Connect Four

1. Introduzione

Questo progetto è un'implementazione del gioco "Forza Quattro" in Python. È un'applicazione client-server che permette agli utenti di sfidare altri giocatori.

2. Requisiti e funzionalità

Il server, quando contattato, permette ai client di: registrarsi con un nome univoco e giocare contro un altro giocatore (se presente). Durante la partita, a turno, ogni giocatore manda la propria mossa e aspetta quella dell'avversario, fino alla fine del gioco.

Tutti i messaggi scambiati tra client e server vengono inoltre memorizzati in file di log (contenuti nell'apposita cartella "*log*").

2.A. Server

Come prima cosa bisogna far partire il server:

```
>> python .\server.py
Server is running and listening. To shutdown press CRTL+C ...
```

Questo script si mette in ascolto sull'indirizzo locale 127.0.0.1 sulla porta 3000 e aspetta nuove connessioni. Il server salva ogni nuova connessione che gli arriva, ne legge i messaggi ricevuti e agisce di conseguenza, come verrà spiegato nel seguito.

2.B. Client

Ogni utente che vuole giocare, deve far partire l'interfaccia client:

```
>> python .\client.py
Choose a name (only letters and numbers are allowed. Maximum supported length is 20) >>>
```

Verrà subito chiesto di scegliere un nome. I controlli fatti sul nome (lato client) sono: lunghezza massima di 20 caratteri e presenza di sole lettere e numeri. Lo script continuerà a chiedere di inserire il nome finché esso risulterà valido.

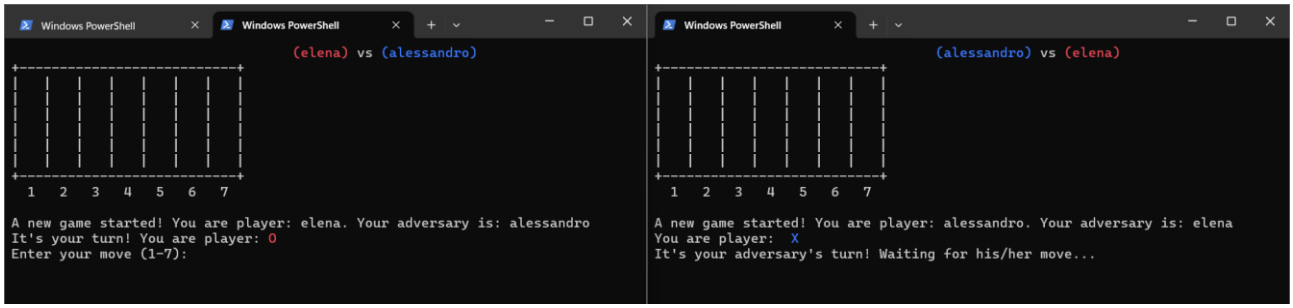
2.C. Messaggi scambiati tra client e server

Una volta inserito un nome valido, questo viene mandato al server. Il server controlla poi che il nome sia univoco. Se non è univoco, si comunica al client di richiedere il nome all'utente. Se invece è univoco, il client viene direttamente memorizzato dal server, che cercherà un avversario:

- Se non dovessero esserci avversari in attesa di giocare, si comunica al client che dovrà attendere che un nuovo utente decida di partecipare:

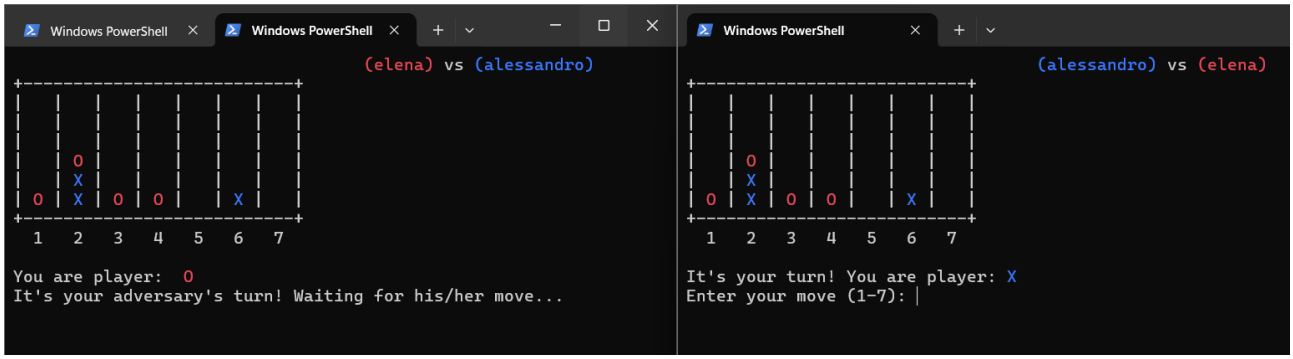
```
>> python .\client.py
Choose a name (only letters and numbers are allowed. Maximum supported length is 20) >>> elena
Waiting for a new player to join the game...
```

- Se un avversario è presente, i due giocatori vengono abbinati dal server e si comunica ad entrambi l'inizio di una nuova partita:

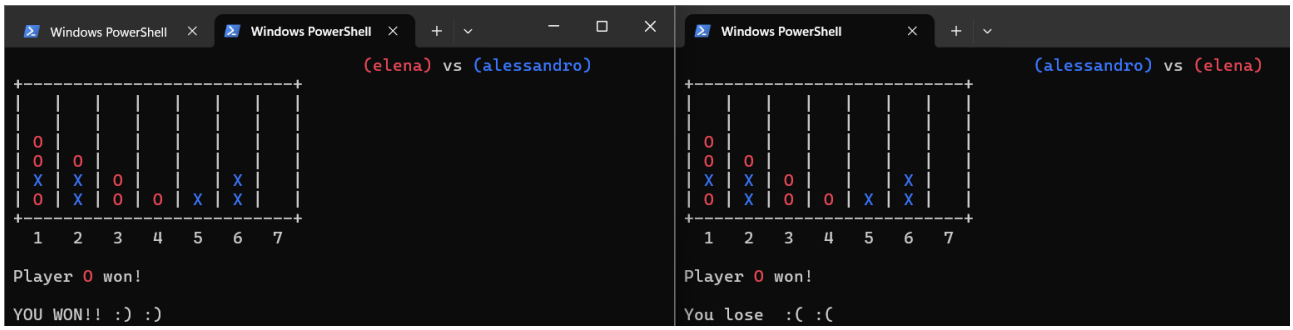


(Il terminale a sinistra è del primo giocatore; quello a destra del secondo)

Il primo turno della partita è del primo giocatore che ha deciso di giocare (colui che per primo si è registrato e che ha aspettato l'inizio della partita per più tempo). L'altro giocatore si metterà in attesa della mossa dell'avversario. Una volta scelta la propria mossa, il client del primo giocatore la comunica al server, che la inoltrerà all'avversario. Il primo giocatore si mette ora in attesa del turno avversario, mentre il secondo, ricevendo la mossa avversaria, potrà stavolta scegliere la propria mossa.



La partita prosegue a turni così finché uno dei giocatori vince o abbandona il gioco.



Una volta finito il gioco, le connessioni tra il server e i clients dei due giocatori vengono chiuse.

2.D. File di log

Durante l'esecuzione sono stati memorizzati in appositi file di log tutti i messaggi in entrata e in uscita scambiati tra i client e il server. Nei file di log vengono messe due colonne: nella prima ci sono informazioni sul socket (in particolare la porta del client), mentre nella seconda il messaggio stesso (con in più una scritta "*IN <*" oppure "*OUT >*" a seconda che sia in entrata o uscita dal server). Per l'esecuzione d'esempio mostrata, il file di log contiene:

- Registrazione di "*elena*" (sulla porta 57641): il server riceve la richiesta di registrazione di un client e risponde con "ok" perché il nome scelto è univoco e quindi la registrazione è andata a buon fine:

```
('127.0.0.1', 57641)>        IN  <   [new_player] elena
('127.0.0.1', 57641)>        OUT >   [ok]
```

- Il server ha poi mandato un messaggio a *elena* dicendo di aspettare un avversario:

```
('127.0.0.1', 57641)>        OUT >   [wait_player]
```

- Segue poi la registrazione di "*alessandro*" (su una porta diversa):

```
('127.0.0.1', 57642)>        IN  <   [new_player] alessandro
('127.0.0.1', 57642)>        OUT >   [ok]
```

- Il server capisce che ci sono due giocatori che vogliono giocare e li associa:
  - Comunica quindi a *elena* che è iniziata una nuova partita contro *alessandro.* Il primo turno è di *elena* dato che per prima era in attesa di giocare (viene quindi messo uno 0 nel messaggio):

```
('127.0.0.1', 57641)>        OUT >   [new_game] 0 alessandro
```

  - Comunica anche a *alessandro* che sfiderà *elena.* Viene messo un 1 nel messaggio per riferire che stavolta è il turno dell'avversario:

```
('127.0.0.1', 57642)>        OUT >   [new_game] 1 elena
```

Nel messaggio viene comunicato il nome dell'avversario, in modo che possa essere stampato in alto a destra nel terminale dei client (come negli screenshot d'esempio).

- I due giocatori fanno poi a turno le loro mosse, che vengono inoltrate all'avversario dal server (c'è un messaggio in entrata e uno in uscita per ogni mossa):

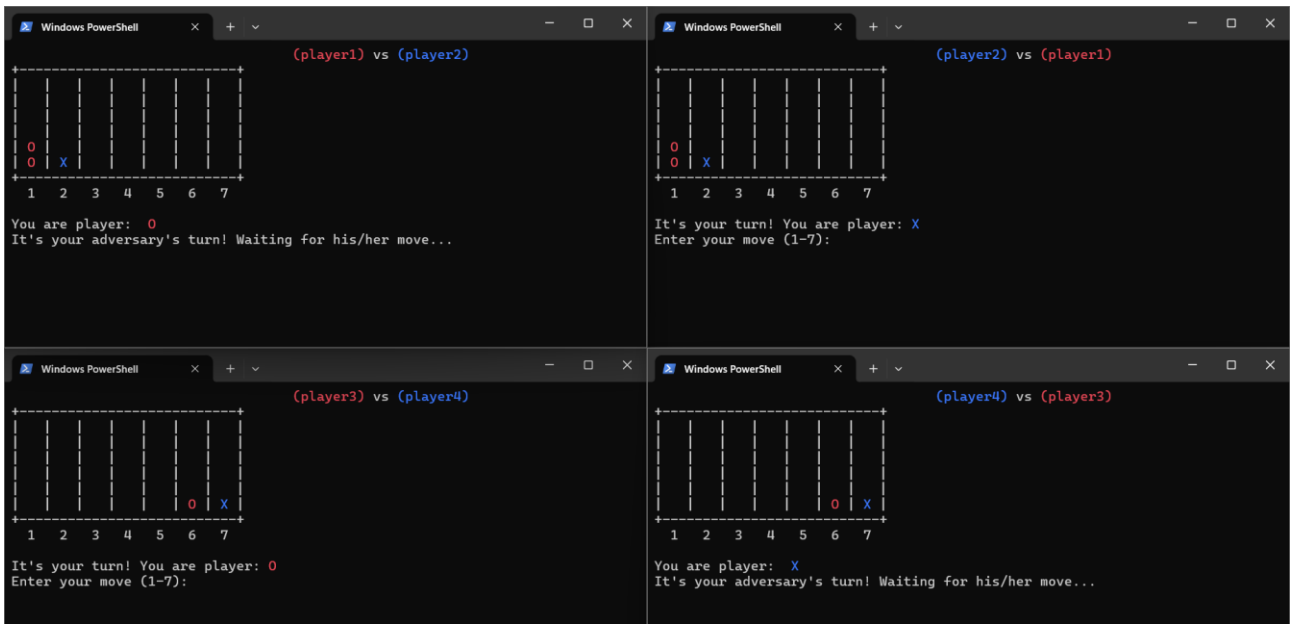
```
('127.0.0.1', 57641)>        IN  <   [new_move] 3 elena
('127.0.0.1', 57642)>        OUT >   [new_move] 3 elena
('127.0.0.1', 57642)>        IN  <   [new_move] 2 alessandro
('127.0.0.1', 57641)>        OUT >   [new_move] 2 alessandro
```

- Nell'ultimo turno, il giocatore vincente manda la sua ultima mossa. Il server la riceve e la inoltra all'avversario. Il vincitore manda infine un messaggio di "fine partita" per comunicare al server di chiudere le connessioni di entrambi i giocatori:

```
('127.0.0.1', 57641)>        IN  <   [end_game] elena
```

2.E. Più partite contemporaneamente

La comunicazione appena mostrata era un esempio tra soli due giocatori. Ovviamente sono possibili più partite in contemporanea:

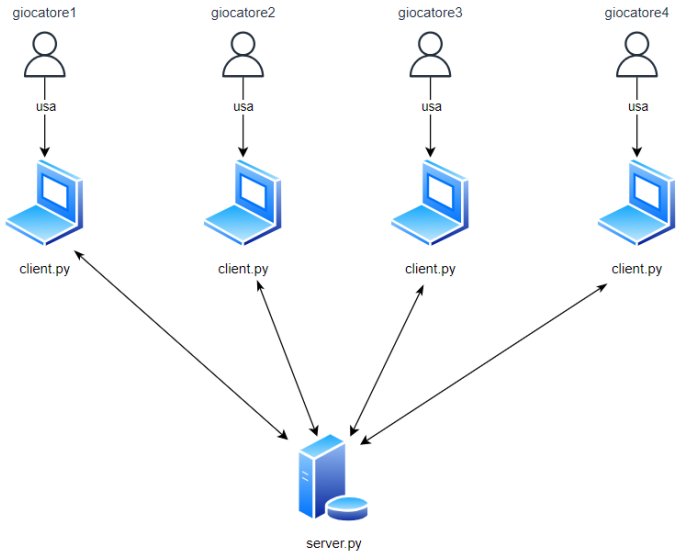


I primi due terminali in alto rappresentano la partita tra player1 e player2 (il terminale a sinistra è di player1, quello a destra di player2). I due terminali in basso sono invece di player3 e player4.

Come si vede dai due tabelloni del gioco, player1 e player2 vedono lo stesso tabellone, ma non vedono quello di player3 e player4 (e viceversa).

3. Architettura

L'architettura dell'applicazione è di tipo client-server. L'utente usa l'interfaccia fornita da client.py per comunicare con il server. Il client contatta il server, il quale la prima volta accetta e memorizza la connessione. Le volte successive il server si mette in ascolto dei messaggi che gli arrivano dalla connessione precedentemente memorizzata e risponde come descritto precedentemente. La connessione tra i due client (i giocatori) e il server viene poi chiusa al termine della partita.



4. Implementazione

L'applicazione è stata interamente creata con Python 3.11. L'implementazione client-server è stata fatta con il pacchetto "socket" fornito da python.

4.A. Server

Il server crea un socket in ascolto su localhost:3000

```
host = '127.0.0.1'
port = 3000

# Creation of the listening socket
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((host, port))
server.listen()
```

In un while True ci si mette in ascolto delle connessioni dei clients:

```
client, address = server.accept()
thread = threading.Thread(target=handle_client, args=(client,))
thread.start()
```

E si fa partire un thread (`handle_client`) che si mette in ascolto dei messaggi in arrivo da tale client. In questo while true si controlla anche se il server viene chiuso (premendo CTRL+C), e in tal caso si chiudono le connessioni con tutti i clients.

In handle_client si leggono i messaggi arrivati e si agisce come descritto in precedenza:

- se arriva una registrazione si controlla che il nome sia univoco ed eventualmente si salva la connessone e si fa partire una nuova partita;
- se arriva una "mossa" di un giocatore, la si inoltra al client avversario
- se arriva il messaggio di fine partita, si chiude la connessione con entrambi i giocatori.
- se arriva un messaggio di un giocatore che ha abbandonato la partita, lo si comunica all'avversario e poi si chiude la connessione.

Per la gestione degli utenti, ho creato una classe apposita "Player" e ho creato tutte le funzioni necessarie per aggiungere un giocatore, abbinare i giocatori, cercare un giocatore senza avversario, …

4.B. Client

Il client, per prima cosa tenta la connessione con il server:

```
try:
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(('127.0.0.1', 3000))
except ConnectionRefusedError:
    print("Server is not running")
    exit(1)
```

Se la connessione va a buon fine, lo script chiede poi all'utente il proprio nome e lo manda al server (con le regole spiegate in precedenza). Dualmente, come spiegato prima:

- Si aspetta poi la risposta dal server (di inizio partita o di attesa dell'avversario).
- Quando arriva il messaggio di inizio partita, si stampa il tabellone vuoto e si iniziano i turni.
- In caso sia il turno dell'utente, si chiede la mossa e la si comunica al server. In caso la mossa fatta sia quella vincente, si manda un messaggio aggiuntivo al server di "fine partita", viene stampato a video "Hai vinto" e la partita finisce.
- In caso arrivi un messaggio con una mossa avversaria, si ristampa il tabellone aggiornato e si prosegue col turno del giocatore. In caso la mossa arrivata sia l'ultima mossa vincente dell'avversario, si stampa "Hai perso" e la partita finisce.
- Si controlla se l'utente ha abbandonato la partita (premendo CTRL+C). In questo caso lo si comunica al server, si chiude la connessione e si termina. Dualmente, se arriva un messaggio dal server in cui si comunica l'abbandono dell'avversario, lo si comunica all'utente e la partita finisce.

Per la gestione della partita ho creato un'apposita classe "ConnectFour" in cui ho creato tutte le funzioni necessarie per la gestione del gioco in sé (memorizzazione del tabellone, controlli che una mossa sia valida, controllo che la partita sia finita, …).

5. Conclusioni

Per concludere, è stato molto interessante implementare una semplice applicazione client-server come questa. Ho pensato anche che, come sviluppo futuro, sarebbe bello dare la possibilità agli utenti di sfidare un amico, chiedendo quindi all'utente il nome dell'avversario da sfidare, dato che attualmente è il server che abbina i giocatori "a due a due" in base all'ordine della registrazione.