Peer-Review 2:
Architettura di Rete

Viola Renne, Sara Resta, Laura Puccioni
Gruppo GC16

6 maggio

1 Lati positivi

Il gruppo CG26 utilizza correttamente la programmazione ad eventi per l'implementazione del lato server. Infatti, il server non manda continui messaggi al client, ma resta passivo aspettando che siano gli utenti a svolgere un'azione. Il gruppo ha iniziato a identificare alcuni dei messaggi che si dovranno scambiare server e client e questa idea è corretta, ma di seguito si trovano alcuni consigli per quanto riguarda la loro implementazione. Seppur il nostro gruppo non abbia utilizzato i listener, riteniamo che il loro utilizzo sia corretto per quanto riguarda la GameHandler che permette di capire se all'interno dell'ascoltatore esiste un metodo in grado di ricevere in ingresso l'evento chiamato e, in caso positivo, chiamarlo. L'utilizzo della "reflection" è apprezzato nel contesto della programmazione ad oggetti, piuttosto dell'utilizzo di uno switch.

1.1 Controller

Per quanto riguarda la programmazione a oggetti, l'utilizzo di overloading nel metodo update della classe Controller, permette di evitare l'uso del costrutto switch o dell'istruzione instanceof.

1.2 Network

Client Riteniamo corretto l'utilizzo dei due thread, uno utilizzato per la lettura e uno per la scrittura.

RemoteView Nella RemoteView è presente un riferimento ad un oggetto di tipo ClientSocket e un metodo run quindi per ogni client connesso viene creato un thread.

1.3 View

View Per quanto riguarda la programmazione a oggetti, l'idea di utilizzare overloading, nel metodo update della classe View, invece di uno switch o instanceOf, è migliore. Inoltre, è corretto che la View sia astratta e che venga implementata dalla classe CliView e dalla classi GuiView.

1.4 Common

Visto che il gruppo non implementa la funzionalità aggiuntiva multipartita, l'utilizzo del pattern Singleton per la classe GamHandler è adatto.

2 Lati negativi

2.1 Controller

Nella classe Controller sono presenti 7 metodi identici tra di loro (stessa interfaccia). Supponiamo che si volessero diversificare i diversi tipi di update fornendo come parametro un diverso tipo di event per ciascuno in modo da sfruttare l'overloading. E' corretto inserire la gestione dei controlli dei messaggi prima di chiamare un metodo di Game, ma sarebbe più coerente inserire tutti i controlli nel Controller e non parte nel Controller e parte nel Model. Ad esempio, nel momento in cui un giocatore prova a giocare una carta assistente si dovrà controllare la fase del gioco, che sia effettivamente il turno del giocatore e che nessuno prima di lui abbia utilizzato quella carta. Questi controlli non sono stati inseriti nel Controller, dunque deduciamo che siano nel Model e di conseguenza sarebbe più coerente inserirli tutti nello stesso punto. Inoltre, riteniamo che l'attributo hasCardBeenUsed dovrebbe essere inserito in ConcreteExpertGame. Infine, manca un attributo che contenga i giocatori connessi oppure i giocatori disconnessi. Infatti, il controller, nel momento in cui un giocatore si disconnette, dovrà far saltare i turni a quel giocatore e gestire la sua disconnessione.

Aggiornamento Il gruppo CG26 ci ha detto che i vari controlli sono stati implementati all'interno di update e che dunque richiama alcuni tra i metodi pubblici "check". Innanzitutto, i metodi "check" potrebbero essere privati perchè richiamati solamente dai metodi update. Inoltre, riteniamo che sia più corretto inserire i controlli in metodi separati dai metodi update che invece dovrebbero gestire solamente i messaggi.

2.2 Network

Client Non riteniamo sia necessario avere un buffer in ingresso nella classe SocketReader, in quanto il metodo readObject è bloccante.

2.3 View

ViewData In ViewData sono contenute tutte le informazioni da mostrare all'utente relative alla partita. L'attributo tiePlayers è la lista di coloro che hanno pareggiato. Questa informazione però è solo presente lato client e non

lato server. Dal punto di vista dell'efficienza non riteniamo ottimale inviare in rete il Player, visto che contiene anche i riferimenti di SchoolBoard e Deck e dato che in ViewData non sono presenti SchoolBoard e Deck, andrebbero mandati anche quelli sovraccaricando la rete. Infine, manca la rappresentazione della carta assistente corrente, non solo nella ViewData, ma anche nel Model. Sicuramente è un valore importante sia da tenere sia da mostrare, in modo che i giocatori sappiano i turni di quella partita. Inoltre, ai fini della strategia di un giocatore, è utile sapere quali carte hanno giocato gli altri giocatori per selezionare una carta che ci permetta di iniziare per primi (o per ultimi se la nostra strategia prevede ciò). La carta assistente corrente mostra anche il numero di step che madre natura può compiere e l'utente dovrebbe poter vedere questo valore.

2.4 Common

GameEvent

- L'evento di tipo UpdateDataEvent dovrebbe aggiornare la View in base alle modifiche apportate dal Server ma non ci è chiaro dove venga istanziato l'oggetto di tipo ViewData presente tra gli attributi di questa classe. Inoltre, dal punto di vista dell'efficienza non ci sembra ottimale inviare in rete l'oggetto DataView per ogni modifica effettuata dal server.
- Tra gli eventi ne mancano alcuni tra cui:
  - Evento che permetta di settare il mago scelto. Inoltre, questa informazione sembra mancare anche nel model.
  - Evento che indica l'inizio della partita (nel momento in cui tutti gli utenti sono connessi e la partita inizia, questo va notificato a tutti i giocatori).

3 Confronto tra le architetture

L'architettura del nostro gruppo è piuttosto diversa, in quanto abbiamo deciso di implementare le partite multiple e di conseguenza abbiamo una Lobby-Controller che contiene le informazioni dei client connessi, sia che essi siano inseriti in una partita, sia che essi siano in attesa di essere inseriti. Questo ci permette di gestire facilmente la scelta dei nicknames, dei maghi e, per il primo giocatore connesso, il numero di giocatori nella partita e la modalità del gioco. Tutti gli utenti che giocheranno nella stessa partita saranno all'interno della stessa Lobby. Inoltre, al posto di utilizzare i listeners, abbiamo implementano noi due classi: Observer e Observable. Infine, abbiamo separato i vari messaggi, quindi non abbiamo solamente una classe 'GameEvent' la quale viene implementata da tutti gli altri messaggi (o eventi), ma abbiamo preferito creare alcune sottoclassi per diversificare i messaggi lato client o lato server e la loro funzionalità (di gioco, di settaggio della partita, ...). Ci siamo accorte di alcune cose che potremmo migliorare nella nostra architettura e le elenchiamo di seguito

Professor Il nostro gruppo non aveva pensato a come mostrare i professori ancora non controllati, di conseguenza aggiungeremo questo aspetto.

CurrentPlayer Il nostro gruppo non aveva pensato nemmeno di mostrare il giocatore che sta giocando il proprio turno, di conseguenza aggiungeremo questo aspetto.

Metodo update Nell'ultima sessione di laboratorio avevamo già deciso di implementare anche noi un metodo simile al metodo update utilizzato da questo gruppo, per cercare di evitare alcuni istanceOf che avevamo inserito nel codice.