Peer-Review 2 - UML di Rete

Restelli Matilde, Salvi Niccolò, Scianna Marco, Villa Alessio

Gruppo 38

POLITECNICO
MILANO 1863

A.S. 2023-2024

Valutazione del diagramma UML di rete del gruppo 28

Lati Positivi

- La gestione unificata della parte di rete indipendentemente dall'implementazione RMI o Socket rende il processo totalmente trasparente al client.
  Riteniamo un'ottima soluzione rendere RMI asincrono per eventuali deadlock e rallentamenti nel flusso di gioco.

- L'utilizzo di una gerarchia fissa di messaggi che vengono costruiti sulla base dell'input corrispondente.
  Rende il processo di gestione dell'input più semplice visto che ogni messaggio ha la sua tipologia di dati attesi si potrà poi facilmente controllare eventuali errori con l'utilizzo delle eccezioni.

- L'implementazione di due rispettivi code (queue) per le due differenti tipologie di messaggi. Questa intuizione rende il flusso di comunicazione più ordinato e facile da gestire.

- La gestione da parte del controller di ogni singola partita della notifica ai client successivamente ad un'azione che ha svolto sul model piuttosto che far svolgere quest'azione al controller generale (GamesManager)

Lati Negativi

- L'implementazione di un ulteriore controller GamesManager che gestisce tutte le partite rende lievemente tortuoso il percorso per la chiamata effettiva di un metodo che modofichi il model della partita.
  Inoltre questa soluzione porta ad aspettare in coda tutti i messaggi rivolti a tutte le partite, quando altrimenti due messaggi rivolti a due partire (e quindi due controller diversi) potrebbero benissimo invece essere eseguiti in parallelo.

- L'inserimento dell'ID del controller dentro ogni messaggio che viene scambiato rende ripetitiva e parametrica una dipendenza del flusso di comunicazione che invece riteniamo abbastanza fondamentale.
  Ripetere lo stesso attributo in ogni messaggio, passarlo nella rete e poi smistare i messaggi in base ad esso rende probabilmente più vulnerabile e soggetta a possibili interferenze la connessione principale tra ogni client e il controller esatto della partita a cui sta giocando.

- La mancanza di un'entità fissa che colleghi ogni client con il controller della partita a cui sta giocando (e relativamente anche gli altri client connessi ad essa).
  Riteniamo che, per esempio, la possibile disconnessione di un client da una partita o altri eventi che coinvolgono tutti i client collegati ad uno stesso GameController potrebbero essere gestiti più facilmente con l'utilizzo di una classe che colleghi queste dipendenze tra loro.

Confronto tra le architetture

- La differenza principale tra le nostre architetture è semplicemente un puro gusto soggettivo su come gestire gli update dello stato del gioco:
  - il gruppo 28 inoltra ai client l'intero stato del gioco modificato (il GameState) evitando così che essi abbiano dei dati del gioco memorizzato, ma così passando un elevato quantitativo di informazioni ogni volta
  - il nostro gruppo ha invece optato per la realizzazione di un ViewModel (una versione molto sintetizzata del model) che contiene solamente i dati strettamente necessari per la partita e con cui il giocatore interagisce direttamente (tramite la view) durante il gioco. Ad ogni update dello stato del gioco vengono inoltrate ai client solamente le modifiche effettuate, che verranno integrate nel ViewModel da loro posseduto.

- La gestione delle dipendenze tra i client e il Game Controller della partita a cui sono collegati è un altro oggetto di implementazione differente da parte del nostro gruppo. Abbiamo infatti realizzato una classe apposita (nel package di rete) che colleghi tutti i clien partecipanti ad una partita con il relativo GameController.

  In questo modo i client comunicano tutti con lo stesso server solo ed unicamente prima dell'inizio della partita per instaurare le connessioni, successivamente però ogni partita è gestita in modo indipendente ed esclusivo, divindendo così le comunicazioni in parallelo.