**Descrizione della Gestione dei Messaggi nel Progetto Client-Server**

**Introduzione**

Per gestire la comunicazione tra client & server per il progetto di my Shelfie abbiamo utilizzato un protocollo di messaggistica particolare, atto a permettere uno scambio di informazioni sufficientemente variegato e preciso, così da poter gestire le varie interazioni dei client con il server e viceversa.

**Architettura del Sistema**

Per la struttura del protocollo di messaggistica del progetto è naturalmente protagonista la parte dei socket (SocketClient,SocketServer), che viene utilizzata come coppia di interfacce per la comunicazione tra un singolo client e il server (ip\_server:16874). Lato server, una volta creato un collegamento client-server, la sua gestione viene demandata all’interno di un thread ad un oggetto di tipo ClientHandler, che gestisce in toto le possibili azioni dal punto di vista del networking del singolo client, oltre a gestire la ricezione lato server dei messaggi da parte del client a lui assegnato; questo viene fatto tramite il metodo readObject() della classe Socket. La gestione vera e propria dello smistamento dei vari messaggi vero e proprio viene svolta nella classe Server, tramite l’utilizzo di un design pattern di tipo Visitor.

Lato client, invece, una volta creata una connessione con il server da Cli o Gui, viene creato un thread per la lettura dei messaggi ricevuti dal server, che, quando riceve un messaggio, notifica un oggetto MessageHandler, al quale è demandata la gestione dello smistamento dei vari messaggi ricevuti, sempre tramite l’ utilizzo di un design pattern di tipo Visitor; inoltre, si occupa della modifica e aggiornamento del VirtualModel salvato in locale dal client.

Oltre alla semplice gestione dei messaggi veri e propri per il trasferimento di informazioni tra client e server, su un altro thread viene effettuato uno scambio periodico di messaggi senza un contenuto informativo per il semplice mantenimento della connessione; infatti, se il server non riceve alcun messaggio dal client per più di 5 secondi, il client viene disconnesso in automatico.

**Protocollo di Comunicazione**

Come protocollo di comunicazione viene utilizzato, al livello 3 della pila TCP/IP, il protocollo di trasporto affidabile TCP, in modo da avere una gestione automatica del flusso, controllo di congestione ed un controllo di errore; oltre che ad un servizio di trasporto affidabile ed una consegna delle PDU in ordine.

**Strutture dei Messaggi**

Tutti i messaggi vengono strutturati sulla base della classe Message, che viene estesa da ogni classe degli specifici messaggi, oltre ad implementare a sua volta l’interfaccia Serializable, in modo da poter essere utilizzati come oggetti da inviare mediante le socket.

Dalla classe padre Message, ogni sottoclasse eredita un parametro type(String) ed un parametro nickname(String), dove type contiene una stringa che specifica il tipo di messaggio che viene mandato, utilizzata lato server per un primo smistamento dei messaggi; mentre nickname indica il giocatore oggetto del contenuto del messaggio.Viene ereditato dalle sottoclassi anche dei Getter & Setter per i parametri del messaggio, 2 metodi per l’implementazione del Visitor pattern(uno lato client ed uno lato server) e un metodo che viene utilizzato per la gestione dei messaggi dal ClientHandler.

Le sottoclassi di Message formano l’insieme totale dei messaggi di tipo Server\_TO\_Client e Client\_TO\_Server; ognuno di essi fornisce una specifica informazione nel caso Client\_TO\_Server, che viene utilizzata dal server per modificare il modello, tramite il MatchController, in base alle scelte del client; oltre che per ricevere richieste di informazioni contenute nel server da parte di uno o più client. Nel caso dei messaggi Server\_TO\_Client, oltre a fornire informazioni al client oltre che a aggiornamenti del suo VirtualModel, fornisce ad esso anche il contesto della partita in cui si trova il server, fondamentale per lo sviluppo della View; infatti, tramite una amplia varietà di questo tipo di messaggi, siamo in grado di caratterizzare in modo più preciso ogni richiesta da fare al client.

In alcuni messaggi, nei quali è necessario trasportare un’informazione superiore alla semplice String, serializzabile per sua natura, viene implementato uno o più attributi aggiuntivi per allegare l’informazione di cui si necessita, andando cosi ad implementare Serializable nella classe che viene aggiunta come oggetto attributo al messaggio.

**Flusso di Gestione dei Messaggi**

Il flusso dei messaggi, partendo dal client, è questo:

1. Utente svolge un azione che richiede l’invio di un messaggio al server;
2. View notifica il ClientController dell’azione svolta;
3. Il ClientController tramite il suo attributo di tipo SocketClient e il metodo annesso sendMessage() procede a creare un messaggio del tipo richiesto e ad inviarlo al server;
4. Il ClientHanlder del client in questione riceve il messaggio procede ad una prima verifica del messaggio, per verificare se si tratta di un messaggio di ping, connessione,disconnessione o chat;
5. Se cio non si verifica, il messaggio viene passato ad un oggetto di tipo Server che tramite Visitor pattern affida il messaggio, ora non piu generico ma specifico, ad un oggetto di tipo MatchController;
6. L’oggetto di tipo MatchController procede alla gestione specifica del messaggio tramite handle();
7. L’informazione del messaggio può o passare al modello, generare una variazione nello stato del server o semplicemente far recepire una richeista del client;
8. In tutti i casi, comunque si procede, tramite un oggetto di tipo VirtualView(specifico per ogni client), alla creazione di un messaggio ed al suo inoltro al client mediante l’attributo ClientHandler, che possiede un attributo SocketServer;
9. Ora, il nuovo messaggio, ritornato al client, viene gestito da un oggetto di tipo MessageHandler contenuto nel ClientController, che procede all’aggiornamento del VirtualModel secondo le nuove informazioni fornite dal server e notifica la view della ricezione dello specifico messaggio passando come parametri le nuove informazioni;
10. A questo punto, chiusosi il ciclo, la view puo o mostrare a schermo le nuove informazioni oppure notificare il ClientController ora che ha le informzioni richieste per inviare un nuovo messaggio.

**Conclusioni**

boh