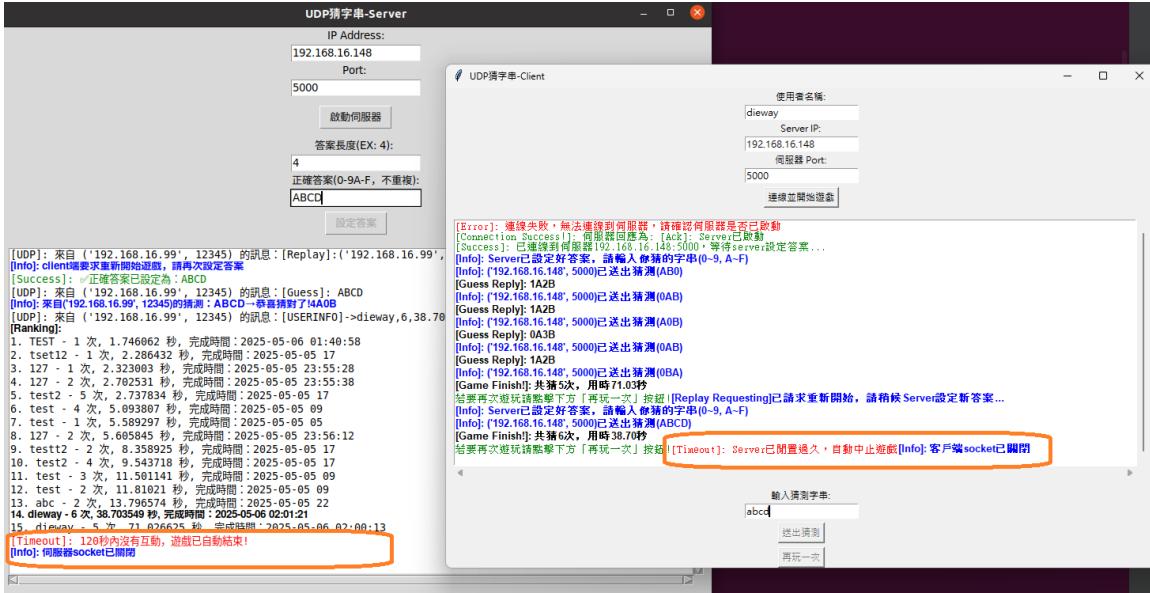
▲當 timeout 發生時，會去中止目前所有操作並回到最一開始的遊戲階段，同時也會告知另一個 Server/Client 端也要進行 timeout

<pre>elif response.startswith("[Timeout]:"): self.modify_output_text(f"{response}\n", "error") self.guess_button.config(state="disabled") self.replay_button.config(state="disabled") self.quit_button.config(state='disabled') self.game_running = False self.start_button.config(state="normal") self.modify_output_text("[Info]: 客戶端socket已關閉\n", "info") self.socket.close()</pre>	<pre>elif msg.startswith("[Timeout]:"): self.modify_output_text(f"{msg}\n", "error") self.answer = "" self.answer_length = 0 self.client_address = None self.socket_running = False self.game_running = False self.set_answer_button.config(state="normal") self.server_socket.close()</pre>
--	--

Client

Server

▲兩個端口都有對應處理收到來自對方 timeout 封包時的處理步驟



▲當其中一端由於過久沒有互動而導致 Timeout 時，也會透過 socket 告知另一端中止運行

◆ 訊息樣式標記

在本遊戲的 GUI 介面中，訊息輸出會集中顯示於一個 Text 區塊若顯示訊息全部都是單一顏色且同樣樣式會導致閱讀不易，因此在顯示訊息時，都能客製化修改顯示訊息的顏色及字體樣式，方便閱讀

例如：

錯誤訊息 → 紅色字

成功連線或猜對提示 → 綠色字

資訊提示（如開始遊戲、送出猜測） → 粗體藍色

排行榜本次成績 → 粗體標示

```
def modify_output_text(self, text, tag=None):
    # 輸出訊息至訊息區域，支援標籤樣式
    self.output_text.config(state="normal")
    if tag:
        self.output_text.insert(tk.END, text, tag)
    else:
        self.output_text.insert(tk.END, text)
    self.output_text.config(state="disabled")
    self.output_text.see(tk.END)
```

```

self.output_text.tag_config("bold", font=("Helvetica", 10, "bold"))
self.output_text.tag_config("error", foreground="red")
self.output_text.tag_config("success", foreground="green")
self.output_text.tag_config("info", foreground="blue", font=("Helvetica", 10, "bold"))

[Info]: client端要求重新開始遊戲，請再次設定答案
[Success]: ◎正確答案已設定為：ABCD
[UPP]: 來自 ('192.168.16.99', 12345) 的訊息：[Guess]: ABCD
[Info]: 來自('192.168.16.99', 12345)的猜測：ABCD—恭喜猜對了!4AOB
[UPP]: 來自 ('192.168.16.99', 12345) 的訊息：[USERINFO] ->dieway,6,3
[Ranking]:
1. TEST - 1 次, 1.746062 秒, 完成時間 : 2025-05-06 01:40:58
2. tset12 - 1 次, 2.286432 秒, 完成時間 : 2025-05-05 17
3. 127 - 1 次, 2.323803 秒, 完成時間 : 2025-05-05 23:55:28
4. 127 - 2 次, 2.702531 秒, 完成時間 : 2025-05-05 23:55:38
5. test2 - 5 次, 2.737834 秒, 完成時間 : 2025-05-05 17
6. test - 4 次, 5.093807 秒, 完成時間 : 2025-05-05 09
7. test - 1 次, 5.589297 秒, 完成時間 : 2025-05-05 05
8. 127 - 2 次, 5.605845 秒, 完成時間 : 2025-05-05 23:56:12
9. test2 - 2 次, 8.358925 秒, 完成時間 : 2025-05-05 17
10. test2 - 4 次, 9.543718 秒, 完成時間 : 2025-05-05 17
11. test - 3 次, 11.501144 秒, 完成時間 : 2025-05-05 09
12. test - 2 次, 11.81021 秒, 完成時間 : 2025-05-05 09
13. ab - 2 次, 13.796574 秒, 完成時間 : 2025-05-05 22
14. dieway - 6 次, 38.703549 秒, 完成時間 : 2025-05-06 02:01:21
15. dieway - 5 次, 71.026625 秒, 完成時間 : 2025-05-06 02:00:13
[Timeout]: 120秒內沒有互動，遊戲已自動結束!
[Info]: 伺服器socket已關閉

```

▲要想修改顯示樣式只要修改 tag 即可，若想要其他不同樣式也只要，這樣看起來是不是清楚明瞭許多！

2. 程式碼註解與功能對應說明

以下為 Server/Client 端的各個主要函式功能說明：

Server 函式功能對應表

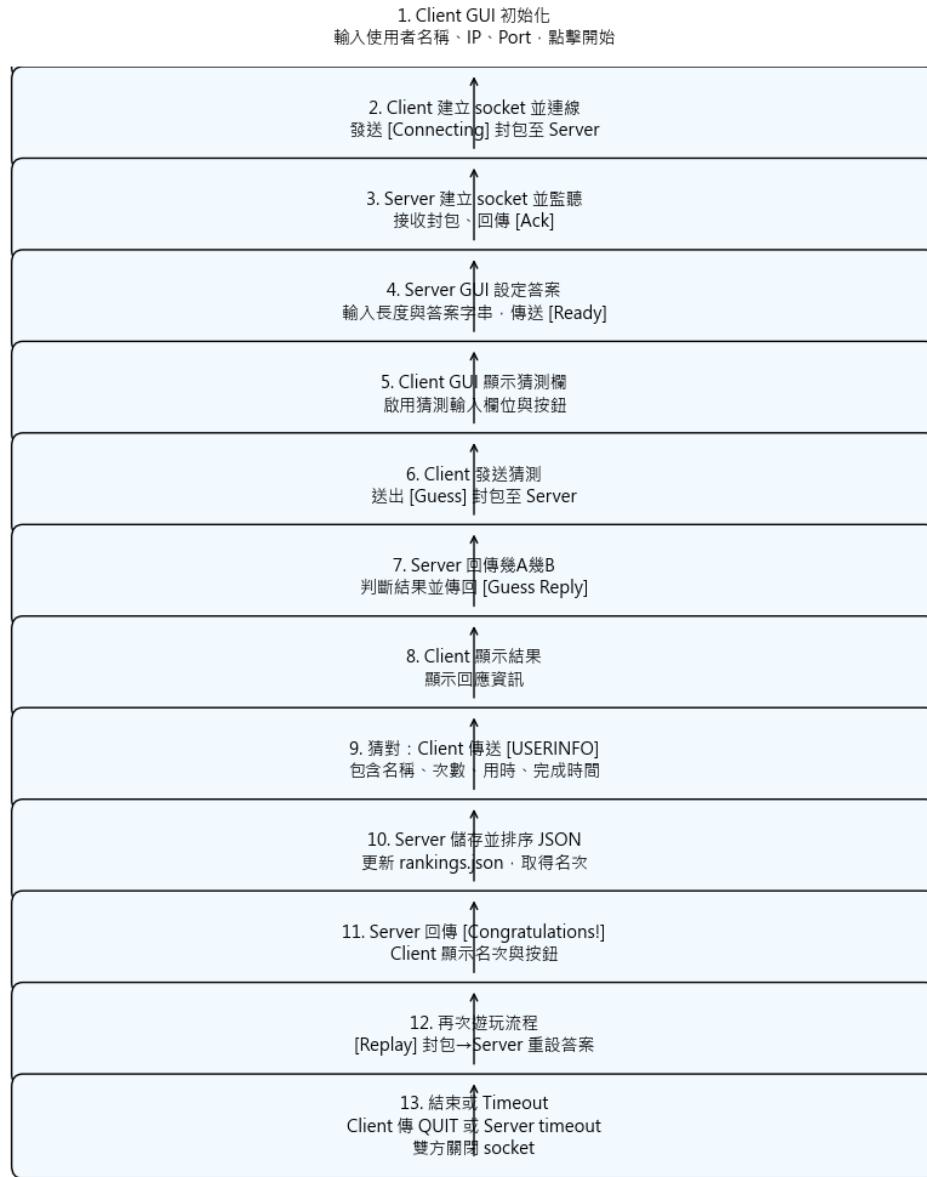
函式名稱	功能說明
start_server	啟動伺服器 socket 並開始監聽
check_client_guess	比對玩家猜測與答案，回傳幾 A 幾 B
receive_messages	接收 client 訊息並依據訊息類型做回應
stop_server	關閉伺服器與釋放資源
set_answer	設定答案與驗證格式，通知 client 遊戲開始
modify_output_text	更新 GUI 輸出區域文字，支援樣式
load_rankings	從 JSON 載入排行榜資料
save_rankings	儲存排行榜至 JSON 檔案
add_user_rankings	新增玩家成績並排序儲存

get_rank	查找玩家在排行榜中的名次
show_rankings	顯示排行榜，並標示本次成績
reset_timeout_timer	重設 timeout 計時器
handle_timeout	timeout 時關閉 socket 並通知 client

Client 函式功能對應表

函式名稱	功能說明
quit_game	傳送離線訊息並結束程式
replay_game	傳送 replay 請求並重置介面
test_server_connection	測試與伺服器的初始連線
start_game	初始化遊戲並啟動 socket、連線與接收
send_guess	送出玩家的猜測並更新畫面
receive_response	持續接收 server 回應並處理
modify_output_text	更新 GUI 輸出區域文字，支援樣式
reset_timeout_timer	重設 timeout 計時器
handle_timeout	timeout 時通知 server 並關閉自己

3. 程式流程說明



1. Client GUI 初始

使用者於 Client 端介面輸入使用者名稱、伺服器 IP 與 Port，並點擊「開始遊戲」以初始化連線流程。

2. Client 建立 socket 並嘗試連線

Client 利用 UDP socket 傳送 [Connecting] 封包至 Server 端，請求建立連線。

3. Server 建立 socket 並監聽

Server 接收到 Client 的連線請求，並回傳 [Ack] 封包，確認連線成功。

4. Server GUI 設定答案

Server 使用者於 GUI 介面中設定答案長度與實際答案字串，並向 Client 傳送 [Ready] 封包，表示可以開始猜測。

5. Client 顯示猜測欄位

Client 收到 [Ready] 封包後，顯示可輸入猜測的欄位與送出按鈕，等待使用者操作。

6. Client 傳送猜測字串

使用者在 GUI 輸入猜測字串後，Client 透過 UDP 傳送 [Guess] 封包至 Server。

7. Server 比對字串並回傳結果

Server 對收到的猜測字串進行比對，計算幾 A 幾 B，並回傳 [Guess Reply] 封包。

8. Client 顯示結果

Client 收到回覆後，在 GUI 上顯示猜測結果，提示使用者是否猜中。

9. 猜中：Client 傳送 [USERINFO]

若猜中答案，Client 傳送 [USERINFO] 封包，內含使用者名稱、猜測次數、用時與完成時間等資訊。

10. Server 儲存成績並更新排行榜

Server 將資訊寫入 rankings.json，並依照時間排序以更新排行榜。

11. Server 回傳 [Congratulations!]

Server 回傳玩家的排名資訊給 Client，Client 顯示名次並開啟「再玩一次」與「結束遊戲」按鈕。

12. 再次遊玩流程

若 Client 點擊「再玩一次」，則傳送 [Replay] 封包，Server 重設答案，流程回到第 4 步驟。

13. 遊戲結束 / Timeout 處理

若使用者選擇「結束遊戲」，或任一端在一段時間內無操作而觸發 Timeout，Client 傳送 QUIT 封包或 Server 自動判斷 timeout，雙方皆會關閉 socket 並結束遊戲流程。

4. 心得與問題解決

這次的 UDP 作業，作業要求是只要做出文字介面能互動就可以了，不過在寫到在製作這個 GUI 的過程中，在設計遊戲流程的管控時吃了不少苦頭，必須去想這個遊戲階段哪些按鈕不應該被啟用，在哪個階段又該被啟用，整體設計起來稍微有點麻煩。

不過這次作業最重要的還是 UDP socket 通訊的設計，這次使用 UDP 實作雙向封包交換，使我更深入理解資料傳輸機制，包含封包格式、位址綁定與 UDP 的特性。

而由於是第一次設計 socket，在寫作業過程中也遇到不少的問題：

問題描述	解決方式
要怎麼樣可以持久化保存排名資料	額外儲存一個 json 檔案用以存放排名
UDP 無法確定是否已建立連線	模擬 TCP，在實際開始遊戲前，先傳送 [Connecting] 封包確認和 Server 端連線
Socket close 後無法重新綁定 port	設定 setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
Timeout 功能要如何實現	使用 threading.Timer
封包訊息內容無法區分用途	使用封包標記如 [Connecting], [Guess] 區分

透過這次作業，我更熟悉了 UDP socket 的運作，也更理解了封包運作的原理，同時也因為要連線所以更了解網路網域到底是一個什麼樣的東西。從設計、通訊協定選擇、UI 製作到錯誤處理與使用者體驗優化的開發流程，不僅提升了我的 Python 技能，也加深了我對網路程式與跨模組整合的理解。

5. 程式碼

Server 端程式碼：

```
1. import tkinter as tk
2. from tkinter import messagebox
3. import socket
4. import threading
5. import json
6. import os
7.
8. RANKING_FILE = "rankings.json"
9.
10. class ServerGUI:
11.     def __init__(self, root):
12.         # 初始化視窗基本設定
13.         self.root = root
14.         self.root.title("UDP 猜字串-Server")
15.         self.root.minsize(900, 540)
16.
```

```
17. # ===== 初始化變數 =====
18. # socket 與遊戲相關變數
19. self.server_socket = None # server socket 實體
20. self.client_address = None # client 的地址
21. self.receive_thread = None # 用於接收訊息的執行緒
22. self.answer = "" # 正確答案
23. self.answer_length = 0 # 答案長度
24. self.game_running = False # 遊戲是否正在進行
25. self.timeout_timer = None # timeout 計時器
26. self.TIMEOUT_DURATION = 120 # 2 分鐘無動作 timeout
27. self.socket_running = False # socket 是否啟動中
28. self.rankings = None # 排行榜紀錄
29.
30. # ===== IP 與 Port 設定區塊 =====
31. self.IP_frame = tk.Frame(self.root)
32. tk.Label(self.IP_frame, text="IP Address:").pack()
33. self.ip_entry = tk.Entry(self.IP_frame)
34. self.ip_entry.insert(0, "127.0.0.1")
35. self.ip_entry.pack()
36. tk.Label(self.IP_frame, text="Port: ").pack()
37. self.port_entry = tk.Entry(self.IP_frame)
38. self.port_entry.insert(0, "5000")
39. self.port_entry.pack()
40. self.start_button = tk.Button(self.IP_frame, text="啟動伺服器", command=self.start_server)
41. self.start_button.pack(pady=10)
42.
43. # ===== 訊息輸出區塊 =====
44. self.text_frame = tk.Frame(self.root)
45. self.text_frame.rowconfigure(0, weight=1)
46. self.text_frame.columnconfigure(0, weight=1)
47. self.output_text = tk.Text(self.text_frame, state="disabled")
48. self.output_text.grid(row=0, column=0, sticky="nsew")
49. self.output_text.tag_config("bold", font=("Helvetica", 10, "bold"))
50. self.output_text.tag_config("error", foreground="red")
51. self.output_text.tag_config("success", foreground="green")
52. self.output_text.tag_config("info", foreground="blue", font=("Helvetica", 10, "bold"))
53. scrollbar_x = tk.Scrollbar(self.text_frame, orient="horizontal",
   command=self.output_text.xview)
54. scrollbar_y = tk.Scrollbar(self.text_frame, orient='vertical',
   command=self.output_text.yview)
55. scrollbar_x.grid(row=1, column=0, sticky="ew")
56. scrollbar_y.grid(row=0, column=1, sticky="ns")
57. self.output_text.config(yscrollcommand=scrollbar_y.set, xscrollcommand=scrollbar_x.set)
58.
59. # ===== 答案設定區塊 =====
60. self.answer_input_frame = tk.Frame()
61. tk.Label(self.answer_input_frame, text="答案長度(EX: 4): ").pack()
62. self.answer_len_entry = tk.Entry(self.answer_input_frame)
63. self.answer_len_entry.pack()
64. tk.Label(self.answer_input_frame, text="正確答案(0-9A-F, 不重複):").pack()
65. self.answer_entry = tk.Entry(self.answer_input_frame)
66. self.answer_entry.pack()
67. self.set_answer_button = tk.Button(self.answer_input_frame, text="設定答案",
   command=self.set_answer, state='disabled')
```

```

68. self.set_answer_button.pack(pady=5)
69.
70. # GUI 整體版面配置
71. self.root.rowconfigure(2, weight=1)
72. self.root.columnconfigure(0, weight=1)
73. self.IP_frame.grid(column=0, row=0)
74. self.answer_input_frame.grid(column=0, row=1, sticky="nsew")
75. self.text_frame.grid(column=0, row=2, sticky="nsew", padx=10, pady=10)
76.
77. # 載入排行榜資料
78. self.load_rankings()
79.
80. def start_server(self):
81.     # 啟動伺服器按鈕邏輯
82.     ip = self.ip_entry.get()
83.     port = self.port_entry.get()
84.
85.     if not ip or not port:
86.         self.modify_output_text("[Error]: 輸入錯誤，請輸入 IP 和 Port\n", "error")
87.         return
88.
89.     try:
90.         port = int(port)
91.     except ValueError:
92.         self.modify_output_text("[Error]: 請確認 Port 輸入為數字\n", "error")
93.         return
94.
95.     try:
96.         self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
97.         self.server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
98.         self.server_socket.bind((ip, port))
99.     except Exception as e:
100.        self.modify_output_text(f"[Error]: Socket 繁定失敗: {e}\n", "error")
101.    return
102.
103. self.socket_running = True
104. self.set_answer_button.config(state="normal")
105. self.modify_output_text(f"[Info]: 伺服器已啟動，監聽 {ip}:{port}\n", "info")
106. self.receive_thread = threading.Thread(target=self.receive_messages, daemon=True)
107. self.receive_thread.start()
108. self.reset_timeout_timer()
109.
110. def check_client_guess(self, guess: str):
111.     # 驗證 client 猜測的結果
112.     if len(guess) != self.answer_length:
113.         return f"[Error]: 格式錯誤，請輸入{self.answer_length}位數字"
114.     A = sum(a == b for a, b in zip(guess, self.answer))
115.     B = sum(min(guess.count(x), self.answer.count(x)) for x in set(guess)) - A
116.     if A == self.answer_length:
117.         return f"恭喜猜對了! {A}A{B}B"
118.     return f"{A}A{B}B"
119.
120. def receive_messages(self):
121.     # 接收 client 傳送的封包，並進行處理

```

```
122.while self.socket_running:
123.try:
124.if self.server_socket:
125.data, addr = self.server_socket.recvfrom(1024)
126.self.client_address = addr
127.self.reset_timeout_timer()
128.msg = data.decode()
129.self.modify_output_text(f"[UDP]: 來自 {addr} 的訊息: {msg}\n")
130.
131.# 判斷各類封包種類做處理
132.if msg.startswith("[Connecting]:"):
133.self.modify_output_text(f"[Success]: 收到來自{addr}的連接訊息\n", "success")
134.self.client_address = addr
135.self.game_running = True
136.self.set_answer_button.config(state='normal')
137.self.server_socket.sendto("[Ack]: Server 已啟動".encode(), addr)
138.elif msg.startswith("[Guess]:"):
139.if not self.answer:
140.server_reply = "[Error]: 驚服器尚未設定好答案"
141.else:
142.guess = msg.split(":")[1].strip()
143.server_reply = self.check_client_guess(guess)
144.self.modify_output_text(f"[Info]: 來自{addr}的猜測: {guess}→{server_reply}\n", "info")
145.self.server_socket.sendto(f"[Guess Reply]: {server_reply}".encode(), self.client_address)
146.elif msg.startswith("[USERINFO]"):
147.data = msg.split("->")[1].split(",")
148.username, guess_count, duration, finish_time_str = data[0], int(data[1]), float(data[2]),
data[3]
149.self.add_user_rankings(username, guess_count, duration, finish_time_str)
150.rank = self.get_rank(username, duration, finish_time_str)
151.self.server_socket.sendto(f"[Congratulations!]: {username}! 你是第 {rank}名!\n".encode(),
self.client_address)
152.self.show_rankings(highlight_username=username, highlight_time=duration,
highlight_finish=finish_time_str)
153.elif msg.startswith("[Replay]:"):
154.self.modify_output_text(f"[Info]: client 端要求重新開始遊戲, 請再次設定答案\n", "info")
155.self.answer = ""
156.self.answer_length = 0
157.self.set_answer_button.config(state='normal')
158.self.client_address = addr
159.elif msg.startswith("[Timeout]:"):
160.self.modify_output_text(f"{msg}\n", "error")
161.self.answer = ""
162.self.answer_length = 0
163.self.client_address = None
164.self.socket_running = False
165.self.game_running = False
166.self.set_answer_button.config(state="normal")
167.self.server_socket.close()
168.elif msg == "QUIT":
169.self.stop_server()
170.exception OSError:
171.break
172.exception Exception as e:
```

```
173.self.modify_output_text(f"[Error]: {e}", "error")
174.continue
175.
176.if not self.socket_running:
177.self.server_socket.close()
178.self.server_socket = None
179.
180.def stop_server(self):
181.# 關閉 socket 與 GUI
182.self.socket_running = False
183.self.game_running = False
184.if self.server_socket:
185.self.server_socket.close()
186.self.server_socket = None
187.root.destroy()
188.
189.def set_answer(self):
190.# 設定正確答案邏輯
191.self.reset_timeout_timer()
192.answer = self.answer_entry.get().upper()
193.length = 0
194.
195.if not self.game_running:
196.self.modify_output_text("[Info]: 目前無已連接之 client, 請再次確認是否已有連線\n", "info")
197.self.set_answer_button.config(state="disabled")
198.return
199.
200.try:
201.length = int(self.answer_len_entry.get())
202.except ValueError:
203.self.modify_output_text("[Error]: 請確認答案長度為一整數\n")
204.return
205.
206.allowed_input_char = set("0123456789ABCDEF")
207.if len(answer) != length:
208.self.modify_output_text(f"[Error]: 答案長度應為 {length} 位\n", "error")
209.return
210.if any(c not in allowed_input_char for c in answer):
211.self.modify_output_text("[Error]: 請確認答案僅包含 0~9 或 A~F\n", "error")
212.return
213.if len(set(answer)) != length:
214.self.modify_output_text("[Error]: 答案中有重複字元)\n", "error")
215.return
216.
217.self.answer = answer
218.self.answer_length = length
219.self.set_answer_button.config(state='disabled')
220.self.modify_output_text(f"[Success]: 正確答案已設定為: {answer}\n", "success")
221.
222.if self.client_address:
223.self.server_socket.sendto(f"[Ready]: {length}, 開始猜數字遊戲, 請輸入{length}個數字/文字
".encode(), self.client_address)
224.
225.def modify_output_text(self, text, tag=None):
```

```
226.# 輸出訊息至訊息區域，支援標籤樣式
227.self.output_text.config(state="normal")
228.if tag:
229.self.output_text.insert(tk.END, text, tag)
230.else:
231.self.output_text.insert(tk.END, text)
232.self.output_text.config(state="disabled")
233.self.output_text.see(tk.END)
234.
235.def load_rankings(self):
236.# 載入歷史排行榜資料
237.if os.path.exists(RANKING_FILE):
238.with open(RANKING_FILE, "r") as f:
239.try:
240.self.rankings = json.load(f)
241.except json.JSONDecodeError:
242.self.rankings = []
243.else:
244.self.rankings = []
245.
246.def save_rankings(self):
247.# 儲存排行榜資料至 JSON 檔
248.with open(RANKING_FILE, "w") as f:
249.json.dump(self.rankings, f, indent=2)
250.
251.def add_user_rankings(self, username, guess_count, duration, finish_time_str):
252.# 將猜對使用者資料加入排行榜並排序
253.self.rankings.append({
254."name": username,
255."guesses": guess_count,
256."time": duration,
257."finish_time": finish_time_str
258.})
259.self.rankings.sort(key=lambda x: x["time"])
260.self.save_rankings()
261.
262.def get_rank(self, username, time_used, finish_time_str):
263.# 查詢該使用者的名次
264.for i, rec in enumerate(self.rankings):
265.if rec["name"] == username and rec["time"] == time_used and rec["finish_time"] ==
finish_time_str:
266.return i + 1
267.return -1
268.
269.def show_rankings(self, highlight_username=None, highlight_time=None,
highlight_finish=None):
270.# 顯示排行榜內容，並標示本次紀錄
271.self.modify_output_text("[Ranking]:\n", "bold")
272.for i, rec in enumerate(self.rankings, 1):
273.is_highlight = (
274.rec["name"] == highlight_username and
275.rec["time"] == highlight_time and
276.rec["finish_time"] == highlight_finish
277.)
```

```

278.line = f"{i}. {rec['name']} - {rec['guesses']} 次, {rec['time']} 秒, 完成時間:
    {rec['finish_time']}\n"
279.if is_highlight:
280.self.output_text.config(state="normal")
281.self.modify_output_text(line, "bold")
282.self.output_text.config(state="disabled")
283.else:
284.self.modify_output_text(line)
285.
286.def reset_timeout_timer(self):
287.# 重置 timeout 計時器
288.print("timeout reset")
289.if self.timeout_timer:
290.self.timeout_timer.cancel()
291.self.timeout_timer = threading.Timer(self.TIMEOUT_DURATION, self.handle_timeout)
292.self.timeout_timer.start()
293.
294.def handle_timeout(self):
295.# timeout 處理邏輯
296.self.modify_output_text(f"[Timeout]: {self.TIMEOUT_DURATION}秒內沒有互動，遊戲已自動結
    束!\n", "error")
297.if self.client_address:
298.try:
299.self.server_socket.sendto(f"[Timeout]: Server 已閒置過久，自動中止遊戲".encode(),
    self.client_address)
300.except Exception as e:
301.self.modify_output_text(f"[Error]: 傳送 timeout 通知失敗: {e}\n", "error")
302.finally:
303.self.client_address = None
304.if self.server_socket:
305.try:
306.self.server_socket.close()
307.self.modify_output_text("[Info]: 伺服器 socket 已關閉\n", "info")
308.except Exception as e:
309.self.modify_output_text(f"[Error]: 關閉 socket 時發生錯誤: {e}\n", "error")
310.finally:
311.self.server_socket = None
312.
313.self.socket_running = False
314.self.game_running = False
315.self.answer = ""
316.self.answer_length = 0
317.self.set_answer_button.config(state="disabled")
318.
319.if __name__ == "__main__":
320.root = tk.Tk()
321.app = ServerGUI(root)
322.root.protocol("WM_DELETE_WINDOW", app.stop_server)
323.root.mainloop()

```

Client 端程式碼:

```

1. import tkinter as tk
2. import socket
3. import threading

```

```
4. from datetime import datetime
5.
6. class ClientGUI:
7.     def __init__(self, root):
8.         """初始化 GUI 與變數"""
9.         # socket 連線參數
10.        self.socket = None
11.        self.server_address = None
12.        self.client_address = ('0.0.0.0', 12345)
13.        self.receive_thread = None
14.
15.        # 猜數字相關參數
16.        self.ready_to_guess = None
17.        self.answer_length = 0
18.        self.username = None
19.        self.guess_count = 0
20.        self.start_time = None
21.
22.        # timeout 計時器相關
23.        self.timeout_timer = None
24.        self.TIMEOUT_DURATION = 120 # 玩家若 120 秒內沒動作就 timeout
25.        self.game_running = False
26.
27.        # Tkinter GUI 初始化
28.        self.root = root
29.        self.root.title("UDP 猜字串-Client")
30.        self.root.minsize(900, 540)
31.
32.        # === 連線資訊輸入區 ===
33.        self.input_entry_frame = tk.Frame(self.root)
34.        tk.Label(self.input_entry_frame, text="使用者名稱: ").pack()
35.        self.name_entry = tk.Entry(self.input_entry_frame)
36.        self.name_entry.pack()
37.
38.        tk.Label(self.input_entry_frame, text="Server IP: ").pack()
39.        self.ip_entry = tk.Entry(self.input_entry_frame)
40.        self.ip_entry.pack()
41.
42.        tk.Label(self.input_entry_frame, text="伺服器 Port: ").pack()
43.        self.port_entry = tk.Entry(self.input_entry_frame)
44.        self.port_entry.pack()
45.
46.        self.start_button = tk.Button(self.input_entry_frame, text="連線並開始遊戲",
47.                                     command=self.start_game)
47.        self.start_button.pack(pady=5)
48.
49.        # === 訊息輸出區 (帶捲軸) ===
50.        self.text_frame = tk.Frame(self.root)
51.        self.text_frame.rowconfigure(0, weight=1)
52.        self.text_frame.columnconfigure(0, weight=1)
53.
54.        self.output_text = tk.Text(self.text_frame, state="disabled")
55.        self.output_text.grid(row=0, column=0, sticky="nsew")
56.        self.output_text.tag_config("bold", font=("Helvetica", 10, "bold"))
```

```
57. self.output_text.tag_config("error", foreground="red")
58. self.output_text.tag_config("success", foreground="green")
59. self.output_text.tag_config("info", foreground="blue", font=("Helvetica", 10, "bold"))
60.
61. scrollbar_x = tk.Scrollbar(self.text_frame, orient="horizontal",
   command=self.output_text.xview)
62. scrollbar_y = tk.Scrollbar(self.text_frame, orient='vertical',
   command=self.output_text.yview)
63. scrollbar_x.grid(row=1, column=0, sticky="ew")
64. scrollbar_y.grid(row=0, column=1, sticky="ns")
65. self.output_text.config(yscrollcommand=scrollbar_y.set, xscrollcommand=scrollbar_x.set)
66.
67. # === 猜測輸入區 ===
68. self.guess_frame = tk.Frame(self.root)
69. tk.Label(self.guess_frame, text="輸入猜測字串: ").pack()
70. self.guess_entry = tk.Entry(self.guess_frame)
71. self.guess_entry.pack()
72.
73. self.guess_button = tk.Button(self.guess_frame, text="送出猜測", command=self.send_guess,
   state='disabled')
74. self.guess_button.pack(pady=5)
75.
76. self.replay_button = tk.Button(self.guess_frame, text="再玩一次",
   command=self.replay_game, state="disabled")
77. self.replay_button.pack()
78.
79. self.quit_button = tk.Button(self.guess_frame, text="結束遊戲", command=self.quit_game,
   state='disabled')
80. self.quit_button.pack()
81.
82. # === GUI 版面配置 ===
83. self.root.rowconfigure(1, weight=1)
84. self.root.columnconfigure(0, weight=1)
85. self.input_entry_frame.grid(row=0, column=0)
86. self.text_frame.grid(row=1, column=0, sticky="nsew", padx=10, pady=10)
87. self.guess_frame.grid(row=2, column=0)
88.
89. def quit_game(self):
90.     """結束遊戲，傳送 QUIT 給伺服器並關閉視窗"""
91.     try:
92.         quit_message = "QUIT"
93.         self.socket.sendto(quit_message.encode(), self.server_address)
94.     except Exception as e:
95.         print(f"Error sending quit message: {e}")
96.     finally:
97.         self.socket.close()
98.         self.root.destroy()
99.
100. def replay_game(self):
101.     """重新開始遊戲流程"""
102.     self.reset_timeout_timer()
103.     if self.socket:
104.         self.socket.sendto(f"[Replay]:{self.client_address}".encode(), self.server_address)
105.         self.replay_button.config(state="disabled")
```

```
106.self.quit_button.config(state="disabled")
107.self.guess_entry.delete(0, tk.END)
108.self.modify_output_text("[Replay Requesting]已請求重新開始，請稍候 Server 設定新答案...\n",
    "info")
109.
110.def test_server_connection(self, server_address):
111."""測試連線用，發出測試封包並回傳是否成功"""
112.ip = server_address[0]
113.port = server_address[1]
114.try:
115.self.modify_output_text(f"[Info]: 正在確認連線
    {self.client_address[0]}:{self.client_address[1]}->{ip}:{port}\n")
116.self.socket.sendto(f"[Connecting]: client->{ip}:{port}".encode(), self.server_address)
117.data, addr = self.socket.recvfrom(1024)
118.self.modify_output_text(f"[Connection Success!]: 伺服器回應為: {data.decode()}\n",
    "success")
119.return True
120.exception socket.timeout:
121.return False
122.exception Exception as e:
123.self.modify_output_text(f"[Error]: {e}\n", "error")
124.return False
125.finally:
126.self.socket.settimeout(None)
127.
128.def start_game(self):
129."""啟動遊戲流程：建立 socket，發出初始訊息並啟動接收執行緒"""
130.ip = self.ip_entry.get()
131.port = self.port_entry.get()
132.username = self.name_entry.get()
133.
134.if not ip or not port or not username:
135.self.modify_output_text("[Error]: 請填寫完整 IP、Port 和 username\n", "error")
136.return
137.
138.try:
139.port = int(port)
140.exception ValueError:
141.self.modify_output_text("[Error]: 請確認填寫之 port 為整數數字\n", "error")
142.return
143.
144.self.server_address = (ip, port)
145.self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
146.self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
147.self.socket.bind(self.client_address)
148.self.socket.settimeout(3)
149.
150.if not self.test_server_connection(self.server_address):
151.self.modify_output_text("[Error]: 連線失敗，請確認伺服器是否啟動\n", "error")
152.self.socket.close()
153.return
154.
155.self.modify_output_text(f"[Success]: 已連線到伺服器{ip}:{port}，等待 Server 設定答案...\n",
    "success")
```

```

156.self.game_running = True
157.self.username = username
158.self.guess_count = 0
159.self.start_button.config(state="disabled")
160.self.receive_thread = threading.Thread(target=self.receive_response, daemon=True)
161.self.receive_thread.start()
162.self.reset_timeout_timer()
163.
164.def send_guess(self):
165.""""送出猜測值至伺服器"""
166.guess = self.guess_entry.get().strip().upper()
167.self.reset_timeout_timer()
168.
169.if not self.ready_to_guess:
170.self.modify_output_text("[Waiting...]: 等待 Server 設定答案\n")
171.return
172.
173.if any(c not in "0123456789ABCDEF" for c in guess):
174.self.modify_output_text("[Error]: 請只輸入 0-9 或 A-F 的字元\n", "error")
175.return
176.if len(set(guess)) != len(guess):
177.self.modify_output_text("[Error]: 請確認輸入字元皆不重複\n", "error")
178.return
179.if len(guess) != self.answer_length:
180.self.modify_output_text(f"[Error]: 請輸入{self.answer_length}個字元\n", "error")
181.return
182.
183.self.guess_count += 1
184.try:
185.self.socket.sendto(f"[Guess]: {guess}".encode(), self.server_address)
186.self.modify_output_text(f"[Info]: 已送出猜測({guess})\n", "info")
187.exception as e:
188.self.modify_output_text(f"[Error]: 傳送失敗({e})\n", "error")
189.
190.def receive_response(self):
191.""""接收伺服器的所有訊息並處理各種回應"""
192.while self.game_running:
193.try:
194.data, _ = self.socket.recvfrom(1024)
195.response = data.decode()
196.self.reset_timeout_timer()
197.
198.if response.startswith("[Ready]:"):
199.self.ready_to_guess = True
200.self.answer_length = int(response.split(":")[1].split(", ")[0])
201.self.modify_output_text(f"[Info]: Server 已設定答案(長度{self.answer_length}), 可開始猜測\n", "info")
202.self.guess_button.config(state='normal')
203.self.start_time = datetime.now()
204.self.guess_count = 0
205.elif response.startswith("[Guess Reply]:") and "恭喜猜對" in response:
206.duration = (datetime.now() - self.start_time).total_seconds()
207.finish_time_str = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

```

```
208.self.modify_output_text(f"[Game Finish!]: 共猜{self.guess_count}次, 用時{duration:.2f}秒\n", "bold")
209.self.modify_output_text("若要再次遊玩請點下方「再玩一次」或按「結束遊戲」離開\n", "success")
210.self.guess_button.config(state="disabled")
211.self.replay_button.config(state="normal")
212.self.quit_button.config(state="normal")
213 userinfo_msg = f"[USERINFO]->{self.username},{self.guess_count},{duration},{finish_time_str}"
214.self.socket.sendto(userinfo_msg.encode(), self.server_address)
215.elif response.startswith("[Guess Reply]"):
216.self.modify_output_text(f"{response}\n", 'bold')
217.elif response.startswith("[Congratulations!]"):
218.txt = response.split(":")[1].strip()
219.self.modify_output_text(f"{txt}\n", 'bold')
220.elif response.startswith("[Timeout]:"):
221.self.modify_output_text(f"{response}\n", "error")
222.self.guess_button.config(state="disabled")
223.self.replay_button.config(state="disabled")
224.self.quit_button.config(state='disabled')
225.self.game_running = False
226.self.start_button.config(state="normal")
227.self.modify_output_text("[Info]: 客戶端 socket 已關閉\n", "info")
228.self.socket.close()
229.exception OSError:
230.break
231.exception Exception:
232.continue
233.
234.def modify_output_text(self, text, tag=None):
235."""更新輸出訊息框"""
236.self.output_text.config(state="normal")
237.if tag:
238.self.output_text.insert(tk.END, text, tag)
239.else:
240.self.output_text.insert(tk.END, text)
241.self.output_text.config(state="disabled")
242.self.output_text.see(tk.END)
243.
244.def reset_timeout_timer(self):
245."""重設閒置計時器 (用於遊戲自動 timeout) """
246.if self.timeout_timer:
247.self.timeout_timer.cancel()
248.self.timeout_timer = threading.Timer(self.TIMEOUT_DURATION, self.handle_timeout)
249.self.timeout_timer.start()
250.
251.def handle_timeout(self):
252."""處理 timeout (玩家太久沒動作自動結束) """
253.self.modify_output_text(f"[Timeout]: {self.TIMEOUT_DURATION}秒內無操作, 遊戲自動結束\n",
"error")
254.self.game_running = False
255.try:
256.self.socket.sendto("[Timeout]: Client 已閒置過久, 自動中止遊戲".encode(),
self.server_address)
257.exception Exception as e:
```

```
258.self.modify_output_text(f"[Error]: 傳送 timeout 通知失敗: {e}\n", "error")
259.if self.socket:
260.try:
261.self.socket.close()
262.self.modify_output_text("[Info]: 客戶端 socket 已關閉\n", "info")
263.exception as e:
264.self.modify_output_text(f"[Error]: 關閉 socket 時錯誤: {e}\n", "error")
265.finally:
266.self.socket = None
267.
268.# 重設所有狀態
269.self.answer_length = 0
270.self.guess_count = 0
271.self.server_address = None
272.self.start_button.config(state="normal")
273.self.guess_button.config(state="disabled")
274.self.replay_button.config(state="disabled")
275.self.quit_button.config(state='disabled')
276.
277.# 啟動 GUI
278.if __name__ == "__main__":
279.root = tk.Tk()
280.app = ClientGUI(root)
281.root.mainloop()
```