Peer-Review 2: Network

<Agostino Contemi>, <Federico Consorte>, <Filippo Dodi>, <Jacopo Corsi>
Gruppo <AM04>

Valutazione del diagramma UML delle classi del gruppo <AM13 >.

Lati positivi

Implementare il design pattern Observer consente una gestione efficiente degli aggiornamenti dello stato della partita e delle View dei giocatori. Questo approccio favorisce la modularità e la manutenibilità del codice, consentendo una separazione chiara tra la logica di business e l'interfaccia utente.

L'utilizzo di un'interfaccia per definire i metodi update() offre flessibilità nella gestione degli aggiornamenti dello stato della partita sul client e nella loro visualizzazione. Questo permette una personalizzazione più precisa delle View dei giocatori in base alle loro esigenze specifiche.

Il ServerMain rimane in ascolto della connessione di nuovi client, il che consente una gestione efficiente delle richieste di connessione in arrivo. Il server in questo modo gestisce simultaneamente più client senza incorrere in problemi di sovraccarico.

ListenerHandler si occupa della trasmissione in broadcast dei messaggi dal server al client permettendo la distribuzione delle informazioni a tutti i giocatori interessati, riducendo il carico sul server.

La distinzione tra messaggi command e messaggi result fornisce una struttura chiara e ben definita per la comunicazione tra server e client.

Lati negativi

L'eccezione per la richiesta getRooms(), che viene gestita in modo diverso dalle altre richieste, può aumentare la complessità del codice e della logica di gestione delle richieste.

Se i messaggi command e result sono molto dettagliati o se venissero scambiati frequentemente, potrebbe verificarsi un overhead di rete significativi, causando ritardi nella trasmissione dei messaggi.

Poiché i messaggi result vengono inviati a tutti i client partecipanti, c'è un rischio di errori di sincronizzazione se la gestione delle operazioni non è attentamente coordinata tra server e client.

Confronto tra le architetture

Le architetture presentano alcuni concetti comuni ma abbiamo rilevato varie differenze. Per quanto riguarda i messaggi abbiamo utilizzato approcci simili, cioè una divisione di messaggi che vengono mandati da client a server e viceversa. Tuttavia abbiamo preferito non inviare tutti i messaggi in broadcast, poiché alcuni possono essere inviati ad un singolo client anziché tutti, ad esempio è superfluo inviare le informazioni delle carte obiettivo segrete di tutti i giocatori a tutti i client.

Per ridurre l'overhead di rete, si potrebbero implementare meccanismi di caching per memorizzare temporaneamente le risposte alle richieste frequenti e evitarne la ritrasmissione se non sono state apportate

modifiche allo stato di gioco. Per il resto le architetture risultano abbastanza simili.