Peer-Review 2: Server

Valutazione del diagramma UML delle classi del gruppo GC10

Letizia Maria Chiara Torre, Riccardo Speroni
Vlad Alexandru Robu, Davide Vola

Gruppo GC20

9 maggio 2023

Indice

1 Lati positivi

L'utilizzo di un'enumerazione per tenere traccia dello stato del singolo giocatore è senza dubbio utile (*), cosí come l'utilizzo della classe `Token` per la gestione dei punti delle carte obiettivo comune.

La scelta di interpellare il Server solo per la fase di setup della connessione può semplificare la comunicazione tra Controller e View del Client, il che é senza dubbio un vantaggio.

L'intero MVC sembra completo ed organizzato in modo chiaro ed efficace.

2 Lati negativi

2.1 Invio delle tessere scelte

Si può evitare di mandare le `Tiles` avanti e indietro tra il Server implementando il parsing dei comandi lato Client: dall'UML infatti si può presumere che il parsing sia incorporato sparsamente dal lato del Model, mentre nel sequence diagram è apparente che il Server rimandi le tessere indietro, e questo non è necessario; basterebbe che il Server rispondesse indicando se l'azione è andata a buon fine (es: se le `Tiles` selezionate sono state correttamente inserite).

Anche rimandare indietro le `Tiles` per mostrarle a schermo non sarebbe troppo sensato: nel momento in cui il Client sceglie le tessere, queste dovrebbero essere già evidenziate dalla GUI (ad esempio) e dovrebbero restare selezionate finché non arriva la risposta del Server (es: le tessere sono invalide, si mostra sullo schermo il messaggio di errore).

Invece che mandare le `Tiles` può avere senso mandare delle coordinate, così da avere un riferimento chiaro rispetto a cosa stia scegliendo il Client. Dall'UML sembra che i Client mandino una lista di istanze, il che per il Server non ha molto significato visto che il Server non può sapere esattamente a quali tessere si stiano riferendo.

2.2 Client

Non é necessario che il Client chieda al Server `iAmFirst`: quest'informazione può essere inviata dal Server stesso quando inizia la partita, ma in ogni caso al Client non interessa se è primo, bensì solo se è il suo turno o meno.

(*) Gli stati peró sembrano indicare solo quando il Client sta aspettando il proprio turno (`READY`) e quando sta giocando (`POP`), e non è chiara l'utilità

di questa scelta: avrebbe più senso sapere a che punto si trova il Client nel turno, cioè se sta scegliendo le tessere, se le sta riordinando, o se le sta inserendo nella colonna.

2.3 Gestione dei Token

Passare il `Token` ai Client non è ottimale, in quanto basterebbe mandare un boolean che indica se il Client ha preso o meno il punto in più. Ci penserà poi la View locale a mostrare a schermo il token; di conseguenza salvare il `Token` nella `Shelf` non è indispensabile.

2.4 Sequence diagram

Nel sequence diagram manca il metodo `ReceiveUpdateShelf(ShelfView)` per aggiornare la View di ogni Client dopo ogni mossa.

Alla fine della partita viene inviato ai Client solo il nome del vincitore, mentre potrebbe essere un'idea inviare una classifica con i punteggi di ogni giocatore, ma non si tratta necessariamente di una cattiva scelta di implementazione del gioco.

3 Confronto tra le architetture

L'utilizzo di stati per identificare a che punto del turno è il giocatore farebbe comodo anche alla nostra implementazione del gioco, cosí da poter permettere al Server di tenere traccia del Client, anche nel caso in cui quest'ultimo annulla un'azione durante il suo turno.

La suddivisione della `ModelView` in più parti permetterebbe di aggiornare le View dei Client mandando solo le informazioni strettamente necessarie, invece che l'intera visualizzazione del model, il che potrebbe semplificare la gestione degli update.

L'utilizzo di una classe `Game` nel Model potrebbe esserci utile per una gestione delle View più facile e separarla dalla logica di gioco, che resterebbe al `GameController`.