Word Quizzle

Relazione progetto Reti e Laboratorio 2019/2020

Matteo Stefanelli Matricola: 543781
Corso B

Overview

Il progetto consiste in una gara di traduzione dall'italiano all'inglese tra due utenti.

Gli utenti possono registrarsi, chiedere l'amicizia ad un altro giocatore e sfidarsi in una gara che prevede la traduzione di 10 parole in 2 minuti.

Il progetto è strutturato in tre packages principali: Server, Client e Commons, in quest'ultimo è possibile trovare l'interfaccia RMI utilizzata per la registrazione di un utente, una classe di constanti frequenti e comuni a più classi e infine un package di eccezioni.

Il server

Il server è strutturato in più flussi di esecuzione:

- Il main, presente nella classe Server, comicia l'esecuzione istanziando un oggetto database (unico per esecuzione) che ha il compito di memorizzare e gestire gli utenti. Il database recupera, se possibile, le informazioni da un file json, altrimenti crea il file e inizializza tutte le variabili. In seguito, viene creato un oggetto remoto che ha il compito di gestire la registrazione di un utente, questo avviene tramite RMI con l'esposizione dell'oggetto sulla porta 55010. Infine, si entra in ciclo infinito dove il server si mette in ascolto sulla porta TCP (STCPport) 60152, creando un nuovo thread ad ogni richiesta, il quale viene inserito in una ThreadPool (newChachedThreadPool) con coda non limitata.
- I thread nella threadpool gestiscono ognuno un client. L'intera classe UserThread (che ha proprio questo compito) è strutturata secondo uno schema di interpretazione di messaggi preimpostati. L'intero ciclo di vita di questo thread si basa su uno schema molto semplice: lettura dalla socket di un comando da parte del client e richiesta al database centrale dell'esecuzione dello stesso. Tutti i messaggi ricevuti da parte del client hanno lo stesso schema "KEYWORD <args>"; il server invece risponde con "codice_risposta, messaggio di risposta" (dove un codice maggiore o uguale a dieci vuol dire errore) oppure con una risposta ad una richiesta ricevuta (per esempio ad una richiesta da parte del client: LEADERBOARD <username>, il server risponde direttamente con la lista in formato json.
- Terzo ed ultimo flusso di esecuzione è quello avviato dal gestore del client quando un client richiede di giocare con un altro utente: se i due sono entrambi online, non sono occupati in altre sfide e sono amici, possono (se il secondo risponde con <ACCEPTED> entro dieci secondi) avviare una partita in un nuovo thread che gestirà entrambe le connessioni.

Il nuovo thread attraverso una select gestisce lettura e scrittura delle parole con entrambi i client.
Dopo aver selezionato le parole e richiesto la traduzione al servizio online https://api.mymemory.translated.net tramite GET, il thread entra in un ciclo in cui fino a quando entrambi gli utenti non hanno finito di tradurre le parole a loro inviate o fino a quando non scade il timer di 2 minuti, continua a leggere e scrivere parole e a verificare la loro traduzione sia esatta.
Il timer parte solo dopo che entrambi gli utenti sono collegati con la partita.

Il controllo della concorrenza è stato implicitamente gestito nel database attraverso l'uso di una Concurrent HashMap, che garantisce l'accesso a parti diverse della stessa struttura in maniera concorrente: la modifica ai vari clients avviene in maniera concorrente da parte dei vari thread che durante l'esecuzione ne modificano lo stato (e.g. il client finisce una partita e il thread che gestisce il match chiede al database l'aggiornamento dei suoi punti). Inoltre, l'unico metodo che scrive su file json per memorizzare i dati è dichiarato con la keyword synchronized in modo da rendere sequenziali le scritture su file.

Il sistema del punteggio consiste nell'assegnare +2 ad ogni risposta corretta, -1 ad ognuna errata e +3 al vincitore come bonus per aver vinto la partita.

Il client

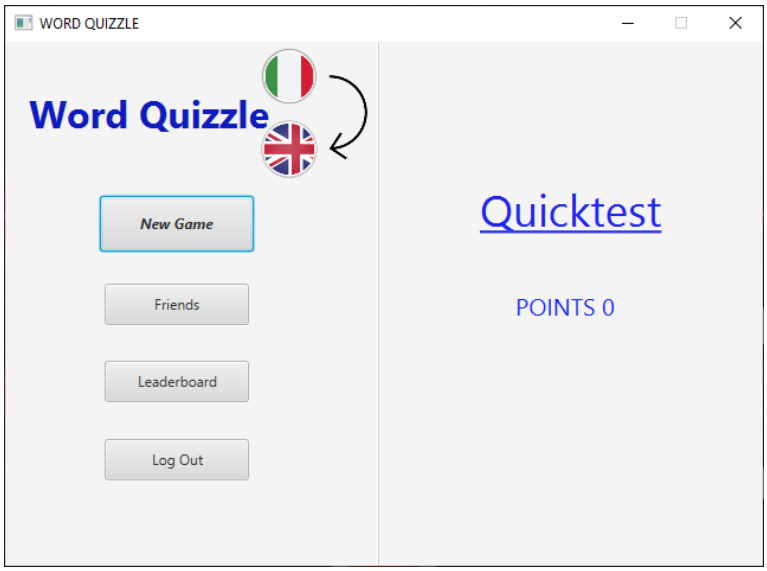
Il client ha un flusso principale di esecuzione, guidato dagli eventi provenienti dall'interfaccia grafica, e un thread che ha il compito di rimanere in ascolto su una porta UDP generata casualmente nell'intervallo [48152, 49152]. Il listener viene creato solo dopo aver effettuato il login con successo e ha il compito di notificare la ricezione di una richiesta di sfida da parte di un altro client amico.

Login page

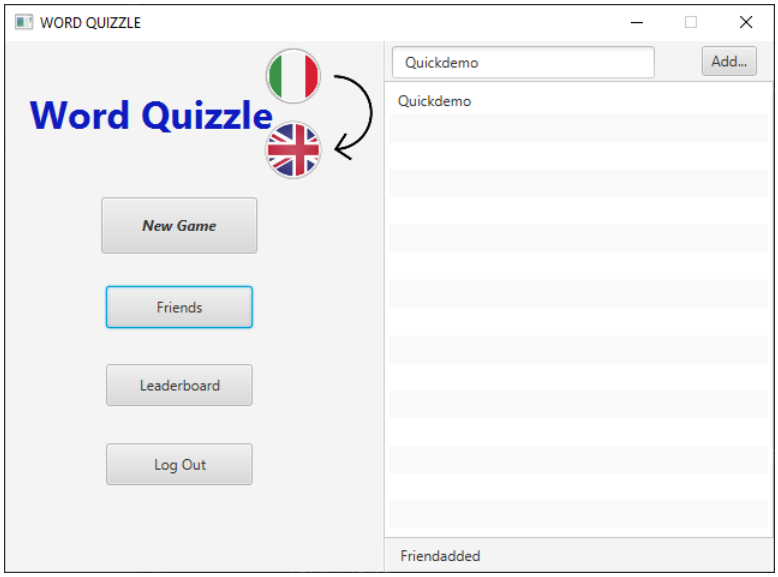
La schermata di Log In molto semplice e intuitiva è la prima schermata che viene lanciata dall'applicazione.

I dati inseriti nei rispettivi Field vengono presi e mandati al server senza controlli in quanto sarà quest'ultimo a occuparsi di eventuali errori e a mandare il relativo codice di errore, che verrà mostrato in rosso appena sotto il titolo. La fase di registrazione avviene attraverso l'invocazione di un metodo in remoto esposto dal server e raggiunto dal client attraverso una lookup. Eventuali errori gestiti dal server verranno mostrati nella relativa label.

Main page



La schermata principale si presenta divisa in due parti n cui gli eventi generati dalla pressione di uno dei bottoni a sinistra modifica la porzione a destra. Inizialmente viene mostrato l'username e il suo punteggio attuale.

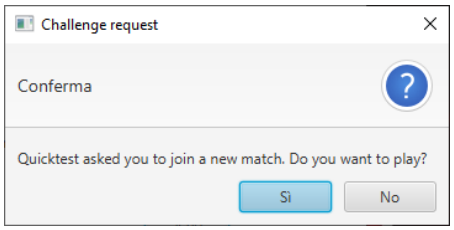


Dalla pressione del tasto Friends, otteniamo nella porzione superiore a sinistra un TextField per inserire l'username di un altro utente da aggiungere agli amici, il tasto Add per generare la richiesta al server, la lista amici, e in basso abbiamo la label che ci comunica l'esito delle operazioni ed eventuali errori. Questa label viene usata durante tutta la navigazione in questa schermata.

Il tasto Leaderboard mostra una schermata simile a quella per gli amici ma vengono nascosti le TextField e il Button per aggiungere un amico, in questo caso inoltre la lista mostra un ordinamento per punti.

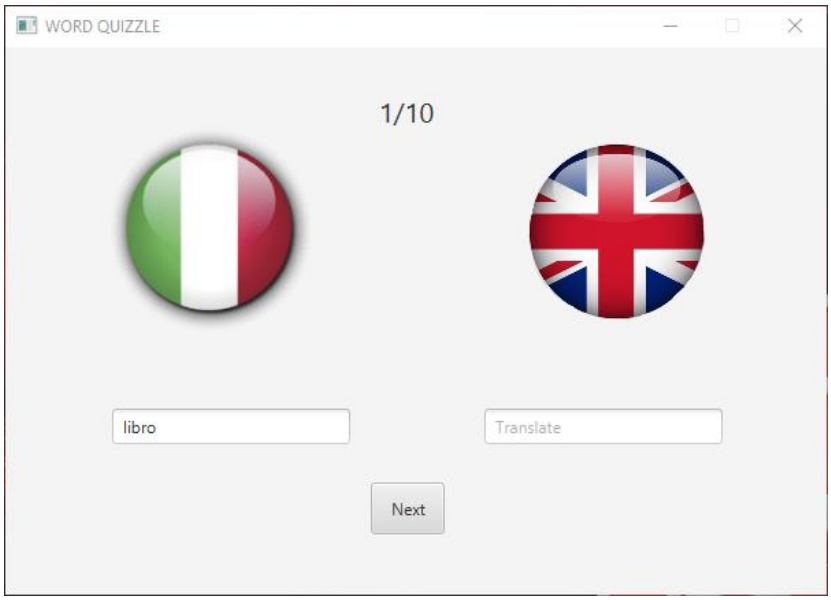
Il tasto Log Out tenta di effettuare una richiesta di LogOut al server e torna alla schermata di Log In, evento che avviene di default alla chiusura della finestra.

Con un click sul tasto New Game, la schermata mostra una selezione tra gli amici (in questo modo evito implicitamente di fare la richiesta di gioco a utenti che non sono nella mia lista amici) e il tasto Send inoltra la richiesta al server il quale provvederà ad inoltrarla, tramite porta UDP, al client sfidato. Se l'utente non è online o si verifica qualsiasi altro errore viene utilizzata la label in basso della figura precedente per restituire il messaggio di errore.



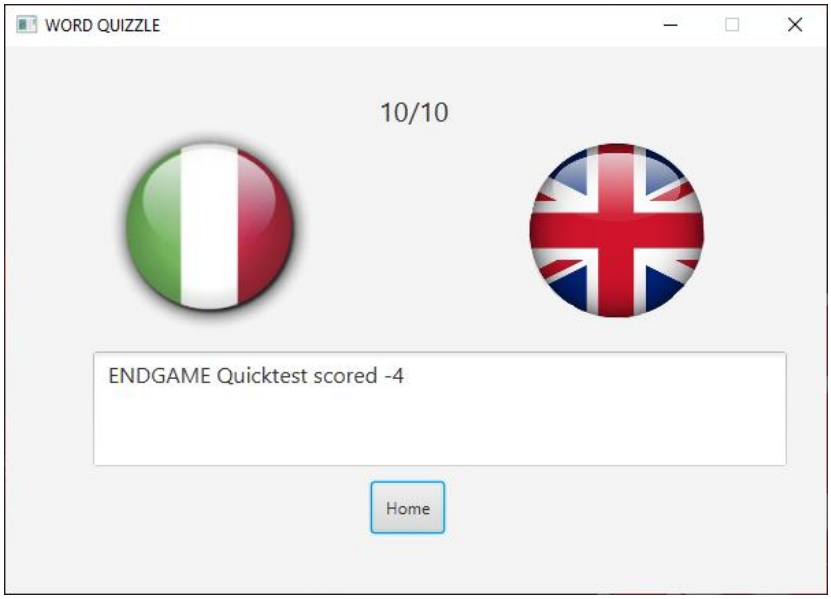
Allo sfidato appare una finestra di Alert di richiesta di sfida, questo ha un tempo limite di 3 secondi per poter rispondere, prima che l'AlertBox si chiuda da solo inoltrando al server che la richiesta è stata declinata.

Game page



La schermata di gioco si presenta molto semplice, questo a seguito di un breve periodo di caricamento nel quale la finestra sembra non rispondere, ciò è causato dalla sospensione del client sulla richiesta della prima parola presente nella Inizialize.

Il client non gestisce nessun timer per quanto riguarda la sfida, in quanto è già gestito dal server che allo scadere del tempo invia un messaggio di ENDGAME che viene riconosciuto dal client.



Al termine della sfida non viene mostrata una vera e propria schermata di vittoria o di sconfitta ma vengono mostrati solo i propri punti. Per come è stato implementato il server (analizza i punteggi solo dopo che entrambi hanno terminato o dopo che il tempo è scaduto), per mandare i risultati del match avrei dovuto tenere il client in attesa sulla read dopo che fossero state mandate tutte le parole in attesa della fine del match, ma questo causava un freeze dell'interfaccia grafica, pertanto ho preferito mostrare soltanto il punteggio del giocatore.

Istruzioni per la compilazione

Il progetto è stato interamente svolto e testato usando l'editor IntelliJ, con sdk 11.

Per compilare il software da riga di comando importare le librerie esterne presenti nella cartella jars e importare anche il package Commons e Commons/Exception utillizando il comando

javac -cp "./PATH/jars/*" *.java java -cp "./PATH/jars/*" Server (Client)

Nota: seguendo il path della cartella out si trovano già i file .class