Ingegneria del Software 1: Progetto

Due on Martedi, Aprile 15, 2014

Contents

1 Introduzione

Il progetto consiste nello sviluppo di una versione software del gioco da tavolo sheepland.
Il progetto finale dovrà includere:

- diagramma UML iniziale dell'applicazione
- diagrammi UML che mostrino come è stato progettato il software;
- implementazione funzionante del gioco conforme alle regole del gioco e alle specifiche presenti in questo documento
- codice sorgente dell'implementazione
- codice sorgente dei test di unità

La data di consegna e valutazione 17 Giugno 2014 (durante il laboratorio).

2 Regole del gioco

Le regole del gioco sono descritte nel file Progetto di Laboratorio, caricato su BeeP. In aggiunta vengono aggiunte le seguenti **regole addizionali**.

2.1 Regole Addizionali

- **Lupo**: UN lupo si muove in modo casuale (tirando un dado) da una regione ad una limitrofa. Il lupo si muove dopo che tutti i giocatori hanno effettuato il loro turno. Se il lupo si sposta in una regione con delle pecore ne mangia (casualmente) una. Il lupo si può muovere da una regione all'altra solo se non esiste un recinto sulla strada che separa due regioni. Se il lupo si trova in una regione con tutti i cancelli chiusi allora salterà sopra uno dei cancelli (si muoverà nella direzione indicata dal lancio di dadi nonostante il cancello sia chiuso). Inizialmente il lupo si trova a Sheepsburg.

Le seguenti **azioni addizionali** possono essere eseguite in alternativa a muovere il pastore, la pecora o comprare il terreno, seguendo le regole tradizionali del gioco.

- **Accoppiamento 1**: se il pastore è in una strada vicino a una regione con almeno 2 pecore, può eseguire una azione di accoppiamento. Il pastore tira il dado, se il numero equivale a quello della cella sopra cui è posto una nuova pecora viene generata e aggiunta nella regione. Nota che se il pastore è vicino a due regioni entrambe con due o più pecore deve decidere in quale delle due regioni eseguire l'azione di accoppiamento.
- **Accoppiamento 2**: oltre alle pecore sono presenti montoni e agnelli. Il pastore può eseguire l'azione di accoppiamento solo se nella stessa regione ci sono almeno un montone e una pecora. Se l'accoppiamento ha successo un nuovo agnello viene generato. L'agnello diventa in maniera randomica montone o pecora dopo due cambi giocatore di gioco (Nota che quando viene inizializzato il gioco ogni pecora posizionata su un terreno puó essere randomicamente agnello, montone o pecora).
- **Abbattimento 1**: se un pastore è su una strada vicino un terreno con almeno una pecora, un montone o un agnello può decidere di abbatterlo. Tira il dado, se il dado corrisponde al numero della strada puó decidere di abbattere *a suo piacimento* una pecora, un montone o un agnello.
- **Abbattimento 2**: quando un pastore esegue una azione di abbattimento tutti gli altri pastori in una strada limitrofa (direttamente connessa) a quella del pastore tireranno il dado. Il pastore che ha abbattuto il capo darà due denari a tutti quei pastori che hanno ottenuto un punteggio maggiore o

uguale di 5 nel lancio del dado in cambio del loro silenzio. L'azione di abbattimento puó essere eseguita solamente nel caso in cui il numero di denari del pastore sia sufficiente a comprare il silenzio dei pastori nelle strade limitrofe. Questo controllo viene eseguito prima che l'azione di abbattimento abbia inizio.

- **Market**: dopo che tutti i giocatori hanno giocato il loro turno di ogni turno c'è la fase del mercato. Partendo dal primo giocatore fino all'ultimo ogni giocatore puó scegliere i terreni che intende vendere e il prezzo richiesto per ognuno di essi (da 1 a 4 denari). Quando TUTTI i giocatori hanno scelto quali terreni vendere, partendo da un giocatore a caso e seguendo l'ordine di gioco ogni giocatore puó comprare i terreni messi in vendita dagli altri pastori .

3 Funzionalita Avanzate

Di seguito sono proposte alcune funzionalità avanzate che concorrono alla valutazione. Attenzione: il loro contributo non è necessariamente additivo. Design e codice verranno comunque valutati in quanto tali e contribuiranno al giudizio globale.

- **Gestione degli utenti**. Realizzare un sistema di gestione degli utenti che supporti il login dei giocatori, conservi per ciascuno le statistiche di gioco (numero di vittorie, tempo di gioco, numero di sconfitte numero medio di denari ottenuti in ogni partita) e produca una classifica ordinata prima per numero di vittorie e quindi per numero di denari ottenuti in carriera. Inoltre per ogni parita si desidera memorizzare in un frame laterale la sequenza di mosse effettuate dai vari giocatori in ordine di occorrenza.
- **Stateful server**. implementare la possibilità di salvare lo stato del server su disco e di ricaricarlo all'avvio successivo. In particolare il server potrà inviare un comando pause ai client per indicare il su spegnimento e da li in vanati non accetterà ulteriori comandi, salverà lo stato su disco e terminerà la sua esecuzione. Al riavvio dovrà ricaricare la partita dal punto in cui si è interrotta.
- **Generazione automatica delle configurazioni di gioco**. il server deve essere in grado di generare automaticamente una configurazione del gioco partendo da alcuni parametri dati. Per esempio dato il numero dei terreni e numero di regioni per ogni tipo di terreno il server deve generare una mappa random, consigliare un numero di giocatori adeguato. Il server deve anche consentire di progettare mediante un editor opportuno delle mappe da memorizzare sul server.

4 Valutazione

Il progetto dovrà essere svolto in gruppi di due studenti e terminato entro il laboratorio del 17-06-2014. La tabella 1 presenta le funzionalità da implementare nel caso di gruppi da 1,2 e 3 studenti. I gruppi da 1 e 3 studenti sono **sconsigliati**.
Saranno oggetto di valutazione

- la qualità della progettazione con particolare riferimento a un uso appropriato di interfacce, ereditarietà, composizione tra classi, uso dei design pattern;
- il livello di autonomia e impegno nello svolgimento del progetto;
- la stabilità dell'implementazione e la sua conformità alle specifiche date;
- la qualità e la leggibilità del codice scritto, con particolare riferimento a nomi di variabili/metodi/-classi/package, all'inserimento di commenti e documentazione JavaDoc (preferibilmente in inglese), la mancanza di codice ripetuto e metodi di eccessiva lunghezza;

Table 1: Tabella Di Valutazione

		Numero studenti		
		1	2	3
Punteggio Max	0-22	RMI+ Command Line Client	RMI+ **Socket** + Command Line Client	RMI+ Socket + **Static GUI**
	0-24	Static GUI	Static GUI	Dynamic GUI
	0-26	Dynamic GUI	Dynamic GUI	Regole addizionali
	0-30L	Regole addizionali	Regole addizionali	1 Funzionalità Avanzata

- la qualità e la copertura dei casi di test: il nome e i commenti ogni test dovranno chiaramente specificare le funzionalità testate e i componenti coinvolti.
- il rispetto delle metriche sonar
- il corretto utilizzo di SVN

I gruppi da 1,2,3 studenti devono implementare le specifiche descritte in tabella 1 in relazione al punteggio desiderato. Nota: in tabella sono rappresentati i **massimi** punteggi ottenibili in relazione alle features implementate. Per esempio, per un gruppo di 2 studenti e un punteggio desiderato di al massimo 24 punti è necessario implementare RMI, Socket, Command Line Client e static GUI.

- **Command Line Client**: il client e' implementato come un interfaccia testuale e i vari giocatori che si alternano nei turni utilizzeranno la tastiera. Per gli studenti che intendono proseguire nell'implementazione con l'interfaccia grafica é necessario utilizzare in maniera ragionata il paradigma Model-View-Controller (MVC) e gli altri pattern in modo da sostituire in maniera semplice la parte della logica di gioco dalla parte di interazione con i giocatori.
- **Static GUI**: consiste in un interfaccia grafica swing minimale, semplice e senza particolari "effetti grafici"
- **Dynamic GUI**: consiste in un interfaccia grafica più complessa che consente di eseguire azioni come per esempio zoom, rotazione della mappa etc.
- **Regole Addizionali**: sono le regole presentate in sezione 2.1
- **Funzionalità avanzate**: sono presentate in sezione 3
- **Socket**: la comunicazione avviene mediante scambiati attraverso socket. Lo studente deve autonomamente definire e implementare un protocollo di comunicazione tra il client e il server.

5 Specifiche implementative

In questa sezione vengono presentati i requisiti tecnici dell'applicazione che deve funzionare in rete secondo il paradigma client-server. Si raccomanda l'utilizzo del pattern **MVC** (Model-View-Controller) per progettare l'intero sistema.

5.1 Client

- I client devono essere sviluppati utilizzando JavaSE.
- L'interfaccia grafica deve essere realizzata obbligatoriamente in Swing,

- il client deve supportare RMI e Socket in relazione al numero di studenti del gruppo come specificato in tabella 1.
- Nel caso in cui sia richiesta sia l'implementazione RMI che per mezzo di socket, l'applicazione all'avvio deve permettere all'utente di selezionare il metodo utilizzato per la comunicazione.

5.2 Server

Il server deve gestire le partite. Deve permettere di

- creare una nuova partita, inizializzarla, giocarla e concluderla secondo le regole del gioco.
- deve essere in grado di gestire più partite contemporaneamente
- deve essere implementato secondo la logica JavaSE
- nel caso in cui sia l'implementazione via soket che RMI sia richiesta all'avvio deve essere possibile scegliere il metodo utilizzato per la comunicazione

5.3 Avvio della partita

L'assunzione base è che ogni client che voglia partecipare a una partita conosca l'indirizzo (numerico o simbolico) del server. Quando un giocatore si connette al server,

- se cè una partita in fase di avvio di un giocatore viene automaticamente aggiunto alla partita
- quando la partita raggiunge 6 giocatori oppure quando viene raggiunto un timeout e ci sono almeno 2 giocatori la partita inizia
- se non ci sono partite in fase di avvio una nuova partita viene creata.

Si precisa che una nuova partita viene creata solamente quando un utente si connette e non ci sono partite in attesa altrimenti l'utente entra automaticamente a far parte della partita in fase di avvio.

5.4 Gioco

Il server consente ai vari giocatori di svolgere i propri turni secondo le regole di sheepland e le regole addizionali presentate in questo documento. E' necessario gestire il caso in cui i client si disconnettano. Ai giocatori è permesso abbandonare temporaneamente la partita per via della perdita di connettività.

- se un giocatore va offline il server sospende la partita per un periodo di tempo fissato a priori
- se il giocatore si riconnette prima del tempo prestabilito la partita riprende da dove è stata interrotta
- trascorso il periodo di tempo il server sospende il giocatore (nota il giocatore non esegue mosse ma viene comunque considerato nel conteggio dei punti etc.)
- dopo la sospensione del giocatore il gioco ricomincia senza il giocatore sospeso
- in caso di sospensione tutti i giocatori vengono notificati e non possono interagire con il gioco
- se il giocatore si riconnette prima della fine della partita può ricominciare a giocare, come se nulla fosse successo (ovviamente perde i turni durante i quali non ha giocato