DESCRIZIONE UML PER PEER-REVIEW 2

Gruppo revisore: CG31
Gruppo revisionato: GC21, Toninelli, Priuli, Sabino, Zuccoli

In seguito alla prima peer-review abbiamo apportato i miglioramenti suggeriti all' UML, in particolare:

- sono stati aggiunti package per migliorare la suddivisione di alcune parti del progetto;
- è stata esplicitata la classe che si occupa della gestione delle carte in formato Json nel package utils del progetto.

Per quanto riguarda l'implementazione del protocollo di rete abbiamo deciso di permettere ai giocatori di scegliere se utilizzare Socket oppure RMI.
In entrambi i casi abbiamo deciso di avvalerci del passaggio di oggetti di tipo Bean. Queste sono delle classi immutabili che rappresentano lo stato del gioco e sono trasmesse ai singoli Client affinché possano accedere allo stato della partita senza poterla influenzare direttamente. In particolare abbiamo implementato nel package bean all'interno del package model le seguenti classi: GameBean, PlayerBean, ObjectiveCardBean, FieldBean, CardBean e sottotipi per le varie carte. Le classi appena citate presentano solo metodi getter, così da evitare possibili azioni illecite da parte del Client. Questa soluzione permette ai vari giocatori connessi di poter interagire solamente con la view (che stiamo implementando sia in forma testuale, sia in forma grafica).

Per quanto riguarda l'interazione generica: tutto parte dalla classe CodexNaturalis che permette di avviare un Server o un Client. Il primo realizzerà un oggetto di tipo MasterController e lo renderà accessibile sulla rete mediante Socket ed RMI. Quest'ultimo si occuperà della fase iniziale del login da parte dei giocatori e renderà possibile la creazione di una nuova partita, l'accesso a una partita creata da un altro giocatore o permetterà la ripresa di una partita precedentemente avviata e che era stata interrotta (FA: Persistenza). Avviando un Client, invece, è possibile scegliere quale dei due protocolli di rete utilizzare.

SOCKET:
Lato server la prima connessione per la creazione del socket viene gestita dalla classe MasterServerSocket, dove una volta costruito il socket verrà creato e lanciato un thread che gestirà le future comunicazioni con il client attraverso MasterServerHandler. Quest'ultima si occupa dell' eventuale creazione del gameController qualora la partita fosse nuova, e della creazione del gameControllerHandler il quale gestisce la comunicazione col Client per il proseguimento della partita.
La comunicazione tra Client e GameControllerHandler/MasterServerHandler è gestita tramite messaggi testuali, utilizzando lato client la classe MasterServerProxy

per la comunicazione sincrona che effettuerà il login e successivamete la classe GameControllerProxy (asincrona) per la comunicazione dei messaggi di gioco. GameControllerHandler presenta un metodo run, eseguito su un thread separato, che in maniera ciclica osserva le notifiche da parte del Client (che sta interagendo con la view) e successivamente tramite listener modifica il model.

RMI:
Il Client si connette prendendo sul registry lo stub del server. Quest'ultimo è la classe RMIMasterController. Connettendosi a questa classe il Client ottiene lo stub RMIGameController che si occuperà dello svolgimento della singola partita