Peer-Review 2: UML and COMUNICATION PROTOCOL

<LORENZO CODEGA>, <ANDREA D'ALBA>, <SHAHRION ALAM>, <DESHAN THARINDU EDIRISINGHE EDIRISINGHE MUDIYANSELAGE>
Gruppo <AM46>

8 maggio 2023

Valutazione del sequence diagram del gruppo <AM09>.

1 Lati positivi

Il testo è molto chiaro e semplice da capire, ottima l'idea di inserire anche il class diagram del network per contestualizzare il funzionamento della rete.

Ottima l'idea di usare l'adapter pattern per rendere più facile l'implementazione di nuovi metodi e la gestione di errori.

Buona l'idea di usare tryLogin(...), attenzione però a sincronizzare il timer altrimenti potreste dover aspettare la fine di un'altro task per l'invio del prossimo tryLogin(...) o addirittura potreste entrare in un loop (altamente improbabile ma con i timer non si sa mai).

Buona l'idea di scorporare la Lobby dal MODEL, semplifica la gestione del login a lato server.

Ottima l'idea dei messaggi privati o publici.

2 Lati negativi

Non ci sono lati negativi, solo filosofie di pensero diverse.
Un esempio è il lobby admin. Il nostro Model ha un metodo che permette di avviare il turno di gioco SOLO al PlayerToPlay (SOLO UNA VOLTA). Alla fine di ogni turno il giocatore deve mandare una richiesta di END-TURN, questa ritorna un valore boolean e il nuovo PlayerToPlay. Quest'ultimo viene mandato in broadcast a tutti i giocatori e così il giocatore il cui username corrisponde al PlayerToPlay saprà che deve iniziare il turno. Quando il PlayerToPlay si disconnette o non inizia il turno (dopo n secondi) il server sà automaticamente di dover "terminare" il suo turno e aggiornare il PlayerToPlay mentre nel vostro caso il server è succube del Admin. Come detto prima non è una cosa negativa ma solo una differenza di filosofia di gioco.

Se dobbiamo proprio essere pignoli, per il sequence diagram da inserire su GitHub cosigliamo di mettere il protocollo di comunicazione completo di tutti i messaggi tra client e server. Anche se è più chiaro avere le varie sequenze di comunicazione scorporate potrebbe comunque essere utile avere tutta la sequenza completa (ALMENO DELLA MAIN LINE) per avere un'idea generale del protocollo di comunicazione. A questo si possono aggiungere altre componenti che mostrano la comunicazione in caso di errori di gioco o di comunicazione.
Per esempio:
se il giocatore X manda la risposta a Client.pickedFromBoard(...) MA non ci sono tiles pescabili perché la Board è ormai quasi vuota, sarebbe meglio mostrare come viene gestita questa casistica particolare, e così via per tutti i casi particolari.

3 Confronto tra le architetture

I punti di forza del vostro sequence diagram sono sicuramente la chiarezza e la semplicità, in particolare è geniale l'uso del adapter pattern. Nel nostro network non abbiamo usato nessun pattern specifico ma solo creato in parallelo il serverTCP e il serverRMI, che lavorano con lo stesso oggetto controller lato server e quindi con gli stessi metodi. Non abbiamo un adapter come nel vostro caso in quanto ogni messaggio TCP è incapsulato in un oggettto Command che trasformiamo ad ogni ricezione con FromJson(...). Dopodichè il server passa gli atributi corretti dell'oggetto Commad in base a cosa è scritto nel primo attributo di tipo stringa.
Per essere chiari:

Gson g;
String Messaggio-ricevuto;
Command Messaggio-da-elaborare=g.FromJson(Messaggio-ricevuto,Command);
Switch(Command.cmd)
    case("LOGIN"): ...
    case("   ...   "): ...

Sistema rudimentale rispetto al vostro ma pur sempre efficace. Presa visione del vostro pattern potremmo applicarlo in maniera simile anche noi, oppure semplicemente incorporare i Command.cmd in un'interfaccia con tutti i comandi.