SONIDO EN VIDEOJUEGOS – MEMORIA DEL PROYECTO FINAL

1. Introducción

El proyecto final que he realizado se trata de un prototipo de una discoteca con diferentes localizaciones que podría servir como entorno en 3D para muchos videojuegos, sobre todo videojuegos de sigilo, exploración o narrativos. No obstante, dado que lo que he desarrollado en profundidad es el diseño de audio del escenario, realmente es algo que se puede aplicar a un escenario de una discoteca adaptado a casi cualquier videojuego que necesite.

La decisión se basó en que una discoteca o entornos festivos es algo muy común en videojuegos modernos, y al mismo tiempo un gran campo de pruebas para poner en práctica diferentes efectos sonoros y elementos en el entorno.

He realizado el prototipo utilizando Unity y el motor de audio de FMOD Studio.

2. Descripción del escenario y elementos sonoros

El escenario está dividido en cuatro zonas: las afueras de la discoteca (OUTSIDE), la pista de baile (DISCO), el baño (BATHROOM) y la piscina (POOL). Encontraremos diferentes tipos de emisores de sonido, algunos que no están ligados a las diferentes zonas y otros que son exclusivos a las mismas.

1. Elementos sonoros generales

Estés donde estés, siempre se escuchará la música que pincha el DJ desde la pista de baile. Dado a que siempre estará escuchándose, he decidido hacer que no sea un sonido espacial, sino 2D. Por eso mismo, para simular la distancia y el aislamiento según la zona (sobre todo en las afueras y el baño, donde predominan los bajos) apliqué efectos que detallaré más adelante.

A parte de este sonido, algo que siempre el jugador tendrá que escuchar en todo momento serán los sonidos generados por él mismo, en este caso, los pasos al caminar. No obstante, también se diferenciarán según la zona, dado que son diferentes tipos de suelo. Este sonido tampoco es espacial, ya que siempre estarás al lado de la fuente de sonido (de hecho, el mismo jugador es la fuente de sonido)

Por último, unos emisores que se pueden encontrar compartidos en varias zonas son los NPC y los animales. Un NPC son un objeto más complejo que gestionará diferentes estados (con la posibilidad de modificarlos desde código) en los que emitirá sonidos diferentes. Esta funcionalidad es escalable a una IA más compleja, aunque ahora mismo está limitada a estados simples estáticos dado que es un prototipo de sonido. Por su parte, los animales que pueden sonar en las zonas abiertas (las afueras y la piscina, que no está techada) son un objeto más simple (tan simple que se ejecuta sólo con el evento de FMOD) que elegirá un sonido aleatorio que simulará un animal. Al igual que los NPC, esta funcionalidad también es escalable: se pueden añadir parámetros para elegir qué animal queremos que pueda sonar o no, limitar las fuentes de sonidos de animales a zonas concretas, etc. Ambos emisores de sonido son espaciales ya que estos sí dependen de la cercanía a la que el jugador se encuentre de ellos.

2. Elementos sonoros exclusivos de cada zona

Antes de señalar los elementos exclusivos de cada zona o los efectos que se aplican a los sonidos generales, cabe destacar lo remarcado en el párrafo anterior: los NPC se encontrarán en todas las

zonas, los animales en las afueras y en la piscina, y la música de la discoteca se escuchará en todas las zonas, a parte de los sonidos de las pisadas.

a. Afueras

En las afueras nos encontraremos con un sonido ambiente, aplicado de manera similar a la música de la discoteca. El emisor de sonido "NightAmbience" funciona como "banda sonora" de lo que sería estar por la noche en la calle. No obstante, muchos de los sonidos que se escuchan (coches pasando, pequeños sonidos de grillos, etc) sí dependen de la espacialidad en la realidad, pero aplicarlo como un sonido 3D haría que apenas pudiese escucharse debido al gran tamaño de la zona, por lo que decidí usar un sonido 2D el cual modificaría mediante parámetros para simular espacialidad.

A parte del sonido ambiente, también tenemos unos coches que emiten el sonido 3D de un motor encendido. Es así de simple, pero también podría escalarse a un objeto que pudiese gestionar si el coche está en marcha o no, aunque lo vi innecesario para este prototipo.

La música de la discoteca en las afueras se escuchará muy aislada, y apreciaremos casi exclusivamente los bajos retumbando.

b. Pista de baile

Aquí lo principal es la música de la discoteca, ya que se escuchará con todo su esplendor y a mayor volumen y calidad. Esta zona podría decirse que es la zona central de la que sale el sonido, pero de nuevo, al no ser un sonido 3D por el gran tamaño que tiene (y me parecía que daba peor resultado aplicar un radio que solía ser insuficiente), la espacialidad está simulada mediante efectos. Para dicha simulación, disponemos de unos "triggers" en los puntos que conectan el resto de las salas con la discoteca, los cuales el gestor de la zona utilizará para comparar la distancia entre estos y el jugador y así aplicar el filtro correspondiente a la pista.

Dado que la música suena mucho más fuerte en esta zona (y más fuerte de lo que suele sonar una música de fondo en cualquier videojuego, pero es comprensible debido al contexto de la zona), pocos sonidos podrían destacar aquí, aunque también he añadido un sonido aleatorio que suena con poca probabilidad en el cual escucharemos como se rompe un vaso de cristal contra el suelo.

c. Cuarto de baño

El cuarto de baño es una zona diferente porque he añadido unos interactuables que generan sonido. Concretamente, los elementos típicos de un baño: un grifo que puedes encender y apagar y un váter del que puedes tirar de la cadena. Además, hay colocados unos NPC.

Según te acercas al fondo del baño, más se aislará la música. En medio de la sala se encuentra el trigger que medirá cómo de cerca estamos de la puerta, y cuanto más cerca más se escuchará la música hasta que salgas de nuevo del baño. El resultado del sonido aislado es similar al de la calle pero pudiendo escuchar mucho más las frecuencias altas ya que las paredes deberían ser mucho menos gruesas.

d. Piscina

La zona de la piscina tiene unos NPC y animales en los alrededores de la propia piscina, pero lo importante aquí es cuando el jugador se sumerge en el agua. Si el jugador baja la rampa y se mete en la piscina, escuchará primero el sonido del agua al hacer la inmersión y se escuchará un sonido ambiente de agua. Además, los pasos al caminar pasan a ser un sonido de una persona nadando, y para más inmersión, todos los sonidos que suenen pasarán a tener un filtro cuyo resultado suena como si escuchásemos con la cabeza sumergida en el agua.

Además de esto, cuando estás fuera de la piscina, de la misma manera que pasaba con los triggers del baño o de la salida a las afueras, habrá una reducción de la ganancia en la música de la discoteca, ya que el jugador se encuentra más lejos de la fuente de sonido y no está techado, por lo que el sonido no debería escucharse aislado.

3. Eventos de FMOD

En este apartado comentaré brevemente los eventos de FMOD más importantes. Los eventos no destacados simplemente son eventos que no tienen mucha complicación, ya sea porque son acciones simples (como Tap, Toilet o Splash), o son instrumentos o multi-instrumentos en loop cuyo volumen será 0db o -∞ según parámetros que indiquen en qué zona está el jugador (Underwater, Broken Glass, Animals o CarEngine).

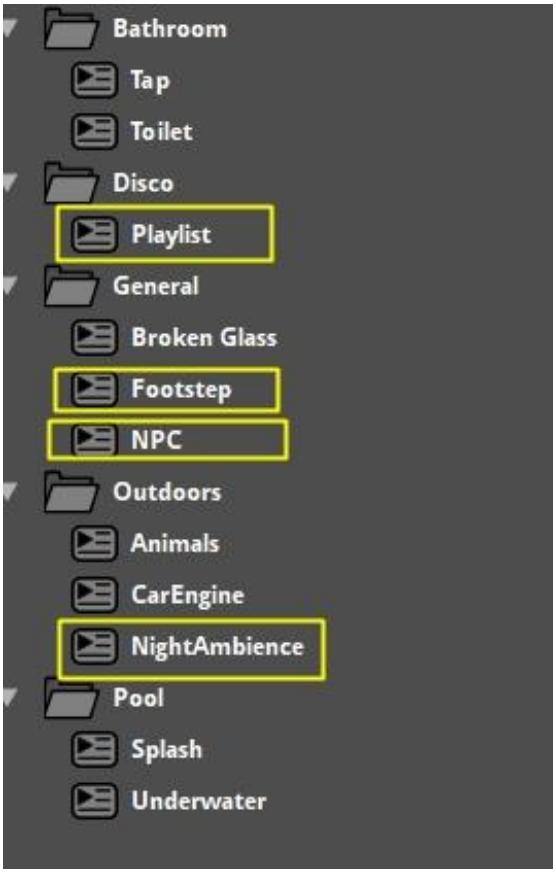
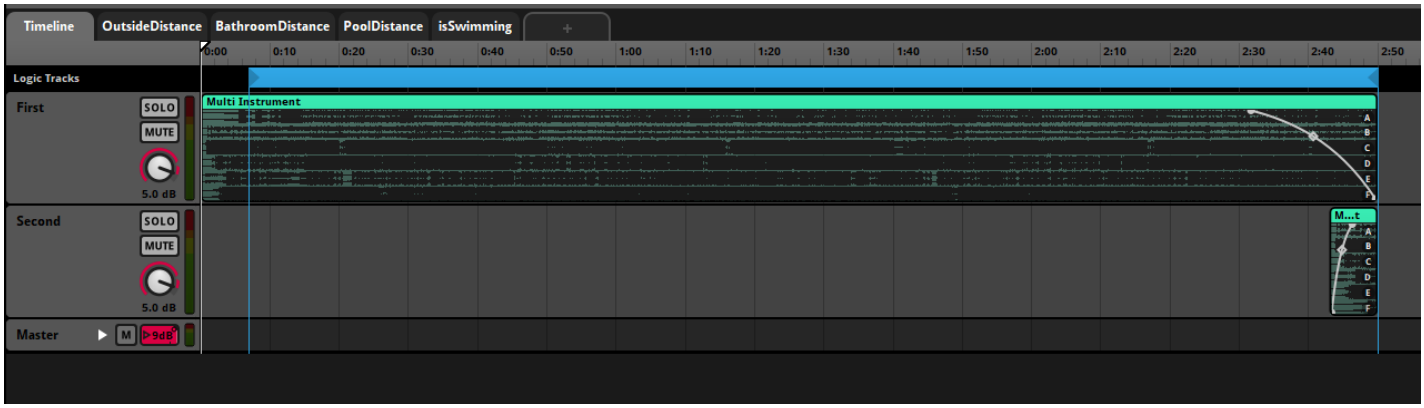


Ilustración 1: Lista de eventos

1. Playlist



La playlist principal es un conjunto de canciones que se reproducen de manera secuencial. Para poder simular la mezcla que haría un DJ para acabar una canción y comenzar otra, he recortado los primeros segundos del instrumento y los he colocado al final en una segunda pista, con la única diferencia que la primera pista de este recorte será la segunda pista del instrumento original. Esto quiere decir que, cuando los dos instrumentos coinciden, el segundo estará reproduciendo los primeros segundos de la canción que se reproducirá secuencialmente en el primer instrumento. Como el loop está colocado para que no repita los primeros segundos de la canción al volver a empezar, y esos primeros segundos se habrán reproducido al final del loop anterior gracias al segundo instrumento, encajará perfectamente.

Como expliqué antes, es un sonido que debe acompañar al jugador en toda la zona, por lo que me pareció más conveniente usar un sonido 2D y utilizar parámetros de distancias para controlar efectos en el sonido. Concretamente, dependerá de los siguientes parámetros continuos y locales:

- OutsideDistance [0,20]: 0 = en la puerta de la discoteca, se escucha perfectamente. 20 = lejos de la puerta de la discoteca, se escucha un sonido muy aislado.
- BathroomDistance [-20,20]: -20 = en la puerta del baño, se escucha perfectamente. 20 = en el fondo del baño, se escucha un sonido parcialmente aislado.
- PoolDistance [0,20]: igual que OutsideDistance. Reduce un poco la ganancia y decibelios.

También depende del parámetro global isSwimming que aísla mucho más el sonido. Para conseguir estos efectos, he utilizado principalmente el efecto Multiband EQ. En el siguiente gráfico, A representa OutsideDistance, B representa BathroomDistance, C representa PoolDistance y D representa isSwimming.

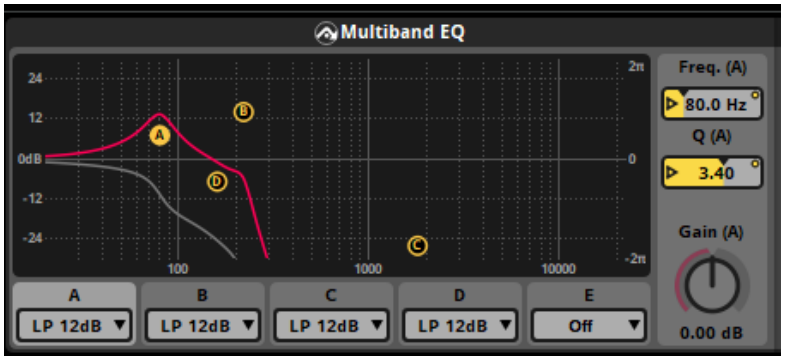


Ilustración 2: Efectos aplicados

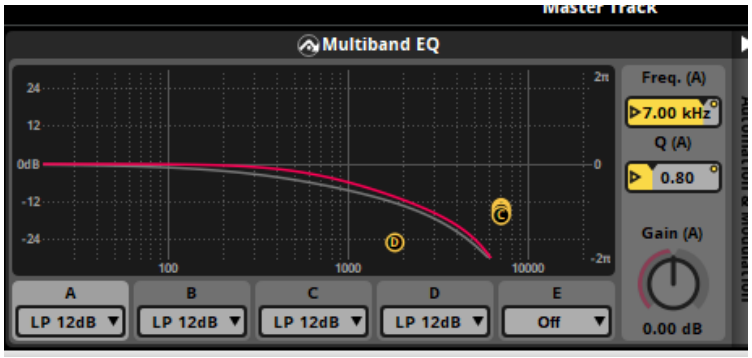
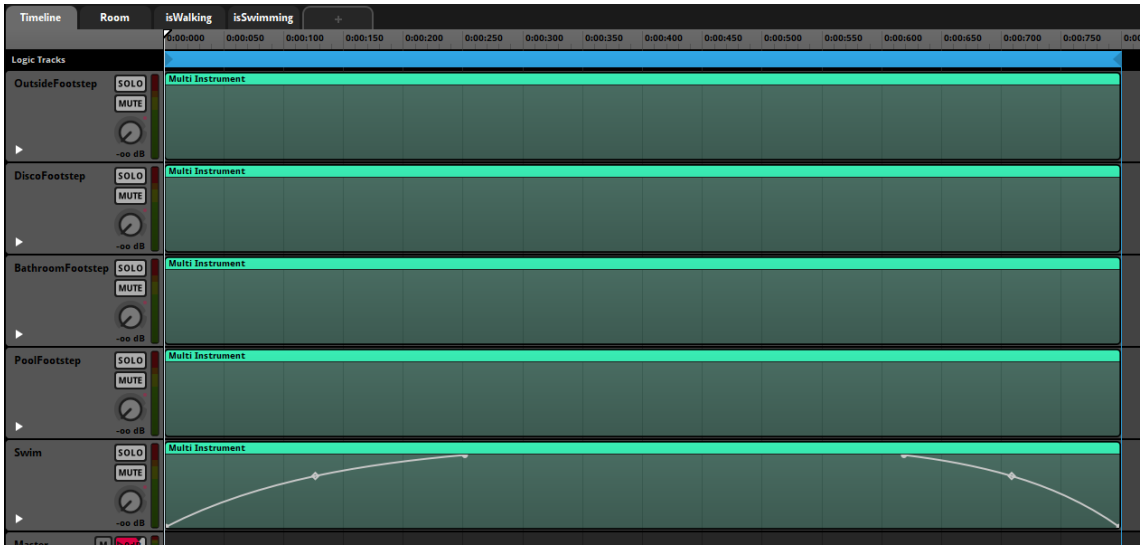


Ilustración 3:Sonido tal cual se escucha en la pista de baile

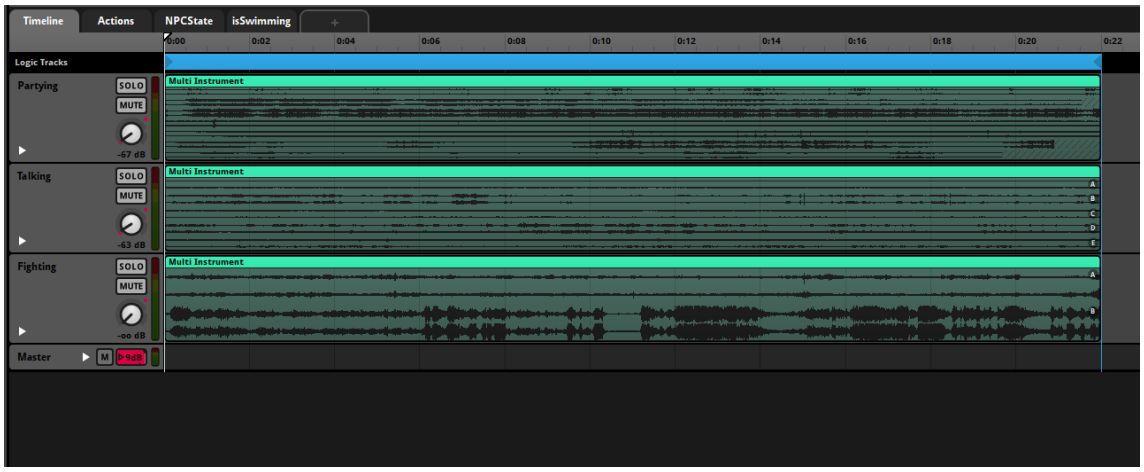
2. Footstep



Footstep es un evento que depende únicamente de 3 parámetros globales, que son parámetros concretos que van de 0 a 1 funcionando como booleanos. Concretamente, Room es un parámetro "labeled" que nos indica en qué habitación estamos, para elegir qué sonidos de pasos aplicar. isWalking nos indica si el jugador está caminando, por lo que puede reproducir los pasos. isSwimming nos indica, al igual que antes, si el jugador está nadando en la piscina, para reproducir el tipo de paso específico que es nadar.

Cada pista es un multi-instrumento de entre 10 y 20 grabaciones de pasos cuyo pitch está modificado aleatoriamente, por lo que cada paso sonará distinto siempre al oído del jugador.

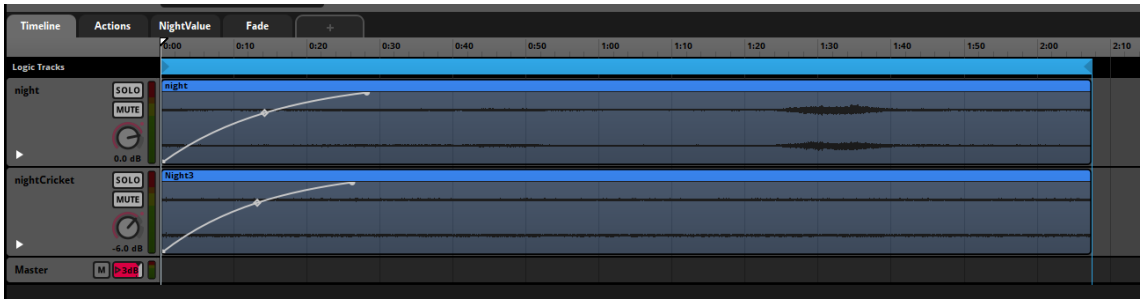
3. NPC



NPC funciona de manera muy similar a Footstep. Depende de unos parámetros globales, que, como los NPC pueden estar en cualquier localización, no dependen de la zona actual. En lugar de eso, dependen del estado en el que se encuentra el NPC. Los estados en los que puede estar NPC en mi prototipo son TALK, PARTY, FIGHTING y NONE. Como parámetro labeled, lo que indicará es que pista escoger. Cada pista, al igual que sucedía con Footstep, tiene un multi-instrumento con diferentes grabaciones para cada estado de la conversación: más tranquilas en Talk, más fiesteras en Party y agresivas en Fighting.

Al igual que otros sonidos, le apliqué el efecto Multiband EQ de la misma manera que con la playlist principal por si hay NPC cerca de la piscina en la que se puede meter el jugador.

4. NightAmbience



NightAmbience tiene un funcionamiento parecido a DiscoPlaylist, dado que ambos son eventos 2D que deben sonar todo el rato en un espacio muy grande, pero utiliza parámetros para controlar la espacialidad. Concretamente, dispone de dos pistas de audio que suenan simultáneamente, con diferentes sonidos ambientes que suenan por la noche. El parámetro NightValue lo que hace es ir de 0 a 1 (de manera continua), y forma una curva que cambia el volumen y el paneo de ambas pistas, para así controlando el valor del parámetro mediante código, hacer que el sonido "se mueva" de izquierda a derecha, como si los sonidos de la noche estuviesen en constante movimiento.

Además, para que no fuese brusco el cambio al apagar el sonido ambiente (ya que dentro de la discoteca no suena), le añadí el parámetro Fade que tiene activada la propiedad Seek Speed para que al pasar de 1 a 0 como un booleano al activar y desactivar la reproducción del evento, no corte de golpe sino que pase progresivamente durante x segundos hasta llegar a desactivarse por completo.

5. Integración en Unity y scripts más importantes

No detallaré los scripts del movimiento del jugador o la cámara, ya que al ser un prototipo centrado en el audio son bastante simples y no proceden.

He utilizado la integración de FMOD en Unity y he creado una estructura de código para gestionar la reproducción de sonido de manera más compleja, mediante un GameManager, y no tener que depender de las limitadas opciones que ofrecen desde el editor de Unity los componentes de FMOD (ya que no permite personalizar una acción para encender o apagar un evento, modificar parámetros, etc). La estructura es la siguiente:

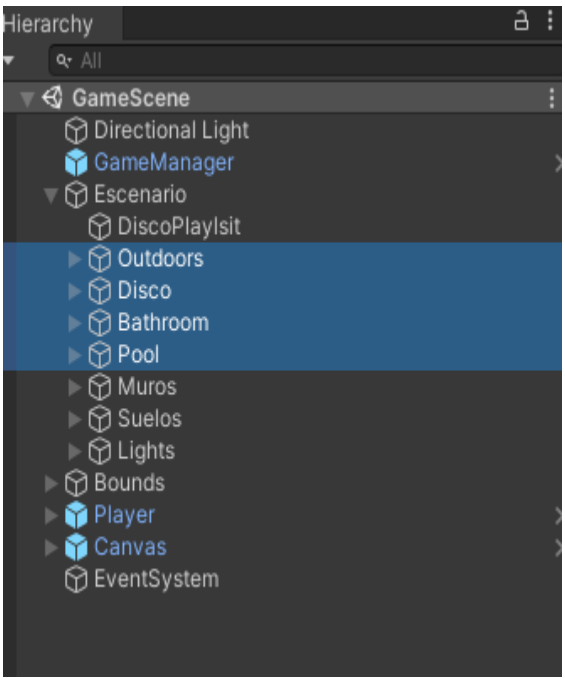
El escenario está dividido por zonas. Para controlar cada zona, creé un componente RoomBehaviour, del cual heredan los componentes específicos de cada zona (OutsideBehaviour, DiscoBehaviour, etc), que añaden a la funcionalidad común (reproducir los sonidos específicos de la sala) unas funcionalidades específicas para la sala.

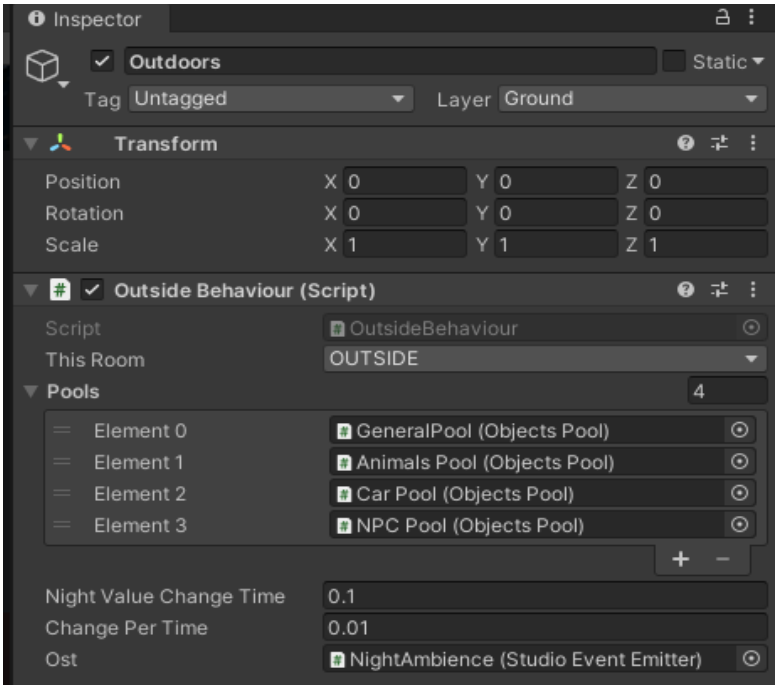
Cada una de estas salas son gameobjects vacíos que solo tienen este componente, y de los cuales cuelgan diferentes objetos vacíos que funcionarán como "pools" de objetos emisores de sonidos. Las pools (componente ObjectPool) controlan los GameObjects que tengan el componente SoundEmitterObject, el cual se encarga de guardar el emisor de sonido de FMOD y controlar su reproducción según el jugador permita o no dicha reproducción. Los objetos más complejos como los NPC tendrán componentes propios que heredarán de SoundEmitterObject y que añadirán sus propias funcionalidades.

SONIDO EN VIDEOJUEGOS – MEMORIA DEL PROYECTO FINAL

Para crear una sala con sonidos propios, el funcionamiento es el siguiente:

1. Añadir mediante el editor las zonas con las pools de objetos que tengan dentro





2. En el Awake, cada pool se encarga mediante código de detectar a sus hijos que sean de tipo SoundEmitterObject y guardarlos en un array

```
4 referencias
public class ObjectsPool : MonoBehaviour
{
    private SoundEmitterObject[] objs;
    private int size;

    0 referencias
    private void Awake()
    {
        size = this.transform.childCount;
        objs = new SoundEmitterObject[size];

        for(int i = 0; i<size; i++)
        {
            var child = transform.GetChild(i).GetComponent<SoundEmitterObject>();
            if (child != null)
                objs[i] = child;
        }
    }
```

3. Cuando el GameManager indica que se inicie una sala (método que se llama cuando el jugador colisiona con ciertos triggers colocados en las puertas), éste llama a todos sus pools para que se inicien también

```
2 referencias
public void changeRoom(ROOM newRoom)
{

    if (newRoom == _actualRoom)
    {
        Debug.LogWarning("Estás indicando que entra y sale de la misma habitación");
        return;
    }
    if(_actualRoom != ROOM._MAX) //En caso de que no esté aún en ninguna sala, no hay que desactivar ninguna
    {
        //le indica a los sonidos de esa sala que paren de emitir todos los eventos
        rooms[(int)_actualRoom].disable();
    }
    //le indica a los sonidos de esa sala que empiecen a emnitir los eventos
    _actualRoom = newRoom;
    roomTrigger.TriggerParameters((int)_actualRoom);
    rooms[(int)_actualRoom].enable();


    Debug.Log(_actualRoom);
    zoneText.text = "ZONE: " + _actualRoom.ToString();

}
```

4. Cada pool llama a todos sus objetos SoundEmitterObject, que comienzan el sonido (en caso de estar activado en el editor), y sus parámetros (en este orden, ya que si inicializo los parámetros antes de comenzar el sonido, no se aplicarán)

```
1 referencia
public void Initialize()
{
    foreach(SoundEmitterObject obj in objs)
    {
        obj.playSound();        //sound
        obj.setInitialState(); //params
    }
}

1 referencia
public void Stop()
{
    foreach (SoundEmitterObject obj in objs)
    {
        obj.stopSound();
    }
}
```

```
public virtual void setInitialState()
{
    ;
}

1 referencia
public void playSound()
{
    if(_soundEmitter.emmiter.enabled ==true)
        _soundEmitter.emmiter.Play();


}

1 referencia
public void stopSound()
{
    if (_soundEmitter.emmiter.enabled == false)
        return;

    if (_soundEmitter.fadeOut)
    {
        StartCoroutine(fader(_soundEmitter.emmiter));
    }
    else
        _soundEmitter.emmiter.Stop();
}


1 referencia
IEnumerator fader(FMODUnity.StudioEventEmitter e)
{
    e.SetParameter("Fade", 0);
    yield return new WaitForSeconds(3);
    e.Stop();
    yield return null;


}
```

Otros scripts importantes:

```
1 referencia
public class ActivateAction : MonoBehaviour
{
    public FMODUnity.StudioEventEmitter actionSound;
    public KeyCode activationKey;
    public bool loop = false;
    private bool isPlaying = false;

    0 referencias
    private void OnTriggerStay(Collider other)
    {
        if (Input.GetKeyDown(activationKey))
            playObjectSound();
    }

    2 referencias
    public void playObjectSound()
    {
        if (loop && isPlaying)
        {
            Debug.Log("DESACTIVADO");
            actionSound.Stop();
            isPlaying = false;
        }
        else
        {
            Debug.Log("ACTIVADO");
            actionSound.Play();
            isPlaying = true;
        }
    }
```

Ilustración 4:Interactuables del baño

```
    9 referencias
    public virtual void enable()
    {
        roomEnabled = true;
        startSounds();
    }
    9 referencias
    public virtual void disable()
    {
        roomEnabled = false;
        stopSounds();
    }
    1 referencia
    public void startSounds()
    {

        foreach(ObjectsPool p in _pools)
        {
            p.Initialize();
        }

    }
    1 referencia
    public void stopSounds()
    {
        foreach (ObjectsPool p in _pools)
        {
            p.Stop();
        }
    }
    0 referencias
    private void OnTriggerEnter(Collider other)
    {
        if (thisRoom == GameManager._instance.checkRoom())
            return;
        Debug.Log("Entra a " + thisRoom);
        GameManager._instance.changeRoom(thisRoom);
    }
```

Ilustración 5:RoomBehaviour.cs

```
20  private void Update()
21  {
22
23      float outsideDistance = Vector3.Distance(player.position, outsideDoor.position);
24      float poolDistance = Vector3.Distance(player.position, poolDoor.position);
25      float bathroomDistance = Vector3.Distance(player.position, bathroomCenter.position);
26
27      switch (GameManager._instance.checkRoom())
28      {
29          case ROOM.OUTSIDE:
30              discoPlaylist.SetParameter("OutsideDistance", outsideDistance);
31              break;
32          case ROOM.BATHROOM:
33              int factor = checkBathroomPosition(); //comprueba si estás más cerca de la puerta del baño o más lejos (izquierda o derecha del centro del baño)
34              discoPlaylist.SetParameter("BathroomDistance", bathroomDistance*factor);
35              break;
36          case ROOM.POOL:
37              discoPlaylist.SetParameter("PoolDistance", poolDistance);
38              break;
39          case ROOM.DISCO:
40              float pimba = poolDistance;
41              if (checkPoolPosition())//comprueba si está delante del objeto (más cerca de la piscina que de la disco)
42                  pimba = poolDistance;
43              else
44                  pimba = 0;
45              discoPlaylist.SetParameter("PoolDistance", pimba);
46              Debug.Log(pimba);
47              break;
48          default:
49              break;
50      }
51
52
53  }
```

Ilustración 6: DiscoBehaviour.cs