Commento a UML

Prova finale di ingegneria del sw a.a. 23-24

Oltre alla rappresentazione UML, alleghiamo un breve commento sui metodi utilizzati e alcune scelte effettuate.
Ci teniamo a sottolineare che, per alleggerire la lettura di un grafo già corposo, sono state effettuate le seguenti modifiche:

1. Rimozione delle classi di stampa delle carte per la CLI
2. È stata tenuta soltanto la classe astratta di messaggio, invece che allegare tutti i tipi di messaggio.

Non forniamo una descrizione del model in quanto è rimasto invariato dalla precedente peer-rewiew. L'unica modifica che è stata sviluppata è l'aggiunta di una classe ModelEventProvider che implementa un pattern Observable-Observer. Questo pattern ci permette di generare degli eventi nel model che saranno utilizzati come trigger per svolgere una serie di azioni in cascata (p.e. quando viene piazzata una carta che ha un punteggio viene generato un UpadatePlateauScoreEvent che aggiornerà tutti gli osservatori di quell'evento che a loro volta eseguiranno le azioni previste).

Descrizione controller

Il package controller contiene due classi principali:

- *MainController:* classe singleton che gestisce tutti i giochi in corso, permette di creare un gioco, entrare nella prima partita disponibile o entrare in una partita a scelta (inserendo il gameID). I giochi e i rispettivi gameID sono gestiti grazie all'utilizzo di una HashMap, dove le chiavi sono gli ID dei vari giochi, e gli oggetti sono di tipo GameController.
- *GameController:* classe che permette di gestire tutti i metodi che permettono di avere accesso al model, e quindi di permettere il proseguimento di tutta la logica di gioco.

Rete

Per implementare la rete, abbiamo scelto di creare due interfacce: *VirtualServer* e *VirtualClient*, comuni sia per RMI che per Socket in modo tale da rendere "trasparente" la scelta di rete al client ed avere gli stessi metodi disponibili per entrambi.

La classe *ClientController* instrada le chiamate dei vari metodi da parte del client al server corretto in base al tipo di connessione scelta.

1. RMI

L'implementazione RMI è composta da una classe *Server*, che implementa l'interfaccia *VirtualServer.* I metodi sono quelli che permettono di effettuare le azioni sul gioco, Server è quindi la classe che chiama i metodi di MainController, ed esegue le modifiche al model quando il client compie un'azione (piazzare una carta, pescare da uno dei due mazzi...).

Server, inoltre, permette di inviare degli update a uno o più client in base all'esigenza, al verificarsi di un cambiamento.
L'altra classe è *RMIClient*, che implementa invece *VirtualClient,* interfaccia che permette al client di ricevere gli update dal server e salvarli in una classe ViewModel. Il client successivamente accederà direttamente al ViewModel per recuperare le informazioni necessarie per le varie azioni.

Attualmente le chiamate al Server RMI vengono effettuate direttamente dal client, ma stiamo modificando proprio ora in modo tale da effettuare le chiamate attraverso dei messaggi inviati da parte del client.

2. Socket

L'implementazione Socket risulta invece leggermente più complessa.
La classe *SocketServer* crea il server, e accetta gli accessi dei diversi client creando un *SocketClientHandler* per ciascuno di loro.
La classe *SocketClientHandler* è quella che gestisce l'arrivo e l'invio di messaggi da parte del client, possiede infatti un canale ObjectInputStream e un canale ObjectOutputStream.
Invece la classe *SocketClient* permette di inviare e attendere arrivo dei messaggi da parte dell'handler.
I messaggi sono racchiusi in due package fondamentali:

- *SocketMessages*: messaggi da client a server, implementano tutti l'interfaccia *ISocketMessage* che possiede un solo metodo execute() che ritorna Object.
- Messages: messaggi da server a client, implementano tutti l'interfaccia *IMessage*, che possiede un metodo void execute().

Ogni messaggio, avrà una sua implementazione personale del metodo execute(), permettendo di gestire ogni tipo di messaggio con le rispettive azioni conseguenti.

3. Sequence Diagram

Ecco un esempio di comunicazione RMI tra client e server. Come già detto RMI chiama direttamente i metodi del server ma verranno sostituiti da messaggi.