Peer-Review 2: Rete

<Matteo Panarotto>, <Maria Sole Rossi>, <Cristian Pasinetti>,<Daniel Straface>

Gruppo <GC-48>

Valutazione dell’implementazione UML di rete del gruppo <GC-38>.

# Lati positivi

* Le classi relative a Client e Server sono molto compatte e hanno rispettivamente quattro e tre metodi, evitando ridondanze eccessive e mantenendo pulita la struttura dell’UML.
* Gestione semplice ma incisiva della registrazione dei Client dei giocatori sul Server.
* Ottima gestione del flusso di chiamate per reperire il nome del giocatore, inizializzazione del Server e registrazione del primo Client (chiamata a Server.register(this)).
* La creazione del controller e aggiunta dell’osservatore ben implementata sia nel caso del primo giocatore sia nel caso dei restanti.
* L’utilizzo delle interfacce ClientRMInterface e ServerRMInterface permette di disaccoppiare i metodi relativi alla parte di rete da ciò che compete le classi Client e ServerImpl.

# Lati negativi

* Mancanza di coerenza circa le implementazioni delle interfacce ClientRMInterface e ServerRMInterface dal momento che la prima si chiama Client e la seconda ServerImpl.
* Non è stato mostrato né sul sequence diagram né sulla descrizione testuale l’ordine delle chiamate lato Server e lato Client relative alla dinamica di un turno (scelta delle tessere e il loro ordinamento, scelta della colonna in cui inserire come notificano il Server e come questo modifichi il model).
* Non è stato mostrato né sul sequence diagram né sulla descrizione testuale l’ordine delle chiamate da Server a Client per la gestione di fine turno (come il modello deve notificare i Client le modifiche occorse e il passaggio del turno).
* Poca chiarezza del sequence diagram circa la sequenza di chiamate:

1. Client.startGame()
2. View.numberOfPlayers()
3. ReturnNumberOfPlayers

Queste chiamate vengono effettuate prima della chiamata Game: new Game(client.startGame()) in quanto la chiamata 1. è il parametro del costruttore di Game.

* La chiamata a startGame() nel Server come notifica i Client e come detta il primo giocatore non è indicato nella documentazione fornita sia per iscritto che per diagramma.
* L’utilizzo delle eccezioni nel controller posto Server-side può generare conflitti in merito al pattern Observer/Observable dal momento che potrebbe interrompere la catena dell’MVC. Sarebbe consigliabile aggiungere nell’enumerazione degli stati del modello un ulteriore stato di errore in modo che il Server notifichi i Client chiudendo correttamente il giro del pattern oppure eseguire il controllo Client-side.

# Confronto tra le architetture

* Anche noi abbiamo agito allo stesso modo riguardo la registrazione dei Client alla singola partita con conseguente creazione del modello, del controller e della notifica a schermo dell’attesa di nuovi giocatori.
* A differenza nostra, il gruppo revisionato ha prima registrato il Client del primo giocatore e poi ha chiesto lui il numero dei giocatori della partita anziché passare come parametro al metodo register non solo il Client da registrare ma anche il numero di giocatori della partita e il nickname scelto dal giocatore.
* Vi è presente un’enumerazione che rappresenta lo stato del modello, scelta da noi non adottata.
* Sono stati implementati da noi l’architettura di rete mediante l’utilizzo di Socket e la possibilità di gestire multiple partite nel Server contemporaneamente. Ciò comporta una differenza nei modi in cui vengono chieste all’utente informazioni oppure registrati i Client sul Server: abbiamo diviso il Server in due classi, la prima che gestisce la registrazione degli utenti e lo smistamento fra le varie partite e la seconda che si occupa invece di gestire la singola partita e di portarla a compimento.