Descrizione implementazione di rete

Mattia Pergola, Vittorio Paolo Pisacane, Roland Riva, Lorenzo Tavani
Gruppo 54

Maggio 2024

Descrizione della parte di progettazione di rete.

1 Introduzione

Abbiamo deciso di implementare sia comunicazione mediante protocollo **RMI** che **Socket-TCP**, purtroppo al momento possiamo fornire una descrizione dettagliata del funzionamento solo della connessione con RMI, in quanto al momento è l'unica che abbiamo implementato, in ogni caso sarà spiegata in maniera più generale anche l'idea per implemetare la parte di socket.
I messaggi che vengono inviati possono riguardare il Controller o il GameController:

- **Controller:** Gestisce tutte le partite presenti sul server, implementato per la FA: "Partite Multiple"
- **GameController:** gestisce le singole partite

I messaggi che possono essere inviati sono i seguenti:

- Per il Controller:
  - createGame
  - joinFirstGameAvailable
  - joinGame
  - reconnect
  - leaveGame

- Per il GameController:
  - playerIsReadyToStart
  - isMyTurn
  - disconnectPlayer
  - sentMessage
  - getGameID
  - putCardOnPlayerBoard
  - leave
  - getNumberOfPlayersOnline
  - extractGoldCard
  - extractResourceCard

Questi sono tutti i messaggi che i client possono imviare al server con entrambi i tipi di connessione.

2 Protocollo RMI

La classe ClientRMI fa il lookup sul registry e ottiene un Controller che gli permetterà di andare a creare una partita oppure di unirsi o riconnettersi ad una già esistente.
Tutte le richieste hanno sempre come parametro un GameListener il quale è un oggetto remoto che il server sfrutterà per notificare al client tutte le modifiche che vengono apportate al model.
Sfruttiamo il fatto che ad ogni richiesta verso il controller il server restituisca un oggetto di tipo GameControllerInterface il quale ci permetterà di capire se tutte le richieste fatte dopo essersi collegato ad una partita(ad esempio posizionare una carta) vanno a buon fine.

3 Protocollo Socket-TCP

Per la parte di Socket si è pensato che la classe ClientSocket debba funzionare uguale a quella corrispetiva in RMI, e per ovviare alla differenza che non può invocare metodi su oggetti remoti come RMI, pensavamo di creare dei messaggi specifici per ogni tipo di azione che il client vuole eseguire.

Poi per distinguere se il messaggio è per il Controller o per il GameController, tutti questi tipi di messaggi erediteranno da una classe astratta, che rappresenterà un messaggio generale, la quale avrà due metodi che avranno lo stesso nome ma che prenderanno parametri diversi (se si tratta di un messaggio per il controller prenderà il GameListener e il Controller al quale è riferito che per la nostra implementazione sarà unico, mentre se è per il GameController prenderà in input solo il GameController al quale è riferito).
Questa strategia si rivela comoda perchè ci dovrebbe permettere di non fare gli istanceof per ogni messaggio per capire a chi è riferito.