Peer-Review 2: Network

<Agostino Contemi>, <Federico Consorte>, <Jacopo Corsi>, <Filippo Dodi>

Gruppo <AM04>

1 Introduzione

La gestione della comunicazione dal lato Server avviene attraverso la classe **ServerConnectionHandler** che gira su un thread a parte e si occupa di accettare le connessioni in ingresso. Per ogni connessione il **ServerConnectionHandler** associa un **ClientHandler** che comunicherà al client attraverso i propri **ServerSender** e **ServerReceiver,** i quali girano ognuno sul corrispettivo thread. Il **ServerSender** si occuperà dell'invio dei messaggi da parte del Controller, il quale passa prima attraverso il **ServerConnectionHandler** e poi il **ClientHandler**. Il **ServerReceiver** invece ha un riferimento al Controller e usa i metodi pubblici di quest' ultimo per comunicare i messaggi.

Dal lato Client viene utilizzato lo stesso procedimento ma **ClientConnectionHandler** crea direttamente il **ClientSender** e **ClientReceiver**.

Abbiamo scelto di implementare le funzionalità "Chat", "Resilienza alle disconnessioni" e "Persistenza".

2 Socket

Il **ServerConnectionHandler** resta in ascolto della connessione di nuovi client, effettuando il binding e poi istanziando **ClientHandler** che si occuperà di tutte le successive comunicazioni in ingresso da tale client.

Tutti i messaggi sono trasmessi in formato Serializable. Il **ServerReceiver** è in continuo ascolto sulla socket, mentre il **ServerSender** inoltra le informazioni dal client al controller.

Viene utilizzato lo stesso procedimento per il client.

3.1 Messaggi

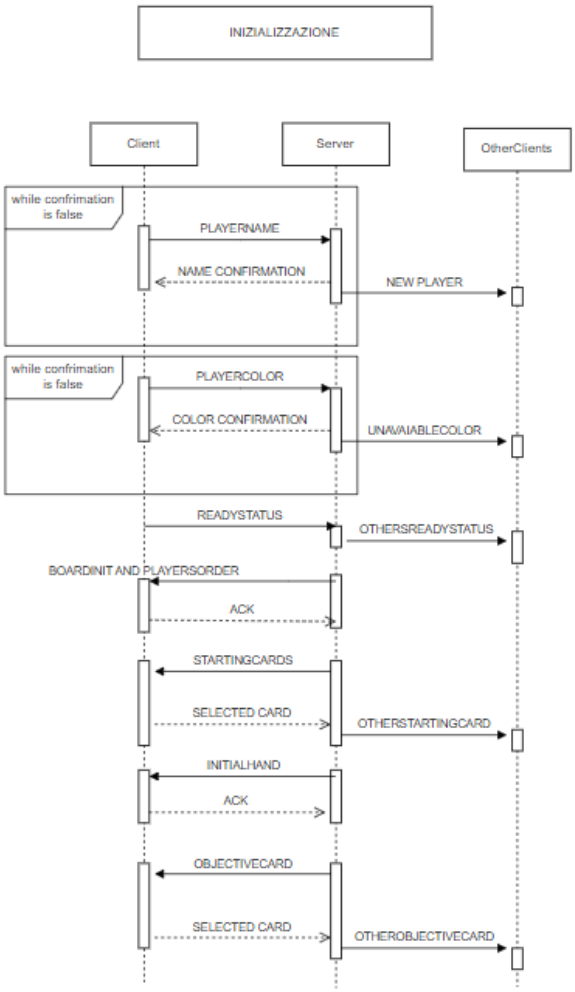
Per lo scambio di pacchetti tra server e client viene utilizzata la classe MessagePacket, questa classe fornisce un modo per incapsulare i messaggi da inviare tra un server e un client, consentendo la serializzazione e la deserializzazione dei pacchetti di messaggi e garantendo che i tipi di messaggi siano correttamente gestiti durante il processo. La classe **MessagePacket** è un wrapper che ha un attributo **payload**, un oggetto di tipo **GeneralMessage** che contiene i dati del messaggio.

Ci sono due costruttori, uno viene utilizzato per creare un nuovo messaggio a partire da un oggetto **GeneralMessage**. L'altro costruttore è utilizzato per ricostruire un messaggio da una stringa serializzata. Questo processo coinvolge la decodifica della stringa, la lettura di un oggetto dalla rappresentazione binaria utilizzando un **ObjectInputStream**, quindi il ripristino del pacchetto originale. La deserializzazione converte la stringa letta in un GeneralMessage.

GeneralMessage è un'interfaccia che estende Serializable, in cui vengono implementati due metodi **ClientExecute** e **ServerExecute** che si occuperanno della corrispettiva gestione del messaggio dal lato client o server.

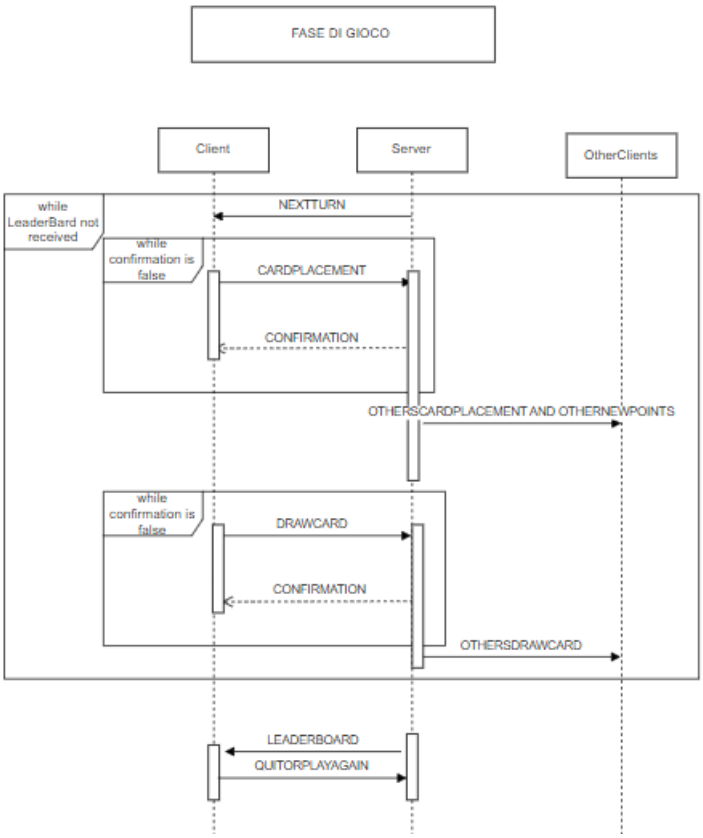
3.2 Sequence diagram per la fase di inizializzazione del gioco

Questa è la fase iniziale del gioco, in cui il client sceglierà dapprima il nome e il colore e infine dichiarerà di essere pronto. Successivamente gli vengono trasmessi la BoardInit, contenente le carte iniziali e l'ordine di gioco, ed infine vengono distribuite le StartingCards e la mano iniziale. Ack è l'acknowlegdment che conferma la ricezione del messaggio.



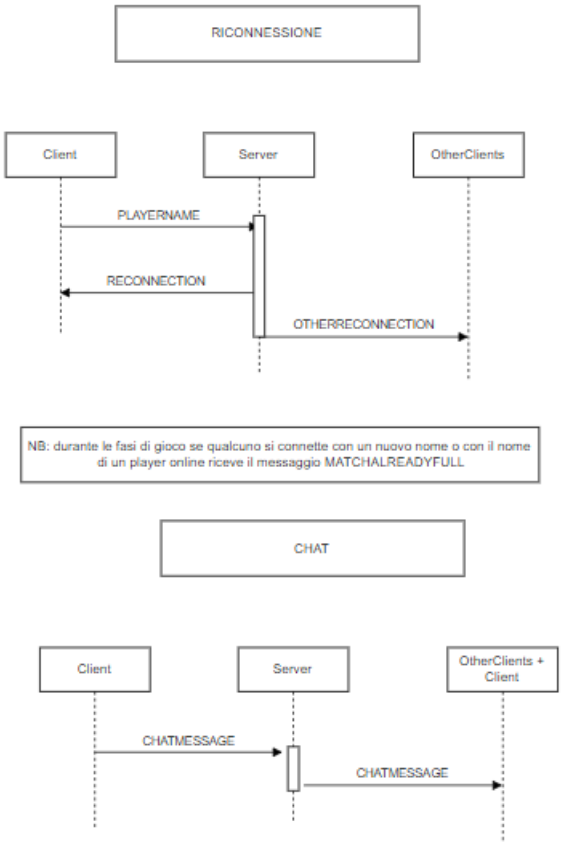
3.3 Sequence diagram per la fase di gioco

Nella fase di gioco tutti i giocatori pescano e giocano le carte secondo l'ordine stabilito. Si distingue una prima fase che termina al raggiungimento dei 20 punti da parte di uno dei giocatori o una volta finiti entrambi i deck. In seguito c'è la fase finale in cui si completa il turno in corso e ciascun giocatore gioca un turno addizionale. Infine vengono notificati a tutti i giocatori i loro punteggi e il vincitore con la LeaderBoard.



3.3 Sequence diagram per la chat e riconnessione

In questo sequence diagram mostriamo un esempio di sconnessione e riconnessione di un giocatore durante la fase a turni del gioco (per la relativa funzionalità avanzata), se il giocatore si disconnetti i suoi turni vengono saltati fino a che il giocatore non si riconnette.



4 RMI

Work in progress...