Desarrollo de una aplicación para el aprendizaje de lenguajes de programación mediante el uso de IA y gamificación

Juan José Jurado Romo

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación
Aplicaciones multimedia para e-learning

Tutor/a de TF
Aleix López Antón
Profesor/a responsable de la asignatura
Jose Antonio Morán Moreno

12/06/2023

Ficha del Trabajo Final

Título del trabajo:	Desarrollo de una aplicación para el aprendizaje de lenguajes de programación mediante el uso de IA y gamificación
Nombre del autor/a:	Juan José Jurado Romo
Nombre del Tutor/a de TF:	Aleix López Antón
Nombre del/de la PRA:	Jose Antonio Morán Moreno
Fecha de entrega:	06/2023
Titulación o programa:	Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación
Área del Trabajo Final:	Aplicaciones multimedia para e-learning
Idioma del trabajo:	Castellano
Palabras clave	E-learning, gamificación, aprendizaje

Resumen del Trabajo

En este proyecto trataremos de mejorar y facilitar el aprendizaje de diferentes lenguajes de programación, para ello haremos uso de la gamificación. Esto lo haremos mediante el desarrollo de una aplicación con un sistema de juego para facilitar y hacer más divertido el aprendizaje de varios lenguajes.

Desarrollaremos una aplicación basada en el aprendizaje mediante la resolución de preguntas o problemas sobre un determinado código y que llevará al alumno a través de una serie de niveles, los cuales irán aumentando su complejidad progresivamente vaya progresando y dominando cada estructura o nivel de complejidad. Todo esto estará integrado con una interfaz basada en una temática y trama sencilla a lo largo de los niveles que haga de este aprendizaje algo ameno y divertido al alumno, y que de ser posible lo anime a seguir aprendiendo otros lenguajes o niveles más complejos.

Para generar los fragmentos de código en torno a los que se harán las preguntas de cada lenguaje se desarrollará una IA que generará pequeños fragmentos de código de forma aleatoria. Estos fragmentos de código estarán se adaptarán al nivel de progreso del alumno, y evitará la sensación de continua repetición y estancamiento que se puede producir al resolver los mismos problemas repetidos. Esto permitirá al alumno enfrentarse a una mayor variedad de situaciones y estructuras de código diferente.

Abstract

In this project we will try to improve and make easier the learning process for different programming languages, and we will make use of gamification. We will develop an application with a game system to make easier the learning of this different languages.

We will develop an app to learn resolving different questions or problems about a certain code snippet and take the student through a series of levels, which will increase his complexity as the student progresses and masters each structure or complexity level. All will be integrated with an interface based on a simple theme and plot throughout levels that will make this learning process something enjoyable and fun for the student, and maybe encourage him to continue learning other languages or more complex structures.

To generate the code snippets that we will use on each test, the AI will generate small code fragments randomly. The code snippets will be adapted to student and progress, and will avoid the feeling of repetition and stagnation that can occur when we are solving the same problems every time. This will allow the user to deal with more diversity and different code structures.

Índice

Lista de Figuras

1. Introducción

En la actualidad podemos encontrar una gran cantidad de plataformas de aprendizaje online e incluso universidades cuya formación es 100% online, como la UOC, donde se utilizan metodologías más modernas adaptadas al hecho de que no hay una interacción personal con los alumnos en clase. Además, al no tener que asistir a una clase de forma presencial durante unas horas concretas cada persona tiene la posibilidad de organizar su propio horario, compatibilizando en muchas ocasiones, trabajo y familia con los estudios.

Además, en la docencia de grados con asignaturas complejas como puede ser la ingeniería de telecomunicaciones o ingeniería informática encontramos otro obstáculo más que es el de impartir materias que no tienen una docencia al uso como pueden ser fundamentos de la programación y los diferentes tipos de lenguajes.

Teniendo en cuenta todo esto, nuestro proyecto tiene como principal objetivo el desarrollo de una aplicación que facilite, tanto al profesorado como al alumnado, el aprendizaje y enseñanza de dichas materias, intentando además alcanzar un equilibrio entre las clases y métodos actuales, algo más rutinarias, y un sistema basado en gamificación que motive y anime al estudiante en el estudio de estas materias.

Por todo lo anterior, vamos a basar el desarrollo de nuestro proyecto en dos áreas claramente diferenciadas:

- Aplicación e-learning: La parte fundamental de nuestro proyecto, la creación de una plataforma o aplicación de e-learning que resulte interesante y atractiva para los alumnos en el aprendizaje de diferentes

lenguajes de programación. Para ello utilizaremos técnicas y dinámicas basadas en la gamificación para conseguir que el alumno no sienta que este proceso de aprendizaje es un proceso aburrido o tedioso.

- Generador de código mediante IA: Para intentar optimizar el aprendizaje de dichos lenguajes y adaptar los ejercicios al nivel y frecuencia de uso de cada alumno utilizaremos algoritmos de aprendizaje basados en IA. Con estos algoritmos generaremos un conjunto de ejercicios y fragmentos de código que se utilizaran para ayudar al alumno en la práctica de los diferentes lenguajes, consiguiendo así una variedad mas personal y adecuada para cada alumno. Esto permitirá reducir el aburrimiento por repetición que suele darse en este tipo de cursos y por otro lado un aprendizaje personalizado y adaptado al ritmo de cada persona.

1.1 Objetivos

- Desarrollar un software capaz de generar cuestiones y/o fragmentos de código más variado con diferentes niveles para hacer menos monótono el aprendizaje.
- Desarrollar una aplicación que mediante cuestiones permita el aprendizaje de diferentes lenguajes de desarrollo.
- Utilización de la gamificación para motivar y hacer ameno dicho aprendizaje.

1.2 Motivación

La motivación para realizar este proyecto viene dada principalmente tras muchos años aprendiendo lenguajes de desarrollo de forma autodidacta, utilizando cursos o tutoriales disponibles en internet.

Estos cursos suelen ser básicos pero muy útiles para aprender la estructura y funciones básicas de cada lenguaje de desarrollo partiendo desde cero. Sin embargo, suelen motivar mucho en las primeras horas, pero el interés va decayendo a lo largo del curso debido a la falta de capítulos prácticos o ejercicios, ya que el hecho de leer código suele ser algo monótono y aburrido.

Es por esto por lo que la creación de una aplicación donde poder practicar un lenguaje de programación de una forma más amena y divertida mediante a la gamificación, puede ser un avance para mejorar el aprendizaje de nuevos lenguajes de programación. Al mismo tiempo el hecho de crear un algoritmo de generación de código aleatorio para tener mayor variedad puede aumentar la motivación del alumno al enfrentarse a una mayor diversidad de situaciones mas allá de las que se ha utilizado de forma habitual y que todos conocemos.

1.3 Planificación

La realización de este proyecto la hemos dividido en varios entregables los cuales irán formando parte del *software* que compondrá nuestra aplicación de aprendizaje. El proyecto se divide en dos partes, una dedicada a la IA y otra a la interfaz de la aplicación que se encargará de utilizar los fragmentos de código generados por el anterior bloque para crear los cuestionarios.

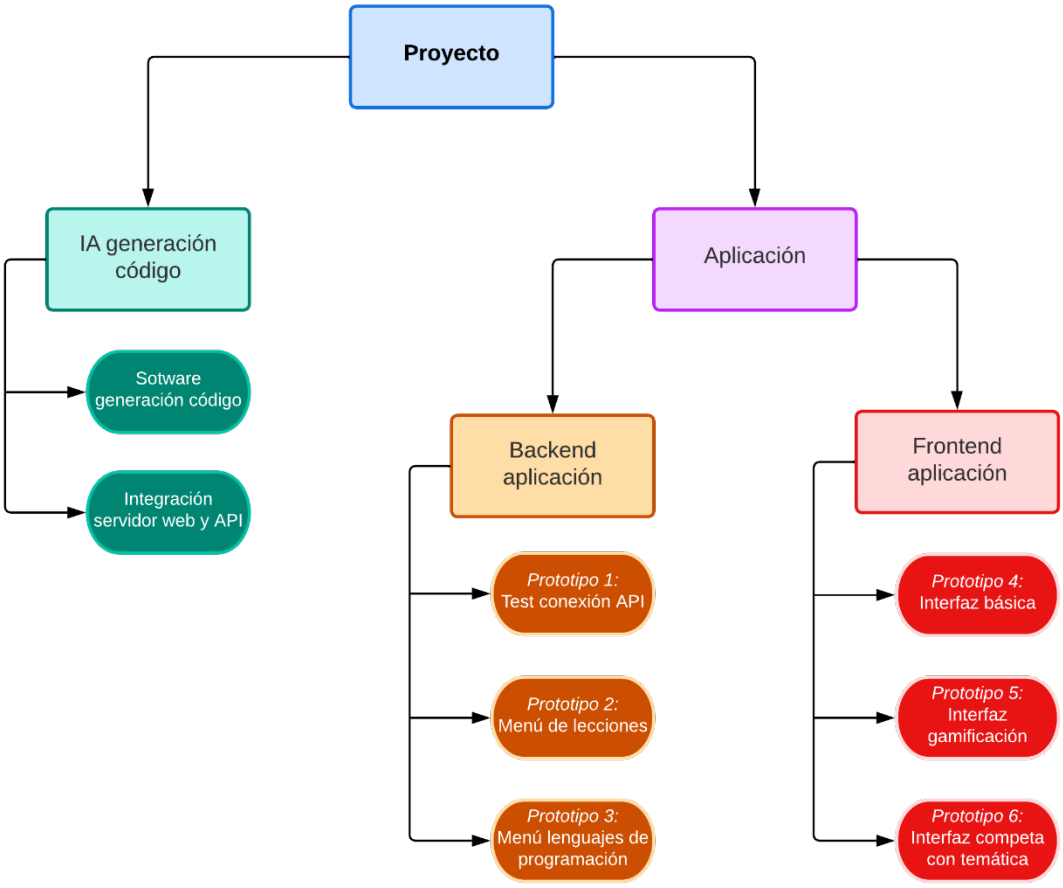

Figura 1. Planificación del trabajo

2. Estado del arte

2.1 Algoritmos de aprendizaje

2.1.1 Definición

En los últimos meses cada día surge o se habla sobre la aparición de nuevas inteligencias artificiales que se van desarrollando, cada una de ellas con diferentes finalidades. Algunas de estas inteligencias artificiales más conocidas son chatGPT como ejemplo de IA conversacional, Midjourney / Stable Diffusion como ejemplo de IAs para la generación de imágenes o GitHub Copilot X como ejemplo de IA para la generación de código.

Todos estos modelos, coloquialmente conocidos como *IA*, son modelos que han sido entrenados mediante un algoritmo de aprendizaje automático (Machine learning) o aprendizaje profundo (Deep learning), el cual permite a nuestro sistema de algoritmos aprender a reconocer patrones complejos analizando ejemplos y repetir dichos patrones según nuestras necesidades.

2.1.2 Machine learning

Las técnicas de *machine learning* o aprendizaje automático consisten en crear algoritmos capaces de identificar patrones complejos de forma autónoma mediante el análisis de grandes volúmenes de datos. Estos sistemas inteligentes son capaces de mejorar su propio rendimiento realizando continuos análisis, incluido de los datos que el propio algoritmo genera, para crear nuevos patrones mejorados o mejor adaptados a los requisitos de cada usuario.

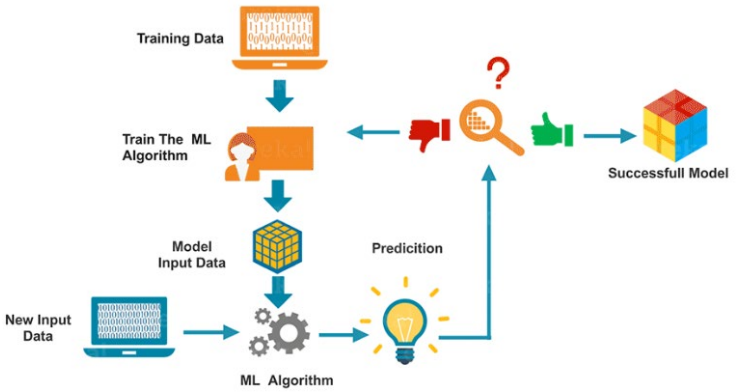

Figura 2. Flujo de desarrollo de un sistema machine learning
https://brainstation-23.com/machine-learning-simplified-approach-for-beginners/

En nuestro caso utilizaremos un algoritmo de aprendizaje autorregresivo comúnmente conocido como GPT (actualmente muy conocido por ser el que utiliza la IA conversacional más popular, chatGPT) el cual utiliza *machine learning* para intentar aprender patrones mediante el análisis de textos pudiendo así recrear e imitar la redacción humana. En nuestro caso, utilizaremos como datos de entrenamiento fragmentos de código, como si se tratara de textos, en diferentes lenguajes y solicitaremos que nos genere nuevo código en función de nuestras necesidades.

2.2 Generative Pretrained Transformer (GPT)

El algoritmo GPT se utiliza para reconocer patrones en los textos, en la forma de escribir humana y conocer las palabras más utilizadas en los textos convencionales. De esta forma el modelo es capaz de predecir que frases necesita utilizar según nuestras preguntas, y dentro de su respuesta elegir palabra a palabra cual será, con mayor probabilidad, la siguiente palabra o frase a utilizar según va generando el texto y según los patrones aprendidos tras analizar los datos.

Para que nos hagamos una idea de la complejidad de estos algoritmos actualmente nos basaremos en los datos del más utilizado y conocido, GPT-

3. Se trata de la tercera generación de estos modelos, y es en el cual está basado el conocido chatGPT desarrollado por OpenAI, este modelo ha sido generado a partir del uso de billones de palabras y utiliza 175 mil millones de parámetros para generar textos difícilmente diferenciables de uno escrito por un humano. Para que nos hagamos una idea de su complejidad el tamaño aproximado de este modelo es de unos 800 GB.

En nuestro caso, es evidente que no tenemos ni los recursos ni el volumen de datos para generar un modelo tan complejo. Sin embargo, como en nuestro caso vamos a aplicar este tipo de algoritmos para reconocer y generar código no necesitaremos utilizar un volumen tan grande de parámetros ni un algoritmo tan complejo, ya que los lenguajes de programación no tienen tantos matices ni necesitan de un contexto como ocurre con la lengua humana.

Para nuestro caso nos basaremos en unas versiones basadas en este modelo, pero más reducidas y adaptadas a usos mas sencillos, como es el nuestro. Estas versiones son mas adaptables y personalizables según los datos a utilizar para generar nuestro modelo.

2.2.1 GitHub Copilot

Un ejemplo de una IA ya existente capaz de generar código es GitHub Copilot. Este modelo de IA permite a los programadores autocompletar su código con sugerencias creadas por el algoritmo. También permite crear bloques completos de código cuando la entrada del modelo es un texto explicativo sobre que tipo de función se necesita programar y que valores debe tomar.

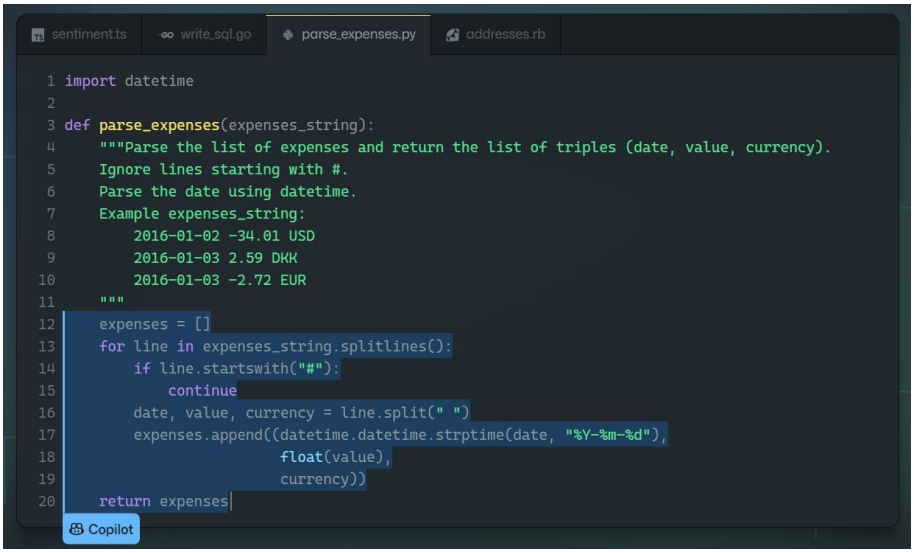

Figura 3. Ejemplo de generación de código utilizando GitHub Copilot
https://openwebinars.net/blog/github-copilot-que-es-y-primeras-impresiones/

Basándonos en esta idea, intentaremos crear nuestro propio modelo capaz de generar pequeños bloques de código. En nuestro caso, la entrada del modelo serán dos parámetros, un primer parámetro que indique que lenguaje de programación estamos practicando, y el segundo parámetro un valor que indique el nivel de complejidad que queremos que tenga este código, según la situación y el alumno.

2.2.2 minGPT

Para nuestro proyecto vamos a hacer uso de esta reinterpretación de la implementación del modelo GPT realizada por Andrej Karpathy. Se trata de un modelo que utiliza librerías denominadas *transformers* que convierten las palabras a valores numéricos o *tokens*, ya que estos modelos trabajan solo con números para crear los patrones de decisión. De esta forma la red neuronal convertirá los datos de entrada que proporcionemos en números asociados a palabras/código para crear sus propios patrones de diseño.

Una vez entrenada la red neuronal de nuestro modelo, este tendrá la capacidad de utilizar un algoritmo de decisión que cuando reciba una

entrada solicitando un tipo de código para un nivel o lenguaje determinado pueda calcular cual o cuales son los valores (Y, por tanto, su correspondiente palabra o código que obtendremos al *destokenizar* el valor) más adecuados a nuestra solicitud.

Esto debería permitirnos poder crear nuestra IA para la generación automática de código en base a los datos que nosotros preparemos bajo un criterio pedagógico según dificultad y lenguaje de programación. Además, este modelo permite de manera sencilla configurar la versión de GPT a utilizar, limitar el número de palabras y el tamaño máximo de los bloques de código, ya que como hemos indicado anteriormente vamos a simular un aprendizaje de lenguaje natural, pero utilizando bloques de código como si de párrafos de texto se tratase.

Sin embargo, la versión actual de minGPT se encuentra desactualizada y archivada, por lo que haremos uso de una nueva versión llamada nanoGPT.

2.2.3 nanoGPT

Se trata de una versión reescrita, más actualizada y eficiente de minGPT realizada por su mismo autor. La característica más interesante de esta nueva versión es el uso de OpenWebText, que contiene gran parte de la información y datos usados para el entrenamiento de *GPT-2*, lo cual nos permite partir de un modelo ya entrenado.

Además, algunas de las novedades más interesantes de esta nueva versión respecto a minGPT son:

- Posibilidad de utilizar *GPU* para entrenamientos mas complejos y extensos.
- Mayor número de opciones para el afinado del modelo.

- Aumento de la eficiencia mediante el uso de PyTorch 2.0.

2.3 E-learning

2.3.1 Definición y tipos

El e-learning (*Electronic Learning*) o aprendizaje electrónico es el concepto que hace referencia al aprendizaje mediante el uso de materiales o sistemas que se encuentran alojados en la nube o funcionan mediante el uso de Internet. Este aprendizaje está basado en el aprendizaje a distancia que surge mucho antes de la aparición de Internet con los primeros programas educativos por radio o correspondencia. Sin embargo, es a partir de la popularización del uso de Internet en las viviendas particulares cuando surge el concepto de aprendizaje electrónico con la aparición de los primeros cursos y plataformas online de enseñanza.

Aunque este tipo de enseñanza empieza a asentarse sobre el año 2000 es a partir del año 2020, con las restricciones de movilidad debido a la pandemia mundial por COVID cuando alcanza su mayor popularidad. Muchos de los alumnos y de las entidades educativas «tradicionales» buscan adaptarse a las limitaciones que establece dicha pandemia al impedir desplazamientos y grandes reuniones de personas, por lo que la realización de clases presenciales como era habitual resulta inviable.

Dentro del aprendizaje electrónico o e-learning encontramos varios tipos:

- Clases virtuales: Principalmente videollamadas a través de las cuales el profesor imparte clases de forma similar a las presenciales. Su principal ventaja es que dichas clases pueden ser grabadas y vistas con posterioridad y detenimiento por los alumnos.

- *M-learning*: Se trata de aprendizaje mediante el uso de dispositivos móviles como *smartphones* y *tablets*. Está pensada para permitir el aprendizaje utilizando cortos espacios de tiempo libre donde se suele usar estos dispositivos o mayor movilidad que un PC.
- Gamificación: Hace uso de juegos y estrategias basados en estos para conseguir una experiencia más divertida y variada a la vez que consigue crear un hábito de aprendizaje gracias al entretenimiento.

Será este último tipo (aunque puede ser combinado con M-learning) el que utilizaremos en nuestro proyecto.

2.3.2 Uso de la gamificación en e-learning

En la actualidad el uso de la gamificación se expande a muchos mas ámbitos que la docencia. Por ejemplo, en algunas empresas se empieza a utilizar, tanto de forma interna para optimizar los procesos de selección de personal intentado conseguir los mejores talentos entre los candidatos, como de forma externa buscando fidelizar clientes y obtener nuevos clientes mediante el uso de juegos como refuerzo positivo tras sus compras.

En nuestro caso, haremos uso de esta gamificación en el ámbito del e-learning para generar una experiencia lo más positiva posible para los alumnos e intentando motivarlos para que así disminuya la curva de aprendizaje. Además, estas técnicas de gamificación permiten crear un compromiso no forzado con el aprendizaje continuo lo cual permite una mayor asimilación de los conceptos por parte de los alumnos.

Sin embargo, es muy importante tener en cuenta que la ultima finalidad de este tipo de diseños no es la de crear un juego como tal, sino la de utilizar y crear mecánicas basadas en los sistemas de recompensa y objetivos en los

que se basa cualquier tipo de juego. De esta forma conseguiremos aumentar la motivación del alumno y crearemos una dinámica positiva de aprendizaje.

Para la gamificación algunas de las técnicas y dinámicas más utilizadas son:

- Competición: Basada en el instinto de competición humano y en la búsqueda de ser el mejor de una clasificación.
- Logros: Basado en la autosuperación o satisfacción personal al cumplir un objetivo propuesto.
- Recompensa: Búsqueda personal de satisfacción o beneficio que consideramos merecido.

2.3.3 Ventajas e inconvenientes de la gamificación en e-learning

En la múltiple documentación que encontramos sobre e-learning y gamificación destacan las siguientes ventajas:

- Actitud más positiva de cara al aprendizaje.
- Permite experimentar y conocer nuevas formas de aprendizaje.
- Aumenta la comunicación con otros compañeros y fomenta la colaboración y participación en grupo.
- Disminuye la curva de aprendizaje.
- Aumenta la capacidad de inventiva.

Y entre los inconvenientes destacan los siguientes:

- Posible coste elevado de desarrollo y mantenimiento de la aplicación.
- Alta probabilidad de distracción y de consumo excesivo de tiempo.

- Difícil equilibrio entre lo lúdico y el aprendizaje, lo cual puede provocar que no se obtengan los objetivos pedagógicos.
- Posible motivación pasajera que se pierde rápidamente sobre todo en el aprendizaje de materias complejas, por lo que debe complementarse con otro tipo de enseñanza más convencional.

Teniendo en cuenta todos estos factores habrá que buscar un equilibrio entre recompensas y aprendizaje, para evitar así caer en alguno de los aspectos negativos que tiene este tipo de gamificación. También deberemos tener en cuenta el tipo de recompensa que se da al alumno, y si fuera necesario establecer limitaciones de tiempo de uso al mismo. De esta forma evitaremos provocar el efecto contrario debido a un exceso de tiempo dedicado solo a la práctica en el uso de lenguajes de programación, y dando de lado a los aspectos teóricos que son base en el aprendizaje de los fundamentos de la programación.

Para esto habrá que tener en cuenta los pasos a seguir para intentar crear una herramienta lúdico-pedagógica equilibrada que nos permita realizar un proyecto interesante y atractivo para el alumno a la vez que cuidamos el contenido.

2.3.4 Pasos a seguir para una crear una gamificación e-learning

Teniendo en cuenta lo indicado en los apartados anteriores queda claro que desarrollar una aplicación de e-learning utilizando gamificación no es algo arbitrario y hay que tener en cuenta múltiples factores para obtener los resultados esperados.

Por esto, tendremos en cuenta los siguientes pasos a seguir, según la ITMadrid Digital School, para una correcta gamificación basada en e-learning:

Figura 4. Gamificación paso a paso
https://www.itmadrid.com/la-gamificacion-e-learning-paso-a-paso/

1. Empatizar con la audiencia: Como primer paso analizaremos de forma previa al desarrollo a que tipo de audiencia va dirigido nuestro proyecto de gamificación, sobre todo teniendo en cuenta cual es la motivación de nuestro alumnado y sus necesidades.

2. Crear el proyecto: Habrá que tener en cuenta la plataforma que va a utilizar el alumnado para su aprendizaje y adaptarlo a esta.

3. Curar el contenido: Consultar fuentes o expertos que nos recomienden que tipo de recompensas y que juegos son los mas apropiados para nuestro contenido.

4. Diseñar el curso: Seleccionaremos los tipos de juegos a realizar y las recompensas que otorgará cada tipo. Habrá que decidir si realizamos un ranking de alumnos o no.

5. Crear un guion o storyboard: Se plantearán bocetos sobre la interfaz y el diseño multimedia de nuestro proyecto, de forma que sea lo más ameno y cómodo para el alumno.

6. Desarrollar el juego: Con toda la información recopilada en los anteriores pasos decidiremos cual es la plataforma más adecuada para nuestro juego y procederemos al desarrollo.

7. Implementar el juego en la plataforma: Publicaremos nuestro juego para que los alumnos puedan utilizarlo, lo cual permitirá recibir feedback de su funcionamiento permitiendo así solventar posibles errores e implementar futuras mejoras.

2.3.5 Plataformas gamificación e-learning ya existentes

En la actualidad, existen algunas plataformas online que utilizan la gamificación en la enseñanza, como por ejemplo Gametize, Kahoot o Duolingo. En el caso de Gametize o Kahoot, se trata de plataformas con herramientas que permiten al profesorado crear sus propios tests o ejercicios para los alumnos para que se adapten al formato de juego que proponen en estas plataformas. En el caso de Duolingo hablamos de una aplicación de aprendizaje de idiomas mediante el uso de recompensas y notificaciones para motivar al alumno.

En nuestro caso, crearemos nuestra propia aplicación basándonos en la idea de gamificación que utilizan estas plataformas. Pero, a diferencia de estas, nuestra aplicación busca que no sea necesaria la participación del profesor y que nuestra IA sea la encargada de generar y adaptar el código para cada alumno. Además, en nuestro caso no se hará uso de texto al uso, sino de

bloques de código en diferentes lenguajes al estar nuestra aplicación orientada al aprendizaje y práctica con código de programación.

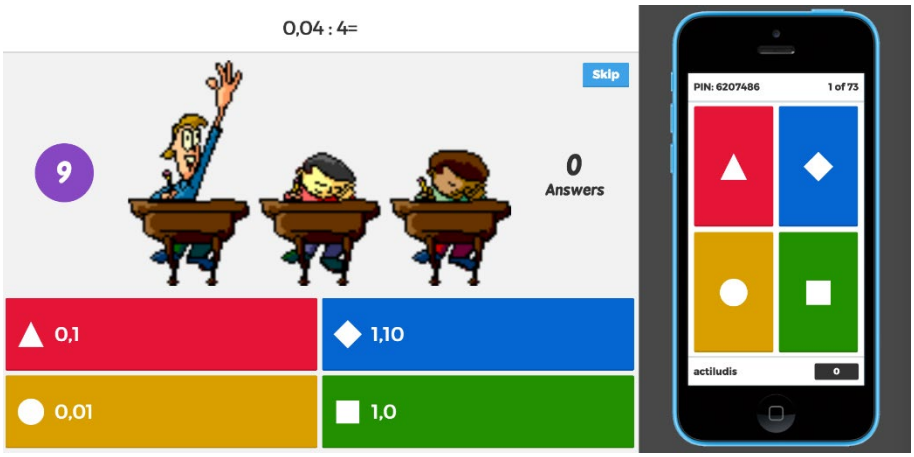

Figura 5. Ejemplo de la plataforma Kahoot

https://www.actiludis.com/2017/02/15/kahoot-tablas-multiplicar-extendidas-i/

3. Diseño

3.1 Requisitos y especificaciones

Nuestro proyecto tiene como objetivo principal la creación de una aplicación que permita a los usuarios mejorar su habilidad con el desarrollo en diferentes lenguajes mediante la práctica de dicho lenguaje. Para ello se utilizará un algoritmo basado en redes neuronales que generará fragmentos de código. El usuario tendrá que resolver y responder cual es la salida por pantalla que devuelve dicho código.

Con la intención de cumplir dicho objetivo se definen los siguientes requisitos y especificaciones del proyecto:

- Crear una red neuronal capaz de generar fragmentos de código utilizando código y datos obtenidos de forma ética y respetando los derechos de autor.
- Integrar dicha generación automática de código en un entorno web con un rendimiento aceptable y de forma invisible al usuario.
- Para este proyecto el número de lenguajes integrados será de dos (Python y Java).
- También se integrarán tres áreas de aprendizaje para cada lenguaje, teniendo cada área dos o más niveles de complejidad.
- En nuestro proyecto quedará establecida una base que permita al desarrollador integrar nuevos lenguajes, áreas y dificultades con relativa facilidad.
- La aplicación web tendrá sus textos en varios lenguajes (español, catalán e inglés).
- Todo nuestro diseño e implementación estará basado en software libre.

3.2 Descripción del proyecto

Nuestro proyecto está separado en dos bloques principales:

- Red neuronal
- Servicios web

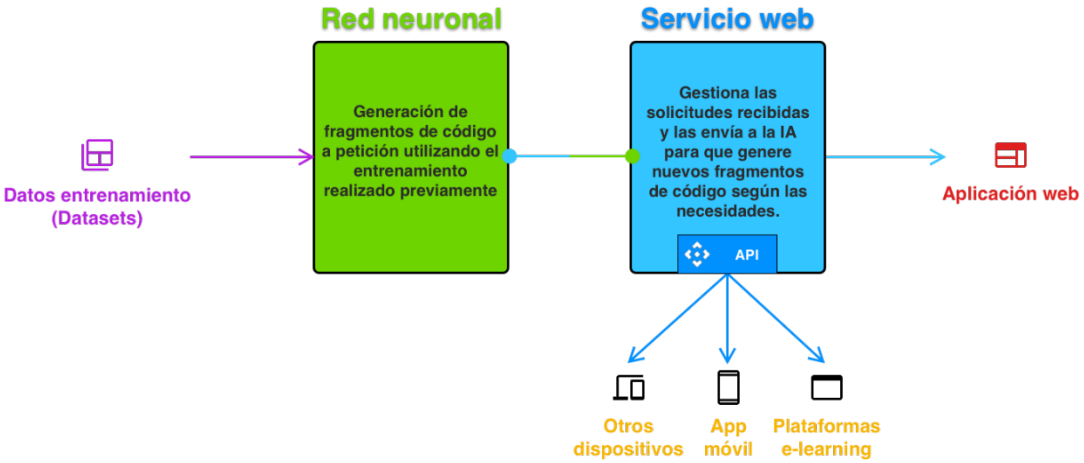

Figura 6. Bloques y elementos que componen el sistema

Como observamos en la figura nuestro servicio web se puede dividir en dos bloques dependiendo de la plataforma de origen de la solicitud por parte del usuario. En nuestro caso desarrollaremos una aplicación web con el generador de código integrado en la misma, por lo que no hará falta hacer uso de la API.

El funcionamiento de nuestro sistema sería el siguiente:

- El usuario accede a la aplicación y selecciona el lenguaje y el nivel que quiere practicar.
- El servicio web envía al servidor los datos seleccionados por el usuario. Estos datos son procesados y utilizados para generar un fragmento de código adecuado a la solicitud del usuario.

- El servidor devuelve a la aplicación el fragmento de código. Esté código es procesado y se muestra por pantalla con 4 posibles respuestas.

A continuación, se muestra cómo sería dicho flujo de datos:

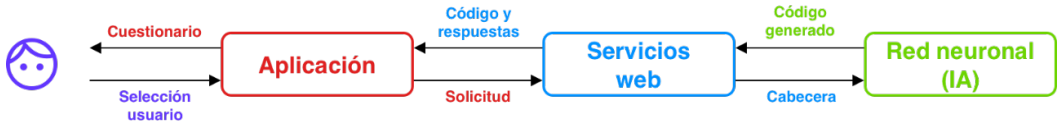

Figura 7. Flujo de datos del sistema

3.3 Red neuronal (IA)

Como hemos indicado anteriormente nuestros fragmentos de código serán generados por una inteligencia artificial al que previamente habremos entrenado con código adecuado.

Aunque actualmente existen múltiples repositorios con grandes cantidades de código disponibles, ninguno nos asegura que dicho código haya sido obtenido de forma lícita o todo su contenido sea de uso libre.

Por esto, hemos decidido que utilizaremos un algoritmo para generar nuestros propios *datasets* de código de forma aleatoria. Esto nos supone dos principales ventajas:

- Tendremos la certeza absoluta que todo el código que utilizaremos para entrenar nuestra IA es de uso libre y está libre de errores.
- Podremos adaptar los datos de entrenamiento a un formato que nos sea más adecuado para nuestras necesidades. De esta forma, será más alta la probabilidad de obtener código correcto y adecuado al nivel cuando la IA devuelva los fragmentos que solicitemos.

3.3.1 GPT (Generative Pre-Trained Transformer)

Nuestra red neuronal estará basada en el modelo GPT para generar textos coherentes según un contexto dado. GPT se basa en aprendizaje automático y procesamiento del lenguaje natural basado en estadísticas. El modelo GPT no tiene comprensión real del significado de dicho lenguaje. Es por esto por lo que lo utilizaremos para nuestra generación de código, ya que, a efectos prácticos, nuestro código y el lenguaje natural es indiferente para este modelo.

Para entrenar este modelo se le proporciona gran cantidad de textos de diversas fuentes y este aprende a capturar patrones y estructuras. Con esto es capaz de predecir la siguiente palabra o párrafo basándose en un contexto, de forma que cuando se le proporciona un contexto inicial este modelo utiliza dicho aprendizaje para general el texto que sigue a dicho contexto.

Nosotros utilizaremos estas posibilidades creando nuestros propios textos (código) de entrenamiento adaptados a nuestras necesidades, es decir, código de diferentes lenguajes, áreas y dificultades.

3.3.2 Creación de datos para entrenamiento

Para generar los *datasets* que utilizaremos en el entrenamiento de la IA hemos desarrollado un algoritmo mediante el cual se generarán fragmentos de código con el siguiente formato:

```
@tipo_nivel-complejidad@
Fragmento de código
```

Este formato se ha decidido por una razón importante ya que nuestra IA está desarrollada pensando en el entrenamiento con textos con lenguaje natural dividido en párrafos.

Así pues, nosotros crearemos una «imitación» de un tipo de textos naturales como son los guiones, donde cada párrafo tiene dos partes: Una cabecera con el nombre de personaje que habla seguido de dos puntos (:) y justo debajo un texto con sus líneas. En nuestro caso la cabecera indicará la solicitud del usuario con el lenguaje, tipo y nivel a practicar, y las líneas de texto será el código correspondiente.

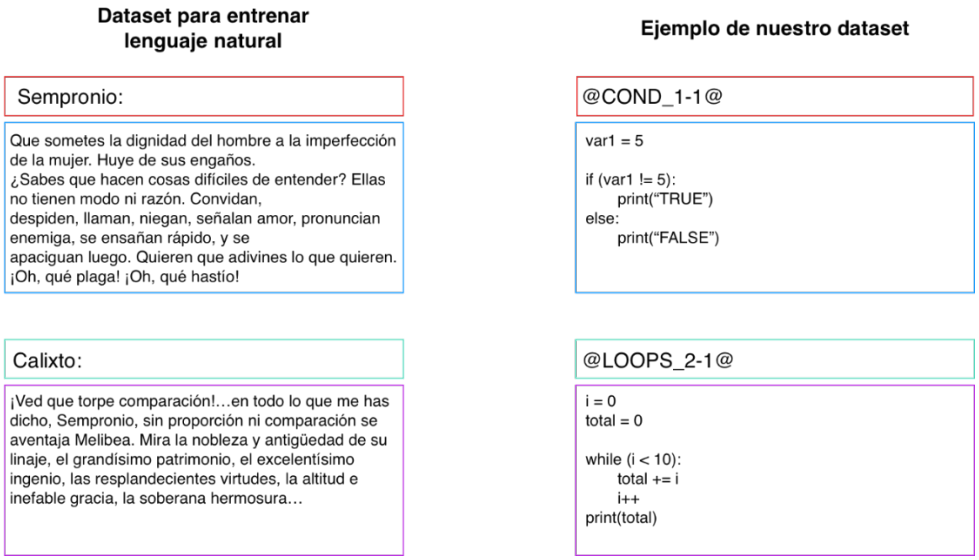

Figura 8. Comparación entre dataset de lenguaje natural y nuestro dataset

De esta forma, podremos solicitar a nuestra red que genere código adecuado a la solicitud solo indicándole sobre que «personaje» será sobre el que tiene que generar un párrafo de texto, y nuestra IA nos generará un posible texto (código en nuestro caso) que es el que corresponde a dicha cabecera.

Teniendo en cuenta todo esto, el algoritmo que generara un archivo CSV donde estarán los datos de entrenamiento que utilizaremos para general el modelo IA de cada lenguaje.

La entrada de nuestro algoritmo será el número de muestras diferentes que queremos generar para nuestro *dataset*. Este valor dependerá de lo complejo y variado que queremos que sea el modelo y los fragmentos de código que genera nuestra IA.

3.4 Aplicación web

Para la aplicación crearemos una web sencilla que nos permita con pocas pantallas y una interfaz simple utilizar los datos generados por la IA para hacer cuestionarios y guardar una puntuación que permita al usuario ver su progreso.

Utilizaremos el *framework Laravel* para desarrollar dicha web, ya que además de ser un *framework* de código abierto tiene otras ventajas que pueden resultar interesantes:

- Está basado en arquitectura MVC (Modelo-Vista-Controlador), lo cual nos permitirá separar de forma clara la parte lógica (Generación de código mediante IA) de la parte de presentación (Interfaz de usuario y gamificación) de forma mucho más clara.
- Tiene gran cantidad de librerías y paquetes de fácil integración que podremos utilizar para comunicar los algoritmos de generación de código (Python) con nuestro back-end (PHP).
- Dispone de un sistema sencillo y rápido para la creación de API que nos permitirá comunicar nuestro proyecto con otras plataformas.

3.5 API

Por último, también desarrollaremos una API en nuestro servicio web para poder hacer solicitudes de código a nuestra IA desde otras plataformas que no sean nuestra aplicación web, esto permitirá utilizar nuestra IA en otros proyectos o en aplicaciones móviles.

4. Implementación

4.1 Pasos previos y preparación del entorno de trabajo

Para el desarrollo de nuestra IA hemos realizado varias pruebas teniendo en cuenta que no se necesitaba una gran potencia para nuestro proyecto y que los recursos son limitados. Finalmente se ha optado por utilizar como base para nuestra red neuronal una variante de código libre llamada nanoGPT la cual permite entrenar un modelo basado en GPT para textos de tamaño medio.

Tanto para la creación de los archivos con los datos de entrenamiento como para la creación del modelo IA ha sido necesaria la instalación de la versión 3.10 de Python, además de las librerías *pytorch*, *numpy*, *transformers* y *pandas*. Dicha instalación y todo el desarrollo de la IA ha sido realizado con el sistema operativo Windows 10. También ha sido necesario la instalación de la herramienta *CUDA* de Nvidia para poder entrenar nuestra IA utilizando una tarjeta gráfica y reducir así significativamente los tiempos de entrenamiento necesarios.

Para el desarrollo de la aplicación web hemos necesitado la instalación de una máquina virtual (VirtualBox + Vagrant) en la que hemos instalado el paquete Laravel Homestead en su versión 10 para desarrollar la aplicación web. También ha sido necesaria la instalación del paquete «Symfony/Process» para la ejecución de código Python desde PHP. Toda esta instalación y el desarrollo de la aplicación web se ha realizado con el sistema operativo macOS Ventura 13.

La implementación de todo este proyecto se puede dividir en los siguientes apartados teniendo en cuenta los entregables indicados en el primer apartado:

- Generación de código mediante IA:
- Algoritmo para generar *datasets*
- Creación de un modelo IA mediante entrenamiento
- Pruebas de generación de código utilizando el modelo
    - Aplicación web (*Back-end*):
- Ejecución de código Python desde Laravel (PHP)
- Solicitud para generación de código
- Solicitud de código y generación de respuestas
    - Aplicación web (*Front-end*):
- Selector de lenguajes y niveles
- Cuestionarios
- Almacenamiento de datos de puntuación y nivel
    - API

4.2 Generación de código mediante IA

4.2.1 Algoritmo para generar *datasets*

Como vamos a generar nuestros propios conjuntos de datos con los que entrenar nuestro modelo necesitaremos ser capaces de generar en poco tiempo una gran cantidad de datos que nos permita que este sea capaz de aprender los patrones y poder reproducirlos prediciendo cada palabra.

Para esto vamos a crear un script en Python que contenga fragmentos de código «muestra» cuyos valores aleatorizaremos en cada iteración de un bucle para tener una gran cantidad de datos diferentes con los que entrenar nuestra IA.

Este bucle que contiene los códigos «muestra» realizará un número de iteraciones que decidiremos nosotros. Tendremos en cuenta que a mayor número de iteraciones mayor tamaño ocupará nuestro CSV y más lento será el proceso de preparación de los datos. Cada fragmento de código estará precedido por su correspondiente cabecera indicando a que tipo, nivel y complejidad pertenece.

Por ejemplo, uno de estos «código muestra» sería el siguiente:

```
### POO_1-1 ###
tipo = "POO"
nivel = 1
complejidad = 1
code = (f"""@{tipo}_{nivel}-{complejidad}@
class Punto:
  def __init__(self, x, y):
    self.x = x
    self.y = y

  def mover(self, nueva_x, nueva_y):
    self.x = nueva_x
    self.y = nueva_y

nuevo_punto = Punto({random.randint(0, 10)}, {random.randint(0, 10)})
nuevo_punto.mover({random.randint(0, 10)}, {random.randint(0, 10)})
print(f"(¿[nuevo_punto.x]?, ¿[nuevo_punto.y]?")
""")
```

Figura 9. Código muestra para la creación de un dataset

De esta forma se creará, en cada iteración un conjunto de fragmentos de código, cada uno encabezado por su tipo, nivel y complejidad, que guardaremos en un array.

Podemos observar que algunos símbolos como “{“ han sido sustituidos por “¿[“. Esto se debe a que a la hora de crear los fragmentos de código maestro estos símbolos son detectados por Python como indicadores de un parámetro, que no es nuestro caso, por lo tanto, debemos «codificarlos» con

otros símbolos diferentes y una vez generado el código serán «decodificados» a su valor original

Finalmente, antes de incluirlos en el archivo CSV el orden de los fragmentos en dicho array será aleatorizado para que todos los fragmentos no sigan siempre el mismo orden, como ocurriría con los párrafos en un guion de texto natural.

Utilizando este script hemos generado 10.000 iteraciones de código, con 35 fragmentos modelo en cada iteración. Por lo que entrenaremos nuestro modelo IA utilizando 350.000 fragmentos diferentes de código en total.

4.2.2 Creación de un modelo IA mediante entrenamiento

Antes de empezar a entrenar nuestro modelo, tendremos que convertir el archivo CSV generado con el script anterior en un conjunto de «párrafos» que conformen un texto largo, ya que este tipo de modelos trabajan con textos, y que será el que utilizaremos para entrenar nuestra IA. Una vez convertido el contenido del *dataset* en un único texto compuesto por todos los fragmentos de código generado, lo siguiente será preparar los datos.

Para preparar estos datos utilizaremos un script que separará estos datos en dos grupos. Un primer grupo denominado «Datos de entrenamiento» que será el que utilizaremos para entrenar y generar nuestro modelo. Y un segundo grupo, denominado «Datos de validación», que será con el que iremos comprobando la desviación de nuestro modelo respecto de los datos.

Algo que hay que tener en cuenta a la hora de entrenar el modelo es que, en nuestro caso, esta desviación en la predicción no es importante, puesto que los fragmentos de código que genere la IA van a ser siempre variaciones y no van a ser completamente iguales a lo esperado, puesto que es lo que

buscamos, generar un código que tenga una estructura similar, pero con pequeñas variaciones.

Para nuestro proyecto, tras haber realizado múltiples pruebas, algunos de los parámetros que hemos configurado y que resulta interesante indicar son los siguientes:

- *block_size*: Con este valor indicamos al modelo la longitud máxima que será utilizada para evaluar los datos, es decir, el número de palabras totales por delante y por detrás de cada palabra que analizará para obtener un patrón. En nuestro caso hemos utilizado un valor de 32, ya que necesitamos que analice gran parte del código alrededor de cada palabra para obtener buenos resultados.
- *n_layer*: Índica el número de capas que tendrá nuestra red neuronal. Incrementar este valor mejora los resultados de nuestro modelo, sin embargo, también incrementa considerablemente el tiempo de entrenamiento. En nuestro caso hemos utilizado un valor de 4 ya que obtenemos un buen resultado con un tiempo de entrenamiento razonable.
- *max_iters*: Este valor indicará en número máximo de iteraciones que haremos entrenando nuestro modelo. Para nuestro modelo se han utilizado 3000 iteraciones.

4.2.3 Pruebas de generación de código utilizando el modelo

Para generar los fragmentos de código con nuestro modelo IA entrenado habrá que indicar a este una cabecera de inicio según el tipo de código que deseemos generar. Esto lo haremos utilizando parámetros de entrada por línea de comandos (*python generarCodigo.py –start=”@POO_1-1@”*).

Además, habrá que proporcionar al script otros parámetros de entrada para configurar la salida. A continuación, indicamos aquellos que consideramos importantes:

- *max_new_tokens*: Índica el número de máximo de tokens (palabras) que deseamos generar. Usaremos un valor alto (300) para que nos permita generar cualquiera de los códigos posibles sin cortarlo. Esto generará en algunas ocasiones código «sobrante», pero lo eliminaremos como se indicará más adelante.
- *temperature*: Este valor indica cuanto queremos que se ciña a los datos las predicciones de nuestro modelo, siendo cerca de cero muy conservadoras, y por encima de 1 muy creativas y aleatorias. En nuestro caso un valor de 0,9 nos proporciona resultados variados y sin errores en la mayoría de los casos.
- *seed*: Este valor se utiliza para generar cada respuesta, permitiendo así obtener la misma respuesta si se utiliza la misma semilla. Como en nuestro caso queremos generar respuestas diferentes cada vez vamos a establecer un valor aleatorio entre 1 y 2000 cada vez que generemos un fragmento de código.

Tras ejecutar nuestro modelo debemos extraer el párrafo generado (ya que habrá código inservible debido al exceso de tokens) y cambiar algunos símbolos que hemos utilizado en nuestro *dataset* para evitar errores.

A continuación, se muestran como ejemplo algunos resultados obtenidos utilizando como parámetro de entrada *"@POO_1-1@"* con el modelo de lenguaje Python:

```
Overriding: out_dir = out-TFG
Overriding: start = @POO_1-1@
number of parameters: 7.23M
No meta.pkl found, assuming GPT-2 encodings...
```

```
@POO_1-1@
class Punto:
        def __init__(self, x, y):
                self.x = x
                self.y = y

        def mover(self, nueva_x, nueva_y):
                self.x = nueva_x
                self.y = nueva_y

nuevo_punto = Punto(8, 8)
nuevo_punto.mover(6, 8)
print(f"(¿[nuevo_punto.x]?, ¿[nuevo_punto.y]?")


@bucle_2-2@
numeros = [3,5,5,9,10]
total = 0
```

Figura 10. Salida obtenida de nuestro modelo basado en Python

Como podemos observar, el código viene con la cabecera incluida (`Overriding: out_dir…`) y fragmentos de otro código inservible (`@bucle_2-2@ …`). Por ello, hemos creado la función «*cleanCode*» que permite obtener el fragmento de código sin cabeceras, con los símbolos correctos y listo para utilizar por nuestra aplicación.

```
def cleanCode(codigo):
  # Extraemos el primer párrafo
  codigo = codigo.split('\n\n\n')
  # Eliminamos la cabecera
  codigo = re.sub(r'@.*?@\n', '', codigo[0])
  # Sustituimos los símbolos
  codigo = re.sub(r'\¿\[', '{', codigo)
  codigo = re.sub(r'\]\?', '}', codigo)
  # Devolvemos el código limpio
  return codigo
```

Figura 11. Algoritmo de limpieza de código

4.3 Aplicación web (*Back-end*)

4.3.1 Ejecución de código Python desde Laravel (PHP)

Para poder solicitar a nuestro modelo fragmentos de código tendremos que poder ejecutar el script de Python en nuestro servidor desde PHP. Para esto vamos a utilizar el paquete «*Symfony/Process*», el cual nos permite ejecutar código Python desde nuestro servidor web y obtener su salida.

```
$process = new Process(["python3", "script.py"]);
$process->run();

if ($process->isSuccessful()) {
   $output = $process->getOutput();
} else {
   throw new ProcessFailedException($process);
}
```

Figura 12. Ejecución de código Python desde Laravel

4.3.2 Solicitud para generación de código

Una vez hemos conseguido ejecutar nuestro modelo desde nuestra web, el siguiente paso será configurar nuestro código para que nos permita enviar parámetros según sea necesario según la selección del usuario en la aplicación web.

Para esto utilizaremos el siguiente código:

```
$process = new Process(["python3", "python/generateSample.py", $input]);
$process->setWorkingDirectory("/home/vagrant/code/TFG/storage/");
$process->run();

if (!$process->isSuccessful()) {
   $code = “ERROR”;
}

$code = $process->getOutput();
$code = trim($code, "\n\n");
```

Figura 13. Ejecución del script IA utilizando parámetros de entrada

Mediante la variable "*$input*" le indicaremos, usando los argumentos de entrada por la línea de comandos, el tipo de código que queremos generar.

Además, hemos de comprobar si ha habido algún error en la ejecución del script de Python, si es así devolveremos el valor «ERROR», de lo contrario obtendremos el código que ha sido representado en pantalla, al cual le eliminaremos los saltos de página extra que hacen de separador.

4.3.3 Solicitud de código y generación de respuestas

Una vez llegados a este punto, hemos conseguido ejecutar nuestro modelo desde el servidor web indicándole que tipo de código queremos generar y que modelo usar. Ahora solo queda crear las posibles respuestas que ofreceremos al usuario. Como el código se genera de forma aleatoria no conocemos de antemano el resultado correcto de dicho código, así que lo que haremos será ejecutar el código obtenido y almacenar la respuesta que este devuelve.

Para ello, almacenaremos el código recibido en un archivo temporal, lo ejecutaremos del mismo modo que hemos hecho anteriormente con el modelo IA, almacenando el resultado en el primer elemento de un array que contendrá las posibles respuestas:

```
$respuestas = [];

$archivo_temporal = tempnam(sys_get_temp_dir(), 'pythonCode');
file_put_contents($archivo_temporal, $codigo);

$process = new Process(["python3", $archivo_temporal]);
$process->setWorkingDirectory("/home/vagrant/code/TFG/storage/");
$process->run();

if (!$process->isSuccessful()) {
   $respuestas[0] = "ERROR";
```

```
} else {
   $respuestas[0] = $process->getOutput();
}
```

Figura 14. Ejecución del código obtenido utilizando un archivo temporal

Como podemos observar también tendremos en cuenta la posibilidad de que el código pueda devolver un error y lo utilizaremos como respuesta correcta ya que el usuario también tendrá que saber identificar errores sencillos en fragmentos de código.

Se han realizado múltiples pruebas y en todos los casos los errores son fáciles de identificar a simple vista como sería el uso de variables con nombres incorrectos, en ningún caso se genera código con errores complejos que no puedan ser identificados sin realizar algún tipo de prueba.

Por último, nos queda generar el resto de las respuestas erróneas de las 4 respuestas posibles. Como nosotros definimos los datos de entrenamiento de nuestro *dataset* sabemos de antemano el tipo de respuestas posibles que puede devolver el código generado.

Estas posibles respuestas serán un número, una letra, una opción TRUE/FALSE o ERROR. Teniendo esto en cuenta generamos el resto de las respuestas posibles de la siguiente forma:

- Si es una letra: Se rellenan el resto de las respuestas con otras letras y una opción de ERROR.
- Si es un número: Se buscan más números que aparezcan en el código y se utilizan como posibles respuestas.
- Si es TRUE/FALSE: Se añade como posible respuesta la contraria a la que aparezca.

Por último, se comprueba si el array contiene 4 valores, que será el número de posibles respuestas, y de no ser así (como ocurrirá en todos los casos excepto si es una letra la respuesta correcta) se rellenan el resto de las respuestas con valores aleatorios.

4.4 Aplicación web (*Front-end*)

4.4.1 Selector de lenguajes y niveles

Nuestra pantalla principal tendrá un estilo sencillo ya que buscamos no agobiar al usuario y motivar a este para que este empiece con la práctica sin que esto suponga un umbral a superar. Por esto, solo se le solicitará el lenguaje a practicar con sencillos botones. Además, mediante el uso de cookies se guardarán las opciones que elija el usuario para posteriores sesiones.

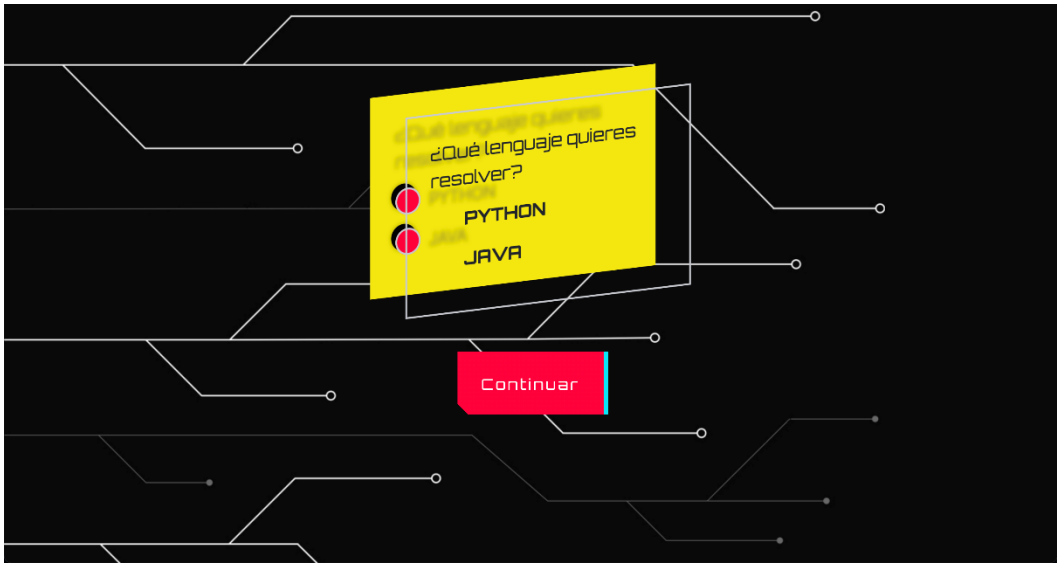

Figura 15. Pantalla de selección de lenguaje

Figura 16. Pantalla de selección de nivel

4.4.2 Cuestionarios

En el diseño de los cuestionarios se pretende crear una inmersión en el «juego» que haga que el usuario no sienta que está realizando una plantilla tipo test habitual.

Para ello, además de la temática estilo «cibernética» se ha añadido en la parte superior una barra de progreso que permitirá ver su progresión al usuario al mismo tiempo que lo animará a superar varios retos seguidos. Además, este progreso nos servirá como referencia para incrementar o disminuir la dificultad si fuera necesario.

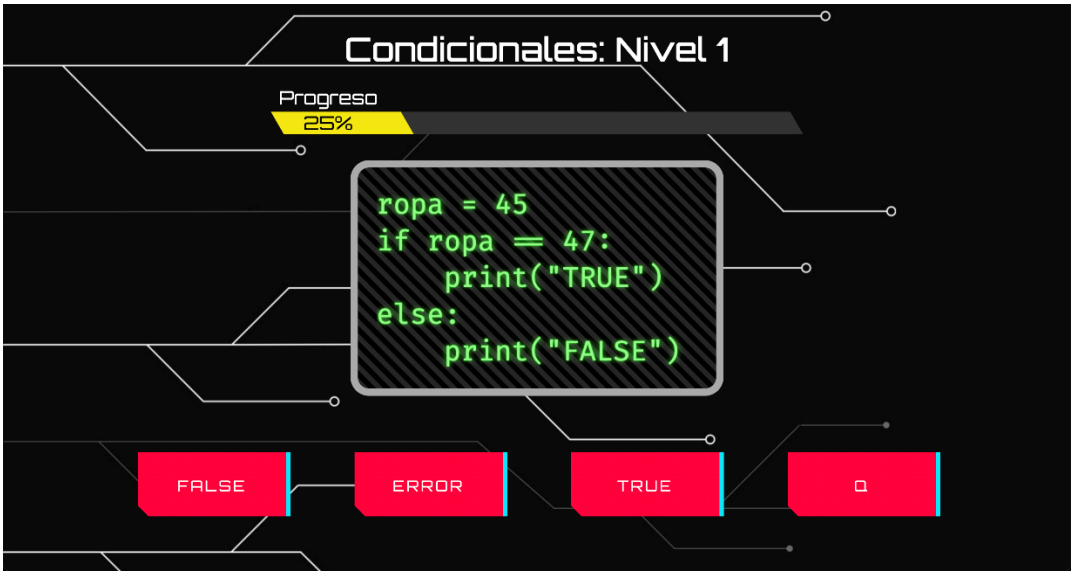

Figura 17. Pantalla de cuestionario

4.4.3 Almacenamiento de datos de puntuación y nivel

El alcance de nuestro proyecto no incluye la creación de una base de datos de usuarios que permita guardar los progresos de cada persona. Sin embargo, en nuestra aplicación web vamos a integrar un sencillo sistema de almacenamiento de la progresión de forma local mediante el uso de cookies, de forma que no sea necesario crear usuarios ni utilizar una base de datos, pero permita tomar descansos y continuar el aprendizaje posteriormente, siempre que se utilice el mismo equipo y navegador.

4.5 API

Con la intención de poder utilizar el código generado por nuestra IA desde otras plataformas diferentes a nuestra aplicación web hemos desarrollado una sencilla API que con una sencilla petición GET nos permite obtener el código generado y las posibles respuestas.

Para esto hemos declarado dos rutas posibles, "*/api/getCode*" que solo devuelve el fragmento generado sin respuestas para poder generar tus propias respuestas y "*/api/getTest*" que devuelve el código y las respuestas generadas. En ambos casos habrá que incluir en la petición los parámetros *modelo*, *tipo*, *nivel* y *dificultad*. Utilizando la aplicación *Postman* hacemos las peticiones GET a la API y obtenemos el siguiente resultado:

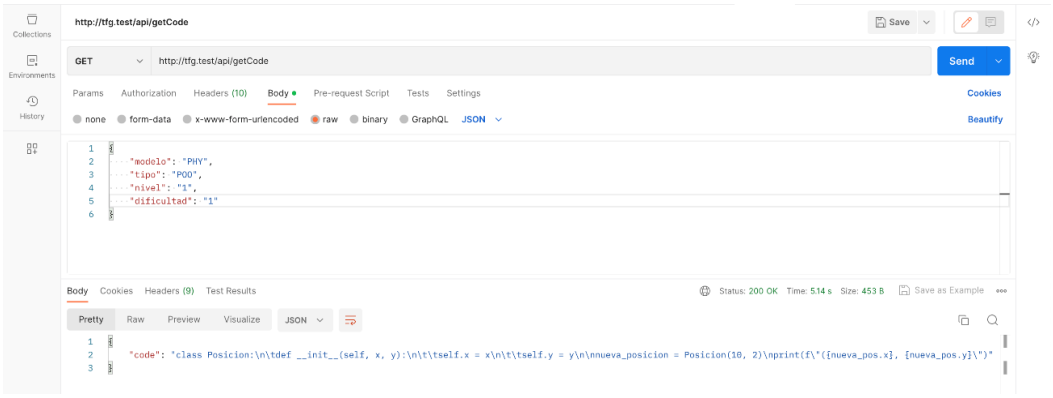

Figura 18. Petición GET a /api/getCode

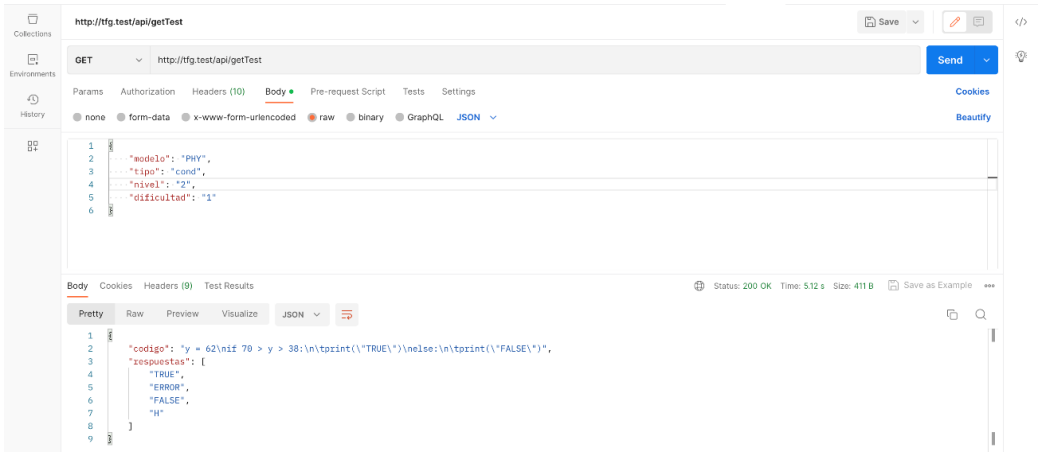

Figura 19. Petición GET a /api/getTest

De esta forma podemos obtener de forma sencilla el código y las respuestas para integrarlas en cualquier plataforma o aplicación móvil utilizando nuestra API.

5. Conclusiones y trabajos futuros

5.1 Conclusiones

Al finalizar este proyecto se han alcanzado los objetivos propuestos al inicio del curso. Se ha logrado el objetivo principal, desarrollar una IA que permita generar estructuras y fragmentos de código aleatorio, pero funcionales, los cuales serán más variados que los fragmentos de código predefinidos que encontramos en los habituales cursos de programación.

Además, se ha desarrollado una aplicación web que permite su ejecución y uso desde cualquier dispositivo que disponga de un navegador web y con un bajo uso de recursos ya que todas las operaciones se realizan en el servidor web. Esto permitirá su uso sin tener en cuenta las limitaciones de recursos de cada usuario o dispositivo.

Por último, se ha creado un entorno diferente al cuestionario básico de respuestas múltiples que normalmente encontramos en este tipo de cursos y la posibilidad de progresar y almacenar dicho progreso para motivar al usuario y facilitar el aprendizaje.

5.2 Opciones de actualización futuras

Como hemos visto se han conseguido alcanzar los objetivos especificados al inicio del proyecto. Sin embargo, existen características y opciones de mejora que no han sido incorporadas debido a la limitación de plazo para realizar este proyecto por lo que podrán ser añadidas en futuras versiones de este:

- Creación de aplicación para dispositivos móviles haciendo uso de la API desarrollada.
- Adaptación de los *datasets* para añadir nuevos lenguajes de programación.
- Añadir nuevos idiomas a los textos de la aplicación web/app.
- Añadir nuevos tipos de estructura de código con menor y mayor complejidad de los ya disponibles.
- Creación de una base de datos de usuarios que permita conservar el progreso realizado independientemente de la plataforma en que se utilice.
- Profundizar en la gamificación y mejorar los incentivos añadiendo más opciones como tabla de puntuaciones, trama o retos basados en tiempo.
- Añadir un tutorial y ayuda para facilitar el uso de la aplicación.
- Añadir opciones de accesibilidad para usuarios con necesidades especiales.

6. Glosario

E-learning: Actividad de formación a través de un dispositivo conectado a la red.

M-learning: Actividad de formación utilizando dispositivos móviles como *smartphones* o *tablets*.

IA (Inteligencia artificial): Conjunto de algoritmos informáticos que simulan los procesos de inteligencia humana.

Machine learning: Algoritmos y modelos informáticos capaces de llevar a cabo tareas sin instrucciones explicitas.

Red neuronal: Modelo computacional inspirado en el cerebro humano que permite crear una red de reconocimiento, clasificación y predicción de patrones.

GPT (Generative Pre-trained Transformer): Red neuronal utilizada para el procesamiento de lenguaje natural y generación automática de texto.

API (Application Programming Interface): Conjunto de protocolos y definiciones que permiten comunicar dos plataformas diferentes a través de la red.

Dataset: Conjunto de datos almacenados de forma estructurada.

CSV (Comma Separated Values): Archivo cuyos valores están separados por comas.

Laravel: Framework para el desarrollo de web con PHP.

MVC (Modelo Vista-Controlador): Arquitectura software que divide el código según su uso.

Back-end: Códigos y algoritmos de una web que se ejecutan en el servidor donde se aloja dicha web.

Front-end: Código y algoritmos de una web que se ejecutan en el dispositivo del usuario.

App: Aplicación informática destinada a su uso en móviles, *tablets* u otro dispositivo portátil.

7. Bibliografía

[1] Núñez Artalejo, L.M. (2022) 'Form4Press, Diseño de una plataforma e-learning'. Trabajo final de grado. Universitat Oberta de Catalunya. Disponible en: https://openaccess.uoc.edu/handle/10609/145847

[2] Bermejo Guerrero, C.J. (2018) 'Diseño de una aplicación móvil para potenciar la lectura infantil a través de cuentos interactivos'. Trabajo final de master. Universitat Oberta de Catalunya. Disponible en: https://openaccess.uoc.edu/handle/10609/72905

[3] Galiana Rubio, R. (2020) 'El juego de la ciberseguridad: Securiza2'. Trabajo final de master. Universitat Oberta de Catalunya. Disponible en: https://openaccess.uoc.edu/handle/10609/118986

[4] 'Wikipedia: GPT-3'. Disponible en: https://en.wikipedia.org/wiki/GPT-3

[5] 'Understanding GPT-3: OpenAI's Latest Language Model'. Gaurav Shekhar, 1 septiembre 2020. Disponible en: https://medium.com/swlh/understanding-gpt-3-openais-latest-language-model-a3ef89cffac2

[6] 'Building the Machine Learning Infrastructure'. Pat Alvarado', 20 junio 2017. Disponible en: https://www.teradata.com/Blogs/Building-the-Machine-Learning-Infrastructure

[7] 'Tendencias en Inteligencia Artificial en 2023: el boom de Stable Difussion o ChatGPT'. Manuel Arenas, 27 de enero 2023. Disponible en: https://www.noticias3d.com/articulo/3515/p2/tendencias-inteligencia-artificial-2023-boom-stable-difussion-o-chatgpt.html

[8] 'GitHub - karpathy/minGPT: A minimal PyTorch re-implementation of the OpenAI GPT (Generative Pretrained Transformer) training'. Andrej Karpathy, 8 de enero 2023. Disponible en: https://github.com/karpathy/minGPT

[9] 'GitHub - karpathy/nanoGPT: The simplest, fastest repository for training/finetuning medium-sized GPTs.'. Andrej Karpathy, 8 de enero 2023. Disponible en: https://github.com/karpathy/nanoGPT

[10] 'Introducing ChatGPT'. Disponible en: https://openai.com/blog/chatgpt
'¿Cuáles son los tipos de algoritmos del machine learning?'. Redacción APD, 4 de abril 2019. Disponible en: https://www.apd.es/algoritmos-del-machine-learning/

[11] 'Can GPT-3 write an academic paper on itself, with minimal human input?'. Gpt Generative Pretrained Transformer, Almira Osmanovic Thunström,Steinn Steingrimsson. 21 de junio 2022. Disponible en: https://hal.science/hal-03701250/document

[12] 'GPT-3, un algoritmo que elabora una investigación científica en 2 horas'. Blanca Montoya Gago. Julio 2022. Disponible en: https://blogthinkbig.com/gpt-3-algoritmo-inteligencia-artificial-crea-investigacion-cientifica

[13] 'How ChatGPT works and AI, ML & NLP Fundamentals'. Alexandru Hutanu. 8 de febrero 2023. Disponible en: https://www.pentalog.com/blog/tech-trends/chatgpt-fundamentals/

[14] 'GitHub Copilot'. Disponible en: https://github.com/features/copilot

[15] 'GitHub - openai/gpt-2: Code for the paper Language Models are Unsupervised Multitask Learners'. Disponible en: https://github.com/openai/gpt-2

[16] 'minGPT – How It Works'. ZORODAREVOHNCODER. 16 de junio 2022. Disponible en: https://www.signalpop.com/2022/06/16/mingpt-how-it-works/

[17] 'GitHub Copilot: Qué es y primeras impresiones'. Pablo Huet. 27 de septiembre 2021. Disponible en: https://openwebinars.net/blog/github-copilot-que-es-y-primeras-impresiones/

[18] 'minGPT VS nanoGPT'. Disponible en: https://www.libhunt.com/compare-minGPT-vs-nanoGPT

[19] 'Summary of the tokenizers'. Disponible en: https://huggingface.co/docs/transformers/tokenizer_summary

[20] 'Sensio CoPilot'. Sensio coders. 28 de diciembre 2021. Disponible en: https://www.sensiocoders.com/blog/085_sensio_copilot

[21] 'La Gamificación e-Learning Paso a Paso'. ITMadrid. 14 de octubre 2022. Disponible en: https://www.itmadrid.com/la-gamificacion-e-learning-paso-a-paso/

[22] 'Gamificación en el aula: ventajas y desventajas'. Smartmind. Disponible en: https://www.smartmind.net/blog/gamificacion-en-el-aula-ventajas-y-desventajas/

[23] 'Gamificación: el aprendizaje divertido'. Virginia Gaitán. Disponible en: https://www.educativa.com/blog-articulos/gamificacion-el-aprendizaje-divertido/

[24] Afonso Folhento, M.P. (2022) UOC Assistant'. Trabajo final de grado. Universitat Oberta de Catalunya. Disponible en: https://openaccess.uoc.edu/handle/10609/132727

[25] 'Understanding GPT-3: OpenAI's Latest Language Model'. Gaurav Shekhar, 1 septiembre 2020. Disponible en: https://medium.com/swlh/understanding-gpt-3-openais-latest-language-model-a3ef89cffac2

[26] 'Laravel - The PHP Framework For Web Artisans' Disponible en: https://laravel.com/

[27] 'Python.org' Disponible en: https://www.python.org/

[28] 'Summary of the tokenizers'. Disponible en: https://huggingface.co/docs/transformers/tokenizer_summary

[29] 'Sensio CoPilot'. Sensio coders. 28 de diciembre 2021. Disponible en: https://www.sensiocoders.com/blog/085_sensio_copilot

[30] 'La Gamificación e-Learning Paso a Paso'. ITMadrid. 14 de octubre 2022. Disponible en: https://www.itmadrid.com/la-gamificacion-e-learning-paso-a-paso/

[31] Lupton, Ellen. El diseño como storytelling (Editorial GG), 2019. ISBN:9788425231865. Disponible en: https://editorialgg.com/el-diseno-como-storytelling-libro.html