COMMUNICATION PROTOCOL **IS23-AM05**

Il seguente documento è stato scritto al fine di spiegare ed illustrare il protocollo di comunicazione utilizzato dai componenti del client e del server dell'applicazione.

Nel seguito verranno affrontati i seguenti aspetti:

1. Formato dei messaggi JSON scambiati da questi ultimi;
2. Scelte implementative per le chiamate a metodi remoti (tra client e server).

NOTA1.0:
la gestione delle chiamate da remoto è stata pensata con il fine di far eseguire la medesima funzionalità al ricevente (che sia client o server), di conseguenza per ognuna di esse verranno illustrate graficamente le differenze in termini di implementazione, tra la comunicazione tramite Java RMI e la comunicazione tramite Socket TCP.

1. Formato dei messaggi JSON

Ci è sembrato naturale partire dal formato dei messaggi JSON scambiati in quanto questi sono stati pensati per avere la medesima forma per ogni metodo invocato.

Come abbiamo detto precedentemente nella NOTA1.0 le funzionalità eseguite dal componente che riceve la chiamata devono essere indipendenti dal tipo di comunicazione, di conseguenza il messaggio JSON sarà composto dai seguenti campi:

1. "method": //metodo che si intende chiamare nella macchina remota (String)
2. "param1": //primo parametro del metodo che si intende invocare(Object)
3. "param2": //secondo parametro del metodo che si intende invocare(Object)
4. "param3": // secondo parametro del metodo che si intende invocare(Object)
5. Ecc.

method	param1	param2	param3	…….

Chiariamo il concetto con un esempio:
Ci troviamo nel client e dobbiamo eseguire il metodo login(String nickname, Object client, ConnectionTypeEnum connectionType). Se questo viene eseguito dal RMIClient allora semplicemente lo eseguo invocando lo stub del server ed inserendo i rispettivi parametri. Se invece il metodo è eseguito dal SocketClient allora le cose cambiano. Per fare in modo che sul server venga eseguito il medesimo metodo allora costruiamo un oggetto JSON in questo modo:

- "method" : "login"

- “param1” : (String)nickname
- “param2” : this (riferimento al client corrente)
- “param3” : ConnectionTypeEnum.SOCKET

In questo modo il server riceverà l’oggetto e dopo avere effettuato il parsing per ogni campo potrà invocare il medesimo metodo che si sarebbe invocato tramite Java RMI. Ciò rende la logica di gioco pressochè indipendente dal tipo di comunicazione che si intende utilizzare.

NOTA2.0:

Abbiamo usato l’espressione “pressochè indipendente” in quanto se viene invocato un metodo che a sua volta richiede l’invocazione di un altro metodo remoto allora la gestione risulta leggermente diversa.

NOTA2.1:

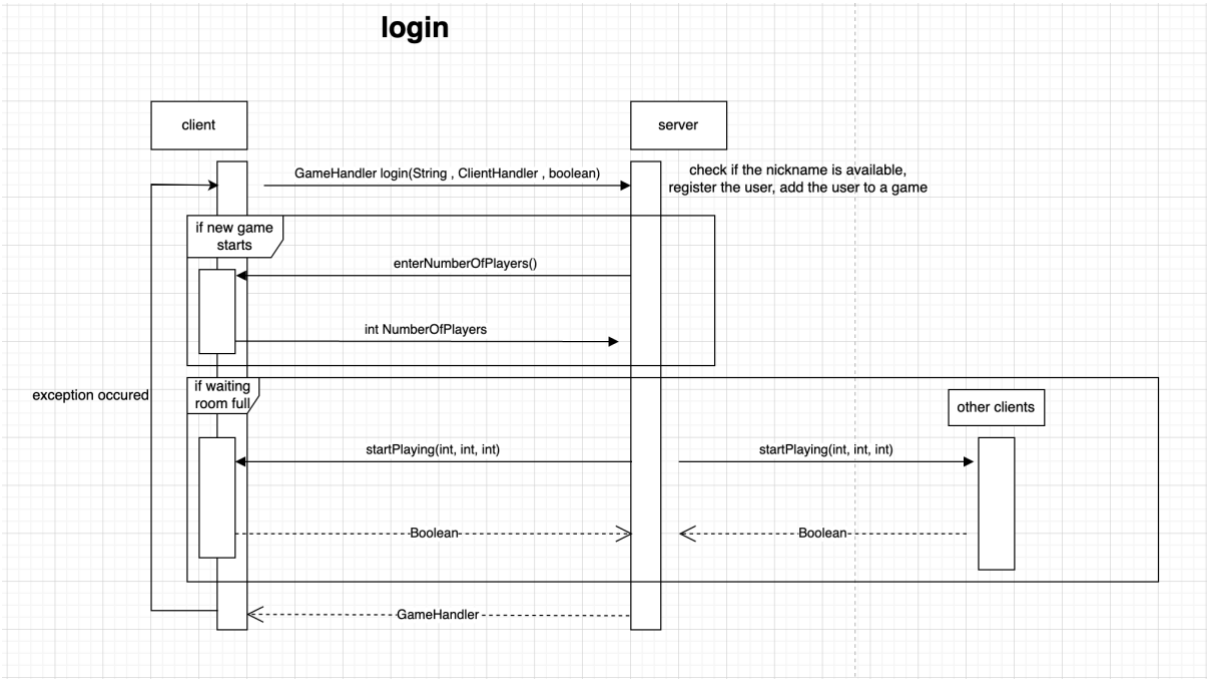
Risulta evidente che il numero di parametri “paramX” è assegnato in base al numero di parametri che richiede il metodo remoto. Si specifica che i campi vengono assegnati staticamente dal metodo richiedente.

2. Metodi remoti

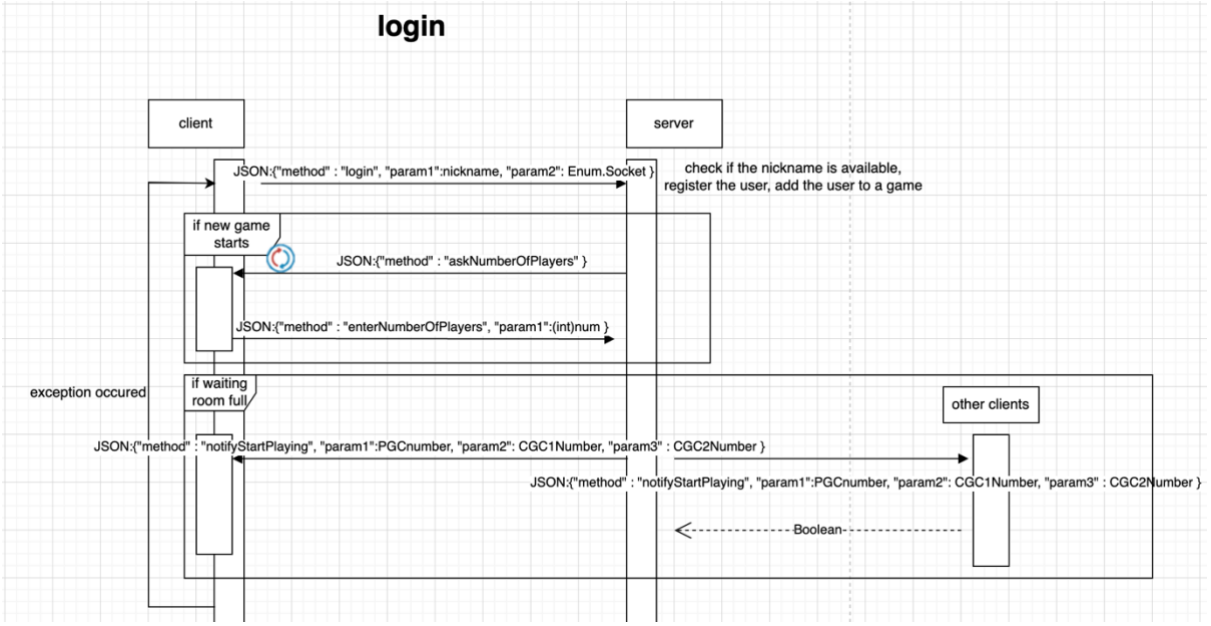
2.1 LOGIN

Quando un client accede per fare il login ad un game il GameHandler manda una richiesta al server con oggetto il nickname scelto dal player e il relativo ClientHandler.
Dopodiche avverrà un check per valutare se il nickname è disponibile, la registrazione dell’utente e l’aggiunta al relativo game (a meno che non venga lanciata un’eccezione).
Dopo questa fase iniziale se il player che ha fatto accesso è il primo del game, avviene la creazione di un nuovo game e il server chiede al client l’inserimento del numero di player con cui vuole avviare la partita.
Quando l’ultimo player di un game fa il login e quindi la waiting room diventa full, il server comunica a tutti i client presenti nella waiting room che è stato raggiunto il numero esatto di player del game e che si sta iniziando a giocare.

JAVA RMI



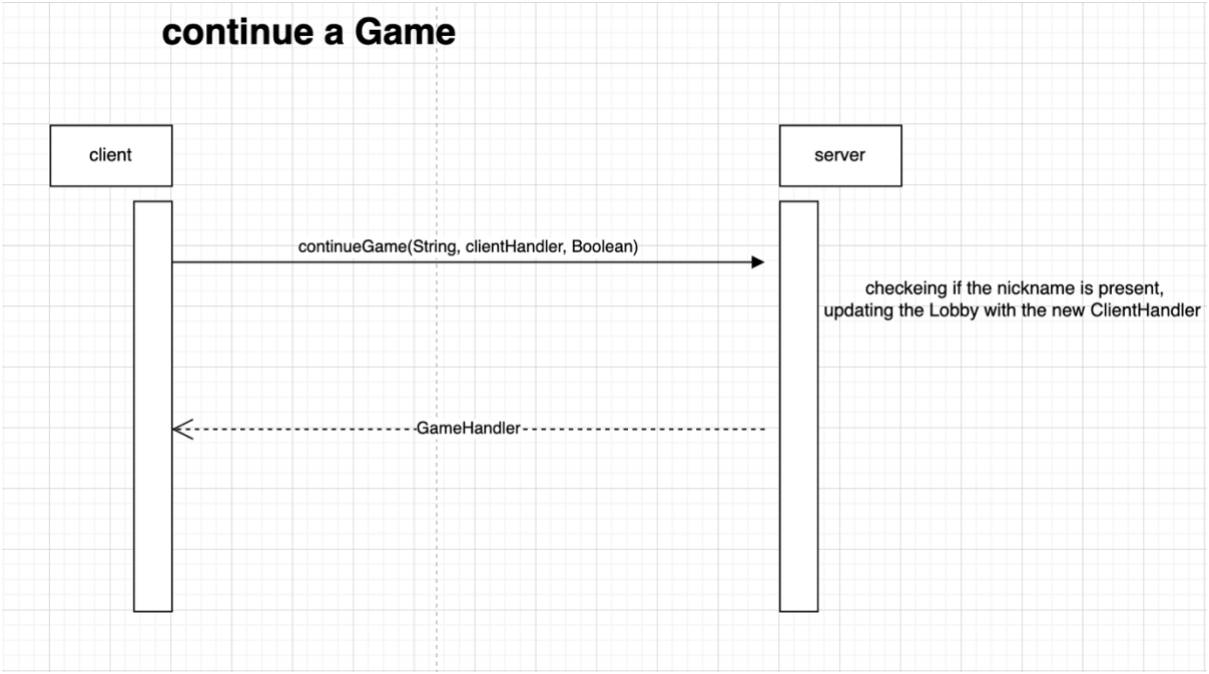
SOCKET TCP



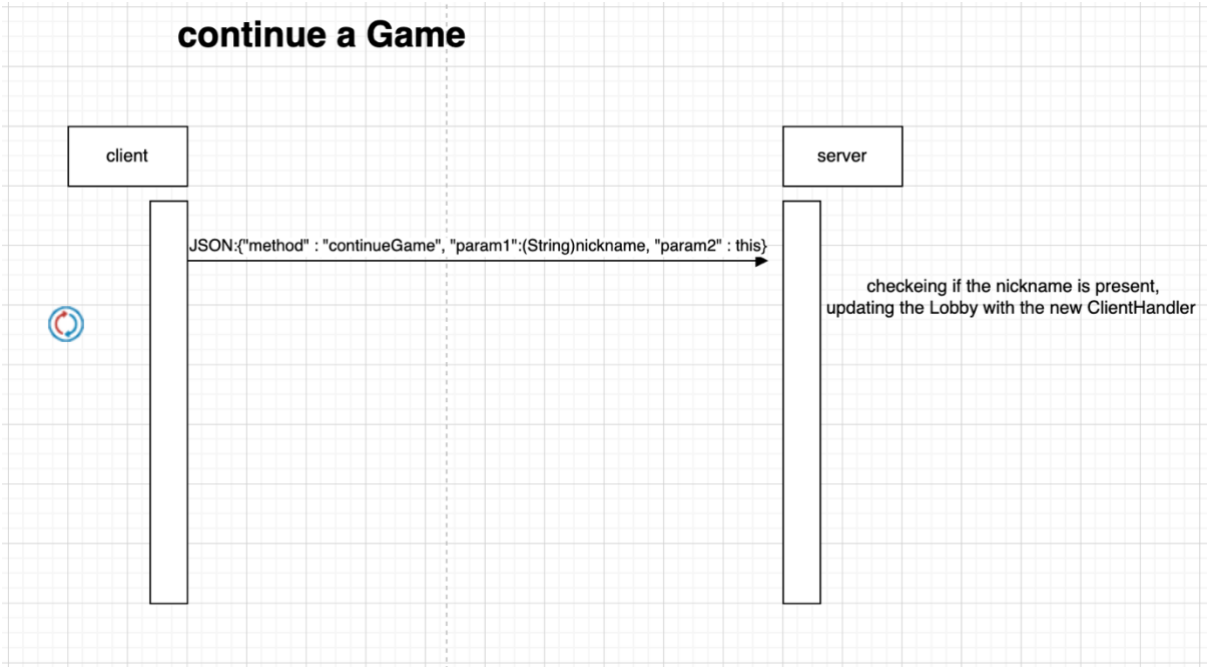
2.2 CONTINUE GAME

Quando un client vuole continuare una partita manda una richiesta al server che ha come oggetto il nickname del player e il relativo ClientHandler.
Il server andrà ad effettuare un check per vedere se il nickname è presente e dopo di che aggiornerà la lobby con il nuovo ClientHandler.
Il server ritornerà il GameHandler.

JAVA RMI



SOCKET TCP

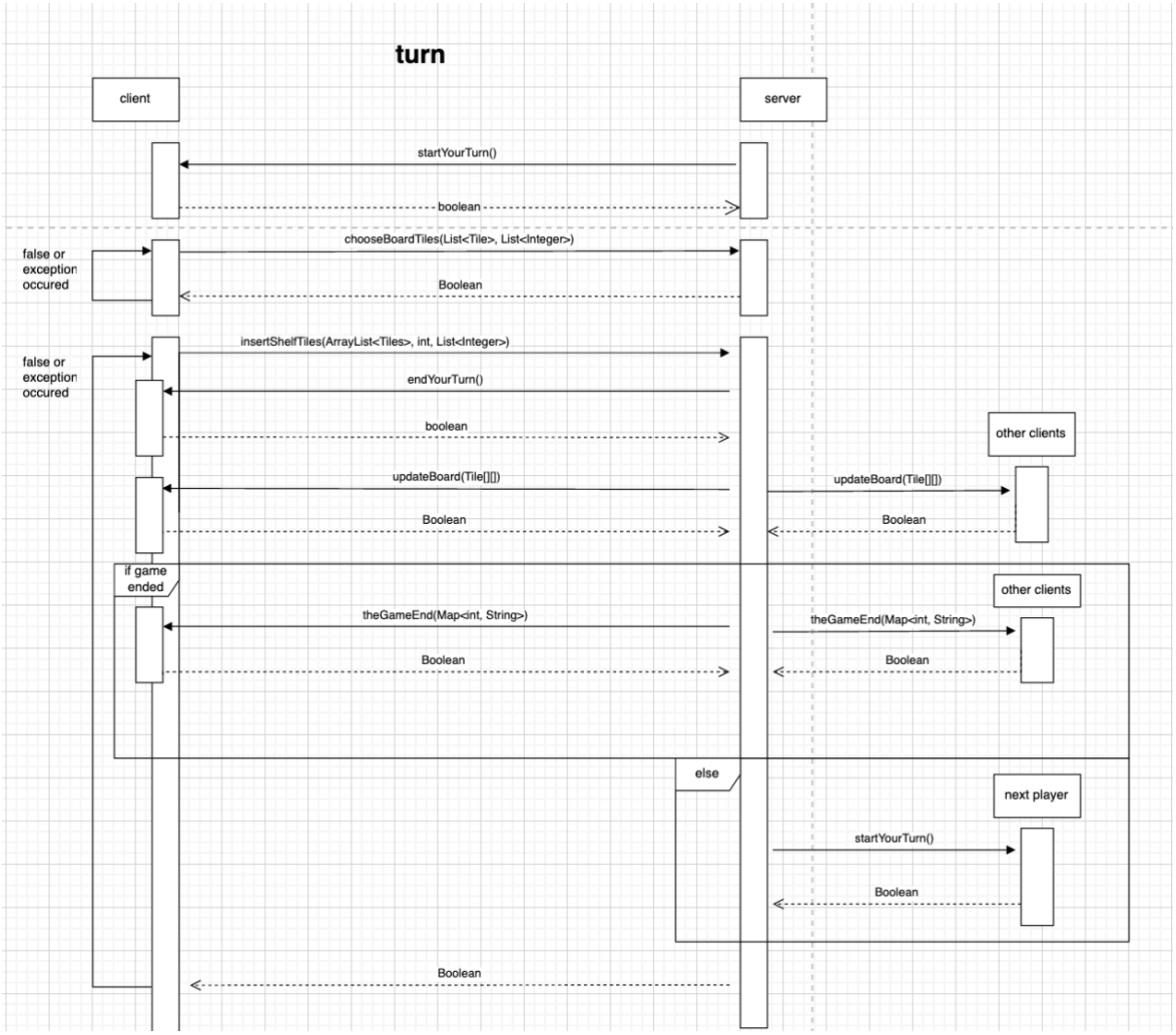


2.3 TURN

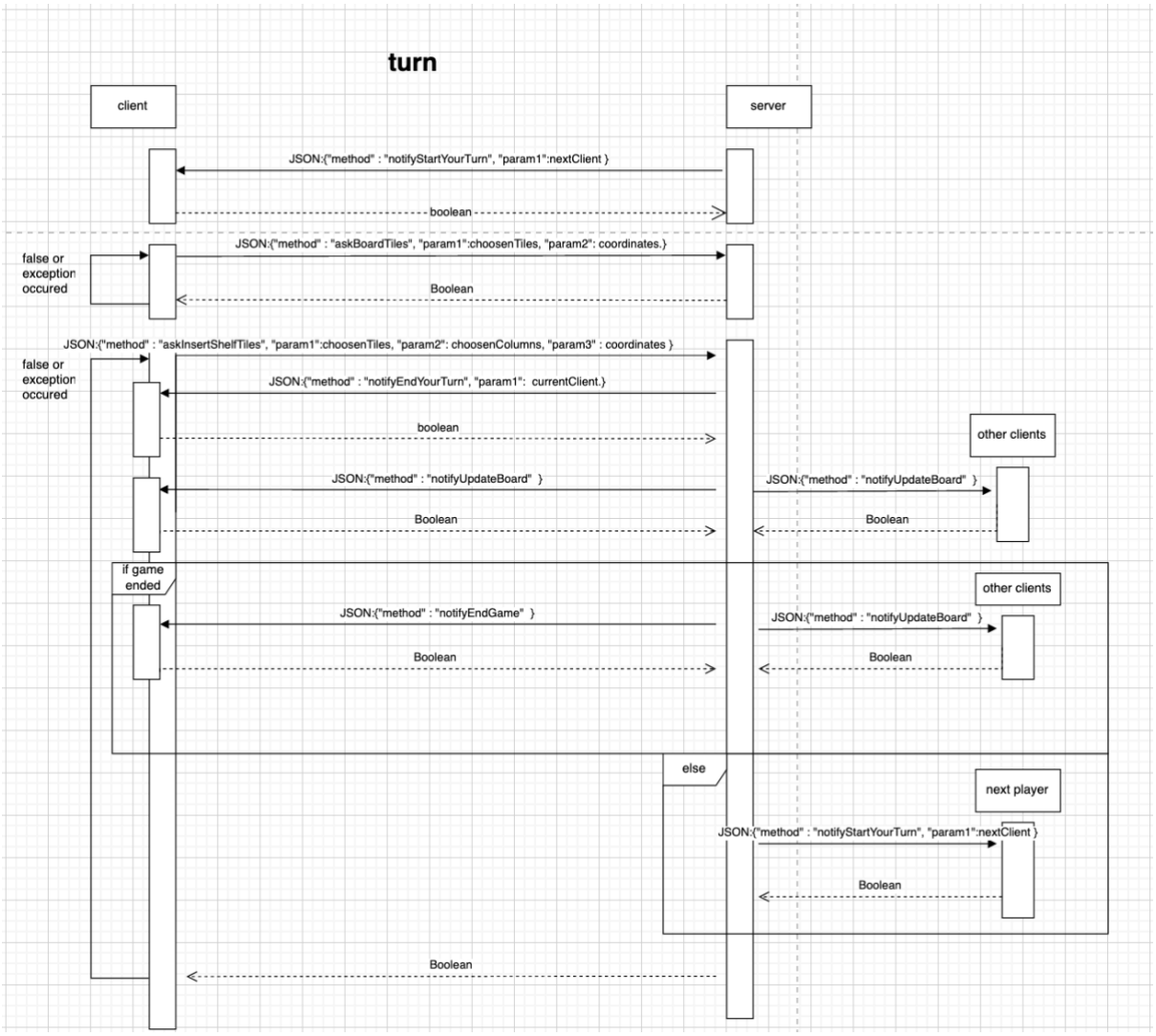
Quando parte il turno di un player il server lo comunica al client.
Il client andrà a scegliere le tiles dalla board comunicandolo al server che farà un check e ritornerà true se l'esito è positivo, altrimenti ritornerà false o un eccezione.

Dopo questo primo passaggio il client dovrà inserire le tiles nella propria bookshelf e anche in questo caso avverà un check da parte del server.
Il client ha finito il suo turno e ciò gli verrà comunicato dal server, il quale andrà a fare un update della board e lo comunicherà a tutti i client cosi che non vi siano più le tiles scelte e inserite dal client durante il proprio turno.
Se il client in questione è l'ultimo che doveva giocare e quindi il game è finito, il server lo comunicherà a tutti i client, altrimenti il server comunicherà al prossimo player (client) che è il suo turno.

JAVA RMI



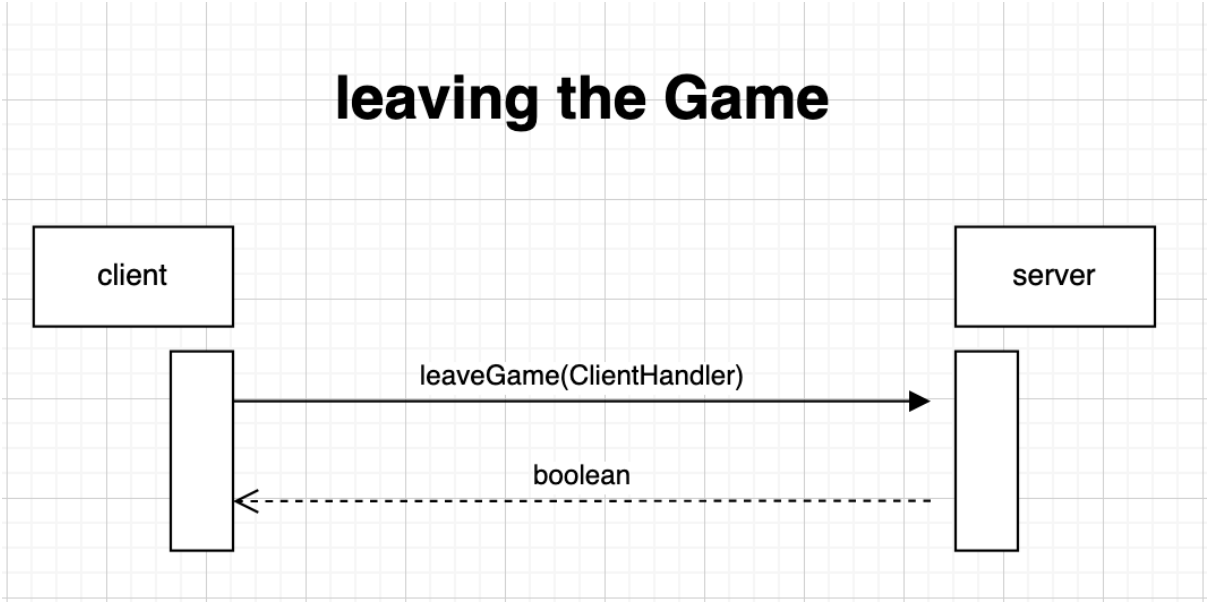
SOCKET TCP



2.3 LEAVE GAME

Quando un client vuole lasciare una partita manda una richiesta al server che ha come oggetto il relativo ClientHandler.
Il server ritornerà un boolean per comunicare che l'azione è andata a buon fine.

JAVA RMI



SOCKET TCP