Università di Pisa
Reti di Calcolatori e Laboratorio
Relazione del progetto - Word Quizzle

Russo Stefano
Matricola: 544341

Indice

Introduzione

La richiesta del progetto era quella di implementare una rete di amici che potessero interagire tra loro aggiungendosi vicendevolmente alle rispettive liste di amicizie e sfidandosi nel **WordQuizzle**: una sfida a cui prendono parte due utenti, in cui lo scopo è quello di tradurre in un dato tempo (**tempoSfida**) un certo numero di parole (**numeroParole**) dalla lingua italiana a quella inglese.

Tecnicamente era richiesto di utilizzare un misto tra UDP, TCP e RMI. La prima per la comunicazione riguardante la richiesta di accettazione della sfida da parte del **Server** e l'accettazione della sfida da parte del **Client**, la seconda per il resto delle comunicazioni eccetto la registrazione dell'utente che utilizza invece RMI. Inoltre, era richiesto l'utilizzo di richieste http GET, per comunicare con un API che fornisce un servizio esterno di traduzione italiano-inglese, e anche che tutte le informazioni riguardante gli utenti registrati al servizio debbano essere rese persistenti su file JSON.

1 Compilazione ed Esecuzione del codice

Il progetto è stato sviluppato utilizzando l'IDE Eclipse versione JSE 1.8, appoggiandosi alla libreria esterna **GSON-2.6.2** che bisogna includere nel build path per compilare correttamente il codice. Su Eclipse: Tasto destro sul progetto / Build Path / Add External Archives... e scegliere il file GSON (fornito in allegato).

L'implementazione si divide in **Client** e **Server** di cui si forniscono gli eseguibili .JAR da poter eseguire rispettivamente tramite i comandi:

```
java -jar WQServer.jar
```
Da eseguire in una directory contenente il file **wordlist** che funge da dizionario da cui prelevare le parole utilizzate durante la sfida.

```
java -jar WQClient.jar -help
```
Per visualizzare i comandi messi a disposizione dal **Client**.

```
java -jar WQClient.jar
```
Per l'esecuzione del **Client** vero e proprio.

2 Server

Carica il dizionario **wordlist** presente nella directory in cui viene eseguito.
Controlla la presenza o meno di file JSON di backup, nella medesima directory, e nel caso in cui esistano provvede a caricarli per aggiornare la struttura dati utilizzata per la memorizzazione della lista di utenti registrati e relative informazioni riguardante punteggio e liste di amicizie.
Successivamente rende disponibile a RMI la struttura dati stessa, in modo da rendere disponibile il servizio di registrazione. Fatto ciò, si mette in ascolto di comunicazioni TCP attraverso l'utilizzo di una Select che gestisca semi-contemporaneamente le richieste in entrata di diversi **Client**.
Ad ogni scrittura nella struttura dati, vengono lanciati due Thread (**threadBackup0, threadBackup1**) che si occupano di effettuare rispettivamente due scritture di backup in modo tale che se una scrittura venga interrotta o corrotta, viene messo a disposizione un backup aggiuntivo.
Nel momento in cui arrivi una richiesta di sfida da parte di un certo utente nei confronti di un altro, appartenente alla sua lista di amicizie, il server lancia un Thread (**threadSfidante**) per la gestione della sfida e si rimette in ascolto attraverso la Select.

2.1 Metodi Server

Descrizione dei metodi più complessi.

- **accept**, **read e write:**
Svolgono le relative operazioni sui channel dei **Client** connessi al Server tramite la Select.

- login, **logout**, **aggiungi_amico**, **lista_amici**, **mostra_punteggio e mostra_classifica:**
Eseguono i comandi richiesti dal **Client**, aggiornando o restituendo informazioni contenuti nella struttura dati **GrafoAmici**.

- sfida:
Effettua prima dei controlli sui partecipanti e poi lancia un Thread (**threadSfidante**) che si occupa della sfida.

2.2 Classi utilizzate da Server

- Allegato:
Implementa un insieme di dati che contiene le informazioni riguardanti un determinato utente, utilizzato come valore per la struttura dati **ConcurrentHashMap<String, Allegato> nodi** contenuta nella classe **GrafoAmici**.

- Arco:
Implementa un arco che rappresenta una relazione di amicizia tra due utenti, utilizzato in **Set<Arco> archi** contenuto nella classe **Allegato** per tenere conto della rete di amicizie di un

determinato utente.

- ClassificaComparator:
Effettua un override del metodo **compare**, utilizzato per effettuare un sort dei punteggi in ordine decrescente, nel momento in cui viene chiamato il metodo **mostra_classifica.**

- HttpGET:
Implementa un metodo, chiamato da **threadSfidante**, che permette di inviare una richiesta http GET all'indirizzo https://mymemory.translated.net/doc/spec.php . per la traduzione di una lista di parole da italiano a inglese, per verificarne la correttezza durante la sfida in corso.

- Matches:
Definisce una variabile (**translation**) di tipo String per contenere la traduzione, ad una determinata parola, fornita dal servizio di traduzione esterno in risposta alla richiesta http GET. Utilizzato nella classe **Traduzioni.**

- Traduzioni:
Classe di supporto utilizzata per leggere la stringa JSON ricevuta come risposta alla richiesta http GET, contiene un array di oggetti di tipo **Matches**, definiti nell'omonima classe, per immagazzinare le traduzioni a tutte le parole necessarie alla sfida.

- TaskBackup:
Thread che gestisce il backup dei file persistenti su file JSON. Utilizzato in **Server** e **TaskSfida.**

- GrafoAmici:
Implementa la struttura dati principale per immagazzinare informazioni sui vari utenti collegati al servizio, attraverso l'utilizzo della variabile **ConcurrentHashMap<String, Allegato> nodi.** In questa classe sono definiti inoltre i metodi per interagire con la struttura dati stessa, necessari al **Server** per la gestione dei comandi resi disponibili al **Client**.

- GrafoAmiciRMIimpl:
Implementazione dell'interfaccia **GrafoAmiciRMI**. Contiene il metodo **registra_utente** utilizzato direttamente dal **Client** da remoto per registrarsi presso la struttura dati **GrafoAmici**.

- TaskSfida:
Thread che gestisce la sfida tra due utenti. Utilizzato in **Server**.

3 Client

La prima cosa che il **Client** richiede all'utente è quello di fornire un numero di porta UDP, questa verrà utilizzata per l'ascolto di eventuali richieste di sfida, da parte di altri utenti, sfruttando il protocollo UDP. Successivamente tramite RMI si collega al **Server** in modo da poter chiamare il metodo di registrazione da remoto. Il **Client** chiede all'utente se egli sia un nuovo fruitore del servizio o meno, in modo da poter effettuare la registrazione o direttamente il login. Si entra in questo modo in un loop, in cui l'utente può provare a registrarsi ed effettuare il login, da cui può uscire digitando il comando **exit**. Una volta effettuato il login, il **Client** lancia un Thread (**threadUdp**) che si mette in ascolto UDP di eventuali richieste di sfida, e dunque entra in un ciclo di ascolto di input da tastiera, che permette all'utente di digitare un comando da inviare al **Server**, comprendente anche il comando **help** che stampa a schermo i comandi disponibili.

3.1 Metodi Client

Descrizione dei metodi più complessi, che implementano i comandi messi a disposizione dell'utente.

- login:
Invia al **Server** una stringa contenente l'operazione richiesta, il proprio nome, la propria password e il numero di porta UDP scelto all'avvio del **Client**.
Fatto ciò, legge e stampa a schermo il messaggio di esito ricevuto.

- logout:
Invia al **Server** una stringa contenente l'operazione richiesta e il proprio nome.
Legge e stampa a schermo il messaggio di esito.

- aggiungi_amico:
Invia al **Server** una stringa contenente l'operazione richiesta, il proprio nome e il nome dell'amico che si desidera aggiungere alla propria lista di amicizie.
Legge e stampa a schermo il messaggio di esito.

- lista_amici:
Invia al **Server** una stringa contenente l'operazione richiesta e il proprio nome.
Legge il messaggio di esito ricevuto, che consiste in un messaggio di errore, oppure nella lista dei nomi degli utenti presenti nella propria lista di amicizie.

- sfida:
Invia al **Server** una stringa contenente l'operazione richiesta, il proprio nome e il nome dell'amico che si desidera sfidare. Controlla il messaggio di esito e distingue tre casi:

> **-Sfida accettata**
> Tiene traccia del tempo trascorso dall'inizio della sfida in modo da terminare alla scadenza del tempo prefissato. Entra in un loop in cui legge le parole ricevute da **Server**
> e permette all'utente di digitare da tastiera la traduzione, che viene inoltrata. Proseguendo così fino al termine del tempo disponibile o finché il **Server** non invia un messaggio che

notifica di aver finito il numero di parole totale da tradurre, mettendosi così in attesa della terminazione dell'avversario. Alla fine, stampa a schermo i risultati della sfida.

-Sfida rifiutata o scaduta
Stampa a schermo il messaggio di sfida rifiutata o scaduta.

-Sfida interrotta
Sfida terminata a causa di un errore, stampa a schermo il messaggio di errore.

- accettaSfida:
Nel momento in cui vi è una richiesta di sfida, e l'utente digita il comando "Accetto" o "Rifiuto", viene chiamato questo metodo che distingue due casi:

-Sfida accettata
Invia al **Server** un messaggio di accettazione, e si mette in attesa del responso.
Se la sfida inizia senza problemi, agisce allo stesso modo del metodo **sfida**, altrimenti stampa a schermo un messaggio di errore

-Sfida rifiutata
Invia al **Server** un messaggio di rifiuto.

- mostra_punteggio:
Invia al **Server** una stringa contenente l'operazione richiesta e il proprio nome.
Legge il messaggio di esito, che consiste in un messaggio di errore, oppure nel proprio punteggio totalizzato durante tutte le sfide.

- mostra_classifica:
Invia al **Server** una stringa contenente l'operazione richiesta e il proprio nome.
Legge il messaggio di esito, che consiste in un messaggio di errore, oppure nella classifica dei punteggi dei propri amici e di sé stesso, in ordine decrescente.

3.2 Classi utilizzate da Client

- TaskUdp:
Thread lanciato dal **Client** dopo l'avvenuto login. Resta in ascolto sulla porta UDP specificata all'avvio, in attesa di eventuali datagrammi UDP che rappresentano richieste di sfida da parte di un altro utente.

4 Classi utilizzate da Client e Server

- Variabili:
Contiene variabili condivise tra **Client** e **Server** ed alcune utilizzate solamente da quest'ultimo, riguardanti valori utilizzati per la gestione delle sfide e numeri di porta, valori che possono essere liberamente cambiati.

- GrafoAmiciRMI:
Interfaccia che definisce il metodo **registra_utente** messo a disposizione dal **Server** tramite RMI.

5 Focus sui Thread utilizzati

5.1 TaskSfida

Chiamato da:

- **Server,** quando arriva una richiesta di sfida da parte di un utente, lancia il Thread **threadSfidante**, passando tra i vari argomenti, un valore booleano (**sfidante=true**) che identifica lo sfidante.

- **threadSfidante**, che si occuperà, nel caso in cui la sfida venga accettata, della gestione esclusiva dello sfidante e di generare un altro Thread **threadSfidato**, passando, tra i vari argomenti, un valore booleano (**sfidante=false**) che identifica invece lo sfidato, in modo tale si occupi esclusivamente della gestione dello sfidato.

Al termine della sfida da parte dello sfidante, **threadSfidante** esegue una join per aspettare che anche **threadSfidato** finisca. Così, al termine dell'esecuzione di **threadSfidante** viene calcolato il vincitore della sfida e vengono assegnati i punteggi ai rispettivi utenti, direttamente nella struttura dati **GrafoAmici**, di cui ha un riferimento. Fatto ciò, vengono eseguiti i due Thread di backup (**backup0, backup1**) per rendere persistenti i cambiamenti appena effettuati.
Entrambi i Thread che gestiscono la sfida hanno anche un riferimento alla struttura **ConcurrentHashMap <SelectionKey, String> Connessi**, che, nel caso in cui un utente collegato alla sfida si disconnetta in modo brusco, viene aggiornato.
La richiesta di sfida generata dallo sfidante viene inviata, da **threadSfidante** al destinatario, mediante un datagram packet sfruttando il protocollo UDP. Il datagramma di risposta, quindi UDP, viene atteso per un periodo di tempo (**timeOut**), dopo il quale la richiesta viene considerata automaticamente rifiutata. Nel caso in cui, invece, la sfida venga accettata, **threadSfidante** prosegue, prima di generare **threadSfidato**, a scegliere numero di parole (**numeroParole**) randomiche tra quelle contenute nella struttura **List<String> dizionario**, inizializzata all'avvio di **Server** con il contenuto del dizionario esterno **wordlist**, generando altrettanti numeri randomici e inseriti nella struttura **Lista<Integer> index** comune ai due Thread gestori, e ne richiede la traduzione al servizio API esterno tramite richieste http GET, immagazzinando le traduzioni richieste nella struttura **List<String> traduzioni** anch'esso comune a entrambi i Thread.

5.1.1 Metodi TaskSfida

Descrizione dei metodi più complessi.

-sfidante:
Rappresenta ciò che deve eseguire **threadSfidante**, come gestore della sfida nei confronti dello sfidante. Inizialmente invia la richiesta UDP allo sfidato e in base all'esito di questa, eseguire i metodi: **rifiuto**, **timeout** oppure **random**, **leggiTraduzioni** e **accetto**.

- sfidato:
Rappresenta ciò che deve esegue **threadSfidante**, ovvero il metodo **accetto.**

- rifiuto:
In seguito al rifiuto della richiesta di sfida da parte dell'utente sfidato, invia una risposta di richiesta rifiutata allo sfidante.

- timeout:
In seguito alla terminazione dell'intervallo di tempo **timeOut**, invia una risposta di richiesta scaduta allo sfidante.

- accetto:
Metodo per la gestione della sfida in sé, utilizzato sia da **threadSfidante** che da **threadSfidato**. Vengono inviate al destinatario le parole da tradurre e quelle ricevute come risposta vengono confrontate alle traduzioni presenti nelle liste, associate alle chiavi identificate dalle parole stesse. Alla fine del tempo (**tempoSfida**) prefissato per la sfida, o nel caso in cui vengano tradotte tutte le parole entro questo intervallo di tempo, il metodo termina ritornando così a **threadSfidante** che calcolerà l'esito della partita.

- random:
Genera **numeroParole** numeri randomici e li inserisce in **Lista<Integer> index**.

- leggi Traduzioni:
Effettua una richiesta http GET al servizio di traduzione esterno, mediante il metodo **sendGET** fornito dalla classe **HttpGET**. Nel caso in cui il sito non sia raggiungibile per via della mancanza di connessione da parte del Thread, questo aspetterà finché la connessione non sia ristabilita per effettuare nuovamente la richiesta, fino a un massimo di **maxTry** tentativi, dopo i quali la sfida viene interrotta e ai partecipanti vengono avvisati con un messaggio di errore.
Le parole tradotte ottenute vengono inserite in **ConcurrentHashMap<String, List<String>> matches**, struttura che mappa per ogni parola italiana, una lista di parole tradotte dal servizio.

5.2 TaskBackup

Chiamato da:

- **Server**, quando viene modificata la struttura dati **GrafoAmici** in seguito ad un comando da parte del client.

- **TaskSfida**, nel momento in cui i punteggi totalizzati durante la partita vengono inseriti in **GrafoAmici**.

threadBackup, mediante l'utilizzo della libreria esterna **GSON-2.6.2**, converte **GrafoAmici** in una stringa JSON e scrive il suo contenuto in un file **backup0** o **backup1**.
Se il file non esiste, ne viene creato uno nuovo con quel nome.

5.3 TaskUdp

Chiamato da:

- **Client**, subito dopo aver instaurato il login al servizio offerto dal **Server**.

threadUdp si mette in ascolto UDP alla porta specificata all'avvio del **Client**, e quando riceve una richiesta di sfida, stampa a schermo la notifica di ricezione della stessa e attende che venga fornita una risposta tramite linea di comando. Se entro un intervallo di tempo **timeOut**, non viene data una risposta alla richiesta, **threadUdp** torna in ascolto di ulteriori richieste, dal momento che al termine di quel lasso di tempo la richiesta viene considerata automaticamente rifiutata da parte del **Server**.

6.0 Strutture dati

-GrafoAmici:
ConcurrentHashMap<String, Allegato> nodi, definita in **GrafoAmici.**
Associa ad ogni nome utente, un valore di tipo **Allegato**, che contiene informazioni come: password, lista amici e punteggio.
L'accesso a questa struttura dati avviene da parte di:

- **Client**, quando utilizza il comando di registrazione tramite RMI.

- **Server**, quando aggiorna le relazioni di amicizia tra utenti.

- I Thread che gestiscono le sfide, quando, prima di terminare, aggiornano i valori riguardanti i punteggi degli utenti che vi hanno partecipato.

- Connessi:
ConcurrentHashMap<SelectionKey, String> Connessi, definita in **Server**.
Associa ad ogni SelectionKey, un nome utente, tenendo traccia degli utenti che in quel momento sono in collegamento con il **Server**. Viene acceduta da:

- **Server,** quando un utente aggiorna il proprio stato di connessione, o quando i metodi controllano se un determinato comando necessita che l'utente sia connesso.

-I Thread che gestiscono le sfide, nel remoto caso in cui un utente effettui una disconnessione non regolare durante la sfida.

- matches:
HashMap<String, List<String>> matches, definita in **TaskSfida**.
Associa ad ogni parola scelta per la sfida, una lista di traduzioni fornite dal servizio di traduzione esterno, così che durante la sfida si possa controllare la traduzione fornita dal giocatore con tutte quelle possibili fornite per quella determinata parola. Non è necessario garantire un accesso concorrente in quanto la scrittura avviene una volta soltanto da parte di un unico Thread, e successivamente viene utilizzata solo per le letture.

- Dizionario:
List<String> dizionario, definita in **Server**.
Inizializzato all'avio del **Server**, il dizionario di parole viene passato ai vari Thread che gestiscono le sfide, in modo che possano accedere alla lista di parole fornite dal dizionario esterno.

- Traduzioni:
List<String> traduzioni, definita in **TaskSfida**.
Raccoglie le traduzioni richieste al servizio esterno di traduzioni, e viene acceduto da entrambi i Thread di gestione della sfida, per il controllo della correttezza delle traduzioni fornite dal giocatore.

- Index:
List<Integer> index, definita in **TaskSfida**.
Tiene traccia di **numeroParole** numeri, generati randomicamente, senza ripetizioni, di modo che entrambi i Thread che si occupano della gestione di una sfida, possano attingere alle stesse parole da proporre nel corso della sfida.

- Archi:
Set<Arco> archi, definito in **Allegato**.
Tiene traccia di tutte le relazioni di amicizia di un determinato utente.

7.0 Scelte implementative

- Select:
Al contrario di un ThreadPool, che utilizzando operazioni non bloccanti dovrebbe utilizzare dei loop non efficienti, nel momento in cui ci siano più channel da leggere, la Select, che viene eseguita in un unico Thread, permette più efficientemente la gestione di più **Client** garantendo migliori performance e scalabilità.

- ConcurrentHashMap:
Dato che il Server si basa su una Select in ascolto sul Thread main, ma che la struttura dati principale **GrafoAmici** viene acceduta anche dai Thread che gestiscono le sfide, nel momento in cui terminano di gestirle, per aggiornare i valori dei punteggi degli utenti, è stata presa la decisione di utilizzare una struttura dati concorrente quale la ConcurrentHashMap che permette accessi simultanei per quanto riguarda le letture, mentre accessi concorrenti per le scritture. Inoltre, la struttura viene suddivisa in sezioni che permettono accessi contemporanei anche in scrittura nel caso in cui si tratti di sezioni diverse.

- Gson:
La libreria esterna **GSON-2.6.2** permette un agevole conversione, mediante l'utilizzo di pochi semplici comandi, di un oggetto in una stringa JSON che lo rappresenta, e viceversa, nel caso in cui però, si sia a conoscenza del tipo di oggetto.