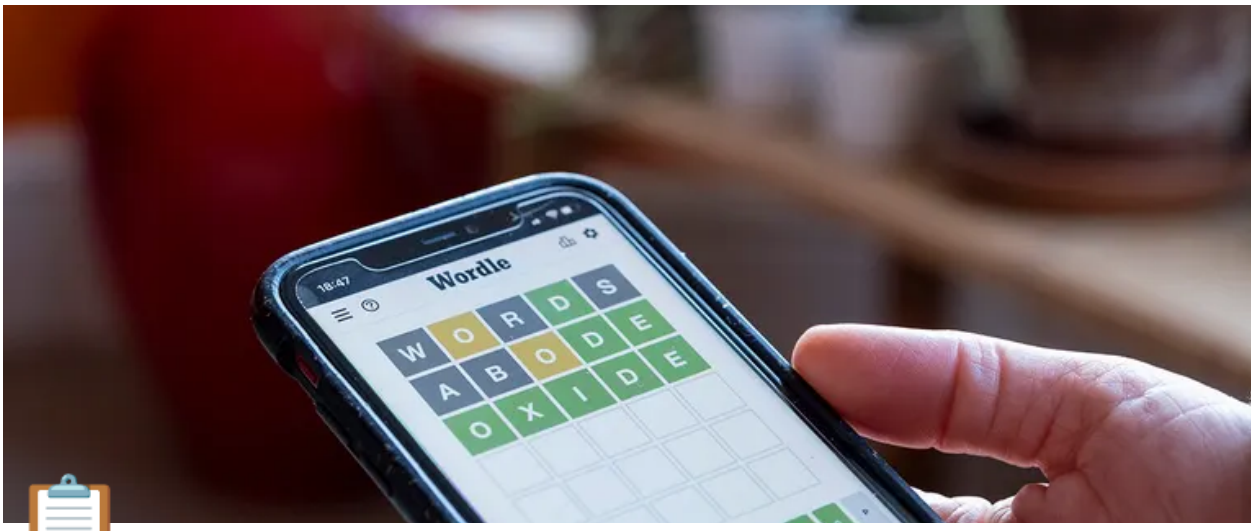


Relazione progetto: Wordle un gioco di parole 3.0

Introduzione

Wordle è un mini-gioco del New York Times divenuto virale nel 2021. Questo progetto implementa lo stesso gioco con un paradigma Client-Server implementato con Java, dando la possibilità a più utenti di giocare in contemporanea e condividere le proprie statistiche in un gruppo social. Ogni client, dopo essersi connesso con il server può giocare con la parola da indovinare in quel momento, in seguito può condividere i suoi risultati ed aspettare che la parola cambi per poter rigiocare.

Struttura generale

Il progetto è basato sul paradigma Client-Server con connessione persistente TCP. Inizialmente il server viene avviato e sceglie una nuova parola dal file, in seguito ogni client che desidera partecipare può connettersi al Server. Ogni client, dopo aver fatto login può iniziare a giocare a Wordle. Alla fine di ogni partita il client può condividere le proprie statistiche nel gruppo multicast e visualizzare le statistiche degli altri utenti. La comunicazione è un semplice “botta e risposta” tra Client e Server.

Struttura del progetto Java

Partendo dall'alto si trova la cartella `Compilazione` che contiene un file di testo compilazione.txt con le direttive per la compilazione e l'esecuzione del programma.

Di seguito una cartella `GSON Jar` che contiene i file relativi alla libreria GSON, utilizzata per interagire con i file json.

All'interno della cartella `Wordle` sono contenute le cartelle `bin`, `lib` `config` e `src`.

- `bin`: contiene tutte le classi compilate per l'esecuzione del programma e la lista degli utenti registrati al sistema.
- `lib`: cartella contenente la libreria `gson-2.10.jar` per la serializzazione e deserializzazione della lista degli utenti registrati salvata nel file registeredUsers.json così da garantire che non venga persa una volta che tutto il progetto viene chiuso.
- `config`: cartella contenente i file di configurazione delle classi client, server e wordlegame.
- `src`: cartella contente tutte le classi sorgente scritte in Java, in particolare le classi client e server, la classe per gestire la terminazione del server, la classe che gestisce il gioco e la

comunicazione con i client e la classe relativa all'utente.

Nella cartella `Words` è presente il vocabolario di parole words.txt dal quale il server periodicamente sceglie la nuova parola segreta da trovare.

File di proprietà, archivio degli utenti e file delle parole

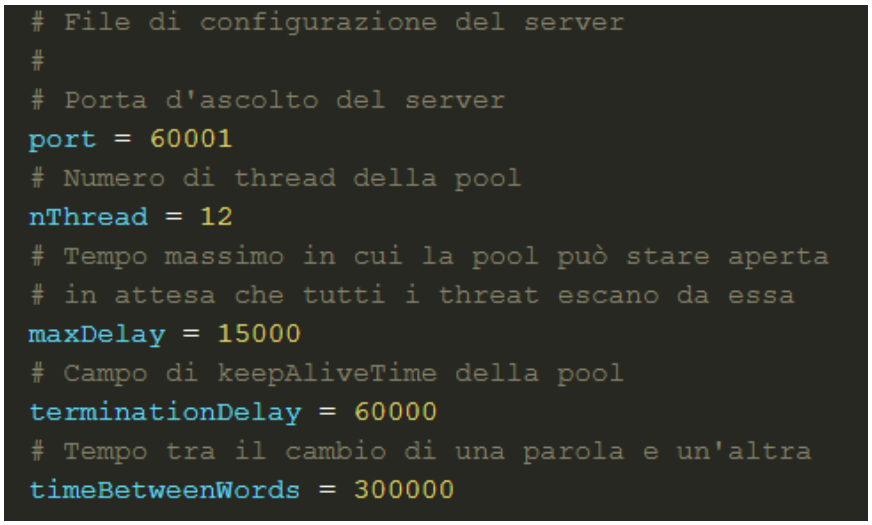
Nella cartella `config` si trovano i 3 file di configurazione per le classi principali:

- client.properties: file di configurazione del client. Al suo interno vengono assegnati i valori al nome dell'host, la porta di ascolto e i campi relativi al gruppo multicast.

```
# File di configurazione del client
#
# Nome dell'host del client.
hostname=localhost
# Porta di ascolto del client.
port=60001
# Porta e Host del gruppo multicast [224.0.0.0 - 239.255.255.255]
multicastPort = 55555
multicastHost = 226.226.226.226
```

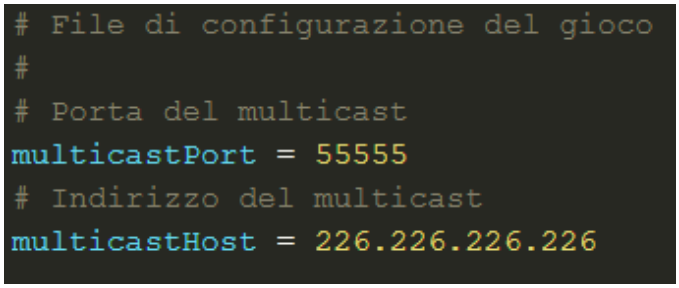
client.properties

- server.properties: file di configurazione del server principale. Qui vengono assegnati i campi per la comunicazione con i client e in particolare i campi per gestire i `thread` di comunicazione.



server.properties

- wordlegame.properties: file di configurazione per la classe `WordleGame.java` contente i soli campi relativi al multicast.



wordlegame.properties

- registeredUsers.json: all'interno della cartella `bin` si trova questo file contenente tutti gli utenti registrati al sistema. Ciò garantisce la persistenza dei dati relativi agli utenti in caso di chiusura del server o dell'intero progetto.

Esempio di utente registrato nel file json

Nella cartella `Words` si trova il file words.txt: in questo file sono contenute 30824 parole lunghe 10 caratteri. Da questo file il server periodicamente sceglie una nuova parola segreta.

Classi del progetto

All'interno della cartella `src` vi sono le 5 classi Java del progetto, in particolare:

- `ClientMain.java`: permette la comunicazione tra l'utente umano e il client tramite il terminale. L'utente può inserire i comandi specificati nel menu, la classe si occupa di inviarli al server ed elaborare la risposta. Inoltre durante la fase di gioco stampa a video i suggerimenti posti dal server (parola con le lettere colorate).
- `ServerMain.java`: questa classe permette ai client di connettersi alla porta in ascolto, legge la lista degli utenti in registeredUsers.json trasformandola in una

`ConcurrentHashMap` e si occupa periodicamente di scegliere la nuova parola segreta dal vocabolario delle parole.

- `WordleGame.java`: classe task che viene chiamata dal server principale sotto forma di `thread` all'interno della `threadpool`. É la classe che gestisce la comunicazione tra l'utente (client) e il server, che risponde ai comandi ed esegue le operazioni richieste.
- `User.java`: classe che rappresenta l'utente iscritto al sistema. Contiene tutti i campi e i metodi relativi al calcolo delle statistiche.
- `TerminationHandler.java`: classe che gestisce la terminazione del server. Se si interrompe il server (con ad esempio il comando ^C), questa classe viene chiamata e mandata in esecuzione per salvare la lista degli utenti nel file registeredUsers.json garantendo che non vada persa.

Scelte implementative

Classe ClientMain

Nel metodo `main` per prima cosa si legge il file di configurazione con il metodo `readConfig()` e si inizializzano le variabili che servono per la comunicazione TCP con il server. Si inizializzano anche le stream di input/output per lo scambio dei messaggi e il socket del multicast.

```
// Leggo file di configurazione "client.properties"
readConfig();

// Strutture dati per comunicare col server
input = new InputStreamReader(System.in);
br = new BufferedReader(input);
socket = new Socket(hostname,port);
out = new PrintWriter(socket.getOutputStream(),autoFlush:true);
in  = new Scanner(socket.getInputStream());
multicastSocket = new MulticastSocket(multicastPort);
```

Parte iniziale del main del client

```java
public static void readConfig() throws FileNotFoundException, IOException {
  InputStream input = new FileInputStream(configFile);
  Properties prop = new Properties();
  prop.load(input);
  hostname = prop.getProperty(key:"hostname");
  port = Integer.parseInt(prop.getProperty(key:"port"));
  multicastPort = Integer.parseInt(prop.getProperty(key:"multicastPort"));
  multicastHost = prop.getProperty(key:"multicastHost");
  input.close();
}
```

Metodo che legge il file di configurazione

Questo metodo semplicemente legge il file client.properties e assegna alle variabili sopracitate il proprio valore.

Subito dopo si istanzia il thread che si occupa di partecipare al gruppo multicast tramite il protocollo UDP.

```java
// Thread che si inizializza non appena il client fa la join nel gruppo multicast
Thread threadListenerUDPs = new Thread () {
  public void run () {
    System.out.println(x:"CLIENT: Waiting shares...");
    while(!endMulticast){
      // Se ricevo condivisioni tramite UDP le stampo nel client che ha mandato show me sharing
      DatagramPacket response = new DatagramPacket(new byte[1024], length:1024);
      try {
        multicastSocket.receive(response);
      } catch (IOException e) {
        System.out.println(x:"CLIENT: Client is not reciving messages.");
      }
      String responseUDPfromServer = new String(response.getData(), offset:0, response.getLength());
      notifications.add(responseUDPfromServer);
      // System.out.println("CLIENT: Results from server: " + responseUDPfromServer);
    }
  }
};
```

Istanza del thread relativo al multicast

Il metodo al suo interno fa si che all'interno del while si attenda una risposta dal server, ossia che qualcuno condivida le sue statistiche nel gruppo. Dal momento in cui viene ricevuta una notifica si costruisce la risposta sotto forma di stringa e si salva nella variabile `responseUDPFromServer`.

In seguito si stampa a video il menu delle azioni con il metodo `printMainMenu()`, questo metodo non fa altro che stampare l'elenco delle azioni disponibili dopo aver pulito il terminale.

```java
public static void printMainMenu(){
  System.out.print(s:"\033[H\033[2J");
  System.out.flush();
  System.out.println(x:"----------------");
  System.out.println(x:"Register");
  System.out.println(x:"Login");
  System.out.println(x:"Logout");
  System.out.println(x:"Play Wordle");
  System.out.println(x:"Player statistics");
  System.out.println(x:"Share my stats");
  System.out.println(x:"Show me sharing");
  System.out.println(x:"Main menu");
  System.out.println(x:"Exit");
  System.out.println(x:"----------------");
}
```

Stampa dell’elenco delle azioni

Successivamente si crea il comando da inviare al server e per prima cosa si controlla che non sia il comando di uscita “exit”. In quel caso si interrompe la connessione con il server e col gruppo multicast, se invece è un altro comando si invia al server.

```java
// Scrivo il comando da mandare al server
toServer = br.readLine();

// Se il client desidera resettare il menu
if(toServer.toLowerCase().equals(anObject:"menu")){
  printMainMenu();
  continue;
}

// Se il client vuole uscire interrompo la comunicazione con il server e con il gruppo multicast
if(toServer.toLowerCase().equals(anObject:"exit")){
  System.out.println(x:"CLIENT: Exiting...");
  end=true;
  endMulticast=true;
  out.println(toServer);
  continue;
}else{
  out.println(toServer);
}
```

Invio del comando verso il server

É interessante vedere cosa succede quando il login è andato a buon fine. In questo caso il client resetta l’array delle notifiche e si unisce al gruppo di multicast assieme agli altri utenti che si sono loggati.

```
// Ricevo la risposta dal server e faccio i controlli su di essa successivamente.
fromServer=in.nextLine();

// Se il login è andato a buon fine allora rimuovo le vecchie notifiche dall'arra
if(fromServer.equals(anObject:"LOGIN: Login Successful.")){
  //System.out.println(fromServer);
  notifications.clear();
  endMulticast=false;

  InetAddress group = InetAddress.getByName(multicastHost);
  InetSocketAddress address = new InetSocketAddress(group, multicastPort);
  NetworkInterface networkInterface = NetworkInterface.getByInetAddress(group);

  multicastSocket.joinGroup(address, networkInterface);
  System.out.println("CLIENT: You've just joined this group: " + multicastHost);
  threadListenerUDPs.start();

  continue;
}
```

Partecipazione al gruppo multicast dopo il login

Nel caso in cui si invia il comando di “show” oppure “logout” si eseguono le seguenti operazioni. Per il comando “show” semplicemente si stampa a video il messaggio del server, per il logout invece si chiude il gruppo multicast.

```
// Se il client vuole vedere le notifiche e il server acconsente il client
if(fromServer.equals(anObject:"SHOW ME SHARING: Request sent.")){
  System.out.println(fromServer);
  System.out.println("CLIENT: Notifications: "+ notifications.toString());
  continue;
}

// Se il client riesce a fare logout allora esce dal gruppo multicast
if(fromServer.equals(anObject:"LOGOUT: Success.")){
  System.out.println(fromServer);
  multicastSocket.close();
  endMulticast = true;
  System.out.println(x:"CLIENT: You are no longer in the group.");
  continue;
}
```

Comandi “show” e “logout”

Nel caso in cui si è in fase di gioco e il server risponde con la parola contenente gli indirizzi si stampa la parola a colori sul terminale.

```
// In fase di gioco: se ricevo una parola dal server nel
if(fromServer.length()==10){ // è una parola da colorare
  printColoredWord(toServer,fromServer);
  continue;
}else{ // altrimenti è un altro messaggio
  System.out.println(fromServer);
  continue;
}
```

Durante la fase di gioco

Se è un altro messaggio dal server lo si stampa per vedere quale messaggio è stato ricevuto.

Il metodo chiamato per colorare la parola con gli indizi è il seguente:

```
private static void printColoredWord(String guess, String s){
  for(int i=0; i<s.length();i++){
    if(s.charAt(i)=='-'){
      System.out.print(guess.charAt(i));
    }
    if(s.charAt(i)=='v'){
      System.out.print(COLOR_GREEN + guess.charAt(i) + COLOR_RESET);

    }
    if(s.charAt(i)=='g'){
      System.out.print(COLOR_YELLOW + guess.charAt(i) + COLOR_RESET);
    }
  }
  System.out.println(x:"\n");
}
```

Metodo che colora le parole con l'indizio

La parola ricevuta dal server è nel formato "vvgg—", ossia una parola di 10 caratteri dove un carattere può essere:

- v → lettera verde, lettera giusta nella posizione corretta

- g → lettera gialla, lettera giusta nella posizione sbagliata
- - → lettera che non appare nella parola segreta

Questo semplice metodo non fa altro che controllare carattere per carattere la parola inserita dall'utente `guess` e la stringa con l'indizio chiama `s` ricevuta dal server. A seconda delle lettere ine entrambe le parole stampa carattere per carattere la parola con le lettere colorate, come nel vero gioco di Wordle.

Una volta terminata la comunicazione con il server si chiude la socket con il comando `socket.close()`.

Classe ServerMain

Inizialmente questa classe legge il proprio file di configurazione, inizializza i socket per la comunicazione con il client e per il multicast. Istanzia la coda dei thread e la pool sulla quale gireranno i thread.

In seguito costruisce la mappa con il metodo `buildMap(path)` e istanzia un `ShutdownHook` per la terminazione tramite il `TerminationHandler`.

Dopodiché sceglie la parola dal vocabolario e inizia a rimanere in ascolto sulla porta in attesa che i client si connettano.

```java
// Per prmia cosa leggo il file di confgurazione
readConfig();

System.out.println(x:"SERVER: Server running...");

// Inizializzazione dei socket del server
ServerSocket serverSocket = new ServerSocket(port);
MulticastSocket multicastSocket = new MulticastSocket();

// Creazione threadpool con la coda dei thread
BlockingQueue<Runnable> queue = new LinkedBlockingQueue<Runnable>();
ExecutorService pool = new ThreadPoolExecutor(
nThread,
nThread,
terminationDelay,
TimeUnit.MILLISECONDS,
queue,
new ThreadPoolExecutor.AbortPolicy()
);

// Costruisco la map degli utenti leggendo dal file JSON
buildMap(path);

// Utilizzo handler di terminazione, ad esempio se terminassi il server con CTRL+C mi salva una mappa
// con tutti gli utenti e le caratteristiche nel file json "registeredUsers.json"
Runtime.getRuntime().addShutdownHook(new TerminationHandler(maxDelay, pool, serverSocket, map));

// Scelgo una nuova parola dal file delle parole
chooseNewWord();
```

Fase iniziale del server principale

Nel ciclo di ascolto semplicemente il server aspetta che arrivino delle richieste di connessione da parte dei client e per ogni connessione ricevuta si esegue nella pool un thread. Questo thread è un oggetto della classe `WordleGame` con i relativi parametri. Al termine del ciclo di ascolto si chiude la socket del server con `serverSocket.close()`.

```java
// Ciclo di ascolto dei client, gestito con una threadpool
while (true) {
  Socket socket = null;

  try {
    // Accetto le richieste dei client connessi
    socket = serverSocket.accept();
  } catch (SocketException e) {
    // Quando il TerminationHandler chiude la serverSocket, si solleva una SocketException ed esco dal ciclo
    break;
  }
  // Avvio un thread dalla pool per avviare il gioco per il client
  pool.execute(new WordleGame(socket, multicastSocket, map, secretWord));
}
serverSocket.close();
```

Ciclo d’ascolto del server

Il server per eseguire le sue operazioni usa questi metodi:

- `newWord()`: genera un numero casuale e cerca con l'oggetto `RandomAccesFile` una parola casuale all'interno del vocabolario.

```
private static String newWord() throws IOException{
  String newWord;
  String path = "../../Words/words.txt";
  RandomAccessFile raf = new RandomAccessFile(path,mode:"r");
  Random random = new Random();
  int n=1;
  int numElements = (((int) raf.length())-30824)/10; // Calcolo
  n = random.nextInt(numElements);
  raf.seek(n*11-11); // Vado ad una linea a caso nel file
  newWord = raf.readLine(); // Prendo la parola a quella linea
  System.out.println("SERVER: Secret word -> "+ newWord);
  raf.close();
  return newWord;
}
```

- `readConfig()`: metodo simile a quello utilizzato dalle altre classi per leggere il file di configurazione.
- `chooseNewWord()`: metodo che, allo scadere del timer del cambio della parola, chiama il metodo `newWord()`. Così facendo periodicamente viene cambiata la parola segreta.

```
private static void chooseNewWord(){
  // Scelta della nuova parola da trovare, il
  Timer timer = new Timer();
  timer.scheduleAtFixedRate(new TimerTask(){
    public void run() {
      try{
        String word = newWord();
        secretWord.set(word);
        resetPlayers(); // Al cambio della pa

      }catch(Exception e){
        e.printStackTrace();
      }
    }
  } , delay:0, timeBetweenWords);
}
```

- `resetPlayers()`: metodo che resetta il campo `canPlay` degli utenti loggati. Questo metodo è chiamato al momento in cui la parola cambia. Serve per dare la possibilità di giocare nuovamente ai giocatori che hanno fatto il login e hanno giocato la parola precedente.

```
private static void resetPlayers(){
  for(Map.Entry<String,User> entry : map.entrySet()) {
    User user = entry.getValue();
    user.setCanPlay(canPlay:true);
  }
}
```

- `buildMap()`: metodo che serve a costruire la `ConcurrentHashMap` che contiene gli utenti registrati al sistema. Per prima cosa trasforma il file registeredUsers.json in una stringa, successivamente controlla che non sia vuota a `null` (in questo caso inizializza una mappa vuota). Nel caso in cui ci sia già all'interno del file una mappa già ben formata, inizializza la map con i dati che trova nel file registeredUsers.json.

```
private static void buildMap(String path) throws Exception{
  String jsonString = new String(Files.readAllBytes(Paths.get(path)));
  if(jsonString.equals(anObject:"") || jsonString.equals(anObject:"null")){
      map = new ConcurrentHashMap<String, User>();
  }else{
      map = new Gson().fromJson(jsonString, new TypeToken<ConcurrentHashMap<String, User>>() {}.getType());
      // System.out.println("SERVER: Current map -> " + map);
  }
}
```

Classe WordleGame

Classe che viene istanziata dal server al momento in cui un client vuole connettersi con esso. Appena il server accetta la richiesta avvia un thread istanziato come oggetto di questa classe. La classe implementa l'interfaccia `Runnable`.

Questa classe si occupa dell'interazione fra un utente (client) e il server, riceve i comandi e li elabora per costruire le risposte, gestisce la fase di gioco e il calcolo delle statistiche di ogni giocatore.

Il costruttore di WordleGame prende 4 parametri in input che sono passati dal server: il socket relativo al client, il socket relativo al multicast, la map degli utenti e la parola segreta.

```java
public WordleGame(Socket clientSocket, MulticastSocket multicastSocket,
ConcurrentHashMap<String, User> map, AtomicReference<String> secretWord) {
    this.clientSocket = clientSocket;
    this.multicastSocket = multicastSocket;
    this.map = map;
    WordleGame.secretWord = secretWord;
}
```

Il metodo principale di questa classe è il metodo `run()`, metodo che viene subito eseguito alla chiamata del thread relativo a questa classe.

Per prima cosa all'interno di questo metodo viene letto il file di configurazione con `readConfig()`, vengono istanziate le variabili per la comunicazione con il client e il gruppo multicast.

```java
readConfig();

// Il server inizializza il gruppo multicast
group = InetAddress.getByName(multicastHost);

// Stream per la comunicazione con il client che si è connesso
PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), autoFlush:true);
Scanner in = new Scanner(clientSocket.getInputStream());
String toClient="";
String fromClient="";
```

Da qui inizia il ciclo di comunicazione con il client, all'interno di questo ciclo vi è uno `switch` che legge il comando ricevuto dal client. A seconda del comando esegue le operazioni che il client desidera fare.

Le operazioni disponibili sono:

- Register: con questo comando l'utente desidera registrarsi al sistema, come prima cosa si chiede al client di fornire un nuovo username (non deve essere vuoto), dopodiché se non è presente nella mappa, si chiede di fornire una nuova password. Nel caso in cui sia già presente nella mappa si avvisa ala client di loggare con quell'account.

```
case "register": // L'utente desidera registrarsi

    boolean userExistsRegister = false;
    out.println(x:"Register: Insert username:");
    // Chiede all'utente con quale username vorrebbe registrarsi
    String reg_username = in.nextLine();

    // Non si può registrare un utente senza nome
    if(reg_username.equals(anObject:"")){
        out.println(x:"REGISTER: Please enter a valid userneame.");
        break start;
    }

    // Se viene inserito un username valido ontrollo in tutta la map che l'utente ir
    // Se l'utente è già registrato allora si torna all'inizio dello switch, comunic
    for(Map.Entry<String,User> entry : map.entrySet()) {
        User user = entry.getValue();
        if(user.getUsername().equalsIgnoreCase(reg_username)){
            out.println(x:"REGISTER: Username is already registered. Try again.");
            userExistsRegister = true;
            break start;
        }
    }

    // Se l'utente è nuovo allora posso registrarlo, si richiede una password da ass
    // poi si aggiunge il nuovo utente alla mappa
    if(!userExistsRegister){
        out.println(x:"REGISTER: Insert password: ");
        String reg_password = in.nextLine();
        User user = new User(reg_username,reg_password);
        map.put(reg_username, user);
    }

    // Togliere il commento al comando sottostante per stampare la mappa aggiornata
    // System.out.println("SERVER: Updated map -> " + map);
    out.println(x:"SERVER: Enter new command.");

break start;
```

Registrazione

- Login: con questo comando il client vuole fare login, la logica alla base è la stessa del comando di registrazione, con la differenza che un utente loggato fa anche parte del gruppo multicast e può iniziare a giocare a Wordle. Il login è molto semplice, si controlla che l'utente sia nella map degli utenti registrati (manda un messaggio di errore se l'utente è già loggato), si richiede la password che deve coincidere con quella salvata nella map e si settano i campi `canPlay` e `isLogged` a `true` dello user.

```java
if(currentUser==null || (currentUser!=null && !currentUser.getIsLogged())){
    out.println("LOGIN: Username: ");
    String log_username = in.nextLine();
    boolean userExistsLogin = false;

    for(Map.Entry<String,User> entry : map.entrySet()) {
        User user = entry.getValue();
        if(user.getUsername().equalsIgnoreCase(log_username) && user.getIsLogged()){
            out.println("LOGIN: Username is already logged. Try again.");
            userExistsLogin = true;
            break start;
        }
        if(user.getUsername().equalsIgnoreCase(log_username) && !userExistsLogin){
            userExistsLogin = true;
            out.println("LOGIN: Password:");
            String log_password = in.nextLine();

            if(user.getPassword().equalsIgnoreCase(log_password)){
                out.println("LOGIN: Login Successful.");
                currentUser = user;
                currentUser.setIsLogged(true);
                currentUser.setCanPlay(true);

                break start;
            }else{
                out.println("LOGIN: Invalid password, login again.");
                break start;
            }
        }
    }

    if(!userExistsLogin){
        out.println("LOGIN: User not existing, register first.");
        break start;
    }
    // out.println("SERVER: Enter new command.");
    break start;

} else { // Se dopo un login il client volesse loggarsi nuovamente
    out.println("LOGIN: Already logged.");
    break start;
}
```

Login

- Play: terzo comando della lista, qui l'utente desidera giocare a Wordle. L'utente deve essere loggato e il campo `canPlay` deve essere `true`. Per ogni "guess" dell'utente si controlla che sia valida la parola (esistente nel dizionario) e da qui si costruisce la stringa nel formato "vvgg—" contenente gli indizi da mandare in risposta al client. Il ciclo di controllo viene effettuato 12 volte, dato che è il numero massimo di tentativi concesso per indovinare la parola.

Una volta terminata la partita la classe chiama tutti metodi dello User relativi al calcolo delle sue statistiche.

Nel caso in cui un utente abbia già giocato la parola segreta oppure non è loggato, il server lo comunica con un messaggio di risposta.

```
// Fase effettiva di gioco: i tentativi massimi sono 12.
while(!wordGuessed && currentUser.getTries()<12){
    // Leggo che parola ha inviato e controllo
    guess = in.nextLine();

    if(isValid(guess)){ // La parola inserita è valida (presente nel file)
        if (guess.toLowerCase().equals(secretWord.toString())){ // Ha indovinato
            currentUser.setTries(currentUser.getTries()+1);
            toClient = "vvvvvvvvvv"; // Parola tutta verde
            out.println(toClient);
            wordGuessed=true;
            continue;
        }else{ // parola valida (presente nel file) ma scorretta
            result=buildResult(guess); // result è la stringa nel formato "vvgg--"
            //System.out.println(result);
            toClient=result;
            out.println(toClient);
            currentUser.setTries(currentUser.getTries()+1);
            continue;
        }
    }else{ // Parola inserita non valida (non presente nel file), non conta come t
        toClient="Invalid word, retry";
        out.println(toClient);
    }
}
```

Fase di gioco

```
currentUser.increaseGamesPlayed();
if(wordGuessed){
    currentUser.increaseStreak();
    currentUser.calculateWS();
    currentUser.increaseTotalWins();
    currentUser.setHasGuessed(result:true);
}else{
    // ha perso quindi resetto la streak
    currentUser.setHasGuessed(result:false);
    currentUser.calculateWS();
    currentUser.setInStreak(inStreak:false);
    currentUser.setStreak(streak:0);
}
currentUser.addTriesToList(currentUser.getTries());
currentUser.setLastTries(currentUser.getTries());
currentUser.setTries(tries:0);
currentUser.calculateGuessDistribution();
currentUser.calculateWinRate();
currentUser.setIsPlaying(isPlaying:false);
currentUser.setCanPlay(canPlay:false);
```

Calcolo delle statistiche post partita

- Logout: quarto comando della lista, qui l'utente desidera fare logout, quindi i campi `isLogged` e `canPlay` dell'utente vengono resettati a `false`.

```
case "logout": // L'utente desidera fare logout
/**
 * Per poter fare logout ovviamente un utente deve essere
 * Si settano i campi isLogged e canPlay a false dell'uter
 */
    if(currentUser!=null && currentUser.getIsLogged()){

        currentUser.setIsLogged(isLogged:false);
        currentUser.setCanPlay(canPlay:false);

        out.println(x:"LOGOUT: Success.");

        break start;
    }else{
        out.println(x:"LOGOUT: You have to login first");
    }
break start;
```

- Send me statistics: quinto comando della lista, a questo punto l'utente vuole visualizzare le sue statistiche relative al gioco da quando si è registrato. Il server quindi manda un messaggio in risposta al client con le sue statistiche.

```java
case "stats", "send stats", "send me statistics", "send me stats": // Un utente desidera veri
    /**
     * Un utente per poter vedere le sue statistiche deve ssere loggato ed aver almeno fatto
     * Il messaggio del server contiene tutte le statistiche relative a quell'utente.
     */
    if(currentUser != null && currentUser.getIsLogged() && currentUser.getGamesPlayed()!=0){
        out.println("STATISTICS: "+currentUser.getUsername()
        + ", Win Rate: "+ String.format(format:"%.2f",currentUser.getWinRate())
        + ", Last winstreak: " + currentUser.getLastWS()
        + ", Max winstreak: " + currentUser.getMaxWS()
        + ", Guess distribution: "
        + String.format(format:"%.2f",currentUser.getGuessDistribution())+".");
    }else{
        out.println(x:"ERROR: Cannot send statistics, you have to play one game first.");
    }
break;
```

- Share: sesto comando della lista, il client dopo aver immesso questo comando vuole condividere nel gruppo multicast il risultato della sua ultima partita. Per poterlo fare deve intanto essere loggato e deve aver almeno giocato una partita.

  Il messaggio inviato conterrà il risultato della sua ultima partita, quindi se ha vinto o perso e con quanti tentativi.

  Quest'ultimo verrà messo in un `byteBuffer` e spedito come datagramma UDP nel gruppo.

```java
case "share": // L'utente desidera condividere le proprie statistiche (dell'ultima partita fatta) nel gruppo multicast
    /**
     * Come per il comando precedente, un utente può condividere le proprie statistiche solo se è loggato e
     * ha giocato almeno una partita da quando si è registrato.
     * Questa procedura una volta che costruisce il mesasggio in base all'esito dell'ultima partita, lo invia nel gruppo
     */
    String result ="";
    if(currentUser != null && currentUser.getGamesPlayed()!=0 && currentUser.getIsLogged()){
        System.out.println("WORDLE: Sharing "+ currentUser.getUsername() +" last game stats...");
        if(currentUser.getHasGuessed()){
            result="won";
        }else{
            result="lost";
        }
        String messageUDP = "User: "+ currentUser.getUsername()
                            + " has " + result
                            + " with "+ (currentUser.getLastTries())
                            + " tries.";
        byte[] buf = messageUDP.getBytes();
        DatagramPacket shareResults= new DatagramPacket(buf, buf.length, group, multicastPort);
        multicastSocket.send(shareResults);
        out.println(x:"SHARE: Result sent succesfully.");
        // System.out.println("SERVER: Result sent.");
        break start;
    }else{
        out.println(x:"SHARE ERROR: Can't send statistics. You are not logged in or you have never played before.");
        break start;
    }
```

- Show me sharing: settimo comando della lista, con questo comando il client vuole visualizzare tutte le condivisioni degli altri utenti inviate nel gruppo multicast.

Dato che l'implementazione di questo comando è stata fatta lato client, il server si limita a mandare un messaggio di conferma o di errore. Il messaggio di errore è mandato nel caso in cui l'utente non è loggato.

```
case "show", "show me sharing": // L'utente desidera vedere le statistiche condivise dagl
    /**
     * Come per i comandi precedenti, l'utente deve essere loggato per poter vedere le st
     * In caso contrario invia un messaggio di errore.
     */
    if(currentUser != null && currentUser.getIsLogged()){
        out.println(x:"SHOW ME SHARING: Request sent.");
        break start;
    }else{
        out.println(x:"SHOW ME SHARING ERROR: Request error, you have to login first.");
        break start;
    }
```

- Exit: ultimo comando della lista. Questo è uno dei modi in cui un client interrompe la comunicazione con il server. In questo caso vengono effettuati i reset a `false` delle variabili `canPlay` e `isLogged`, ciò viene fatto nel caso in cui l'utente non fa logout prima di uscire. A questo punto si va a chiudere la socket del client con il comando `clientSocket.close()`.

```
case "exit": // L'utente vuole disconnetersi dal server ed
    /**
     * Questa procedura setta i campi dell'utente di login
     * Ciò serve nel caso in cui l'utente esce senza fare l
     * In seguito si chiude la socket di comunicazione con
     */
    System.out.println("SERVER WORDLE: Client "
    + clientSocket.getInetAddress()+" just disconnected");
    currentUser.setCanPlay(canPlay:false);
    currentUser.setIsLogged(isLogged:false);
    clientSocket.close();

break start;
```

I metodi che implementa questa classe sono:

- `buildResult(String guess)`: in questo metodo si confronta la "guess" del client con la parola segreta da indovinare. Carattere per carattere i confrontano le parole e si costruisce la stringa nel formato "vvgg—" e si restituisce.

```java
private static String buildResult(String guess){
    StringBuilder result = new StringBuilder(guess);
    String word = secretWord.toString();
    //System.out.println(result);
    String cguess = null;
    String cword = null;
    int i=0;
    int j=0;

    while(i<guess.length()){
        cguess = guess.substring(i, i+1);
        cword = word.substring(j, j+1);
        if(word.contains(cguess)){
            result.setCharAt(i, ch:'g');
        }
        if(cguess.equals(cword)){
            result.setCharAt(i, ch:'v');
        }
        if(!cguess.equals(cword) && !word.contains(cguess)){
            result.setCharAt(i, ch:'-');
        }
        i++;
        j++;
    }
    return result.toString();
}
```

- `isValid(String guess)`: metodo che controlla se la parola inserita dall'utente è valida o meno. Per farlo usa la ricerca binaria nel file di test words.txt, la ricerca è implementata nel metodo `binarySearch(RandomAccessFile raf, String key)` subito sotto.

```java
private static boolean isValid(String guess){
    try (RandomAccessFile raf = new RandomAccessFile(name:"../../Words/words.txt", mode:"r")) {
    // Search key
        int pos = binarySearch(raf, guess);

        // Return true or false
        if (pos != -1) return true;
        else return false;

        }catch (Exception e) {
            e.printStackTrace();
            System.exit(status:1);
        }
    return false;
}
```

```java
private static int binarySearch(RandomAccessFile raf, String key) throws Exception{

    int numElements = (((int) raf.length())-30824)/10; // number of elements
    int lower = 0, upper = numElements, mid=0; // indexes for search
    int index;
    String w="";

    while (lower <= upper) {
        mid = (lower + upper) / 2;
        index= mid*11-11;
        if(index>0)
            raf.seek(index);
        else return -1;
            w = raf.readLine();
        int comparison = w.compareTo(key);

        if (comparison == 0){
            // found it
            return (index);
        }
        else if (comparison < 0){
            // w comes after key
            lower = mid + 1;
        }
        else {
            // w comes before key
            upper = mid - 1;
        }
    }
    return -1;
}
```

- i metodi `readConfig()` e `readFileAsString()` sono i medesimi che sono implementati anche nelle classi `ClientMain` e `ServerMain`.

Classe TerminationHandler

All'interno di questa classe viene gestita la terminazione del server (ad esempio se viene effettuata con ^C).

Nel costruttore di questa classe vengono richiesti i seguenti parametri:

- `maxDelay`: tempo massimo in cui la pool può stare in attesa che tutti i thread escano da essa.
- `pool`
- `serverSocket`
- `map`

Anch'essa implementa l'interfaccia `Runnable` quindi ha il suo metodo `run()`: per prima cosa si chiude la `serverSocket` e si prova a spegnere la pool. Nel caso in cui ci siano ancora dei thread al suo interno si aspetta fino al massimo di `maxDelay` per spegnerla forzatamente.
Nuovamente nel caso in cui i client non abbiano fatto un logout manuale o sono usciti senza usare il comando "exit" si resettano a false i campi `isLogged` e `canPlay` di tutti gli utenti.
Come ultima cosa si sovrascrive con i nuovi dati il file registeredUsers.json così da non perdere questi dati.

```
public void run() {

    System.out.println(x:"SERVER TERMINATION: Termination...");

    try {
        // Prima di tutto chiude la socket del server almeno non accetta più nuove richieste
        serverSocket.close();
        // Spegne la pool se è vuota oppure aspetta che si svuota fino ad un massimo delay, d
        pool.shutdown();
        if(!pool.awaitTermination(maxDelay, TimeUnit.MILLISECONDS)){
            pool.shutdownNow();
        }
        System.out.println(x:"SERVER TERMINATION: Server shut down.");

        // Resetto i campi degli user che sono usciti senza aver fatto logout o exit
        for(Map.Entry<String,User> entry : map.entrySet()){
            User user = entry.getValue();
            user.setCanPlay(canPlay:false);
            user.setIsLogged(isLogged:false);
        }

        // Sovrascrittura della mappa nel file "registeredUsers.json"
        FileWriter fileWriter = new FileWriter(fileName:"registeredUsers.json");
        // System.out.println("SERVER TERMINATION: map after termination: " + map);
        Gson gson = new GsonBuilder().setPrettyPrinting().create();
        gson.toJson(map, fileWriter);
        fileWriter.close();
    }catch (Exception e) {
        e.printStackTrace();
    }
}
```

Classe User

Questa è la classe che racchiude tutti i campi relativi ad un utente. Dall'username alla password a tutta una serie di campi per controlli e calcolo delle statistiche.

Il costruttore chiede solo di fornire `username` e `password`, il resto dei campi sono settati a `0` di default al momento della creazione di un nuovo utente.

```java
public User (String username, String password) {
  this.username=username;
  this.password=password;
  this.gamesPlayed=0;
  this.streak=0;
  this.winRate=0;
  this.totalWins=0;
  this.triesInGame=0;
  this.lastws=0;
  this.maxws=0;
  this.winRate=0;
  this.guessDistribution=0;
  this.triesList= new ArrayList<Integer>();
}
```

Questi sono i campi che andranno ad essere salvati nella lista degli utenti, campi utili per controlli e anche per la visualizzazione e il calcolo delle statistiche.

Subito sotto al costruttore vi sono tutti i *setters* e *getters* per i campi privati dell'utente.

In seguito si trovano tutti metodi per il calcolo delle statistiche come ad esempio il calcolo del `winRate` e della `guessDistribution`.

```java
public void calculateWinRate(){
  double wr = (double)totalWins/(double)gamesPlayed * 100;
  this.setWinRate(wr);
}

public double calculateGuessDistribution(){
  double sum=0;
  for(int n : triesList){
    sum+=n;
  }
  this.setGuessDistribution(sum/(double)triesList.size());
  return this.guessDistribution;
}
```

Istruzioni per la compilazione ed esecuzione

Nella cartella `Compilazione` si trova il file compilazione.txt contenente le direttive per la compilazione e l'esecuzione del progetto.

Queste direttive permettono di compilare ed eseguire il programma utilizzando la libreria GSON. In particolare bisogna prima entrare da terminale nella cartella `src` e copiare la direttiva a linea 4. Una volta che la compilazione ha avuto successo bisogna entrare della cartella `bin` (sempre da terminale) e copiare le direttive successive.

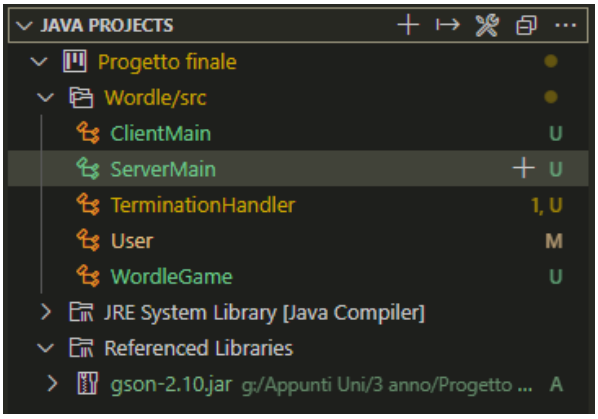
Se ci si trova in un sistema Windows allora bisogna eseguire prima la direttiva a linea 10 per avviare il server e successivamente (in un altro terminale) la direttiva a linea 11 per avviare il client.

Per un sistema con macOS il procedimento è lo stesso ma con le direttive a linea 15 e 16.

```
1  COMPILAZIONE CON GSON:
2
3  (Dalla cartella src)
4  javac -cp "./../lib/gson-2.10.jar" ./*.java -d ./../bin
5
6  RUNNARE CON GSON:
7
8  Windows:
9  (Dalla cartella bin)
10 java -cp ".;./../lib/gson-2.10.jar" ServerMain
11 java -cp ".;./../lib/gson-2.10.jar" ClientMain
12
13 macOS:
14 (Dalla cartella bin)
15 java -cp ".:./../lib/gson-2.10.jar" ServerMain
16 java -cp ".:./../lib/gson-2.10.jar" ClientMain
17
18 Esecuzione jar:
19 java -jar Server.jar
20 java -jar Client.jar
21
```

compilazione.txt

Da questo pannello è possibile creare i file `.jar` delle classi principali, ossia `Client.jar` e `Server.jar`.



Pannello creazione file jar

Cliccando sulla "↦" in alto è possibile creare i `jar`, facendo attenzione a includere la libreria gson-2.10 quando lo si va a creare.
A riga 19 e 20 ci sono i comandi per eseguire questi file.