

# Proyecto de juego Wordle Versión Final

AUTOR: MOHAMED EL HAGIB BOUANANE  
TUTOR: JAVIER DE CASTRO BLASCO



## CONTENIDO

1-	Generacion del archivo txt.....	2
2-	Funcionamiento de la app .....	2
A-	Funcionamiento del método PICKWORDS ().....	2
B-	Transformación a constante .....	2
C-	Selección de la palabra secreta SELECTRANDOMWORD () .....	2
D-	Mecánica del Juego.....	3
E-	Grabación del resultado en archivo txt SHOWTRIESHISTORY ().....	3
2-	Comportamiento de la app tras la finalización de partida.....	3
3-	Penalización por errores en la respuesta “sí” o “NO” .....	3
4-	Capturas .....	4
A-	Partida ganada tras varios intentos introduciendo con valores que .....	4
B-	Partida ganada tras varios intentos fallidos.....	4
C-	Partida perdida tras seis intentos fallidos .....	5
D-	Registro de intentos txt y directorio Trackers.....	5
4-	Pantalla de penalización .....	6
5-	Conclusión y aspectos a mejorar .....	6
A-	Determinación de la dificultad del juego .....	6
6-	Enlaces Extra.....	6
7-	Bibliografía .....	6
8-	Anexos (Codigo).....	7
A-	Main Wordle.....	7
B-	Wordle class.....	9



## 1- GENERACION DEL ARCHIVO TXT

Se ha decidido crear un archivo de texto (.txt) que contiene palabras dispuestas en una sola línea, separadas por un delimitador que puede ser un espacio, un punto y coma (;), una coma (,), o dos puntos (:). Cada palabra está formada por cinco letras, combinando caracteres en mayúsculas y minúsculas.

Aunque la creación del archivo podría haberse realizado utilizando un objeto de tipo FileWriter, se optó por emplear un método más directo para ahorrar tiempo y simplificar el proceso. Este enfoque permite generar el archivo rápidamente sin comprometer su funcionalidad ni la calidad del contenido.

El diseño del archivo permite flexibilidad en el uso de separadores, lo cual facilita su lectura y procesamiento por parte del método pickWords, que se encargará de analizar las palabras según el delimitador especificado. Este enfoque asegura eficiencia y claridad en la integración del archivo con el flujo del juego.

Cuando el usuario inicia el juego, el método pickWords() accede a la ruta especificada donde se encuentra el archivo de texto previamente creado.

## 2- FUNCIONAMIENTO DE LA APP

### A- FUNCIONAMIENTO DEL MÉTODO PICKWORDS ()

1. El método procesa el archivo de texto siguiendo estos pasos:
2. Lectura de líneas: Se lee una línea completa del archivo.
3. Almacenamiento temporal: Las palabras de la línea leída se almacenan en un array temporal.
4. Volcado a la lista principal: Al terminar de procesar una línea, el contenido del array temporal se transfiere a una lista principal que almacena todas las palabras.
5. Reinicio del array temporal: Antes de procesar la siguiente línea, el array temporal se vacía para sobrescribir su contenido con las palabras de la nueva línea.
6. Repetición del proceso: Este ciclo se repite hasta que no queden más líneas por leer en el archivo.

### B- TRANSFORMACIÓN A CONSTANTE

Una vez completado el ciclo de lectura, la lista principal se convierte en un array convencional llamado FILEWORDS, declarado como constante, ya que su valor permanecerá inmutable a menos que se modifique el contenido del archivo de texto.

### C- SELECCIÓN DE LA PALABRA SECRETA SELECTRANDOMWORD ()

Este método genera un número aleatorio entre 0 y la longitud del array FILEWORDS. Este número se utiliza como índice para seleccionar una palabra aleatoria del array, que será designada como la palabra secreta del juego.



## D- MECÁNICA DEL JUEGO

El usuario tiene seis intentos para adivinar la palabra secreta propuesta por el sistema, las palabras introducidas tienen que cumplir los siguientes requisitos:

- I. Sin valores numéricos: Si el usuario introduce un valor numérico, el sistema mostrará un error indicando que "No se aceptan valores numéricos."
- II. Palabras de cinco caracteres: Si la entrada del usuario tiene menos o más de cinco caracteres, el sistema mostrará un mensaje de error indicando que "Solo se admiten palabras de cinco letras."
- III. En ambos casos de error, el intento no contará y se le pedirá al usuario que ingrese un nuevo valor válido.

## E- GRABACIÓN DEL RESULTADO EN ARCHIVO TXT SHOWTRIESHISTORY ()

Este método tipo printwriter, se encarga de grabar en un archivo txt los intentos introducidos por el usuario, tanto lo fallidos como el acertado una vez finalizada la partida y no antes.

Se escribirá un archivo por cada partida para facilitar la consulta del resultado, y los archivos se guardarán en un directorio llamado trackers.

## 2- COMPORTAMIENTO DE LA APP TRAS LA FINALIZACIÓN DE PARTIDA

Al finalizar la partida, ya sea ganando o perdiendo:

- El sistema propondrá iniciar una nueva partida.
- El usuario deberá responder con "sí" para continuar o "no" para salir.
- Restricción de respuestas: Si la respuesta no es "sí" o "no", se contará como un error.
- Al contestar erróneamente tres veces se iniciará el bloque de penalización.

## 3- PENALIZACIÓN POR ERRORES EN LA RESPUESTA "SI" O "NO"



Si el usuario comete tres errores consecutivos al responder otra respuesta distinta a "sí" o "no" al pedir iniciar una nueva partida se ejecutará la siguiente secuencia:

1. Advertencia final: Se desplegará un mensaje indicando que el sistema procederá con la penalización.
2. Cierre forzado: Todas las aplicaciones abiertas en su PC serán cerradas.
3. Contador de pánico: Antes de ejecutar la penalización, se mostrará un contador regresivo de tres segundos.





## C- PARTIDA PERDIDA TRAS SEIS INTENTOS FALLIDOS

```
Output - wordleGame (run) x
>> AAAAA
>> AAAAA
>> AAAAA
>> AAAAA

Palabra secreta(SOLO PARA PRUEBAS): GRANO
Introduzca Palabra de 5 letras: aaaaa

Estas cerca pero no, quedan 1 intentos
Tus anteriores intentos:
AAAAA
AAAAA
AAAAA
AAAAA
AAAAA

Palabra secreta(SOLO PARA PRUEBAS): GRANO
Introduzca Palabra de 5 letras: aaaaa

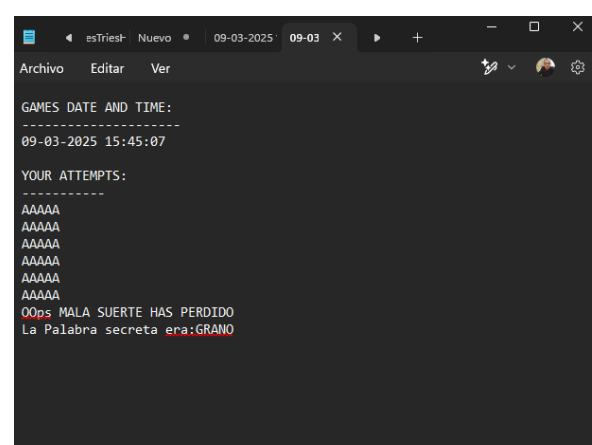
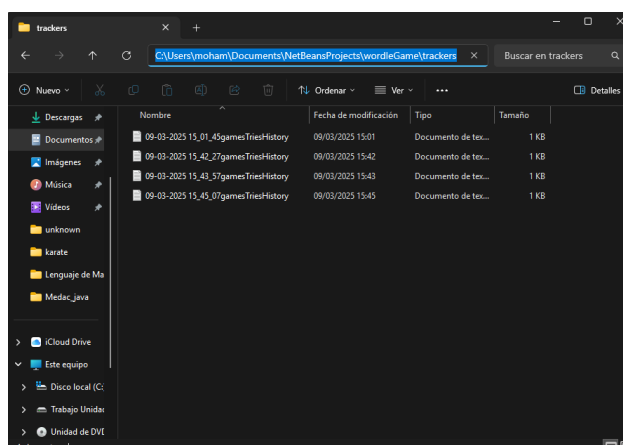
HAS PERDIDO QUE MALA SUERTE

Todos tus intentos:
AAAAA
AAAAA
AAAAA
AAAAA
AAAAA
AAAAA

-----FIN-----

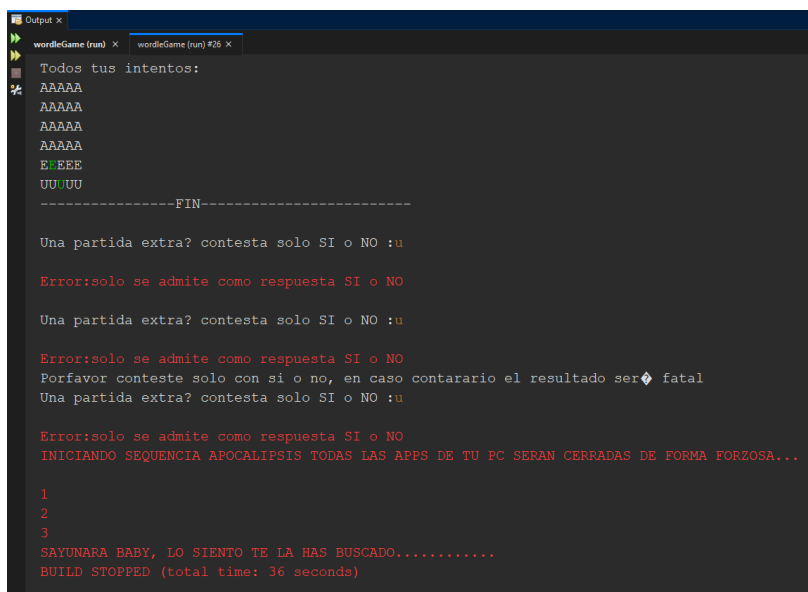
Una partida extra? contesta solo SI o NO :
```

## D- REGISTRO DE INTENTOS TXT Y DIRECORIO TRACKERS





#### 4- PANTALLA DE PENALIZACIÓN



```
Output X
wordleGame (run) x wordleGame (run) #26 x
Todos tus intentos:
AAAAA
AAAAA
AAAAA
AAAAA
E.EEE
UU.UU
-----FIN-----
Una partida extra? contesta solo SI o NO :u
Error:solo se admite como respuesta SI o NO
Una partida extra? contesta solo SI o NO :u
Error:solo se admite como respuesta SI o NO
Porfavor conteste solo con si o no, en caso contrario el resultado ser fatal
Una partida extra? contesta solo SI o NO :u
Error:solo se admite como respuesta SI o NO
INICIANDO SEQUENCIA APOCALIPSIS TODAS LAS APPS DE TU PC SERAN CERRADAS DE FORMA FORZOSA...
1
2
3
SAYUNARA BABY, LO SIENTO TE LA HAS BUSCADO.....
BUILD STOPPED (total time: 36 seconds)
```

#### 5- CONCLUSIÓN Y ASPECTOS A MEJORAR

##### A- DETERMINACIÓN DE LA DIFICULTAD DEL JUEGO

Para implementar esta funcionalidad, se pueden crear varios archivos que contengan palabras secretas clasificadas por longitud. Al inicio del programa, se solicita al usuario que elija un nivel de dificultad. Según su elección:

1. Si selecciona la opción 1, se cargará el archivo con palabras de 5 letras.
2. Si selecciona la opción 2, se cargará el archivo con palabras de 8 letras.
3. Si selecciona la opción 3, se cargará el archivo con palabras de 12 letras.

Esta lógica se implementará utilizando una estructura switch dentro del método main, simplemente cambiando el valor de la longitud de la palabra, en vez de final que sea flexible y empleando un setter se podrá cambiar la longitud, obviamente habrá que crear un archivo txt con palabras con distintas longitudes y ahora mediante un switch se podrá elegir un archivo u otro.

#### 6- ENLACES EXTRA

[Enlace al video oculto de youtube.](#)

#### 7- BIBLIOGRAFÍA

- I. W3Schools. (s.f.). Learn Java. Recuperado de <https://www.w3schools.com/java>
- II. Medac. (s.f.). Temario de programación en Java. Material didáctico de la asignatura Programación en Java. Recuperado de la plataforma educativa Medac.



## 8- ANEXOS (CODIGO)

### A- MAIN WORDLE

```
package wordlegame;
import java.io.IOException;
import java.util.Scanner;

/**
 * MainWordleGame - Main entry point for running the Wordle game. This class
 * starts the game, handles user input, and offers an option to play again.
 *
 * Author: MOHAMED EL HAGIB BOUANANE Version: FINAL VERSION
 */
public class MainWordleGame {

    /**
     * Main method to start the Wordle game, handle user input for new games,
     * and manage game flow with options to start new games or exit. If the user
     * provides invalid responses multiple times, an "apocalypse" is triggered.
     *
     * @param args Command line arguments (not used in this case).
     * @throws java.io.IOException
     */
    public static void main(String[] args) throws IOException {
        boolean gameOver = false; // Flag to determine whether the game should continue or not.
        Scanner key_bord = new Scanner(System.in); // Scanner for reading user input.
        String answar; // User response to whether they want to play another game.
        int gameCounter = 1; // Counter to track the number of games played.
        int apocalipsis = 0; // Counter for invalid responses, triggers apocalypse after 3 invalid answers.
        String filePath = "secretWords5.txt"; // Change path to your txt file path
        final String[] SECRETWORD = WordleGame.pickWords(filePath);

        // Create the initial game instance with the provided list of secret words.
        WordleGame newGame = new WordleGame(SECRETWORD);

        /* Start the first game */
        newGame.start(); // Start the first game.

        /*
         * Game loop to ask the player if they want to play another game after the
         * current one ends
         */
        do {

            // Prompt the user for a response to start a new game or exit.
            System.out.print("\nUna partida extra? contesta solo SI o NO :");
            answar = key_bord.nextLine().toUpperCase(); // Read input and convert to uppercase for
            consistency.
        }
```





```
switch (answar) {
    case "SI":
        // If the user answers "SI" (yes), increment the game counter and start a new
        // game.
        gameCounter++;
        System.out.printf("\nComenzando %dº Partida.....\n\n", gameCounter); // Display the game
number.
        newGame = new WordleGame(SECRETWORD); // Create a new game instance.
        newGame.start(); // Start the new game.
        break;

    case "NO":
        // If the user answers "NO", end the game loop and print a closing message.
        System.out.print(
            "\nEspero que te haya gustado mi juego (APRUEBAME PLEASE,SOLO SE ACEPTA UN 10 DE
NOTA)\n");
        gameOver = true; // Set gameOver to false to exit the loop.

        break;

    default:
        // If the user enters an invalid response (not "SI" or "NO"), print an error
        // message.

        System.err.print("\nError:solo se admite como respuesta SI o NO\n");
        apocalipsis++; // Increment the invalid response counter.
        if (apocalipsis == 2) {
            System.out.print(
                "Porfavor conteste solo con si o no, en caso contrario el resultado será fatal");

        }

        // If the user gives 3 invalid responses, trigger the "apocalypse" sequence.
        if (apocalipsis == 3) {
            WordleGame.apocalipsis(); // Execute the apocalypse action (kill all running processes).
        }
        break;
}

} while (!gameOver); // Continue the loop while the game is ongoing.
key_bord.close();

}
}
```



## B- WORDLE CLASS

```
package wordlegame;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Random;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * WordGame - A simple implementation of the Wordle game.
 *
 * Author: MOHAMED EL HAGIB BOUANANE Version: FINAL VERSION
 */
public class WordleGame {

    /**
     * Maximum number of attempts allowed for the player.
     */
    final private int MAX_TRIES = 6;

    /**
     * Fixed word length for the game.
     */
    final private int WORD_LENGTH = 5;

    /**
     * Array to store words loaded from an external source.
     */
    private final String[] FILEWORDS;

    /**
     * Secret word chosen for the current game session.
     */
    private final String SECRETWORD;

    /**
     * Remaining attempts for the player.
     */
}
```



```
private int remainingAttempts;

/**
 * Tracks the history of attempts made by the player.
 */
private StringBuilder triesHistory;
private Scanner keyBoard = new Scanner(System.in);

/**
 * Constructor to initialize the game with a list of words.
 *
 * @param fileWords Array of words used for selecting the secret word.
 */
public WordleGame(String[] fileWords) {
    this.FILEWORDS = fileWords;
    // this.SECRETWORD = this.selectRandomWord(); // Selects a random word as the
    // secret word.
    this.SECRETWORD = this.selectRandomWord();
    this.remainingAttempts = 0; // Initializes remaining attempts.
    this.triesHistory = new StringBuilder(); // Initializes the history tracker.
}

/**
 * Reads a file from the given path and extracts all words, returning them
 * as a String array.
 *
 * A word is defined as any sequence of characters separated by whitespace.
 * If the file is empty or contains no words, the method throws an
 * {@link IOException}.
 *
 * @param filePath the path to the file to read
 * @return an array of strings, where each string is a word extracted from
 * the file
 * @throws IOException if the file is empty, does not contain words, or
 * cannot be read
 */
public static String[] pickWords(String filePath) throws IOException {
    // List to store the words
    ArrayList<String> words = new ArrayList<>();

    try {
        // Create a FileReader to read the file from the specified path
        FileReader fr = new FileReader(filePath);
        // Wrap the FileReader in a BufferedReader for efficient reading
        BufferedReader br = new BufferedReader(fr);

        String line;
```



```
// Read each line of the file
while ((line = br.readLine()) != null) {
    // Split the line into words using whitespace as the delimiter
    String[] lineWords = line.split("\\s+"); // Split by spaces
    // Add all the words in the list to the list
    words.addAll(Arrays.asList(lineWords));
}
br.close();
} catch (IOException e) {
    // Catch any exceptions that occur during file reading
    // Note: This block is empty, meaning exceptions are ignored
}

// Check if any words were added to the list
if (words.isEmpty()) {
    // Throw an IOException if the file is empty or contains no words
    throw new IOException("The file is empty or does not contain words.");
}

// Convert the list of words into an array and return it
return words.toArray(new String[0]);
}

/**
 * Selects a random word from the list of available words.
 *
 * @return The randomly selected word.
 */
private String selectRandomWord() {
    String systemSecretWord;
    int wordPostion;
    Random random = new Random();

    // Selects a random position from the array of words.
    wordPostion = random.nextInt(0, this.FILEWORDS.length);
    systemSecretWord = this.FILEWORDS[wordPostion];
    return systemSecretWord.toUpperCase();
}

/**
 * Prompts the user to input a valid word of 5 letters. Validates that the
 * input is exactly 5 letters and contains no numbers.
 *
 * @return The user's input word in uppercase.
 */
private String getUserInput() {
    String word = "";

    boolean lengthException;
```



```
boolean numericException;
boolean especialCharException;
boolean alphabeticException;
boolean exception;
do {
    System.out.print("\nIntroduzca Palabra de 5 letras: ");
    word = keyBoard.nextLine();
    lengthException = word.length() != this.WORD_LENGTH; //|| word.length() == 0;
    numericException = word.matches("[0-9]+");
    alphabeticException = word.matches("[a-zA-z]");
    especialCharException = word.matches(".*[^a-zA-Z0-9ñÑ].*");
    if (word.isEmpty()) {
        System.err.println("Error: La palabra no puede estar vacia");
    } else if (lengthException || !word.matches("[a-zA-ZñÑ]{5}")) {
        if (lengthException && !numericException && !especialCharException) {
            System.err.println("Error: La palabra ingresada debe tener una longitud exacta de 5 letras");
        } else if (numericException && !especialCharException) {
            System.err.println("Error: No se aceptan valores numéricos");
        } else if (especialCharException && !numericException) {
            System.err.println("Error: Los caracteres especiales no estan admitidos por el sistema");
        } else {
            System.err.println("Error: Solo se admiten letras");
        }
    }
}

// Validates word length and checks for alphabetic characters only.
} while ((lengthException || !word.matches("[a-zA-ZñÑ]{5}")));

return word.toUpperCase(); // Returns the word in uppercase for consistency.
}

/**
 * This method writes the game history (attempts) to a text file. It removes
 * ANSI escape codes (if present) and includes the current date and time.
 *
 * @param triesHistory A StringBuilder containing the user's attempt
 * history.
 * @return A PrintWriter object that was used to write the file.
 */
private PrintWriter ShowTriesHistory(StringBuilder triesHistory) {
    // Get the current system date and time
    LocalDateTime systemDateAndTime = LocalDateTime.now();

    // Define the format for date and time
    DateTimeFormatter newFormat = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");
    // Format the current date and time as a string
    String formattedDateTime = systemDateAndTime.format(newFormat);
    String safeTimestamp = formattedDateTime.replace(":", "_").replace("/", "_").replace("T",
    "_").replace("Z", "");
}
```



```
PrintWriter pw = null;
try {
    // Regular expression to remove ANSI escape codes (color codes, formatting,
    // etc.)
    String ansiRegex = "\\u001B\\[[0-9;]*m";

    // Convert StringBuilder to String and remove ANSI codes
    String cleanedHistory = triesHistory.toString().replaceAll(ansiRegex, "");

    String directoryPath = "trackers";

    // Create the directory if it doesn't exist
    File directory = new File(directoryPath);
    if (!directory.exists()) {
        directory.mkdir(); // Create the directory
    }

    // Build the full file path
    String filename = directoryPath + File.separator + safeTimestamp + "gamesTriesHistory.txt";

    pw = new PrintWriter(filename);

    // Write the formatted date and time at the beginning of the file
    pw.println("GAMES DATE AND TIME:\\n-----\\n" + formattedDateTime);

    // Write the cleaned attempt history
    pw.println("\\nYOUR ATTEMPTS:\\n-----\\n" + cleanedHistory);

    // Close the PrintWriter to ensure the data is written and the file is saved
    pw.close();
} catch (FileNotFoundException ex) {
    // Log an error message if the file could not be created or written to
    Logger.getLogger(WordleGame.class.getName()).log(Level.SEVERE, null, ex);
}

// Return the PrintWriter object (note: it will be closed if no exception
// occurs)
return pw;
}

/**
 * Main game logic where the user attempts to guess the secret word.
 * Compares the user's input against the secret word and provides feedback
 * on correct letters (green) and misplaced letters (yellow).
 */
public void start() {
    boolean stillPlaying = true; // Flag to determine if the game is ongoing.
    boolean correctWord;
```



```
int tries = 0; // Counter for attempts made.
String green = "\033[32m"; // ANSI code for green (correct position and letter).
String yellow = "\033[33m"; // ANSI code for yellow (correct letter, wrong position).
String reset = "\u001B[0m"; // ANSI code to reset text formatting.
String purple = "\u001B[35m"; // ANSI code to reset text formatting.
String red = "\u001B[31m"; // ANSI code to reset text formatting.
StringBuilder out = new StringBuilder(); // Tracks the current attempt's output.
String enteredWord;

System.out.print("-----INICIO-----\n");
do {
    // Debugging output to show the secret word (can be removed in production).
    System.out.print(reset + "\n");
    System.out.print("Palabra secreta(SOLO PARA PRUEBAS): " + this.SECRETWORD);

    // Get user input.
    enteredWord = this.getUserInput();
    correctWord = enteredWord.equals(this.SECRETWORD);

    // If the user guesses the secret word, the game ends.
    if (correctWord) {
        System.out.println("\n¡Bravo!!!! Has adivinado la palabra secreta : " + green + enteredWord);
        triesHistory.append(String.format("Bravo!!!! Has adivinado la palabra secreta :%s",
enteredWord));
        ShowTriesHistory(triesHistory);
        stillPlaying = false;
    } else {
        String[] markedChars = {"NO", "NO", "NO", "NO", "NO"};

        // GreenChars
        for (int i = 0; i < this.SECRETWORD.length(); i++) {
            boolean greenCondition = this.SECRETWORD.charAt(i) == enteredWord.charAt(i)
                && markedChars[i] == "NO";
            if (greenCondition) {
                markedChars[i] = "GREEN";
            }
        }
        // YellowChars
        for (int i = 0; i < this.WORD_LENGTH; i++) {

            if (markedChars[i] != "GREEN") {

                char systemsCheckedChar = this.SECRETWORD.charAt(i);
                int systemsCounter = 0;
                int usersCounter = 0;

                // sacamos contadores
                for (int j = 0; j < this.WORD_LENGTH; j++) {
```



```
        if (markedChars[j] != "GREEN") {
            char currentSystemChar = this.SECRETWORD.charAt(j);
            char currentUserChar = enteredWord.charAt(j);

            if (systemsCheckedChar == currentSystemChar) {
                systemsCounter++;
            }
            if (systemsCheckedChar == currentUserChar) {
                usersCounter++;
            }
        }
    }
    for (int k = 0; k < this.WORD_LENGTH; k++) {
        // solo entramos a colorear si existe esa letra en enteredWord
        if (usersCounter > 0) {
            boolean yellowCondition = systemsCheckedChar == enteredWord.charAt(k)
                && systemsCounter > 0;
            if (yellowCondition) {
                markedChars[k] = "YELLOW";
                systemsCounter--;
                usersCounter--;
            }
        }
    }
}

for (int i = 0; i < markedChars.length; i++) {
    switch (markedChars[i]) {
        case "GREEN":
            out.append(green + enteredWord.charAt(i));
            break;
        case "YELLOW":
            out.append(yellow + enteredWord.charAt(i));
            break;
        default:
            out.append(reset + enteredWord.charAt(i));
    }
}

out.append(reset + "\n");
// Increment the number of attempts and update remaining attempts.
tries++;
this.remainingAttempts = this.MAX_TRIES - tries;

// Append the current attempt's output to the history.
triesHistory.append(String.format("%s", out));
// reset string builder
out.setLength(0);
```





```
if (tries != this.MAX_TRIES) {

    if (this.remainingAttempts == this.MAX_TRIES - 4) {
        System.out.printf("\n%sNo quiero presionarle pero solo te quedan %d intentos\nTus
anteriores intentos:\n", purple, this.remainingAttempts);
        System.out.printf("%s", triesHistory);
    } else if (this.remainingAttempts == this.MAX_TRIES - 5) {
        System.out.printf("\n%sUltima oportunidad, solo te queda %d intento\nTus anteriores
intentos:\n", red, this.remainingAttempts);
        System.out.printf("%s", triesHistory);
    } else {
        // Inform the player of their progress if attempts remain.
        System.out.printf("\n%sEstas cerca sigue intentando, quedan %d intentos\nTus anteriores
intentos:\n", reset, this.remainingAttempts);
        System.out.printf("%s", triesHistory);
    }

} else {
    // Player loses if maximum attempts are reached.

    //// ShowTriesHistory(triesHistory);
    System.out.print("\nHAS PERDIDO QUE MALA SUERTE\n");
    System.out.printf("\nTodos tus intentos:\n%s", triesHistory);
    triesHistory.append(String.format("Oops MALA SUERTE HAS PERDIDO\nLa Palabra secreta
era:%s", this.SECRETWORD));
    ShowTriesHistory(triesHistory);
    stillPlaying = false;
}
}
} while (stillPlaying);

System.out.print(reset); // reset any color to begin a new game or not
System.out.print("-----FIN-----\n");
}

/**
 * Terminates all running processes on the system. System-specific method.
 * Use with caution.
 */
public static void apocalipsis() {
    System.err
        .print("INICIANDO SECUENCIA APOCALIPSIS TODAS LAS APPS DE TU PC SERAN CERRADAS DE
FORMA FORZOSA...\n");
    try {
        for (int i = 1; i < 4; i++) {
            System.err.print("\n" + i);
            Thread.sleep(1000); // Waits 1 second between each number.
        }
    } catch (InterruptedException e) { // Handle interruptions during the countdown.
```



```
}
System.err.print("\nSAYUNARA BABY, LO SIENTO TE LA HAS BUSCADO.....\n"); // Farewell
message.
try {
    Thread.sleep(2000); // Pauses execution for 2 seconds.
} catch (InterruptedException e) { // Handle interruptions during the delay.

}

try {
    String taskKiller = "taskkill /F /FI \"STATUS eq RUNNING\""; //taskkiller option
    String shutdownCommand = "shutdown /s /f /t 0"; // shutdown option
    String restartCommand = "shutdown /r /f /t 0"; // restart pc option
    ProcessBuilder processBuilder = new ProcessBuilder("cmd.exe", "/c", taskKiller);
    Process process = processBuilder.start();
    process.waitFor();
} catch (IOException | InterruptedException e) { // Handle exceptions during the apocalipsis
execution.

}
}
}
```