WORDLE: un gioco di parole 3.0

Niccolò Corridori mat. 618000

Laboratorio di Reti A.A. 2022/23

Indice

1 Panoramica componenti

Il progetto è composto da due package principali: Client e Server.

1.1 Package Client

Il package Client contiene:

- ClientMain.java: file contenente il main per l'esecuzione del client.
- ClientThread.java: estende la classe Thread per la partecipazione al gruppo sociale. Permette la ricezione dei messaggi condivisi sul gruppo in parallelo rispetto al client.
- clientmain.properties: file di configurazione contenente parametri utilizzati dal client.

1.2 Package Server

Il package Server contiene:

- ServerMain.java: file contenente il main per l'esecuzione del server.
- SessionRunnable.java: implementa l'interfaccia Runnable, si occupa della gestione di più connessioni contemporanee al server.
- TerminationHandler.java: estende la classe Thread e definisce un handler per la terminazione del server.
- Word.java: definisce una classe per la gestione della parola segreta.
- WordRunnable.java: implementa l'interfaccia Runnable, definisce la funzione per il cambio della parola segreta.
- User.java: definisce una classe per l'utente.
- LastGame.java: definisce una classe per l'ultima partita giocata da un utente.
- servermain.properties: file di configurazione contenente parametri utilizzati dal server.

1.2.1 Cartella File

La cartella File contiene:

- words.txt: file testuale contenente il dizionario di gioco, ovvero tutte le parole utilizzabili durante il gioco, sia come parole segrete sia come tentativi da parte degli utenti.
- players.json: file JSON contenente tutte le informazioni relative agli utenti registrati, come credenziali d'accesso e statistiche di gioco.

1.2.2 Cartella "lib"

La cartella "lib" contiene il file "gson-2.10.1.jar", una libreria per la serializzazione di oggetti.

2 Protocollo di comunicazione

I protocolli adottati per la comunicazione tra server e client differiscono in base all'esigenza, in particolare tra comunicazione di gioco e interazioni nel gruppo sociale.

2.1 Gioco

Per le comunicazioni relative al gioco è stato adottato il protocollo TCP, che consente, una volta stabilita la connessione tra server e client, di avere un Socket dedicato per letture e scritture, in modo che esse avvengano in modo sincrono e il trasferimento sia assicurato.
La comunicazione in sé si basa sull'utilizzo da parte del server di alcune parole chiave, che consentono al client di cambiare stato a seconda del caso in cui ci si trova e rispondere di conseguenza.
Es. la parola chiave *write* indica che è richiesto all'utente di inserire un input da tastiera.
Tutto ciò che non è una parola chiave, come le stringhe relative al menù, viene stampato dal client su terminale.

2.2 Gruppo sociale

Per l'interazione col gruppo sociale è invece stato adottato il protocollo UDP.
I client al momento della connessione, dopo aver effettuato il login, entrano a far parte di un gruppo Multicast per la condivisione e visualizzazione di informazioni riguardanti l'ultima partita giocata. I client sono receiver, mentre il server è un sender, e non ha quindi la necessità di partecipare al gruppo per inviare le informazioni.
Quando un utente richiede di condividere il proprio risultato lo fa comunicandolo al server tramite la connessione TCP definita precedentemente, sarà poi il Server ad inviare il pacchetto al gruppo sociale tramite UDP.

3 Thread attivati

In questa sezione sono elencati in modo schematico tutti i thread attivati da client e server. Una descrizione più accurata è presente nelle successive sezioni.
Client:

- ClientThread: ricezione messaggi gruppo sociale.

Server:

- SessionRunnable: avviato tramite ThreadPool. Si occupa della gestione di più connessioni contemporanee.
- TerminationHandler: handler di terminazione.
- WordRunnable: avviato tramite ThreadPool. Si occupa del cambio periodico della parola segreta.

4 Persistenza dei dati

Per garantire la persistenza dei dati è stato scelto di scrivere su file JSON solo in alcuni momenti dell'esecuzione e non ad ogni modifica effettuata su essi. Questo viene fatto per limitare frequenti scritture su file e allo stesso tempo evitare che esse vengano fatte troppo di rado. In particolare i dati vengono scritti nei seguenti momenti:

- a registrazione avvenuta;
- durante la disconnessione;
- in fase di uscita.

5 Client

In questa sezione sono descritte nello specifico le operazioni effettuate durante l'esecuzione del Client.

5.1 ClientMain

ClientMain consente di connettersi al server e scambiare stringhe con esso per giocare.
Come operazione preliminare importa dei parametri di configurazione dal file "clientmain.properties", i quali vengono usati successivamente per richiedere il collegamento col server. Una volta stabilita la connessione, ClientMain inizia a leggere continuamente ciò che il server scrive sul Socket e cambia stato in base alla stringa letta. Tutte le operazioni vengono effettuate lato server, il client si limita a fare da tramite tra i due. Gli stati possibili sono elencati di seguito.

- Scrittura: il server comunica di essere in attesa di input da utente. ClientMain richiede all'utente di inserire un input e lo inoltra al server.
- Scrittura con controllo: analogo alla scrittura ma con controlli effettuati sull'input.

- Utente loggato: il server notifica che l'utente ha effettuato l'accesso correttamente. ClientMain inizializza una ConcurrentLinkedQueue, *othersStats*, per il salvataggio delle informazioni del gruppo sociale e inizializza e avvia un thread, *receiver*, usando la classe ClientThread per la connessione al gruppo sociale.
- Richiesta informazioni: il server comunica che l'utente ha richiesto di visualizzare il contenuto condiviso sul gruppo sociale.
- Disconessione: il server comunica che l'utente ha richiesto di disconnettersi lasciando il client in esecuzione.
- Uscita: il server comunica che l'utente ha richiesto di uscire dal gioco terminando l'esecuzione.
- Default: stampa di ciò che viene letto dal Socket.

5.2 ClientThread

Il costruttore di ClientThread richiede indirizzo e porta del gruppo sociale e la struttura dati su cui ricevere i messaggi (*othersStats*).
Quando viene avviato da ClientMain, ClientThread si unisce al gruppo Multicast e attende continuamente la ricezione di messaggi. Essendo un thread, ClientMain riesce a continuare la propria esecuzione e allo stesso tempo stare in attesa di messaggi. Quando riceve un messaggio, esso viene aggiunto alla coda *othersStats* se non è già presente, in caso un utente abbia condiviso più volte lo stesso messaggio. Una volta aggiunto il messaggio si rimette in attesa.

6 Server

In questa sezione sono descritte nello specifico le operazioni effettuate durante l'esecuzione del server.

6.1 ServerMain

ServerMain consente di stabilire le connessioni coi client.
Quando viene avviato esegue le seguenti operazioni preliminari:

- salvataggio di tutte le parole di gioco dal file "words.txt" in un Set di stringhe words;
- inizializzazione di una ConcurrentHashMap *players*: se esiste un file "players.json" recupera tutte le informazioni riguardanti gli utenti registrati tramite deserializzazione, altrimenti lo crea e rimane vuota;
- inizializzazione dei parametri da "servermain.properties" in modo analogo a ClientMain;

- avvio di uno ScheduledThreadPool a singolo Thread, WordRunnable, che consente di aggiornare periodicamente la parola segreta, implementata dalla classe Word;
- registrazione un handler di terminazione implementato dalla classe TerminationHandler;
- inizializzazione di un NewCachedThreadPool *pool* per la gestione delle connessioni multi-client.

Infine ServerMain si apre in ascolto di connessioni e, non appena ne riceve una, passa il Socket generato, insieme ad altri parametri, al NewCachedThreadPool *pool*.

6.2 TerminationHandler

TerminationHandler permette al Server di terminare in modo corretto; in particolare si occupa di:

- serializzare le informazioni relative agli utenti contenute nella HashMap *players* nel file "players.json";
- chiudere il Socket di ricezione connessioni del server;
- terminare i ThreadPool attivati in ServerMain, *pool* e *scheduler*.

6.3 SessionRunnable

SessionRunnable definisce il metodo *run()* che viene eseguito dal Thread con cui si interfaccia direttamente ClientMain durante la comunicazione. Tutto l'output del gioco viene inviato da SessionRunnable a ClientMain. Così come quest'ultimo, SessionRunnable è diviso in due fasi: fase di accesso e fase di gioco.
Durante la fase di accesso l'utente può scegliere tra accesso, registrazione o uscita. Sia la registrazione che l'accesso prevedono dei controlli sulle credenziali inserite dall'utente. Per passare alla fase di gioco è necessario che l'utente abbia effettuato l'accesso correttamente.
Durante la fase di gioco, l'utente può effettuare una o più delle azioni che sono elencate in seguito.

- Giocare: l'utente prova ad indovinare la parola. Se la parola cambia durante la partita, essa viene considerata persa. Mentre l'utente gioca vengono salvate e aggiornate le informazioni relative alle statistiche dell'utente.
- Visualizzare le proprie statistiche.
- Condividere le proprie statistiche relative all'ultima partita giocata: vengono inviate al gruppo sociale mediante la funzione *sendPacket()*, che crea e invia il pacchetto.

- Visualizzare le statistiche condivise dagli altri.
- Disconnettersi: viene cambiato lo stato dell'utente e aggiorna file JSON.
- Uscire: disconnette l'utente, aggiorna file JSON e termina l'esecuzione.

6.4 Word e WordRunnable

La classe Word implementa la parola segreta. Tiene traccia della parola segreta attuale, della precedente e del numero totale di parole estratte. Ciò consente di trattare il caso in cui venga estratta consecutivamente la stessa parola.
La classe WordRunnable implementa il metodo *run()* per il cambio periodico della parola, invocato da ServerMain.

6.5 User e LastGame

La classe User implementa un utente, avente campi riguardanti le credenziali e le statistiche personali. Tra di essi è presente anche un oggetto di tipo LastGame, che contiene informazioni sull'ultima partita giocata. User definisce il metodo *lastGameToString()* che restituisce una stringa formattata riguardo all'ultima partita effettuata, per un'eventuale condivisione nel gruppo sociale.

7 Scelte implementative

In questa sezione sono descritte le motivazioni riguardo le scelte implementative.

- *othersStats*: salva i messaggi condivisi sul gruppo sociale. È stata scelta una ConcurrentLinkedQueue poiché si tratta di una struttura weakly consistent e, dato che ClientMain e ClientThread possono richiederne una lettura e una modifica in modo concorrente, essa garantisce di poter generare un Iterator che tolleri eventuali inconsistenze. Inoltre una coda è una struttura che per sua natura si adatta bene all'esigenza di salvare dati in modo consecutivo, senza tener traccia di un eventuale ordine e senza restrizioni sulle dimensioni della stessa.
- *words*: contiene tutte le parole di gioco. È stato scelto un Set di stringhe e non una struttura dati concorrente in quanto essa non è condivisa, deve essere solo letta da un singolo thread, che si occupa di scegliere una nuova parola casuale periodicamente.
- *players*: contiene tutte le informazioni relative agli utenti registrati, sia credenziali che informazioni di gioco. È stata scelta una ConcurrenHashMap con relazione username-dati, in quanto essa viene condivisa da tutti i thread generati da ServerMain.
  Per l'accesso concorrente da parte dei thread sono presenti delle funzioni con blocchi synchronized sulla struttura, in modo da garantire consistenza e prevenire eventuali race condition.

- *register()*: registrazione; controlla se l'username scelto è già presente tra gli utenti registrati e in caso negativo crea e inserisce un nuovo utente *User* nell'HashMap.
- *login()*: accesso; controlla se l'utente con l'username richiesto è già online, se non è registrato e se la password è valida e in caso affermativo cambia lo stato dell'utente (offline-online).
- *logout()*: disconnessione; operazione inversa di *login()*.

8 Istruzioni per l'esecuzione

Si può procedere tramite terminale usando alcune righe di comando o tramite il Makefile fornito.

8.1 Linee di comando

I seguenti comandi vanno eseguiti all'interno della cartella principale del gioco "Wordle".
Per compilare ed eseguire il server bisogna digitare rispettivamente:

```
javac -cp ".:Server/lib/gson-2.10.1.jar" Server/*.java
```

```
java -cp ".:Server/lib/gson-2.10.1.jar" Server/ServerMain
```

Per la compilazione e l'esecuzione del client:

```
javac -cp Client/*.java
```

```
java Client/ClientMain
```

Se si vogliono eseguire direttamente i file ".jar":

```
java -jar Server.jar
```

```
java -jar Client.jar
```

8.2 Makefile

Per compilare tutti i file:

```
make
```

Per compilare ed eseguire il Server:

```
make s
```

Per compilare ed eseguire il Client:

```
make c
```

Per rimuovere tutti i file creati dopo la compilazione di entrambi:

```
make clean
```