Relazione di laboratorio III
WORDLE

Raffaele Cadau
616669

Indice

1 Introduzione

Il gioco di Wordle è un gioco di parole in cui il giocatore deve indovinare una parola di 10 lettere, estratta casualmente, entro un massimo di 12 tentativi. Per implementare questo gioco, il codice è stato suddiviso in tre package principali: Client, Server e Common. Il package Client contiene il codice del programma client, che consente all'utente di interagire con il gioco attraverso un'interfaccia utente intuitiva. Il package Server, invece, contiene il codice del programma server, che gestisce l'elaborazione dei dati del gioco e la comunicazione con il client. Il package Common contiene tre oggetti che facilitano la comunicazione tra client e server, tra cui Player.java, che contiene le proprietà del giocatore:

- Il numero di partite vinte;
- Il numero di partite giocate;
- Il numero di partite streak:
  - Rappresenta il numero di partite consecutive vinte dall'utente.
- Il numero di streak massimo consecutivi;
- Distribution:
  - Una lista che, ad ogni parola indovinata con successo, associa il numero di tentativi utilizzati dall'utente per indovinare la parola.

Inoltre, WordleCodici.java contiene una lista di codici di feedback utilizzati dai metodi del server per mandare una risposta al client, mentre GameWord.java implementa la classe che rappresenta la parola da indovinare e il numero di tentativi utilizzati dall'utente per completare il gioco. L'organizzazione del codice in package distinti e l'utilizzo di oggetti comuni, per la comunicazione tra client e server, conferiscono al progetto una struttura solida e ben definita, in grado di gestire in modo efficiente l'interazione tra utente e gioco.

2 Client

Il programma client è l'applicativo che utilizzeranno i giocatori. Nel package Client si trovano tre file: ClientMain.java, ClientGui.java e TaskMulticast.java.
Il primo contiene il metodo main e la logica del client, il secondo costruisce l'interfaccia grafica del gioco e gestisce gli eventi del mouse e infine il terzo, che implementa l'interfaccia Runnable, è un thread che viene attivato a seguito della risposta affermativa rispetto al login.
Quest'ultimo thread aspetta i messaggi inviati in multicast dal server relativi agli share degli utenti e li salva in una LinkedBlockingDeque. Ho scelto questo tipo di struttura dati, ad accesso concorrente (quindi thread-safe), in quanto doveva essere accessibile da entrambi i thread del client.
Il metodo main di ClientMain legge un file di configurazione in formato json e chiama il costruttore dell'oggetto che gestisce l'interfaccia grafica. [Per maggiori dettagli sul file di configurazione si rimanda il lettore alla sezione del manuale utente].
Il thread "main" dopo che inizializza l'interfaccia grafica, chiama il costruttore del ClientMain che inizializza la connessione TCP per comunicare con il server. La connessione è realizzata usando l'oggetto Socket di java.net.
Il ClientMain implementa i metodi dell'interfaccia WordleClient, richiamati dall'oggetto ClientGui quando il giocatore fa il click del mouse sui rispettivi oggetti JButton, della libreria javax.swing. Le operazioni di interesse sono:

- Register: permette all'utente di registrarsi con un username e una password a sua scelta, a patto che l'username non sia già stato utilizzato da un altro utente.
- Login: permette all'utente di accedere al sistema specificando l'username e password.
- Logout: esce dal gioco e comunica al server di chiudere la connessione.
- Gioca: comunica al server la volontà dell'utente di iniziare una nuova partita o continuare una partita in corso.
- Invio: permette all'utente di effettuare un tentativo per la parola corrente se questo è autorizzato a giocare.
- Le mie statistiche: l'utente richiede al server le sue statistiche.
- Share: permette di condividere, al termine del gioco, i tentativi fatti rispetto alla parola corrente.
- Partite condivise: visualizza i messaggi relativi alle partite condivise.

Il client non fa controlli esaustivi, che sono rimandati al server.

3 Server

Il programma che implementa il server è il cuore del progetto e gestisce tutti i controlli necessari. Il package "Server" contiene due sottopackage: "Task" e "User". Nel primo sono presenti i codici dei thread che vengono eseguiti quando l'utente richiede determinate operazioni, mentre nel secondo è presente il codice per la gestione degli utenti.
All'avvio, il thread "main" legge il file di configurazione del server e crea un'istanza dell'oggetto WordleServer. Questo oggetto inizializza il ServerSocketChannel, due ConcurrentHashMap e un selettore (selector). È stato scelto di utilizzare un ThreadPoolExecutor con un massimo di 20 thread e 10 di core per limitare il numero di thread istanziati dal server. Questa scelta è motivata dal fatto che, in questa applicazione, il tempo di attesa dell'input da parte dei giocatori è maggiore del tempo di elaborazione della CPU. Per evitare che i thread che servono i giocatori rimangano principalmente in uno stato di attesa, viene utilizzato un selettore (selector) che monitora i canali delle connessioni e "sveglia" i thread quando ci sono eventi nelle connessioni monitorate. In questo modo, ogni thread può gestire più utenti, passando ad un altro utente appena ha completato l'operazione richiesta dal precedente, anziché aspettare la prossima operazione dello stesso utente.
Quando arriva una nuova connessione da parte di un client, questa viene accettata utilizzando il metodo "accept" del ServerSocketChannel e il canale viene configurato come non bloccante e registrato nel selettore (selector) specificando l'interesse per l'evento di lettura (read).
Quando si verifica un evento di lettura da parte di un SocketChannel, viene letto l'intero che simboleggia l'azione del utente. In base all'azione richiesta dall'utente, vengono effettuati i primi controlli, ad esempio verificando che lo username e la password non siano vuoti in caso di richiesta di login o registrazione. Se i campi sono vuoti, vengono inviati al client dei codici di errore. Inoltre, se l'utente desidera effettuare il login, viene verificato che lo stesso utente non sia già connesso con un altro client. A tal fine, viene utilizzata la ConcurrentHashMap degli utenti loggati. In caso di utente già connesso, viene inviato al client un codice di errore "NoLogin".
Il thread "main" gestisce le richieste a basso costo computazionale, come il logout, l'invio delle statistiche e la condivisione degli share. L'operazione di logout consiste nella rimozione dell'utente dalla ConcurrentHashMap principale e nell'inserimento nella ConcurrentHashMap temporanea (tempUser). Questa operazione viene eseguita anche se il programma client viene chiuso in modo anomalo.
Le operazioni di login, registrazione, gioco (play) e invio di una parola (send) sono delegate a un thread del ThreadPoolExecutor poiché richiedono letture/scritture su file o controlli più lenti.
Nel costruttore del WordleServer viene inizializzato il thread "WordleWord", che ha il compito di scegliere una parola casualmente e salvare le statistiche

degli utenti. Inoltre, nella classe WordleWord è definito il metodo "suggerimenti" che confronta la parola inviata dall'utente con la parola estratta casualmente. La stringa di suggerimento è composta dai seguenti caratteri: "+", "?", "X". Questi caratteri indicano, rispettivamente, che un carattere è nella posizione corretta, che il carattere è presente nella parola ma non nella posizione corretta e che il carattere non è presente nella parola. La stringa di suggerimento viene generata in tre fasi. Nella prima fase, vengono individuati i caratteri nella posizione corretta. Nella seconda fase, con due iterazioni, viene controllato per ogni carattere rimanente se è presente nella parola estratta casualmente. Infine, i caratteri rimanenti vengono sostituiti con "X" per indicarne l'assenza nella parola estratta.

3.1 Gestione degli utenti

Il sottopackage User implementa il codice relativo agli utenti del gioco, con due file: PlayUser.java e User.java. Nel file User.java sono implementati due metodi statici: login e signIn. Entrambi prendono in input un nome utente e una password, consentendo rispettivamente di effettuare il login o registrare un nuovo account. Poiché entrambi i metodi vengono eseguiti da thread del ThreadPoolExecutor e utilizzano una proprietà statica della classe User chiamata fileObject, è necessario garantire l'esecuzione in mutua esclusione. Per ottenere ciò, viene utilizzato un monitor sull'oggetto JsonObject, fileObject.
L'oggetto JsonObject legge e tiene in memoria tutto il file degli utenti, dato che nel file si trovano solo gli username e le relative password. Aprendo il file con un editor di testo vediamo che assomiglia ad una serializzazione di una Map. Questo perché per la gestione degli utenti, username e password, è stata sfruttata un'implementazione efficiente dell'interfaccia Map implementata da Json. Tale mappa è recuperabile tramite il metodo asMap di JsonObject, e il tipo effettivo è com.google.gson.internal.LinkedTreeMap.
Il metodo login restituisce un valore intero, in particolare restituisce "OKLogin" se il login è riuscito, "ErrorPassword" se la password è errata e "usernameNotRegistered" se il nome utente non è presente nella mappa (ovvero se nessun utente si è registrato con quel nome utente). Anche il metodo signIn restituisce un valore intero. Restituisce "EmptyPassword" se la password inserita è vuota e "usernameNotAvailable" se il nome utente scelto è già in uso. In caso di successo, il metodo aggiunge il nome utente e la relativa password alla mappa utilizzando il metodo addProperty e restituisce "OKRegister".
La classe PlayUser estende Player e rappresenta un utente dal punto di vista del server. Questa classe definisce metodi per gestire un utente, tra cui i più rilevanti sono "savePlayUser" e "newPlayUser". Questi metodi leggono e scrivono sul file delle statistiche degli utenti e devono essere eseguiti in mutua esclusione, in particolare le scritture. Per farlo, viene utilizzato un monitor sulla proprietà statica "file", che rappresenta il nome del file.

“SavePlayUser” scrive nel file per prime le statistiche degli utenti che sono attivi, cioè connessi, poi quelle degli utenti che hanno fatto il logout, e infine aggiunge le statistiche degli utenti che non hanno giocato all'ultima parola. In questo modo gli utenti più attivi sono in cima al file ovvero per loro il recupero delle statistiche dal file è più veloce. Il file viene scritto usando un oggetto JsonWriter, l'API streaming di Gson. Per minimizzare il numero delle scritture su disco è stato usato un bufferedWriter e il file è scritto al cambio della parola e non ogni volta che un utente fa un operazione.
Il metodo "newPlayUser" restituisce un oggetto PlayUser con le statistiche aggiornate. Recupera le statistiche dalla ConcurrentHashMaop o dal file. Quando un utente effettua il login per la prima volta, il metodo "newPlayUser" chiama il costruttore per inizializzare le statistiche a 0.
Il file delle statistiche viene gestito sia in lettura che in scrittura utilizzando le API di streaming di Gson. Questo approccio è adottato perché il file può diventare molto grande, a differenza del file contenente solo nomi utente e password. Il file delle statistiche contiene tutte le statistiche di tutti gli utenti, compresi gli array delle parole indovinate con successo che possono diventare molto grandi.
Il metodo "addTentativo" restituisce una stringa che rappresenta il suggerimento da inviare al client. In alcuni casi particolari, genera un'eccezione di tipo GenericException che deve essere gestita dal chiamante. L'eccezione può verificarsi se un utente non ha richiesto di giocare, se un utente ha già completato il gioco (sia con successo che senza successo) o se un utente termina il gioco. Se il gioco è completato con successo, le statistiche e l'array delle parole completate con successo vengono aggiornati. Mentre il metodo "newPartita" restituisce un valore intero che indica se è possibile iniziare una nuova partita. Restituisce "OK" se l'utente non ha ancora giocato la partita corrente, mentre restituisce un valore che indica che non può giocare nuovamente se ha già completato il gioco. Infine, il metodo "share" restituisce la lista dei tentativi (suggerimenti) della partita corrente se l'utente ha completato il gioco.

3.2 Thread

Il server utilizza un thread principale e schedula periodicamente il thread "WordleWord". Inoltre, è presente un ThreadPoolExecutor con un massimo di 20 thread. Il thread "WordleWord" fa parte di un ScheduledThreadPoolExecutor di dimensione 1 e viene schedulato ogni "time" secondi, dove "time" è un parametro del file di configurazione. I thread del ThreadPoolExecutor eseguono il codice dei file nel sotto package "Task". Di questi, 10 thread non fanno parte del core, quindi vengono deallocati se rimangono inattivi per 1 secondo.
Il sotto package "Task" contiene due file: TaskPlay.java e TaskRegister.java, entrambi implementano l'interfaccia Runnable. Il file TaskPlay gestisce le

richieste di gioco o di nuovi tentativi nel gioco corrente. Vengono invocati i metodi corrispondenti della classe PlayUser: "newPartita" per una nuova partita e "addTentativo" per un nuovo tentativo. Il risultato viene inoltrato al client. Nel caso di un nuovo tentativo, viene inviato un codice intero al client, che può rappresentare un errore, l'indicazione che la parola corrente è errata o che i tentativi sono esauriti (game over), oppure che la parola è corretta. Successivamente, viene inviata al client la stringa di suggerimento generata dal metodo "suggerimenti" della classe WordleWord.
La classe TaskRegister gestisce la fase di registrazione e accesso degli utenti. Durante la registrazione, se lo username e la password sono vuoti, vengono inviati codici di errore al client oppure il risultato del metodo "signIn" della classe User viene inoltrato. Nel caso di accesso (login), viene verificato che l'utente non sia già loggato. Se l'utente non è già loggato, vengono controllate le credenziali invocando il metodo "login" della classe User. Se le credenziali sono corrette, lo username viene aggiunto alla ConcurrentHashMap degli utenti che hanno effettuato il login, insieme alle relative statistiche.

4 Manuale Utente

4.1 compila e esegui

Per compilare e eseguire il progetto Wordle, segui le istruzioni riportate di seguito:

1. Apri il terminale e naviga fino alla cartella "Wordle" dove si trovano i file del progetto.

2. Compila il server eseguendo il seguente comando:

$ javac -cp library/gson-2.10.jar Server/*.java Server/Task/*.java Server/User/*.java common/*.java -d bin

3. Compila il client eseguendo il comando successivo:

$ javac -cp library/gson-2.10.jar Client/*.java common/*.java -d bin

4. Al termine dell'esecuzione dei comandi, verrà creata una nuova cartella chiamata "bin" nella directory "Wordle". All'interno di questa cartella troverai i file compilati con estensione ".class".

5. Ora puoi creare l'eseguibile jar del server eseguendo il seguente comando:

$ jar cmvf MANIFESTserver.MF server.jar library/gson-2.10.jar -C bin Server/ -C bin/ common/

6. Successivamente, crea l'eseguibile jar del client con il comando:

$ jar cmvf MANIFESTclient.MF client.jar library/gson-2.10.jar -C bin Client/ -C bin common/

7. A questo punto, hai generato due file JAR: "server.jar" e "client.jar". Per eseguire il server, utilizza il seguente comando:

$ java -jar server.jar [file di configurazione]

8. Per eseguire il client, in un altro terminale, utilizza il comando:

$ java -jar client.jar [file di configurazione]

4.2 File di configurazione

Quando si esegue il server e il client di Wordle, è possibile specificare un file di configurazione in formato JSON come parametro opzionale. Di default, il server utilizza il file di configurazione "configServer.json" e il client utilizza il file "configClient.json". Di seguito sono descritti i campi richiesti per entrambi i file di configurazione:
File di configurazione del server (configServer.json):

- ‘address‘: L'indirizzo IP su cui il server deve essere eseguito. Per funzionare in locale, si utilizza l'indirizzo "127.0.0.1".
- ‘port‘: La porta a cui il server si deve collegare per ascoltare le richieste dei client. Assicurati che sia un numero intero compreso tra 1 e 65535, e che sia una porta libera, non utilizzata da altri processi.
- ‘addressMulticast‘: L'indirizzo IP per il gruppo multicast. Questo indirizzo deve essere compreso tra "224.0.0.0" (escluso) e "239.255.255.255" (incluso).
- ‘portMulticast‘: La porta per il gruppo multicast. Assicurati che sia un numero intero compreso tra 1 e 65535 e che sia una porta libera.
- ‘fileLogin‘: Il nome del file in cui il server salva le credenziali degli utenti.
- ‘filePlayer‘: Il nome del file in cui il server salva le statistiche degli utenti.
- ‘dict‘: Il nome del file dizionario che contiene le parole valide per il gioco Wordle. Assicurati che il file contenga una parola per riga e che sia ordinato in modo alfabetico.
- ‘time‘: Il tempo di pianificazione del thread WordleWord espresso in secondi.

File di configurazione del client (configClient.json):

- ‘address‘: L'indirizzo IP del server a cui il client deve connettersi.
- ‘port‘: La porta a cui il server si collega per ascoltare le richieste dei client. Deve corrispondere al valore specificato nel file di configurazione del server.
- ‘addressMulticast‘: L'indirizzo IP per il gruppo multicast. Deve corrispondere al valore specificato nel file di configurazione del server.
- ‘portMulticast‘: La porta per il gruppo multicast. Deve corrispondere al valore specificato nel file di configurazione del server.