Wordle: un gioco di parole 3.0

Introduzione

Il progetto consiste nell'implementazione di una versione semplificata (e in locale) del gioco **wordle**. In questo caso, consiste di due applicativi (.jar) uno per il server (**Server.jar**) e uno per il Client (**Client.jar**). Il server deve essere attivo per far funzionare i client, ed è pensato per essere acceso per tempo indefinito. I client sono invece pensati per essere accesi e spenti multiple volte nell'arco dello stesso ciclo di vita del server. La scelta del protocollo di comunicazione è stata fatta basandosi sulla più comprensibile e semplice possibile e cercando di forzare molto il comportamento per evitare errori. Questo ha portato a scegliere la struttura Codice Operazione -> Eventuali dati su un **ObjectOutput/InputStream** che rende molto facile e intuitiva la comunicazione di dati che sono oggetti conosciuti da entrambi gli interlocutori. C'è bufferizzazione dello stream per minimizzare letture e scritture.

Server.jar

Contiene le classi:

-**WordleServerMain.java**

La classe "entry point" dell'eseguibile, inizializza tutti i thread ausiliari e gestisce, attraverso opportune classi private interne, sia la connessione dei client che alcune parti del gioco in sé.

Da sottolineare: la scelta di usare I/O classico invece di NIO per evitare complessità di codice e aumentare la leggibilità, dato che le comunicazioni da gestire sono molto semplici e di piccole dimensioni. (Si prevede anche un basso numero di client simultanei).

Thread pool di dimensione fissata per gli handler dei client connessi.

Una lock per l'accesso allo status dei giocatori, per evitare che le scritture e le letture avvengano nell'ordine sbagliato.

Una lock per l'accesso alle informazioni dei giocatori per evitare dati incoerenti e possibili "doppie registrazioni"

Il server viene avviato all'apertura dell'eseguibile e viene chiuso o tramite un errore nell'esecuzione, o digitando "**exit**" da linea di comando. Alla fine dell'esecuzione c'è sempre il tentativo di salvare la **coreMap** (contenente nomi password e statistiche) nel file **players.json**, oltre alla chiusura di tutti gli stream e i socket aperti.

-**GameLogic.java**

Contiene la logica di gioco, i calcoli necessari per una partita. In pratica mantiene memorizzata (per ogni giocatore) lo stato della partita (se è in corso oppure no) e mette a disposizione il metodo per confrontare la parola da indovinare (che memorizza staticamente essendo uguale per tutti) con il tentativo, restituendo la stringa di "colori". Provvede inoltre all'aggiornamento delle statistiche opportunamente.

-**PlayerInfo.java**

La struttura che contiene password e statistiche dei giocatori, ce n'è una per giocatore registrato nella **coreMap**

Thread attivati

Il thread **main** inizializza gli altri thread e si inserisce in un loop che accetta connessioni e fa partire un handler per ognuna di esse.

Una thread pool di dimensione fissata configurabile gestisce appunto tutte le connessioni e le comunicazioni TCP con i client.

Una **ScheduledExecutorService** gestisce il sorteggio della nuova parola a intervalli prefissati, configurabile.

Un thread in più è riservato all'ascolto della CLI per il comando **"exit"**, per chiudere bene il server.

Client.jar

Contiene le classi:

-WorldeClientMain.java

Entry point dell'eseguibile, attiva il thread di gioco e si limita a aspettare che termini, senza fare niente.

-Game.java

Contiene tutta la logica del gioco e la comunicazione con il server, si occupa anche del gruppo di multicast.

Unica sincronizzazione necessaria, quella per evitare che la struttura dati mantenuta per le partite condivise ricevute sul gruppo di multicast venga modificata mentre viene copiata.

-Renderer.java

Utilizza java **swing** per visualizzare la GUI del gioco, comunica con la classe **Game** per le funzionalità dei tasti dell'interfaccia e per la visualizzazione dei dati ricevuti da server.

Thread attivati

Il thread **main** si limita ad aspettare la terminazione del thread **Game**, scelta forse inefficiente ma ritenevo più corretto semanticamente lasciare al **main** le fasi di inizializzazione e alla classe **Game** il gioco in sé e per sé.

Il thread **Game** legge dall'input stream in un loop per le comunicazioni del server e chiama le funzioni della classe **Renderer** in accordo con ciò che riceve, termina quando c'è un errore o quando si clicca la x della finestra di gioco.

L'event dispatch thread viene chiamato tutte le volte che si preme qualcosa sull'interfaccia o che si scrive una lettera, e chiama i metodi della classe **Game** che comunicano con il server.

Un thread in background ascolta sul gruppo di multicast e attende partite condivise per poi salvarle in una **ArrayList**.

Il thread **Renderer** viene attivato all'inizio e serve a inizializzare la GUI e termina subito.

Shared.jar

File .jar usato come "libreria", contiene le classi condivise tra entrambi gli applicativi:

ShCon.java, Stat.java, StatSnap.java, ShareInstance.java

-**Stat.java**

La struttura dati che incapsula le informazioni delle statistiche di un giocatore. C'è un'istanza dell'oggetto **Stat** per giocatore registrato all'interno della **coreMap.**

-**StatSnap.java**

Classe ausiliaria per copiare un'oggetto **Stat** e mandarlo al client per la visualizzazione. Soluzione a problemi di caching che rendevano complicato l'aggiornamento delle statistiche, o meglio la visibilità dell'aggiornamento.

-**ShCon.java**

Una classe che contiene alcuni alias per numeri interi utilizzati nel protocollo di comunicazione in modo da aumentare di molto la leggibilità del codice.

-**ShareInstance.java**

Struttura che rappresenta una singola partita condivisa sottoforma di nome e lista di stringhe di "colori"

Guida (Developer)

Per modificare i valori di configurazione, si modifichino i file **ServerSettings.json** e **ClientSettings.json** facendo attenzione a non alterarne la struttura ma solo i valori dei campi.

Per compilare si usa (supponendo di avere solo i file .java, quelli di configurazione, i json, i manifest e i .bat) (terminale di windows)

Dalla cartella **"root\libs\":**

>**javac *.java**

>jar cvf Shared.jar *.class

>del *.class (non necessari per l'esecuzione)

Per creare il jar di classi condivise. Poi in ordine indifferente,

Dalla cartella **"root\Server\"**

>javac -cp ".\;..\libs\gson-2.10.1.jar;..\libs\Shared.jar" WordleServerMain.java

>jar cvfm Server.jar ServerManifest.MF *.class

>del *.class

E dalla cartella **"root\Client\"**

>javac -cp ".\;..\libs\gson-2.10.1.jar ;..\libs\Shared.jar" WordleClientMain.java

>jar cvfm Client.jar ClientManifest.MF *.class

>del *.class

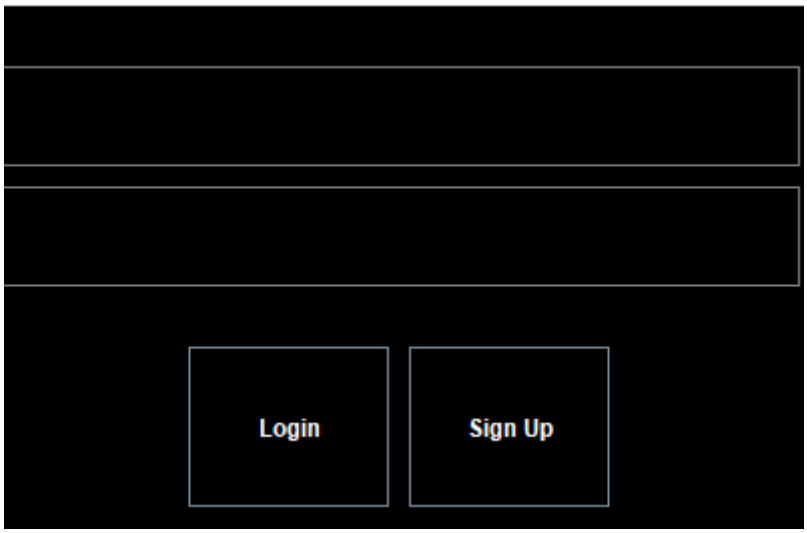
Ora l'applicazione è pronta ad essere eseguita.

Guida (User)

Il server **deve** essere avviato prima di qualsiasi client **una volta sola** eseguendo il file **Server.bat** Questo assicura che venga aperto un terminale da cui chiudere il server successivamente attraverso la scrittura di **"exit"** **unico** modo corretto di interrompere il server.

Dopodichè è possibile aprire un numero qualsiasi di client, sia attraverso **Client.bat** che **Client.jar**. Il primo permette di avere un log di eventuali errori a tempo di esecuzione, il secondo no. Non è necessario per il funzionamento avere il terminale a schermo.

Il client dopo l'attivazione presenta questa

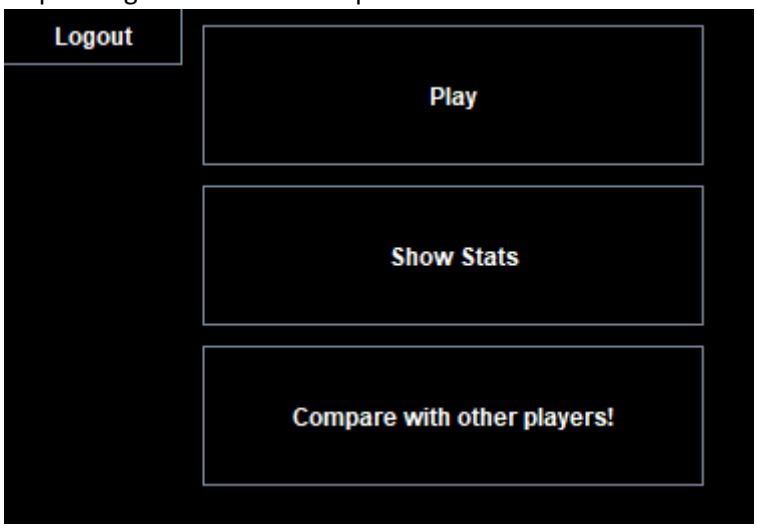


Schermata, dove si possono inserire nome e password e cliccare su **"Login"** o **"Sign Up"**.

"Sign Up" serve per registrarsi per la prima volta al sistema, univocamente per username. Fallisce se si è già registrati, o se la password è vuota.

"Login" serve per entrare "online", mettendo la password usata per registrarsi corrispondente all'username. Fallisce se già online, se nome e password non corrispondono a quelli presenti nei dati del server.

Dopo il Login la schermata è questa:

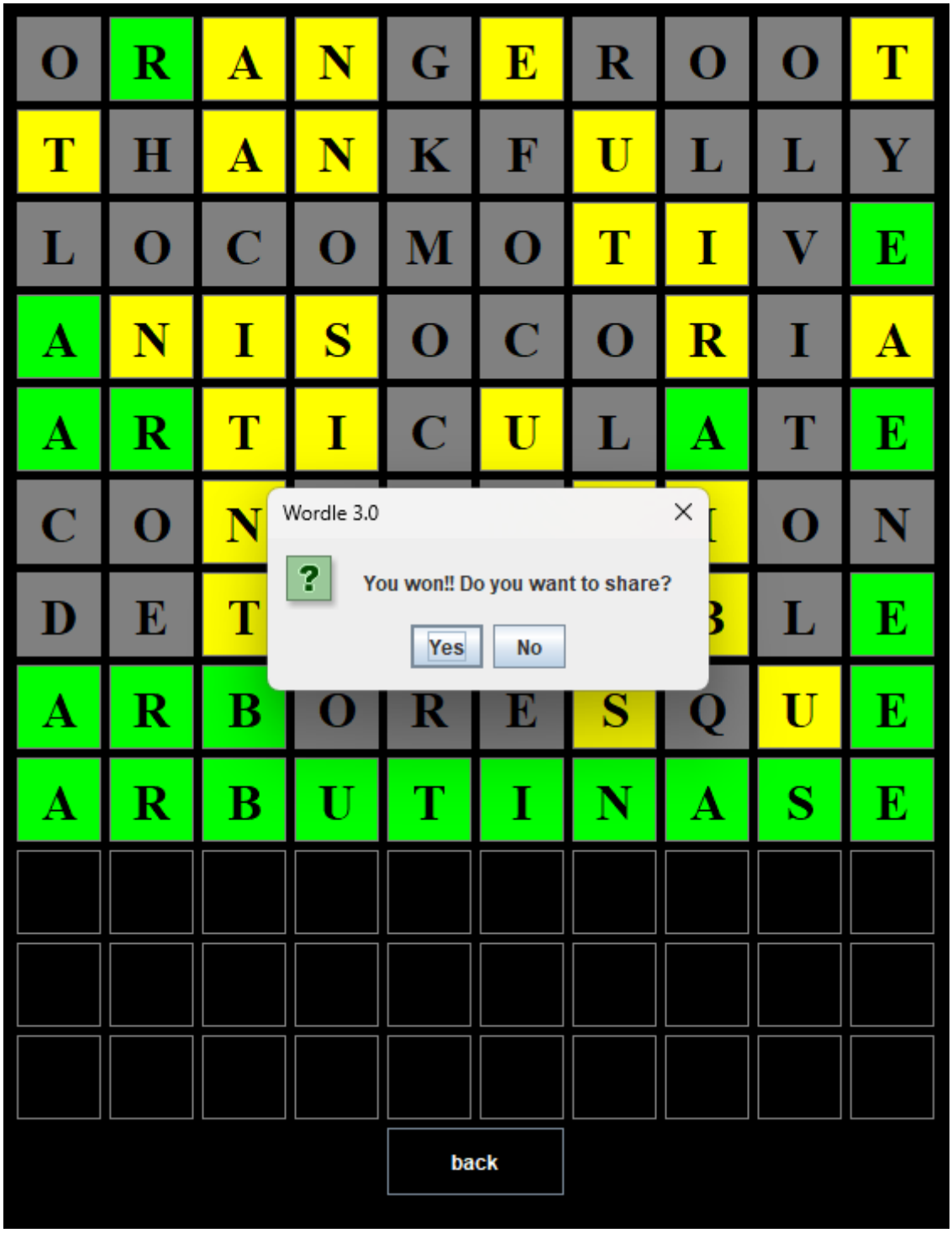


“Logout”: esegue il logout

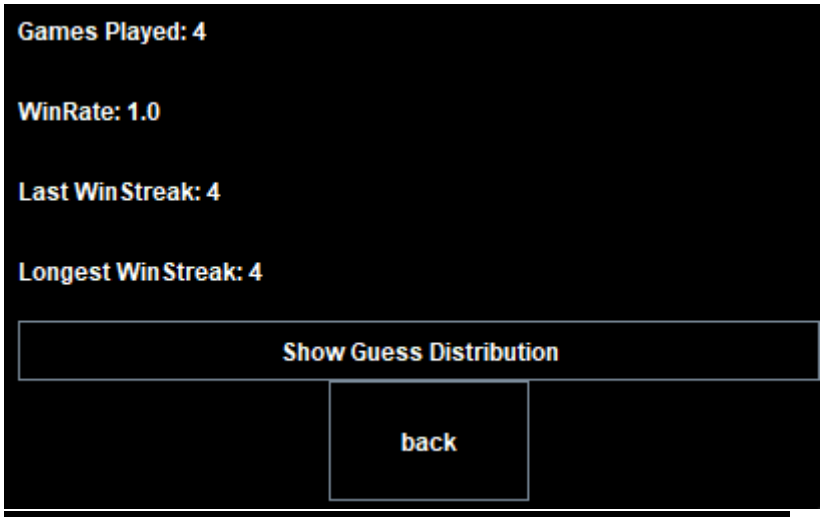
“Play”: Per tentare di indovinare la parola “del giorno”, fallisce se si è già finito un tentativo oggi. Se una partita è in corso (né vinto né perso), **“Play”** riprende quella partita.

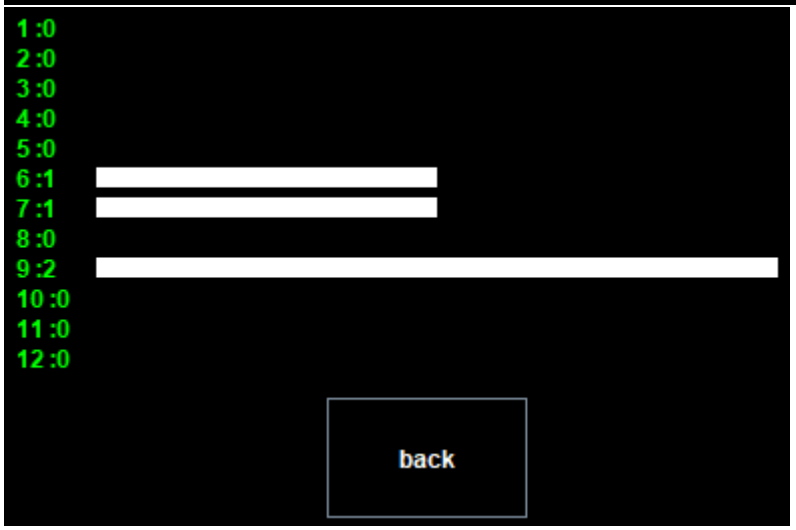
“Show Stats”: mostra le statistiche aggiornate all’ultima partita **completata**

“Compare with other players!”: Mostra la lista di partite condivise dall’avvio del client in poi, in ordine di arrivo.

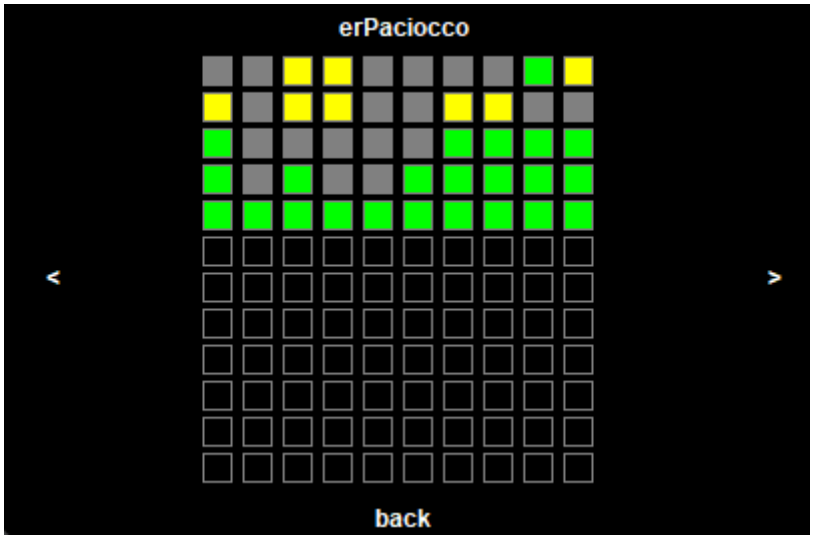


Il gioco in sé è piuttosto intuitivo, si scrive la parola con la tastiera e si manda con invio, le notifiche a schermo e i colori guidano il resto. Il tasto **“back”** non perde automaticamente. Il logout, o la chiusura della finestra di gioco invece sì.





Statistiche e guess distribution sono anch'esse piuttosto intuitive.



Il pannello delle partite condivise mostra le partite completate con successo e condivise dagli altri giocatori in ordine di condivisione dalla meno alla più recente. Le frecce non fanno niente

se si tenta di andare prima della meno recente o oltre la più recente. L'aggiornamento delle partite non avviene a tab aperta, ma si deve premere "**back**" e **"Compare with other players!"** di nuovo.

Note per la configurazione

Il numero di thread limita il numero di client responsive simultaneamente. (E.G. **NTHREAD** = 5, si aprono 6 client, il sesto sarà unresponsive) questo significa che le azioni verranno messe in coda ed eseguite appena si libererà un thread nella pool per eseguire il client. Si consiglia quindi di settare **NTHREAD** a un numero superiore al numero di client previsti. Per il resto dei parametro bastano valori "legali" e porte nel range corretto.

Per un'esperienza di gioco piacevole si suggerisce di settare l'intervallo tra una parola e la successiva ad almeno qualche ora, per evitare caos nella sezione condivisione con partite su parole diverse mostrate in fila, magari addirittura della stessa persona.