WordQuizzle

Esame finale Laboratorio di Reti

Progetto individuale di Luca Murgia

Università di Pisa

Maggio 2020

Indice generale

Introduzione

Questo progetto di fine anno, realizzato per il corso di Laboratorio di Reti, prevede la realizzazione di WordQuizzle, un sistema di sfide di traduzione italiano-inglese tra utenti registrati al servizio.

Per questo progetto ho voluto dare importanza alla modularità del codice e all'interfaccia grafica, cercando di rendere il funzionamento del programma il più intuitivo e semplice possibile per l'utente.

A questo proposito ho scelto di utilizzare una interfaccia grafica minimale per il menù principale, creata tramite la libreria grafica Javax swing.

L'utente ha quindi la possibilità di visualizzare in ogni momento una lista di tutti amici aggiunti tramite il sistema e tutte le funzioni che esso può eseguire.

Architettura software e scelte di progettazione

Il software è diviso in due package principali:

- **client**: Fornisce un insieme di interfacce grafiche, reader UDP e funzioni da richiedere al server tramite TCP stream.
- **server**: Fornisce un sistema di gestione dati centralizzato, capace di memorizzare e modificare dati utente e punteggi, comunicare con client tramite canali TCP e UDP e utilizzare il servizio REST "Mymemory" per valutare le risposte ricevute.

CLIENT

Il Client prevede due diverse classi grafiche create tramite javax swing:

- la classe **LoginGUI** contiene l'interfaccia di login
- la classe **MainMenuGUI** contiene l'interfaccia del menù principale

Il progetto possiede una struttura pseudo-MVC, in cui la classe **ControlClient** rappresenta il controller.
Essa, infatti, crea e comunica con le interfacce grafiche (View) precedentemente citate e gestisce la comunicazione con il server, che elabora i dati ricevuti.

La **comunicazione Client-Server** avviene tramite invio di messaggi di formato standard (JSON) per mezzo di una connessione TCP avviata al momento del login del client.

I messaggi spediti e ricevuti sono gestiti in modo univoco per mezzo del **campo ID**, rappresentante l'operazione richiesta da parte del client o la risposta specifica da parte del server.

SERVER

Il controller del server è costituito dalla classe **ControlServer**, la quale:

- Crea e gestisce il **registro challenge** ed il **registro utenti**, implementati per mezzo di **ConcurrentHashMap**, in modo che le operazioni di modifica vengano rese thread safe.
- Crea e gestisce la connessione TCP con i vari client: un Threadpool multithread mette in comunicazione le classi ControlClient con il loro specifico **Handler** nel lato server, uno per ogni client, che riceve e gestisce le operazioni che vengono richieste.

Il registro utenti viene automaticamente memorizzato all'interno di un rispettivo file tramite i metodi della classe astratta **SaveLibrary**, invocati alla chiusura del server.

La registrazione degli utenti viene effettuata tramite l'invocazione del metodo remoto register.

Le classi ControlClient e ControlServer, a questo proposito, istanziano due altre classi: la classe **RegistrationClient** e la classe **RegistrationServer.**
La prima invoca il metodo remoto sulla seconda, passando come parametro l'username e la password dell'utente da registrare nel **Registro Utenti**.

Thread, Concorrenza e Strutture Dati

In questa sezione sono elencati i vari thread che vengono avviati durante l'esecuzione del programma

LATO CLIENT

Le classi che gestiscono l'interfaccia grafica vengono avviate come **thread separati**, in questo modo più interfacce di visualizzazione possono potenzialmente essere visualizzate insieme da un solo client.

La classe ControlClient avvia diversi thread grafici che verranno successivamente chiusi tramite il metodo **dispose()** fornito dalla libreria javax swing.

ControlClient genera inoltre un **thread UDP** il cui compito è ricevere e gestire l'inoltro delle richieste di sfida.

L'insieme di tutti i thread del client è associato ad un **unico handler** nel lato server che, quando necessario, può richiedere la lock sulle strutture dati condivise.

LATO SERVER

La funzione **main** del server genera un **unico thread grafico** avente al suo interno il log delle informazioni sul sistema.

Il thread grafico istanzia inoltre un **ControlServer thread** che rimane in ascolto sul canale TCP.

ControlServer genera, tramite un threadpool, tanti **thread Handler** quanti sono i client che hanno effettuato il login.

Un **thread RMI** rimane attivo per gestire le richieste remote di registrazione dei client.

Quando la finestra del server viene chiusa
- Vengono salvati tutti i dati utente presenti nel registro all'interno del file userMemory.
- Il registro RMI, il suo binding e lo stub del ControlServer precedentemente esportati vengono rimossi tramite **unexport()** e **unbind()**
- I thread handler vengono interrotti e terminati.

STRUTTURE DATI

Le strutture dati che ho scelto di utilizzare sono:

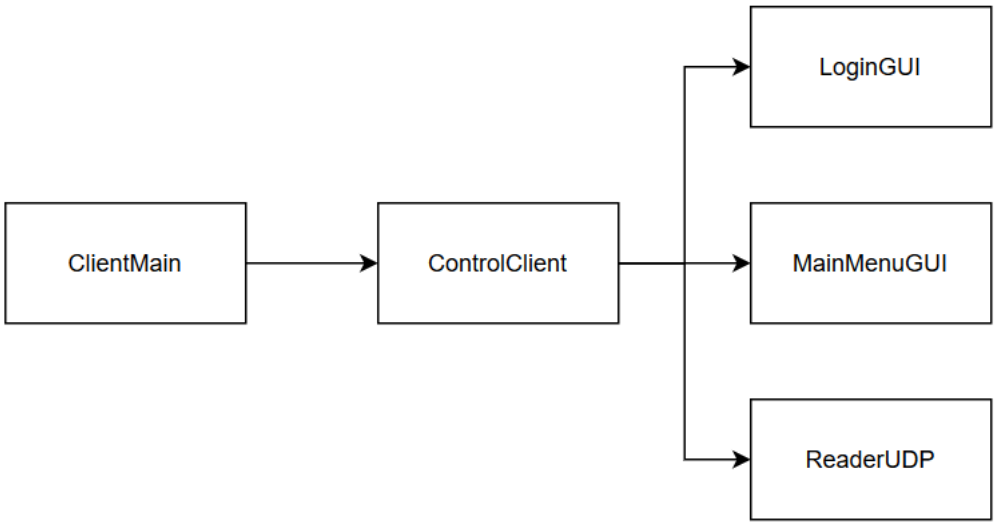
- Un **ArrayList dictionary**:
  - Contiene stringhe rappresentanti solo termini in italiano
  - Il suo contenuto viene prelevato da un file omonimo all'avvio del server
  - La struttura è accessibile in sola lettura.
- Una **ConcurrentHashMap** per il **registro utenti** in cui:
  - Il nome utente ha la funzione di chiave della mappa
  - I valori contenuti nel registro sono stringhe in formato **Json**
  - Il registro utenti è l'unica struttura dati persistente.
- Una **ConcurrentHashMap** che tiene traccia degli **utenti online** o disponibili:
  - Il nome utente ha la funzione di chiave della mappa
  - I valori contenuti sono valori booleani equivalenti a true quando un utente è disponibile e false quando non lo è
  - Quando un utente effettua il logout la coppia chiave-valore relativa ad esso nella hashMap viene rimossa.
- Una **ConcurrentHashMap** che rappresenta il **registro delle challenge** in corso:
  - Utilizza la combinazione di due nomi utente come chiave
  - Il valore associato alle chiavi è una istanza della classe Challenge

Una istanza della classe Challenge, unica per ogni coppia di utenti, viene creata nel momento in cui due utenti si sfidano e inserita nella relativa HashMap, accessibile dagli handler di entrambi i client. Contiene metadati relativi alla sfida, funzioni di gestione e timer

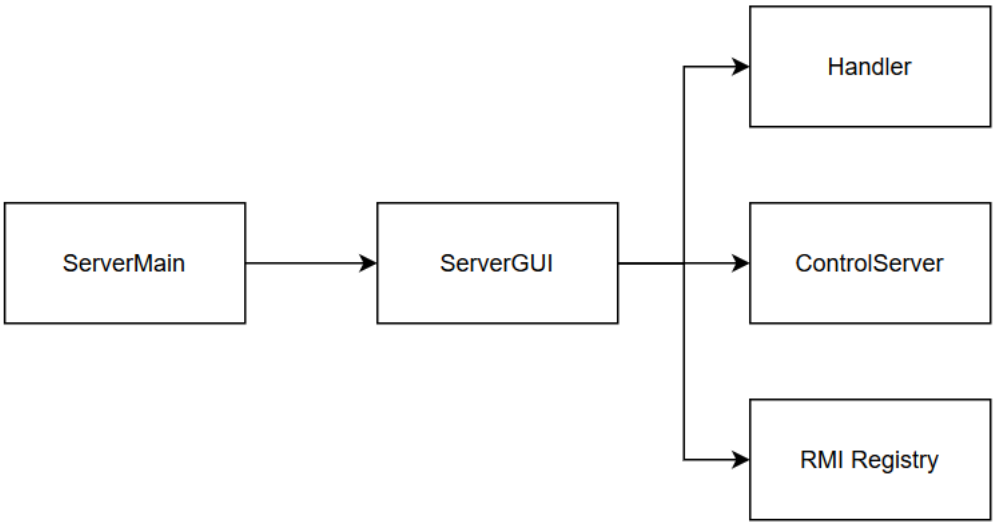
Un particolare Message ID fa sì che quando una sfida viene iniziata gli utenti coinvolti vengano riconosciuti come **"unavailable"** dagli altri utenti.

- Un utente unavailable è incapace di sfidare altri utenti o ricevere richieste di sfida
- I due utenti tornano **available** quando la sfida termina o viene annullata
- La lock sul booleano viene gestita implicitamente dalla struttura dati concorrente

SCHEMA THREAD: LATO CLIENT



SCHEMA THREAD: LATO SERVER



Classi

In questa sezione viene fornita un elenco delle principali classi del programma, affiancate a una breve descrizione del loro funzionamento.

CLASSI: LATO CLIENT

- **ClientMain**: Avvia il ControlClient.
- **ControlClient**:
  - Riceve i comandi dai thread di interfaccia
  - Spedisce richieste al server in formato JSON ed elabora le risposte
  - Istanzia la classe RMI che lancia la funzione remota register all'interno del server.
- **ReaderUDP**: Servizio UDP multicast, riceve ed elabora richieste dal server sotto forma di datagrammi, è utilizzato per l'inoltro di richieste di sfida, la terminazione o cancellazione delle sfide e per l'invio risultati.
- **RegistrationClient**: classe contenente il registry del server e il suo stub, questa classe può invocare sul server il metodo register, che salva i dati di un nuovo utente all'interno del registro utenti, passando come parametro il nome utente e la password del client.
- **LoginGUI**: interfaccia utente che mostra una semplice schermata di login, possiede:
  - Due campi di testo dove inserire nome utente e password
  - Due bottoni per effettuare le operazioni di login e registrazione.
- **MainMenuGUI**: interfaccia utente che mostra il menù principale, possiede:
  - Bottoni rappresentanti le funzioni invocabili
    - I parametri delle funzioni invocate dai bottoni vengono scelti sia da riga di testo, per mezzo di JInputPane, che dagli elementi selezionati all'interno delle Jlists
  - Una JList interna a un JScrollPane, rappresentante la lista di amici aggiunti e con cui è possibile iniziare una sfida
  - Dei JLabel rappresentanti il nome utente e il punteggio complessivo raggiunto

CLASSI: LATO SERVER

- **ServerMain**: Avvia il ControlServer
- **ControlServer**:
  - Avvia il servizio di registrazione e quello di login
  - Entra in un loop in cui accetta le richieste di login del client e, per ogni richiesta, genera un Handler che viene eseguito da un Threadpool multithread.
  - Ha accesso al Registro dei documenti e al Registro Utenti, che può caricare da file e aggiornare.
- **RegistrationServer**: Offre un servizio RMI con registry e stub del ControlServer, contiene il metodo remoto register().
- **Handler**: gestore del client, possiede un loop continuo in cui accetta le richieste, le elabora e restituisce un messaggio di risposta.
- **SaveLibrary**:
  - Contiene funzioni invocate dal server per il salvataggio su file dei dati utente e dei documenti.
  - I dati salvati sono riutilizzabili dal server in future esecuzioni.
- TranslationLibrary:
  - Libreria di funzioni di traduzione relative al servizio MyMemory e al dizionario ad esso associato
- **ServerGui**: Interfaccia grafica minimale del server

Modalità di esecuzione

In questa sezione viene descritta la sequenza di eventi principale, sia dal lato client che dal lato server, nel caso in cui un utente C1 voglia registrarsi al servizio, aggiungere un utente C2 come amico e coinvolgerlo in una sfida di traduzione

pre-condizioni: Il server è attivo al momento in cui il client viene avviato

REGISTRAZIONE, LOGIN E RICHIESTA DI CHALLENGE

- Quando il server viene avviato la classe **ServerMain** istanzia la classe **ServerGUI**, viene quindi mostrata l'interfaccia grafica del server
  - Una istanza della classe **ControlServer** viene avviata
  - ControlServer scarica dai file **UserMemory** e **Dictionary** il registro utenti e il dizionario
  - Viene attivato il servizio RMI di registrazione, tramite la classe **RegistrationServer**, esportando uno **stub di ControlServer**
  - Viene creato un socket e un **ThreadPool**.
- **ControlServer** entra in un loop fornendo il **servizio di login**, rimane in attesa di richieste di Login da parte dei client
- **ClientMain** comincia l'esecuzione, istanzia e avvia **ControlClient**
- **ControlClient** mostra l'interfaccia di login avviando **LoginGUI** in un thread separato.

- C1 preme il bottone Register
  - Vengono prelevati **username** e **password** dai relativi campi di testo
  - La registrazione viene avviata nel server con i parametri prelevati, attraverso **RegistrationClient**
  - l'**Username** usato come parametro della funzione register() viene confrontato con quelli presenti nel registro utenti
  - nel caso non ci fosse un match viene creato un JsonObject relativo al nuovo utente e inserito nel registro utenti.
  - I dati inseriti in fase di registrazione possono ora essere utilizzati per effettuare il login al servizio.

- C1 preme il bottone **Login**
  - **Username** e **Password** vengono prelevati dai relativi campi di testo
  - **ControlClient** incapsula i dati in una apposita **richiesta JSON**
  - Viene lanciata la **Server.accept()** a seguito della connessione da parte di C1
  - Il ClientSocket viene quindi passato al suo **Handler** appena istanziato, che viene eseguito da uno dei thread del pool ed **entra in un loop di attesa richieste**
  - Viene inviato un messaggio di riscontro a C1
  - Se il messaggio di riscontro è positivo il ControlClient chiude **LoginGUI** lanciando una dispose() e apre **MainMenuGUI**.

- C1 preme il bottone **Add Friend** in **MainMenuGUI**
  - La pressione del bottone viene catturata dal suo **actionListener**
  - Viene richiesto, tramite **JOptionPane**, il **nome dell'utente** da aggiungere alla propria lista di amici
  - I dati vengono passati al **ControlClient**, incapsulati in un messaggio di richiesta e inviati all'handler relativo
  - Nel caso in cui l'utente C2 sia registrato al servizio, viene aggiunto il suo nome alla lista di amicizie di C1, nel registro utenti, la lista amici presente nel menù principale di C1 viene aggiornata
  - la funzione **refresh** del MainMenuGUI di C2 mostrerà ora C1 all'interno della propria lista di amici
  - Una volta selezionato un amico è possibile premere il bottone **Challenge**.

- C1 preme il bottone **Challenge (C2)**
  - Alla sua pressione viene inviato al server un messaggio **CHALLENGE**
  - Il server controlla l'availability di C1 e C2, se questi sono entrambi disponibili viene inoltrato il messaggio di richiesta a C2 e visualizzato tramite un JSelectionPane
  - Viene istanziata la Challenge e inserita all'interno del **Challenge registry**
  - viene fatto partire il timer di annullamento sfida

- C2 accetta la sfida
  - Viene inviato al server un messaggio READY
  - Alla ricezione del messaggio il server cancella il timer di annullamento e avvia il timer sfida
  - Il server invia a entrambi i client un datagram BEGIN
  - la sfida ha inizio, il bottone Challenge di C1 e C2 diventa non visibile, al suo posto viene reso visibile il bottone Question

- C1 o C2 preme il bottone **Question**
  - Viene inviato al server un messaggio **QUESTION**
  - Il server risponde con la domanda da inviare, prelevata dalla classe Challenge
  - Vengono successivamente inviate tre parole da tradurre e visualizzate tramite un JinputPane, la risposta viene inviata al server tramite un messaggio **ANSWER**
  - Il server verifica la correttezza della risposta e salva i relativi punti all'interno della challenge, i punti verranno poi aggiunti al registro utenti una volta terminata la sfida
  - Alla quarta pressione del bottone question il client torna in challlenge mode e viene inviato al server un messaggio di **FINISH**

- La sfida termina
  - Il server invia un datagram ai due utenti con i risultati della sfida
  - Vengono aggiunti i punti della challenge al registro utenti
  - Entrambi gli utenti tornano nello stato available, il bottone question viene sostituito dal bottone challlenge e si ritorna allo stato iniziale.

- L'interfaccia del server viene chiusa
  - Il **ControlServer** riceve una interrupt
  - Il **registro utenti** viene salvato nel file **UserMemory**
  - La funzionalità **EXIT_ON_CLOSE** di Java swing garantisce che vengano terminati tutti i thread dipendenti da ServerGUI

(Da implementare: graceful termination)

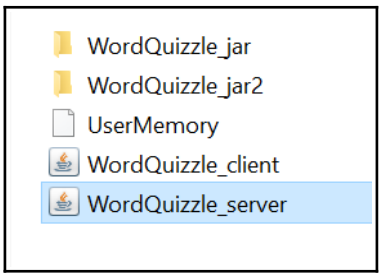
- Viene lanciata una **Unexport** verso tutti gli oggetti remoti esportati nel **RegistrationServer**
- Viene lanciata una **Unbind** verso il registro
- I thread in esecuzione nel **ThreadPool** ricevono una interrupt
- il server smette di accettare richieste di login e registrazione, termina la sua esecuzione.

Manuale Utente

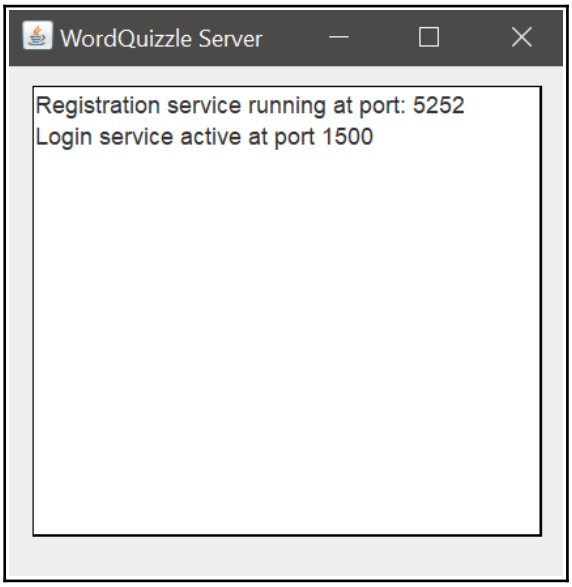
In questa sezione è presente un insieme sintetico di istruzioni per avviare e utilizzare il software

LATO SERVER

Per avviare il server aprire il file WordQuizzle_server.jar nella cartella artifacts.



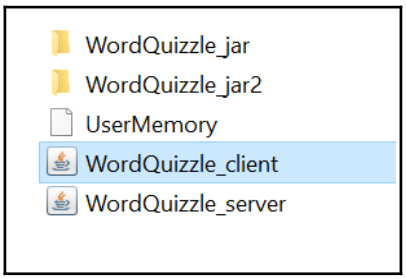
Attendere che vengano visualizzati i messaggi di avvenuto avvio



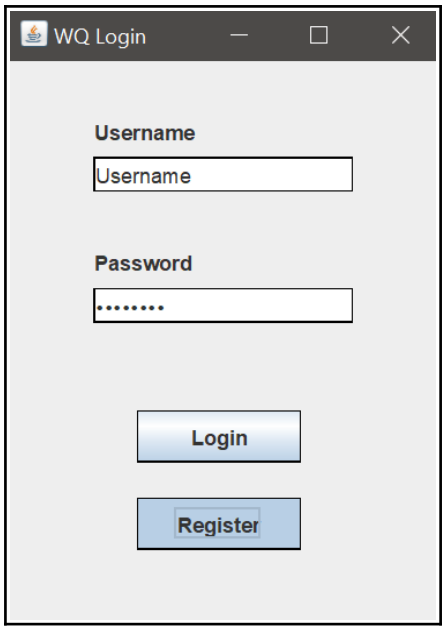
Per interrompere l'esecuzione e terminare il programma salvando i dati chiudere semplicemente la finestra.

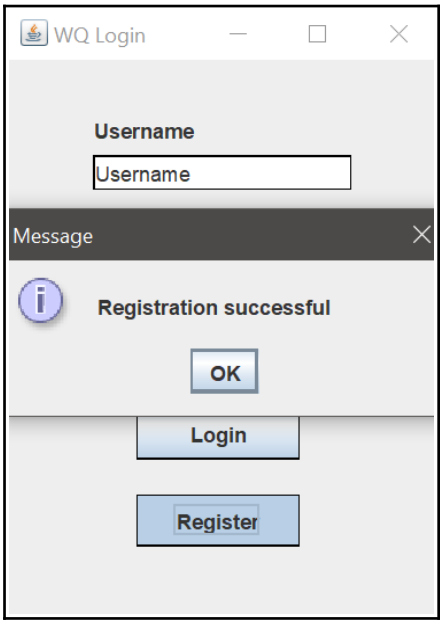
LATO CLIENT

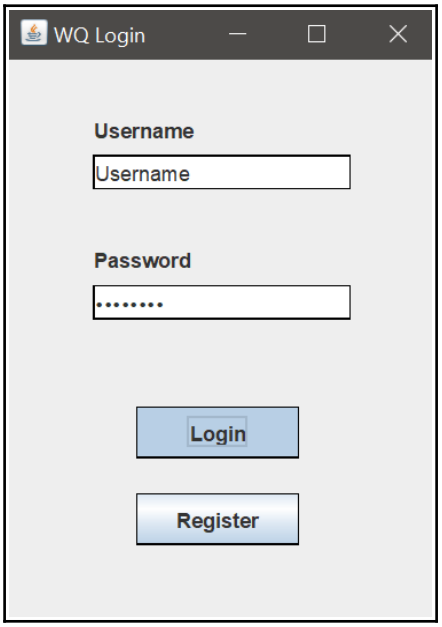
Per avviare il client aprire il file WordQuizzle_client nella cartella artifacts.



Verrà visualizzata la schermata di Login, per effettuare la registrazione digitare il proprio nome utente e la propria password e premere il bottone Register.



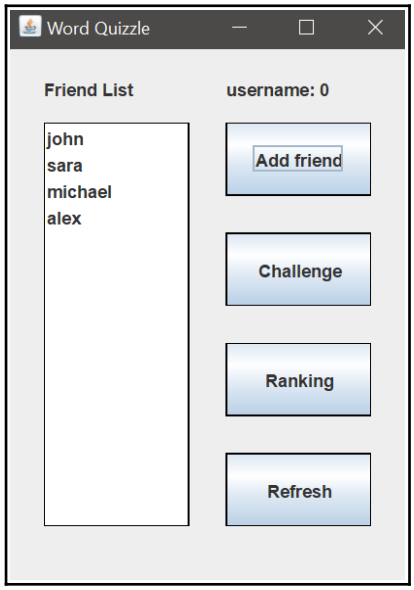


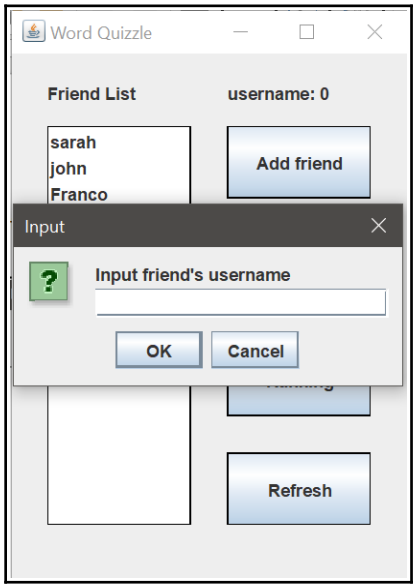


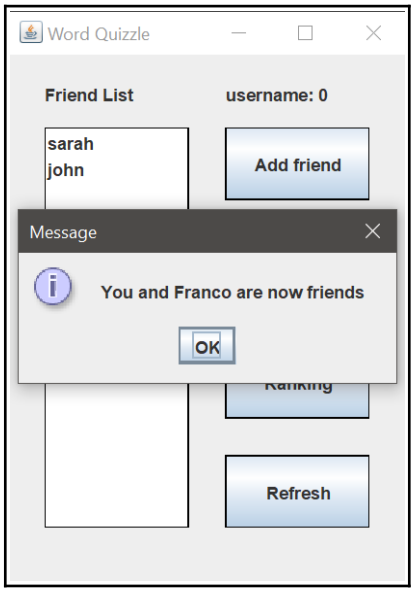
Da questo momento in poi potrà essere effettuato il login con le credenziali inserite in fase di registrazione.

Una volta effettuato il login verrà mostrata la schermata principale del programma.

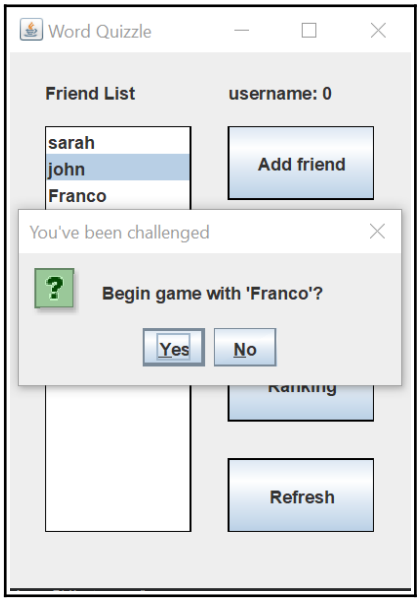
- Il tasto **Add Friend** permette di aggiungere un utente alla lista degli amici

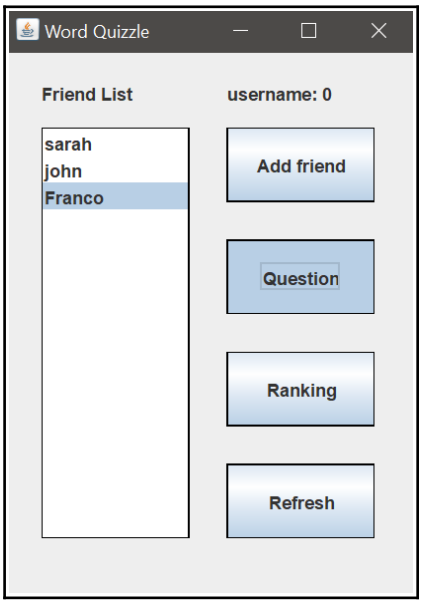


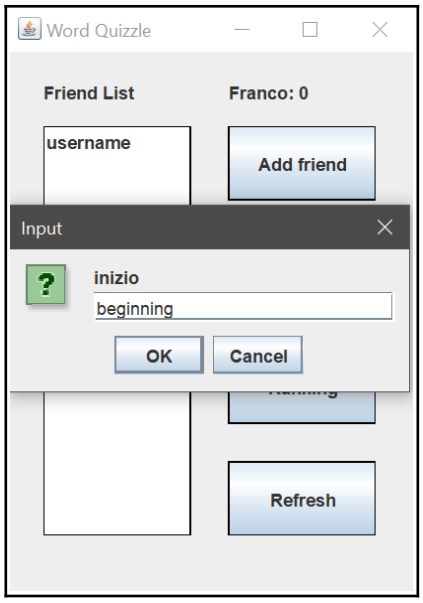




- Il tasto **Challenge** permette di sfidare l'utente selezionato
- Il tasto **Question** permette di richiedere una domanda al server e inviare la risposta







- Il tasto **Refresh** permette di aggiornare i dati relativi al proprio punteggio e alla propria lista amici
- Il tasto **Ranking** permette di mostrare in un JTextDialog la classifica dei propri amici, in ordine di punteggio