Relazione Progetto Text-Twist

Gabriele Barreca

Indice

1 Descrizione Progetto

Il progetto è diviso in tre Packages:

- Client: contenente tutte le classi riguardanti il Client
- Server: contenente tutte le classi riguardanti il Server
- Other: contenente tutte le strutture comuni ad entrambi, come gli utenti (classe User), i messaggi (classe Packet) e la lista degli utenti (classe UserList)

Sono presenti inoltre:

1 Due file .txt (ConfigurationC eConfigurationS) con lo scopo di inizializzare, rispettivamente, Client e Server
2 Il Dizionario fornito
3 Due strutture dati che vengono opportunamente create per gli utenti (Registro.json) e i punteggi (ScoreList.dat)

2. Comunicazione

Lo scambio di messaggi TCP avviene con oggetti di tipo Packet che vengono serializzati su un ObjectOutputStream e spediti. Ogni pacchetto è caratterizzato da:

1. Un tipo, che può essere: INVITE, CONFIRM, ERROR, REFUSE, UDP, RANKING.
2. Un array di User, usato per salvare la lista degli utenti con il loro punteggio.
3. Un vettore di Stringhe, contenenti
4. Le Stringhe Owner, Master e User. Le prime due sono praticamente equivalenti perché rappresentano la Stessa persona ma sono volutamente separati per motivi di chiarezza e leggibilità (sono usati in ambiti diversi). User rappresenta, appunto, il nome dell'utente che ha inviato il pacchetto.

La classe Packet ha due costruttori. Il primo, classico, prende come input solo il tipo che vogliamo assegnare al Pacchetto, e le altre strutture vengono riempite con i metodi appositi. Il secondo, invece, è specifico per il tipo UDP che prende insieme al tipo anche le stringhe da assegnare a Master e User e l'intero Size che indica il numero di parole che il Server si deve aspettare. Questa distinzione è dovuta al fatto che un pacchetto UDP non esiste, viene creato solo per facilitare il passaggio di informazioni fra il Thread che gestisce i Messaggi UDP del server e quello della partita, come verrà spiegato in seguito. È fatto semplicemente per non creare un'altra struttura dati apposita ed arrangiarsi con le classi già presenti.

2.1 FindOnlineServer/Client

Queste due classi gestiscono costantemente lo status degli utenti.

- Lato server: si manda un messaggio ogni 10 secondi (ONLINE?) per richiedere agli utenti registrati al canale multicast se sono ancora attivi. Invio e ricezione sono gestiti da due Thread differenti nella stessa classe.
- Lato client: si richiede la lista degli utenti (tramite Callback) al server e si aspetta il messaggio "ONLINE?" al quale si risponde con il proprio nome.

Considerando che un pacchetto può essere perso (viaggia con UDP) ogni qual volta che l'utente interagisce con il server segnalerà la sua presenza a quest'ultimo, evitando così di andare offline per non aver risposto al messaggio

3 Client

L'interfaccia grafica del Client si divide in due finestre, una per il Login/Registrazione (che successivamente fungerà anche per la visualizzazione degli Utenti) e una per la Partita. Il loro funzionamento verrà spiegato più avanti. Questa decisione è voluta non per ragioni pratiche, ma semplicemente estetiche.

3.1 Funzionamento Client

La parte centrale del Client è rappresentata nella classe GuiClient dove, oltre ad esserci tutte le informazioni relative all'interfaccia grafica, viene:

- Effettuata la connessione RMI con il Server e di conseguenza vengono gestiti i risultati ottenuti dopo le operazioni di Login, Registrazione e Logout.
- Mandato in esecuzione il Thread che si occupa di vedere l'elenco degli utenti Registrati.
- Creati i vari messaggi che vengono poi inviati al Server tramite TCP.

Fra le varie funzioni effettuate una nota particolare si pone per WordCheck. Sostanzialmente quando un utente scrive le parole che intende inviare al server, esse vengono inserite in una struttura dati solo se passano il WordCheck. Questa funzione consiste nel controllare che le parole non siano stringhe vuote o spazi e che le lettere usate siano appartenenti a quelle messe a disposizione. Il metodo con il quale viene fatto il controllo è il seguente:

1. La parola permutata, o semplicemente le lettere messe a disposizione vengono salvate in un Array di Caratteri.
2. Si controlla, carattere per carattere, se la lettera della parola appartiene a questo array. In caso affermativo si elimina dall' array tale carattere per evitare l'uso di lettere doppione. In caso negativo si restituisce False e la parola non viene inserita. Viene, quindi, restituito True se vengono rimossi tutti i caratteri in comune fra le due parole.

3.2 ClientDo

È la classe che si occupa di instaurare una connessione TCP(Autoclosable) con il Server e inviare i messaggi (o meglio Pacchetti) che sono stati precedentemente creati. In base al tipo di Pacchetto il metodo effettuerà le dovute operazioni:

- **INVITE**: viene inviata al Server la lista scelta dall'utente comprendente tutte le persone con il quale intende giocare. Questa lista viene inizializzata alla pressione del tasto 'Invite' e dopo svuotata per permettere di fare una nuova partita senza re-invitare i vecchi avversari. Successivamente si attenderà la conferma di inoltro da parte del Server.
- **CONFIRM**: inviata al Server quando si intende confermare l'invito da parte di un Utente. In caso di conferma, da parte di tutti, viene creato un nuovo Runnable che si occupa solamente della gestione della partita. Altrimenti viene notificato un errore.
- **RANKING**: inviata la Richiesta di Ranking al server si riceverà un pacchetto contenente la lista dei giocatori con il loro punteggio. Questa lista verrà trasformata in un'unica stringa (con i dovuti \n per ogni giocatore) e stampata a schermo trame un Information_Message.
- **REFUSE**: notifica il rifiuto della partita al Server.

3.3 MatchClient

È il Runnable che si occupa della partita e di instaurare una connessione UDP e MULTICAST con il Server.

1. Prima di tutto però vengono disabilitati tutti quei componenti superflui durante la partita (E.S. tasto 'Invite'), per impedire all'utente di compiere azioni non inerenti ad essa, ed abilitati quelli fondamentali.
2. Il Thread va in Sleep per due minuti per consentire all'utente di scrivere le parole, che verranno inserite in una struttura dati poi controllata da quest'ultimo. In caso la struttura dati non fosse vuota (il che vuol dire che non tutte le parole hanno passato il WordCheck), per ogni parola viene creato un DatagramPacket dalla Stringa con il seguente formato: *ParolaScrittaDall'Utente* + *NomeUtente* + *MasterDellaPartita* + *NumeroDiParoleScritte.* Questa decisione è stata presa in quanto, essendo UDP un protocollo non affidabile, sarebbe stato poco sicuro affidarsi ad un unico DatagramPacket contenente l'elenco di tutte le parole. Così in caso di problemi si minimizza il danno perdendo, per esempio, una sola parola anziché tutte. Nel caso non vengano inserite parole si invia comunque una stringa del formato sopra descritto dove, ovviamente, si ha *NumeroDiParoleScritto* = 0. Si è scelto così per evitare di aspettare quei 5 minuti nel server e si è inteso "l'utente non invia un messaggio" come un generico problema di invio (o ricezione). Infatti in caso di errore può comunque mandare i risultati dopo lo scadere del tempo, e il client riceverli.
3. Una volta inviate tutte le parole si aspettano i DatagramPacket provenienti dal Server comprendenti una stringa composta da: *Punteggio* + *NomeUtente*, che vengono inseriti in una struttura dati, ordinati in modo decrescente e mostrati all'utente.
4. Infine vengono riabilitati e disabilitati i vari componenti e azzerata la struttura dati delle Parole.

4 Server

Il Server usa diversi Thread per gestire tutta la struttura del gioco. Prima di avviare le varie connessioni però, si salva in una lista di tipo UserList tutto l'elenco degli utenti registrati, salvati nel file .json, e successivamente si assegnano i punteggi letti dal file .dat. Quest'ultimo è memorizzato e letto tramite NIO con un FileChannel e un ByteBuffer di 1024 bytes. Vengono poi avviati la connessione RMI, un Thread ClientHandler che gestisce tutti i messaggi UDP e un Thread ClientHandler per ogni connessione TCP che si viene a creare con il Client.

4.1 ClientHandlerUDP

Il compito principale del Thread, oltre a quello di ricevere tutti i DatagramPacket e il rispettivo pacchetto (come spiegato sopra), è lo smistamento di quest'ultimi al Thread della partita corrispondente. Lo smistamento avviene attraverso una CuncurrentHashMap<String, ArrayBlockingQueue<Packet>> secondo questo ragionamento:

1. Ricevuto il DatagramPacket dal Client si estrae il Master della partita, che considerando l'unicità dell'Username di un utente è da considerarsi come un ID univoco della partita.
2. Si accede così alla BlockingQueue corrispondente che simboleggia l'elenco di tutti gli utenti di quella determinata partita con le rispettive parole. Viene usata questa struttura perché ottimale per il problema Produttore/Consumatore, in quanto gestisce automaticamente le attese di inserimento e prelievo.

3.

4.2 ClientHandlerTCP

È Thread che si occupa di gestire tutti i pacchetti TCP ricevuti. In base al tipo di pacchetto effettua le dovute operazioni:

- **INVITE:** Viene controllato lo stato di tutti gli utenti invitati. Se tutti Online allora vengono richiamate le CallBack per notificare l'invito e inviano un pacchetto di conferma al Client. Successivamente viene avviato un Callable che aspetterà la risposta di tutti gli utenti. Il Callable ritornerà un risultato che verrà poi inserito in una CuncurrentHashMap<String,Future<String>> per rendere concorrente l'accesso ai risultati. Sono previste altre due CuncurrentHashMap: Una che rappresenta l'elenco degli utenti online(UserList), che devono accettare e l'altra che indica il Thread di ogni singolo Callable, necessario per spegnere successivamente il Thread.
- **CONFIRM:** Qui si sfrutta principalmente la seconda HashMap descritta sopra. Dal pacchetto ricevuto si estrae il Master della partita per andare a cercare quindi la lista inerente a quest'ultima. Essendo il pacchetto di tipo conferma verrà inoltre incrementato un AtomicInteger all'interno della lista, con lo scopo di tener conto il numero di utenti che ha accettato. Si è scelto un AtomicInteger per andare a rendere Thread Safe la scrittura e la lettura del valore, in quanto può essere scritto e letto nello stesso momento da Thread differenti.
- **REFUSE:** Si cambia il valore dell'AtomicInteger in negativo per andare ad annullare la partita (il meccanismo sarà spiegato meglio nel prossimo paragrafo)
- **RANKING:** Si sincronizza la struttura dati iniziale del server, quella contenente tutti gli utenti e i rispettivi punteggi, per far sì che nessun altro nel frattempo possa accedervi. La si inserisce in un pacchetto con solo Nome e Punteggio, la si ordina e la si spedisce.

4.3 PreMatch

Lungo il tempo di attesa, viene costantemente controllato il valore dell'AtomicInteger che tiene traccia degli utenti che hanno accettato. Se negativo, allora il Callable ritorna null e viene dunque segnalato all'utente un messaggio d'errore. Se non-negativo viene confrontato con la dimensione della lista degli utenti associati alla partita. Dunque se il numero risulta essere uguale allora vuol dire che tutti gli utenti hanno accettato. Di conseguenza viene creato un Runnable che si occupa di gestire i messaggi della partita e viene fatta ritornare, al Callable, la sequenza di lettere che vengono poi notificate all'utente. Per scegliere la sequenza di lettere si eseguono i seguenti passaggi:

1. Si setta il file-pointer del dizionario ad un punto casuale, grazie alle classi Random e RandomAccessFile. Successivamente vengono fatte due ReadLine perché dopo il primo Seek(n) c'è una piccola probabilità che la linea letta non inizi dal primo carattere. Viene dunque memorizzata la seconda, se non è EOF o se di 6 o 7 caratteri, altrimenti si riprova settando il file-pointer in un altro punto.
2. Trovata la parola si salvano in un Array tutte le permutazioni, ricavate ricorsivamente, e se ne sceglie una in modo casuale. Questa permutazione sarà la sequenza di lettere che verrà notificata al giocatore.

4.4 MatchServer

È il Thread che controlla la validità delle parole e di conseguenza che aggiorna i punteggi associati agli utenti. Prima di controllare per un massimo di 5 minuti le condizioni necessarie che precedono di eseguire il compito principale, il Thread si occupa di 3 cose importanti:

1. Inizializza un HashMap dove assegna come chiave: utente e valore: False, per indicare che non sono state ricevute tutte le parole che ci si aspetta che quell'utente abbia inviato.
2. Inizializza una CuncurrentHashMap<String,ArrayList<String>> creando per ogni utente un Array vuoto che verrà, eventualmente, riempito con le parole ricevute
3. Va in sleep per 2 minuti. Questa scelta è dovuta al fatto che il Cliente invierà le parole solo dopo che i 2 minuti di tempo messi a disposizione sono scaduti, non durante. Considerando inoltre che questo Thread viene creato appena tutti gli utenti hanno accettato, si avrebbero 2 minuti in più di attesa passiva.

Successivamente il Thread attende l'attesa di tutti i messaggi previsti. L'attesa, lunga al più 5 minuti, viene interrotta se la struttura dati descritta nel punto 1. Non contiene più nessun valore di tipo False. Infatti quando un messaggio viene ricevuto si controlla se la dimensione dell'array delle parole ricevuto, a lui associato, è uguale al numero indicato dal messaggio. In tal caso il valore della mappa viene settato a True e quando non vi sono più False allora vuol dire che tutti i messaggi sono stati ricevuti. Prima di inviare via Multicast i messaggi contenenti i punteggi della partita va controllata la validità delle parole e aggiornato il database.

Ogni parola appartenente all'array viene controllata se presente nel dizionario. In caso positivo si aggiorna il valore del punteggio associato all'utente e salvato in una struttura temporanea che salva solo i punteggi della partita corrente. Servirà dopo per l'invio dei dati. L'aggiornamento nel Database, però, viene effettuato solo dopo aver controllato tutte le parole dell'array, per limitare così le scritture e gli accessi al file. Verrà discusso più avanti in modo dettagliato.

Per quanto riguarda la ricerca sul Dizionario, quest'ultima è fatta tramite una Ricerca Binaria per ridurre il numero di confronti, considerando la dimensione del file e il numero delle sue righe. La Ricerca Binaria è fatta in modo ricorsivo ed in modo molto simile alla scelta delle lettere per la partita. Ossia vengono usati RandomAccesFile con la loro funzione Seek e la doppia ReadLine per scegliere la parola da confrontare, il resto è noto.

5 Database

Si è deciso di dividere il database degli utenti con le rispettive password da quello degli utenti con i rispettivi punteggi semplicemente per volere lavorare su più tipi di file in modo diverso e gestire meglio la concorrenza su di essi.

5.1 .Json

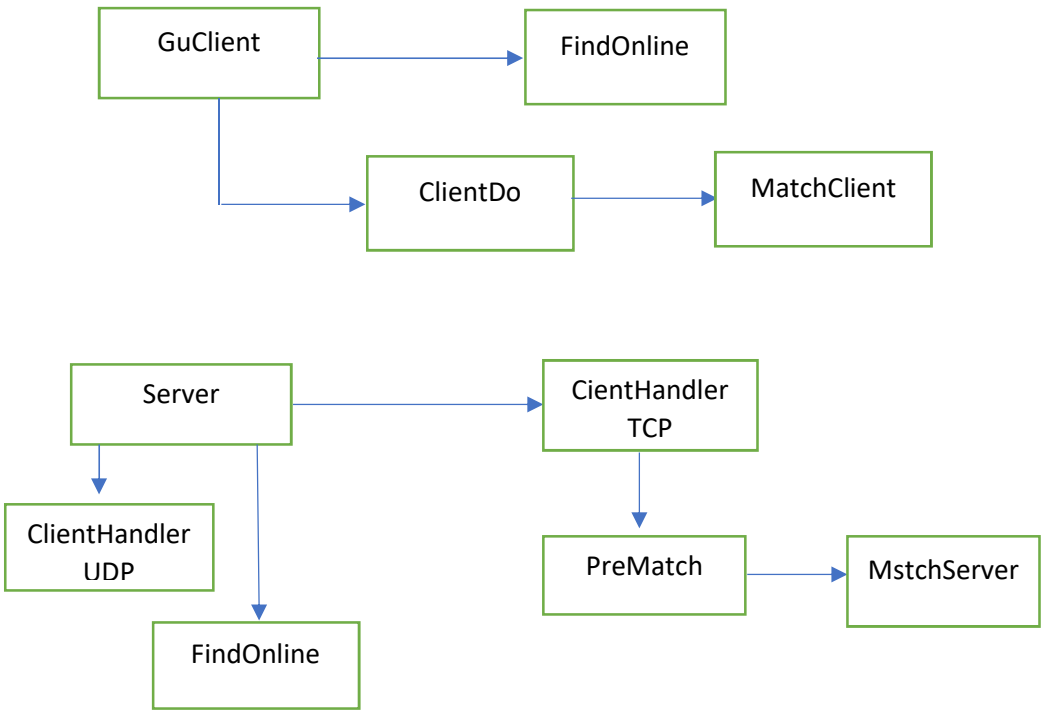
Il registro Utenti-Password viene scritto grazie a dei FileWriter che scrivono la stringa Json parsata dall'utente. Infatti la classe User estende la classe JsonWriter ereditandone tutte le caratteristiche e i metodi che contraddistinguono il json. Quando, tramite RMI si effettua una registrazione, prima di scrivere l'utente nel file, viene copiato l'intero registro in un JsonArray. La cosa è inefficiente ma serve e scrivere tutti gli utenti in un unico array json e rendere così la lettura del registro più facile.

La scrittura e la lettura sicura del file è garantita dal fatto che i metodi RMI che si occupano di ciò sono Synchronized.

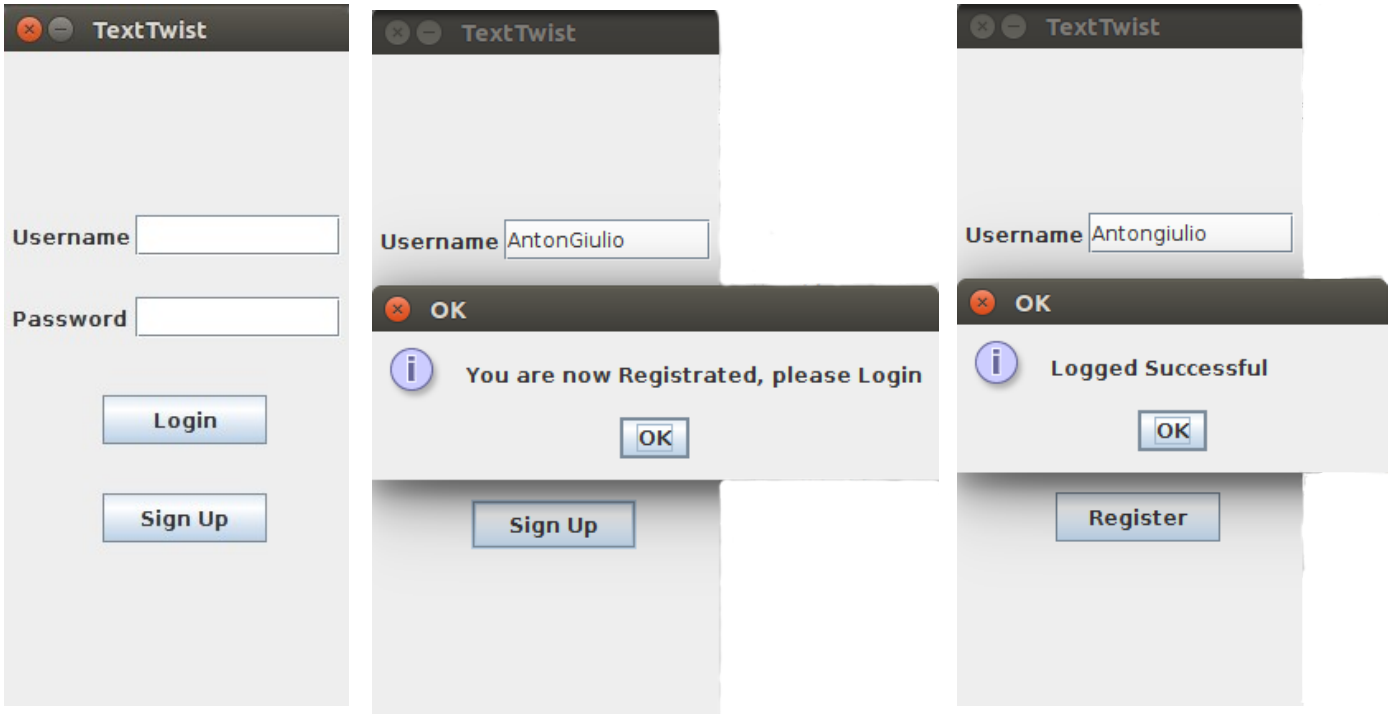
5.2 .Dat

La scrittura e la lettura del file contenente gli utenti e i loro punteggi viene fatta tramite FileChannel e quasi sempre con un ByteBuffer da 1024 bytes. La concorrenza è garantita dall'uso di un oggetto di tipo FileLock. Sostanzialmente viene creata una sorta di Lock Globale che impedisce ad altri Thread di scrivere/leggere su un determinato file nonostante si usino FileChannel diversi per lo stesso percorso. La Lock viene acquisita prima di ogni operazione ed automaticamente rilasciata alla chiusura del file. Il quasi all'inizio del paragrafo si riferisce al fatto che quando si va a modificare il punteggio di un singolo utente anziché usare un ByteBuffer si usa una MappedByteBuffer in quanto molto più veloce per modificare il contenuto di un file.

6 Istruzioni per l'uso e schema

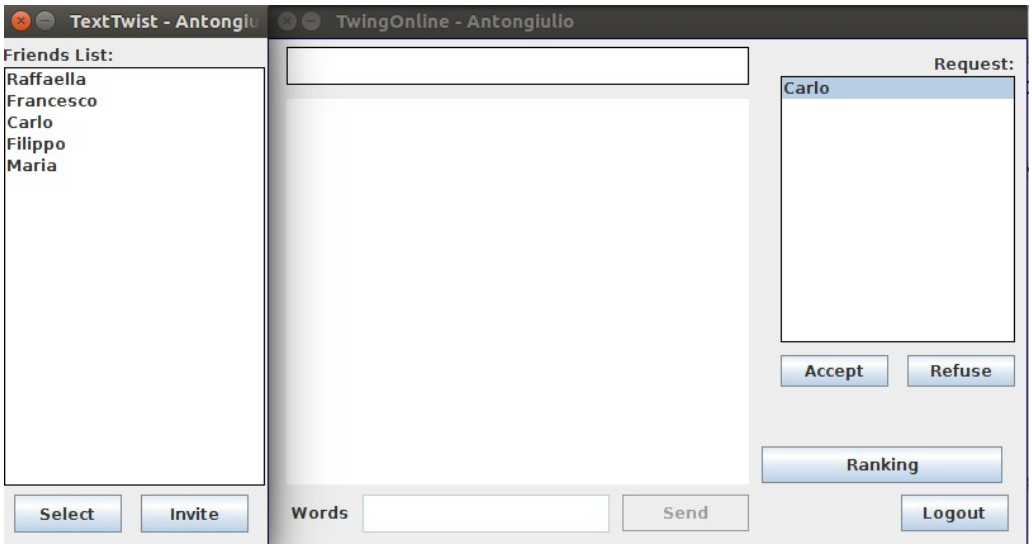


Fare doppio click sui file jar, oppure da terminale con i comandi: java -jar TwistClient.jar/ TwistServer.jar. I due eseguibili si devono trovare nello stesso percorso della cartella FILE. Potrebbe essere richiesto l'uso di Java8.



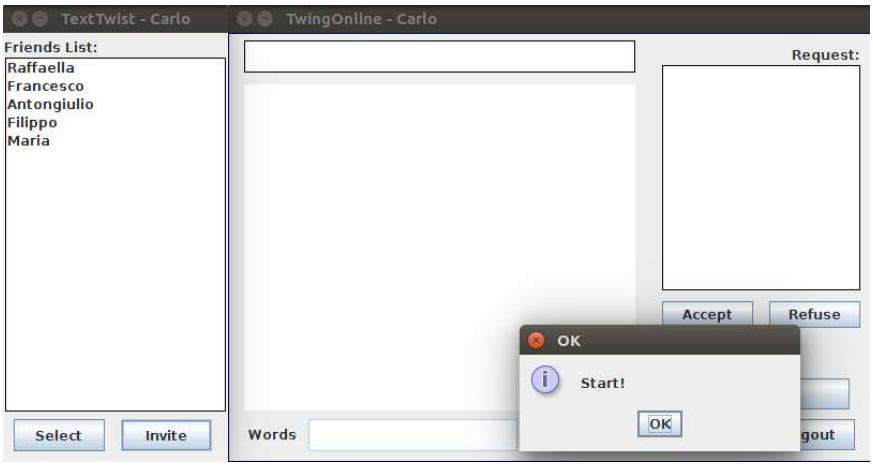
Durante la registrazione non ci sono restrizioni per il tipo di Username. L'unica cosa è che non vi possono essere 2 Username uguali.

Il pannello di Login si trasformerà in quello della lista amici. Verrà visualizzato inoltre un nuovo pannello contenente tutto il necessario per effettuare il resto delle funzioni.

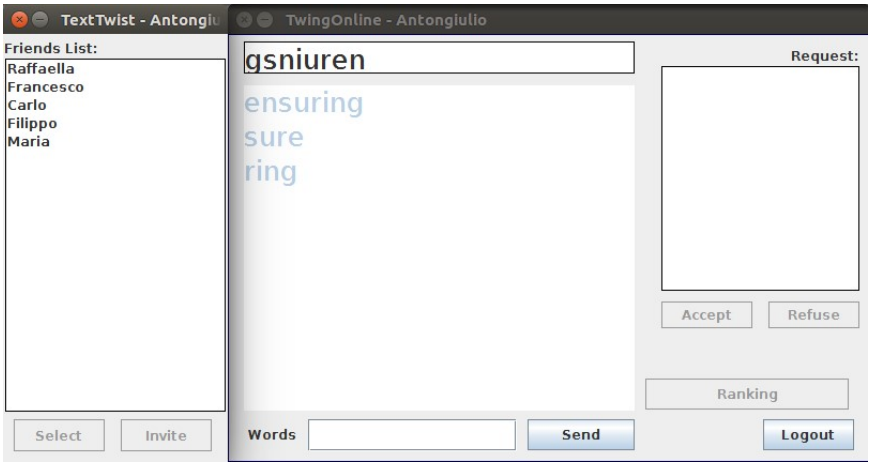


Per selezionare gli avversari bisogna sceglierli nella lista amici e selezionarli uno ad uno. Solo dopo aver scelto tutti gli avversari si può premere Invite per spedire l'invito.

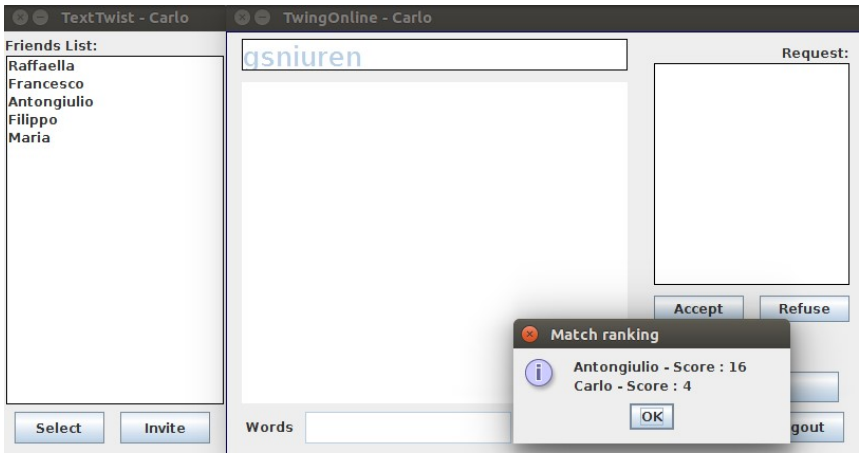
Nel secondo pannello verrà raffigurato l'elenco degli inviti, che possono essere rifiutati o accettati con i rispettivi bottoni, dopo aver selezionato la richiesta.



La partita inizia solo dopo aver premuto ok!



Verrà fornita la parola dal server e tutti i tasti non inerenti alla partita verranno disabilitati. Per inserire la parola bisogna semplicemente scriverla nell'apposito spazio dedicato e premere Send una volta finito. Le parole scritte vengono stampate nell'area di testo per averne un resoconto ed evitare di scrivere doppioni.



Una volta finita la partita la classifica temporanea si presenta così, in ordine decrescente in base al punteggio. Cliccando sul bottone Ranking verrà visualizzata la classifica generale nel medesimo formato.

Il tasto Logout segnala tramite RMI al server che l'utente si sta scollegando. Vengono così chiusi i Thread aperti e il secondo pannello tornando alla sola schermata di Login.

Se invece si chiude il programma tramite il pulsante messo a disposizione della finestra si ottengono gli stessi risultati del pulsante Logout con la differenza che qui verranno chiusi entrambi i pannelli.