Protocollo di comunicazione client-server

Gruppo AM24 Gabriele Benedetti, Giuseppe Buccheri, Michele Canepari, Mattia Belfiore

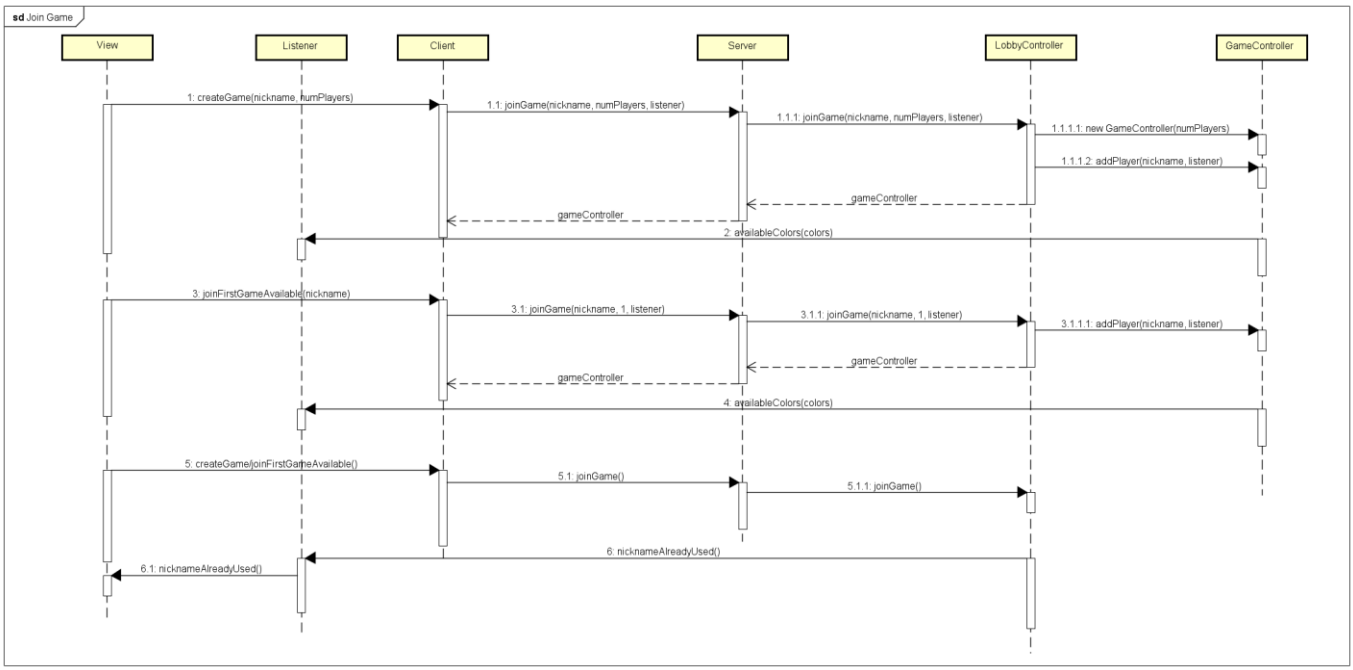
Architettura e Threading

Per gestire al meglio il flusso di gioco e la comunicazione tra client e server abbiamo deciso di implementare le seguenti classi con la seguente gestione tramite threading:

- Flow: classe utilizzata per gestire il flusso lato client e l'interazione con l'utente. Al suo interno vengono creati e gestiti due thread, uno che rimane in ascolto sulla tastiera e un secondo per interpretare le chiamate o i messaggi ricevuti dal server. Abbiamo poi aggiunto un altro thread che si occupa di interpretare i comandi scritti dall'utente (creato per la chat)
- Client: classe utilizzata per gestire le chiamate al server. Ci saranno quindi effettivamente due classi specifiche di questo tipo (una per RMI e una per Socket) che si occuperanno di inviare la richieste al server mediante invocazione di metodi remoti oppure messaggi (oggetti serializzati java)
- Server: classe che si occupa di gestire le richieste dei client per connettersi o creare le partite
- LobbyController: si occupa di gestire le lobby, creandone di nuove oppure aggiungendo i player in lobby esistenti in base alle richieste ricevute
- GameController: si occupa di gestire le partite. Alla creazione di una nuova lobby viene istanziato un nuovo oggetto di questo tipo su un nuovo thread per permettere la funzionalità aggiuntiva partite multiple
- GameListener: interfaccia di metodi che possono essere chiamati dal server verso il client

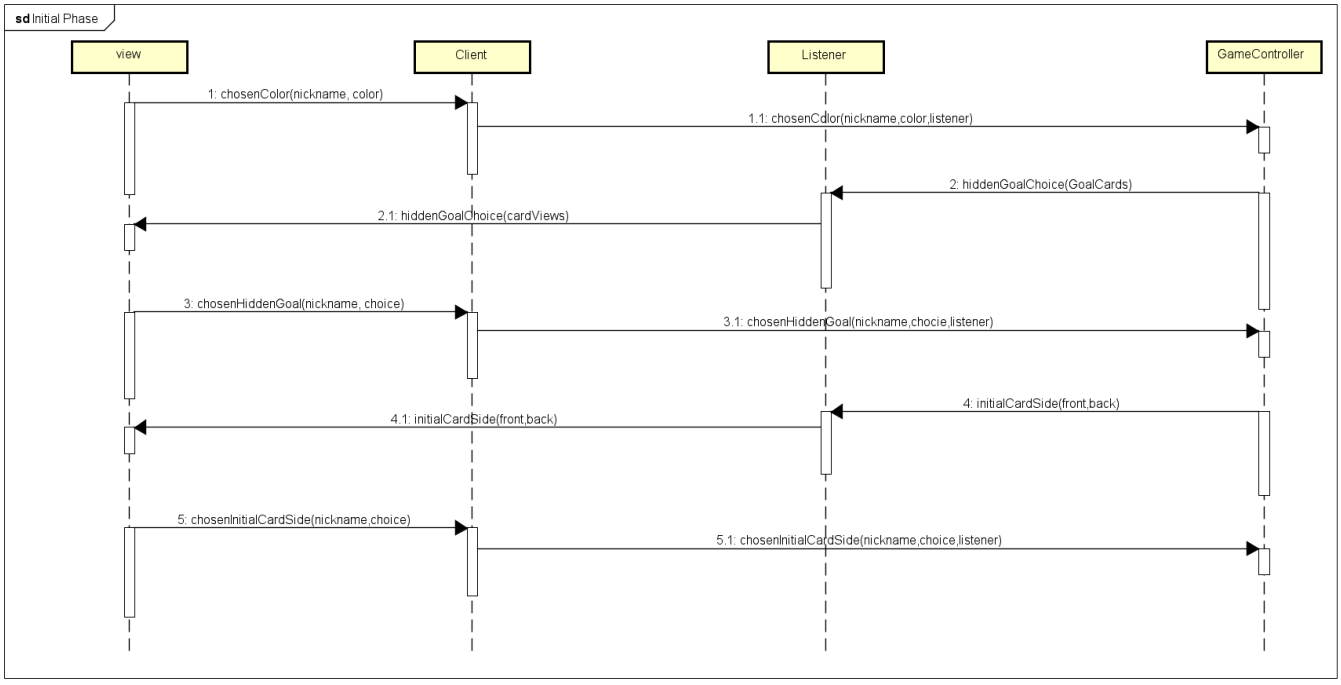
Il nostro flusso di gioco prevede 3 fasi principali:

1. **Creazione partita:**
   Una volta collegato al server, il client comunica ad esso se vuole creare una nuova partita (e quindi una nuova lobby) oppure se entrare in una già esistente. In entrambi i casi la View comunica al Client (oggetto) la scelta tramite i metodi createGame e joinFirstAvailableGame. Il Client invia il rispettivo messaggio al Server che si occupa di chiamare il metodo desiderato del LobbyController. In realtà abbiamo scelto di implementare un unico metodo (joinGame) per gestire entrambe le situazioni in base al parametro "numPlayers", nello specifico 1 per la join e un numero tra 2 e 4 per la create.
   Nel primo caso viene istanziato un nuovo GameController, aggiungendoci anche il giocatore; nel secondo il giocatore viene aggiunto alla prima lobby disponibile.
   Inoltre il LobbyController ritorna al Client un riferimento dell'oggetto remoto GameController nel quale il giocatore è stato inserito per permettere le successive comumicazioni riguardanti la partita. In caso di dati errati, ad esempio nickname già utilizzato, il controller lo comunica mediante un messaggio di errore. In assenza di errori vengono comunicati al giocatore i colori della pedina disponibili, sia per proseguire che come conferma della corretta aggiunta.



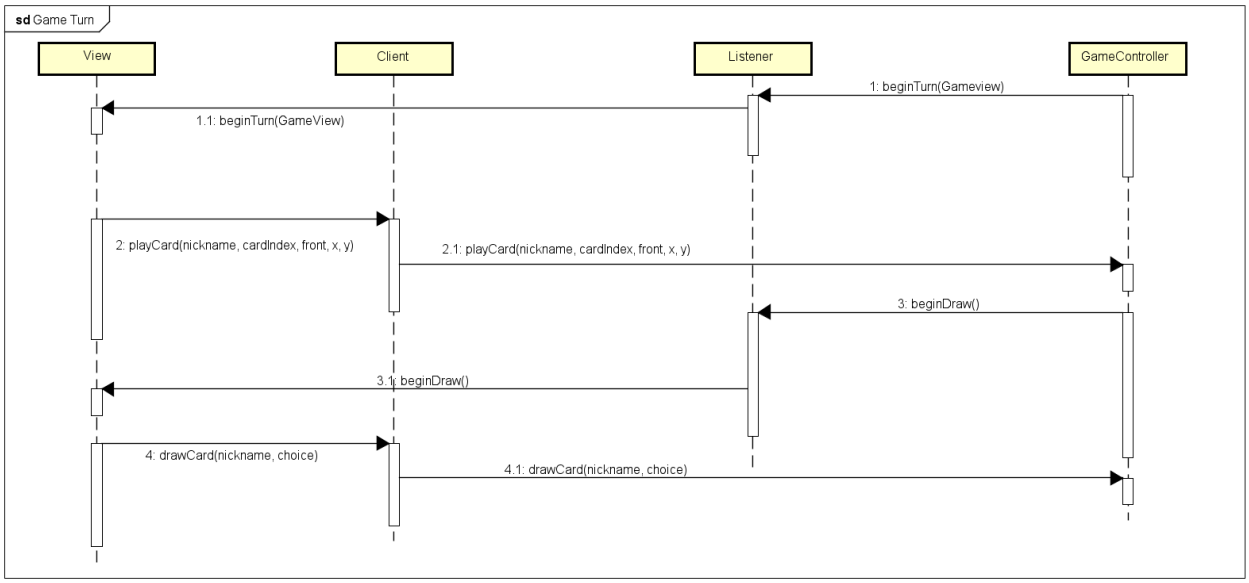
2. Fase iniziale di gioco:

In questo momento della partita il Client sceglie un colore tra quelli proposti, passandolo insieme al proprio nickname. Il Controller una volta ricevuta la scelta manda al Client, passando dal Listener, una lista di due carte obiettivo (di tipo GameCardView, una versione semplificata dell'oggetto GameCard presente nel Model) che saranno nuovamente oggetto di scelta da parte del player. Gli stessi passaggi vengono ripetuti per la scelta dello schieramento (front o back) della carta iniziale pescata ed assegnata al giocatore.
Ogni risposta del controller, oltre a servire come inizio del passaggio successivo, rappresenta una conferma che i dati ricevuti sono corretti. In caso contrario notifica al Client un malfunzionamento tramite un messaggio di errore (non inseriti nel sequence diagram)



3. Turno di gioco:

Il Controller richiama un metodo del Listener che ha come parametro un elemento di tipo GameView che ha al suo interno una serie di informazioni riguardanti lo stato attuale della partita (plancia di gioco e le carte in mano ai giocatori). Il Listener a sua volta richiamerà un metodo della View che svolge lo stesso compito, andando a mostrare al giocatore la sua area di gioco. A questo punto segue il normale proseguire del turno di gioco, il player sceglie una carta della sua mano, la sua disposizione e dove posizionarla nella mappa. La View andrà dunque a richiamare un metodo del Client, quest'ultimo inoltrerà le informazioni ricevute al Controller. Il Controller una volta piazzata la carta con successo, richiama un metodo del Listener che notifica il successo dell'azione e dà inizio alla fase di pesca del turno invocando un metodo della View che permette di mostrare al giocatore le carte disponibili per la pesca. In maniera analoga alla fase di gioco della carta (View -> Client -> Controller) si procede con la pesca indicando la scelta fatta. Il gioco prosegue in questo modo fino alla ricezione di un messaggio dal server che comunica la fine del gioco con la rispettiva classifica e vincitore.



Threading

- **Server:** come detto in precedenza il nostro controller prevede l'utilizzo di due classi. LobbyController sarà sempre una unica istanza, mentre per ogni partita creata avremo un GameController eseguito su un thread. Inoltre per migliorare la velocità le chiamate/invio di messaggi al client sono svolte da un thread. Per la gestione tramite Socket utilizziamo un thread principale per l'accettazione e gestione dei canali con i client, mentre un altro thread per ogni canale creato, in ascolto della ricezione e interpretazione di messaggi
- **Client:** la gestione lato client prevede un thread principale in ascolto per le chiamate che arrivano dal server. Ad ogni chiamata viene creato un "Evento", interpretato poi da un thread secondario. Per permettere ai giocatori di utilizzare la chat in qualsiasi momento è attivo un thread in ascolto sulla tastiera, e un secondo per l'interpretazione di essi (nello specifico si occupa di interpretare i messaggi legati alla chat e di inserire gli altri in un buffer).