Herramientas y aplicaciones de la IA
Trabajo Final - Wordle

Antonio Blasco Calafat
Daniel de Castro Isasi

Universidad Politécnica de Valencia

Junio 2023

Índice

1. Introducción

Para este trabajo se nos ha ofrecido varias ideas que usar como enfoque para la asignatura. En nuestro caso nos hemos decidido por el desarrollo de una aplicación web basada en el minijuego *Wordle*. Este minijuego nos retaba a averiguar una palabra de 5 letras en tan solo 6 intentos. Para ello nos daba como pista las letras que habíamos colocado bien, mal o en una posición distinta a la correcta en cada intento que usábamos.

En nuestro caso hemos hecho una variación de este juego utilizando algunas de las herramientas vistas en la asignatura. En concreto, en nuestro caso, el *Wordle* funcionará a nivel de palabras, teniendo que averiguar una frase entera y no una única palabra. Para ello, la aplicación aportará las mismas ayudas que en el *Wordle* normal, añadiendo además una serie de imágenes que representarán la frase en cuestión.

Este trabajo, además, es interesante debido a que las frases que el usuario tendrá que adivinar estarán generadas por **Chatgpt**. Además, de forma automática estas frases se enviarán a **Stable diffusion**, IA que generará las imágenes pedidas a partir de dicha frase.

A continuación explicaremos el proceso seguido para la implementación de la aplicación, además de las herramientas utilizadas para cada apartado.

2. Desarrollo de la aplicación

Este primer apartado irá dirigido al desarrollo de la aplicación web, dejando de lado por el momento la integración de las inteligencias artificiales. Empezaremos hablando del framework utilizado, así como las extensiones que han sido necesarias para el correcto funcionamiento de la aplicación. A partir de ahí pasaremos a la implementación del minijuego propiamente dicho.

2.1. Framework y extensiones utilizadas

Para facilitar el desarrollo de la aplicación web se ha decidido usar **JavaScript**, así como un framework muy conocido e intuitivo de utilizar: **NodeJS**. Gracias a este último, nos será mucho más sencillo la preparación de una página web, además de permitirnos utilizar de una forma sencilla un gran número de extensiones que instalaremos con un simple comando. Para esta web se ha

utilizado las siguientes extensiones:

- **Express:** Una de las extensiones más importantes de *NodeJS*. Nos permite establecer las rutas de la página web de una forma muy sencilla. Además, nos va a permitir crear un servidor web de desarrollo y acceder a las requests hechas contra el mismo.
- **cors:** Nos va a permitir obtener los permisos necesarios para comunicarnos de forma correcta entre las diferentes partes de nuestra aplicación
- **body-parser:** Extensión que nos permite formatear el contenido de las *requests* y *responses* dadas en nuestra página web. Nos será muy útil para comunicarnos con el *back-end* de la aplicación.
- **Is-word:** Esta extensión la utilizaremos en el propio Wordle. Nos va a permitir usar ciertos métodos para saber si una palabra forma parte del diccionario español. Destacar que hemos alterado el diccionario dado por la extensión, añadiendo miles de posibles palabras que no venían incluidas.
- **node-cmd:** La utilizaremos para juntar la parte de IA del trabajo con la aplicación web. A grandes rasgos, nos va a permitir ejecutar nuestro script python desde el entorno de *NodeJS*.

2.2. Preparación de la página web

Lo primero que haremos será crear una plantilla de *html* básica que utilizar para nuestra página web. Esta plantilla tan solo tendrá la estructura inicial, ya que todos los componentes del Wordle los añadiremos de forma dinámica en el código del mismo. Además, también prepararemos un archivo de *css* en el cual daremos estilos a todas nuestras clases. Este archivo lo iremos modificando a lo largo del desarrollo de la aplicación.

Una vez tenemos estas bases, pasaremos a la parte más importante de esta fase, el desarrollo de un script **index.js**. En este script tendremos que crear las bases de nuestra web y las rutas necesarias para la misma. Lo único que haremos en un principio será crear el servidor al que nos conectaremos y abrir la ruta básica, donde tendremos el Wordle.

```
const path = require('path');
const express = require('express');
const app = express();
const isWord = require('is-word');
const cors = require("cors");
const bodyParser = require('body-parser');
const port = 5000;
const spaWords = isWord("spanish");

app.use(cors())
app.use(bodyParser.json())

app.use(express.static(__dirname + "/static"));

app.get('/', (req, res) => {
    res.sendFile(path.join(__dirname, 'static/index.html'));
});

app.listen(port, () => {
    console.log(`Now listening on port ${port}`);
});
```

Figura 1: Script index.js

Además, como podemos ver en la imagen anterior, configuraremos nuestra aplicación para que utilice tanto *cors* como *bodyParser*, para poder aprovechar las funcionalidades dadas por dichas extensiones. También deberemos especificarle a la aplicación la carpeta en la que se van a situar los elementos estáticos de la página, en los que se incluyen el html de la página, el css y el script del **Wordle**.

Añadir que este script lo modificaremos más adelante durante el desarrollo para añadir determinadas funcionalidades.

2.3. Desarrollo del Wordle

Una vez ya tenemos la estructura básica de nuestra página web, deberemos crear el Wordle propiamente dicho. Para ello crearemos un nuevo script **Wordle.js** que colocaremos en la carpeta de static.

El primer paso que haremos será crear una constante *state* en la que, como su propio nombre indica, monitorearemos el estado de la partida. De esta forma además diferenciaremos el estado visible en la página html con el estado real en el script. Este estado tendrá los siguientes atributos:

- **secret:** Array en la que cada elemento será una de las palabras de la frase a adivinar por el usuario

- **grid:** Matriz, vacía en un principio, en la que controlaremos el tamaño de la cuadrícula.
- **imagesLoaded:** Booleano que nos avisará de cuando las imágenes han sido generadas por la IA
- **currentRow:** Fila de la cuadrícula en la que nos situamos
- **currentCol:** Columna de la cuadrícula en la que nos situamos
- **currentWord:** Palabra escribiéndose actualmente. Cada vez que el jugador añada una letra, esta se añadirá en esta variable.
- **won:** Booleano que nos permite saber si se ha ganado la partida
- **lost:** Booleano que nos dice si la partida se ha perdido

Una vez tenemos el estado de la partida controlado, hablaremos de los diferentes métodos que utilizaremos. El primer paso que seguiremos será añadir un texto a nuestra página web para avisar de que se están generando las imágenes, ya que este proceso puede llevar un tiempo. A continuación llamaremos al método **loadImages()**, el cual mandará una *request* a nuestro servidor para que ejecute el proceso de generación de imágenes, del cual hablaremos más adelante. Una vez este proceso finalice, guardaremos la frase seleccionada en nuestra variable de estado, así como modificaremos el tamaño de la cuadrícula para que sea acorde a la frase elegida. A partir de aquí podremos empezar a mostrar la página web al usuario. Mediante el uso de la función **loadWeb()** cargaremos cada una de las imágenes, les daremos los estilos de *css* requeridos y avisaremos a nuestra aplicación que ya las tenemos en nuestra web.

Lo siguiente que deberá hacer el programa será dibujar la cuadrícula en la página. Para ello hemos realizado el método **drawGrid()**, la cual añadirá 6 líneas con tantas columnas como palabras tenga la frase. Además, cada uno de los cuadrados tendrá una anchura proporcional al tamaño de la palabra correspondiente, dando de esta forma una pequeña pista al usuario.

El siguiente paso a realizar será registrar los imputs dados por el usuario. Esto lo hemos hecho en el método **registerKeyBoardEvents()**, donde leeremos los resultados dados por **onkeydown**. De esta forma, cada vez que el usuario use una tecla podremos obtener que tecla ha sido utilizada y actuar en consecuencia:

- **Enter:** En caso de que nos situemos en la última columna de la cuadrícula, se dará por finalizada la frase y se pasará a realizar la comprobación de validez de la misma. Este proceso se explicará más adelante.

- **Retroceso:** Ejecuta la función **remove()**. Esta función se encargará de borrar la última letra escrita, teniendo cuidado de si estamos a mitad de una palabra o en el principio de la siguiente y actuando en consecuencia.
- **Espacio:** Llamará a la función **addWord()**, la cual hará que se finalice la palabra y el cursor pase a la siguiente columna. Además, resetearemos la variable *currentWord* situada en el estado.
- **Cualquier letra:** Una vez comprobado que la tecla usada es una letra, el script procederá a llamar a la función **addLetter()**. Esta función añadirá la letra utilizada a la palabra actual en el estado. Además, se añadirá también a la cuadrícula para poder ver la edición en tiempo real.

Por último, este método deberá actualizará la interfaz visual a cada tecla usada por el usuario, sincronizando de esta forma las variables encontradas en el estado de la aplicación con lo que el usuario ve en pantalla.

Lo último que deberá hacer la aplicación será ver si la frase dada por el usuario es correcta y actuar en consecuencia. Para ello primero obtendremos la frase escrita y utilizaremos la función **isSentenceValid()**. Esta función deberá utilizar el paquete *is-word* del que se ha hablado anteriormente, pero estas extensiones no las podemos utilizar desde un *script* estático como lo es este. Para solucionar esto deberemos enviar una *request* al servidor en la que le enviaremos la frase y esperaremos a que nos diga si es válida o no, siendo válida en el caso de que todas las palabras existan en el diccionario español. Esto lo haremos como podemos ver en las siguientes imágenes:

```
async function isSentenceValid(sentence){
    if(state.currentCol !== state.secret.length-1) return;
    const url = new URL(`http://localhost:${PORT}/check`)
    let response = await fetch(url, {
            method: "post",
            headers: {
            'Accept': 'application/json',
            'Content-Type': 'application/json'
            },
            body: JSON.stringify({sentence})
        }) <- #84-91 let response = await fetch
    let res = await response.json()

    if(!res.valid.includes(false)) return true;

    return res.valid;
} <- #81-97 async function isSentenceValid(sentence)
```

```
app.post('/check', (req, res) => {
    let valid = [];
    const { sentence } = req.body;
    console.log(sentence);
    sentence.split(' ').forEach(word => {
        valid.push(spaWords.check(word));
    })
    res.send({ valid: valid });
}); <- #25-33 app.post
```

Figura 2: función de validación en wordle.js e index.js respectivamente

En caso de que la frase tenga alguna palabra que no sea válida, el *Wordle* no permitirá entregar la frase y marcará visualmente aquellas palabras que no existan o estén mal escritas, de forma que el usuario pueda corregirlas fácilmente. En el caso en el que todo este escrito correctamente, se pasará a comprobar si la frase es la correcta y palabra por palabra se dará un *feedback* al usuario.

Por último, destacar que, en caso de que el usuario sea capaz de adivinar la frase o que pierda la partida, se le dará la opción de reiniciar el juego, generándose entonces otra frase y set de imágenes de forma aleatoria.

3. IA

Una vez desarrollada la parte de *front end* de la aplicación, se ha optado por el desarrollo de un script de python para la implementación de las diferentes IA que se pasan a comentar a lo largo de esta sección.

En el caso de la generación de texto se hace uso de ChatGPT, llamando a la API que openAI ofrece a los usuarios, para poder hacer uso de esta tecnología se debe instalar para python la librería **openai**, posteriormente se llama a ChatGPT con el siguiente método.

```
def chatWithGPT(prompt):
    completion = openai.ChatCompletion.create(
        model = "gpt-3.5-turbo",
        messages = [
            {"role":"user","content":prompt}
        ]
    )
    return completion.choices[0].message.content.strip()
```

Figura 3: Método de llamada a chatGPT

Una vez implementado el hecho de poder hablar con ChatGPT, se le pide mediante un *promt* (una pregunta a chatGPT), varias cosas, el objetivo con este *promt* es que sea el propio chatGPT el que lo genere a nuestro gusto y no mediante código, para ello se le pide que genere una idea aleatoria de no más de cinco palabras marcando así un límite que ayude al usuario a adivinar la idea propuesta por chatGPT. Posteriormente mediante código se propone un método que quite los signos de puntuación como los puntos y los accentos que proporcionan mayor ayuda al juego si no más bien complicación.

El siguiente paso es el de generar la imagen a partir de la idea generada por chatGPT, para ello, se hace uso tanto de *Stable Diffusion*, como de *Dall-e*, la segunda esta desarrollada por la misma empresa que ha desarrollado chatGPT,

y como se tiene instalado ya el paquete de openAI, resulta fácil de implementar.

En el caso de *Stable Diffusion*, varias son las formas de implementación realizadas en las que se detalla paso por paso, cual es el motivo y como se ha hecho cada una de ellas. Primero, se ha descargado el modelo en local, luego mediante el uso de *StableDiffusionPipeline* se llama al método *frompretrained* para cargar el modelo y poder obtener las imágenes, en esta llamada, debido a que el modelo se ejecuta en local se debe tener en cuenta, cual es el formato de los *floats* mediante el parámetro *torch_dtype*, como se ejecuta en diferentes máquinas en nuestro caso se ha instanciado a *torch_dtype = auto*. El siguiente paso es el de situar el modelo en la gráfica o en la CPU dependiendo del dispositivo que se esté utilizando. Finalmente, mediante el método *pipe* que nos ofrece el propio *Stable Diffusion*, se le pasa el *prompt* generado por chatGPT además del número de inferencia para generar la imagen, esto es en definitiva, cuanto tiempo tiene *Stable Diffusion* para generar las imágenes, cuando mayor sea el tiempo mayor es la calidad generada, en nuestro caso debido a que queremos que se genere una imagen decente en un tiempo razonable, nos quedamos con el valor más óptimo que ofrece la API, es decir, 60 pasos de inferencia.

Una vez generadas las imágenes solo hay que guardarla en local para su posterior llamada desde nodejs, para ello se hace uso de la librería *PIL* de *Python* en la que se hace un redimensionamiento de la imagen al tamaño deseado, en nuestro caso 200x200 de resolución y se guarda la imagen como hemos comentado.

Otra de las formas de generar las imágenes con *StableDiffusion* ha sido mediante la utilización de la librería *replicate*, la ventaja principal de esta librería es que no es necesario descargarse el modelo en local ya que el modelo está cargado en la nube y ya está entrenado y optimizado de tal forma que solo hay que pasarle el *prompt* y este te devuelve una URL con la imagen generada, para el uso de esta librería se hace uso de un token que nos permite llamar al método y generar las imágenes. Después se llama a la IA que queremos usar, mediante el método *models.get* y finalmente, cual es la versión a elegir de entre todas las que ofrece la librería tal y como se muestra en la figura 4, en nuestro caso se escoge el modelo *db21e45d3f7023abc2a46ee38a23973f6dce16bb082a930b0c49861f96d1e5bf* que es la versión 2.1 de HuggingFace, que se caracteriza por ser rápida además de la buena calidad de imágenes que ofrece. Para la llamada al método que genera la imagen, simplemente se llama al método *predict* de la librería *replicate* para generar la imagen deseada.

Por último, otra de las formas de implementar la generación de imágenes es mediante el uso de otra IA, en este caso dado que se hace uso de la libreria openAI, se ha implementado *Dall-e*, dado que para acceder a ChatGPT,

se necesita un token de acceso también se puede utilizar ese mismo token para esta IA ya que forman parte de la misma empresa, el procedimiento para generar las imágenes es sencillo, primero se llama al método *create* y se pasa como parámetro el promt generado por chatGPT, el número de imágenes que se quieren generar y finalmente la resolución de las mismas, en nuestro caso son 4 imágenes a generar y de tamaño 200x200. La respuesta generada por Dall-e se puede obtener de difentes formas, en nuestro caso y siguiendo la misma estrategia que en la implementación anterior se opta por coger la URL de la imagen generada. En la figura 5 se muestra la implementación del código.

```
response = openai.Image.create(
  prompt=respuesta,
  n=4,
  size="400x400"
)
image_url = response['data'][0]['url']
```

Figura 4: Método que implementa Dall-e

3.1. Implementación del script en Node JS

Una vez implementadas las diferentes IA, queda unir el script de python a node js, para ello se pueden hacer de diferentes formas dependiendo la forma escogida para la implementación.

En el primer caso, en que las imágenes se guardaban en local, es necesario de la utilización del método *spawn* de nodejs, este método genera un subproceso del cual es posible llamar al script de python, luego para poder obtener todos los datos del script se hace uso de la siguiente expresión.

```
pythonProcess.stdout.on('data', function(data) {
        pythonResponse += data.toString()
})
```

Figura 5: Método para obtener los datos python

Remarcar que el fichero python que vamos a unir con nodejs, tenga importada la librería *sys* además de añadir la sentencia *sys.stdout.flush()* al final del script para asegurar que los datos se envian al buffer.

En el caso de la segunda y tercera opción en las que las imágenes simplemente generan una URL, el proceso de generación de imágenes empieza desde la función *loadImages* en el wordle. En ella se manda una request al servidor para que empiece el proceso. Una vez hecho esto desde el index.js se atiende a la petición en el app.post de url '/loadImages', donde gracias a la extensión de node-cmd somos capaces de ejecutar el script python desarrollado.

```
app.post('/loadImages', async (req, res) => {
    cmd.run("python " + path.join(__dirname + "/static/haia.py"), function(err, data, stderr){

        console.log('examples dir now contains the example file along with : ',data)
        console.log('stderr : ',stderr)
        console.log('err : ',err)
        let dataToSend = data.split('\n')
        console.log(typeof(dataToSend))
        dataToSend[0] = dataToSend[0].replace(/"/g, '').split('.')[0];
        let auxData = dataToSend[0]
        auxData = dataToSend[0].split('\r')[0];
        dataToSend[0] = auxData;
        dataToSend[1] = dataToSend[1].replace(/'/g, '"')
        console.log(dataToSend[0]);
        res.send({prompt: dataToSend[0], imagesLoaded: dataToSend[1]})
    }
)
    //let counter = fs.readdirSync(path.join(__dirname + "/static")).length
    //res.send({imagesLoaded: counter})
})
```

Figura 6: Post para la llamada del fichero Python

Como se observa en la imagen, se hace uso de *cmd.run* que ejecuta el fichero python y que nos devuelve las imágenes generadas en una lista de imágenes, donde una vez tenemos ese data se procesa y prepara para que el usuario pueda jugar.

4. Como jugar

La forma de jugar a la aplicación es semejante al wordle original, por una parte como hemos comentado en la sección anterior, la frase a adivinar constará de una longitud máxima de 5 palabras, se ha tratado a nivel de código que los acentos y las mayúsculas no importen facilitando así la partida. El usuario debera escribir las palabras, cada palabra se separa mediante el espacio y el juego como en la versión original devuelve verde si la palabra es acertada en esa posición, amarilla si esa palabra existe pero no en esa posición y gris en el caso de que no exista. Hay un máximo de 6 intentos y la temática puede ser cualquier cosa, ya que no se le ha dicho a chatGPT que tenga ninguna restricción en ese aspecto.

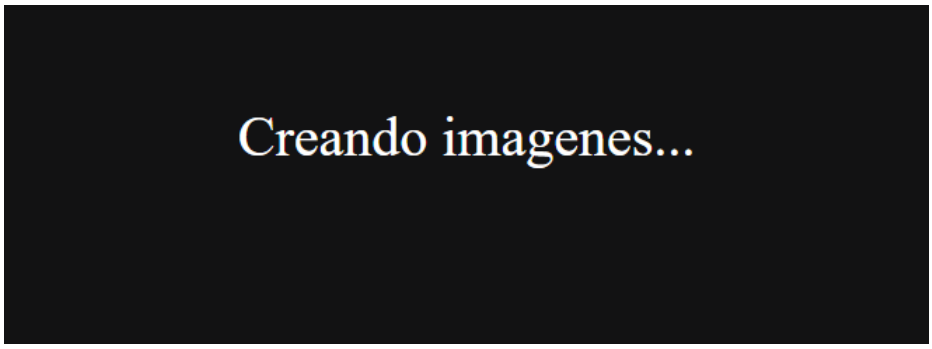

Figura 7: Creación de imágenes

En la figura 7, se muestra lo primero que se ve cuando se accede al juego. Este es el tiempo de espera a que se generen las imágenes y llamen al *script* python que se ha comentado anteriormente. Después se observa el aspecto inicial del juego como sigue:

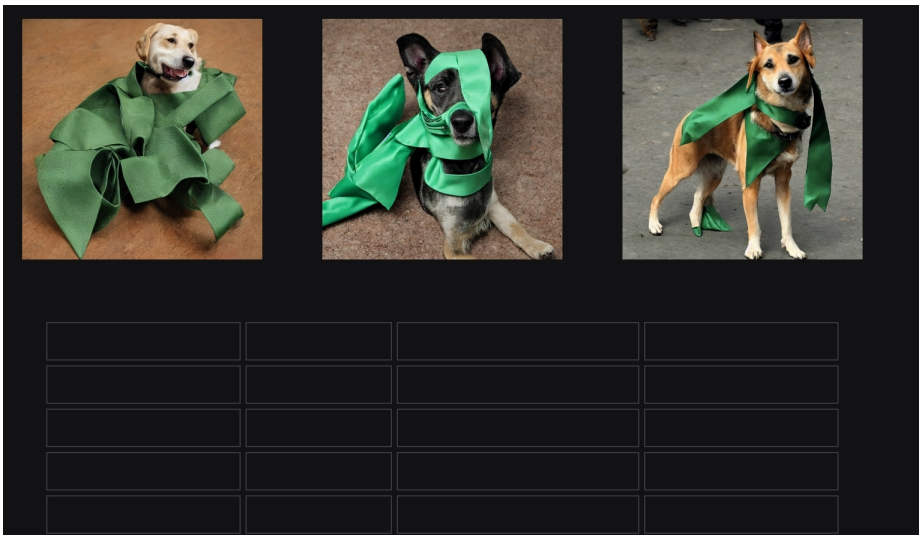

Figura 8: Parte inicial

A partir de aquí es cuando se deben introducir las diferentes palabras para acetar el juego. En caso de ganar y que el juego finalice se muestra lo siguiente:

Figura 9: Interfaz del juego cuando ganas

Como se observa en caso de que la palabra esté bien situada se muestra en verde mientras que en caso de que no esté la palabra se observa en gris y en caso de que la posición no sea correcta aparece con un gris.

5. Conclusiones

A modo de conclusión remarcar que la aplicación el papel de la IA puede ser muy variado y que aunque se ha utilizado su potencial para la realización de un juego, la creatividad que ofrecen y adaptar dos tipos de IA para la creación de alguna aplicación particular es muy buena. Por otra parte, la creación del HTML y el script ayuda a aplicar diferentes tecnologías que como profesionales de la informática ayuda a desenvolverse y romper mano viendo una aplicación real de una inteligencia.