PEER REVIEW #2

Gruppo: GC27

Componenti: Valentini Lorenzo, Vicedomini Andrea, Wang Toni, Vaccaro Lucrezia

Commento UML sulla parte di rete

Nel nostro progetto vogliamo implementare entrambi i tipo di comunicazioni: Socket ed RMI.

Al lancio dell'applicativo lato client, l'utente avrà la possibilità di scegliere la tecnologia a livello di rete da utilizzare, così come l'indirizzo IP necessario a stabilire la connessione col server. In base alla scelta, viene istanziato l'opportuno collegamento e l'utente viene accolto nel gioco tramite il metodo welcomePlayer() dichiarato dall'interfaccia VirtualServer: gli viene chiesto l'ID della partita a cui vuole partecipare (implementiamo la Funzionalità Aggiuntiva delle partite multiple), altrimenti gli viene assegnata una nuova istanza di Game col relativo ID. Se il tutto va a buon fine, il collegamento è stabilito e l'utente rimane in attesa che la lobby si riempia di giocatori per poter iniziare. L'utilizzo dell'interfaccia VirtualServer, implementata sia da RmiServer che SocketServerProxy ci permette di rendere unico il codice della TUI, e in seguito della GUI, poiché viene effettuata una chiamata al metodo del server senza che influisca il tipo di server che ne va a svolgere l'esecuzione.

NB:

- Ad oggi non abbiamo ancora predisposto l'opzione per giocare tramite GUI, motivo per cui le chiamate sono fatte direttamente alla TUI. Prossimamente implementeremo anche la parte grafica.
- Al momento Socket è asincrono mentre RMI è sincrono: dobbiamo ancora rendere RMI asincrono.
- RMI di base non notifica la disconnessione al server: vogliamo implementare una funzione di ping tra client e server che tenga aggiornato il server sullo stato della connessione, soprattutto in vista della FA resilienza alle disconnessioni.

RMI

Il protocollo RMI prevede la creazione di un RmiServer che crea il suo stub per poi esportare l'oggetto remoto e fare il binding nel registry alla porta di default. Ora che il "VirtualServer" e lo stub sono legati, un client può collegarsi liberamente conoscendo l'indirizzo del server.

Un client, quando si connette, viene legato al server tramite il registry nella fase di lookup e aggiunto alla lista di utenti connessi (VirtualServer.connect()). Da qui in poi la connessione è stabilita e si tratta solo di fare chiamate all'oggetto remoto.

Alcuni metodi appaiono duplicati sulla TUI nel sequence diagram perché vengono chiamati diversamente in base agli input.

SOCKET

Il protocollo Socket sfrutta diverse classi per cercare di far apparire la comunicazione tra client e server come se fosse per invocazione di metodi. Di seguito vi è la spiegazione di ogni classe ed il compito che esegue:

- **SocketClient:**
  al suo avvio istanzia un oggetto SocketServerProxy e parla esclusivamente con lui attraverso le invocazioni di metodi, l'uno sull'altro
- **SocketServerProxy:**
  Una volta ricevuto l'indirizzo IP in ingresso, si instaura dapprima una connessione con il SocketServer. Successivamente, si occupa di tradurre i metodi chiamati dal client in un formato trasmissibile via Socket, che verrà ricevuto dal server e eseguito. Inoltre, sarà responsabile anche dell'ascolto degli aggiornamenti da parte del server e della loro traduzione in chiamate ai metodi del client.
- **SocketServer:**
  all'avvio del lato server viene lanciata una sua istanza che si mette in costante ascolto di richieste di connessione. Quando riceve una richiesta crea un oggetto di tipo ClientHandler e lancia un thread per eseguire la sua run().
- **ClientHandler:**
  è approssimabile ad un server personale per ogni singolo utente, quindi eseguito il metodo run() sarà costantemente in ascolto dei messaggi ricevuti dal SocketServerProxy e li tradurrà in comandi eseguibili dai controller.
- **SocketClientProxy:**
  ultima classe per chiudere il ciclo, manda gli update nel formato adatto al trasferimento dati attraverso i Socket al SocketServerProxy.

Per fare un breve riassunto abbiamo: SocketClient crea SocketServerProxy che manda messaggi a ClientHandler (una volta creata la connessione) e riceve messaggi da SocketServerProxy, il Client Handler invoca i metodi del controller, e quando il model manda un update questo viene ricevuto dal SocketClientProxy che lo invia al SocketServerProxy, che infine lo traduce in un metodo del SocketClient per la sua visualizzazione.