Imperiali Emilio Lamonaca Edoardo Lorenzo Francesco Vito

PROGETTO DI INGEGNERIA DEL SOFTWARE

NOTE

I tre jar del progetto sono localizzati in /deliveries/jar.

Per mantenere il diagramma UML il più leggibile possibile sono state adottate le seguenti convenzioni:

- I getter ed i setter banali non sono stati inseriti nelle classi per evitare di renderle ulteriormente grandi.
- I nomi delle relazioni e le relative cardinalità non sono state inserite in quanto poco significative e, vista la lunghezza delle frecce, sarebbero risultate poco visibili.
- Le classi Observer e Observable, con i relativi riferimenti padre/figlio non sono riportate in quanto pesanti a livello di notazione grafica.
- Nel caso in cui la maggior parte delle classi appartenenti ad un package ereditino da una stessa interfaccia, si è preferito usare una singola relazione di ereditarietà relativa all'intero package.
- Nel caso di relazioni di ereditarietà, i metodi con signature Override non sono stati riportati nelle classi figlio. Eccezion fatta per le classi che implementano Runnable, dove si è preferito inserire il metodo run() al posto della relazione di ereditarietà.
- La relazione di ereditarietà con la classe Stopper (implementata da Controller, GameManager, Server, RMIServerConnection e RMIClientConnection) è stata tralasciata perché poco significativa.

FUNZIONALITÀ

Le principali funzionalità che sono state implementate sono:

- Per la parte di rete, connettività sia tramite Socket che RMI, con la possibilità di alternarle in caso di riconnessione al Server. Inizialmente il Client può connettersi con configurazione di rete di default lette da file, oppure specificare indirizzo IP e porta del Server.
- Per l'interfaccia utente, l'utilizzo della CLI oppure della GUI, accessibili da due jar diversi (la scelta tramite linea di comando sarebbe stata poco user-friendly). In caso di riconnessione il Client può cambiare interfaccia grafica senza alterare lo stato della partita.
- Per la disconnessione, i Client in partita vengono notificati immediatamente della mancanza di un giocatore (sia in caso di caduta della rete, che in caso di chiusura dell'applicativo), che può riconnettersi e proseguire la partita anche dal turno stesso. Nel caso in cui un player rimanga da solo, questo viene dichiarato automaticamente vincitore.
- Per la gestione del tempo di gioco, sono presenti dei timer in fase di creazione della lobby, in fase di selezione delle finestre iniziali e durante il turno di un giocatore. I valori dei timer sono caricati da un file di configurazione all'avvio del Server.
  In caso di timeout il server notifica i Client che, gestendo il messaggio in maniera asincrona, avvertono immediatamente i giocatori e fermano qualunque operazione in corso.
- Per la gestione della validità delle mosse, sia con la CLI che con la GUI, sono stati implementati controlli atti ad impedire mosse illecite senza sovraccaricare il Server di richieste inutili.
- Per quanto riguarda infine le funzionalità avanzate, sono state scelte la gestione di più partite in contemporanea da parte del Server (Multiplayer) e la creazione di carte schema dinamiche, accessibili tramite un file JSON e renderizzate dinamicamente dalla GUI.

DESIGN PATTERN

I principali Design Pattern utilizzati nel progetto sono i seguenti:

- Visitor: Gestione dei messaggi inviati fra client e server, delle Tool Card e dei bottoni nella GUI.
- State: Gestione dello stato in cui si trova in un determinato momento la GUI.
- Strategy: Gestione dei comportamenti come gli effetti dei bottoni.
- Observer: Relazioni fra Controller e View, fra View e Model e fra network e View.
- Singleton: Le Tool Card, gli obiettivi privati e pubblici sono Singleton in quanto condivisi fra varie partite senza ristanziare gli oggetti.
- MVC