Hard Deadline 14/05

Communication

Gruppo AM03: Bernasconi, Calici, Comelli

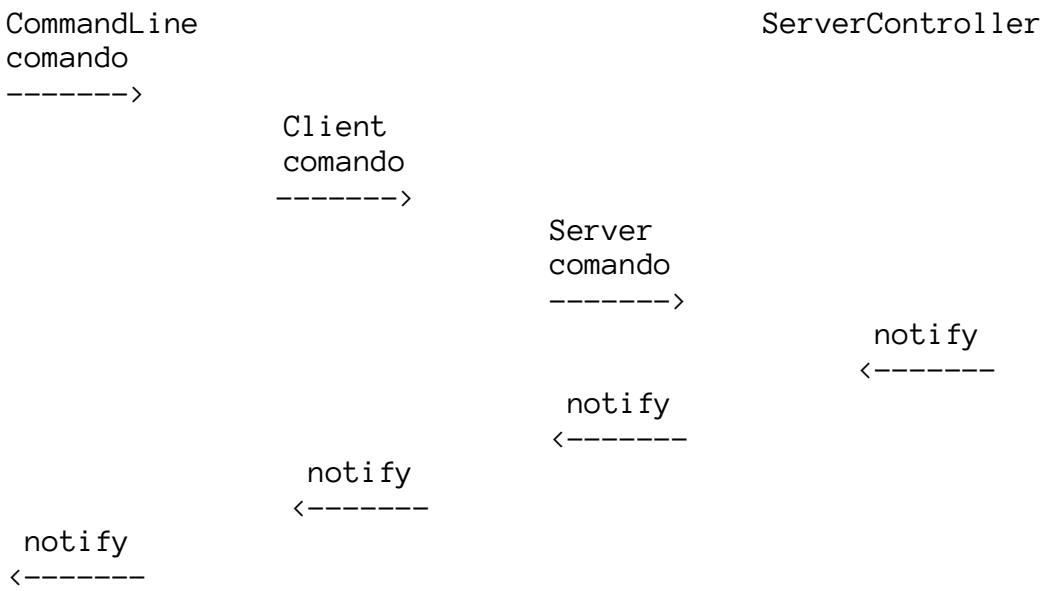
INTRODUZIONE

La parte di network del progetto include sia RMI che Socket. Un'unica classe Server viene lanciata e stampa il proprio indirizzo IP. Ogni Client che desidera partecipare al gioco dovrà scegliere una tra le due tipologie di connessione e la classe Client creerà un oggetto di tipo **ClientInterface** implementato dalla tipologia richiesta (**RMIClient** o **SocketClient**).
La gestione delle connessioni è gestita dal **ConnectionManager** che implementa la **ConnectionInterface** e si occupa sia di creare l'RMI Registry e fornire un oggetto di tipo **RMIServer** tramite metodo remoto, sia di accettare le connessioni Socket.

COMUNICAZIONE

Fondamentale per la comunicazione è la classe **Message**. Nel caso di connessione Socket, infatti sarà necessaria per il colloquio tra Client e Server. Infatti, prendendo ad esempio l'azione di login, una volta scelti i parametri (username, colore), la classe **CommandLine** chiamerà sul Client il metodo login(username, color).
Il **SocketClient** si occuperà quindi di inviare come prima cosa un oggetto Message.LOGIN al **SocketServer** e questo, in perenne attesa di oggetti di tipo **Message**, si preparerà a ricevere sugli stream una stringa che rappresenterà lo username, e un oggetto di tipo **TokenColor** rappresentante il colore. Allo stesso tempo, una volta che il Server terminerà di processare l'azione, risponderà al Client con il metodo notify(). Tramite overload, riceverà come parametri un **Message** rappresentante l'azione svolta, oppure un **Message** e un **Outcome**, oppure un **Message**, un **Outcome** e un oggetto generico.
La classe **Outcome** si presenta come un'enumerazione contenente i possibili esiti dell'azione, ovvero RIGHT, WRONG, ALL, in base ai quali il Client agirà di conseguenza.

ESEMPIO



COMANDI

Allo stato attuale del progetto, l'acquisizione dei comandi è affidata alla classe **CommandLine** che si occupa anche, come dice il nome, dell'interfaccia utente testuale. La filosofia generale implica che l'utente durante la partita possa digitare i comandi sia nel suo turno, sia in quello degli avversari.
Nel caso in cui l'utente desideri vedere i propri dati di gioco, lo potrà fare nel corso di tutta la partita, mentre nel caso in cui volesse effettuare azioni, dovrà attendere il suo turno e gli verrà dato un messaggio di errore se così non fosse.
Per ogni azione viene presa una singola stringa che viene poi convertita per interpellare i corrispondenti metodi del Server. Il Server quindi chiamerà il relativo metodo del **ServerController** che andrà a modificare il model.
Al termine di ogni azione il Server invierà al Client l'esito della stessa tramite il metodo notify().

***LOGIN:** *login <username> <color>*
Il primo comando che viene richiesto all'utente di digitare è quello relativo al login:
-username: nome scelto dal giocatore;
-color: colore scelto dal giocatore fra i 5 disponibili.
Nel caso in cui lo username fosse già stato scelto oppure il colore fosse inesistente o già scelto da qualcuno, il Server notificherà l'esito scorretto. Viceversa creerà un nuovo giocatore e lo aggiungerà tra i partecipanti.

***DISCONNECT:** *disconnect*
Permette all'utente di disconnettersi. Verrà quindi aggiunto dal Server in una lista di utenti disconnessi in modo da permettere l'eventuale riconnessione successiva, senza la perdita dei dati.

***MOVE:** *move <first_direction, ..., last_direction>*
Permette al giocatore di inserire e quindi muoversi fino a 3 posizioni. Ogni direzione (Up, Down, Left, Right) verrà convertita in un oggetto Direction.

***GRAB:** *grab <direction> <0, 1, 2, 3>*
Svolge l'azione di raccoglimento. Può essere scelta anche una direzione in cui muoversi.
-0: per raccogliere una carta munizioni;
-1,2,3: per raccogliere, rispettivamente, la prima, la seconda o la terza arma.

***SHOOT:** *shoot <weapon_name> <powerup_name>*
Svolge l'azione dello sparare. In base all'arma scelta, verranno chiesti all'utente altri parametri per definire meglio l'azione. Nel caso di Lock Rifle, ad esempio, all'utente verranno chieste una o due vittime, per poi chiamare l'azione shoot() con il nome dell'arma e gli avversari scelti.
Il metodo shoot(), tramite overload, riceverà ogni volta un diverso numero di parametri in base all'arma scelta. Si può utilizzare un powerup per utilizzarlo al posto delle munizioni.

***POWERUP:** *powerup <powerup_name>*
Permette di utilizzare un powerup tra i due utilizzabili in qualsiasi momento (Newton, Teleporter).

***END:** *end*
Termina il turno dell'utente per passare al successivo.

CLASSI DI RIFERIMENTO

MESSAGE
Su questa classe si basa l'intera messaggistica tra Client e Server per quanto riguarda la connessione Socket. È un'enumerazione contenente tutte le tipologie di messaggi che si scambiano le due classi:

- LOGIN: utilizzato per gestire la fase di login
- DISCONNECT: utilizzato per richiedere la disconnessione
- MOVE: utilizzato per richiedere la mossa di movimento
- GRAB: utilizzato per richiedere la mossa di raccoglimento
- SHOOT: utilizzato per richiedere la mossa di sparo
- END_TURN: utilizzato per terminare il turno
- NEW_TURN: utilizzato per settare il nuovo turno del giocatore
- GAME: utilizzato nel caso di messaggi inerenti il game in corso
- NOTIFY: utilizzato per notificare l'esito di qualsiasi tipo di azione

OUTCOME
L'esito del metodo notify(), che andrà a mettere al corrente il Client della riuscita dell'azione, è affidato ad un oggetto della classe **Outcome**, in base al quale il Client agirà di conseguenza:

- RIGHT: l'esito dell'azione richiesta è positivo
- WRONG: l'esito dell'azione richiesta è negativo
- ALL: l'esito dell'azione interessa tutti i partecipanti

SERVER
Suddiviso in **RMIServer** e **SocketServer** si occupa di ricevere i messaggi del Client, processarli e lanciare il relativo metodo di **ServerController** che andrà a modificare lo stato della partita, per poi notificare l'esito dell'azione richiesta.

CONNECTION MANAGER
Gestisce le connessioni tra Client e Server:

- RMI: crea il registro alla porta designata ed esporta sé stesso come oggetto remoto. Il metodo remoto enrol(), permette all'**RMIClient** di ottenere il suo **RMIServer**
- SOCKET: crea un pool di Executors, ognuno dei quali accetta una connessione Socket

CLIENT
Suddiviso in **RMIClient** ed **RMIServer** si occupa di inviare al Server i messaggi alle azioni di gioco richieste dall'utente.

CATENA DI AZIONI

CommandLine ---> Client ---> Server ---> ServerController ---> GameController ---> Game

CONCLUSIONE

La filosofia generale della rete è dunque quella di gestire nello stesso modo i due tipi di connessione (RMI e Socket) e permettere all'utente di scegliere quale utilizzare. Di seguito verranno brevemente approfondite altre caratteristiche generali, utili alla piena comprensione del funzionamento.

TIMER

Due timer principali saranno utilizzati per gestire la partita:

-INIZIO PARTITA: il primo timer scatterà quando 3 giocatori si connetteranno per giocare. Infatti le partite si svolgono con un numero che va dai 3 ai 5 giocatori. Di conseguenza appena verrà raggiunto il numero minimo di partecipanti comincerà il conto alla rovescia, al cui termine partirà automaticamente una partita. Chiaramente, se primo dello scadere del tempo si connetteranno 5 giocatori, il timer si interromperà e la partita avrà inizio;

-TURNO: durante ogni turno, il giocatore corrente avrà un tempo preimpostato per svolgere tutte le sue mosse. Ciò è necessario per evitare che la partita si interrompa nel caso in cui un giocatore smettesse di fare mosse o si disconnettesse.

DISCONNESSIONE

Sia per quanto riguarda Socket che per RMI, è svolto un controllo costante sull'effettiva connessione dei giocatori alla partita. Infatti nel caso in cui un giocatore perdesse la connessione o si disconnettesse per sua volontà, sarà compito del Server gestire la situazione e notificare agli altri giocatori quanto accaduto.

-RMI: appena effettuata la connessione viene attivato un timer, tramite un thread. Esso viene messo in pausa per una quantità di secondi decisa in fase di configurazione e successivamente si sveglia e tenta di utilizzare un metodo remoto del client che ha la funzione di ping. Nel caso in cui riscontrasse un'eccezione, sarebbe causata dalla disconnessione del Client.

-SOCKET: la gestione della connessione tramite Socket è gestita attraverso le eccezioni. Infatti nel caso in cui avvenisse IOException, il Server notificherebbe l'avvenimento chiudendo le socket e gli stream di comunicazione.

La disconnessione viene notificata dal Server a tutti i Client rimasti in partita.

RICONNESSIONE

A seguito di una disconnessione, è possibile per un utente riconnettersi senza perdere alcun dato. Infatti, nel caso in cui il Server notificasse una disconnessione, sposterebbe tutti i dati del Client in questione all'interno di una lista relativa agli utenti disconnessi.

L'eventuale riconnessione dovrà avvenire utilizzando lo stesso username e lo stesso colore scelti in partenza. In tal modo, il server verificherà l'esistenza tra gli utenti disconnessi di un profilo con tali caratteristiche e in caso positivo effettuerà la riconnessione.

Mentre un giocatore non è più connesso durante una partita, la sua situazione rimarrà invariata. Esso rimarrà sul campo da gioco e non potrà fare movimenti. A seguito di una riconnessione, ripartirà quindi dal punto in cui era precedentemente.