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 delle classi del gruppo:  
**https://github.com/LucaParsani/ing-sw-2023-luisi-parsani-simoncini-zanardi**

# Lati positivi

L’UML contiene le informazioni principali iniziali per far capire la strada implementativa del networking che si vuole realizzare.  
Buona la scelta di utilizzare interfacce Server e Client per dividere i vari protagonisti della comunicazione di rete.  
Suddivisione in Package accurata e flusso logico da Model a View e da View a Model corretto andando a Serializzare oggetti per implementare sia RMI sia 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 molto ad alto livello non contenendo abbastanza informazioni per capire come verranno effettivamente serializzati gli oggetti che si vorranno inviare tra Client e Server e viceversa.

Oltre a notificare gli update di Board, Shelf, Current Player e Chat, bisogna andare a notificare anche altri eventi come: L’entrata di un player, l’uscita di un player, la disconnessione, l’inizio della partita, la fine, eventuali errori che si possono presentare dopo ogni richiesta (richiesta di grab di tile non valida, etc.), etc.  
Per come è stato presentato, non c’è nessuna informazione in merito a questi eventi.

Inoltre, è stato deciso di implementare “l’update” degli eventi mediante l’utilizzo di Enum; questo peró implica che sia a lato server che lato client ci siano degli switch per caprie, alla ricezione di un messaggio di rete, quale evento si sia verificato. Di conseguenza, l’introduzione di ulteriori eventi deve corrispondere ad una modifica di codice in quanto bisognerà modificare sia l’enum sia i vari switch degli eventi.  
Si potrebbe utilizzare il Pattern di rete (frequente e di comodo utilizzo) che va ad incapsulare l’informazione dell’evento accaduto dentro a classi specifiche.  
Si serializzeranno quindi le sottoclassi a seconda dell’evento che si vuole notificare andando a slegare l’evento accaduto dall’enum ottenendo la massima flessibilità in visione di modifiche future.

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 decido di implementare anche la funzionalità avanzata “Partite Multiple” (informazione non esplicitata) sarebbe necessario introdurre 2 livelli di controller: MainController che gestisce tutte le partite e di un GameController che gestisce la singola partita.

# 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 è stata quella di serializzare sottoclassi ad hoc per implementare il Pattern di rete permettendoci di non rimanere vincolati ad un enum per la comunicazione di quello che è accaduto.  
La loro scelta implementativa è stata quella di utilizzare un enum, che risulta essere comunque una buona soluzione per risolvere il 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 sia la comunicazione Socket.