World Quizzle – Relazione del progetto

Federico Matteoni – Mat. 530257

1 Introduzione

Funzionalità Il progetto richiedeva lo sviluppo di **WordQuizzle**: un sistema di sfide di traduzione italiano–inglese tra gli utenti registrati al servizio.
Era richiesta la possibilità, per gli utenti registrati, di poter sfidare gli utenti appartenenti alla propria rete sociale di amicizie. Le sfide consistevano nella traduzione, nel minor tempo possibile, di una serie di parole italiane proposte dal servizio.

Architettura Si richiedeva che l'applicazione fosse implementata secondo un'architettura client–server, con i componenti che comunicano `TCP`, `UDP` e tramite il meccanismo `RMI`. Inoltre si richiedeva, per il server, la persistenza delle informazioni degli utenti su un file `JSON`, e la richiesta delle traduzioni delle parole scelte ad un servizio esterno tramite richieste `HTTP GET`.
L'interazione con l'utente poteva avvenire tramite una `CLI` oppure, opzionalmente, una `GUI`. Per sperimentazione e sfida personale ho scelto la seconda.

2 Overview del sistema

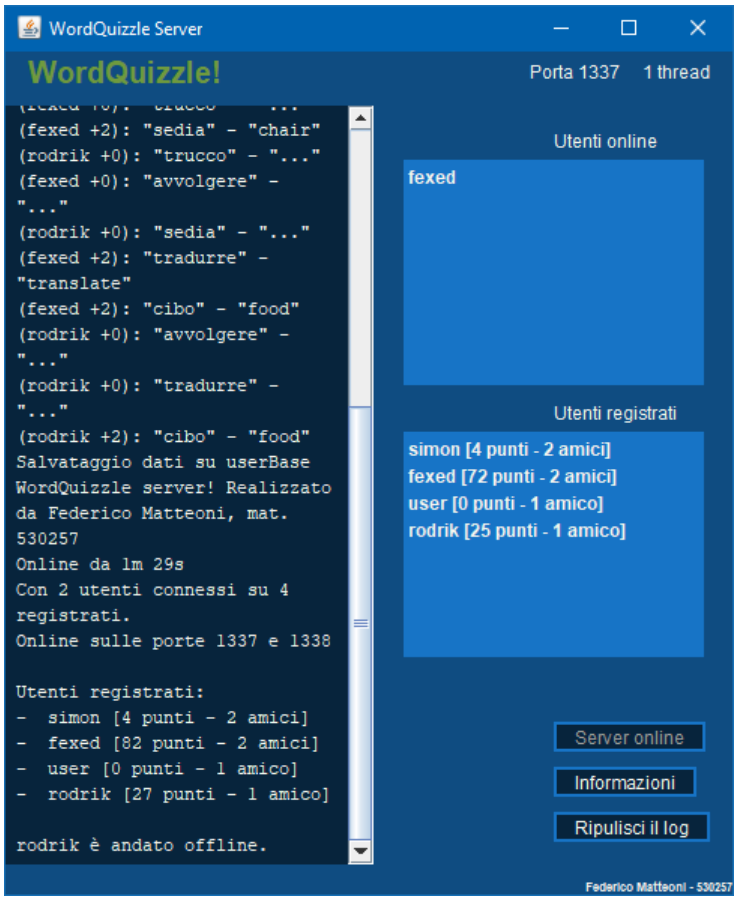
Il sistema è suddiviso in due parti principali: **client** e **server**. Tramite il client è possibile interagire col servizio WordQuizzle, che viene offerto dal server.
L'**interfaccia del server** offre varie **informazioni sullo stato attuale del servizio**, dagli utenti alla porta usata. Può essere usata per verificare il corretto funzionamento del sistema e monitorare le sfide in corso.
L'**interfaccia del client** è usata per l'**interazione vera e propria** col servizio: da essa si possono lanciare le sfide, richiedere le informazioni al server e consultare la propria lista amici e lo stato del proprio profilo utente.

2.1 Istruzioni per l'installazione e l'avvio

Server Il server, eseguito dall'archivio `WordQuizzleServer.jar`, **deve trovarsi in una directory contenente il dizionario** da cui selezionare le parole per la sfida. Il dizionario contiene `N` parole italiane, una per riga. Un dizionario di esempio con 100 parole è incluso.
Per avviare il server, è sufficiente eseguirlo con il comando `java -jar WordQuizzleServer.jar <porta>`, passando il numero di porta su cui stabilire le connessioni con il parametro `<porta>`.

Client Il client viene eseguito a partire dall'archivio `WordQuizzleClient.jar` tramite il comando
`java -jar WordQuizzleClient.jar <porta>`, specificando la porta del server a cui collegarsi tramite il parametro `<porta>`.

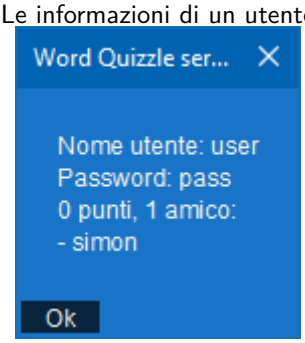
2.2 Server



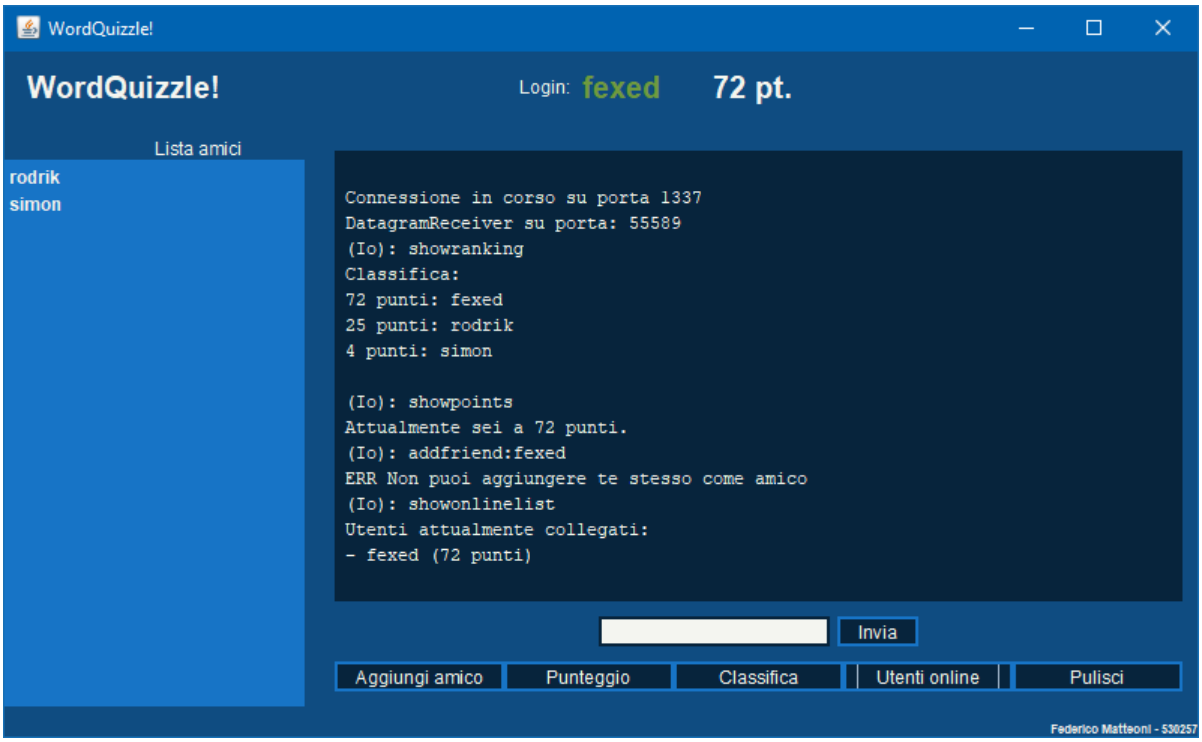
Interfaccia L'interfaccia del server è strutturata in modo da fornire a colpo d'occhio informazioni utili quali: utenti online e registrati, eventi recenti del servizio come sfide, connessioni e disconnessioni.
L'interfaccia cerca di essere il più chiara e intuitiva possibile: sulla sinistra si trova il log del servizio, mentre sulla destra sono elencati gli utenti online, quelli registrati e i comandi per avviare il server, stampare delle informazioni utili sul log o ripulirlo. In alto sono indicati il numero di porta su cui il server ascolta le connessioni in arrivo e il numero di thread attualmente eseguiti dal server.
La scritta "WordQuizzle" in alto a sinistra viene colorata di rosso o di verde a seconda dello stato in cui si trova il server, rispettivamente offline od online.

Funzionalità Il server si avvia automaticamente al lancio dell'eseguibile, ascoltando sul numero di porta specificato. Può essere chiuso come una qualsiasi applicazione dalla `X` in alto a destra sulla barra del titolo della finestra. In caso di errori ed eccezioni, il server stamperà i messaggi di errore sullo standard output (solitamente, la console da dove è stato lanciato) per poter verificare il messaggio dell'errore che si è verificato.
Quanto al servizio WordQuizzle, è possibile fare doppio click su un utente registrato per poter verificare le sue informazioni quali nome utente, password e punteggio, oltre alle amicizie nel dettaglio.

Le informazioni di un utente



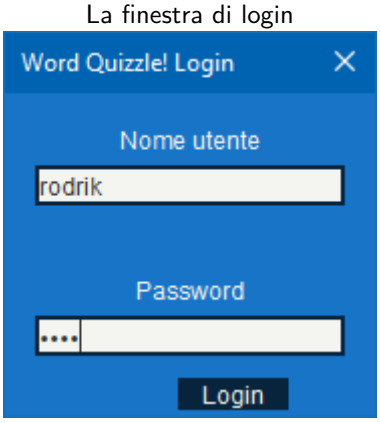
2.3 Client



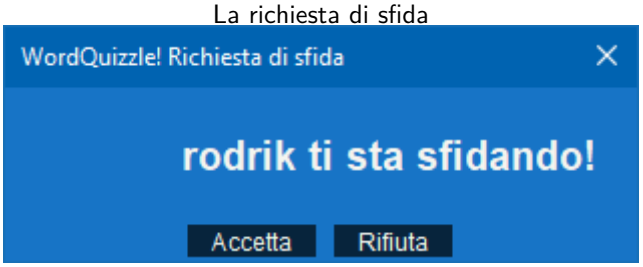
Interfaccia L'interfaccia dell'utente è pensata per essere facilmente fruibile in modo da ottenere le informazioni necessarie dal server o sfidare rapidamente un utente della propria lista amici. La finestra presenta al centro il log dei messaggi spediti e ricevuti dal server, mentre sulla sinistra c'è la lista amici dell'utente connesso. Le informazioni dell'utente sono visibili in alto, con punteggio e nome utente colorato di verde o rosso a seconda se il client è, rispettivamente, connesso o meno.
In basso si trova la barra di testo in cui inserire i comandi o, durante una sfida, le traduzioni proposte. Il pulsante "`Invia`" è selezionabile anche con il tasto `Invio` della tastiera, per permettere allo sfidante di rispondere il più velocemente possibile durante la sfida di traduzione. I pulsanti inferiori mandano dei messaggi che il server interpreta come richiesta di informazioni. Il pulsante "`Aggiungi amico`" apre una finestra di dialogo che richiede il nickname dell'utente da aggiungere.
La finestra di **login** si apre automaticamente all'avvio del client, mentre in caso di caduta della connessione e client offline apparirà un pulsante "`Login`" a destra del punteggio, per poter reinstaurare la connessione.

Funzionalità Per poter **sfidare un amico** è sufficiente fare doppio click sul nome utente dell'amico che si vuole sfidare. Il sistema gestirà la richiesta e la risposta: una finestra di dialogo apparirà sul client dell'amico che si vuole sfidare e attenderà una risposta per 10s. In caso di risposta affermativa, la sfida partirà per entrambi gli utenti.
L'aggiunta di un amico, la richiesta di punteggio, classifica e utenti online sono tutte funzionalità richiedibili tramite gli appositi pulsanti nella parte inferiore dell'interfaccia.
Eventuali errori che richiedono attenzione sono visualizzati come finestra di dialogo contenente il messaggio d'errore.

La finestra di login



La richiesta di sfida



3 Funzionalità

Dettaglio delle varie funzionalità del servizio che sono state richieste e delle scelte in sede di implementazione. Per i dettagli dei messaggi scambiati si veda la sezione apposita.

3.1 Registrazione

La fase di **registrazione** è stata implementata sfruttando la tecnologia **RMI** come richiesto.
Il **server** pubblica, sulla porta successiva a quella indicata come parametro, il riferimento alla propria interfaccia di registrazione con chiave "`WordQuizzle_530257`" per evitare possibili conflitti.
Il **client** esegue la procedura di registrazione in fase di login: se l'utente non esiste allora verrà registrato con il nome utente e la password appena inseriti. La fase di registrazione darà quindi errore in caso di nome utente già registrato oppure in caso di password vuota, oltre al caso in cui si verificano eventuali errori di comunicazione.

3.2 Login

La fase di **login** è eseguita quando l'utente ha inserito nome utente e password nella finestra di dialogo apposita e confermato l'inserimento. Il **client** spedisce via TCP un messaggio del tipo `login:nomeutente password`.
Il **server** verifica la correttezza dei dati e restituisce il risultato, con un messaggio del tipo `answer:esito`.

Errore In caso di errore, come password vuota o sbagliata o utente già collegato, il server ritorna un codice di errore apposito e il client mostrerà l'errore in una finestra di dialogo, per poi richiedere nuovamente i dati di login.

Successo In caso di login eseguito con successo, il server lancerà un **thread gestore** dedicato al client e alla sua connessione. Il server mantiene in memoria un riferimento al gestore legato al nome utente che gestisce: questo è utile per mantenere la lista degli utenti connessi e poter spedire le richieste di sfida.
Subito dopo il server spedisce al client un `JSON` con le informazioni dell'utente connesso: punteggio e lista amici. Questo per consentire al client di aggiornare la propria interfaccia di conseguenza.
Il client comunicherà al server la propria porta UDP per ricevere le richieste di sfida con un messaggio del tipo `challengePort:porta`.

3.3 Logout

Il logout viene eseguito alla chiusura della finestra del client. Il client chiuderà il socket di comunicazione, terminando il thread gestore associato sul server, il quale si preoccuperà di aggiornare il conteggio dei thread attivi e la lista degli utenti connessi.

3.4 Aggiunta di un amico

La fase di aggiunta di un utente alla propria lista amici si avvia cliccando sul relativo pulsante nell'interfaccia del client. Esso aprirà una finestra di dialogo richiedente il nome dell'utente da aggiungere.
In caso di conferma, il client spedirà al server un messaggio del tipo `addfriend:nomeutente` e attenderà il messaggio di risposta dal server, che può essere `anser:OKFREN` oppure `answer:ERR messaggio`.

Errore In caso di errore verrà semplicemente stampato il messaggio d'errore ricevuto sul log del client, consentendo all'utente di verificare cos'è andato storto. L'errore in questa fase può avvenire per aver tentato di aggiungere come amico un utente inesistente, già nella lista amici o sé stessi.

Successo In caso di successo, il server si occuperà di aggiungere gli utenti come amici, aggiornando la lista amici di entrambi. Il client dopodiché richiederà la lista amici al server così da mostrare all'utente l'effettiva aggiunta del nuovo amico e poter aggiornare l'interfaccia.

3.5 Richiesta della lista amici

Alla pressione del relativo pulsante sull'interfaccia del client, verrà spedito un messaggio del tipo `showfriendlist` al server. Esso risponderà con un messaggio del tipo `friendlist:JSON`, con il JSON rappresentante la struttura dati della lista amici (`WQUtente[]`). Per i dettagli delle classi si veda la sezione apposita.
Il client stamperà sul log la lista degli amici e aggiornerà l'interfaccia con la lista amici appena ricevuta.

3.6 Richiesta della lista degli utenti online

Analogamente, alla pressione del pulsante relativo il client invia `showonlinelist` al server, che risponderà con `onlinelist:JSON`. Il JSON rappresenta la lista degli utenti online (`ArrayList<WQUtente>`) che il client visualizzerà nel log.

3.7 Richiesta del punteggio utente

Il client spedisce `showpoints` al server, che risponde con `userpoints:punti`. Il punteggio ricevuto è visualizzato sul log e usato per aggiornare l'interfaccia del client.

3.8 Richiesta della classifica

La classifica è richiesta con un messaggio del tipo `showonlinelist`. Il server risponderà con la classifica come JSON (`ArrayList<WQUtente>`) in un messaggio del tipo `ranking:JSON`.

3.9 Sfida

La sfida è implementata, come richiesto, spedendo la richiesta (`challengeRequest:sfidante`) tramite UDP allo sfidante. La porta UDP su cui spedire la richiesta è comunicata dal client a seguito del login.
La sfida deve essere accettata entro (**T1**) 10 secondi. Se la sfida viene rifiutata (`challengeResponse:NO`, sia per timeout che per rifiuto esplicito), il server lo comunicherà al client che l'ha richiesta, il quale visualizzerà sul log il messaggio "`La sfida è stata rifiutata.`".

Sfida accettata Se la sfida viene accettata (`challengeResponse:OK`), il server comunica ad entrambi i client di prepararsi con un messaggio del tipo `challengeRound:1`.

Parole Il server, quindi, seleziona (**K**) 6 parole casuali dal dizionario, che contiene (**N**) 100 parole, e richiede le traduzioni al servizio esterno `mymemory.translated.net` tramite una serie di richieste `HTTP GET`. Per ogni parola italiana viene memorizzata la lista delle traduzioni fornite dal servizio. Le parole vengono ottenute eseguendo un parsing del `JSON` restituito, con tutte lettere minuscole e ripulendole da caratteri non alfabetici.

Sfida Non appena sono state ottenute tutte le traduzioni, il server lancia un **thread arbitro** che gestisce la sfida. Il thread si occupa di fornire ai thread gestori dei due utenti le parole da tradurre e le traduzioni, e attende che entrambi gli sfidanti abbiano finito la sfida.
Ogni thread gestore spedirà la parola da tradurre in un messaggio `challengeRound:parola` e attende un messaggio di risposta del tipo `challengeAnswer:parola`. Il client concede all'utente 5s per rispondere: la sfida può durare quindi al massimo K·5s cioè (**T2**) 30 secondi. Se il timer scade, il client risponde automaticamente con `challengeAnswer:-1`, altrimenti risponde con la parola inserita dall'utente.
Il thread gestore verifica le traduzioni, assegnando (**X**) 2 punti per traduzione corretta e togliendo (**Y**) 1 punto per ogni risposta sbagliata. In caso di risposta non data, non si assegnano punti.

Fine Non appena entrambi gli utenti hanno concluso la sfida, il thread arbitro si occupa di verificare il vincitore comparando i punteggi ottenuti. L'utente vincitore ottiene (**Z**) 5 punti bonus.
L'arbitro comunica il risultato agli sfidanti, mandando un messaggio del tipo `answer:challengeWin punti` al vincitore e `answer:challengeLose punti` al perdente, indicando anche il nuovo punteggio utente a ciascuno. In caso di pareggio, entrambi gli sfidanti riceveranno `anser:challenge punti`.
Dopodiché, vengono salvati i dati del server e la sfida è conclusa.

4 Classi

Classi comuni ai due componenti

WQGUI: include colori, tema e codice di inizializzazione dei componenti comuni alle `GUI`.

WQInterface: interfaccia per la registrazione tramite `RMI`.

WQUtente: dati del singolo utente, come nome utente, password, punteggio e lista amici.

Server

Logica

WQServer: codice principale del server, che implementa tutte le funzionalità richieste, tra cui la registrazione, login, sfida e strutture dati principali.

WQHandler: **thread gestore** della connessione con il singolo client, che si occupa di spedire e ricevere i messaggi sia su `TCP` che le richieste e risposte di sfida su `UDP`. Si occupa anche della parte centrale della sfida, cioè invio delle parole e ricezione delle risposte con calcolo del punteggio.

WQRefree: **thread arbitro** della sfida. Si occupa di lanciare la parte centrale della sfida e attendere la fine di entrambi gli sfidanti, dopodiché decreta il vincitore e assegna il punteggio definitivo.

Interfaccia

WQServerController: permette la comunicazione tra server e `GUI`.

WQServerGUIInterface: interfaccia della `GUI` del server, per poter aggiornare i vari componenti all'accadere dei vari eventi.

WQServerGUI: codice di creazione, comportamento e visualizzazione della `GUI` del server. Inoltre è la classe principale contenente il **main** del server.

Client

Logica

WQClient: codice principale del client, che implementa le funzionalità di login, registrazione e comunicazione principale del client.

WQClientReceiver: **thread** di ascolto del client per ricevere i messaggi `TCP` del socket provenienti dal server.

WQClientDatagramReceiver: **thread** di ascolto del client per ricevere i messaggi `UDP` provenienti dal server. Gestisce tutte le richieste di sfida ricevute, ma consente di rispondere ad esse solamente se non si è già in una sfida.

WQClientTimerTask: `TimerTask` del client per il tempo di risposta durante la sfida. Viene eseguito allo scadere del timer di 10 secondi, che parte alla ricezione di una parola da tradurre.

Interfaccia

WQClientController: permette la comunicazione tra client e `GUI`.

WQClientGUIInterface: interfaccia della `GUI` del client, per poter aggiornare i vari componenti all'accadere dei vari eventi.

WQClientGUI: codice di creazione, comportamento e visualizzazione della `GUI` del client. Inoltre è la classe principale contenente il **main** del client.