Peer-Review 2: UML

Gianmarco Schifone, Alessia Beatrice Viani, Leonardo Vicentini, Andrea Zhang

Gruppo 29

https://github.com/leonardovicentini/ing-sw-2024-schifone-viani-vicentini-zhang.git

Valutazione del diagramma UML delle classi del gruppo 19.
https://github.com/federicovilla55/ing-sw-2024-sood-villa-vitali-zanzottera

Lati positivi

1. Il gruppo 19 ha un'ottima implementazione della resilienza alle connessioni grazie all'impiego di un token segreto. Inoltre, abbinato all'utilizzo del "cookie" che viene salvato localmente in un file JSON read-only che si mantiene finché l'utente non si disconnette garantisce un ulteriore livello di robustezza della resilienza.

2. Sebbene non sia stato specificato nella relazione, abbiamo ritenuto che l'utilizzo del token potrebbe essere un'ottima soluzione per permettere ad utenti di partite diverse di utilizzare lo stesso nickname.

3. La suddivisione tra client anonimi e client configurabili è un'ottima implementazione per ridurre il carico sulla rete e allo stesso tempo mantenere un pattern che garantisca l'esistenza di observer che ricevono tutte le notifiche.

4. Ottima suddivisione in interfacce, che permette di separare parti di logiche diverse.

Lati negativi

1. "*L'interfaccia VirtualClient rappresenta il proxy del client visto dal server ed ha un unico metodo pushUpdate(MessageToClient)*". In riferimento a questo paragrafo, sebbene RMI possa essere utilizzato per invocare metodi e trasferire dati tra server e client in modo simile a TCP, potrebbe non essere altrettanto efficiente o adatto a tutti i casi d'uso. RMI è più orientato all'invocazione di metodi tra oggetti distribuiti, il suo scopo è permettere a un'applicazione Java di comunicare con un'altra applicazione Java in modo trasparente come se fossero entrambe eseguite sulla stessa macchina.

2. "*Per gestire i messaggi è stato utilizzato il pattern Publisher - Subscriber. I Publisher sono tutte le classi che possono generare messaggi (parte del modello e il GameController)*". Forse non abbiamo compreso appieno l'utilizzo, ma è importante sottolineare che il controller non dovrebbe generare messaggi; questa responsabilità dovrebbe invece essere affidata al modello.

3. I sequence diagram che ci sono stati forniti mostrano principalmente i casi di costruzione, riconnessione e disconnessione. Tuttavia, manca un dettaglio importante riguardante il ciclo di comunicazione tra client e server attraverso le componenti View, Controller e Model.

Questo ciclo dovrebbe includere il modo in cui la View invia richieste al Controller, il Controller comunica con il Model per ottenere o aggiornare i dati e infine il Model notifica la View dei cambiamenti per aggiornare l'interfaccia utente. Integrare questa parte mancante nel sequence diagram fornirebbe una visione più completa e chiara del processo di comunicazione e interazione tra le diverse componenti del gioco.

Confronto tra le architetture

1. Nella nostra architettura di rete, abbiamo adottato un approccio unificato per implementare sia il protocollo socket che il protocollo RMI. Questo approccio è stato progettato tenendo conto dell'implementazione sottostante di RMI e del suo funzionamento interno, anche se questi dettagli sono mascherati all'utente finale. Per implementare il protocollo TCP, abbiamo sviluppato un middleware che opera come intermediario tra il client e il server. Questo middleware include un ClientSkeleton e un ServerStub che si scambiano messaggi utilizzando buffer dedicati. Ad esempio, per quanto riguarda la comunicazione dal server al client, il ClientSkeleton si occupa di inviare le richieste dal server al client tramite la connessione TCP, mentre il ServerStub riceve queste richieste e le inoltra ai metodi appropriati sul client.

2. Per gestire i messaggi tramite socket, anziché adottare un pattern di Visitor, abbiamo optato per un approccio basato su interfacce. Abbiamo definito le interfacce MessageToServer e MessageToClient, e ciascun tipo di messaggio le implementa. Ogni implementazione contiene un metodo run che si occupa di instradare il messaggio al giusto metodo sul server.

3. Nella nostra architettura di rete, abbiamo scelto di inviare un'intera vista immutabile del modello al client per semplificare il processo di progettazione. Tuttavia, siamo consapevoli del fatto che esistono alternative più ottimali, come la vostra, per ridurre il carico di dati scambiati sulla rete.