Peer Review 2: protocollo di rete

Leonardo Bianconi, Alexandru Gabriel Bradatan, Mattia Busso
(Gruppo AM36)

May 6, 2022

1 Lati positivi

- Abbiamo valutato positivamente l'utilizzo di JSON come linguaggio di serializzazione: esso permette l'astrazione dal linguaggio di programmazione utilizzato (a differenza dell'uso di Java serializable) e permette una lettura e un debug più semplici rispetto a, per esempio, un protocollo in *plain-text*.
- Il vostro documento risulta molto dettagliato e non ci sono casistiche (a nostro avviso) che non sono state considerate.

2 Lati negativi

2.1 Informazioni contenute nella stringa `"message"`

Molti dei messaggi inviati dal server contengono informazioni all'interno del valore di `"message"` (a partire dal generico messaggio d'errore `ERR_MESSAGE`).

Ricevuto un messaggio dal server del tipo: {`"message"`¨`Error!!"+Err`} la sua gestione risulta facile per quanto riguarda l'implementazione del client via CLI: il messaggio d'errore può essere semplicemente stampato a schermo, ma, per quanto riguarda la GUI, risulterebbe complicato eseguire il parsing del messaggio d'errore per modificare elementi grafici in base all'errore ricevuto (eventualità che immaginiamo possa verificarsi in una partita).

Altri esempi sono i messaggi `TURN_NUMBER`, `PLANNING_PHASE` e `ACTION_PHASE`: non disponendo di un attributo caratteristico, lato client il parsing è necessario per memorizzare informazioni sullo stato della partita (cosa che è necesssaria per lo meno lato GUI).

Per quanto riguarda i messaggi spediti dal client al server, una chiave unica per tipo di messaggio è presente (ci riferiamo, ad esempio a `"num_of_players"` in `TWO_PLAYERS_GAME`), anche se sarebbe più semplice includere questa informazione nel valore di una chiave `"type"`. In questo modo sarebbe possibile eseguire uno *switch* su questo valore, piuttosto che controllare che esista una determinata chiave.

2.2 Necessità di richiedere manualmente al server gli aggiornamenti

Riteniamo problematico il fatto di non aver incluso un messaggio di *update* da parte del server, lasciando questo compito al client. In particolare, questa scelta si pone in contrasto con l'architettura presentata (che assomiglia in molti aspetti a un'architettura *server-driven*). Questa scelta inoltre, complica la gestione degli aggiornamenti lato client: in questo caso immaginiamo che i client eseguano *polling* continuo per ricevere gli aggiornamenti della board, tramite il messaggio di `REQ_SHOW_BOARD`, soprattutto per quanto riguarda i client in fase d'attesa.

2.3 Messaggi di update di sotto-parti dell'area di gioco

Riteniamo che i messaggi `REQ_SHOW_MY_ENTRANCE`, `REQ_SHOW_MY_TABLE`, `REQ_SHOW_ISLANDS`, `REQ_SHOW_CLOUDS` (e relative risposte del server) siano superflui, e che basti implementare correttamente il messaggio di update generale `SHOW_BOARD`. Questo perché assumiamo la rete essere molto veloce e non ci preoccupiamo del traffico di messaggi scambiati, né della dimensione degli stessi, dunque è possibile implementare il solo messaggio di update globale e inviare sempre quello tra client e server.

3 Confronto tra i due protocolli

- Entrambi i protocolli usano JSON, che riteniamo essere una buona scelta perché è un formato di messaggi *language-agnostic* e che permette una maggiore comprensione dei messaggi e debug rispetto, per esempio, a una serializzazione implementata tramite l'interfaccia Java *Serializable*.

- Il nostro protocollo differisce dal vostro per quanto riguarda la lunghezza e la frequenza dei messaggi: nella nostra architettura i messaggi inviati dal client sono uno per interazione, e gli *update* da parte del server vengono inviati in broadcast e fungono anche da messaggi di *acknowledgement* di avvenuto cambiamento dello stato. Questa scelta comporta un ridotto numero di messaggi implementati ma una maggiore lunghezza degli stessi. Nella vostra architettura, i messaggi sono più frequenti, e spesso singole interazioni client-server richiedono una serie più numerosa di messaggi scambiati, che riflettono i passaggi eseguiti manualmente dal giocatore all'interno del gioco (ad esempio, lo scambio di messaggi relativo all'uso delle carte personaggio).

- Abbiamo deciso, a differenza della vostra architettura, di inviare nel messaggio di update soltanto i cambiamenti del modello dovuti alla precedente azione da parte di un client. Il vostro server, invece, invia tutto lo stato corrente di gioco ad ogni messaggio di `SHOW_BOARD` (e simili). In generale, riteniamo che entrambe le soluzioni siano valide e che la differenza

rifletta principalmente la volontà da parte nostra di sviluppare un client più *"thick"*, mentre la vostra architettura risulta più *server-heavy*.