IES MARCOS ZARAGOZA

CICLO FORMATIVO DE GRADO SUPERIOR EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

KILL THE MONSTER ADVENTURE GAME.

Abel Anaya Pardo 2º DAM

Índice

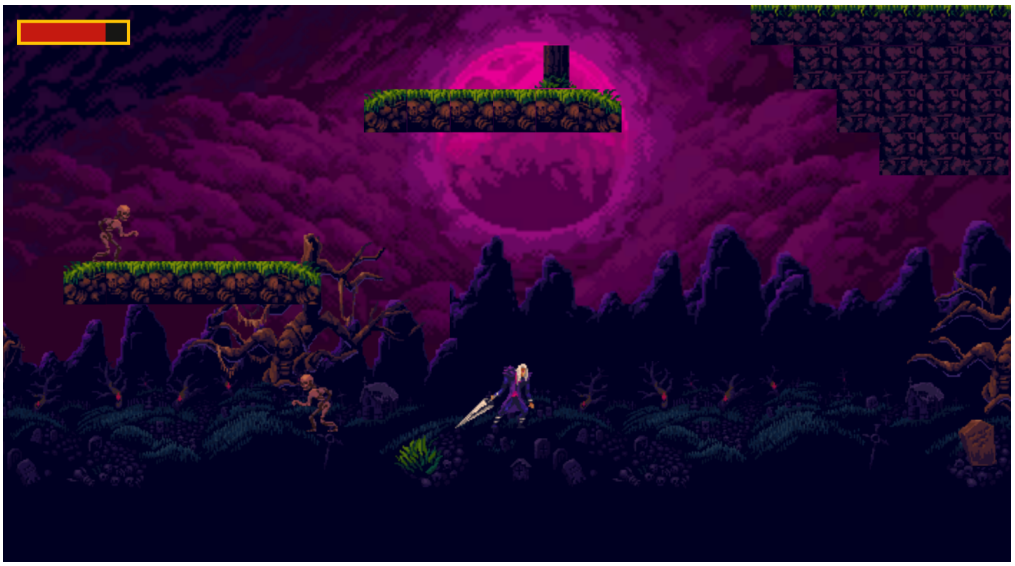
1- Introducción y motivación.

Cuando hablamos de motivación se habla de algo que te impulsa a seguir hacía adelante o te "guia" a conseguir tus metas, también puede ser algo que simplemente te gusta y yo tengo claro que una de las cosas que más me gusta y motiva son los videojuegos.

Por eso no dude a la hora de decidirme en hacer de mi TFG un videojuego aunque eso suponga como dicen algunos "meterme en un jardín", la experiencia ha sido divertida incluso cuando me tiraba horas para conseguir que una tontería funcionara y al final lo conseguía, la sensación era parecida a la que tienes derrotas a un jefe de un nivel que te ha estado destrozando durante toda la tarde y al final lo consigues, por eso se podría decir que el propio TFG ha sido un propia motivación para mi mismo.

2- Descripción del proyecto.

El juego hasta basado en los antiguos juegos de la década de los 80 - 90, un juego de plataformas en scroll lateral en 2D en el que tendrás que conseguir coleccionable, derrotando enemigos y por supuesto enfrentando a un jefe al final del nivel que le pondrá las cosas difíciles a más de un jugador/a.



3- Programas utilizados.

3.1- Unity.

Unity es un motor de videojuego multiplataforma creado por Unity Technologies. Unity está disponible como plataforma de desarrollo para Microsoft Windows, Mac OS, Linux. La plataforma de desarrollo tiene soporte de compilación con diferentes tipos de plataformas.

La versión que se ha utilizado en este proyecto ha sido: Unity 2020.3.18f1 (64-bit).

3.2 - Visual Studio Code.

Visual Studio Code es un editor de código fuente desarrollado por Microsoft para Windows, Linux, macOS y Web. Incluye soporte para la depuración, control integrado de Git.

La versión que se ha utilizado en este proyecto ha sido: 1.67.2 (la versión más actual).

3.3 - Postman.

Postman es una plataforma de API para que los desarrolladores diseñen, construyan, prueben e iteren sus API. Es una buena forma de "trastear" con tus APIs para poder hacer pruebas sobre ellas de forma segura.

La versión que se ha utilizado en este proyecto ha sido: 9.18.3 (la versión más actual).

3.4 - MongoDBCompass.

Es un sistema de base de datos NoSQL, orientado a documentos y de código abierto.En lugar de guardar los datos en tablas, tal y como se hace en las bases de datos relacionales, MongoDB guarda estructuras de datos BSON (una especificación similar a JSON) con un esquema dinámico, haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

Cabe destacar que en la base de datos está almacenada en Atlas un servidor del propio MongoDB, gratuito siempre y cuando no se alcance cierta cantidad de almacenaje.

Versión que se ha utilizado en este proyecto ha sido: 1.31.3.

3.5- GitKraken.

Aplicación de escritorio utilizada para sincronizarse con GitHub en web, muy eficiente para subir, realizar cambios y descargar proyectos propios y ajenos, siempre y cuando estén en público.

Versión que se ha utilizado en este proyecto ha sido: 8.5.0.

3.6 Programas secundarios.

Estos programas no han sido imprescindibles para el proyecto o son sitios web pero han ayudado al desarrollo del mismo.

Wondershare Filmora 11: Programa para grabar la pantalla, imprescindible para grabar los videos de prueba.

GitHub: Previamente mencionada ya que es donde se almacena todo el desarrollo del TFG.

YouTube: Imprescindible para buscar información o tutoriales de como solucionar algún problema.

mp3cut.net: Muy útil para poder dejar la música del juego a punto y que encajen bien los tiempos.

4- Componentes principales de juego.

4.1- Personaje.

Nuestro personaje y protagonista de este juego es el encargado de limpiar el nivel de todos los monstruos con lo que se va a ir encontrando, para eso tiene su arma.

El personaje está programado para que detecte el suelo y así no caer al vacío, también al hacer la animación de atacar esta creará un "Rigidbody2D" y cuando alcance a los enemigos esta le hará daño a los mismos

Cuando atacamos se ejecuta este método:

```
public void Atacar()
{
    if(Input.GetButtonDown("Fire1"))
    {
        anim.SetBool("atacar", true);
        AudioManager.instance.PlayAudio(AudioManager.instance.hit);
    }
    else
    {
        anim.SetBool("atacar", false);
    }
}
```

Lo que hace es que al pulsar el botón izquierdo del ratón hacemos que el Rigidbody2D que hemos mencionado antes sea visible y pueda entrar en contacto con los enemigos, además de eso el ataque cuenta con un efecto de sonido y una animación propia.

El personaje inflige dos puntos de daño por cada ataque.

El personaje también es capaz de saltar, esto lo consigue gracias a que al Rigidbody2D del propio jugador le damos un fuerza de salto nosotros mismos.

```
public void Jump()
{
    rigidbody2D.AddForce(Vector2.up * jumpForce);
}
```

Podemos saltar cuando se cumplan estas dos condiciones a la vez, que son pulsar la tecla W del teclado y que el jugador esté en contacto con una superficie.

```
//Saltar
if (Input.GetKeyDown(KeyCode.W) && Grounded)
{
    Jump();
    AudioManager.instance.PlayAudio(AudioManager.instance.jump);
}
```

Esta acción también cuenta con un efecto de sonido y una animación propia.

Tanto la velocidad del personaje como la fuerza de salto se pueden cambiar de forma “pública” para hacer pruebas o para más diversión.

El personaje puede sufrir daños y perder vida al igual que ganarla (apartado 4.4.3 y 4.4.4).

Cuando el jugador entra en contacto con un elemento del nivel que tenga el tag “enemigo” este recibe daño en proporción a lo que tenga asignado dicho enemigo y la barra de vida baja la misma cantidad de daño infringido.

```
public void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.CompareTag("enemigo") && !Inmune)
    {
        vida -= collision.GetComponent<Enemigo>().damage;
        StartCoroutine(Inmunidad());
        AudioManager.instance.PlayAudio(AudioManager.instance.herido);

        if (vida <= 0)
        {

            vidaImg.fillAmount = vida = 0;
            Instantiate(efectoMuerte, transform.position, Quaternion.identity);
            Destroy(gameObject);

            AudioManager.instance.fondo.Stop();
            AudioManager.instance.PlayAudio(AudioManager.instance.muerte);

            Time.timeScale = 0;
            gameOverImg.SetActive(true);

            StartCoroutine(ResetPanel());
        }
    }
```

El personaje cuenta con 100 puntos de vida y cada enemigo quita una cantidad distinta, si la vida llega a 0 el personaje muere y el juego termina.

Cuando el personaje recibe daño también dispone de una efecto sonoro y durante unos segundos después de ser dañado será inmune y su cuerpo se cubrirá completamente de blanco para indicárselo al jugador.

4.2- Enemigos.

¿Y qué sería de un buen personaje sin unos villanos a la altura? En este proyecto tenemos 4 enemigos diferentes que pondrán las cosas difíciles a nuestro jugador.

4.2.1- Esqueletos normales.

Estos enemigos son los más simples y fáciles y se encuentran en las primeras zonas del nivel.

Estos esqueletos se dedican a patrullar de un lado a otro como si nada les importase, esto es gracia a tomo están programados.

En la función Update que se ejecuta a cada frame que pasa el juego ejecutándose el enemigo intenta detectar si el camino por donde está pisando termina o si se ha topado con un muro para que pueda darse la vuelta y seguir con su patrulla.

```
void Update()
{
    detectaFin = !Physics2D.OverlapCircle(chequear.position, radioDtectar, queSuelo);
    detectaMuro = Physics2D.OverlapCircle(chequearMuro.position, radioDtectar, queSuelo);
    detectaSuelo = Physics2D.OverlapCircle(chequearSuelo.position, radioDtectar, queSuelo);

    if (detectaFin || detectaMuro && detectaSuelo)
    {
        Vuelta();
    }
}
```

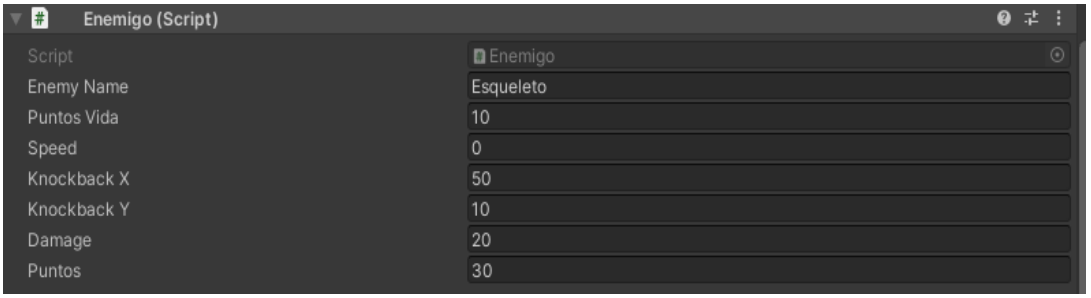
Consigue darse la vuelta gracias a este método:

```
public void Vuelta()
{
    movimientoDer = !movimientoDer;
    transform.localScale *= new Vector2(-1, transform.localScale.y);
}
```

Las estadísticas de este enemigo como el del resto son públicas para poder modificarlas de forma más sencilla a la hora de programar y hacer pruebas.

4.2.2- Esqueletos reforzados.

Estos esqueletos se encuentran en la zona intermedia del nivel y son prácticamente como los otros solo que tienen los huesos más duros por eso tienen más vida e infligen más daño al personaje



4.2.3- Esqueletos que disparan.

Estos enemigos se encuentran en la zona final del nivel y disparan una llama negra verdosa que pueden llegar a ser un problema para el jugador.

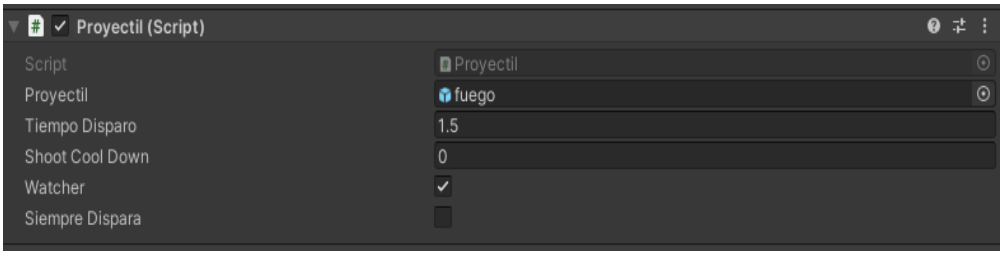
Estos enemigos disponen de dos tipos de ataques:

Disparo por tiempo > Este disparo se efectúa siempre cada 1.5 segundos hasta que el enemigo muera.

Disparo cuando detectan al jugador > En este la cosa es más complicada ya que hay que crear un script en el que el enemigo detecte al enemigo y entonces disparé, lo que hacemos es programar un BoxCollider2D que cuando entre en contacto con un objeto con el tag "Player" y activa el ataque.

```
public class DetectarJugador : MonoBehaviour
{
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if(collision.CompareTag("Player") && transform.GetComponentInParent<Proyectil>().watcher == true)
        {
            transform.GetComponentInParent<Proyectil>().Disparo(); //Ponemos "GetComponentInParent" ya que detectar enemigo es hijo de enemigo
        }
    }
}
```

Los enemigos que pueden disparar se les puede cambiar el estilo de disparo de forma sencilla en el inspector.



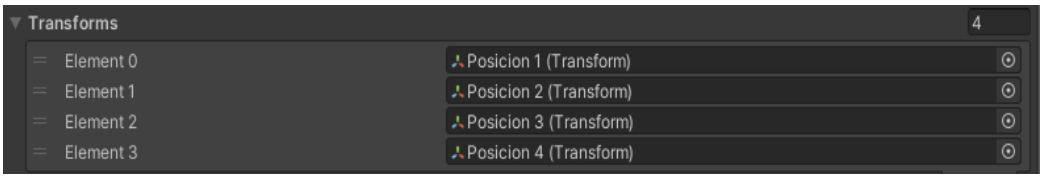
4.3- Jefe.

El jefe es una de las cosas más importantes que tiene que tener este tipo de videojuegos, así que como es lógico es el enemigo más difícil del nivel.

4.3.1- Teletransporte

El jefe está programado para que se teletransporte cada tres segundos de forma aleatoria en cuatro posiciones preestablecidas.

```
public void Teletransporte()
{
    var posicionInicial = Random.Range(0, transforms.Length);
    transform.position = transforms[posicionInicial].position;
}
```



4.3.2- Ataque

El ataque del jefe es parecido al de los enemigos que disparan cada x segundos, pero con una diferencia y es que la llama ira en la dirección en la que se encuentra el jugador en el momento que es disparada.

Esto lo conseguimos "guardando" la posición del jugador y después lanzar la llama en la misma dirección.

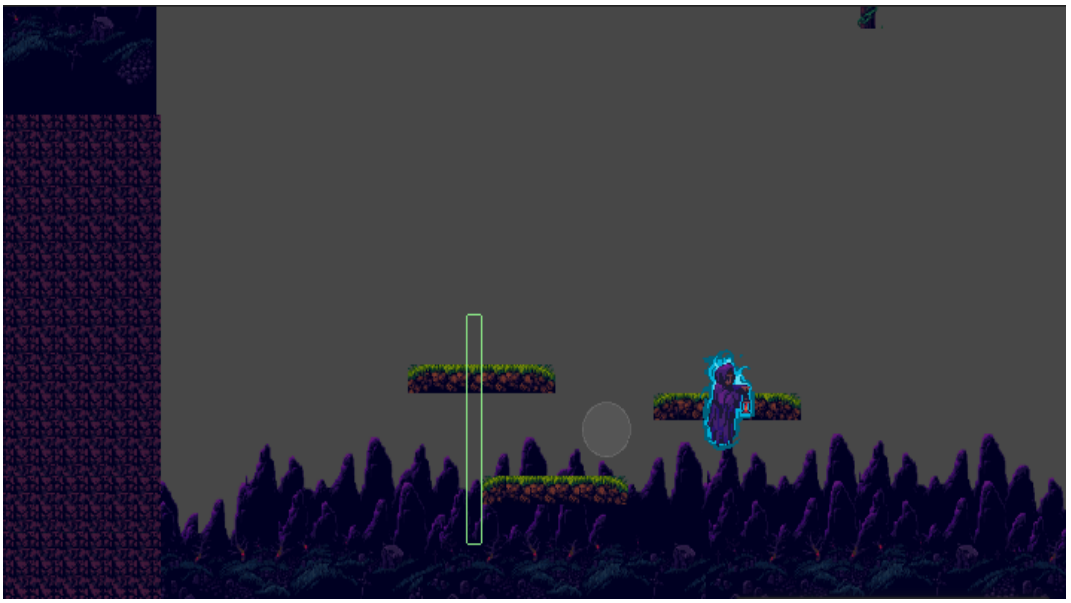
```
void Start()
{
    moveSpeed = GetComponent<Enemigo>().speed;
    rigidbody2D = GetComponent<Rigidbody2D>();
    target = JugadorMovimineto.instance;

    moveDireccion = (target.transform.position - transform.position).normalized * moveSpeed;
    rigidbody2D.velocity = new Vector2 (moveDireccion.x, moveDireccion.y);
}
```

También para darle algo de detalle el jefe siempre estará mirando en la dirección del jugador.

```
public void BossMirar() //Este metodo lo que hace es que el jefe siempre este mirando en la dirección del jugador
{
    if(transform.position.x > JugadorMovimineto.instance.transform.position.x)
    {
        transform.localScale = new Vector3(-1.0f, 1.0f, 1.0f);
    }
    else
    {
        transform.localScale = new Vector3(1.0f, 1.0f, 1.0f);
    }
}
```

4.3.3- Zona

La zona del enemigo está diseñada para que no puedas huir en ninguna dirección y sea un combate a muerte, o ganas tu o gana el.



Al pasar por el BoxCollider2D el muro de la derecha desciende y nos bloqueara el camino, a su vez el jefe aparecerá junto con su barra de vida (funciona de la misma forma que la barra de vida del jugador), al derrotarlo veremos si animación de muerte y unos segundos después la pantalla de victoria.

El jefe tiene dos formas de hacernos daño, si nos toca y si la llama nos alcanza.

4.4- Coleccionables.

Los coleccionables son también una parte esencial en lo videojuegos, así que cada objeto que consigamos y enemigo que matemos nos dará puntos para poder sumarlos a nuestro contador

4.4.1- Monedas.

La clásica moneda dorada de muchos juegos también se encuentra en este, al pasar por ella la moneda se destruirá y nos dará cinco puntos a nuestro marcador, cuenta con efecto de sonido.

4.4.2 - Gemas.

Lo mismo que las monedas pero con un valor de diez puntos, por tanto son más difíciles de conseguir.

4.4.3- Corazones.

Los corazones son un pequeño salvavidas para que podamos recuperar vida y conseguir completar el nivel, los corazones nos curan cinco puntos de vida y nos otorga dos puntos al marcador, también cuenta con efecto de sonido.

4.4.4- Poción.

Solo hay una poción en todo el nivel y se encuentra antes de llegar a la zona del jefe para poder enfrentarlo con todas las fuerzas posibles, la poción restaura 50 puntos de vida y otorga cuatro puntos al marcador, también dispone de efecto de sonido.

4.4.5- Puntos.

Los puntos obtenidos de eliminar enemigos o conseguir los coleccionables se almacenan para poder ser mostrados durante el juego.

```
public class ContadorPuntos : MonoBehaviour
{

    public static ContadorPuntos instance;

    private void Awake()
    {
        if(instance == null)
        {
            instance = this;
        }
    }

    public int PuntosTotales
    {
        get
        {
            return puntosTotales;
        }
    }

    private int puntosTotales;

    public void SumarPuntos(int puntosSuma)
    {
        puntosTotales += puntosSuma;
        Debug.Log(puntosTotales);
    }
}
```

Este script es llamado desde todos los componentes que tengan una puntuación que dar.

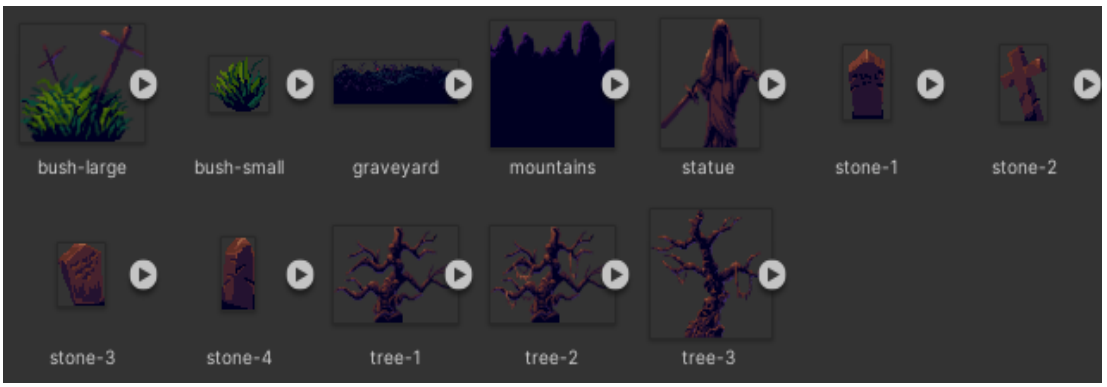
4.5- Música.

La música también es un pilar fundamental a la hora de hacer un videojuego, casi todas las acciones del juego tienen un sonido característico, ya sea que lo provoque el jugador o los enemigos.

La música es controlada por el script AudioManager.cs.

4.6- Entorno.

El entorno es donde transcurre la aventura y el escenario utilizado por el personaje para moverse, la distribución de como usar los elemento varía entre cada uno.



5- Conexiones.

5.1- API.

Se trata de un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones, permitiendo la comunicación entre dos aplicaciones de software a través de un conjunto de reglas.

```
var express = require('express');
var router = express.Router();
var usuarios = require("../controllers/usuarios");

const bodyParser = require('body-parser').json()

router.get("/usuarios", usuarios.list);
router.get("/usuarios/:nombreUsuario", usuarios.get);
router.get("/usuarios/:nombreUsuario/:password", usuarios.log); // LOGIN
router.post("/usuarios", bodyParser, usuarios.add); //BodyParser es el contenido que hay en el json
router.put("/usuarios/:nombreUsuario", bodyParser, usuarios.update); //Equivalente al update
router.delete("/usuarios/:id", usuarios.delete);

module.exports = router;
```

5.2- Login.

Para el login lo que haremos será utilizar “UnityWebRequest.Get” para llamar al usuario que coincida con los campos de escritura de “Usuario” y “Contraseña”.

```
public IEnumerator LoginClick()
{
    Debug.Log(usuario.text +contra.text);

    if(usuario.text == "" && contra.text == "")
    {

        Debug.Log("Rellena los campos");
    }
    else
    {

        UnityWebRequest www = UnityWebRequest.Get($"http://192.168.136.1:8080/usuarios/{usuario.text}/{contra.text}");

        yield return www.SendWebRequest();

        Debug.Log(www.downloadHandler.text);

        JSONNode data = JSON.Parse(www.downloadHandler.text);

        if(data)
        {
            nombreUsuario = data["nombreUsuario"];
            password = data["password"];
            puntos = data["puntos"];

            Debug.Log(nombreUsuario +" " +password +" " +puntos);
        }
        else
        {
            Debug.Log("El usuario no existe, intentelo otra vez");
        }
    }
}
```

5.3- Registro.

Para el registro lo que haremos será usar "UnityWebRequest.Post" para crear al nuevo usuario con su nombre y contraseña, la id y los puntos se añaden de forma hardcode a todos los usuarios creados.

La id se randomiza al hacer el Post y los puntos por defecto en 0.

```
public void BotonPulsado() // Obtenemos los datos de todos los usuarios pero nosotros solo mostraremos nombre y puntos
{
    StartCoroutine(Post(_id, nombreUsuario.text, password.text, punto));

}

public IEnumerator Post(string id, string usuario, string contra, string punto)
{
    if(nombreUsuario.text == string.Empty && password.text == string.Empty)
    {
        Debug.Log("Rellena los campos");
    }
    else
    {
        List<IMultipartFormSection> wwwForm = new List<IMultipartFormSection>();
        wwwForm.Add(new MultipartFormDataSection("id", id));
        wwwForm.Add(new MultipartFormDataSection("nombreUsuario", usuario));
        wwwForm.Add(new MultipartFormDataSection("password", contra));
        wwwForm.Add(new MultipartFormDataSection("puntos", punto));

        UnityWebRequest www = UnityWebRequest.Post("http://192.168.136.1:8080/usuarios", wwwForm);

        yield return www.SendWebRequest();

        if(www.isNetworkError || www.isHttpError)
        {
            Debug.Log(www.error); //Aparecera dicho mensaje alguna o las dos condiciones de arriba se cumplen
        }
        else
        {
            Debug.Log("Usuario creado");
        }
    }
}
```

5.4- Ranking.

En el registro lo que haremos será llamar a todos los usuarios y solo mostrar el nombre y los puntos en la tabla del ranking junto con quien tenga la puntuación más alta.

```
public IEnumerator GetRanking() // Obtenemos los datos de todos los usuarios pero nosotros solo mostraremos nombre y puntos
{
    //infoText.text = "Cargando datos...";
    UnityWebRequest www = UnityWebRequest.Get("http://192.168.136.1:8080/usuarios/");

    yield return www.SendWebRequest();


    if(www.isNetworkError || www.isHttpError)
    {
        Debug.Log(www.error); //Aparecera dicho mensaje alguna o las dos condiciones de arriba se cumplen
    }
    else
    {
        Debug.Log(www.downloadHandler.text); //Muestra el resulta como texto

        JSONNode data = JSON.Parse(www.downloadHandler.text);

        JSONNode id = data[0];

        foreach(JSONNode puntuacion in data)
        {
            if(id["puntos"] < puntuacion["puntos"])
            {
                id = puntuacion;
            }

            infoText.text += "Nombre: " +puntuacion["nombreUsuario"] +"/";
            infoText.text += "Puntos: " +puntuacion["puntos"] +"\n" +"\n";
        }

        infoText.text += "El usuario con mas puntos es " +id["nombreUsuario"] +" con " + id["puntos"] +" puntos.";
    }
}
```

6- Pruebas de funcionamiento efectuadas.

Las pruebas efectuadas se encuentran en formato mp4 en la carpeta "Documentación/Videos".

7- Bibliografía y recursos.

Uso de UnityWebRequest:
https://docs.unity3d.com/es/530/Manual/UnityWebRequest.html
https://www.youtube.com/watch?v=nVz3GBw1kDg

Guía de como consumir APIs en C#:
https://www.luisllamas.es/como-consumir-un-api-rest-como-clientes-con-c/

Assets de personaje, enemigos y entorno:
https://assetstore.unity.com/packages/2d/characters/gothicvania-cemetery-120509#description

Assets de las monedas y gemas:
https://laredgames.itch.io/gems-coins-free

Assets del corazón y la poción:
https://assetstore.unity.com/packages/2d/gui/icons/2d-pixel-item-asset-pack-99645#description

Script SimpleJSON:
https://github.com/Bunny83/SimpleJSON/blob/master/SimpleJSON.cs

Música y efecto de sonido:

Tema del menú principal >
https://www.youtube.com/watch?v=gbjngR7wwYA&t=8s

Tema del juego > https://www.youtube.com/watch?v=btgi3TPL3AE&t=55s

Sonido monedas > https://www.youtube.com/watch?v=62SH-x_H9oA

Sonido gema > https://www.youtube.com/watch?v=oy6_B52UDUo

Sonido corazon > https://www.youtube.com/watch?v=gCiTLZGqu9w

Sonido poción > https://www.youtube.com/watch?v=9hTlqG0C_-M

Efecto muerte jugador > https://www.youtube.com/watch?v=j_nV2jcTFvA

Música victoria > https://www.youtube.com/watch?v=vD9ZgJ8ETm4

8- Conclusión y valoración personal.

La conclusión que puedo sacar de este proyecto es que el juego cumple con casi los puntos mencionados y que he aprendido en muchísima más profundidad un lenguaje que durante el curso solo vimos de pasada (C#), además considero que el juego es completamente funcional y que puede ser perfectamente disfrutado por todo aquel que lo pruebe.

La valoración personal sobre este proyecto es que ha sido de las pruebas más duras que he tenido que afrontar durante mi vida académica pero considero que será un resultado satisfactorio para todos, tanto como lo ha sido para mi como para los profesores que me han enseñado y acompañado estos últimos 5 años.