Wordle

Progetto d’esame di Laboratorio di Reti III - a.a 2022/2023

CdL in Informatica - Giacomo Torbidoni mat. xxxxxx

> Nel seguente documento vengono elencate le caratteristiche e le funzionalità del progetto Worlde che vuole imitare il funzionamento di base del gioco online del 2021 Wordle (attualmente raggiungibile tramite il seguente link: https://www.nytimes.com/games/wordle/index.html).
> Il progetto è composto da 6 classi java: Utente.java, ClientMain.java, GestoreFile.java, RandomSet.java, ServerMain.java, Task.java.
> Nel seguito verranno presentati i vari metodi di cui sono composte le classi e i funzionamenti di base. Nella sezione successiva viene fornita una breve guida per utilizzare il programma ed un'altra per la compilazione e l'esecuzione dello stesso.

Classi:

Utente.java

Questa classe rappresenta la struttura di come è stato pensato l'utente che sarà utilizzato per la gestione delle partite, avrà quindi un username ed una password che permetteranno l'autenticazione ed il riconoscimento dell'utente. Sono conservati all'interno dell'oggetto *utente* della classe *Utente* il numero di vittorie totali, il numero di partite disputate, la serie di vittorie consecutive attuali, la serie migliore di vittorie consecutive, un flag ***log_in*** che permette al programma in determinati punti di capire se l'utente è loggato o meno, ed il flag ***in_playing*** che, come dice il nome, permette di capire se l'utente è attualmente impegnato in una partita. Tra i metodi più importanti della classe risalta ***saveMatch()*** che permette di aggiornare i dati della partita attuale; come parametri accetterà quindi l'esito della partita e il numero di tentativi impiegati.

Task.java

La classe ***Task.java*** contiene i metodi che verranno eseguiti da ciascun Thread di ogni utente. La classe implementa l'interfaccia Runnable. I thread che andranno a generare i task verranno istanziati nella classe ServerMain.java. La

comunicazione tra il client e il task avviene tramite oggetto socket, dopo la ***accept()*** della classe ServerMain.java.Questa classe contiene i metodi più importanti del programma in quanto la gestione del gioco e dei dati è affidata al server. Tra i metodi di rilevanza troviamo:

- ***login(Scanner in, PrintWriter out)***

  Questo metodo permette ad un utente precedentemente registrato di eseguire il login nell'applicazione andando a specificare il proprio username e la password. Ha come paramenti un oggetto di tipo Scanner, utilizzato per ottenere in ingresso dati dal client e un oggetto PrintWriter che viene invece utilizzato per inoltrare via comunicazione TCP al server i dati. Come spiegano i commenti nel codice, la prima cosa che viene eseguita è un controllo per verificare che un utente non sia già connesso utilizzando il flag log_in di un oggetto di tipo Utente che rappresenta l'utente connesso attualmente con il task. In caso affermativo viene restituito l'errore — LE(LoginEffettuato), altrimenti viene controllato se l'utente con cui si cerca di accedere è connesso da un altro dispositivo cercandolo nella **ConcurrentHashMap** che mantiene il ServerMain tramite il metodo ***getUser(String username)*** (Discusso in seguito); se il controllo viene passato, viene controllata la password ed infine si assegna ad userInGame l'utente che ha effettuato il login.

- ***register(Scanner in,PrintWriter out)***

  Il seguente metodo è responsabile della creazione di un nuovo account. Il metodo ha un funzionamento di base che, inserito un username ed una password e una volta controllato che l'username sia univoco, quindi non già presente, e la password non sia vuota, va ad inserire all'interno della struttura dati del ServerMain utenti il nuovo utente tramite il metodo ***registerUser()***.

- ***logout(PrintWriter out)***

  Questo ha lo scopo di effettuare il *log_out*, quindi la disconnessione dell'utente attualmente connesso. Una volta verificato che ci sia un utente effettivamente connesso (in caso contrario viene inviato l'errore — LNE(LoginNonEffettuato)), viene aggiornata la streak tramite

updateStreak(); dopodiché, se il giocatore è ancora in partita, viene salvata la partita attuale come una sconfitta, impostati i vari flag in maniera corretta e terminato il ***run()*** del task andando a impostare a true la variabile globale ***stopServer.*** Nel caso invece in cui l'utente non era in partita si eseguiranno le stesse azioni senza però salvare la partita.

- ***printStats(PrintWriter out)***

  Questo metodo permette di stampare le statistiche dell'utente attualmente connesso (userInGame).

- ***updateStreak()***

  Permette l'aggiornamento delle streak dell'utente connesso.

- ***playWordle(PrintWriter out)***

  Tramite questo metodo il client comunica al server l'inizio di una nuova partita. Viene eseguito un controllo per verificare che il giocatore non abbia già giocato la parola generata. In caso di affermativo si restituisce l'errore —PG(ParolaGiocata). Se i controlli vengono passati si prosegue impostando a true la variabile userInGame.in_playing.

- ***sendWord(Scanner in, PrintWriter out)***

  Si gestisce cosi la ricezione da parte del task di una parola inviata dal client. Si esegue il controllo per verificare se il giocatore ha già giocato la parola, ed in caso affermativo si restituisce un errore in quanto vorrebbe dire che sta cercando di indovinare la parola quando in realtà il server ne ha generata una nuova (si necessita di richiamare playWordle()). Dopodiché viene incrementato il numero di tentativi e si controlla tramite il metodo descritto in seguito ***wordAnalyzer()*** l'esito della partita. Tramite una variabile globale ***win*** viene verificato che l'utente non abbia già vinto la partita ed in caso di vittoria viene inoltrato il messaggio —WIN e aggiornati i vari flag e strutture dati per la gestione delle partite e dell'utente. Se i tentativi sono > o = a 12 la partita viene considerata persa e, dopo aver salvato lo stato, si inoltra —LOSE. Se i tentativi non sono finiti e la parola non è stata trovata si inoltra al client una stringa contenente i suggerimenti.

dopo aver salvato lo stato, si inoltra —LOSE. Se i tentativi non sono finiti e la parola non è stata trovata si inoltra al client una stringa contenente i suggerimenti.

- ***wordAnalyzer(String parola_utente)***

  Rappresenta la logica che determina se una parola è stata indovinata o meno. Presa la stringa da parametro viene eseguito uno split() e inserita in un array di caratteri per confrontarli uno ad uno. La stessa cosa viene fatta con la parola attuale del server. Tramite for si contano i caratteri che sono nella posizione corretta o che sono presenti in entrambi gli array. In base a questo vengono generati i seguenti suggerimenti

  > **+ :** il carattere è presente nella parola da indovinare ed è nella posizione corretta

  > **? :** il carattere è presente nella parola da indovinare ma non è nella corretta posizione

  > **x:** il carattere non è presente nella parola da indovina

Se le lettere corrette sono 10 si considera come vittoria e si restituisce una stringa —WIN

- ***run()***

  Il metodo *run()* è il metodo che entrerà in esecuzione tramite il thread. Vengono creati all'interno di quest'ultimo tramite inputStream e outputStream gli oggetti PrintWriter e Scanner che verranno poi passati ai metodi descritti sopra, e questi saranno responsabili dell'inoltro e della ricezione dei dati tra client e server.

RandomSet.java

Come scelta di implementazione è stato preferito scrivere una nuova classe per il mantenimento del vocabolario di parole da indovinare. Per farlo si è scelto un HashSet come struttura dati, modificata in modo da aggiungere un metodo per

il sorteggio casuale di una parola dall'insieme. Tra i metodi basilari della classe si riporta ***getRandom()*** che ha la funzione convertire il set in un array, generare un indice randomico tramite la funzione random e, una volta oppurtunatamente "castato" il dato, ottenere un valore casuale dall'array.

GestoreFile.java

La classe in questione ha il compito di semplificare il codice del task portando tutte le letture e scritture in una classe separata. Oltre ad una funzione organizzativa del codice, la scelta è stata pensata anche per rendere scalabile il progetto. Infatti, la classe genera le strutture dati necessarie per il ServerMain ma rende completamente trasparente il supporto di memorizzazione persistente dei dati, potendo sostituire, per esempio, il file utenti.json, con un più efficiente database semplicemente cambiando questa classe.

I metodi funzionano tutti con il medesimo meccanismo: viene letto il valore del file da una costante e i metodi restituiscono una struttura dati con i dati inseriti.

Differente è invece il metodo ***saveUtenti(ConcurrentHashMap<String, Utente> mappaUtenti)*** che lavora in maniera inversa, ossia gli viene passata la struttura dati e questo va a scrivere su file i dati.

Attenzione: Per il corretto funzionamento del programma si necessita che il file utenti.json, se privo di dati degli utenti, abbia i caratteri "[]" all'interno.

Anche la lettura delle configurazioni del server e del client sono state spostate all'interno di questa classe.

ServerMain.java

ServerMain.java è la classe responsabile della generazione del thread main che andrà a gestire e a far comunicare i giocatori con i task creati. Il funzionamento è tramite Java I/O con l'utilizzo di threadPool e la comunicazione avviene tramite Socket TCP, quindi si metterà in ascolto tramite un Socket (ServerSocket) in attesa di connessione da parte di nuovi utenti. Questa andrà a restituire un nuovo Socket su cui avverrà la comunicazione effettiva con il task. Alla stessa classe è affidata anche la gestione del server multicast per la condivisione dei risultati delle partite.

Nel main è presente anche un Hook java, questo permette di salvare in maniera persistente i dati modificati durante le varie partite. Un hook è un meccanismo che consente di eseguire del codice personalizzato quando determinati eventi del sistema si verificano, come ad esempio la chiusura dell'applicazione o l'arresto del sistema operativo. In questo caso è stato utilizzato uno Shutdown Hooks: questo viene eseguito prima che l'applicazione venga terminata, ad esempio quando viene chiamato il metodo **System.exit()** o quando il sistema operativo riceve un segnale di interruzione come **SIGTERM** o **CTRL+C**.

I metodi ritenuti più interessanti sono riportati di seguito:

- ***startServer()***

  Questo metodo ha il compito di inizializzare il server con i parametri opportuni, letti dalla struttura dati generata da GestoreFile.java, inizializzare e riempire le varie strutture dati utilizzate poi per la gestione degli utenti e delle partite.

- ***registerUser(String username, String password)***

  La registrazione degli utenti funziona tramite questo metodo (richiamato poi dal task). Una volta fornito l'username come parametro verifica che questo non sia presente all'interno della struttura dati *utenti,* in caso di esito positivo viene restituito l'errore "—AE" (AccountEsistente). Superato questo controllo si verifica che la password non sia vuota altrimenti viene restituito l'errore "—EP" (ErrorePassword).

  > Attenzione: per un implementazione più realistica sarebbe opportuno aggiungere maggiori controlli per la password, in modo che rispetti gli standard di sicurezza e renderla più affidabile. Allo stesso modo sarebbe opportuno utilizzare una cifratura delle password per evitare di salvarle in chiaro nel DB. Per far questo si potrebbe utilizzare la cifratura AES. Java ne permette l'utilizzo tramite le librerie java.crypto.Chiper, java.crypto.spec.SecretKeySpec e java.util.Base64,

  Se la password supera il controllo si procede alla creazione dell'account creando un oggetto User e inserendolo all'interno della struttura dati *utenti.*

- ***wordGenerator()***

  Si riporta questo metodo che non girerà nel thread main, ma sarà in *run* su un thread separato dove, ogni DELAY secondi verrà generata ed assegnata una nuova parola da indovinare tramite il metodo della classe RandomSet visto prima.

- ***matches_played(String username)***

  Ha il compito di controllare le giocate degli utenti. Difatti, si deve evitare che un giocatore giochi più volte una parola (nel caso di vittoria o nel caso di sconfitta per terminazione di tentativi). Se questo controllo restituisce un esito negativo si aggiorna la struttura dati *giocate_utenti* che mantiene un'associazione tra l'username ed un valore booleano true in caso di partita giocata o campo mancante in caso di partita non ancora disputata. Viene restituita la parola da indovinare.

- ***check_matches_played(String username)***

  Ha lo stesso fine del metodo visto sopra con la differenza che qui non si va ad aggiungere l'utente nella HashMap, ha il solo scopo di eseguire un controllo della presenza o meno dell'utente nella struttura dati.

- ***multicast()***

  E' il metodo che gestisce il server multicast per le classifiche del giocatori. Resta in attesa di un messaggio da parte di un client che condivide le proprie statistiche con ***share().*** Una volta ricevuto il messaggio viene eseguito un casting in String e viene aggiunto alla stringa i dati dell'utente mittente. (Formattazione di esempio di una stringa: {dati utente}$ {Statistiche}. Il carattere speciale $ è utilizzato per la separazione più efficace dei due campi.). Infine viene inoltrato al gruppo multicast il messaggio.

ClientMain.java

La classe ClientMain è quella utilizzata dal client per connettersi, poter comunicare con il ServerMain e quindi giocare. All'esecuzione del main la

classe crea un nuovo Socket con il quale tenta di connettersi al Server. La classe è caratterizzata principalmente dai menu di gioco, infatti, non essendoci la GUI si è cercato di rendere l'esperienza di gioco il più intuitiva possibile. I metodi interagiscono con le funzionalità descritte in precedenza. Hanno quindi il compito di contattare il server tramite i codici necessari e, tramite i segnali che il server restituisce, agire di conseguenza. Come nel server anche in questa classe, nel main, viene utilizzato uno shutdownHooks con lo scopo di eseguire il logout se questo non è stato fatto dall'utente connesso prima di terminare il programma. Nel seguito verranno descritti alcuni metodi, ritenuti di maggior interesse.

- ***startClient()***

  Questo metodo, come già descritto per il server, ha il compito di inizializzare e caricare da file le costanti che serviranno per la configurazione e la connessione con il server (SERVER_PORT,SERVER_ADDRESS, ecc..). Inoltre viene creato anche un oggetto Utente a cui sarà associato il giocatore che effettuerà il login.

- ***login(PrintWriter out, Scanner in,Scanner cmd,ExecutorService threadMessaggi)***

  La classe login esegue il login dell'utente: viene inviato tramite PrintWriter out sul Socket al la richiesta di login al server, in base alla risposta ricevuta tramite Scanner in viene eseguita l'istruzione corretta. Scanner cmd invece servirà per prendere le credenziali dell'utente da riga di comando. Se il login va a buon fine viene fatta la submit() del thread threadMessaggi dichiarato a livello globale che gestisce la comunicazione multicast per la ricezione e l'invio dei messaggi.

- ***addNotification()***

  Tramite questo metodo viene gestita la funzionalità multicast del client. Viene istanziato un nuovo oggetto di tipo MulticastSocket, e aggiunto al gruppo multicast tramite .joinGroup(). Dopodiché il metodo rimarrà attivo sul threadMessaggi finchè il giocatore non si disconnette (game=false). Il metodo rimane in attesa di messaggi dal Server e quando riceve una

nuova notifica quest'ultima viene aggiunta alla lista dichiarata a livello globale ***notifiche.***

- ***share(String stat)***

  Per condividere le statistiche della partita invece viene utilizzato il metodo share(String stat). Quando viene chiamato si occupa di inoltrare sul server multicast le statistiche dell'utente.

- ***showMeSharing()***

  Per poter leggere le notifiche è necessario utilizzare il metodo showMeSharing() che non fa altro che scorrere la lista globale notifiche e per ogni elemento viene chiamata una *System.out.println()*. Per comodità verrà stampata anche la lunghezza di tale lista per riportare quanti messaggi non letti sono presenti.

Altri metodi, come la register() e il logout() utilizzano la stessa strategia, hanno come parametri PrintWriter out, Scanner in e tramite questi inoltrano e ricevono tramite Socket i messaggi con il server e in base alla risposta che ottengono viene gestita graficamente la partita.

Compilazione ed esecuzione del progetto

Il progetto è compresso in un archivio zip. Estraendolo si ottiene la cartella contenente i 6 file .java:

> → Utenti.java

> → RandomSet.java

> →GestoreFile.java

> →Task.java

> →ServerMain.java (main server)

> →ClientMain.java (main client)

WordleTorbidoni.zip

WordleTorbidoni

All'interno della solita cartella è presente una sottocartella (config) che contiene tutti i file di configurazione e d'utilizzo del programma (*serverdetail.txt,clientdetail.txt,utenti.json,words.txt*)

Ci sono inoltre altre due sottocartelle:

> → lib : contiene la libreria utilizzata gson-2.2.2.jar

> → class : contiene i file .class generati dalla configurazione

Compilazione:

ClientMain.java

```
javac -cp .:lib/gson-2.2.2.jar ClientMain.java Utente.java
```

ServerMain.java

```
javac -cp .:lib/gson-2.2.2.jar ServerMain.java Utente.java
                          Task.java GestoreFile.java RandomSet
```

Questi comandi genereranno i file .class

Eseguibili

Per eseguire i file .class generati dalla compilazione è sufficiente utilizzare i comandi:

Client

```
java -cp .:lib/gson-2.2.2.jar ClientMain
```

Server

```
java -cp .:lib/gson-2.2.2.jar ServerMain
```

Nella cartella del progetto viene fornita anche la versione del codice eseguibile (.jar) W**ordle_server.java Wordle_client.java**

Per l'esecuzione è necessario eseguire i seguenti comandi una volta situati nello stesso path dei file:

```
java -jar Wordle_server.jar
```

```
java -jar Wordle_client.jar
```

Istruzioni per il gioco

Una volta configurati i parametri di server e client e una volta avviato server è possibile avviare il client che rappresenterà l'interfaccia di gioco:

Schermata iniziale:

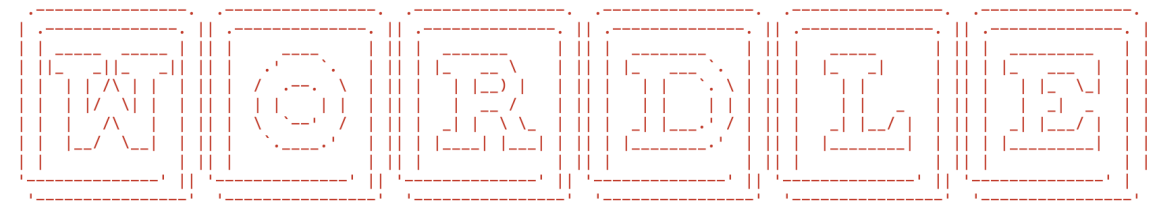
→con "r" è possibile effettuare una registrazione di un nuovo utente

→una volta che abbiamo un account premendo "l" è possibile effettuare il login

→se si vuole sapere come funziona il gioco è possibile farlo richiamando l'help menu con "?"

→con "q" è possibile eseguire il logout e uscire dal gioco

Esempio registrazione



```
Benvenuto a Worlde!

[r] - Registrati!

[l] - Login!

[?] - Regole di gioco!

[q] - Esci dal gioco!

r
 -- REGISTRATI ORA! --

Inserisci l'username!

test
Inserisci la password!

test
```

Menu principale

```
Login efettuato, Benvenuto test!

[p] - Nuova partita!

[s] - Statistiche!

[c] - Classifiche!


[q] - Log-out!
```

Una volta effettuato il login ci viene proposto il menu principale dove è possibile:

→"p" avviare una nuova partita con la parola del giorno

→"s" visualizzare le statistiche del giocatore attualmente connesso

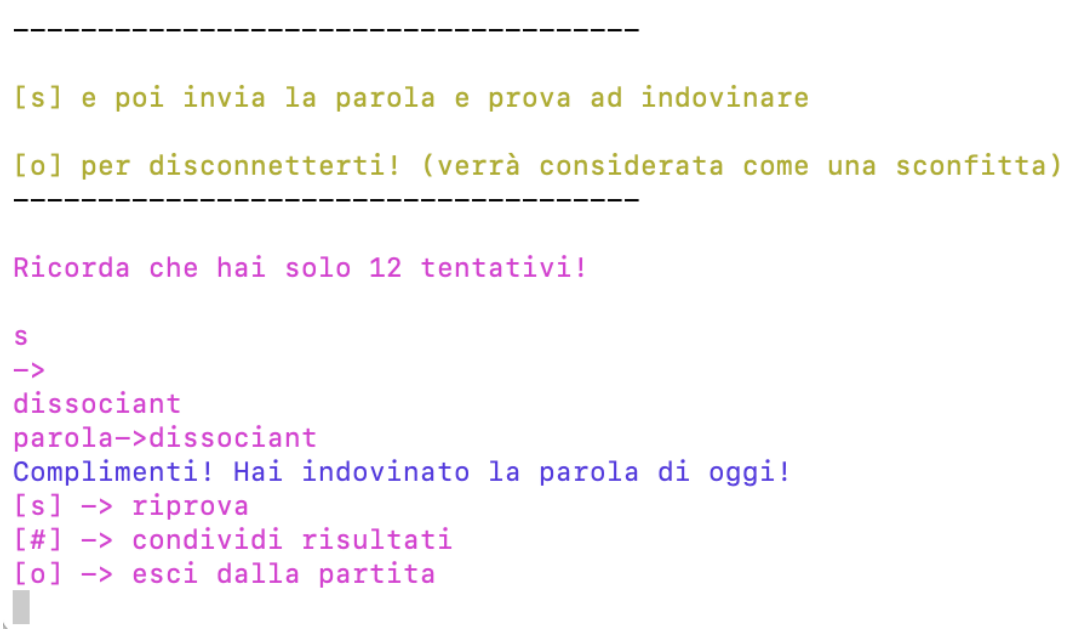
```
----STATISTICHE----
Utente: test
Partite giocate: 1
Vittorie: 1
Streak: 1
Max streak: 1
Distribuzione: :1,0,0,0,0,0,0,0,0,0,0,0,
---------------------------
```

→"c" visualizzare le statistiche pubblicate dagli altri giocatori

→"q" eseguire il logout e uscire dal gioco

Partita in corso

Se si decide di giocare una partita questo è ciò che vedremo



Prima di inviare una parola da indovinare è necessario premere “s”.

Se la parola è corretta ci viene assegnata la vittoria e, se si vuole rigiocare, si deve attendere la generazione di una nuova parola (nella configurazione in allegato viene generata una parola ogni 60 secondi). Altrimenti, se la parola è presente nel vocabolario di parole (words.txt) ma non è stata indovinata avremo altri 12 tentativi.

Una volta finita la partita è possibile:

→”s”: inviare una nuova parola

→”#”: condividere con gli altri giocatori le statistiche

→”o”: uscire dalla partita

Caso: tempo per indovinare la parola terminato

```
[s] e poi invia la parola e prova ad indovinare

[o] per disconnetterti! (verrà considerata come una sconfitta)
------------------------------------

Ricorda che hai solo 12 tentativi!

s
->
indovinare
parola->indovinare
La parola è cambiata, inzia una nuova partita!
[s] -> riprova
[#] -> condividi risultati
[o] -> esci dalla partita
```

In questo caso il giocatore ha giocato ma nel frattempo è stata generata una nuova parola: è sufficiente uscire dalla partita attuale (”o”), iniziare una nuova partita (”p”) e con “s” comunicare la nuova parola.