Peer-Review 2: UML Model, Controller, Rete

GRUPPO REVISORE
Gruppo: GC 34
Polizzi Chiara, Raimondi Francesco, Rubagotti Andrea, Santagata Maria Concetta (referente)

VALUTAZIONE DEL DIAGRAMMA UML DELLE CLASSI DEL GRUPPO GC 24

Lati positivi

- Uml rete: Creare una classe Packet è stato utile per serializzare gli oggetti che subivano modifiche evitando di agire direttamente sul model. Avere un'unica classe astratta Packet (con tutti i figli), inoltre, è utile per avere solamente un metodo remoto (sendPacket e receivePacket).
- Uml rete: ClientConnection è utile, in quanto punto di congiunzione tra le due connessioni TCP e RMI. Infatti il suo uso risulta comodo per tenere traccia dei diversi Client nel ServerNetworkHandler.
- Buono l'uso di stati del Game e del Client per segnalare le varie fasi del gioco.
- Okay l'uso dei Commands per leggere da linea di comando.

Lati negativi

- La documentazione consegnataci non è chiara ed abbastanza esplicativa: come viene implementato il pattern Listeners e quindi le varie callback? Non compreso che cosa siano i Commands nel controller. Sono degli eventi? Se sì, allora servono a cosa? Quali sono le vostre funzionalità aggiuntive (forse chat e resilienza alle disconnessioni)?
- Attenzione al punto 1 del protocollo di comunicazione: assenza della casistica in cui lo username sia già stato utilizzato o sia invalido.
- Attenzione al punto 3 del protocollo di comunicazione: i giocatori scelgono la carta obiettivo e il colore della pedina, tuttavia queste informazioni vengono memorizzate nel server ma non vengono trasmesse agli altri client (callback mancante).
- Attenzione al punto 4 del protocollo di comunicazione: cosa succede invece se la carta piazzata è un'azione invalida? Va comunque comunicato a tutti che è stata tentata un'azione invalida. (pattern listeners)
- Attenzione al punto 1 e 7 del protocollo di comunicazione: potrebbero essere creati due ConnectionPacket differenti per l'ingresso nella lobby di un nuovo giocatore e per la reimmissione nella partita di un giocatore che ha avuto un problema di connessione (al posto di usare 2 booleani).
- Congestione di rete: tutti i messaggi che producono un'azione di gioco vengono inviati all'unico server (non sappiamo se avete fatto le partite multiple, ma non essendo indicato non possiamo saperlo).

- I client contengono le informazioni sullo stato della partita nel loro clientController (ma ad esempio la board? Viene inviata ad ogni richiesta dal server?)
- Non siamo convinti dell'avere un handler di messaggi per ogni specifico Packet(sia lato server che client), spreco di thread e risorse. Si potrebbe utilizzare un unico handler (aggiungendo un Event che caratterizza l'azione per poter poi usare uno switch-case).
- Occhio comunque all'uso dei thread: è necessario garantire il giusto ordine di esecuzione delle richieste (packet) ma i thread, se non controllati adeguatamente, non danno garanzie in merito.

Confronto tra le architetture e punti di forza

- Voi avete una classe Packet, usata sia per RMI che TCP, noi invece per RMI abbiamo sfruttato l'invocazione diretta di metodi mentre per TCP abbiamo una classe SCKMessage che contiene gli oggetti necessari a soddisfare la determinata azione e l'Event (enumerazione) relativo.
- Abbiamo utilizzato il pattern MVC sfruttando altresì il pattern Observer/Observable mentre nel vostro uml non è presente alcuna funzione di "callback".
- Le eccezioni le abbiamo inserite, più che nel Controller, nel Model.
- Voi avete una packetQueue (avete reso asincrono RMI), mentre noi abbiamo reso sincrono TCP in modo da sfruttare tutto il potenziale di invocazione diretta di RMI.
- La vostra classe clientController vi permette di gestire interamente il client, potremmo in parte prenderne spunto.