**Descrizione UML Network**

Abbiamo deciso di realizzare la rete sia con RMI che con Socket.

**Command Pattern**

Per uniformare le due tecnologie, abbiamo implementato il Command Pattern sia per la comunicazione dal client verso il server, sia per quella dal server verso il client.
In particolare, i client (sia RMI che Socket) possono inviare al server dei GamesManagerCommand e dei GameCommand.
I GamesManagerCommand sono i messaggi che i client possono inviare nella fase di scelta di una partita, mentre i GameCommand quelli che i client possono mandare durante lo svolgimento della partita.
La comunicazione dal server al client riguarda l'invio degli aggiornamenti del modello attraverso l'utilizzo di Listener.
Anche l'invio di aggiornamenti utilizza il Command Pattern, in particolare, il server può inviare al client degli Update.
L'utilizzo del Command Pattern ci ha permesso di gestire anche la comunicazione tramite Socket al pari di quella con RMI, ovvero senza dover serializzare e inviare un overhead per indicare il tipo di messaggio da comunicare.

**RMI**

Nella nostra implementazione di RMI ci sono 3 classi:

- RMIClient: rappresenta il client RMI
- RMIServerGamesManager: rappresenta il server RMI che gestisce tutte le partite e l'accesso alle stesse
- RMIServerGame: rappresenta il server RMI per la singola partita

All'avvio del gioco viene creato un RMIServerGamesManager, ovvero un Server che gestisce tutti i client RMI.
Quando un giocatore decide di utilizzare RMI, viene creato un RMIClient, che può richiedere, tramite registry il riferimento a RMIServerGamesManager.
Tramite il server generale, può decidere di creare una partita oppure di visualizzare le partite in attesa di giocatori ed entrare in una di queste.
Dopo questa fase, il server generale imposta al client il riferimento al RMIServerGame per la partita in cui il client è entrato: da questo momento in poi, il client può comunicare con il server della partita per compiere le azioni di gioco.

**Socket**

Nella nostra implementazione di Socket ci sono 4 classi:

- SocketClient: rappresenta il client Socket
- VirtualSocketServer: viene utilizzata lato client come astrazione del server: si occupa di gestire l'invio dei messaggi di tipo GamesManagerCommand e GameCommand sul canale di comunicazione con il server
- SocketServer: rappresenta il server che gestisce l'accesso alle partite, alla connessione di un nuovo client crea inoltre il SocketClientHander ad esso associato; di questa classe viene creata una sola istanza all'avvio del gioco

- SocketClientHander: viene utilizzata lato server come astrazione del client, si occupa di gestire la comunicazione da e verso il client: invia i messaggi di tipo Update al client ed esegue le operazioni sul modello contenute nei messaggi di tipo GamesManagerCommand e GameCommand ricevuti dal client