Word Quizzle

Progetto di Laboratorio di Reti

a.a 2019/20

Ferraro Domenico
Matricola 559813
Corso B

Sommario

1 Introduzione

Word Quizzle è un sistema di sfide di traduzione italiano-inglese. È costituito da un server che ne eroga le funzionalità e da un client il quale, attraverso comunicazioni di rete con il server, usufruisce delle funzionalità da esso erogate. Al fine di garantire ottimizzazione delle risorse gli applicativi coinvolti fanno uso di più thread e del multiplexing dei canali. I protocolli coinvolti nelle comunicazioni di rete sono TCP, UDP e RMI. Viene anche utilizzato un protocollo custom per lo scambio dei messaggi. Inoltre, eventuali contenuti sensibili di un messaggio (ad esempio una password) vengono criptati utilizzando l'algoritmo SHA-256, garantendo la sicurezza delle informazioni.

2 Architettura

Word Quizzle è un sistema realizzato secondo una architettura client-server ed è costituito da: applicativo server, applicativo client e da un livello intermedio che implementa lo scambio dei messaggi.

L'applicativo server implementa tutte le funzionalità di Word Quizzle e le eroga ai clients che ne fanno richiesta. Garantisce, inoltre, la persistenza dei dati su file JSON quali gli utenti registrati ed i loro punteggi e amici (classe Persistence). L'applicativo client, invece, è un utilizzatore del sistema ed è costituito da due componenti principali: la prima si occupa di creare una interfaccia con il server, mentre la seconda utilizza la prima componente per realizzare una interfaccia con l'utente. Questo permette di separare in maniera netta il lato backend del client dal lato frontend. Il livello intermedio garantisce una comunicazione veloce ed efficiente via TCP, UDP e RMI ed astrae la complessità della ricezione e dell'invio dei messaggi. Svolge, quindi, un ruolo primario.

In WordQuizzle, poiché il multiplexing dei canali viene fatto per molteplici funzionalità, esso è realizzato mediante la classe astratta Multiplexer. Essa offre il ciclo principale per lo svolgimento del multiplexing di un canale specificato e fornisce metodi astratti invocati quando è possibile eseguire operazioni non bloccanti sul canale.

3 Server

Lo starting point del programma è caratterizzato dal metodo *main* contenuto nella classe MainClassWQServer. Durante la fase di avvio, vengono caricate le varie impostazioni dal file wordquizzle.properties contenuto nella directory delle risorse. Questo file contiene tutti i parametri fondamentali di Word Quizzle: il nome del file in cui sono contenute tutte le parole italiane, il tempo massimo concesso ad un utente per rispondere ad una richiesta di sfida, la durata di una partita, il numero di parole da tradurre, punteggi da assegnare in caso di parola tradotta correttamente o meno e punteggio extra in caso di vittoria. Tutti i tempi indicati sono espressi in millisecondi. La classe Settings è adibita a tale scopo e offre, per l'intera esecuzione del programma, metodi statici per l'ottenimento di ogni parametro.

Successivamente, vengono caricate le parole italiane dal dizionario (anch'esso contenuto nella directory delle risorse) e viene finalmente avviato il server istanziando un oggetto della classe WQServer. Questa classe svolge le mansioni di un handler, ovvero garantisce la gestione di tutte le richieste arrivate dai client. Il tutto viene realizzato implementando l'interfaccia WQHandler la quale contiene i metodi per la gestione di tutte le possibili richieste che possono arrivare dai clients.

Viene avviato il servizio di registrazione degli utenti tramite RMI (classe RMIServer), un thread adibito al multiplexing del canale UDP (classe UDPServer) ed il main thread procede con il multiplexing del canale TCP mediante un oggetto di tipo TCPServer. Le classi TCPServer e UDPServer, estendono la classe Multiplexer, ed implementando i metodi astratti realizzano le operazioni di lettura e scrittura sul canale TCP e UDP rispettivamente.

La scelta di introdurre il multiplexing dei canali anziché l'esecuzione di un thread per ogni utente connesso al server è motivata dal fatto che il numero di utenti connessi può essere elevato e questo porterebbe ad una perdita di performance perché lo scheduling di così tanti thread potrebbe creare molto overhead. Inoltre, il numero di clienti connessi cresce o diminuisce in maniera non deterministica e renderebbe complessa la gestione dell'avvio e della terminazione dei thread. Il multiplexing dei canali, invece, garantisce una gestione ottimale delle risorse ed un basso overhead.

3.1 Gestione dei dati

Tutte le funzionalità di WordQuizzle prevedono la manipolazione di dati relativi agli utenti. La struttura dati utilizzata per salvare in memoria gli utenti e tutte le loro informazioni è una HashMap che mappa l'username di un utente (che è univoca) ai dati dell'utente stesso (classe UserData). L'intera implementazione è definita nella classe UsersManagement. L'accesso quindi ai dati relativi ad un utente ha complessità O(1) e l'utilizzo di questa struttura dati si rivela una soluzione efficiente durante l'esecuzione del server. In caso di problemi a tempo di runtime viene sollevata una eccezione di tipo UsersManagementException e il messaggio di errore viene inviato al client. In questo modo, è possibile gestire casi limite come la richiesta di registrazione con una username già utilizzata oppure la sfida verso un amico offline e così via.

Le informazioni relative ad un utente sono: username, password criptata, se è online o meno, punteggio e lista degli amici. La classifica viene generata solo quando richiesta e non è ordinata ma sarà compito del client farlo. Questa soluzione è preferibile in quanto l'ordinamento di tutte le possibili classifiche potrebbe essere per il server una operazione molto lenta. Inoltre, non è veramente necessario fornire al client una classifica ordinata in quanto non è certo che il client voglia mostrare la classifica ordinata per punteggio ma potrebbe, ad esempio, mostrarla sottoforma di tabella e ordinarla per colonna in base alle necessità dell'utente.

Il server rende persistenti sul file *serverdata.json* tutti gli utenti e per ogni utente il punteggio, la password (criptata) e la lista degli amici. Di seguito un esempio di come i dati vengono salvati sul file:

```
{
"Eleonora":
    {
        "score":38,
        "password":"064b499e98e27914291dad6bbd764c227034021dde34f440",
        "friends":["Domenico"]
    },
"Domenico":
    {
        "score":12,
        "password":"6eddcee93727cd26fe266e0a78fa0b47084305c959c4632d",
        "friends":["Eleonora"]
    }
}
```

Quando le funzionalità richieste al server portano alla modifica di una delle informazioni citate, il file json viene riscritto garantendo la persistenza dei dati in caso di un crash imprevisto del sistema.

3.2 TCPServer

Le funzionalità di WordQuizzle svolte mediante multiplexing del canale TCP sono tutte (compresa la fase di gioco di una sfida) tranne l'invio della richiesta di sfida (e arrivo dell'eventuale risposta) e la registrazione di un utente le quali vengono svolte, come da specifica, via UDP e via RMI rispettivamente.

Quando arriva una richiesta TCP, viene chiamato l'handler relativo alla richiesta, il quale la gestisce rapidamente se possibile e ritorna un'eventuale risposta da inviare al client. Se l'handler non può gestire rapidamente la richiesta, la prende in carico in parallelo, consentendo nel frattempo la gestione di altre

richieste. Il primo caso avviene ad esempio quando un utente richiede il suo punteggio poiché si tratta di una operazione con complessità O(1). Il secondo caso avviene nel caso di una richiesta di sfida.

Quando un utente vuole sfidare un suo amico, invia una richiesta di sfida via TCP. Il server chiama il relativo handler il quale, se la richiesta può essere inviata, la prende in carico (classe ChallengeRequest) con un thread esecutore. Al fine di ottimizzare il tutto, viene quindi utilizzato un cached thread pool di esecutori i quali si occuperanno delle richieste di sfida.

Il thread che si occupa della challenge request richiede all'UDPServer di inviare il messaggio di sfida ed attende fino a quando la richiesta è scaduta oppure fino a quando viene accettata/declinata. In caso di richiesta accettata, seleziona in maniera casuale K parole italiane (indicato nel file di impostazioni), esegue una chiamata HTTP GET al servizio di traduzione MyMemory per richiederne la traduzione. Prima di terminare, notifica l'handler il quale invierà ad entrambi gli utenti la prima parola della sfida e alcune informazioni ausiliarie.

Un oggetto di tipo Timer è uno speciale thread fornito dalle API java al quale è possibile richiedere l'esecuzione di operazioni nel futuro. Quando la sfida incomincia, il server richiede ad un oggetto di tipo Timer (unico per tutto il ciclo di vita del server) la chiamata di un determinato metodo quando saranno passati i millisecondi massimi per la durata di una sfida. Il metodo chiamato non fa altro che notificare al server che la sfida ha superato il tempo massimo. Viene richiesto al timer di cancellare la chiamata del metodo nel caso in cui entrambi gli utenti traducano tutte le parole prima della fine della partita. Questa soluzione garantisce l'utilizzo di un solo thread per notificare al server le sfide che hanno raggiunto la loro durata massima.

3.3 Traduzioni

La richiesta di traduzione delle K parole potrebbe risultare estremamente lenta soprattutto se sono necessarie K richieste HTTP GET. Per ottenere una maggiore velocità viene fatta una sola richiesta inserendo tutte le parole e separandole con un punto ed uno spazio. Viene sfruttato il fatto che nella lingua italiana il punto prevede la fine di una frase e che quindi il servizio MyMemory tradurrà ogni parola in maniera indipendente. Un esempio di URL per la richiesta di traduzione è il seguente:

https://api.mymemory.translated.net/get?q=Bottiglia.%20Correre.%20Viaggio.%20Cane&langpair=it|en

Gran parte del JSON ritornato dal servizio di traduzioni non è necessaria. In realtà, le traduzioni si trovano all'inizio della stringa JSON ed il resto potrebbe essere tralasciato. La lettura dell'intero JSON non solo non è necessaria ma aumenta il tempo di elaborazione della risposta senza un valido motivo. Di conseguenza, viene soltanto letta la porzione del JSON di cui si ha interesse, ovvero quella contenente la traduzione delle parole e non l'intera risposta.

L'intera implementazione delle operazioni descritte in questo paragrafo è realizzata dalla classe Translations.

3.4 UDPServer

Le comunicazioni UDP sono totalmente destinate alle funzionalità relative alle sfide. In particolar modo una sfida viene fatta arrivare ad un utente utilizzando il protocollo UDP e l'utente può accettare o declinare oppure non rispondere in tempo alla sfida. Di conseguenza è chiaro che è necessario essere in grado di poter leggere e scrivere sul DatagramChannel quando necessario. La classe UDPServer, quindi, esegue il multiplexing del canale UDP. Rimane sempre interessato alla lettura per essere pronto quando un utente accetta o declina una sfida. In tale caso il thread gestore della richiesta di sfida viene notificato. Se invece l'utente non risponde in tempo, gli viene inviato un messaggio di timeout. Mentre questo meccanismo procede con un thread apposito, è anche possibile richiedere l'invio di una richiesta di sfida: il thread mantiene una coda concorrente per memorizzare le richieste di sfida che deve inviare e appena possibile sposta l'interesse sulla scrittura, procede con l'invio del messaggio e l'interesse ritorna nuovamente in lettura. Con questo meccanismo viene evitata

qualsiasi forma di attesa attiva e non è necessaria alcuna sincronizzazione delle operazioni sul canale perchè esse sono svolte da un solo thread.

3.5 Schema dei threads

Di seguito è riportata una tabella riassuntiva con lo schema dei threads coinvolti durante l'esecuzione del server.

Thread	Quantità	Mansione
Main thread (WQHandler e TCPServer)	1	Multiplexing del canale TCP, handling dei messaggi. Gestione delle funzionalità di WordQuizzle e persistenza dei dati.
RMIServer	1	Gestione della funzionalità di registrazione via RMI.
UDPServer	1	Multiplexing del canale UDP e forwarding delle richieste di sfida.
Timer	1	Notifica il server quando una sfida ha raggiunto il tempo massimo.
Challenge Request	Numero di richieste di sfida, quindi massimo la metà del numero di utenti connessi.	Richiede l'invio via UDP della sfida, attende la risposta o il timeout. In caso di sfida accettata richiede le traduzioni.

4 Client

L'applicativo client, invece, è un utilizzatore del sistema ed è costituito da due componenti principali: la prima si occupa di creare una interfaccia con il server che astrae tutte le comunicazioni di rete ed il parallelismo coinvolto, mentre la seconda utilizza la prima componente per realizzare una interfaccia con l'utente. La prima componente è realizzata dalla classe WQClient ed esegue le operazioni in maniera multithreaded ed effettuando il multiplexing dei canali mediante NIO. Per quanto riguarda la seconda componente, è possibile realizzare qualsiasi tipo di UI senza dover gestire l'intera componente client di WordQuizzle. Un esempio è la classe CLI la quale realizza una interfaccia a riga di comando. Questa divisione in componenti permette di separare in maniera netta il lato backend del client dal lato frontend. Inoltre, garantisce una manutenzione efficace ed una capacità di ottimizzazione del client molto elevata.

La classe MainClassWQClient rappresenta lo starting point del programma il quale istanzia un oggetto di tipo WQClient. Questa classe realizza tutte le funzionalità backend implementando l'interfaccia WQInterface la quale contiene tutti i metodi che devono essere implementati per la realizzazione del backend del sistema WordQuizzle. Il backend è costituito da un thread per il multiplexing del canale TCP (classe TCPMultipexer) e da un secondo thread per il multiplexing del canale UDP (UDPClient). Inoltre, esso lancia il servizio RMI (classe RMIClient) il quale si occuperà di richiedere la registrazione di un utente.

Al fine di garantire una buona sicurezza, il backend invia i dati sensibili come ad esempio una password in maniera criptata utilizzando l'algoritmo SHA-256. In questo modo, il server non riceve il dato in chiaro e quest'ultimo non è presente in chiaro sul canale di comunicazione.

Ogni operazione che il client richiede al server ha anche un certo timeout (molto lungo, anche 20 secondi). La motivazione è che il server potrebbe per una qualsiasi ragione diventare offline, magari per un errore di sistema o più semplicemente per una mancanza di connessione di rete. Questa soluzione salva il backend dal rimanere in attesa all'infinito quando il server non risponde.

4.1 Multiplexing del canale TCP

Viene lanciato un thread apposito per il multiplexing del canale TCP (classe TCPMultiplexer) il quale rimane in attesa della ricezione di risposte dal server. Quando è necessario inviare una richiesta al server, questa viene messa in una coda concorrente e non appena è possibile scrivere sul canale, le richieste vengono inviate. Il backend istanzia un oggetto di tipo TCPClient il quale si pone in mezzo tra il backend e l'invio dei messaggi via TCP. Questa classe contiene tutte le funzionalità di WordQuizzle che devono essere richieste attraverso il protocollo TCP.

4.2 UDPClient

Questa classe si occupa di eseguire il multiplexing del canale UDP. Si occupa banalmente di notificare al backend l'arrivo di un messaggio dal server. In particolare, potrebbe arrivare una richiesta di sfida oppure il timeout della richiesta di sfida arrivata precedentemente. Quando l'utente accetta o declina la sfida, viene immediatamente inviata la risposta al server.

La necessità di eseguire multiplexing nasce dal fatto che deve essere possibile ricevere messaggi dal server quando l'utente sta decidendo se accettare o declinare una sfida. Infatti, il server invia il timeout via UDP quando non riceve entro un certo tempo limite la risposta ad una richiesta di sfida. Per questo motivo, quando arriva una richiesta di sfida, viene notificato l'utente mentre viene spostato l'interesse sulla lettura. Se l'utente risponde allora viene spostato l'interesse sulla scrittura e viene inviata la risposta altrimenti viene ricevuto il messaggio di timeout dal server. Nel remoto caso in cui l'utente risponde alla sfida nel momento in cui viene ricevuto il timeout, quest'ultimo ha la precedenza e la risposta alla sfida non viene inviata.

4.3 Una implementazione del frontend: Command Line Interface

Il lato frontend può essere implementato semplicemente avendo un riferimento ad un oggetto che implementa la classe WQInterface come la classe CLI dimostra. Questa soluzione pone una netta separazione e permette di concentrarsi sulla realizzazione del frontend ignorando l'implementazione del backend e viceversa. La soluzione proposta è una interfaccia a riga di comando. Esegue in loop la lettura dell'input da tastiera dell'utente e lo traduce in un comando oppure nella risposta ad una richiesta di sfida o all'invio di una traduzione. Stabilita la natura dell'input dell'utente, chiama il metodo adeguato dell'interfaccia WQInterface.

L'interfaccia a riga di comando proposta, stampa sul terminale i messaggi di risposta del server i quali possono indicare situazioni sia di successo che di fallimento.

Una richiesta di sfida può arrivare in qualsiasi momento quando un utente connesso non sta già partecipando ad un'altra sfida. Per conoscere l'arrivo di questi eventi, il frontend deve implementare l'interfaccia ChallengeListener. Questo si traduce in un modo con il quale il backend è in grado di notificare al frontend l'avvenimento dei seguenti eventi asincroni: richiesta di sfida da parte di un amico, timeout di una richiesta di sfida arrivata e comunicazione della fine di una sfida.

5 Livello intermedio

La comunicazione di rete avviene inviando messagi che seguono un protocollo ben preciso: l'intero messaggio viene preceduto da un intero che indica la lunghezza del messaggio e poi il messaggio stesso. In questo modo il ricevente è in grado di leggere inizialmente il valore intero (perché la dimensione in bytes di un intero è nota) e successivamente è in grado di leggere totalmente il messaggio senza alcun rischio di letture parziali o eccessive che bloccherebbero l'esecuzione. Il messaggio è costituito da un header, chiamato *cmd* il quale permette di interpretare il messaggio e di eseguire un corretto parsing dello stesso limitando la possibilità di errori. In Word Quizzle, un messaggio è rappresentato da un oggetto di tipo ConnectionData. Di seguito è mostrato un esempio di comunicazione tra il server e un client e che mostra come sono fatti i messaggi scambiati.

```
[TCP]: LOGIN_REQUEST Domenico 6eddcee93727cd26fe266e0a78fa0b47084305c959c4632d53436b29bad0697a 58512 <- /127.0.0.1:62452 (Domenico)
[TCP]: SUCCESS_RESPONSE -> /127.0.0.1:62452 (Domenico)
[TCP]: SCORE_REQUEST Domenico <- /127.0.0.1:62452 (Domenico)
[TCP]: SUCCESS_RESPONSE 24 -> /127.0.0.1:62452 (Domenico)
[TCP]: LEADERBOARD_REQUEST Domenico <- /127.0.0.1:62452 (Domenico)
[TCP]: SUCCESS_RESPONSE {"Eleonora":12,"Domenico":24,"Ele":52} -> /127.0.0.1:62452 (Domenico)
```

La classe Connection è una classe astratta che realizza l'invio e la ricezione dei messaggi mediante il protocollo precedentemente specificato. Sono presenti dei metodi astratti atti alla lettura/scrittura dei dati dal/sul canale di comunicazione. Non sono implementati in quanto diversi canali di comunicazione possiedono criteri di lettura e scrittura differenti. Per le rispettive comunicazioni di rete, vi sono le classi TCPConnection e UDPConnection le quali estendono la classe astratta Connection e ne implementano la scrittura e lettura sul canale di comunicazione.

6 Istruzioni

Una volta lanciato il client, esso dà il benvenuto ed attende i comandi dell'utente. In particolare, i primi comandi che possono essere eseguiti sono la registrazione di un utente, il login oppure l'uscita dal programma.

```
Benvenuto su Word Quizzle!
> login Domenico passdom
Login eseguito con successo
> mostra_punteggio
Il tuo punteggio è 24
> mostra_classifica
      Utente       Punteggio
#1      Ele            52
#2   Domenico          24
#3   Eleonora          12
> logout
Logout eseguito con successo
> esci
```

Vi è un comando per ogni funzionalità di WordQuizzle. In figura è mostrato un esempio di esecuzione del client nel quale l'input utente è stato contrassegnato in verde per distinguerlo da ciò che viene stampato a schermo. In questo esempio l'utente con username Domenico ha eseguito il login con la sua username e password, ha richiesto il suo punteggio e la classifica prima di richiedere il logout e la chiusura del client. Quando il terminale mostra il carattere ">" significa che è in attesa che l'utente scriva un comando e per confermarlo bisogna premere il tasto invio.

I comandi sono abbastanza intuitivi ma è possibile digitare "aiuto" o un comando non valido per ottenere la lista di comandi disponibili e ogni sintassi. Di seguito è riportata una tabella con i comandi validi, la sintassi e una breve descrizione.

Comando e sintassi	**Descrizione**
registra_utente <nickUtente> <password>	Registra un nuovo utente con il nick utente e la password indicati
login <nickUtente> <password>	Effettua il login con il nick utente e la password indicati
logout	Effettua il logout se l'utente ha eseguito il login
aggiungi_amico <nickAmico>	Crea relazione di amicizia con nickAmico
lista_amici	Mostra la lista dei propri amici
sfida <nickAmico>	Richiesta di una sfida a nickAmico
mostra_punteggio	Mostra il punteggio dell'utente
mostra_classifica	Mostra una classifica degli amici dell'utente (incluso l'utente stesso)
aiuto	Mostra la lista dei comandi
esci	Esce dal programma

Sfida tra l'utente Eleonora e l'utente Domenico

```
> sfida Eleonora
Sfida inviata a Eleonora. In attesa di risposta...
Eleonora ha accettato la sfida
Via alla sfida di traduzione!
Avete 30 secondi per tradurre correttamente 8 parole.
Challenge 1/8: servire
> to serve
Challenge 2/8: completamente
> completely
Challenge 3/8: sogno
> dreams
Challenge 4/8: domani
Sfida terminata
Hai tradotto correttamente 2 parole, ne hai sbagliate 1 e non risposto a 5.
Hai totalizzato 1 punti.
Il tuo avversario ha totalizzato 3 punti.
Peccato, hai perso!
>
```

Client utente Domenico

```
Arrivata una sfida da Domenico. Vuoi accettare la sfida?
> si
Via alla sfida di traduzione!
Avete 30 secondi per tradurre correttamente 8 parole.
Challenge 1/8: servire
> to serve
Challenge 2/8: completamente
> completely
Challenge 3/8: sogno
> dream
Challenge 4/8: domani
Sfida terminata
Hai tradotto correttamente 3 parole, ne hai sbagliate 0 e non risposto a 5.
Hai totalizzato 3 punti.
Il tuo avversario ha totalizzato 1 punti.
Congratulazioni hai vinto! Hai guadagnato 10 punti extra, per un totale di 13 punti!
>
```

Client utente Eleonora

Per sfidare un amico bisogna utilizzare il comando *sfida*. Nell'esempio mostrato l'utente Domenico ha inviato una sfida all'utente Eleonora. È possibile rifiutarla scrivendo "no" oppure soltanto "n". È possibile accettarla scrivendo "si" oppure soltanto "s". Quando una sfida viene inviata l'altro utente ha un certo periodo di tempo entro il quale può rispondere alla sfida. Nel caso in cui non venga data una risposta entrambi i client mostrano un messaggio che indica che il tempo è scaduto. Nell'esempio, invece, Eleonora accetta la sfida scrivendo "si" e la sfida ha inizio. Il client mostra la parola da tradurre ed attende che venga digitata la traduzione. Bisogna premere invio per confermare. Si hanno 30 secondi per tradurre correttamente 8 parole e una parola tradotta correttamente corrisponde ad un punto mentre una tradotta erroneamente sottrae un punto. Il vincitore ottiene un punteggio extra di 10 punti. Nell'esempio mostrato Eleonora vince la partita perché risponde a 3 parole su 8 e lo fa in maniera corretta ottenendo 3 punti. L'utente Domenico traduce correttamente due parole ma ne sbaglia una, totalizzando un solo punto e perdendo la sfida. Le traduzioni non inviate, come mostrato, non aggiungono o sottraggono alcun punto.

Anche il server ha una sua linea di comando che non prevede alcun input utente ma che mostra la sua esecuzione. I messaggi ricevuti e/o inviati sono scritti sul terminale e viene indicato oltre al messaggio anche il protocollo con il quale il messaggio è stato ricevuto oppure è stato inviato. Sul terminale del server è possibile visualizzare informazioni riguardanti la sua esecuzione, ma anche le parole selezionate per una sfida e la traduzione ottenuta mediante la richiesta HTTP GET al servizio di traduzioni.

```
Challenge arrived: Domenico wants to challenge Eleonora
[TCP]: CHALLENGE_REQUEST Domenico Eleonora <- /127.0.0.1:62615 (Domenico)
[TCP]: SUCCESS_RESPONSE -> /127.0.0.1:62615 (Domenico)
Forwarding challenge request from Domenico to Eleonora
[UDP]: CHALLENGE_REQUEST Domenico Eleonora -> /127.0.0.1:59655 (Eleonora)
[TCP]: SUCCESS_RESPONSE Eleonora ha accettato la sfida -> /127.0.0.1:62615 (Domenico)
Selected words: [(servire, to serve), (completamente, completely), (sogno, dream), (doman
[TCP]: CHALLENGE_START 30000;8;servire -> /127.0.0.1:62628 (Eleonora)
[TCP]: CHALLENGE_START 30000;8;servire -> /127.0.0.1:62615 (Domenico)
[TCP]: CHALLENGE_WORD to serve <- /127.0.0.1:62628 (Eleonora)
[TCP]: CHALLENGE_WORD completamente -> /127.0.0.1:62628 (Eleonora)
[TCP]: CHALLENGE_WORD to serve <- /127.0.0.1:62615 (Domenico)
[TCP]: CHALLENGE_WORD completamente -> /127.0.0.1:62615 (Domenico)
[TCP]: CHALLENGE_WORD completely <- /127.0.0.1:62615 (Domenico)
[TCP]: CHALLENGE_WORD sogno -> /127.0.0.1:62615 (Domenico)
[TCP]: CHALLENGE_WORD completely <- /127.0.0.1:62628 (Eleonora)
[TCP]: CHALLENGE_WORD sogno -> /127.0.0.1:62628 (Eleonora)
[TCP]: CHALLENGE_WORD dream <- /127.0.0.1:62628 (Eleonora)
[TCP]: CHALLENGE_WORD domani -> /127.0.0.1:62628 (Eleonora)
[TCP]: CHALLENGE_WORD dreams <- /127.0.0.1:62615 (Domenico)
[TCP]: CHALLENGE_WORD domani -> /127.0.0.1:62615 (Domenico)
Challenge ended (Domenico vs Eleonora)
[TCP]: CHALLENGE_END 3;0;5;3;1;10 -> /127.0.0.1:62628 (Eleonora)
[TCP]: CHALLENGE_END 2;1;5;1;3;0 -> /127.0.0.1:62615 (Domenico)
```

Questo screenshot si riferisce al server mentre gestiva la sfida mostrata in precedenza. Le frecce indicano a chi è stato inviato il messaggio: se verso sinistra allora il destinatario è il server e viene indicato il mittente altrimenti il destinatario è il client e ne viene indicato l'indirizzo, la porta e l'utente associato.

6.1 Compilazione ed esecuzione

Viene fornito lo script *compile.cmd* all'interno della cartella "scripts/windows" che può essere utilizzato per la compilazione del server e del client in ambiente Windows. Il bytecode generato viene inserito nella cartella bin/WordQuizzle. Lo script per la compilazione prevede anche la copia delle risorse del server.

Vengono forniti gli script *run_client.cmd* e *run_server.cmd* all'interno della cartella "scripts/windows" i quali permettono di lanciare rispettivamente il client ed il server in ambiente Windows. Per garantire il funzionamento desiderato è importante lanciare gli script forniti dalla cartella principale e non all'interno della cartella degli scripts. Se si vuole eseguire il client mostrando soltanto i suoi comandi, è possibile farlo lanciandolo con l'opzione "--help" oppure lanciando lo script *run_client_help.cmd* (su Windows).

Per l'ambiente Linux non è fornito alcuno script ma un file di istruzioni nella cartella "scripts/linux" che elenca i comandi per la compilazione dei due applicativi e i comandi per la loro esecuzione.

L'unica libreria utilizzata è JSON Simple 1.1.1 e viene fornito il relativo jar all'interno della cartella "lib". Gli scripts e le istruzioni fornite si occupano autonomamente dell'inclusione della libreria nella fase di compilazione ed esecuzione.

Nel caso in cui venga utilizzato un IDE come IntelliJIDEA oppure Eclipse essi si occuperanno autonomamente della compilazione del programma. Per l'esecuzione mediante i suddetti IDE è importante ricordare che la main class del server è com.domenico.server.MainClassWQServer mentre quella del client è com.domenico.server.MainClassWQClient. Gli IDE in fase di compilazione andranno anche a generare il bytecode delle classi realizzate per lo Unit Testing. È importante quindi aggiungere la libreria JUnit versione 4.5.2 dal repository Maven o più in generale indicare allo IDE dove poter trovare la libreria JUnit 4.5.2.