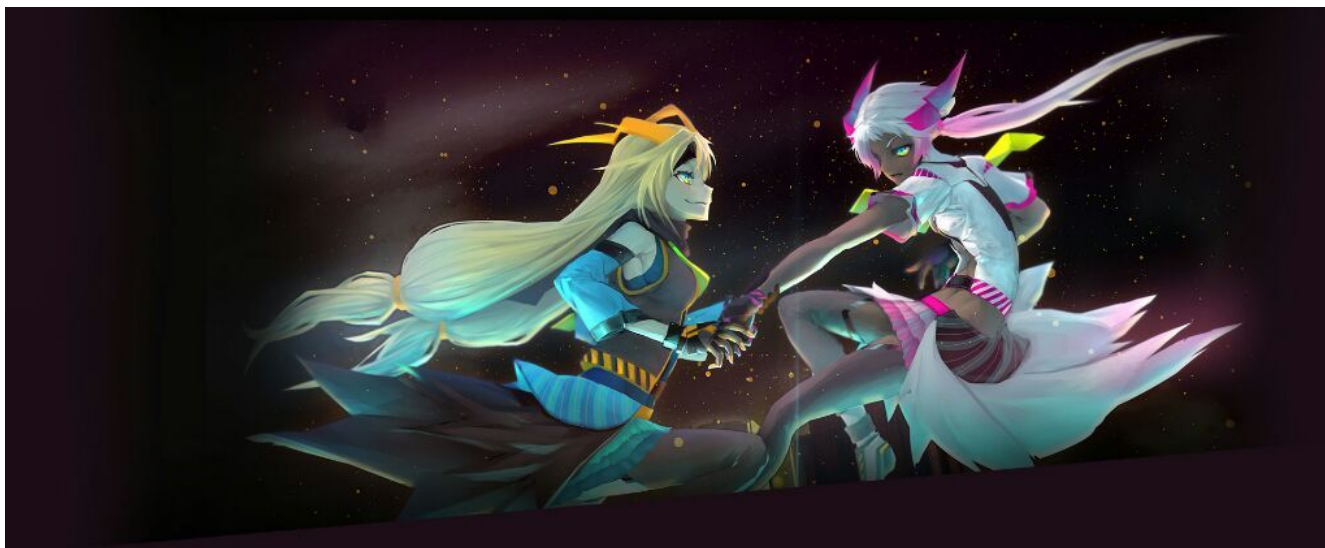


Sister Guardian's



Proyecto Fin de Curso IES Jaume II El Just

Jordi Sales Aleixandre

Tavernes de la Valldigna, 3 – 6 – 2019

Indice

1. PRESENTACIÓN.....	1
2. DESARROLLO.....	2
3. CONTENIDO.....	2
3.1 Escenas.....	2
3.2 Menú Inicio (Start Menú).....	3
3.3 Selección de Personaje (Character Selection).....	6
3.4 Escena del Juego (Game Scene).....	7
3.4.1 Entorno del terreno.....	7
3.4.2 UI dentro del juego.....	9
3.5 Personajes y Cámaras.....	11
3.5.1 Animator Controller.....	12
3.5.2 Sistema de Partículas.....	16
3.5.3 Cámaras.....	17
3.6 Inteligencia Artificial.....	17
3.6.1 Enemigos.....	17
3.6.2 SpawnPoints.....	19
3.7 Unity Genéricos.....	20
3.7.1 Rigidbody.....	20
3.7.2 Colliders.....	20
3.7.3 NavMesh.....	20
4. HERRAMIENTAS.....	21
5 ESTRUCTURA DE SCRIPTS.....	22
6 MEJORAS Y FUTURAS ACTUALIZACIONES.....	24
6.1 Juego Online.....	24
6.2 Objetos e Inventario.....	24
6.3 Razas, Profesiones y Habilidades.....	24
6.4 Sistema de Atributos.....	24
6.5 Quest.....	25
8 CONCLUSIONES.....	25

1. PRESENTACIÓN

El proyecto se trata de un videojuego hecho con el motor grafico de Unity, este juego se clasificaría en el tipo RPG (*Role Playing Game*) el cual trata de llevar a los jugadores a asumir el papel del personaje dentro del juego.

El protagonista tendrá que enfrentarse a los distintos enemigos que aparecerán por las distintas zonas del juego, para así ganar experiencia y poder subir de nivel.

Los usuarios, también podrán comprobar el progreso de su aventura, pudiendo acceder al “Ranking” de jugadores con mas nivel.

2. DESARROLLO

Para desarrollar los diferentes comportamientos dentro del juego se ha utilizado el lenguaje de programación C# para realizar los diferentes “scripts”, los cuales Unity se encargara de gestionar según asignemos estos a los diferentes objetos según nos convenga.

En cuanto a la base de datos, se ha utilizado un Servidor HTTP Apache el cual conectara con la base de datos MySQL. Todo esto mediante dos “scripts” escritos en PHP, uno encargado de la lectura de datos, y otro para actualizar la base de datos, este último tendrá dos funciones, insertar o actualizar los datos.

Para comprobar el funcionamiento y la correcta implementación del servicio, se ha instalado XAMPP y realizado la conexión a través del “localhost”.

3. CONTENIDO

3.1 Escenas

Las escenas son archivos los cuales contienen sus propios entornos, obstáculos y decoraciones, lo que nos permite diseñar y construir el juego en pedazos.

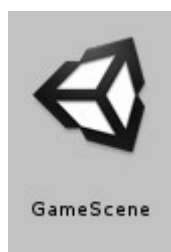
En este caso el juego esta dividido en cuatro escenas, la primera seria el Menú inicial del juego, la segunda sera la escena de selección de personaje, la tercera nos traerá al juego en si y por ultimo la cuarta escena sera para finalizar el juego, en caso de que el usuario decida finalizar este.



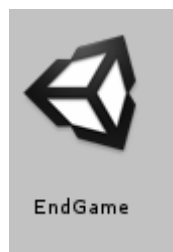
Esta escena contiene botones para que el usuario pueda navegar entre distintas opciones para poder iniciar, ajustar el juego a su gusto.



Escena donde podremos elegir entre dos personajes y el “nick name” que utilizaremos dentro del juego.



Escena con todo el contenido del juego y donde se desarrollara toda la actividad.



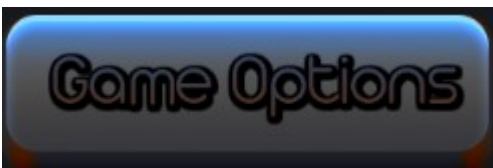
Escena final donde veremos una breve despedida del autor del juego.

3.2 Menú Inicio (Start Menú)

Escena inicial del juego, donde se presentaran al usuario diferentes opciones, el usuario podrá elegir entre ellas a través de unos botones.



Botón que inicia el juego y nos lleva a la selección de personaje.



Botón que nos llevara a las opciones del juego, donde podremos ajustar el sonido, la resolución de pantalla y si queremos que sea completa o no.

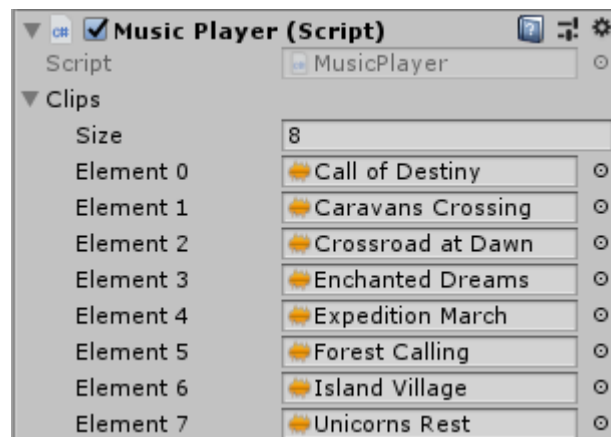


Botón para finalizar y cerrar el juego.

En esta escena también tendremos un reproductor de audio que nos estará reproduciendo una lista de canciones de forma aleatoria.

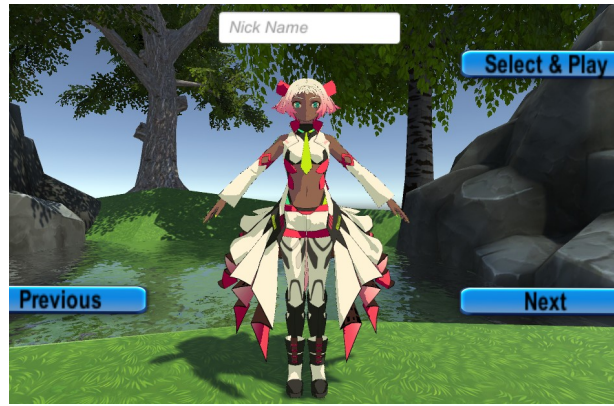
```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class MusicPlayer : MonoBehaviour
6  {
7      public AudioClip[] clips;
8      private AudioSource audioSource;
9      // Start is called before the first frame update
10     void Start()
11     {
12         audioSource = GetComponent<AudioSource>();
13         audioSource.loop = false;
14     }
15
16     private AudioClip GetRandom()
17     {
18         return clips[Random.Range(0, clips.Length)];
19     }
20     // Update is called once per frame
21     void Update()
22     {
23         if (!audioSource.isPlaying)
24         {
25             audioSource.clip = GetRandom();
26             audioSource.Play();
27         }
28     }
29 }
30
```

Unity nos permite llenar el Array desde el inspector y añadir o quitar canciones.



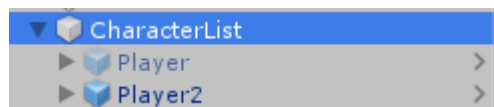
3.3 Selección de Personaje (Character Selection)

Esta escena cuenta con varios elementos UI (User Interface), un fondo decorativo, los personajes, una cámara que permanecerá estática y como en la anterior escena también tendremos el reproductor de música aleatoria.

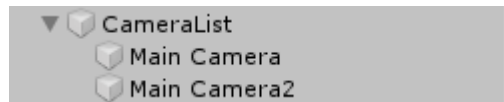


Los dos personajes se sitúan en la misma coordenada en el plano, los dos botones inferiores nos permitirán “cambiar” de personaje, aquí lo que realmente esta pasando es que mediante el “script” estamos haciendo visible uno de ellos y deshabilitando el otro, de esta manera conseguimos que el usuario crea que esta saltando de un personaje a otro.

Para realizar esta transición asignaremos un “script” a un objeto al que vamos a llamar **CharacterList** el cual contendrá los dos personajes.



Realizamos la misma acción para asignar la cámara al personaje (se ha decidido que cada personaje tenga asignada su propia cámara para evitar futuros problemas).



El campo de texto sera donde el jugador tendrá que escribir su “Nick name”.

Por ultimo el botón de jugar comprobará que el campo de texto no se encuentre vacío, ya que esto podría causar problemas en la base de datos, guardara el personaje seleccionado y el “Nick name” para pasar a la siguiente escena.

3.4 Escena del Juego (Game Scene)

3.4.1 Entorno del terreno

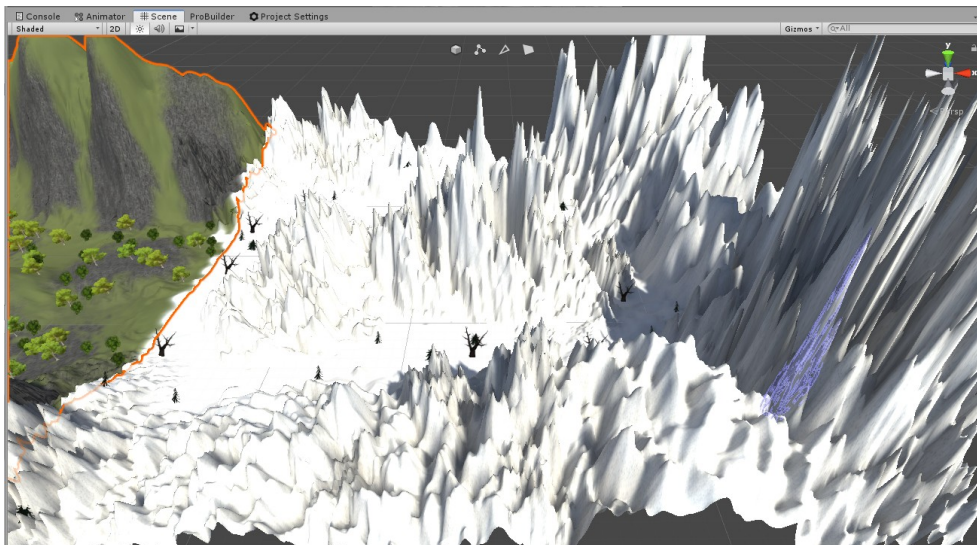
En esta escena es donde va a empezar la aventura el jugador y donde se desarrollan todas las actividades principales.

Empezando por el terreno o mapa del juego, se ha creado dos zonas distintas, para ser mas concretos tenemos 2 elementos “Terrain” (Objeto del editor de Unity).

El primero, es donde ubicamos la zona de inicio, donde empezara nuestro personaje, se caracteriza por ser una zona boscosa rodeada de montañas.






El segundo terreno, es la zona mas alejada del punto de inicio, caracterizado por montañas con forma mas puntiaguda y totalmente nevada, con escasos arboles.



Para decorar el terreno se han utilizado tres tipos de herramientas del editor de terreno, el “Paint Terrain”, el “Paint Tree” y el “Paint Details”, todos ellos utilizan “Prefabs” (sistema que utiliza Unity para guardar objetos ya diseñados y configurados para poder reutilizarlos cuantas veces queramos) para poder insertar elementos en nuestro terreno.

En el Paint Terrain podemos distinguir tres tipos de “Layers” (capas).

- La primera de color verde para simular la hierba 
- La segunda para dar el efecto roca a algunas zonas. 
- La tercera para el efecto nevado. 

En el Paint Tree se ha utilizado cinco tipos de arboles (“Prefabs”).



En la zona del bosque.



En la zona nevada.

En Paint Details he utilizado dos tipos de hierbas (“Pefabs”).



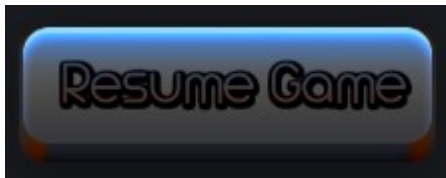
Solo en la zona del bosque.

Para finalizar con el terreno, también se ha añadido una zona de viento para simular el movimiento de las hojas.

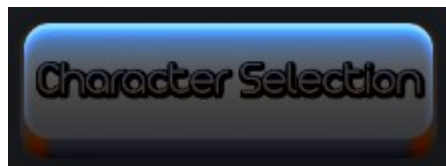
3.4.2 UI dentro del juego

Dentro del juego disponemos de tres “Canvas”, el primero al cual se podrá acceder pulsando la tecla “Esc”, sera para abrir un menú donde podremos acceder a varias opciones. Debemos tener en cuenta que al entrar en este menú, el juego se pondrá en pausa. El segundo se accederá pulsando la tecla “R”, nos mostrara el Ranking de jugadores.

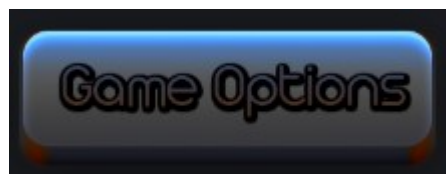
“Canvas” para acceder al Menú dentro del juego.



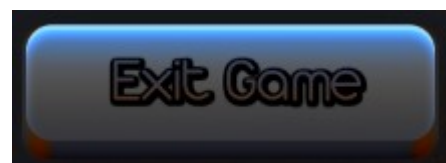
Botón para volver al juego.



Botón para volver a la selección de personaje.



Botón para acceder al menú de configuración.



Botón para salir del juego.

“Canvas” para acceder al Ranking de los mejores jugadores.

Este “Canvas” mostrara la base de datos en tiempo real el progreso de los mejores jugadores, mostrando su “Nick”, el nivel actual y la cantidad de experiencia acumulada.

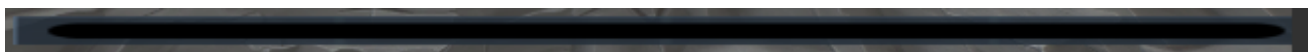


Cabe decir que este “Ranking” también sera accesible desde nuestro navegador web, ya que para ello utilizamos el servidor HTTP.

En la parte superior izquierda, situamos el tercer “Canvas” el cual nos indicara el nombre del jugador (“Nick” elegido en la escena anterior), el nivel actual, la vida del personaje representada con un “Slider”

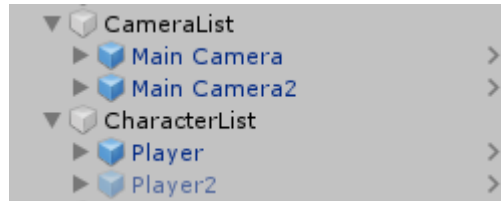


Por ultimo, situado en el mismo “Canvas”, en la parte inferior de la pantalla tenemos otro “Slider”, este representara la cantidad de experiencia actual indicándonos cuanto nos falta para subir de nivel.



3.5 Personajes y Cámaras

En la escena del juego al igual que en la escena de selección de personaje, tenemos los mismos objetos, con la diferencia de que en este punto el personaje ya podrá interactuar con el entorno.

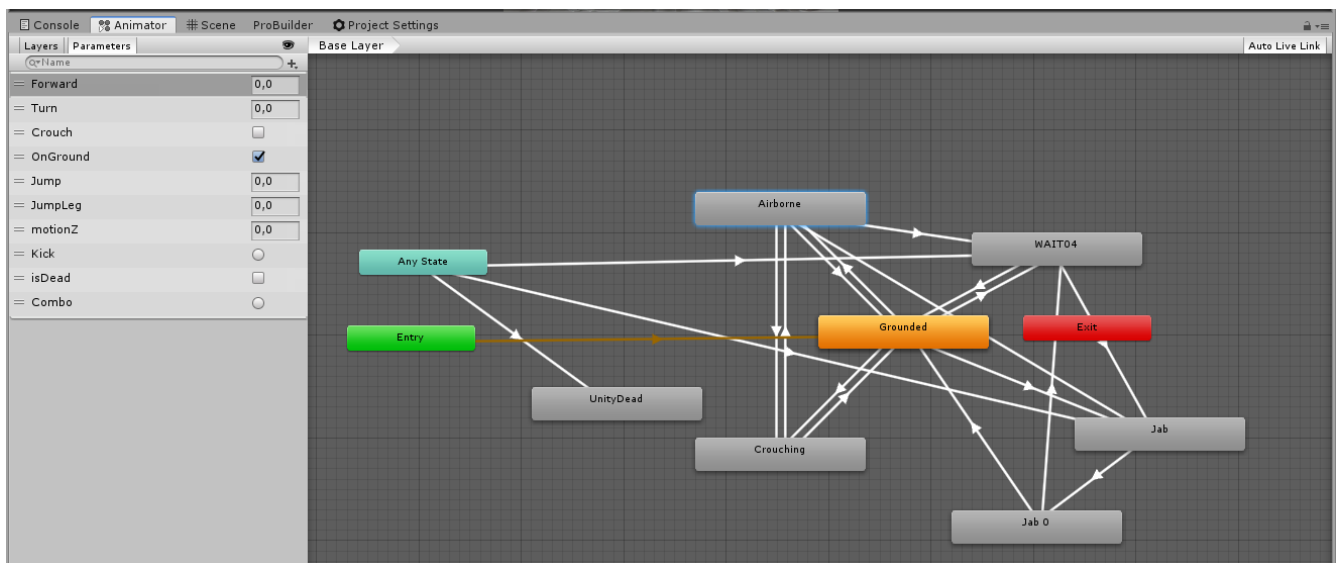


Para realizar la transición de un personaje a otro (incluyendo su respectiva cámara), se ha agregado un “script” a cada uno de los objetos que contienen la lista de personajes y la de las cámaras, su función sera recorrer el indice de las listas para saber en todo momento en que objeto se encuentra la selección del personaje que hemos realizado en la anterior escena.

3.5.1 Animator Controller

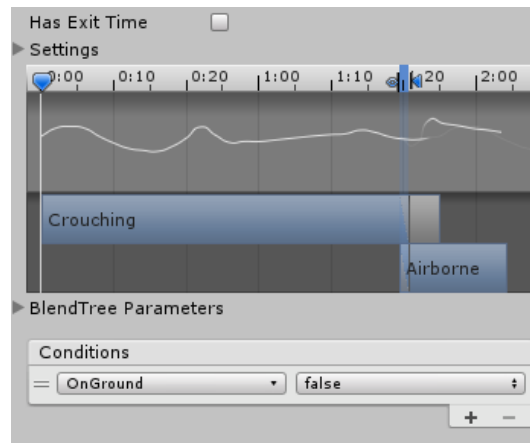
En cuanto al movimiento del personaje, Unity nos facilita el trabajo con su herramienta, el “Animator Controller”, el cual nos muestra un árbol con todas las transiciones (movimientos) que puede realizar este.

En la creación de los dos personajes he utilizado la base de un “Prefab”, Ezan que se encuentra en “Standard Assets”, se ha modificado su “Animator Controller”, y el script para el movimiento.

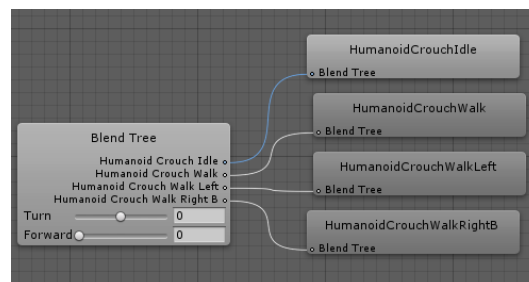
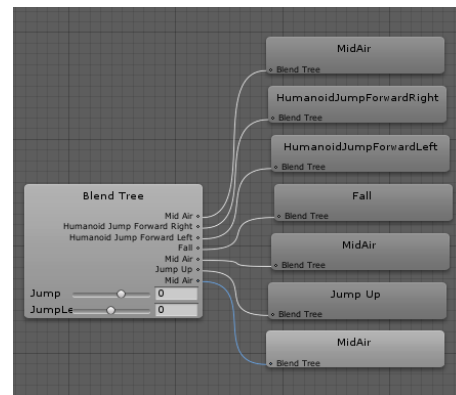
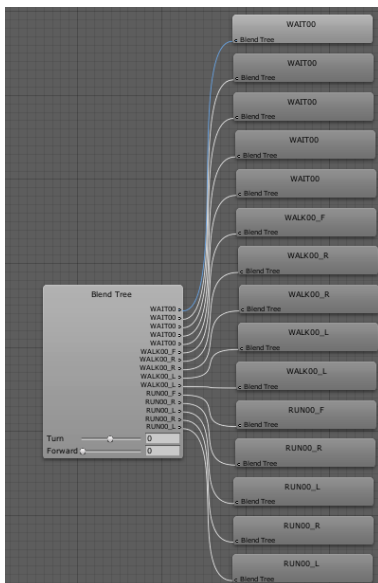


Cada flecha nos indica esa transición entre una animación a otra, en la parte izquierda de la imagen anterior se muestran los parámetros, estos nos dirán si el paso de esa transición cumple los parámetros requeridos para realizarse o no.

Aquí podemos ver una transición entre una animación y otra, y la condición requerida para ello.



En algunos casos y según el tipo de movimiento que implementemos, existe la posibilidad de crear un “Blend Tree”(árbol de mezclas) para aquellos movimientos que al menos cumplen un mismo parámetro, como es el caso de “Airborne”, “Grounded” y “Crouching”.



El Animator Controller, se podría decir que es una de las partes mas complicadas y que requieren mucho trabajo.

Una vez realizado esto, necesitamos que el “Animator Controller” realice las transiciones para lograr el efecto de movimiento del personaje, para ello debemos capturar los controles que el usuario debe pulsar para que este se mueva.

```
// Fixed update is called in sync with physics
private void FixedUpdate()
{
    if (health.value <= 0) return; //No inputs if dead.
    // read inputs
    float h = CrossPlatformInputManager.GetAxis("Horizontal");
    float v = CrossPlatformInputManager.GetAxis("Vertical");
    bool crouch = Input.GetKey(KeyCode.C);
}
```

Aqui podemos ver como el “script” captura el desplazamiento, “CrossPlatformInputManager” nos permite realizar estos movimientos independiente mente de la plataforma que estemos usando, ya sea un teclado, un mando u otros dispositivos.

En cuanto a los movimientos de ataque del personaje, la captura de los controles se realiza del mismo modo, pero con algunas limitaciones, se establece un tiempo de re-uso de las habilidades y se inutiliza los ataques cuando el personaje esta agazapado.

```

if (healthbar.value <= 0) return;

// update the animator parameters
m_Animator.SetFloat("Forward", m_ForwardAmount, 0.1f, Time.deltaTime);
m_Animator.SetFloat("Turn", m_TurnAmount, 0.1f, Time.deltaTime);
m_Animator.SetBool("Crouch", m_Crouching);
m_Animator.SetBool("OnGround", m_IsGrounded);
m_Animator.SetBool("Kick", attack_kick);
m_Animator.SetBool("Combo", attack_combo);

//Boton para atacar, con un delay de n segundos.
if (CrossPlatformInputManager.GetButtonDown("Left Click") && canAttack && !m_Crouching)
{
    canAttack = false;
    m_Animator.SetTrigger("Combo");
    particle.Emit(10);
    PlayAttackSound();
    Invoke("ResetAttack", resetAttackTime);
}

if (CrossPlatformInputManager.GetButtonDown("Right Click") && canAttack2 && !m_Crouching)
{
    canAttack2 = false;
    m_Animator.SetTrigger("Kick");
    particle.Emit(150);
    PlayAttackSound2();
    Invoke("ResetAttack2", resetAttackTime2);
}

if (!m_IsGrounded)
{
    m_Animator.SetFloat("Jump", m_Rigidbody.velocity.y);
}

```

La función Invoke(), sera la encargada de controlar el re-uso de estas habilidades, en función del tiempo que se le indique.

```

[SerializeField] float resetAttackTime = 1.3f;
[SerializeField] float resetAttackTime2 = 6.0f;

```

```

private void ResetAttack() //Ataque Primario
{
    canAttack = true;
}
private void ResetAttack2() //Ataque secundario
{
    canAttack2 = true;
}

```

3.5.2 Sistema de Partículas

Cada personaje cuenta con dos sistemas de partículas, uno encargado de los ataques.

El ataque primario “lanzara” unos pocos proyectiles creados a partir del sistema de partículas de Unity.

El segundo que tiene un daño mayor “lanzara” mas cantidad de proyectiles, pero tiene mas re-uso.

Ataque Primario



Ataque Secundario



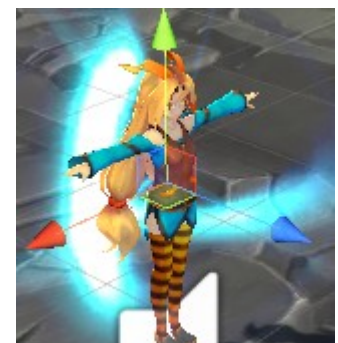
El segundo sistema de partículas lo utilizo para hacer una breve animación cuando el personaje sube de nivel. Para controlar el nivel del personaje y la experiencia obtenida, he creado un script (LevelingSystem) encargado de actualizar la experiencia requerida según el nivel actual, incrementado el valor máximo requerido cuanto mayor sea el nivel del personaje. El script también mandara esta información al “Canvas” para mostrar esta información al jugador.

```
void UpdateXP(int experienceToAdd)
{
    XP += experienceToAdd;
    int currentlvl = (int)(0.1f * Mathf.Sqrt(XP));

    if(currentlvl != currentLevel)
    {
        //Player lvl's Up
        currentLevel = currentlvl;
        textlvlChan.text = currentLevel.ToString();
        textlvlAkaza.text = currentLevel.ToString();
        xpbartozero = XP;
        lvlChan.Emit(1);
        lvlAkaza.Emit(1);
    }

    ExpForlvlUp = 100 * (currentLevel + 1) * (currentLevel + 1);
    int differenceexp = ExpForlvlUp - XP;

    int totaldifference = ExpForlvlUp - (100 * currentLevel * currentLevel);
}
```



3.5.3 Cámaras

Como he explicado antes cada personaje tendrá su respectiva cámara, el movimiento de estas sera orbital, centrándose en el personaje. También disponen de un enfoque (zoom) con el cual podremos alejar o acercar la cámara.

```
//Camara sobre el Personaje, con libertad de rotacion.  
currentX -= Input.GetAxis("Mouse Y");  
currentY += Input.GetAxis("Mouse X");  
  
currentX = Mathf.Clamp(currentX, Y_ANGLE_MIN, Y_ANGLE_MAX);  
  
Vector3 dir = new Vector3(0, 0, -distance);  
Quaternion rotation = Quaternion.Euler(currentX, currentY, 0);  
  
camTransform.position = (lookAt.position) + rotation * dir;  
  
camTransform.LookAt(lookAt.position);
```

3.6 Inteligencia Artificial

3.6.1 Enemigos

Al igual que ocurre con los personajes, los enemigos también disponen de un “Animator Controller” para gestionar sus movimientos, la diferencia es que estos movimientos son total mente controlados por un “script”, de esta manera conseguimos dotar a estos de cierto nivel de inteligencia.

El “script” controlara el rango de visión del enemigo, cuando el personaje se acerca a una distancia concreta este reaccionara. Por el contrario si el personaje se aleja demasiado, el enemigo volverá a su zona inicial.

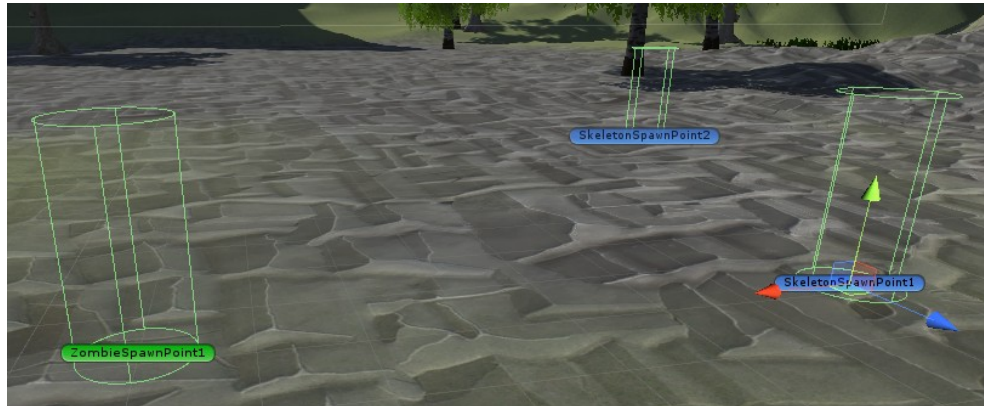
```
void Update()
{
    if (EnemyHealthBar.value <= 0) return;
    if (EnemyHealthBar.value < 99) { EnemyVision = 9999;}

    if(Vector3.Distance(player.position, this.transform.position) < EnemyVision)
    {
        if (UnityChanPlayer.value == 0 || AkazaPlayer.value == 0)
        {
            //Enemigo deja de atacar ya que el player ha muerto
            anim.SetBool("isIdle", true);
            anim.SetBool("isWalking", false);
            anim.SetBool("isAttacking", false);
            Weapon.SetActive(false);
            Weapon2.SetActive(false);
        }
    }
}
```

Cualquiera de estas ordenes sera interrumpida en caso de que el jugador pierda todos los puntos de vida.

3.6.2 SpawnPoints

Estos Objetos son los encargados de mantener una zona a la que están delimitados con una cantidad concreta de enemigos, recorren la zona de forma aleatoria y si el jugador elimina a un enemigo estos reponen hasta llegar a la cantidad máxima.



```
6 public class EnemyManager : MonoBehaviour
7 {
8     public Slider UnityHealth;
9     public Slider AkazaHealth;
10    public GameObject enemy;
11    public float spawnTime = 10f;
12    public Transform[] spawnPoints;
13    private int maxEnemies;
14    public int maxEnemiesAllowed;
15    public static int currentNumberEnemies;
16
17    // Start is called before the first frame update
18    void Start()
19    {
20        InvokeRepeating("Spawn", spawnTime, spawnTime);
21    }
22
23    void Spawn()
24    {
25        if (UnityHealth.value <= 0 || AkazaHealth.value <= 0) return;
26
27        if(maxEnemies >= maxEnemiesAllowed && currentNumberEnemies >= maxEnemiesAllowed)
28        {
29        }
30    }
31    else
32    {
33        int spawnPointIndex = Random.Range(0, spawnPoints.Length);
34        Instantiate(enemy, spawnPoints[spawnPointIndex].position, spawnPoints[spawnPointIndex].rotation);
35        maxEnemies++;
36        currentNumberEnemies++;
37    }
38    }
39
40 }
```

3.7 Unity Genéricos

Documentación genérica sobre elementos Unity.

<https://docs.unity3d.com/es/current/Manual>

3.7.1 Rigidbody

Los Rigidbodies le permite a sus GameObjects actuar bajo el control de la física. El Rigidbody puede recibir fuerza y torque para hacer que sus objetos se muevan en una manera realista. Cualquier GameObject debe contener un Rigidbody para ser influenciado por gravedad, actué debajo fuerzas agregadas vía scripting, o interactuar con otros objetos a través del motor de física NVIDIA Physx.

3.7.2 Colliders

Los componentes Collider definen la forma de un objeto para los propósitos de colisiones físicas. Un collider, el cual es invisible, necesita no estar con la misma forma exacta que el mesh del objeto y de hecho, una aproximación a menudo es más eficiente e indistinguible en el juego.

3.7.3 NavMesh

Los componentes NavMeshAgent le ayudarán a usted crear personajes que se eviten a ellos mismos mientras se mueven hacia su objetivo. Los Agentes razonan acerca del mundo del juego utilizando el NavMesh y saben cómo evitarse entre ellos al igual que otros obstáculos en movimiento. Pathfinding (Encuentra caminos) y el razonamiento espacial son manejados utilizando la API de scripting del NavMesh Agent.

4. HERRAMIENTAS

Unity

Unity es un motor de videojuego multiplataforma creado por Unity Technologies. Unity está disponible como plataforma de desarrollo para Microsoft Windows, OS X, Linux.

Microsoft Visual Studio

Microsoft Visual Studio es un entorno de desarrollo integrado para Windows, Linux y macOS. Es compatible con múltiples lenguajes de programación, tales como C++, C#, Visual Basic .

Xampp

XAMPP es un paquete de software libre, que consiste principalmente en el sistema de gestión de bases de datos MySQL, el servidor web Apache y los intérpretes para lenguajes de script PHP y Perl.

Blender

Blender es un programa informático multiplataforma, dedicado especialmente al modelado, iluminación, renderizado, animación y creación de gráficos tridimensionales.

SDK Tools – Android Studio

Android SDK Platform-Tools es un componente para el Android SDK. Incluye herramientas que interactúan con la plataforma Android, como adb, fastboot y systrace. Estas herramientas son necesarias para el desarrollo de aplicaciones para Android.

5 ESTRUCTURA DE SCRIPTS

Game Scene	
Objeto Unity	Script Asociado
CharacterList	CharacterSelection
CameraList	CameraList
MusicPlayer	MusicPlayer
Player	ThirdPersonCharacter DetectHitOnPlayer ThirdPersonController
Enemy	IA_Script DetectHitOnMe
EnemySpawnPoints	EnemyManager SpawnPointRandomMove
ParticleSystem	ParticleCollision
CanvasInGameMenu	ResumeMenu LevelingSystem
DirectionalLight	DayNightSystem

Start Menu Scene	
Objeto Unity	Script Asociado
Canvas	SettingsMenu
StartMenu	MainMenu
MusicPlayer	MusicPlayer

Character Selection Scene	
Objeto Unity	Script Asociado
CharacterList	CharacterSelection
CameraList	CameraList
MusicPlayer	MusicPlayer

6 MEJORAS Y FUTURAS ACTUALIZACIONES

Cualquier proyecto empieza con una idea simple, pero bien es sabido que tal y como avanzamos en el desarrollo de este, encontramos formas de mejorarlo o segmentos que pueden ser sustituidos por otros con mejores funciones, teniendo en cuenta todo esto, mi limitación como único autor y el tiempo para la presentación del proyecto, muchas de las ideas las he tenido que posponer o dejar para un futuro.

6.1 Juego Online

Una de las mejoras seria la opción de juego Online en tiempo real, donde los jugadores podrían interactuar entre si, hablar (chat de voz o escrito), jugar juntos para superar retos mayores, e incluso hacer pvp (player versus player) entre ellos.

Para esta mejora podemos incluir en la base de datos las cuentas donde almacenar los usuarios y sus contraseñas, los personajes que este ha creado, comprobando que ningún usuario crea un personaje con el mismo nombre que otro, etc.

6.2 Objetos e Inventario

Implementación de un sistema de inventario y “drop” de objetos, donde el jugador podrá obtener “items” matando enemigos, recogiénolos del mismo entorno, etc. Esto también se podría mejorar pudiendo comerciar también con los otros jugadores.

Objetos para mejorar nuestro personaje, ya sean armas como armaduras.

6.3 Razas, Profesiones y Habilidades

Disponer de un mayor numero de personajes con diferentes habilidades supone infinidad de combinaciones para la creación de nuestro personaje con lo cual conseguimos que nuestro juego destaque.

6.4 Sistema de Atributos

Un sistema de atributos donde el usuario pueda configurar a su forma de juego las estadísticas físicas del peonaje, pudiendo incrementar estas con cada nivel obtenido, o mediante recompensas.

6.5 Quest

Distribuidos por el juego encontraríamos NPC que nos mandarían trabajos para obtener recompensas a cambio de ayudarles.

6.6 Comercialización

Para empezar con la publicación del juego elegiría GoG, ya que no nos piden dinero. También estaría disponible en Google Play, puesto que con un único pago de 10 € podemos publicar nuestras aplicaciones.

En un futuro, si la demanda aumenta y nuestro juego es bien recibido, pasaría a publicarlo en Steam, aquí sería algo mas complicado puesto que tiene que pasar una serie de “filtros” y un pago de 100 €.

8 CONCLUSIONES

Antes que nada, agradecer el apoyo y la ayuda prestada de Esperanza Micó Méndez como también la del resto de profesores, que durante estos dos años, han conseguido que realizar este curso haya sido muy gratificante y satisfactorio.

Como jugador desde hace muchos años siempre había tenido la curiosidad de algún día poder crear mis propios videojuegos, así que este puede ser el comienzo de algo nuevo.

