Peer-Review 2: UML

Saccani Federico, Spangaro Francesco, Sanvito Luca, Pedersoli Luca

Gruppo **https://github.com/TheFedelino01/ing-sw-2023-saccani-spangaro-sanvito-pedersoli**

Valutazione del diagramma UML parte Networking del gruppo:
https://github.com/LucaParsani/ing-sw-2023-luisi-parsani-simoncini-zanardi

Lati positivi

L'UML contiene le informazioni necessarie allo scopo di rendere comprendibile la strada implementativa scelta per l'implementazione della parte di networking.
Buona la scelta di utilizzare interfacce Client e Server per dividere i vari protagonisti della comunicazione di rete.
Suddivisione in Package accurata e flusso logico da Model a View e viceversa corretto, andando a serializzare oggetti per implementare sia RMI che Socket con lo stesso approccio.
Controller contenente gran numero di metodi utili per gestire le dinamiche di gioco richieste.

Lati negativi

Di seguito alcuni commenti e osservazioni in merito al protocollo di rete e alle informazioni presentate:

L'UML è chiaro e semplice, tuttavia risulta essere poco dettagliato, non contenendo abbastanza informazioni per capire come verranno effettivamente serializzati gli oggetti che si vorranno inviare tra Client e Server e viceversa.

Si vanno a notificare updates solo di Board, Shelf, Current player e Chat, senza andare a controllare e notificare eventi come: L'ingresso di un nuovo giocatore, l'uscita di un giocatore dalla partita, l'inizio della partita, la fine della partita, qualsiasi errore che potrebbe presentarsi dopo una richiesta (prendere una tile vuota, piazzare in una colonna della shelf piena, etc).
Per come è stato presentato, non c'è nessuna informazione in merito a questi eventi.

È stato deciso di implementare l'aggiornamento degli eventi mediante Enum, questo implica che sia lato server che lato client si utilizzano switches per capire, alla ricezione di un messaggio di rete, quale evento si sia verificato. È una soluzione poco scalabile, in quanto all'eventuale aggiunta di un nuovo evento bisognerà modificare sia la classe enum, sia tutto il codice che si occupa di gestire il nuovo valore ricevuto.

Si consiglia di modificare questa implementazione utilizzando il Pattern di rete, frequente e di comodo utilizzo, che va ad incapsulare l'informazione dell'evento accaduto all'interno di classi specifiche.
Saranno quindi serializzate le sottoclassi a seconda dell'evento da notificare, andando quindi a slegare l'effettivo evento dall'enum, ottenendo la massima flessibilità in previsione di future modifiche.

Non sono presenti metodi per andare a individuare le disconnessioni dei clients.

Non è presente un Socket Dispatcher lato server che accoglie tutte le richieste Socket e crea un Thread che si occupa di gestire una singola connessione Socket.

Non è presente l'informazione di quali oggetti verranno resi remoti e quindi di quali metodi synchronized i client potranno invocare remotamente sia lato server sia lato client.

È presente un solo Controller, se si è deciso di implementare anche la funzionalità avanzata "Partite Multiple" sarebbe necessario introdurre due livelli di controller: MainController, con il compito di gestire tutte le partite, e GameController, con il compito di gestire la singola partita (Come bisognerebbe fare con

il Socked Dispatcher mancante: una classe con il compito di gestire ed accettare tutte le connessioni, ed una classe con il compito di gestire la singola connessione con il singolo client).

Confronto tra le architetture

L'idea generale è più o meno la stessa, la differenza principale risiede su come i vari eventi vengono notificati da Client e Server e viceversa.
La nostra scelta implementativa si basa sulla serializzazione di sottoclassi ad hoc, per implementare il Pattern di rete, permettendoci di non rimanere vincolati ad Enum e Switches per la comunicazione in rete di eventi e notifiche.
La loro scelta implementativa si basa sul vincolo di Enum, che è una soluzione non ottimane ma funzionale per la risoluzione del problema presentato.

L'assenza di informazioni su eccezioni, disconnessioni, metodi remoti, oggetti remoti, sincronizzazione e threads utilizzati non permette di segnalare altre differenze sostanziali in merito alle implementazioni delle due architetture.

Entrambe le architetture sono state studiate per implementare correttamente sia la comunicazione RMI che la comunicazione Socket.