Peer-Review 2

Petrucci, Ponginibbio, Secci, Passoni

Gruppo 65

Peer review parte di rete/protocollo del gruppo 10.

Lati positivi

→ Adozione del pattern Factory Method (GameHandlerBuilder) per istanziare correttamente la classe GameHandler.
→ Progettazione di due classi distinte (GameController e TableController) per la parte di controllo.
→ Progettazione di due sottoclassi (ExpertGameController e ExpertTableController) per la parte di controllo per la funzionalità esperto.
→ Progettazione di una classe (Endpoint) che si occupi della comunicazione nel Client.
→ Le classi del Model estendono la classe Observable: in questo modo possono notificare (direttamente) il Client, aggiornandone la View (change of state).

Lati negativi

→ Nell'interfaccia StandardEffect i metodi activateEffect() e reverseEffect() hanno come secondo parametro un array di Object in quanto ogni effetto ha parametri diversi. Secondo la nostra opinione questo rende l'interfaccia meno descrittiva (non è possibile sapere quali siano i parametri da passare guardando l'intestazione della funzione), inoltre eventuali errori di programmazione nell'uso dell'interfaccia, come passaggio di parametri di tipo sbagliato, non saranno rilevabili dal compilatore poiché tutti gli oggetti sono compatibili col tipo Object.
→ Nel diagramma UML la classe VirtualView implementa l'interfaccia ModelObserver, che dovrebbe invece essere implementata dalla classe GameController, stando a quanto scritto nella descrizione.

Confronto tra le architetture

→ L'architettura in oggetto tiene traccia dello stato attuale dell'utente, che può essere online o offline. Si tratta di una componente fondamentale per l'implementazione della funzionalità di persistenza, in quanto la partita, una volta sospesa, potrà essere continuata solamente se tutti i suoi giocatori sono connessi. Ciò significa che nuovi utenti non potranno adottare un nickname uguale a quello di un utente al momento disconnesso, ma partecipe di una partita interrotta.
La nostra architettura invece non si deve preoccupare di conoscere lo stato corrente dell'utente, poiché non implementa tale funzionalità.
→ L'architettura in oggetto adotta la serializzazione standard di java per lo scambio di messaggi, mentre la nostra utilizza il formato JSON.
→ Entrambe le architetture suddividono il Controller in due sottoparti: una si occupa della gestione del Server, l'altra della logica di gioco.
→ Entrambe le architetture sono provviste di un game-handler, in quanto supportano la simultaneità delle partite.

→ L’architettura in oggetto suddivide la componente del Controller incaricata della gestione della logica di gioco in due sottoparti, GameController e TableController. La nostra invece non vede tale suddivisione.
→ Le architetture a confronto implementano funzionalità differenti, di conseguenza dovranno gestire casi di gioco non analoghi. L’architettura in oggetto garantisce la persistenza delle partite, pertanto dovrà interfacciarsi a problematiche prevalentemente a livello di rete. La nostra invece è più ricca in termini di componenti di gioco, in quanto definisce tutte le carte personaggio, dunque la logica di gioco sarà più complessa e impegnativa da gestire.

Osservazioni

→ I termini “observer” e “listener” sono sinonimi. In linea puramente teorica, il pattern Observer, adottato per la comunicazione tra Model, View e Controller, e la strategia di Listener per la parte di rete, sono equivalenti.
→ In caso di disconnessione (forzata) di un giocatore a partita iniziata, si procede etichettando tale partita come sospesa (ne viene salvato lo stato) oppure conclusa?