Peer-Review 2: Controller e Network

Giacomo Saracino, Alessandro Trimarchi, Diego Zafferani, Francesco Zullo

Gruppo 28

Valutazione dei Controller e Network del gruppo 18.

Lati positivi

- La suddivisione MVC è corretta secondo le regole di definizione
- la presenza di interfacce tra client e controller è una buona scelta in quanto facilita la comunicazione anche attraverso canali diversi come socket o rmi
- La presenza di listener aiuta la gestione dei messaggi, potrebbe essere comodo implementare un gestore di messaggi tramite queue
- la presenza di messaggi specializzati per ogni azione è un'idea adeguata soprattutto per quanto riguarda la scalabilità e l'implementazione di funzioni modularmente
- Molto buona la separazione tra MainController e GameController, per distinguere le interazioni pre partita e durante la partita.

Lati negativi e Suggerimenti

- Manca nell'UML la definizione di GameImmutable con tutti i suoi attributi. E' una classe abbastanza importante, in particolare per capire quali informazioni vengono passate al client.
- Esaustività dei messaggi C-S e S-C: Messaggi per comunicare nickname duplicati: viene prima scritto un nickname, si attende se è unico, poi avviene la richiesta di entrare. Potrebbe essere gestito con un unico messaggio di join che controlla e restituisce eventualmente una exception in un messaggio di ritorno

  Messaggio di leaveGame non è una cosa richiesta dalla specifica.

  Messaggi di errore specifici per ogni azione nelle comunicazione S2C, potrebbe essere meglio generare un unico messaggio di errore che contiene informazioni specifiche piuttosto che molti messaggi da gestire in modo autonomo.

  presenza di molti messaggi S2C non di interesse per il client, dovrebbero essere gestiti e comunicati solo con gamecontroller

  messaggi a mio avviso di troppo in C2S isThisNickTaken, LeveGame, GetGamesStarted, IsMyTurn. Penso siano di utilità solo per il game controller possano essere gestiti da esso senza comunicazione con client

messaggi a mio avviso di troppo in S2C : PreparationGame, GoalCardsNotChoosenYet, NotYourTurn, YourTurn, AddPoints, LastTurn, NotEnoughPlayers.

- Comunicazione modifiche della board tra giocatori: prima di tutto occorre passare dal server, ogni client quando è il proprio turno invia dei messaggi al server, che processa l'azione. Suggeriamo di creare (alcuni sono già presenti) dei messaggi server to client che inviano il nuovo stato della partita a tutti i client.

  Ad esempio, il messaggio msgCardPlaced può essere adattato per fare in modo che nel GameImmutable ci siano le board di tutti i giocatori. In sostanza, pensiamo che dal GameImmutable si debba poter ricostruire lo stato della partita.

  Forse il messaggio era stato pensato per notificare solo il giocatore che ha fatto la mossa, ma cosi è più difficile tenere i client aggiornati. Pensiamo sia meglio passare a tutti i giocatori l'informazione "Giocatore X ha giocato carta Y in posizione Z,W".

  Comunque, sul client sai quale giocatore sei, vale a dire puoi loggare ad esempio messaggi in console a seconda che il giocatore X sei tu o no.

- Controllo delle fasi della partita: questo punto si collega molto al precedente. Si può pensare nel gameImmutable di avere un attributo actionExpected (enum) e un attributo currentPlayer (nome del giocatore che deve fare la mossa actionExpected). In questo modo, dopo ogni mossa, ogni client viene notificato con il nuovo stato della partita, e grazie ai due campi appena detti, è possibile sapere chi deve giocare e quale azione deve eseguire.

  Siccome in questa maniera, tutti i client vengono notificati dopo una mossa, e ogni giocatore sa chi deve giocare e che mossa deve fare, alcuni messaggi come msgNotYourTurn, msgYourTurn, msgLastTurn etc… perdono senso. Il model ha inserito tale informazione dentro il GameImmutable.

  Si può inoltre pensare di creare un messaggio di errore generico che quando ricevuto sul client logga qualcosa in console (se modalità TUI) o mostra qualcosa a schermo (se modalità GUI).

- Non ci è chiarissimo nella frase *"se il controllo dello stato del giocatore debba essere sul model del server o possa essere gestito dal client e notificato successivamente"* cosa intendiate per stato del giocatore. Se vi riferite a come fare a far sapere a ogni giocatore quale mossa deve fare, la risposta del punto precedente dovrebbe rispondere a tale domanda. Se vi riferite anche alla validazione della mossa in sè dopo che l'utente l'ha eseguita (ad esempio l'utente ha scritto un messaggio in console per giocare una carta), pensiamo che la validazione sia meglio eseguirla sul server, che poi notificherà un messaggio di errore al client che l'ha eseguita se non era il suo turno ad esempio. Se invece l'azione è andata a buon fine, tutti i client saranno notificati con un nuovo stato della partita.

Confronto tra le architetture

Le strutture di entrambi i progetti hanno una organizzazione simile per quanti riguarda il modello MVC, è presente un ListenerClient che si relaziona tra ClientController e ClientSocket/RmiClient nell'architettura del gruppo 18, all'interno del nostro progetto l'interfaccia di interrelazione tra i client server e il network è composta tra VirtualView e VirtualServer.

Nel nostro progetto sia RmiClient, che il corrispettivo TCP, implementano VirtualClient, che ha una singola funzione di sendMessage(Message S2C). Nel caso RMI il server chiamerà il metodo sendMessage, con un messaggio creato sul server, sullo "skeleton" del client. Il metodo in questione pone il messaggio in una queue di messaggi nel client che successivamente è interrogata da un thread.

Nel caso TCP il clientHandler implementa il sendMessage inviando un messaggio sul canale TCP. Il messaggio ricevuto sul client è posto nella stessa queue.

Facciamo qualcosa di analogo per messaggi C2S. Il client RMI chiama un metodo sendMessage(Message C2S) sullo stub del server che aggiunge il messaggio in una queue. Il Client TCP crea un messaggio C2S e lo invia sul canale TCP. Nella ricezione è posto in una queue.

Non pensiamo che il nostro gruppo implementerà la FA delle disconnessioni dei client, tuttavia riteniamo buono l'approccio di usare dei messaggi per notificare a intervalli di tempo regolari, che il client è ancora connesso.