Peer Review 2

Pucci, Santarossa, Ruggieri, Sciarretta

May 2024

1 Preliminari

Per quanto riguarda la parte di networking, è stato scelto di implementare un protocollo di comunicazione di tipo *RMI*. Illustriamo anzitutto a chi si riferiscano i messaggi provenienti lato client. Disponiamo di due classi principali:

- *MainController*: Si tratta di un controller di alto livello, principlamente utilizzato per creare una nuova partita o per aggiungere un giocatore ad una partita pre-esistente. Esso gestisce tutte le partite in corso.
- *GameController*: Implementa tutte le funzionalità base di gioco, come il piazzamento di una carta o il pescaggio da mazzo. In sintesi, esso gestisce una e una sola partita.

Come classi intermedie per l'utilizzo delle due precedentemente menzionate, abbiamo scelto di utilizzare le seguenti:

- *ServerImpl*: Un'implementazione di tutte le funzionalità reputate opportune lato server.
- *ClientImpl*: Lo stesso, ma lato client.

Il dialogo tra tali classi è messo in atto tramite interfacce (implementate dalle stesse classi). Ad esempio, qualora un client decidesse di creare una partita, chiamerà il metodo *createGame* in *ClientImpl*; la chiamata sarà dunque delegata alla *ServerImpl*, che in modo coerente agirà sul *MainController*, dove chiaramente è presente il corrispettivo metodo *createGame*. Segue una breve lista dei messaggi che possono giungere alle classi *MainController* e *GameController*: per la prima, si hanno

- *createGame*
- *joinGame*
- *reconnect*
- *leaveGame*
- *getGamesDetails*

Per la seconda, si hanno invece i più basilari metodi:

- *placeCard*
- *drawCard*
- *myTurnIsFinished*
- *isMyTurn*
- *sendMessage*

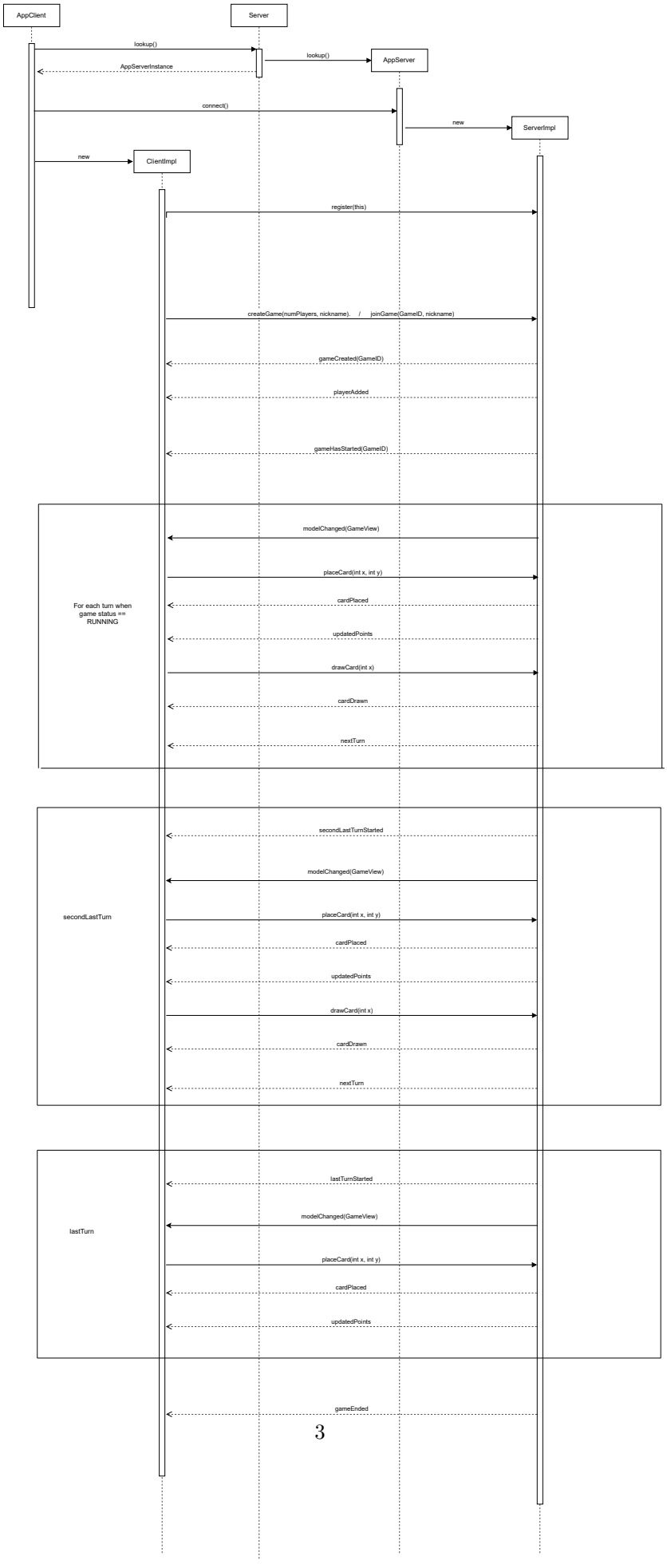
Un semplice attributo con cui identifichiamo un client all'interno della dimensione del networking è il suo nickname (chiave primaria).

2 Protocollo RMI

Descriviamo ora sinteticamente la sequenza dei messaggi tra client e server nel caso di comunicazione *RMI*. In modo standard, il client riceve un'istanza del server tramite *lookup* del registry. Successivamente, viene richiesto lato client di stabilire una connessione col server, che a meno di *RemoteException* crea un'istanza di *ServerImpl*. Contestualmente a ciò, viene creata un'istanza collegata di *clientImpl*, la quale implementa l'interfaccia *Runnable*. Essa vivrà dunque in un thread separato. All'interno di quest'ultima classe, come accennato in precedenza, sarà possibile mandare al server i vari messaggi necessari allo svolgimento di una partita (lato client). Segue per completezza il sequence diagram del protocollo *RMI* (Pagina 3).

3 Protocollo Socket

Di seguito viene descritta brevemente la struttura della comunicazione tramite *Socket*. Seguendo lo standard di programmazione dei sistemi distribuiti, sono stati implementati due moduli intermedi tra client e server: uno Stub (*ServerProxy*) e uno Skeleton (*ClientProxy*). La procedura in sintesi è la seguente: il client istanzia *ServerProxy*, che rappresenta localmente (lato client) il server. Questa classe si occupa quindi di serializzare i messaggi del client, chiamare in remoto il server e inviare il messaggio tramite connessione socket. Tuttavia, *ServerProxy* non dialoga direttamente con l'implementazione del server, ma con l'interfaccia intermedia *ClientProxy*, istanziata dal server e che rappresenta localmente il client (lato server). *ClientProxy* deserializza i messaggi che riceve dal client, effettua la chiamata alla funzione richiesta dal client nell'implementazione del server, riceve i risultati da quest'ultima e manda un messaggio di risposta al client.


AppClient
Server
AppServer
ServerImpl
ClientImpl
lookup()
lookup()
AppServerInstance
connect()
new
new
register(this)
createGame(numPlayers, nickname). / joinGame(GameID, nickname)
gameCreated(GameID)
playerAdded
gameHasStarted(GameID)
For each turn when game status == RUNNING
modelChanged(GameView)
placeCard(int x, int y)
cardPlaced
updatedPoints
drawCard(int x)
cardDrawn
nextTurn
secondLastTurn
secondLastTurnStarted
modelChanged(GameView)
placeCard(int x, int y)
cardPlaced
updatedPoints
drawCard(int x)
cardDrawn
nextTurn
lastTurn
lastTurnStarted
modelChanged(GameView)
placeCard(int x, int y)
cardPlaced
updatedPoints
gameEnded