Peer-Review 2: Network

Davide Preatoni, Federico Sarrocco, Alessandro Vacca
Gruppo GC30

8 maggio 2022

Essendo il client non presente nel diagramma UML e non avendo fornito ulteriori delucidazioni o esempi (UML sequence diagram) sulle interazioni client-server, il protocollo di rete presentato è quindi valutabile parzialmente. Di seguito è presente la valutazione del protocollo di rete del gruppo GC40.

1 Lati positivi

Una nota a favore del progetto in questione è sicuramente la non ambiguità dei messaggi tra client e server: in particolare, il client esegue delle azioni sul server inviando messaggi di tipo *Command* e riceve gli aggiornamenti del model attraverso dei messaggi di tipo *ModelChangeEvent*. Un'interessante spunto è l'implementazione dell'interfaccia *AutoCloseable* da parte del *ServerController*; essa è utile per gestire il caso della corretta chiusura del server, unita ad ulteriori controlli (dovuti alla natura del protocollo TCP). Grande attenzione ai fini della funzionalità aggiuntiva *Partite Multiple* è stata posta fornendo i *GeneralCommand*, che consentono ad un client di scegliere tra la creazione di una partita o l'unione ad una partita in fase di creazione. Inoltre, grazie alle classi *Lobby* e *LobbyCommand* è possibile gestire lobby multiple e di conseguenza, più partite.

2 Lati negativi

Il gruppo in analisi ha sviluppato un metodo personalizzato per aggiornare la View a seguito di un cambiamento nel Model: ogni azione (*GameCommand*) genera un evento di aggiornamento del Model (*ModelChangeEvent*).

A nostro avviso, il pattern architetturale MVC non è stato correttamente applicato: il Model è canonicamente osservabile dalla View (o RemoteView); nel caso presentato è invece osservabile dal Controller, che è aggiornato a seguito dell'esecuzione di un *ModelChangeEvent* (aggiornamento del model a seguito di un'azione). Non è chiara l'utilità di notificare al controller un cambiamento del Model. Non crediamo efficace la presenza (come parametro) dell'intero Model *GameState* al di fuori della sua presenza del Controller *GameController*, ad esempio in *ModelChangeEvent*. Un'altro dubbio è dovuto all'ereditarietà di alcune classi, in particolare la classe *ServerEvent* è estesa da *Server*, *Lobby* e da *ModelChangeEvent*, pur non avendo alcun attributo o metodo condiviso.

3 Confronto tra le architetture

Un grosso punto in comune tra le due architetture risulta essere la netta divisione tra i messaggi Client-Server e Server-Client, nel nostro caso implementati come *Messages* e *Answers*, rispettivamente la serializzazione delle azioni compiute dal client e gli aggiornamenti del Model a seguito di un'azione. Il nostro progetto separa nettamente le azioni dall'aggiornamento della View da parte del Model, attraverso l'implementazione di opportuni *Listeners*. Anche *ServerCommandAccepter* è in comune: anche noi ricevuto un'azione dal client, la traduciamo nell'equivalente comando, poi controlliamo se è effettivamente eseguibile e, nel caso, la eseguiamo.