Documentazione del client

Marco Poletti

2 luglio 2011

Introduzione

Questo documento descrive le varie iterazioni che hanno portato all'implementazione finale della parte client del progetto Cupido.

1 Implementazione del cambio di schermata

La prima iterazione si è posta l'obbiettivo di creare la struttura di base per il cambio di schermata.

Il client in ogni momento mostra un'unica schermata. La classe che gestisce la schermata ha la possibilità di gestire gli eventi, sia quelli generati dall'interazione dell'utente utilizzando i vari controlli grafici visualizzati, sia quelli ricevuti dalla servlet attraverso Comet (ma questi ultimi verranno implementati solo in una iterazione successiva).

In questa iterazione sono stati aggiunti degli stub per le schermate non ancora implementate.

2 Implementazione della grafica di gioco

In questa iterazione è stata implementata la grafica di gioco, comprese le animazioni per il movimento delle carte sul tavolo, ma senza la logica necessaria a decidere quando eseguire le varie animazioni.

La classe più significativa, in relazione a questa iterazione, è `CardsGameWidget`. Questa classe potrebbe essere riutilizzata anche per altri giochi di carte a due o a quattro giocatori, non dipendendo dalle regole di Hearts.

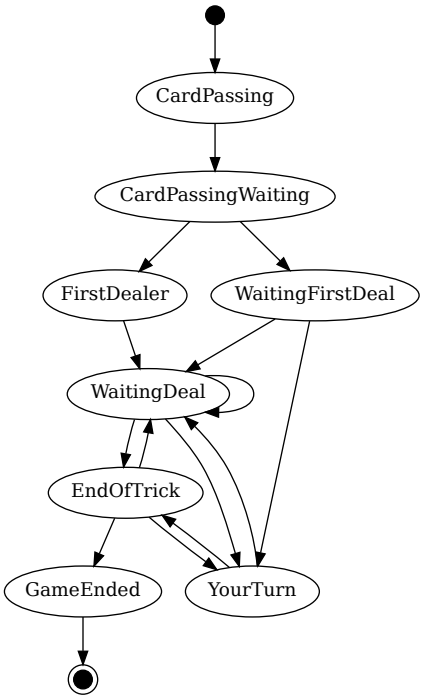

Figura 1: Il diagramma degli stati di gioco

3 Preparazione alla logica di gioco

In questa iterazione è stata creata la struttura necessaria per gestire i vari stati del gioco. La servlet non era ancora pronta per fornire i servizi richiesti, quindi in questa iterazione non è stata implementata la logica di gioco, ma si è preparato il codice in modo da poterla aggiungere facilmente in un secondo momento.

Per la logica di gioco, è stato usato il pattern *State*; per ogni stato del gioco è stata implementata una classe diversa, per poter gestire gli eventi in modi differenti, in base allo stato attuale.

Il diagramma degli stati di gioco è in figura 1.

In particolare, alla ricezione di una notifica Comet essa viene inoltrata alla classe che gestisce la schermata attuale. Molte schermate sono in grado di gestire direttamente questi eventi.

Le schermate di gioco, data la loro complessità, inoltrano tali notifiche allo stato corrente, in modo da ripartire al meglio il codice fra i vari stati, per migliorare la manutenibilità del codice.

Non tutti gli stati sono in grado di gestire tutte le notifiche; ad esempio lo

stato `EndOfTrickState`, che visualizza l'animazione con cui le carte vengono traslate verso il giocatore che ha giocato la carta più alta, non ha modo di gestire la notifica `CardPlayed`, con cui la servlet informa il client che una certa carta è stata giocata.

Si è quindi resa necessaria la possibilità di rimandare la gestione delle notifiche ad uno stato successivo. In questo modo, all'arrivo di una notifica, si prova a farla gestire allo stato corrente; se ciò non è possibile, tale notifica viene mantenuta dal gestore degli stati, in una coda, e verrà notificata nuovamente alla prossima transizione di stato, finché non sarà gestita.

4 Implementazione della logica di gioco e delle schermate rimanenti

Nel design di massima si è cercato di ridurre il carico computazionale del client, per migliorare le prestazioni, delegando il più possibile le computazioni nel backend; per questo motivo, non è stato necessario implementare completamente la logica di gioco, ma solo alcune parti.

A questo punto, la servlet era abbastanza sviluppata da poter comunicare con il client, e quindi è stato possibile implementare la logica di gioco, compresa appunto l'interazione con la servlet.

Si è reso possibile anche lo sviluppo delle rimanenti schermate, caratterizzate da frequenti comunicazioni con la servlet.