



# UNIVERSITÀ DI PISA

**LABORATORIO DI RETI**

**UNIVERSITY of PISA**

**WORDLE A WORD GAME 3.0**

By

Neri Sansone - 621583

Department of Computer Science

UNIVERSITY of PISA

October 16, 2023

# CONTENTS

|          |                                      |           |
|----------|--------------------------------------|-----------|
| <b>1</b> | <b>Introduction</b>                  | <b>2</b>  |
| <b>2</b> | <b>Project Structure</b>             | <b>3</b>  |
| 2.1      | Introduction .....                   | 3         |
| 2.1.1    | Src .....                            | 3         |
| 2.1.2    | Lib.....                             | 4         |
| 2.1.3    | Bin .....                            | 4         |
| 2.2      | WordleClient .....                   | 4         |
| 2.3      | WordleServer .....                   | 5         |
| 2.4      | MenuHandler .....                    | 5         |
| <b>3</b> | <b>Implementation choices</b>        | <b>6</b>  |
| 3.1      | User Experience.....                 | 6         |
| 3.2      | Edge Cases.....                      | 7         |
| <b>4</b> | <b>Data Structures</b>               | <b>8</b>  |
| <b>5</b> | <b>Conclusion and Future Outlook</b> | <b>9</b>  |
| 5.1      | Conclusion .....                     | 9         |
|          | <b>References</b>                    | <b>10</b> |

# 1 INTRODUCTION

Wordle 3.0 is a simplified version of the already known game. It is based on a server-client mechanism which allows different users to play at the same time. After connecting to the server the users will be able to play trying to guess a ten letters long word within twelve tries using the given hints.

## 2 PROJECT STRUCTURE

### 2.1 Introduction

The general structure relies on a server - client paradigm. Every user can access and interact with the game via CLI (Command Line Interface). The client sends the requests to the server which will process and answer to every one of them. The projects is divided in three folders:

- src
- lib
- bin

Firstly I will give a short overview of every folder, then I will bring you to a more deep understanding of the core functions of the project.

#### 2.1.1 Src

In the **src** folder we find the seven different *.java* files:

- **WordleClient.java**
- **WordleServer.java**
- **menuHandler.java**
- **User.java**
- **th\_properties.java**
- **Mess\_Receiver.java**
- **Change\_Word.java**

### 2.1.2 Lib

This folder contains the only referenced library used in this project: *gson2.10.jar*

### 2.1.3 Bin

The **bin** folder contains all the *.class* files produced after having compiled, the *.properties* files used by the client and the server and the *users.json* file along with the file containing all the possible words.

## 2.2 WordleClient

The **WordleClient** class is responsible of interacting with the users via CLI (*Command Line Interface*). It immediately reads the *client\_config.properties* file in order to connect to the server. After having opened the socket and the two streams for the *input* and *output* we get to the core of the program.

The client will print in the terminal the first menu in which is possible choose between different options <sup>1</sup>:

- **REGISTER [1]** → if chosen will ask for an username and a password (*n.b* no empty username and/or password are allowed)
- **LOGIN [2]** → if chosen will lead to the login
- **EXIT [3]** → if chosen it will close the current session

Whether a registration is successful or not it will lead to the first menu again. If the user chooses to log in and such operation is successful the second menu will be printed. We are now able to choose between five options:

- **PLAY [1]** → if the word has not been played by the user it will start the game
- **STATISTICS [2]** → will show the statistics of the logged user
- **SHARE [3]** → will broadcast a message saying how many attempts the user needed to guess the word

---

<sup>1</sup>*n.b* every choice made is sent to the server

- **SHOW ME SHARING [4]** → will show the received messages
- **LOGOUT [5]** → will lead to the first menu while logging out

Concerning the client side the game is managed by the *WordleGame* function which is responsible for sending the entered guess to the server in order to receive back the possible hints and check if the user has won. This function returns the number of attempts used by the user to guess the word, this value will be used by the server for the sharing part.

## 2.3 WordleServer

The server is the very first class that we need to execute. Like the client it immediately reads the *server\_config.properties*. It will then create a an ArrayList object containing all the user in the .json file if it exists, otherwise it will create a new users.json file. Afterwards it creates another ArrayList structure with all the given words along with a thread that is responsible of extracting the secret word that will be changed every two minutes (I choose two minutes in order to have a more loyal representation of the game without waiting for twenty-four hours). With the two ArrayList objects created the server also initialise the *th\_properties* object (called *properties*), this object contains the user list, the word list and the secret word. Eventually with a try-catch block it creates a menuHandler thread for every connection accepted.

## 2.4 MenuHandler

The ***menuHandler*** class is responsible of managing every single request sent by the client and checking if the operations that the user is doing are legal (*i.e.* checking during the registration if the chosen username is available or not). It is also responsible: of adding the new users to the *users.json* file, of creating the hints for every guess received together with the update of the statistics eventually it is also responsible of sending the user' statistics as well as the broadcast message(s).

## 3 IMPLEMENTATION CHOICES

### 3.1 User Experience

As soon as a new user starts the client he or she will be required to register and then afterwards to login. While registering the system will firstly ask to enter a valid username (*i.e.* no empty or already used username can be chosen) and secondly to enter a valid password (*i.e.* an empty password is **not** valid).

```
-----  
REGISTER [1]  
LOGIN [2]  
EXIT [3]  
  
1  
----[ ENTER USERNAME ]----  
  
----[ USERNAME CAN NOT BE EMPTY ]----  
---[ INSERT A DIFFERENT USERNAME: ]---  
Mario  
----[ ENTER PASSWORD ]----  
Rossi  
Registration successful  
-----
```

Figure 3.1: Registration

```
-----  
REGISTER [1]  
LOGIN [2]  
EXIT [3]  
  
2  
Enter username:  
Mario  
Enter password:  
Rossi  
  
Login successful  
-----
```

Figure 3.2: Log in page

An user is **not** able to log in simultaneously onto two different clients in fact there is a variable inside the *User* class that checks if an user is already logged in. After logging in the user will be able to choose one of the options listed in section 2.2. If the user has not guessed the current secret word yet, he will be able to play <sup>1</sup>. After playing he will be allowed to share the number of attempts he needed to guess the word. In order to implement such feature I added a variable that shows if the user has already played the word or not.

---

<sup>1</sup>If the guessed word contains three Cs but the secret word contains two Cs then the hints will show three "?", which means that the hints system I built does **not** count the occurrences of a letter.

```

-----
PLAY [1]
STATISTICS [2]
SHARE [3]
SHOW ME SHARING [4]
LOGOUT [5]

1
----[ GAME STARTING ]----
----[ ENTER GUESS: ]----
abalienate
?_?_??_a_?
----[ ENTER GUESS: ]----
heresimach
----[ YOU WIN ]----
-----

```

**Figure 3.3:** Game play

It is always possible to see the statistics and the received message(s) through the *SHOW ME SHARING* option. The sharing side relies on a UDP connection and with the multicast sends the message to all the connected clients.

## 3.2 Edge Cases

- If it happens that the client crashes unexpectedly in whatsoever moment of the session, no problems: the user will just need to start again a new client.
- If the server crashes unexpectedly, then all the connected clients, as soon as they try to communicate with the server, will be closed and it will be necessary to launch again the server.



## 4 DATA STRUCTURES

- **words\_list** → it is an `ArrayList<String>` object that contains all the words provided in the `words.txt` file.
- **users\_list** → it is an `ArrayList<User>` object that contains all the users that are inside `users.json`
- **User** → it collects all the relevant details of an user such as the *username*, the *password* and a `HashMap<Integer,Integer>` that shows how many words have been guessed at a certain try.

```
"username": "Mario",
"password": "Rossi",
"play_this_word": false,
"games_played": 1,
"games_won": 1,
"victory_rate": 100,
"victory_streak": 1,
"max_victory_streak": 1,
"logged": true,
"guess_distribution": {
  "1": 0,
  "2": 1,
  "3": 0,
  "4": 0,
  "5": 0,
  "6": 0,
  "7": 0,
  "8": 0,
  "9": 0,
  "10": 0,
  "11": 0,
  "12": 0
}
```

**Figure 4.1:** Mario Rossi user's data structure

- **repository** → it is an `ArrayList<String>` object that saves every message received via UDP.

I choose to use `ArrayList` structures and not built-in array because `ArrayList` is easier to manipulate (its size can be modified without any problem).

## **5 CONCLUSION AND FUTURE OUTLOOK**

### **5.1 Conclusion**

## **BIBLIOGRAPHY**