ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Master

DISEÑO DE UNA APLICACIÓN EN UNITY PARA LA ASIGNATURA "ENERGÍA Y TELECOMUNICACIONES"

(Design of an application for the course "Energy and Telecommunications" in Unity)

Para acceder al Título de

Master en
Ingeniería de Tecnologías de Telecomunicación

Autor: Francisco Javier Miguéns Santos

NOVIEMBRE– 2021

E.T.S. DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

MASTER EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE MASTER

Se hace constar que este trabajo es soportado por el proyecto PID2019-107270RB-C21 financiado por MCIN/ AEI /10.13039/501100011033.

Realizado por: Francisco Javier Miguéns Santos
Director del TFG: Jesús Mª Mirapeix Serrano

Título: "Diseño de una aplicación en Unity para la asignatura "Energía y Telecomunicaciones"."

Title: "Design of an application for the course "Energy and Telecommunications" in Unity"

Presentado a examen el día: 02 de Noviembre de 2021

para acceder al Título de

MASTER EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Composición del Tribunal:

Presidente (Apellidos, Nombre): Adolfo Cobo García.
Secretario (Apellidos, Nombre): Beatriz Aja Abelán.
Vocal (Apellidos, Nombre): Ramón Agüero Calvo.

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del TFG

VºBº del Subdirector

Trabajo Fin de Grado N.º
(A asignar por Secretaría)

AGRADECIMIENTOS

Quisiera utilizar estas líneas para manifestar mi agradecimiento a algunas personas cuya colaboración fue indispensable en la elaboración de este proyecto de fin de máster.

En primer lugar, a mi director de proyecto, Jesús Mirapeix Serrano, por su dedicación y ayuda siempre que la he requerido.

Agradecer también a mis padres, y a mi hermana porque son parte importante de mis decisiones, y sin ellos esto no hubiera sido posible.

Por otro lado, a mis amigos, esas personas que forman parte de mi día a día, que me han apoyado y animado en todo momento.

Y por último, al mas importante de todos, a la persona que mas ha confiado en mí y más orgulloso estaría por lo que he conseguido, a mi Abuelo ANGEL SANTOS ALLOZA.

RESUMEN

Este proyecto plantea el diseño de una aplicación a través de Unity para la asignatura de “Energía y Telecomunicaciones”. Su principal objetivo es servir como instrumento de gamificación para ayudar a mejorar el aprendizaje sobre las energías renovables.

En primer lugar, el usuario debe registrarse para poder acceder a la aplicación, por lo que también entra en juego las bases de datos. La aplicación consiste en un Quiz donde el alumno tiene que ir respondiendo correctamente las preguntas para conseguir una bonificación. Por otro lado, el alumno puede personalizar sus partidas en función del número de preguntas que quiere responder, el número de vidas (fallos permitidos) y el tiempo del que dispone para responder las preguntas. A demás el alumno tiene un apartado de entrenamiento, para practicar de forma más detallada sobre una energía renovable especifica.

Con este proyecto se plantea solucionar el problema de tener una plataforma específica para reforzar los conceptos y darle un toque más lúdico.

PALABRAS CLAVES

Unity, Energías renovables, Quiz, Gamificación, Aplicación, Aprendizaje.

ÍNDICE

ILUSTRACIONES y FIGURAS

CAPÍTULO 0. Motivación y Objetivos

Se explicará la motivación para llevar a cabo el proyecto y se comentarán los objetivos que se pretenden conseguir.

0.1 Motivación

La idea del proyecto surge de la necesidad de disponer de una aplicación personalizada para utilizar como técnica de gamificación en la asignatura “Energía y Telecomunicaciones”. De esta forma obtenemos una aplicación con la que mejorar y afianzar los conocimientos a cerca de las energías renovables.

De todas las posibilidades que existen para crear una aplicación finalmente se utilizó Unity, y programada a través del software Jetbrains Rider.

Por otro lado, se valoró la oportunidad de aprender el manejo de Unity y de nuevos lenguajes de programación, en este caso el orientado a base de datos. Es decir, se han utilizado diferentes lenguajes de programación (Php y #C) y se ha perfeccionado el manejo de estos.

En definitiva, la elaboración del presente proyecto ha ofrecido un gran abanico de posibilidades de aprendizaje sobre nuevos contenidos, los cuales son muy demandados en la actualidad como puede ser la programación de diversas aplicaciones.

0.2 Objetivos

Como se ha mencionado con anterioridad la principal finalidad de este proyecto es la de conseguir una herramienta personalizada para utilizar como técnica de gamificación de tal forma que se aprenda sobre las energías renovables de forma más dinámica y rápida que la manera convencional. Cabe mencionar que la aplicación diseñada podría aplicarse fácilmente a cualquier otra temática o asignatura, precisamente por su diseño mediante el uso de bases de datos.

De esta forma el alumno a través de su rapidez y número de acierto conseguirá una bonificación que le podría permitir, por ejemplo, mejorar su calificación. Otro objetivo que se promueve es la creación de la motivación (“rivalidad sana”) entre el alumnado por ver quién es capaz de conseguir mejor puntuación. Surge así la ludificación o gamificación (“que es el uso de técnicas y elementos propios de los juegos con el fin de potenciar la motivación”).

En resumen, los objetivos principales serían:

- Como técnica de gamificación para la asignatura “Energía y Telecomunicaciones”
- Para afianzar los conceptos adquiridos en clase y acelerar su aprendizaje de manera continua.

- Se utilizará para bonificar de manera positiva al alumno que consiga mayor puntuación.
- Crear rivalidad (motivación) “sana” entre el alumnado por ver quien es capaz de conseguir la mejor puntuación.

CAPÍTULO 1. Introducción.

Se realizará una breve introducción acerca de las energías renovables.

1.1 Introducción sobre las energías renovales

Debido a que la aplicación consiste en un Quiz de energías renovables, se hará una breve introducción a esta temática:

Las energías renovables [1] son aquellas que se extraen a partir de fuentes naturales (sol, viento, agua…) capaces de producir energía de forma virtualmente indefinida e inagotable. Dentro de dichas energías encontramos la energía solar, la eólica o la mareomotriz entre otras. Por otro lado, también se consideran energías renovables cuando se adquieren a través de fuentes que se regeneran de forma natural con el tiempo, como por ejemplo la masa forestal (biomasa).

Ilustración 1: Energías renovables.[2]

Una gran ventaja que se ha nombrado anteriormente es que son inagotables, y también tienen un bajo impacto sobre el medio ambiente siendo a veces este último nulo, es decir a las energías renovables se las denominan energías limpias, por lo que no producen gases de efecto invernadero (causas principales del cambio climático) ni emisiones contaminantes. La energía renovable y limpia se denomina "energía verde". Cada vez están más presentes en la sociedad y sus beneficios para el medio ambiente son más que visibles.

Se quiera o no, la sociedad consume energía en mayor o menor medida,[3] pero el gran inconveniente es que la mayoría procede de fuentes de energía no renovables como pueden ser los combustibles fósiles o la energía nuclear, y el impacto de estas últimas en el medio ambiente es superior.

Una característica que subrayar sobre las energías renovables es que pueden ser utilizadas a nivel local, es decir las pequeñas poblaciones no tienen que depender de los grandes productores de energía, mejorando el desarrollo económico y aumentando la creación de empleo.

1.2 Ventajas e inconvenientes

A la hora de comparar [4] las energías renovables con las fuentes convencionales, se pueden enumerar los siguientes aspectos que son ventajosos para la sociedad:

Son capaces de producir energía de forma indefinida debido a que se obtienen de fuentes de energía ilimitadas e inagotables.

Por otro lado, reducen las emisiones de gases de efecto invernadero, contribuyendo de esta forma a reducir el calentamiento global. Debido a eso como se ha mencionado anteriormente se consideran "energías limpias" al ser más beneficiosas con el medio ambiente que las tradicionales.

Otra ventaja que se encuentra es la reducción en los costes de producción energética, permitiendo de esta forma crear más trabajo y se reducen la necesidad de dependencia de las grandes potencias energéticas y con los países que disponen de combustibles fósiles (como puede ser el petróleo, gas o carbón).

Pero también es importante conocer sus inconvenientes como pueden ser:

La producción no es regular, es decir no hay plazos fijos para su producción, sino que dependen de los fenómenos naturales en este caso.

También tienen un gran impacto arquitectónico en su entorno, debido a las infraestructuras construidas para poder generar dicha energía (aerogeneradores, campos solares fotovoltaicos, etc.).

1.3 Energías renovables Vs. No renovables

Cabe recordar que las energías [5] no renovales son las producidas por la combustión de combustibles fósiles (carbón, petróleo y sus derivados). Un inconveniente que tienen las energías no renovables son que las reservas de dichos combustibles son enormes, pero se localizan en cantidades limitadas.

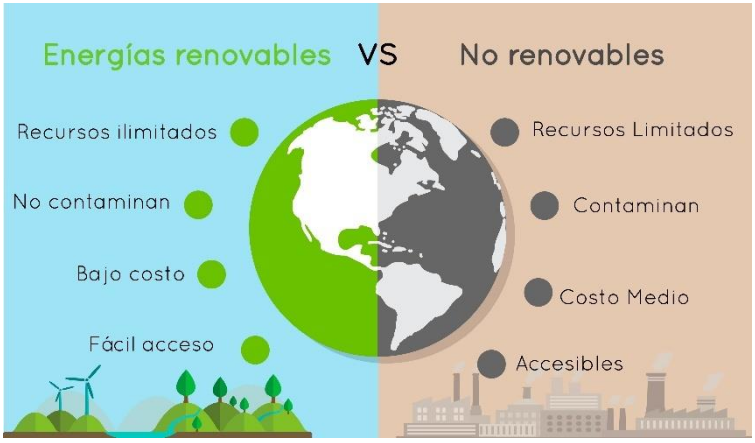

Ilustración 2: Energías renovables Vs. No renovables.[6]

Los recursos fósiles se están agotando, se cree que al ritmo de su consumo actual quedan pocos años para que desaparezcan: [7]

- Petróleo: 50 años.
- Gas natural: 60 años.

- Uranio: 100 años.
- Carbón: 110 años.

A parte de sus limitadas reservas los combustibles fósiles son la principal causa de emisiones de dióxido de carbono en España. Por lo que hoy en día sería necesario reducir la dependencia sobre los combustibles fósiles y apostar por las energías renovables, para de esta forma reducir los gases nocivos causantes del cambio climático.

1.4 Tipos de energías renovables

Hay varios tipos de energías renovables, [8] en este proyecto hablaremos de la energía solar, de la eólica, la hidroeléctrica, la geotérmica y por último la bioenergía.

1.4.1 Energía solar

Es la energía renovable procedente de la radiación electromagnética del sol. Se considera renovable debido a que se obtiene de una fuente natural e inagotable. Es una fuente de energía intermitente (no está disponible de forma continua) debido a que depende de la luz solar.

Encontramos dos tipos de energía solar: En primer lugar, está la energía solar fotovoltaica producida por instalaciones solares fotovoltaicas y por otro lado la energía solar térmica que se corresponde con el calor generado en los colectores solares.

Es una de las fuentes de energía renovables más fáciles de producir sobre todo la solar fotovoltaica.

1.4.2 Energía eólica

Por otro lado, existe la energía eólica que se produce gracias a la energía cinética del viento, gracias a las corrientes de aire. Esta energía al igual que la solar también es intermitente, ya que depende de la velocidad del viento (no siempre tenemos la misma velocidad del viento).

Hoy en día, las turbinas eólicas se instalan en la tierra o en el mar. La instalación de dichas turbinas es más sencilla en tierra que en mar, pero este último es más eficiente en producción de energía y reduce el impacto visual asociado.

Este tipo de energía renovable es de las más utilizadas en el mundo.

1.4.3 Energía hidráulica

La energía hidráulica (también conocida como energía hidroeléctrica) se produce del uso del agua mediante molinos, presas o corrientes marinas. Con el movimiento del agua, crea el impulso de las turbinas y con ello genera electricidad. A mayor presión, el agua el agua es capaz de producir mayor energía. Esto depende del caudal del rio y también de las precipitaciones.

1.4.4 Energía geotérmica

La energía geotérmica se obtiene gracias a las altas temperaturas de yacimiento bajo la superficie terrestre, produciendo energía a través del calor.

Ilustración 3: Energía Geotérmica.[9]

1.4.5 Bioenergía

La energía de la biomasa se produce gracias a la combustión de residuos orgánicos de origen vegetal o animal. Esta energía es utilizada para crear biogás, combustible utilizado para crear electricidad.

Es una de las energías más económicas y ecológicas utilizada en las centrales térmicas para generar energía eléctrica.

1.5 Energías renovables en España

En la actualidad [10] España se ha convertido en el octavo país con mayor potencia instalada de este tipo. Según los expertos se cree que las energías renovables que tienen más proyección en España son la energía eólica y la energía solar fotovoltaica.

La pandemia mundial vivida no ha sido capaz de frenar el desarrollo de estas energías, debido que en 2020 España [11] contaba con 59.108 megavatios "verdes" instalados. Pero aún queda muchos obstáculos que evitar, uno con mayor importancia es conseguir que las energías renovables no tengan únicamente como finalidad generar energía, sino que se puedan utilizar en otros

sectores como pueden ser en la industria, transportes o sistemas de calefacción de hogares.

Poco a poco se está consiguiendo que las energías renovables sean las protagonistas de la generación eléctrica. Según los últimos datos, en el mes de marzo en España aproximadamente el 52,3% de toda la electricidad producida en el país fue procedente de las energías renovables, un 15% más que en el año anterior. La fuente de energía que más aportan en España es liderada por la energía eólica con un 29,5% del total, la siguen la energía hidráulica con un 16,9%, la solar fotovoltaica con un 3,5% y la térmica con 0,5%.

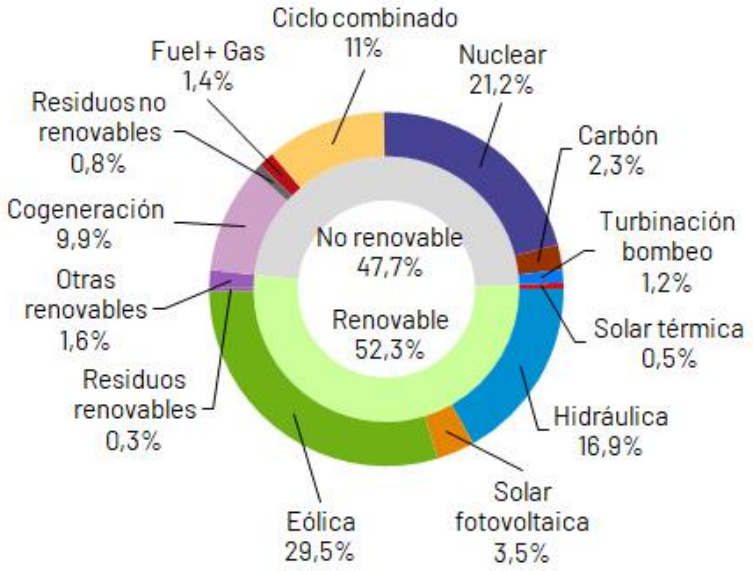

Ilustración 4: Energías presentes en España.[12]

En resumen, se podría decir que la transición energética se caracteriza por el autoconsumo, dar mayor importancia al uso de las energías renovables, no solo para producir electricidad sino incluyéndolos en nuestras vidas (transporte, industria, etc.), otro aspecto podría ser el uso de combustibles menos contaminantes.

CAPÍTULO 2. Software utilizado.

Se comentará los diversos softwares empleados para la realización del proyecto.

2.1 Unity

Se ha utilizado el Software Unity para realizar este proyecto debido a que además de ser uno de los más populares en del sector, ayuda al desarrollador en la creación de juegos multiplataforma (dispositivos móviles como para consola). Por otro lado, [13] Unity tiene una gran variedad de características que llaman la atención a la hora de desarrollar una aplicación y por consiguiente su elección. Estas características son:

- Sencilla interfaz y fácil manejo.
- Store de assets (son os elementos que componen el videojuego, es decir las animaciones, sonidos, modelos, etc.)
- Potencia en todos los entornos.
- Optimización de tiempo.
- Característica multiplataforma.

Unity es un software que se conoce [14] como motor de juegos o de desarrollo, dicho termino se le atribuye al programa que basa su funcionamiento en una serie de rutinas de programación, que permite la creación, diseño y funcionamiento de un entorno interactivo (videojuego)

Ilustración 5: Logo Unity.[15]

Las funcionalidades que podemos encontrar en el videojuego son:

- Animaciones y sonidos.
- Motor gráfico y físico (para renderizar gráficos y para simular las leyes de la física)
- Scripts o programación
- Inteligencia artificial.

UNITY ¿Para qué se usa?

Antes conocido como Unity 3D, es un software que contiene todas las funcionalidades necesarias para desarrollar videojuegos. Se podría definir como la herramienta que permite crear videojuegos para diversas plataformas como puede ser en móviles videoconsolas u ordenadores entre otros. Se pueden conseguir resultados profesionales gracias a su editor visual y su programación

mediante scripts. Se utiliza en la gran mayoría de los desarrollos de videojuegos para móviles.

Al ser un software bastante utilizado, dispone de muchos usuarios por lo que su comunidad es grande. Gracias a ello podemos encontrar multitud de información, foros, documentación y páginas web para resolver dudas.

Una vez que el proyecto esté creado se verá la pantalla del editor de Unity, la cual se puede dividir en 7 partes principales:

A. La barra de herramientas: da el acceso a las funciones de trabajo más importantes. A la izquierda se encuentran las herramientas básicas para editar la escena y los GameObjects que se encuentren dentro de ella. En la zona central se observan los controles de inicio y pausa. Y por último en el lado derecho se observan los botones que dan acceso a Unity Collaborate y a la cuenta de Unity.

B. Ventana de jerarquía: consiste en una representación de cada GameObject que se encuentra en la escena, todos los elementos de la escena se localizan en dicha jerarquía.

C. La vista de juego: es la pestaña que simula como será el juego final una vez renderizado a través de sus cámaras de escena. Una vez se hace clic en el botón “reproducir” empieza la simulación.

D. Vista de escena: utilizada para navegar y editar visualmente la escena. Mostrando su perspectiva (2D o 3D) según el proyecto en el que se trabaje.

E. Ventana del inspector: es la que permite editar y observar todas las propiedades relacionadas con el GameObject que se tenga seleccionado. Todo ello es debido a que no todos los GameObject tienen los mismos conjuntos de propiedades, y esto cambia en función del GameObject que se tenga seleccionado.

F. La ventana de proyecto: finalmente en esta ventana se muestra la biblioteca de activos que se pueden utilizar en dicho proyecto. Todos los activos que se importen en el proyecto aparecerán en esta ventana.

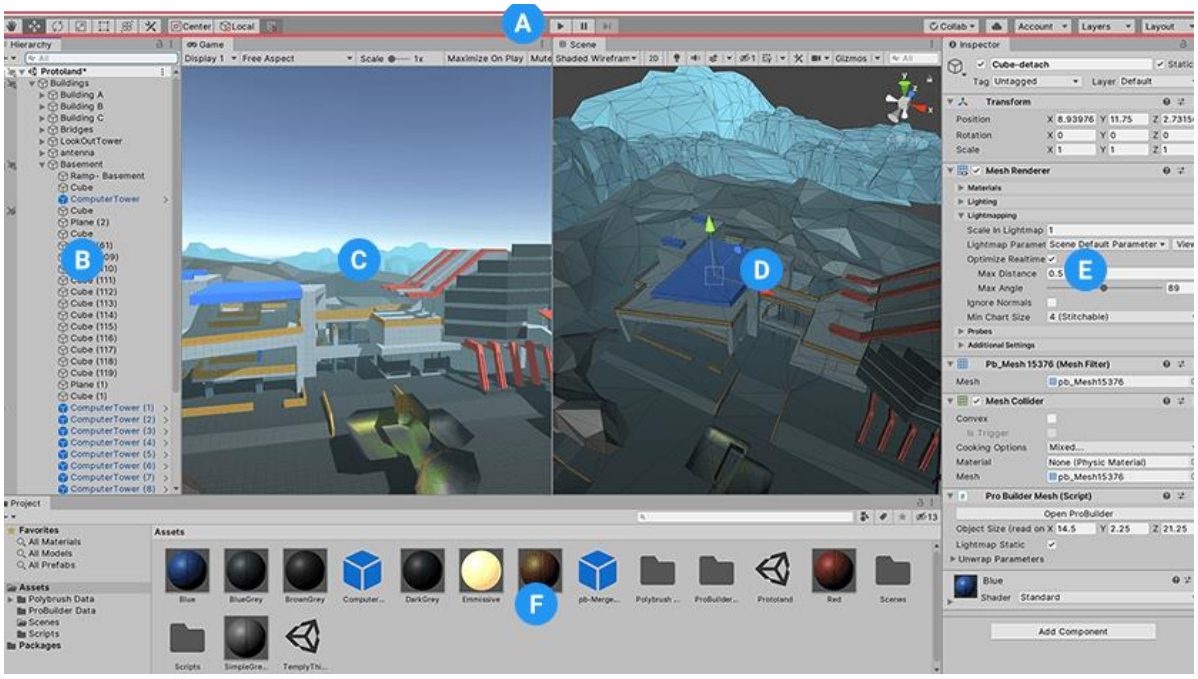

Ilustración 6: Partes principales del editor de Unity.[15]

A parte de lo nombrado anteriormente disponemos de la Vista de la consola: En esta ventana se pueden observar todos los mensajes relacionados con la compilación, como pueden ser los mensajes prints (o de debug de los scripts), errores producidos al compilar el proyecto. Se muestra el error producido y así se puede corregir.

Por otro lado, *Unity Hub* [16] es una herramienta utilizada para la gestión y organización de proyectos Unity. Gracias a Unity Hub se pueden crear nuevos proyectos o abrir los existentes, gestionar instalaciones del editor a la vez que sus componentes.

En resumen, Unity Hub se puede utilizar para:

- Crear un proyecto nuevo asociándole una versión del editor.
- Administrar las licencias de editor y la cuenta de Unity.
- Ejecutar dos versiones de Unity a la vez.
- Añadir componentes a la instalaciones del editor ya instalada.

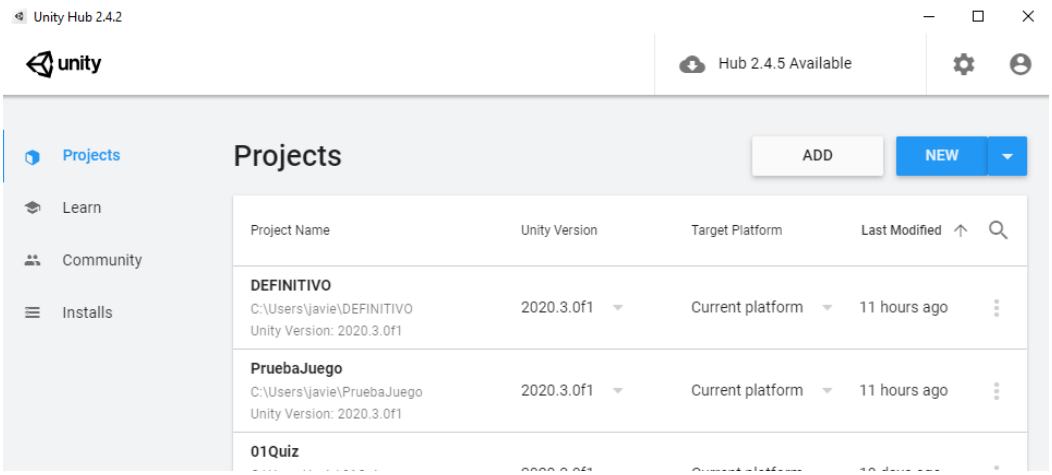

Ilustración 7: Unity Hub

Por lo que, esta herramienta ofrece proyectos de muestra y tutoriales con diversas temáticas. Es decir, cuenta con un apartado de comunidad donde se puede encontrar blogs, lecciones en vivo para poder resolver errores, foros para discusión entre otras.

2.2 Atom

El software [17] Atom es un editor de texto especializado en programación. Es compatible con los lenguajes de programación más populares permitiendo programar y escribir código fácilmente desde tu ordenador con Windows/Linux o maxOs. Es muy completo gracias a la gran variedad de funciones que posee y se pueden instalar para adquirir nuevas características gracias a sus plugins, también se puede cambiar de aspecto debido a la pluralidad de temas disponibles.

Ilustración 8: Logo Atom.[18]

En resumen, Atom dispone de aproximadamente 9000 plugins, los cuales puedes instalar a tu imagen y semejanza para adquirir las funcionalidades deseadas.

Ahora veremos los plugins más recomendables o utilizados:

- KITE: Este plugin incorpora un asistente de programación basado en inteligencia artificial, con el cual puedes escribir código en Java o Python.

- Su función principal es que seas capaz de escribir código de forma más rápida utilizando atajos. Es decir, nos muestra la documentación de cada elemento o símbolo que tengamos señalado por el cursor. A demás añade la función de autocompletado.

- PLATFORMIO-IDE-TERMINAL: Añade un terminal a Atom necesario de tener junto con un editor de código. También integra API y PlatformIO IDE. Una de sus peculiaridades es el uso de sus propios comandos con atajos.

- ATOM-BEAUTIFY: Consiste en un plugin utilizado para adornar el código en diversos lenguajes como puede ser en CSS, PHP, C, C#, Java entre otros.

- SCRIPT: Plugin que ejecuta scripts en función del nombre del archivo, el número de línea o la selección de un código determinado. De esta forma automatizar tareas se vuelve más fácil.

- EMMET: Funcionalidad que permite programar de manera más fácil en CSS y HTML con la ayuda de fragmentos de código. El tabulador también te ayuda a completar abreviaciones. También dispone de unos comandos con los que puedes ejecutar o lanzar directamente con atajos de teclado

- ASK-STACK: Es un plugin que permite preguntar dudas y hacer preguntas en Stack Overflow desde un panel que está integrado en Atom.

- TURBO JAVASCRIPT: Utilizado para escribir código de forma más rápida gracias a sus fragmentos de código con los que no perder tanto tiempo y esfuerzo. Completando código gracias a comandos y fragmentos que se pueden aprender rápidamente.

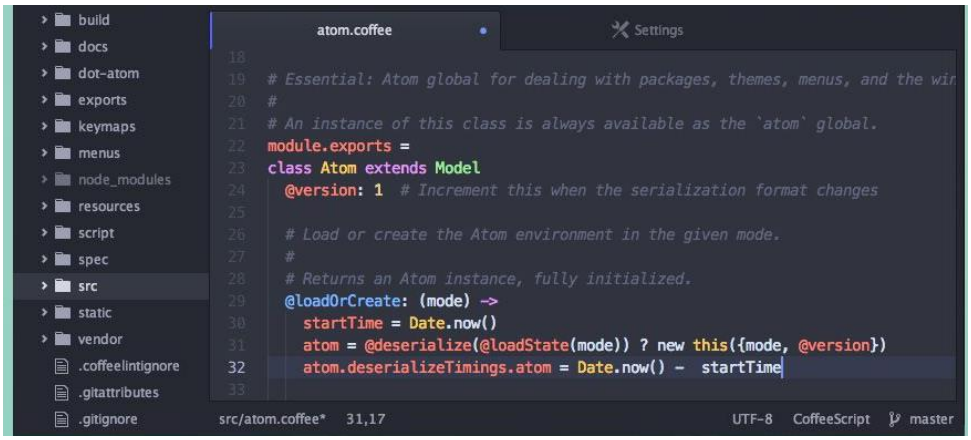

Ilustración 9: Script de ejemplo de Atom. [19]

2.3 Xampp

XAMPP (X (cualquier sistema operativo), Apache, MySQL, PHP, Perl) [20] es un software libre (servidor independiente de plataforma) que se basa sobre todo en MySQL (base de datos), en Apache (Servidor Web) y en Perl YPHP (interpretes para lenguajes de script). Este programa se utiliza como un servidor web libre con facilidad de uso. Este software está operativo para Solaris, GNU/Linux, Windows y MacOS.

En resumen, instala de forma sencilla el servidor web Apache en tu ordenador independientemente del sistema operativo que dispongas. Y todo ello de forma gratuita. Por otro lado, esta herramienta te posibilita el poder probar tu trabajo en tu ordenador sin la necesidad de acceder a internet.

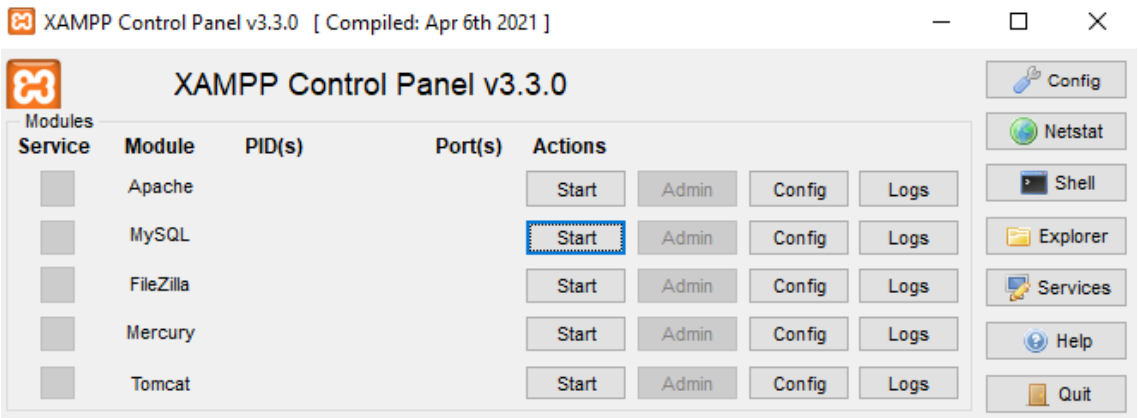

Ilustración 10: Xampp Control Panel.

Requisitos y características: Incluye otros módulos como phpMyAdmin [21] y OpenSSL. Solo basta con descargar un archivo y ejecutar su .exe. XAMPP se actualizará de forma regular y autónoma para de esta forma incorporar las últimas versiones de Apache, MySQL, Perl y PHP.

Aplicaciones: En un principio los creadores de este software libre pretendían usar XAMPP como una herramienta de desarrollo, para de esta forma permitir a los programadores/diseñadores de sitios web probar su trabajo en sus ordenadores sin la necesidad de internet. Sin embargo, hoy en día XAMPP se utiliza para servidores de sitios web (bastante seguros).

Requerimientos del hardware: Para poder instalar este software los requisitos mínimos para poder llevarlo a cabo son 256 MB de RAM y 85 MB de almacenaje, pero esto no bastaría si es un lugar bastante frecuentado o con multitud de archivos subidos.

2.4 Jetbrains Rider

JetBrains Rider es un [22] editor multiplataforma de C# potente y rápido utilizado para Unity que funciona con diversos sistemas operativos (Mac, Windows y Linux). Es un software que permite escribir código de manera más rápida y sin errores gracias a su más de 2500 inspecciones y refactorizaciones de código inteligente.

Ilustración 11: Logo Jetbrains Rider. [23]

Dispone de una compatibilidad incorporada con Unity, que se configurará para que cada vez que utilices Unity, se use Rider como editor preestablecido para los scripts C# y archivos shader.

Se observa una propiedad que ayuda al control del editor de Unity, debido a que sin salir de Rider se puede hacer una pausa, entrar y salir del modo "play" y todo ello gracias a la comunicación bidireccional integrada. Cualquier cambio que se realice en el código C# en Rider, se enviará directamente al editor de Unity.

Este software proporciona también un análisis de código, incluyendo inspecciones específicas de Unity. También depurar los scripts de C# se convierte en una tarea muy sencilla, debido a que se ejecutan en el editor de Unity. Esta tarea basta con hacer únicamente clic en un boto "Debug"

De la misma forma Rider también ayuda a escribir código de mejor rendimiento. Para llevarlo a cabo resalta las llamadas a métodos y métodos que utiliza API. También sugiere soluciones en las líneas de comando.

Tiene a su vez compatibilidad con Shader, es decir es compatible con archivos shader, con resaltado de sintaxis entre otros.

Por último, Rider te muestra documentación adicional para ayudarte en el proceso de escribir el código usado en Unity.

CAPÍTULO 3. Bases de datos.

Se realizará una breve introducción sobre las bases de datos, comentando el diseño y su programación.

3.1 Introducción a la base de datos

A la hora de plantear el proyecto surge la necesidad de utilizar bases de datos. Para de esta forma tener registrado de manera ordenada a todos los usuarios que acceden a la aplicación. Por lo que en la base de datos será necesario almacenar toda la información procedente del internauta. En este caso los datos que se almacenarán son: el nombre de usuario, el email y la contraseña.

A continuación, se hará una breve introducción sobre la base de datos, las cuales, consisten en una colección de datos de forma estructurada, es decir es el lugar donde los datos son almacenados y se organizan. En las bases de datos se puede guardar información sobre productos, pedidos y personas (que es nuestro caso) entre otros.

En este proyecto se utilizará la herramienta phpMyAdmin, es gratuita y permite acceder a todas las funciones que nos ofrece la base de datos MySQL (sistema de gestión de bases de datos), con la ayuda de una interfaz gráfica bastante fácil de usar e intuitiva.

Dicha herramienta nos ofrece una gran cantidad de características y usos, entre los que destacan:

Permite realizar operaciones básicas en base de datos MySQL (crear, eliminar, editar bases de datos), por otro lado, esta aplicación nos permite optimizar realizar búsquedas en las bases de datos. Los usuarios a la hora de utilizarlo no deberían de tener problemas debido a su fácil uso.

3.2 Creación de la base de datos

Se creará un sistema Login/registro. Las herramientas que vamos a utilizar para poder trabajar con el lado del servidor son XAMPP (paquete para trabajar con php y MySQL) y un editor de texto para escribir php (Atom).

En primer lugar, se abre Unity para crear la parte gráfica, será un proyecto en 2D.

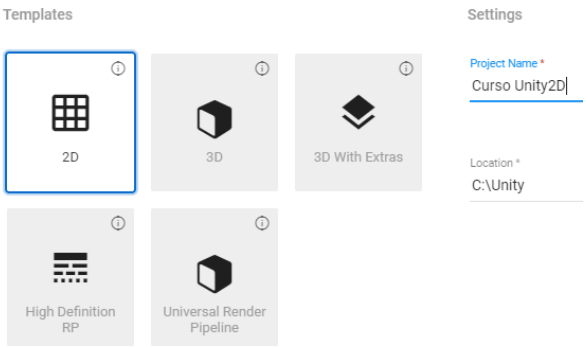

Figura 1: Creación proyecto nuevo en Unity.

Guardaremos nuestra escena y nuestra interfaz gráfica a través de un canvas. El canvas dispondrá de un *background* (Imagen de fondo), de un objeto vacío para crear toda la interfaz Login y otro para crear la interfaz register. Arriba de todo se pondrá un texto que mostrara si todo está correcto, si faltan datos o si los datos coinciden o no.

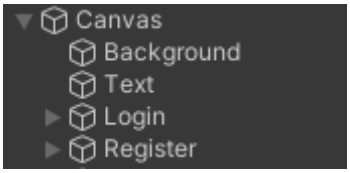

Figura 2: Canvas Base de datos.

En el interfaz Login se tendrá 2 inputs (*UserName y password* respectivamente) con dos botones que servirán para registrarse o para loguearse directamente.

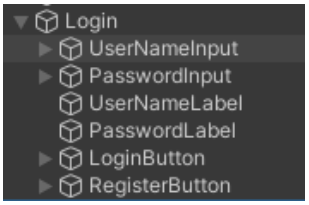

Figura 3: Interfaz Login.

Por otro lado, la interfaz de register dispondrá de 4 Inputs (*UserName, email, password y repeat password*) y de la misma forma que antes habrá dos botones para registrarse y loguearse.

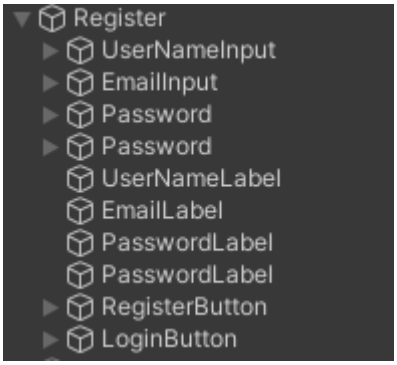

Figura 4: Interfaz Registro.

A continuación, se creará la base de datos para almacenar la información del usuario. Se accede a la carpeta de XAMPP y se abrirá la aplicación xampp-control.exe y se ejecutarán Apache y MySQL para que de esta forma estén funcionando.

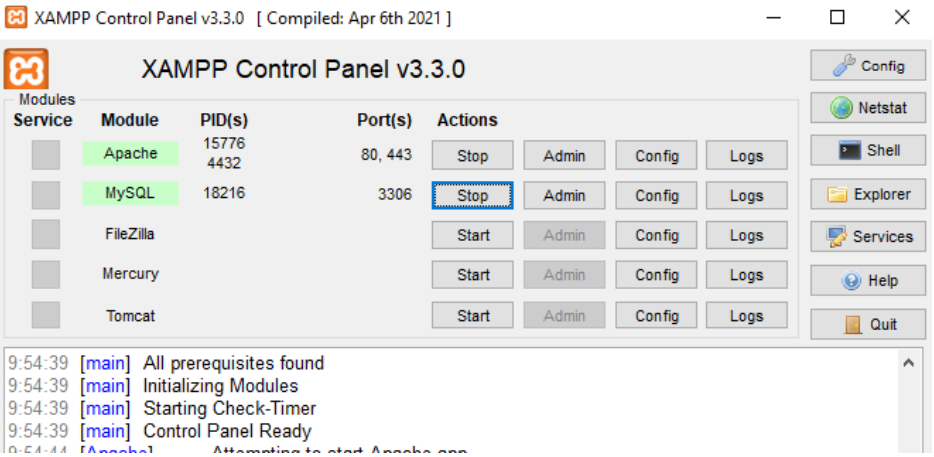

Figura 5: activación Xampp Control Panel.

Una vez se ha realizado lo anterior, en el buscador se accede a http://localhost/phpmyadmin para crear nuestra base de datos. Se le adjudicará un nombre y un cotejamiento, en nuestro caso utf8_spanish_ci.

Figura 6: creación Base de datos.

Después se creará una tabla con tres columnas (UserName, email y password) con sus respectivos campos de características. Las contraseñas las pasaremos a privado para que una vez que se rellenen no se vean.

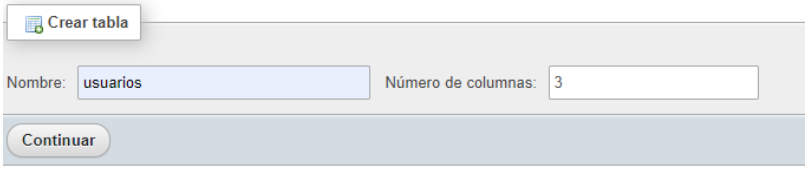

Figura 7: creación Tabla Base de datos 1.

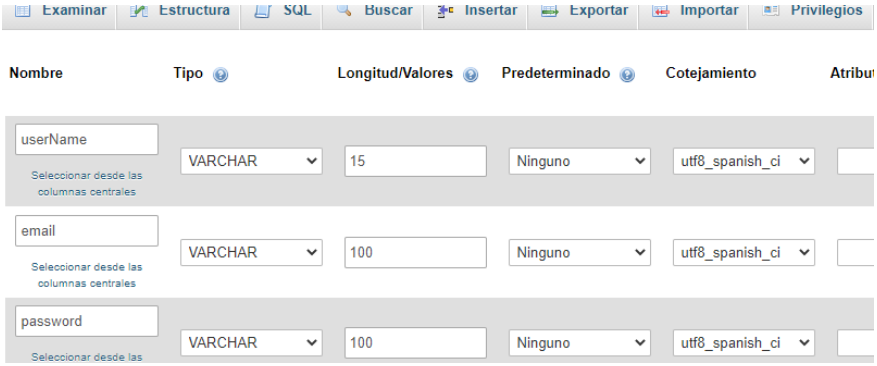

Figura 8: creación Tabla Base de datos 2.

Una vez se ha creado la tabla, se debe establecer nuestra cuenta de usuarios, donde se escribirá el nombre de usuario, el nombre de host y por último la contraseña que utilizaremos.

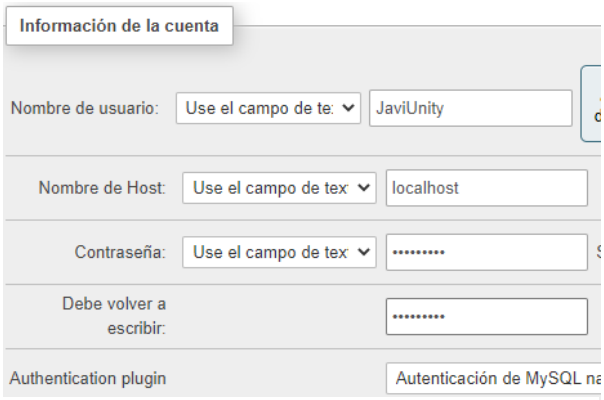

Figura 9: Creación cuenta de Usuarios.

En usuarios debe estar creado correctamente.

Usuarios con acceso a "game.usuarios"

	Nombre de usuario	Nombre del servidor	Tipo	Privilegios	Conceder
☐	JaviUnity	localhost	específico para la base de datos	ALL PRIVILEGES	No
☐	root	127.0.0.1	global	ALL PRIVILEGES	Sí
☐	root	::1	global	ALL PRIVILEGES	Sí
☐	root	localhost	global	ALL PRIVILEGES	Sí

Figura 10: Usuario Creado.

A continuación, se realizará la parte que envía la información y será guardada en la base de datos. En primer lugar, ira la parte de registro. Se crea un nuevo script (*SceneManager*) que se encargará de manejar nuestra escena, y se le añade a *MainCamera* (nuestra cámara principal).

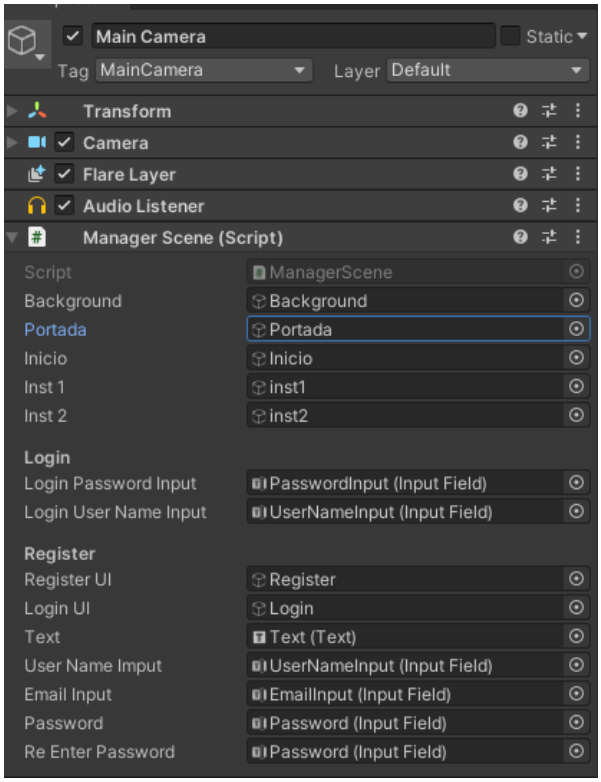

Figura 11: MainCamera proyecto Unity.

En el script SceneManager se crearán los objetos que necesita la interfaz gráfica, así como funciones para mostrar las diversas partes cuando se hace clic en un botón (Login, register, inicio …) Esas funciones se añaden en Unity al botón que le corresponde.

```
public class ManagerScene : MonoBehaviour
{

    [SerializeField] private GameObject m_background = null;  Background
    [SerializeField] private GameObject m_portada = null;  Portada
    [SerializeField] private GameObject m_inicio = null;  Inicio
    [SerializeField] private GameObject m_inst1 = null;  inst1
    [SerializeField] private GameObject m_inst2 = null;  inst2
```

Figura 12: Objetos necesita Interfaz Gráfica.

Las funciones que tendremos serán:

- *SubmitRegister:* encargada de realizar el registro del nuevo usuario. Verificará que todos los campos están rellenados correctamente y que sus contraseñas sean iguales, en caso de no ser así, daría un mensaje de error y habría que volver a registrarse.
- *SubmitLogin:* encargado de realizar el Login del nuevo usuario comprobando que el nombre de usuario y contraseña coincide con el guardado en registro, de ser así, te lleva directamente al juego.

```
public void submitLogin()
{
    if (m_loginUserNameInput.text == "" || m_loginPasswordInput .text == "" )
    {
        m_text.text = "Rellene todos los campos";
        return;
    }

    m_text.text = "Procesando...";

    m_networkManager.CheckUser(m_loginUserNameInput.text , pass: m_loginPasswordInput.text, response: delegate(Response response)
    {
        m_text.text = response.message;


    });

}
```

Figura 13: Función SubmitLogin

- *ShowLogin:* Una vez registrado te lleva a la ventana para que te puedas loguear.
- *ShowRegister:* Te lleva a la ventana para que puedas llevar a cabo el registro.
- Otras funciones Show que habilitan y deshabilitan ventanas en función de a cuál se quiera ir a la hora de clicar un botón.

```
public void ShowLogin()
{
    m_registerUI.SetActive(false);
    m_loginUI.SetActive(true);
    m_portada.SetActive(false);
    m_inst1.SetActive(false);
    m_inst2.SetActive(false);
    m_inicio.SetActive(false);
```

Figura 14: Función ShowLogin.

Se llevará a cabo después la parte encargada de enviar nuestros datos al servidor, creando un nuevo script *(NetworkManager*) que se encargará de enviar y recibir la información del servidor.

En el script *NetworkManager*, se creará primero la función encargada de crear un nuevo usuario con la información del usuario que estará colocada en los InputField. Dentro se empezará una corrutina con el método en cuestión (*CreateUser*), que tendrán los mismos datos que la función para que espere hasta que el servidor responda y dé la respuesta. Dicha función se utilizará en submitRegister.

```
public class NetworkManager : MonoBehaviour
{
    1 usage
    public void CreateUser(string userName, string email, string pass, Action<Response> response)
    {
        StartCoroutine( routine: CO_CreateUser( userName , email , pass , response));
    }
```

Figura 15: Corrutina CreateUser.

Dentro del método habrá que crear un formulario con los datos que se quieran añadir, y un WWW encargado de enviar nuestro formulario a nuestro servidor (http://localhost/Login/createUser.php), acto seguido esperaremos a que envíe su respuesta, esta respuesta que nos envía el servidor está en *w.text* y se transforma a tipo response y se envía a nuestro delegado.

```
private IEnumerator CO_CreateUser(string userName, string email, string pass, Action<Response> response)
{
    SecureForm form = new SecureForm();
    form.secureForm.AddField("userName", value: userName);
    form.secureForm.AddField("email", value: email);
    form.secureForm.AddField("pass", value: pass);

    WWW w = new WWW( url: "http://localhost/Login/createUser.php", form.secureForm);

    yield return w;
    response( obj: JsonUtility.FromJson<Response>(w.text));
}
```

Figura 16: Script formulario con los datos.

De la misma forma que se creó la función CreateUser, se creará otra función llamada CheckUser, con su respectivo formulario con sus datos correspondientes y conectado al servidor (http://localhost/Login/checkUser.php). Esta función se utilizará en *SubmitLogin* para comprobar que se ha logueado correctamente.

```
public void CheckUser(string userName, string pass, Action<Response> response)
{
    StartCoroutine( routine: CO_CheckUser( userName , pass , response));
}

Frequently called  1 usage
private IEnumerator CO_CheckUser(string userName, string pass, Action<Response> response)
{
    SecureForm form = new SecureForm();
    form.secureForm.AddField("userName",  value: userName);
    form.secureForm.AddField("pass",  value: pass);

    WWW w = new WWW( url: "http://localhost/Login/checkUser.php", form.secureForm);

    yield return w;
    Debug.Log(w.text);
    response( obj: JsonUtility.FromJson<Response>(w.text));
}
```

Figura 17: String CheckUser.

También se ha creado una clase serializable (para poder ser transformada a JSon) llamada *Response* que nos dirá si todo ha sido satisfactorio y el mensaje que nos devuelve el servidor.

```
[Serializable]
8 usages
public class Response
{
    public bool done = false;  Serializable
    public string message = "";  Serializable
}
```

Figura 18: Script Clase Response.

Seguiremos con nuestro sistema Login de registro, se creará el proyecto en php para poder enviar la información. Habrá que seguir los siguientes pasos:

- Se abre la carpeta de nuestra instalación de XAMPP
- Se accede a la carpeta "htdocs"
- Se crea una carpeta "X" (nombre que se desee) donde se guardarán todos los scripts de php que se mencionarán a continuación.
- Abrir Atom y se le añade la carpeta de proyecto creada anteriormente.

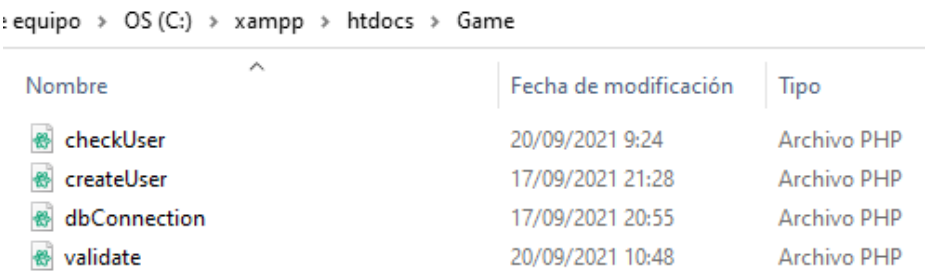

Figura 19: Scripts Carpeta "htdocs" de Xampp.

En primer lugar, en Atom se creará un script encargado de realizar la conexión con la base de datos llamado Connection.php. En él se conectará la base de datos mediante un objeto PDO añadiendo nuestro dbname, nombre de usuario y contraseña que se ha utilizado anteriormente para crear nuestra base de datos, y más abajo si existe algún error se imprimirá en pantalla.

```php
try
{
  $pdo = new PDO('mysql:host=localhost;dbname=game' , 'JaviUnity' , '**********');
  $pdo->setAttribute(PDO::ATTR_ERRMODE , PDO::ERRMODE_EXCEPTION);
  $pdo->exec('SET NAMES "utf8"');
}
catch (PDOException $e)
{
  echo  "ERROR CONECTING TO DATABASE" . $e->getMessage();
  exit();
}
```

Figura 20: Scripts Connection.

En segundo lugar, se creará el script CreateUser.php (este nombre debe coincidir con el WWW encargado de crear nuestro formulario en el script *NetworkManager*). Hay que añadir un include a *"Connection"* que lo sustituirá por el código que se programó anteriormente.

Mediante un post se escribirán los datos que se quieren que sean enviados a la base de datos, se tienen que escribir de la misma forma que se puso en Unity. A la hora de pasar la contraseña esto se hará de forma encriptada utilizando hash y concretamente "Sha256".

```php
$userName = $_POST['userName'];
$email = $_POST['email'];
$pass = hash("sha256" , $_POST['pass']);
```

Figura 21: Variables creadas PHP.

A continuación, hay que comprobar que el nombre de usuario no existe ya anteriormente mediante un SQL, que conectaremos a la base de datos mediante un pdo y se pasa el valor almacenado en sql a una variable (*"result"),* si por un casual el resultado devuelve que ya existe otro usuario con ese nombre se reporta para que dicho nombre de usuario sea cambiado.

```
$sql = "SELECT userName From usuarios WHERE userName = '$userName'";
$result = $pdo->query($sql);
if($result->rowCount() > 0)
{
  $data = array('done' => false , 'message' => "Nombre de usuario existente");
  Header('Content-Type: application/json');
  echo json_encode($data);
  exit();
```

Figura 22: Script comprobación.

Por otro lado, de la misma forma hay que comprobar que el email introducido no exista. Una vez el nombre de usuario y email sean validos se almacenarán en la base de datos y saldrá un mensaje diciendo que el usuario ha sido creado con éxito.

De igual forma que para la función/método *CreateUser* se programó CreateUser.php ahora se necesita crear un archivo para la función *CheckUser* programada en el script *NetworkManager*.

Este script (programado en Atom, y que se encuentra en la carpeta creada dentro de "htdocs" de XAMPP) se llamará *CheckUser.php*. En primer lugar, se crean las etiquetas de php y se incluirá dbConnection.php para poder acceder a la base de datos. Acto seguido de la misma manera que se envía desde Unity se escriben las variables que se necesitan en este caso (*UserName y pass*), la contraseña se codificará de nuevo para poder compararlas.

```
$userName = $_POST['userName'];
$pass = hash("sha256" , $_POST['pass']);
```

Figura 23: Variables en CheckUser.

Por último, se crea un sql para poder comprobar que en la base de datos exista algún usuario que se ha registrado con ese nombre y con esa contraseña. El sql se guardará en una variable. Una vez se ha creado el sql, hay que comparar que existe alguien en la base de datos, de no ser así saltará un mensaje que avise de que el usuario o la contraseña no son correctos. Si los datos han sido introducidos satisfactoriamente aparecerá un mensaje de bienvenida.

```php
$sql = "SELECT userName From usuarios WHERE userName = '$userName' AND password = '$pass'";
$result = $pdo->query($sql);

if($result->rowCount() > 0)
{
  $data = array('done' => true , 'message' => "Bienvenido $userName");
  Header('Content-Type: application/json');
  echo json_encode($data);
  exit();
}
```

Figura 24: comprobación variables.

Ya se tiene creado con existo el Login y registro y su almacenamiento en la base de datos.

Antes de finalizar el apartado de Login y Registro se añadirán claves para impedir que cualquier persona que tenga el link de nuestro servidor (o su acceso) pueda introducir información en nuestra base de datos sin que nos demos cuenta, o robar información de dicha base lo cual podría ser dañino para nuestra aplicación.

Para ello crearemos en Unity un nuevo Script llamado *secureForm.cs*. Se comenzará creando la clase *SecureForm* (no desciende de *MonoBehaviour* debido a que no se va a añadir en ningún *GameObject)*. Cabe recordar que hay que intentar añadir la mínima cantidad de permisos que sean necesarios, debido a que, si se crean más, se puedan llegar a producir errores.

Se añaden las contraseñas que necesitaremos en función de la plataforma en que nos encontremos. En nuestro caso habrá tres tipos: la contraseña de conexión que se utilizará en todas las peticiones a nuestro servidor, la contraseña Android que mediante el uso de directivas dependientes de plataforma solo se utilizará cuando se esté en un dispositivo Android. Y por último la contraseña iOS que de igual forma que la anterior solo se utilizará cuando se esté en una plataforma iOS.

Se crea el constructor de la clase form, iniciando en primer lugar el formulario y acto seguido añadiremos las contraseñas.

```
public class SecureForm
{
    private WWWForm m_secureForm = null;
    private const string CONNECTION_PASSWORD = "****************";
    private const string ANDROID_PASSWORD = "***************";
    private const string IOS_PASSWORD = "****************";
```

Figura 25: Script SecureForm.

Por último, añadimos un acceso público a nuestro formulario. Dicho formulario creado deberá ser sustituido en el script *NetworkManager*.

```
public SecureForm()
{
    m_secureForm = new WWWForm();

    m_secureForm.AddField("connectionPass" , value: CONNECTION_PASSWORD);

    #if UNITY_ANDROID
    m_secureForm.AddField("os" , value: "android");
    m_secureForm.AddField("platformPass" , value: ANDROID_PASSWORD);
    #endif

    #if UNITY_IOS
    m_secureForm.AddField("os" , "ios");
    m_secureForm.AddField("platformPass" , IOS_PASSWORD);
    #endif
}
```

Figura 26: Formulario SecureForm.

El funcionamiento se resumiría en que una vez que se llama al constructor se añade directamente la contraseña de conexión, las contraseñas que depende de la plataforma en la que se encuentre el dispositivo. Es decir que en cada petición que se haga a nuestro servidor no es necesario que se añadan las contraseñas en cada petición, basta con llamar al constructor que hará que se añadan directamente.

Ahora se trabajará en la parte del servidor por lo que se necesitará un archivo php para verificar que las contraseñas que se han enviado sean las correctas.

Por lo que se crea el archivo *validate.php* añadiendo los target de php. A continuación, añadimos las tres contraseñas que se utilizarán (la contraseña de conexión y las de plataforma) también se añadirá la variable para saber en qué sistema operativo nos encontramos "os".

```
$connectionPass = $_POST["connectionPass"];
$os = $_POST["os"];
$platformPass = $_POST["platformPass"];
```

Figura 27: Script Valídate.

Primero se comprobará que la contraseña de conexión es la correcta, de no ser así saltará un mensaje de error "autentificación fallida". Luego se comprueba la contraseña en función del sistema operativo en el que se encuentre. Si el S.O es de Android habrá que comprobar que su contraseña especifica de plataforma

sea la correcta, de no ser así de la misma forma que para la de "conexión" aparecerá un mensaje de error.

```
if($connectionPass == "************")
{

  if($os == "android" && $platformPass != "******************")
  {

    $data = array('done' => false , 'message' => "Autentificacion fallida");
    Header('Content-Type: application/json');
    echo json_encode($data);
    exit();
  }
```

Figura 28: Script comprobación Sistema Operativo.

Este proceso se realiza de la misma forma para cuando el sistema operativo en el que se encuentre sea iOS.

En resumen, si la contraseña de conexión no es igual, mandará un error. De la misma forma si es Android, pero a contraseña de plataforma (Android) no es igual mandará error y si es IOS y la contraseña de plataforma (IOS) no coincide mandará también error.

En todos nuestros scripts.php hay que incluir validate.php (en *CreateUser.php* y en *CheckUser.php*). Hay que aclarar que en los scripts de php el software utilizado es Atom, mientras que los que se crean desde Unity se hace a través del software JetBrains Rider.

3.3 Diseño de la base de datos

El login y registro del jugador en la aplicación consta de dos ventanas, en primer lugar, nada más encender la aplicación aparecerá la pestaña Login, donde el usuario deberá loguearse, si por un casual no está registrado debe realizar esta acción primero. Como se ha dicho anteriormente en el registro se le pedirá al usuario el user Name, el email y un password con el que poder acceder a la aplicación. Por otro lado, a la hora de loguearse deberá introducir el user name y la contraseña únicamente. Se le podrían pedir multitud de datos más al usuario como la edad, curso entre otras, pero se ha optado por pedir los datos básicos para poder registrarse correctamente.

A continuación, se mostrarán dos imágenes correspondientes al diseño de la ventana *Login* y la de *Registro*:

Figura 29: Diseño Login y Registro.

También cabe enseñar como se almacenan los datos en la herramienta phpMyAdmin cada vez que un usuario se registra:

userName	email	password
prueba	prueba	03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e...
prueba8	prueba8	03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e...
prueba9	prueba9	03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e...
Prueba10	prueba10	03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e...
prueba11	prueba11	03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e...
prueba13	prueba13	03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e...
prueba14	prueba14	03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e...
prueba16	prueba16	03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e...
prueba20	prueba20	03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e...
juan	juan	03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e...
javi	javi	03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e...
javier	javier	03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e...
fco	fco	03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e...
miguens	miguens	03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e...
usuario	usuario	03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e...
usuario1	usuario1	03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e...
usuario2	usuario2	03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e...

Figura 30: Almacenaje Usuarios.

En la imagen anterior se muestra diversos usuarios registrados con su email y contraseña, estos usuarios son “ficticios” y han sido utilizados para llevar a cabo el correcto funcionamiento de las bases de datos y de la aplicación.

CAPÍTULO 4. Programación de la APP.

Se explicará el código programado para
llevar a cabo el funcionamiento
de la aplicación.

4.1 Programación estructura general

La aplicación que vamos a programar tendrá cuatro modos de uso diferentes;

- El primer modo será el modo “fácil” en el que el jugador dispondrá de un tiempo limitado e irá sumando el número de preguntas que conteste correctamente (puntuación total). Una vez que falle una pregunta, el juego finalizará automáticamente.
- El segundo tipo será el modo “experto” en el que el jugador sumará puntos en función de la rapidez con la que acierte la pregunta (a mayor velocidad de respuesta mayor puntuación conseguirá el usuario y viceversa), el tiempo de cada pregunta será inferior que el modo anterior. Pero al igual que antes, una vez que falle una pregunta el juego se acabará.
- El tercer modo que podemos encontrar es el modo “entrenamiento” donde el usuario podrá practicar la energía renovable que desee, de forma que solo le aparezcan preguntas de ese tipo, y cuando quiera puede finalizar la partida.
- Por último, tenemos el modo “Partida personalizada” donde el jugador puede elegir qué características quiere que tenga su partida. Eligiendo de esta forma el número de preguntas que responder, si quiere vidas o no y en caso afirmativo cuantas y por último el tiempo del que dispone para responder esa pregunta. A la hora de finalizar la partida le dirá al usuario cuantas ha respondido de manera satisfactoria.

El diseño de los cuatro modos será muy similar con alguna modificación a la hora de las puntuaciones, por lo que primero se explicarán las partes en común y por último especificaremos las diferencias que podemos encontrar en cada modo.

Comenzaremos con los scripts y objetos necesarios de programar en el juego.

En primer lugar, se crea un nuevo proyecto (2D), se comenzará con nuestra base de datos, que será donde se irán almacenando nuestras preguntas. Se crea el script encargado de almacenar la opción *(“option”*) y se creará otro script (*“question”*).

En el *script option*, no descenderá de *MonoBehaviour* y se encargará de almacenar la opción, tendrá dos variables, la primera *text* (opción que se le da al

```
[System.Serializable]
3 usages   2 exposing APIs
public class Option
{
    public string text = null;
    public bool correct = false;
```

Figura 31: Script "Option"

jugador) y *bool* que te dice si la opción es correcta o no. Hay que tener en cuenta que la clase debe ser *serializable* para poder editarla desde el editor.

En el script *"Question"* si desciende de *MonoBehaviour* porque si se quiere añadir a un *GameObject*, en dicho script tendremos un *string* que será el texto de la pregunta y una lista donde se almacenarán las opciones que se le dan al jugador. La función de este script consiste en almacenar cada opción por separado.

A continuación, crearemos un objeto (*Question_X*) y se le añade nuestro Script *"question"* para poder editar nuestra pregunta desde el editor. Dentro de cada pregunta podemos seleccionar el número de respuestas, el texto de cada respuesta e indicar cual es la correcta.

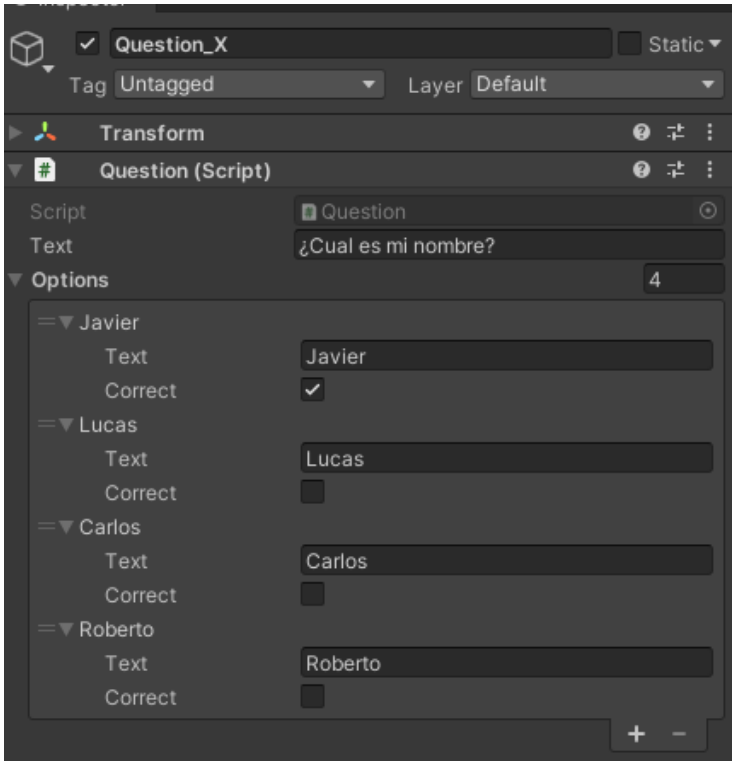

Figura 32: Objeto Question_X.

En *Question_X* donde X se refiere al número de preguntas que se tienen. Se creará una carpeta llamada *"DB"* y dentro se almacenarán todas las preguntas que se escriban para el juego.

Todas nuestras preguntas serán almacenadas como *GameObject* y se creará otro script *("QuizDB")* donde se almacenarán todas las preguntas, es decir será como nuestra base de datos.

El *"QuizDB"* será una lista donde se guardarán todas las preguntas que queramos utilizar.

```
public class QuizBD : MonoBehaviour
{
    [SerializeField] private List<Question> m_questionList = null;

    private List<Question> m_backup = null;
```

Figura 33: Script QuizDB

A continuación, se creará otro objeto en la escena *QuizDB* donde se asignará el script *"QuizDB"* a dicho objeto. Aquí se asignarán las preguntas que queremos utilizar dentro de nuestro juego.

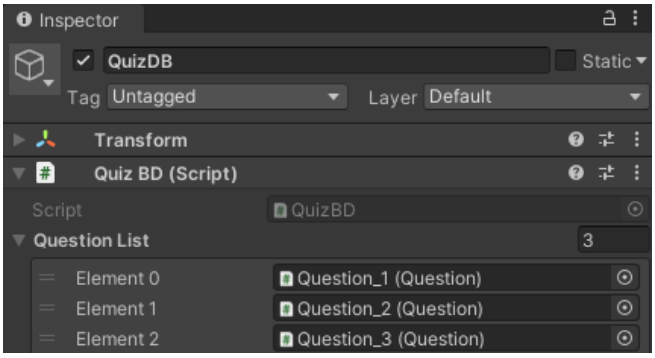

Figura 34: Objeto QuizDB.

También se añadirá un método que ayudará a obtener una pregunta al azar. Dentro de ese método se añade la opción de que la pregunta una vez haya salido y se responda correctamente no sea repetida. Por lo que, si aciertas una pregunta, esta no te volverá a salir en esa ronda de la partida.

```
public Question GetRandom(bool remove = true)
{
    if(m_questionList.Count==0)
        RestoreBackup();

    int index = Random.Range(0, m_questionList.Count);

    Question q = m_questionList[index];
    m_questionList.RemoveAt(index);

    return q;
}
```

Figura 35: Script Generar Pregunta aleatoria.

Si por un casual se acaban todas las preguntas existentes, se reiniciaría la base de datos, colocando de nuevo todas las preguntas y volviendo a empezar. No se

tiene que remover la pregunta obligatoriamente una vez que se acierte, se puede dejar y que surja la posibilidad de que se repita, pero es menos efectivo.

```
private void RestoreBackup()
{
    m_questionList = m_backup.ToList();
}
```

Figura 36: Script No repetición pregunta.

Después se crea el script *("OptionButton")* que ayudará a representar como serían las opciones del juego, es decir las opciones que el jugador va a tener. En este script mediante código se obliga a que haya un componente button (añade un componente button de manera propia), también se obliga a que haya una imagen y así no tener que crearla.

```
[RequireComponent(typeof(Button))]
[RequireComponent(typeof(Image))]
```

Figura 37: Componente botón e Imagen.

Dentro habrá un método publico utilizado como constructor que recibe la opción que queremos mostrar (indicándonos el texto). Dentro se programará todo lo relacionado con el botón, desde si está habilitado o no, la opción que esta seleccionando, hasta el color (verde o rojo) en función de si se acierta la pregunta o no.

```
public void Construct(Option option, Action<OptionButton> callback)
{
    m_text.text = option.text;

    m_button.onClick.RemoveAllListeners();
    m_button.enabled = true;
    m_image.color = m_originalColor;

    Option = option;

    m_button.onClick.AddListener( call: delegate
    {
        callback( obj: this);
    });
}
```

Figura 38: Script Construct.

Hay que tener en cuenta que, para colocar el color, hay que deshabilitar el botón, debido a que este va a querer hacer transición de la imagen, por lo que se deshabilita para que no controle el color de la imagen, y lo colocamos manualmente.

```
public void SetColor(Color c)
{
    m_button.enabled = false;
    m_image.color = c;
}
```

Figura 39: Script Colocar Color.

Ahora se pasará a crear la interfaz gráfica (*Canvas*) escalándolo con el tamaño de la pantalla. Dentro del canvas se creará un objeto vacío (*QuizUI*) para guardar toda la interfaz gráfica necesaria.

Dentro de este objeto se creará un objeto *texto*, que será el texto que mostrará la pregunta. También se crea el objeto vacío *Buttons* que serán las opciones que el jugador tendrá, seleccionando el ancho y largo que queremos conseguir, de igual forma se hizo el objeto Texto.

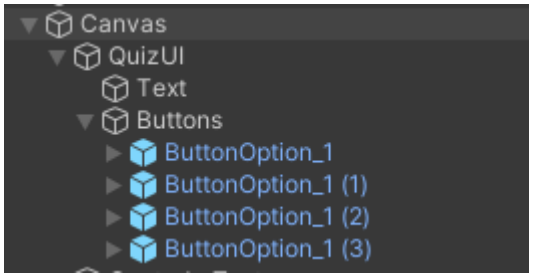

Figura 41: Canvas Interfaz Gráfica.

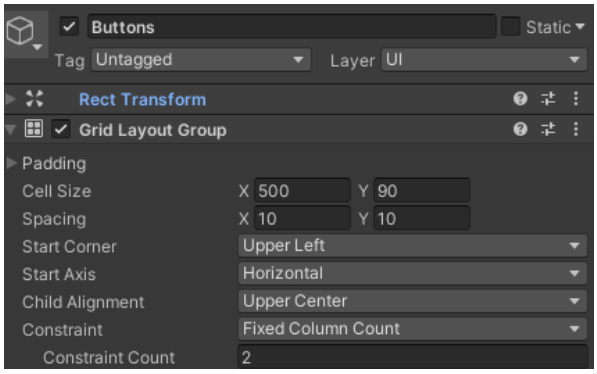

Figura 40: Objeto Buttons.

En el objeto *Buttons* se utilizará el componente *Grid Layout Group*, que ayuda a organizar los hijos de este objeto.

Dentro de este objeto se añadirán otro objeto vacío que denominaremos *ButtonOption_1* donde se añadirá el script *"OptionButton"* programado con anterioridad. Al añadir dicho script se añade directamente el componente imagen y el componente button que mencionamos con anterioridad.

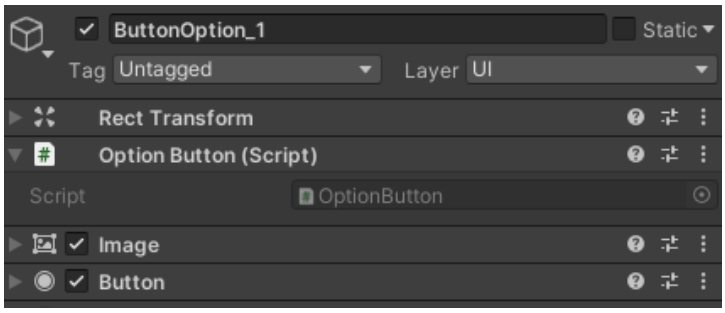

Figura 42: Objeto ButtonOption_1.

A *ButtonOption_1* se le añade un componente de tipo texto (texto de la opción).

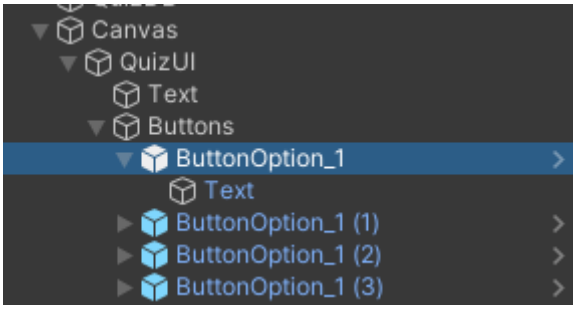

Figura 43: Canvas ButtonOption_1.

Se crea a continuación un prefab para nuestro *botón* y así poder modificarlo, para ello creamos una carpeta *prefab* y añadimos el objeto *ButtonOption_1*.

Figura 44: Prefab.

Se pegará tantos *ButtonOption* como se necesiten, en nuestro caso serán 4, por lo que las preguntas que tengamos tendrán 4 opciones. A través del componente que se ha mencionado anteriormente (*Grid Layout Group*) se colocan como se quiere visualizar sus botones.

En nuestro prefab se modificarán algunas opciones como puede ser el "*Highlighted Color"* donde se cambia el color para que se pueda ver mejor.

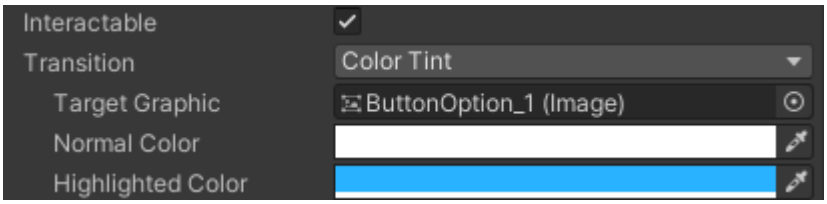

Figura 45: Opciones Prefab.

De esta forma ya se ha creado la interfaz gráfica del juego, encargada de controlar las opciones de nuestro juego.

Antes de acabar se necesitará otro script *("QuizUI")* encargado de controlar la interfaz gráfica de la pregunta en concreto.

En este script se utilizará parte de la interfaz gráfica de *UnityEngine.UI*, se necesitará un texto para colocar la pregunta, y una referencia a los botones que se han creado con anterioridad.

```
using UnityEngine.UI;
1 asset usage  2 usages
public class QuizUI : MonoBehaviour
{
    [SerializeField] private Text m_question = null;  Text (MonoBehavio
    [SerializeField] private List<OptionButton> m_buttonList = null;
```

Figura 46: Script QuizUI.

Dentro se creará el método construct (constructor) donde se enviará la pregunta que se quiere hacer con su respectivo *callback* (método que se pasa a una función que tiene como variable de regreso aquello que introducimos dentro de <>). Primero se coloca el texto de la pregunta que queremos realizar e inicializaremos los botones que son las opciones para las preguntas. Es decir, cada opción tomará el texto de la pregunta.

```
public void Construct(Question q, Action<OptionButton> callback)
{
    m_question.text = q.text;
    for (int n = 0; n < m_buttonList.Count; n++)
    {
        m_buttonList[n].Construct(q.options[n], callback);
    }
```

Figura 47: Script Construct Question.

Por último, se creará el ultimo script *("GameManager")* que se usará para manejar el control del juego, diciéndonos que pregunta será la siguiente, cuando finaliza, etc. Para ello se hará uso de la base de datos y se necesitará una referencia a la interfaz gráfica.

```
private QuizBD m_quizDB = null;
private QuizUI m_quizUI = null;
```

Figura 48: Uso Base de datos e Interfaz Gráfica.

Se añadirán los sonidos que vamos a utilizar (en función de si se acierta o no), también se tendrá el color que se asignará a la opción en función de si es correcta o no. Para poder reproducir sonidos hace falta tener un componente de tipo *AudioSource*. Cabe destacar que todos los componentes anteriores tienen que ser serializados debido a que los queremos modificar desde el inspector.

```
[SerializeField] private  AudioClip m_correctSound = null;
[SerializeField] private AudioClip m_incorrectSound = null;
[SerializeField] private Color m_correctColor = Color.black;
[SerializeField] private Color m_incorrectColor = Color.black;
[SerializeField] private float m_waitTime = 0.0f;  "2.5"

private AudioSource m_audioSource = null;
```

Figura 49: Variables Color y Sonido.

Dentro se creará un método *Start*, donde se obtendrá la referencia a la base de datos, y una referencia a nuestra interfaz gráfica. Se hace esto para obtener la referencia automáticamente y no se olvide añadirla.

```
private void Start()//empezar juego
{
    m_quizDB = GameObject.FindObjectOfType<QuizBD>();
    m_quizUI = GameObject.FindObjectOfType<QuizUI>();
    m_audioSource = GetComponent<AudioSource>();

    NextQuestion();
}
```

Figura 50: Script Función Start.

A continuación, se añade el método *NextQuestion* (siguiente pregunta), dicho método se añadirá en el *Start* para empezar de esta forma directamente con una pregunta.

En *NextQuestion* llamaremos al *QuizU*I para sacar una pregunta aleatoria de la base de datos y se la mandaremos a nuestro *construct*.

```
private void NextQuestion()
{
    m_quizUI.Construct( q: m_quizDB.GetRandom(), GiveAnswer);
}
```

Figura 51: Script NextQuestion.

Acto seguido se crea el método *GiveAnswer* (es el método que será llamado cuando el jugador seleccione una respuesta, da igual si es correcta o no. Nos mandara el botón (*Opción*) que ha seleccionado y se verificara si es correcta. Aquí se iniciará la corrutina de la que se habla a continuación y se pasa el *OptionButton*.

```
private void GiveAnswer(OptionButton optionButton)
{
    StartCoroutine(GiveAnswerRoutine(optionButton));
}
```

Figura 52: Script GiveAnswer.

Se necesita una corrutina *(GiveAnswerRoutine)* que pinte la opción con el color correcto/incorrecto, que se escuche el sonido pertinente y que se espere un tiempo determinado para poder ver dichos efectos. A esta corrutina hay que enviarle el *OptionButton*. Acto seguido reproduciremos el sonido para ello deshabilitaremos el *AudioSource* para poder reproducir un nuevo sonido. Donde sonará el sonido de acierto cuando sea la opción correcta y viceversa (se realizará mediante un operador ternario). De igual forma se hará para el color.

```
private IEnumerator GiveAnswerRoutine(OptionButton optionButton)
{
    if (m_audioSource.isPlaying)
        m_audioSource.Stop();

    m_audioSource.clip = optionButton.Option.correct ? m_correctSound : m_incorrectSound;
    optionButton.SetColor(optionButton.Option.correct ? m_correctColor : m_incorrectColor );

    m_audioSource.Play();

    yield return new WaitForSeconds(m_waitTime);
```

Figura 53: GiveAnswerRoutine.

Una vez que todo ello haya pasado, volveremos al método *NextQuestion.*

Por último, se definirá la función *GameOver*, que se encargará de finalizar el juego una vez que te has equivocado de pregunta o que el tiempo se ha terminado. Nos aparecerá una pantalla que indicará la puntación que se ha conseguido y saldrá al menú de inicio del juego.

En resumen, cuando comenzamos el juego se llama a la función *NextQuestion* la cual construye la interfaz gráfica para que cree la interfaz para una pregunta rondón que se está sacando de la base de datos, pasando el método *GiveAnswer* que se llamará cuando el jugador seleccione alguna opción, cuando esto ocurra se inicia la corrutina *GiveAnswerRoutine* donde se detendrá el AudioSource si se está reproduciendo algún sonido. Se colocará el audio que se quiera reproducir en función de si se acierte o no con su respectivo color y acto seguido se esperará un tiempo determinado para poder visualizar dichas características. Una vez acertada la pregunta, nos lleva a la siguiente. Si por un casual la respuesta es errónea, saldrá de la partida indicándonos la puntuación conseguida.

Este último Script *("GameManager")* se activa en la cámara y de esta forma se colocarán los colores, verde para la respuesta correcta y rojo para la respuesta incorrecta. De igual forma se asignarán los sonidos de acierto y error que se han descargado.

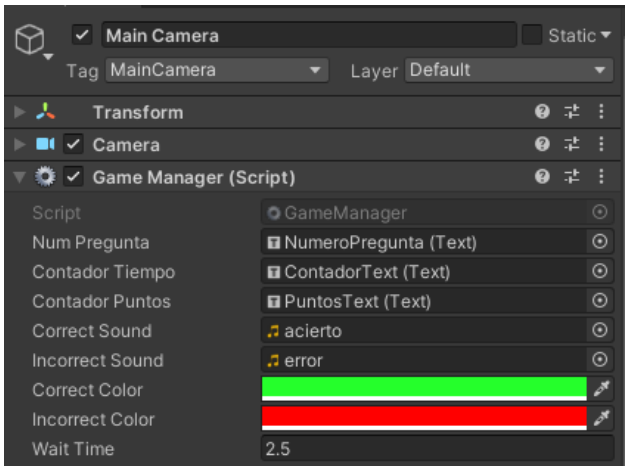

Figura 54: Script Asociado a MainCamera.

Hay que asignar en cada objeto todo lo necesario, como por ejemplo puede ser en el objeto *QuizUI* es necesario asignar el script *"QuizUI"* y dentro asignar el texto que se utilizará para cada pregunta y los 4 botones que se necesitarán añadir.

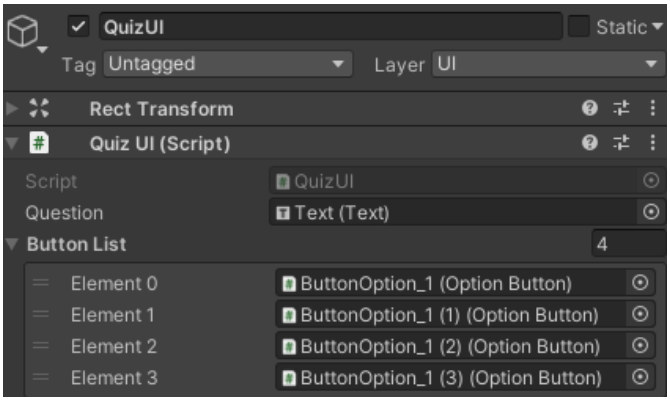

Figura 55: Objeto QuizUI.

De esta forma se finaliza la parte común de los diversos modos.

4.2 Programación de diversos modos

4.2.1 Modo "Fácil"

Para el modo "fácil" sería necesario añadir un temporizador hacia atrás desde el tiempo que queramos empezar y que una vez que acabe ese tiempo se finalice el juego.

```
public Text contadorTiempo;
private float tiempo = 10f;
```

Figura 56: Variable Tiempo.

```
void Update()
{
    tiempo -= Time.deltaTime;
    contadorTiempo.text = " " + tiempo.ToString( format: "f0");

    if (tiempo <= 0)
    {
        contadorTiempo.text = "0";

        GameOver();
    }
```

Figura 57: Script Cuenta atrás temporizador.

También hay que añadir un contador que nos ira diciendo el número de pregunta en el que estamos.

```
public Text numPregunta;
private int numero = 1;
```

Figura 578: Contador.

```
numPregunta.text = " " + numero;
```

Figura 569: Mostrar variable Contador.

4.2.2 Modo "experto"

Para el segundo modo, a parte del temporizador se le añadirá una ventana que nos ira avisando cuantos puntos vamos obteniendo, recordar que estos puntos dependen de la velocidad que se tarde en contestar la pregunta.

```
public Text contadorPuntos;
private float puntos = 0f;
private float puntuacion = 0f;
```

Figura 590: Variables.

```
puntuacion = puntuacion + puntos;
contadorPuntos.text = " " + puntuacion;
```

Figura 581: Mostrar variables por pantalla.

```
if (optionButton.Option.correct)
{
    if (tiempo >= 5)
    {
        puntos = 100f;
    }
    else
    {
        puntos = 50f;
    }
    NextQuestion();
```

Figura 60: Distribución de puntos.

4.2.3 Modo "Entrenamiento"

Por otro lado, en el tercer modo no será necesario disponer de temporizador debido a que el usuario podrá tardar el tiempo que quiera en responder la pregunta. Por otro lado, desaparecerá la función *GameOver* debido a que una vez que el usuario falle, podrá seguir respondiendo las preguntas.

```
if (optionButton.Option.correct)
{
    NextQuestion();
}
else
{
    NextQuestion();
}
```

Figura 61: Script Acertar pregunta.

Una vez que este quiera finalizar el entrenamiento, se ha habilitado un botón en la parte superior izquierda que al pulsarlo le lleva a la ventana de inicio.

4.2.4 Modo "Partida personalizada"

Finalmente, el cuarto y último modo es el de la partida personalizada, donde el usuario podrá elegir qué características quiere, esto lo hará a través de 3 *ImputFiles:*

```
[SerializeField] private InputField m_numPreguntas= null;
[SerializeField] private InputField m_numVidas = null;
[SerializeField] private InputField m_numTiempo = null;
```

Figura 62: InputFields necesarios.

Por otro lado, la función de *GameOver* también tendrá modificaciones, debido a que la partida finalizará en el momento en el que el jugador se quede sin vidas o en cuando se ha respondido a todas las preguntas que quería el usuario.

Por lo que la programación será de la siguiente forma:

```
numero++;
numVidas = numVidas - 1;
if (numVidas == -1 || numero == numPregTotal)
{
    GameOver();
}
else
{
    NextQuestion();
}
```

Figura 63: Script GameOver.

Cada vez que se responda una pregunta se ira sumando en una unidad la variable número, que en cuanto sea igual al número de preguntas que se quieren responder se finalizará la partida, por otro lado, cada vez que se falle una pregunta se restará una vida de forma que si se acaban también se cerrará el programa. Si por un casual el usuario dice que no quiere ninguna vida se entenderá que quiere responder a todas las preguntas elegidas, así que la partida finalizará únicamente cuando se respondan todas mostrando en una pantalla el número de aciertos en función de las preguntas que ha respondido.

Por ultimo y no menos importante se ha programado también el cambio entre escenas en Unity, que se realiza a través del siguiente script:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

1 asset usage
public class ChangeSceneWithButton : MonoBehaviour
{

    1 asset usage
    public void LoadScene(string sceneName)
    {
        SceneManager.LoadScene(sceneName);
    }
}
```

Figura 64: Script Cambiar escena.

Ese script hay que añadirlo en el inspector de cada botón (que cambie de escena) indicando a que escena queremos ir.

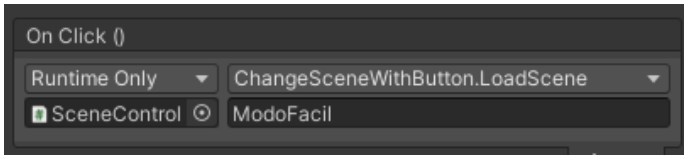

Figura 65: Añadir Script al Inspector.

De esta forma al clicar en el modo fácil te llevará directamente a dicha escena.

Existirán varios cambios de escena a la hora de acceder a los diferentes modos, es decir la aplicación tiene cinco escenas, la primera donde encontramos la portada, el Login/Registro y la pantalla de inicio. Las otras se corresponden a cada modo existente, dicho de otra forma, cada modo es una escena diferente.

CAPÍTULO 5. Diseño de la APP.

Se mostrará mediante imágenes el diseño
de la aplicación finalizada.

5.1 Diseño global

Se podría decir que el diseño de la aplicación se divide en cuatro partes:

En primer lugar, la portada, donde aparecerá el título del juego, con los logos correspondientes y un botón donde comenzar a navegar por la aplicación.

Figura 66: Portada Aplicación.

Acto seguido una vez pulsado el botón se accede a la segunda parte que es la de Login y Registro (nos la saltaremos porque se ha comentado anteriormente), donde el usuario se registrará en un pequeño formulario, y esos datos serán almacenados a una base de datos, que se comprobarán a la hora de hacer el Login.

Después pasas a la tercera parte de la aplicación, una vez que el usuario se ha logueado de manera satisfactoria se accede a la pantalla principal del juego en el que se puede observar los diversos modos que tenemos y un botón para salir de la aplicación en el momento que se quiera (ese botón le llevará a la portada).

Figura 67: Pantalla principal.

En la pantalla de inicio aparte de los diversos modos de juego se observa también que hay instrucciones que te explican cada modo, de esta forma el usuario dispondrá de una explicación previa antes de comenzar el Quiz.

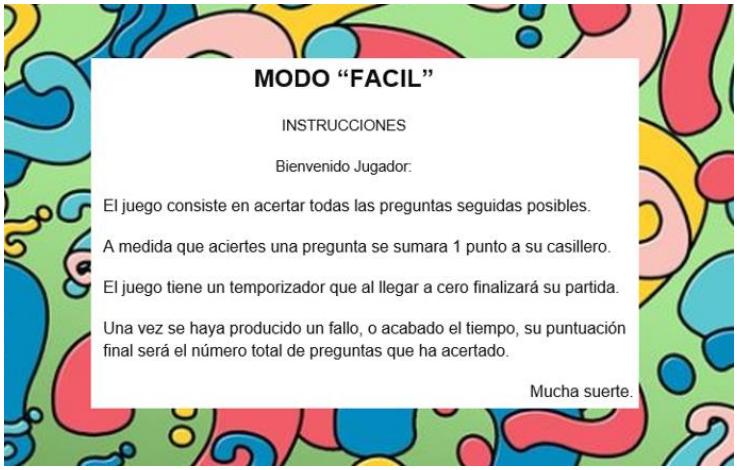

Figura 68: Instrucciones Modo "Fácil".

5.2 Diseño de los diferentes modos

Ahora se pasará a comentar el diseño de cada uno de los modos:

El diseño principal de todos los modos tendrá la misma forma:

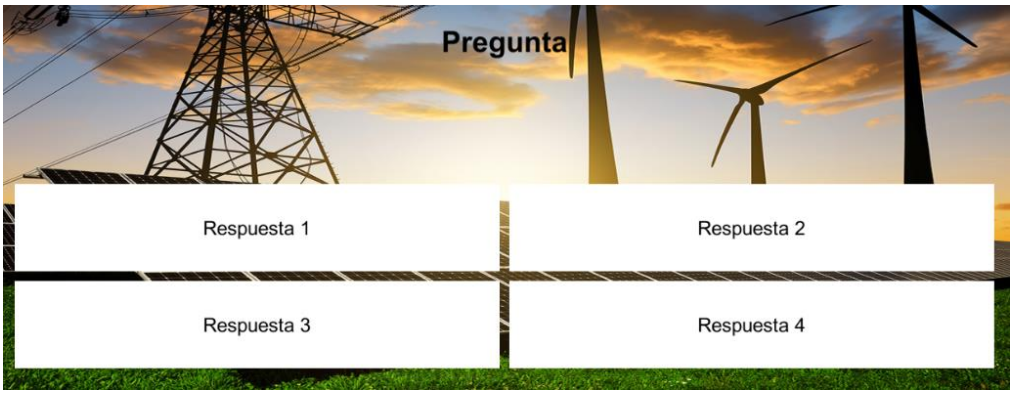

Figura 69: Diseño Quiz.

Poniéndose en verde cuando se acierta y en rojo si se falla.

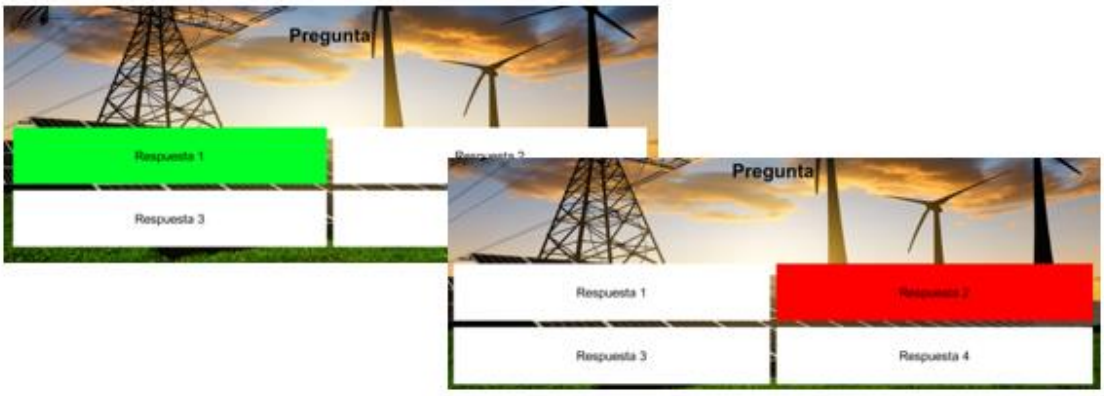

Figura 70: Acierto y error.

Lo que cambia de cada modo es la parte superior donde en cada uno mostrará un tipo de puntuación, el tiempo o simplemente un botón para salir de la partida como puede ser en el caso del modo “Entrenamiento”.

5.2.1 Diseño modo “Fácil”

El diseño de esta parte tiene un marcador que te dice el número de pregunta que vas en la esquina superior izquierda y el temporizador que marca el tiempo que te queda para cada pregunta.

Figura 71:Diseño modo “Fácil”.

Una vez que finaliza la partida aparece una pantalla diciéndote la puntuación final que has hecho.

Figura 72: Diseño Partida Finalizada.

En dicha ventana se ve un botón de cerrar al margen derecho superior y un botón en la parte inferior derecha “Récord” donde ira almacenando la puntuación máxima que consigue el jugador.

Figura 73: Ventana récord.

Si el usuario lo desea aparece la opción de borrar el récord y empezar de cero.

5.2.2 Diseño modo "Experto"

La parte de la pantalla de la puntuación inicial y el récord es igual al modo anterior, exceptuando que en el récord no almacena las respuestas correctas respondidas sino la máxima puntuación alcanzada, recordando que en este modo a mayor velocidad de respuesta mayor bonificación se conseguirá.

La parte del Quiz tiene además del número de pregunta y temporizador un recuadro que te dice que puntuación vas en ese momento.

Figura 74: Diseño modo "Experto".

5.2.3 Diseño modo "Entrenamiento"

Al clicar en la pantalla de inicio al botón "Entrenamiento" este te llevará a una ventana donde da la opción de la energía renovable que quieres practicar, hay tres tipos, tenemos la energía solar, la eólica y otras energías renovables.

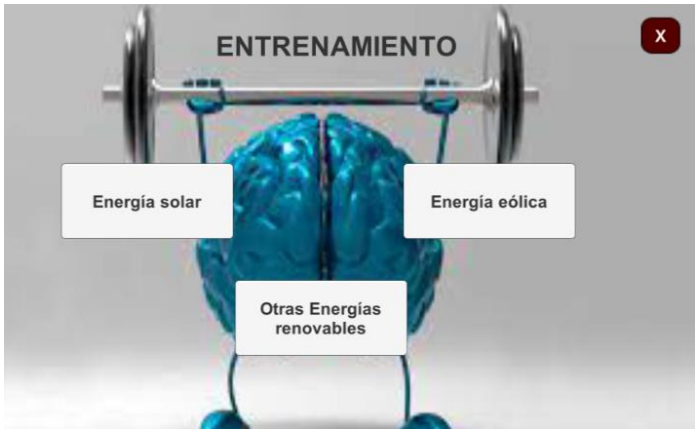

Figura 75: Pantalla modo "Entrenamiento".

La parte superior del Quiz solo tendrá un botón en la parte derecha para cuando el usuario quiera finalizar su entrenamiento.

Figura 76:Diseño modo "Entrenamiento".

5.2.4 Diseño modo “Partida personalizada”

Cuando estas en la pestaña de inicio y das al botón de “Partida personalizada” aparecerá una ventana donde el usuario pondrá las características que desea, también se observa un botón “Play” que una vez el usuario ponga sus particularidades deberá pulsar para comenzar el Quiz.

Figura 77: Ventana "Partida Personalizada".

En la parte superior del Quiz se puede ver el número de pregunta en el que estas, las vidas que te quedan y el tiempo.

Figura 78: Diseño modo “Partida personalizada”.

Al finalizar la partida aparecerá una ventana diciéndote cuantas preguntas has acertado.

CAPÍTULO 6. Ejemplo práctico.

Se expondrá un ejemplo de funcionamiento
de la app por un nuevo usuario.

6.1 Ejemplo práctico

Se creará un usuario “ficticio” para poder explicar un ejemplo práctico de la aplicación, este usuario tendrá los siguientes datos:

- Nombre: Luis
- Email: Luis96@gmail.com
- Contraseña: 12345

En primer lugar, accederá a la pantalla “comenzar” donde deberá pulsar dicho botón. Acto seguido, le aparecerá la pestaña de loguearse, al tratarse de un usuario nuevo deberá registrarse primero:

Figura 79: Registro Fallido.

Al registrarse ve que ya hay alguien con ese nombre así que lo cambia y pasa a llamarse LuisDiaz, si por un casual se le hubiera olvidado rellenar algún dato le hubiera saltado un aviso.

Figura 812: Registro de Nuevo Usuario.

Figura 8380: Login Usuario.

Una vez registrado de forma satisfactoria, se loguea correctamente apareciendo en la pantalla de inicio, donde puede elegir el modo que desee.

Quiere hacer una partida personalizada con las siguientes características:

- Número de preguntas: 10
- Vidas: 3
- Tiempo por pregunta: 10 seg.

Figura 82: características personalizadas.

Comienza a jugar y falla dos preguntas perdiendo así dos vidas:

Figura 83: "Partida personalizada" en curso.

Finalmente termina la partida sin ningún error más acertando de esta forma 8 preguntas de 10, una vez que finaliza por pantalla le sale su resultado final:

Figura 84: Puntuación conseguida.

Al no estar contento con su resultado, decide practicar en “energía eólica” debido a que son las preguntas sobre las que mas ha fallado en su “partida personalizada”.

De esta forma comienza su entrenamiento en energía eólica:

Figura 85: Entrenamiento "Energía eólica".

Transcurrido un tiempo, cree que ya está preparado para el Modo “experto”, y así ver capaz de cuantos puntos es capaz de conseguir y de intentar batir su récord que está en 550 puntos.

Responde unas cuantas preguntas bien, pero finalmente se equivoca y falla la respuesta.

Figura 86: Partida Modo "Experto".

Una vez se ha fallado la pregunta, le dice que puntuación ha conseguido, y va a mirar si ha superado su récord anterior, debido a que no se acuerda:

Figura 87: Récord Usuario.

Una vez completada la partida en Modo “experto” quiere finalizar su tiempo en la aplicación pulsando por consiguiente el botón “Salir” de la pantalla de inicio.

CAPÍTULO 7. Conclusiones y líneas futuras.

Se comentarán de forma breve las conclusiones extraídas del proyecto y se presentarán las posibles líneas futuras que complementan este trabajo.

7.1 Conclusiones

En este proyecto se ha logrado cumplir con el objetivo principal el cual era desarrollar una aplicación propia para utilizar como herramienta en la asignatura “Energía y Telecomunicaciones”, dentro del enfoque de gamificación que se emplea en la misma.

Al tener que buscar preguntas en los apuntes de dicha asignatura se han afianzado y aprendido conceptos de las energías renovables.

Por otro lado, en la parte del software es en la que más he aprendido debido a que he empezado de cero en un programa que no había oído hablar nunca de él como es el caso de Unity, y me ha servido para mejorar la capacidad de aprendizaje y resolución de los problemas que se presenten. También he aprendido a manejar a nivel usuario el lenguaje Php y a perfeccionar mi programación en lenguaje #C.

En relación con los otros objetivos que se buscan en este proyecto creo que han sido resueltos de manera satisfactoria, debido a que creará mayor motivación a los alumnos por conseguir una mejor puntuación y les servirá para aprender conceptos y afianzarlos de forma más “lúdica”.

En resumen, este proyecto me ha hecho aprender muchos aspectos en relación con diversos softwares que no había utilizado nunca y a aprender a programar con mayor soltura.

Ha sido un proyecto bastante entretenido de realizar, pero también algo estresante debido a que quería alcanzar otros objetivos los cuales no he sido capaz y se dejarán para líneas futuras y para la mejora de esta aplicación. Por otro lado, es bastante gratificante el hecho de ver que funciona algo en lo que has invertido bastante tiempo.

7.2 Líneas futuras

Como posibles líneas futuras se puede destacar el dar una vuelta a la aplicación y poder transformarla en una aplicación multijugador (objetivo que no se ha conseguido y he mencionado anteriormente en las conclusiones), es decir que un usuario pueda crear partidas privadas y puedan acceder a esa partidas más gente, de esta forma todos las personas responderán a las mismas preguntas.

Una vez observado si es de gran ayuda al alumnado este tipo de aplicaciones podrían diseñarse más para otras asignaturas de forma que se aprenda de forma más entretenida y así recordar mejor los conceptos básicos de las energías renovables o de otros aspectos pertinentes.

REFERENCIAS

[1] https://climate.selectra.com/es/que-es/energias-renovables

[2] https://www.muycomputer.com/2017/06/16/energias-renovables/ *[Imagen]*

[3]https://www.consumoresponde.es/art%C3%ADculos/las_energias_renovables_caracteristicas_y_tipos

[4]http://www.lineaverdehuelva.com/lv/consejos-ambientales/energias-renovables/Beneficios-de-las-energias-renovables-en-Espana.asp

[5] https://tusplacassolares.es/energias-renovables-y-no-renovables/

[6] https://www.thinglink.com/scene/931241073097834498 *[Imagen]*

[7] https://climate.selectra.com/es/que-es/energias-renovables

[8] https://climate.selectra.com/es/que-es/energias-renovables

[9]https://encolombia.com/medio-ambiente/interes-a/energia-geotermica/ *[Imagen]*

[10]https://www.expansion.com/especiales/2021/06/05/60b8c916468aeb113e8b4573.html

[11]https://www.expansion.com/especiales/2021/06/05/60b8c916468aeb113e8b4573.html

[12]https://www.ree.es/sites/default/files/07_SALA_PRENSA/2021/02/image/210131_Generacion_enero.jpg *[Imagen]*

[13] https://negociosyestrategia.com/blog/ventajas-videojuego-unity/

[14] https://www.masterd.es/blog/que-es-unity-3d-tutorial

[15] https://niixer.com/index.php/2020/11/10/que-es-unity-hub/ *[Imagen]*

[16] https://niixer.com/index.php/2020/11/10/que-es-unity-hub/

[17] https://ull-esit-dsi-1617.github.io/estudiar-las-rutas-en-expressjs-alberto-diego/Diego/Atom/queesatom.html

[18] https://anthoncode.com/logo-atom-software-vector-png/ *[Imagen]*

[19] https://ubunlog.com/como-instalar-atom-en-ubuntu/ *[Imagen]*

[20] https://www.ecured.cu/XAMPP

[21] https://www.coriaweb.hosting/nos-ofrece-phpmyadmin/

[22] https://www.jetbrains.com/help/rider/Introduction.html

[23] https://www.jetbrains.com/es-es/rider/ *[Imagen]*

BIBLIOGRAFÍA

1) https://unity.com/es

2) https://docs.unity3d.com/es/530/Manual/UnityManual.html

3) https://unity.com/es/learn/get-started

4) https://www.phpmyadmin.net/

5) Sue Blackman, *"Unity for Absolute Begginers".*

6) Marc Lidon, *"Unity 3D".*

7) https://atom.io/

8) https://www.jetbrains.com/es-es/rider/

9) https://www.apachefriends.org/es/index.html

10) Jesús Mirapeix Serrano, *Apuntes de la asignatura "Energía y Telecomunicaciones".* Grupo de Ingeniería Fotónica

11) https://ocw.unican.es/pluginfile.php/193/course/section/97/Apuntes%20EyT%20Intro%202021.pdf

12) https://ocw.unican.es/pluginfile.php/193/course/section/97/Apuntes%20EyT%20Solar%202021.pdf

13) https://ocw.unican.es/pluginfile.php/193/course/section/97/Apuntes%20EyT%20Eolica%202021.pdf

14) https://ocw.unican.es/pluginfile.php/193/course/section/97/Apuntes%20EyT%20OtrasEERR%202021.pdf

ANEXO I. Códigos

Se expondrán parte del código utilizado en el proyecto.

Código base de la programación del Quiz principal (el resto del código esta explicado en la parte superior en el apartado de programación)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class GameManager : MonoBehaviour
{
   //VARIABLES A UTILIZAR

   [SerializeField] private AudioClip soundCorrect = null; //sonidos a utilizar al acertar
   [SerializeField] private AudioClip soundIncorrect = null; //sonido para cuando se falle
   [SerializeField] private Color colorCorrect = Color.black; //color de la opcion correcta
   [SerializeField] private Color colorIncorrect = Color.black; //color si se falla
   [SerializeField] private float waitTime = 0.0f; //tiempo de espera entre preguntas

   private QuizBD quizDB = null; //hacer uso base de datos
   private QuizUI quizUI = null; //uso interfaz grafica
   private AudioSource audioSource = null; //para poder emitir sonidos

   private void Start() //empezar juego
   {
      //VARIABLES MOSTRAR PANTALLA


      quizDB = GameObject.FindObjectOfType<QuizBD>(); //referencia a la BD
      quizUI = GameObject.FindObjectOfType<QuizUI>(); //referencia a la interfaz
grafica (UI)
      audioSource = GetComponent<AudioSource>(); //referencia audios

      NextQuestion(); //pasa directamente a una pregunta
   }
   private void NextQuestion() //para hacer la siguiente pregunta
   {

      //VARIABLES

      quizUI.Construct(quizDB.GetRandom(), GiveAnswer); //para sacar una pregunta
aleatoria de la BD
   }
   private void GiveAnswer(ButtonOption buttonOption) //callback que se llama cuando
el jugador seleccione una resuesta
   {
      StartCoroutine(GiveAnswerRoutine(buttonOption)); //verificar si la respuesta es
correcta, poner color y sonido
   }
```

```
 private IEnumerator GiveAnswerRoutine(ButtonOption optionButton) //
    {

        if (audioSource.isPlaying) //si se escucha algun sonido
            audioSource.Stop(); //se para para poner el sonido que queremos

        audioSource.clip = optionButton.Option.correct ? soundCorrect : soundIncorrect;
//asignamos sonido de opcion correcta o incorrecta
        optionButton.SetColor(optionButton.Option.correct ? colorCorrect : colorIncorrect);
//asignamos color de la opcion corrrecta o incorrecta

        audioSource.Play(); //retomamos el sonido inicial

        yield return new WaitForSeconds(waitTime); //esperar tiempo para ver sonido y
cambio de color y si es true o false

        if (optionButton.Option.correct) //Si opción correcta
        {
            //OPERACIONES QUE SE DESEE

            NextQuestion();
        }
        else
        {
            GameOver();
        }
    }

    private void GameOver() //para cuando fallas
    {
        //OPERACIONES QUE SE DESEE

        ShowPantallaFinal(); //Sale de la partida
    }
    void Update()
    {
        //CONTROLAR VARIABLE TEMPORIZADOR

        if (tiempo > 0 )//tiempo positivo y el temporizador este en funcionamiento
        {
            tiempo -= Time.deltaTime; //resta segundo a segundo
            contadorTiempo.text = " " + tiempo.ToString("f0");
        }
        if (tiempo <0)
        {
            GameOver();
        }
    }
    public void ShowPantallaFinal()
    {
        //ACTIVAR O DESACTIVAR GAMEOBJECT
    }
}
```

ANEXO II. Ejemplos de preguntas

Se expondrán algunas preguntas que aparecen en la aplicación.

Todas las preguntas se pueden encontrar en el siguiente link: (https://docs.google.com/document/d/1E_M-7YczvTXbkhPciwdOfPHCmGdjDxpB/edit?usp=sharing&ouid=101154936922919242410&rtpof=true&sd=true).

1. **El uso de energías renovables no afecta a los recursos naturales para las generaciones futuras.**
   a. Falso **b. Verdadero**

2. **Las fuentes de energía renovable no emiten gases de efecto invernadero.**
   a. Verdadero b. Falso

3. **¿Qué diferencia hay entre la biomasa y el biocombustible?**
   a. Ninguna Diferencia
   b. El biocombustible se refiere a la materia orgánica como la madera y los desechos y la biomasa se extrae de la materia orgánica para crear combustibles.
   c. La biomasa se refiere a la materia orgánica como la madera y los desechos y el biocombustible se extrae de la materia orgánica para crear combustibles.
   d. Ninguna de las anteriores.

4. **¿Qué significa que una energía sea renovable?**
   a. No se pueden auto regenerar, se usan una vez y se gastan.
   b. Son aquellas energías contaminantes.
   c. Se puede auto regenerar y se les consideran inagotables.
   d. Son las llamadas energías "alternativas".

5. **¿Cuáles energías son no renovables?**
   a. Eólica, maremotríz, geotérmica, hidráulica.
   b. Petróleo, carbón, geotérmica y gas natural.
   c. Petróleo, solar, geotérmica, nuclear.
   d. Petróleo, carbón, gas natural, nuclear, biomasa.

6. **¿Quién descubrió la energía solar como "fuente alternativa de energía"?**
   a. Edmond Becquerel
   b. Charles Fritts.
   c. Calvin Fuller.
   d. Ninguno de los anteriores.

7. **La preferencia para orientar los captadores de radiación solara es:**
   a. Sur.
   b. Norte.
   c. Este.
   d. Oeste.

8. **La transformación parcial de la energía luminosa en energía eléctrica se denomina efecto ...**
   a. Piezoeléctrico.
   b. Fotovoltaico.
   c. Solar.
   d. Ninguna de las anteriores.

9. **La energía eólica procede del viento que es producido por el movimiento de grandes masas de aire debido a la diferencia de:**
   a. Ninguna de las anteriores.
   b. Presión.
   c. Velocidad del aire.
   d. Temperatura.

10. **La torre de un aerogenerador soporta:**
    a. Palas.
    b. Generador.
    c. Multiplicador.
    d. Rotor y góndola.

11. **De las siguientes afirmaciones, ¿Cuál es la función de un multiplicador?**
    a. Transforma la energía mecánica en energía eléctrica alterna.
    b. Permite el accionamiento del giro de las palas sobre su eje.
    c. Incrementa la velocidad de giro que recibe rotor para adaptarla a las necesidades del generador.
    d. Coloca siempre el rotor de manera perpendicular al viento

12. **¿Qué tipo de turbina eólica es el más utilizado actualmente?**
    a. Bipala.
    b. Tripala.
    c. Giromill.
    d. Windside.

13. **Como se llama el muro que retiene el agua y provoca una elevación notoria del nivel del río:**
    a. Aliviadero
    b. Azud
    c. Toma de agua.
    d. Presa.

14. **¿Qué tipo de energía del mar consiste en cerrar una bahía o un estuario con un dique?**
    a. Energía undimotriz.
    b. Energía azul.
    c. Energía maremotriz.
    d. Energía maremotérmica.

15. **¿Qué tipo de energía del mar consiste en el aprovechamiento energético producido por el movimiento de las olas?**
    a. Energía undimotriz.
    b. Energía azul.
    c. Energía maremotriz.
    d. Energía maremotérmica.

16. La energía cinética es generada por efecto de….
    a. La mecánica de las aspas
    b. La luz.
    c. La electricidad.
    d. Las turbinas de aire.

17. ¿Porque la torre mantiene la góndola a una gran altura?
    a. Porque la velocidad del viento es menor.
    b. Porque al tener más gravedad las aspas giran más rápido.
    c. **La velocidad del viento es mayor.**
    d. Ninguna de las anteriores.