Memoria Proyecto de Sonido

Javier Cano
Daniel Ruíz
José Miguel Villacañas

Índice

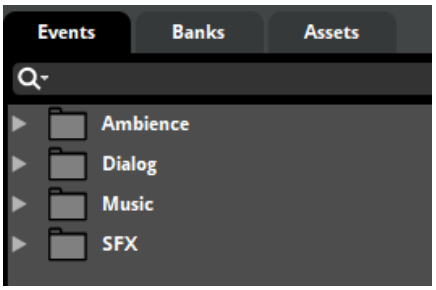
1. Descripción breve

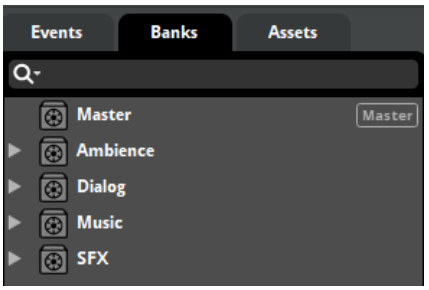
Hemos decidido sonorizar nuestro juego llamado **The Explorer**, un juego serio desarrollado por Javier Cano con el objetivo de divulgar el conocimiento de las pinturas rupestres de la Cañaica del Calar (Murcia, España) y su posible significado.

El juego está hecho usando el motor de Unity y como herramienta para sonorizar hemos usado **FMODStudio,** usando la guía oficial de FMOD para integrarlo al proyecto.

2. Distribución del proyecto de FMOD

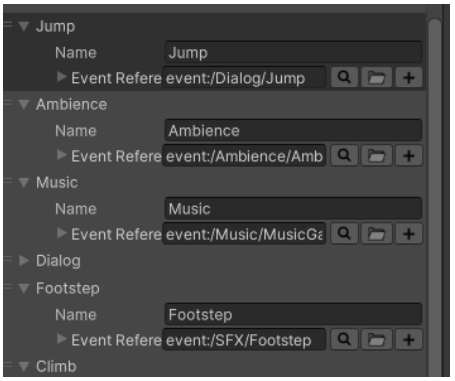
Hemos organizado todo el proyecto respecto a las categorías de sonidos que íbamos a implementar. Las pestañas de *Eventos* y *Banks* se disponen de la siguiente manera:





3. Carga de eventos

Para cargar un evento de FMOD a Unity hemos usado una clase llamada **FMODEvents** que contiene todas las *Event References* de los eventos de FMOD asociadas a un nombre identificativo.



Así, cuando queramos reproducir un sonido, accederemos a la clase FMODEvents con el nombre identificador del evento como parámetro de entrada y ella nos devolverá el evento correspondiente.

4. Sonorización del juego

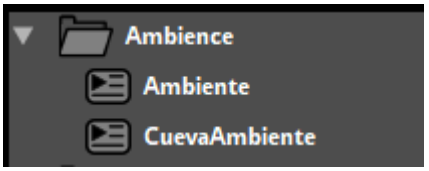
Además de tener una clase que contenga la información de todas las *Event References*, hemos creado una clase **AudioManager** que se encarga de gestionar toda la información respecto a los sonidos que se reproducen. Es decir, si se quiere instanciar un sonido, tendremos que acceder a la función que se encarga de instanciarlo desde el **AudioManager** y además, pasarle como parámetro de entrada la referencia del evento, a la que accederemos desde **FMODEvents**.

Para realizar un menú de volumen ajustable para el juego, todos los efectos de sonido, música y diálogos se deben añadir a grupos en el Mixer de Fmod. Así, desde el AudioManager, se pueden modificar los valores de salida de los buses para cambiar el volumen del juego.

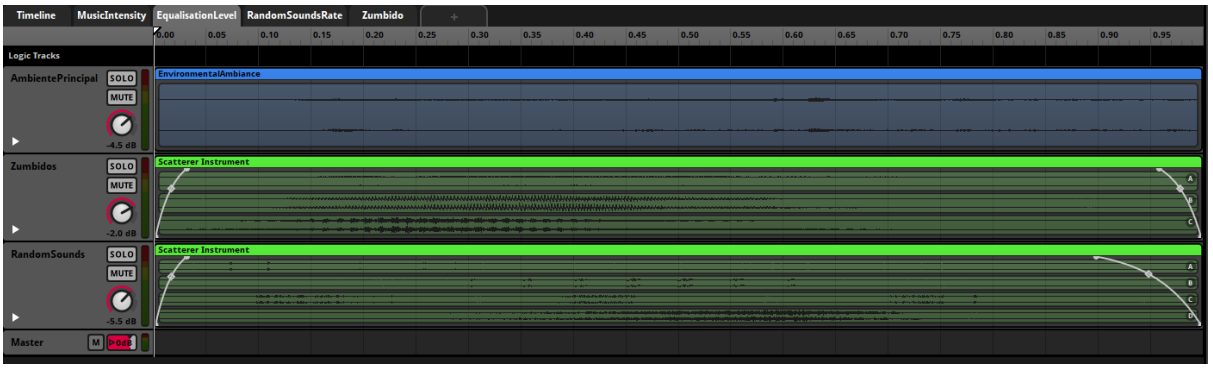
```
public void PlayOneShotSound(EventReference eventReference, Vector3 worldPosition)
{
    RuntimeManager.PlayOneShot(eventReference, worldPosition);
}

public EventInstance CreateInstance(EventReference eventReference)
{
    EventInstance eventInstance = RuntimeManager.CreateInstance(eventReference);
    eventInstances.Add(eventInstance);
    return eventInstance;
}
```

4.1 Ambiente

Comenzando con la sonorización del juego, en la carpeta *Ambience* contamos con los siguientes eventos: **el ambiente exterior** (Ambiente) y **el ambiente de la cueva** (CuevaAmbiente).



El evento **Ambiente** controla todos los sonidos ambiente que suceden en el juego por medio de parámetros relacionados con el ratio de aparición de dichos sonidos, la intensidad del mismo y la ecualización.



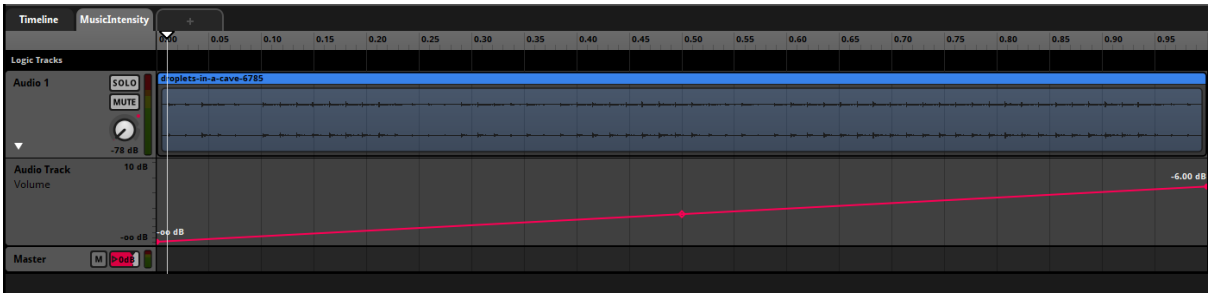
Los sonidos aleatorios como los zumbidos de las abejas u otros sonidos como animales, el sonido del césped cuando se pisa, o pájaros volando vienen controlados por dos parámetros de tipo discreto: **Random Sounds Rate** y **Zumbidos.**

Todos los sonidos añadidos al sonido ambiente por defecto estan hechos con un **Scatterer Instrument** (uno solo para abejas y otro para sonidos más generales) y con su ratio de aparición en función de donde nos encontremos (cueva, menú, escena principal) ajustable por los parámetros mencionados anteriormente.

La intensidad del ambiente y el nivel de ecualización están controlados por los parámetros **Music Intensity** y **Equalisation Level.** Estos también se ajustan en función de la zona y añaden un toque más de realismo cuando estás en el juego, en pausa o cuando estás en distintas zonas.

La ecualización se hace por medio del efecto **3Eq** nivelando en función del parámetro los tres tipos de frecuencia (low, mid y high) consiguiendo así un sonido más envolvente. Por ejemplo, cuando accedes al menú en comparación a cuando estás en la escena principal del juego.

En el evento **Cueva Ambiente** tenemos los sonidos de cueva, los cuales se activan también por un parámetro de tipo discreto (el parámetro ya creado **Music Intensity**).



Por otra parte, controlamos el sonido de las gotas de agua añadiendo otra modulación al sonido de forma aleatoria y una **envolvente ADSR** para la transición entre entrar y salir de la cueva, consiguiendo así el efecto de un crossfade.

Este evento se activa cuando entras a la cueva y sirve para dar una sensación más realista mientras estás jugando al juego.

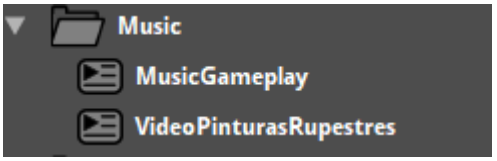
```
public void adjustEQINCave(bool isInCave) {
    inCave = isInCave;
    if (!isInCave) {
        ambience.setParameterByName("MusicIntensity", 0.4f);
        ambience.setParameterByName("EqualisationLevel", 0.0f);
        ambience.setParameterByName("Zumbido", 0.5f);
        ambience.setParameterByName("RandomSoundsRate", 0.4f);

        music.setParameterByName("EqualisationLevel", 0.0f);

        cave.setParameterByName("MusicIntensity", 0.0f);
    }
    else {
        ambience.setParameterByName("MusicIntensity", 0.1f);
        ambience.setParameterByName("EqualisationLevel", 0f);
        ambience.setParameterByName("Zumbido", 0.0f);
        ambience.setParameterByName("RandomSoundsRate", 0.0f);

        music.setParameterByName("EqualisationLevel", 0.4f);

        cave.setParameterByName("MusicIntensity", 0.8f);
    }
}
```

Para no hacer esta transición de forma abrupta, los parámetros cuentan con un **retardo** (seek speed).

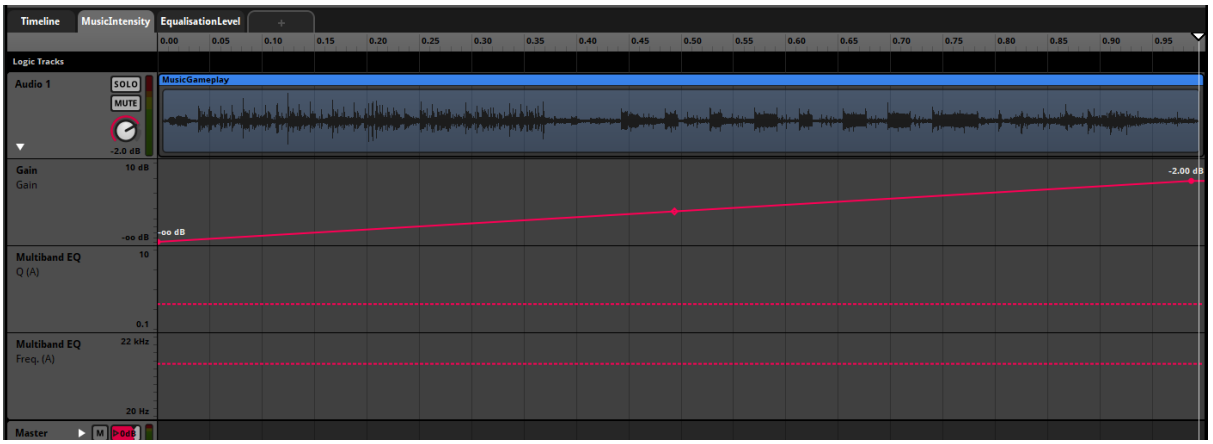
Al abrir el menú de pausa, se ignora este retraso porque queremos que sea un cambio instantáneo de forma intencionada.

4.2 Música

En la carpeta *Music* contamos con los siguientes eventos: **la música del juego** (MusicGameplay) y **el sonido del vídeo.** (VideoPinturasRupestres).



El evento **MusicGameplay** controla la música que suena a lo largo de todo el gameplay del juego y cuenta un parámetro que ajusta la intensidad y otro que ajusta la ecualización.



Es importante decir que todos los parámetros usados tanto en la música como en el ambiente son locales, es decir, cada instancia del evento tiene su propia intensidad o ecualización, debido a que en ciertas zonas queremos aplicar más intensidad al ambiente que a la música o viceversa.

Para reproducir la música del juego en Unity, basta con crear las instancias en el Start() del *MusicManager*, script encargado de reproducir la música de fondo/ambiente

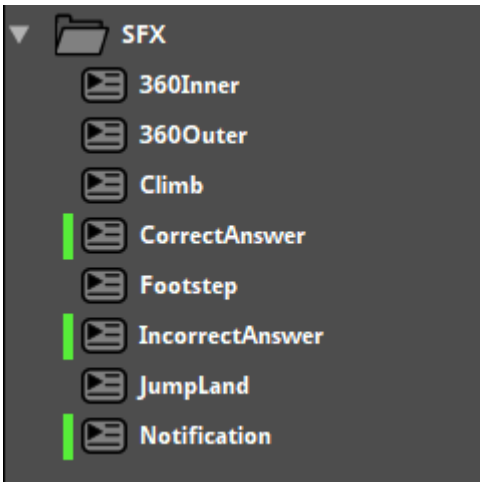
```
void Start() {
    music = GameManager.Instance.audioManager.CreateInstance(GameManager.Instance.fmodEvents.GetEvent("Music"));
    ambience = GameManager.Instance.audioManager.CreateInstance(GameManager.Instance.fmodEvents.GetEvent("Ambience"));
    cave = GameManager.Instance.audioManager.CreateInstance(GameManager.Instance.fmodEvents.GetEvent("CaveAmbience"));

    ambience.setParameterByName("MusicIntensity", 0.4f);
    ambience.setParameterByName("EqualisationLevel", 0.0f);
    ambience.setParameterByName("Zumbido", 0.2f);
    ambience.setParameterByName("RandomSoundsRate", 0.2f);

    music.setParameterByName("MusicIntensity", 0.30f);
    music.setParameterByName("EqualisationLevel", 0.0f);

    cave.setParameterByName("MusicIntensity", 0.0f);

    if (SceneManager.GetActiveScene().name == "Calar") {
        music.start();
        ambience.start();
        cave.start();
    }

    DontDestroyOnLoad(gameObject);

    SceneManager.sceneLoaded += SceneLoaded;
}
```

```
music.getPlaybackState(out PLAYBACK_STATE musicState);
if(musicState != PLAYBACK_STATE.PLAYING)
    music.start();
```

Debido a que no se puede reproducir audio de FMOD desde el VideoPlayer de Unity, el evento **VideoPinturasRupestres** cuenta con el audio extraído del vídeo, que se reproduce a la vez que el vídeo cuando el jugador se pasa el juego.

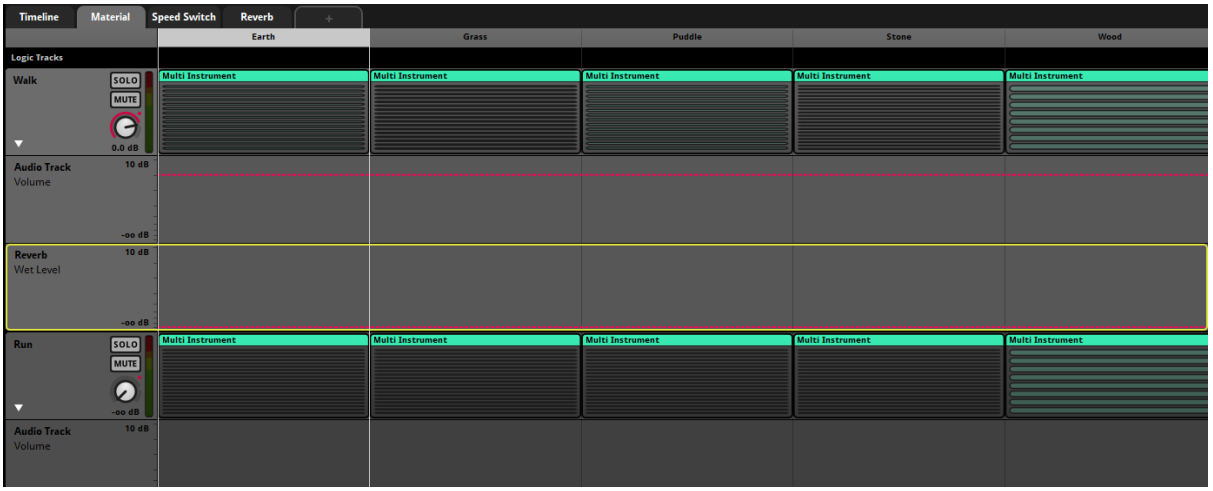
4.3 Efectos de sonido

En la carpeta *SFX (efectos de sonido)* contamos con los siguientes eventos: **los dos eventos para los puntos 360** (360Inner y 360Outer), **el sonido de escalada** (Climb), **sonidos de feedback para las encuestas** (CorrectAnswer e IncorrectAnswer), **el sonido de los pasos del personaje** (Footstep), **el sonido del salto y aterrizaje del personaje** (JumpLand) y **el sonido cuanto coges un objeto** (Notification).



El evento **Footstep** (pasos del personaje) está hecho usando un *multi-instrument*, que cuenta con varios parámetros para añadir más realismo al efecto de sonido:

- Material: Dependiendo de qué material pise el personaje, se reproducirán unos sonidos u otros para dar más realismo a los pasos. Es decir, cada parámetro etiquetado cuenta con un multi-instrument con los respectivos sonidos de ese material.
- Speed Switch: Existe una diferenciación entre los sonidos de los pasos del personaje dependiendo si éste corre o anda. Si el personaje empieza a correr, se reproducirá el segundo track, que contiene todos los multi-instruments con sus respectivos sonidos.
- Reverb: Si el personaje se encuentra dentro de la cueva, a todos los efectos de sonido que componen los pasos del personaje se les añadirá un efecto de Reverb para añadir mucho más realismo.



Los parámetros se han implementado en Unity de la siguiente forma:

- Material: Un script se encarga de almacenar la información del tipo de material del suelo. Con un raycast lanzado desde el personaje hacia abajo, al chocar con un objeto y comprobar el material del objeto gracias a ese script, podemos reproducir el sonido correspondiente fácilmente pasando como parámetro de entrada el material del suelo.

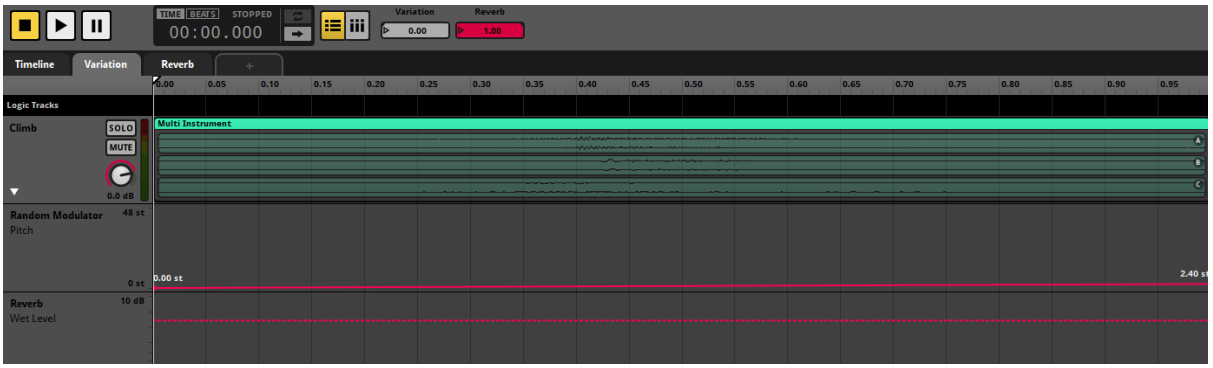
- Speed Switch: Si el personaje está corriendo, se le pasará el valor de "1" al valor continuo del parámetro de FMOD, si no, se le pasará el valor de "0".

- Reverb: De la misma forma que el Speed Switch, si el personaje se encuentra dentro de la cueva se le pasará el valor de "1" al valor continuo del parámetro de FMOD, si no, se le pasará el valor de "0".

```
if (_playerController.IsGrounded)
{
    EventInstance footstep = GameManager.Instance.audioManager.CreateInstance(GameManager.Instance.fmodEvents.GetEvent("Footstep"));
    FMODUnity.RuntimeManager.AttachInstanceToGameObject(footstep, transform, GetComponent<Rigidbody2D>());
    footstep.setParameterByName(_materialParameterName, _fMaterialValue);
    footstep.setParameterByName(_speedParameterName, _fPlayerRunning);

    if (_playerCharacter.EstoyCueva)
    {
        footstep.setParameterByName(_reverbParameterName, 1);
    }
    else
    {
        footstep.setParameterByName(_reverbParameterName, 0);
    }

    footstep.start();
}
```

El evento **Climb** (sonido de escalada del personaje) está hecho usando un *multi-instrument*, que cuenta con un parámetro *Variation* que permite realizar una variación en el pitch, además, cuenta con el mismo parámetro de *Reverb* que usan los pasos del personaje. Así, a pesar de tener pocos

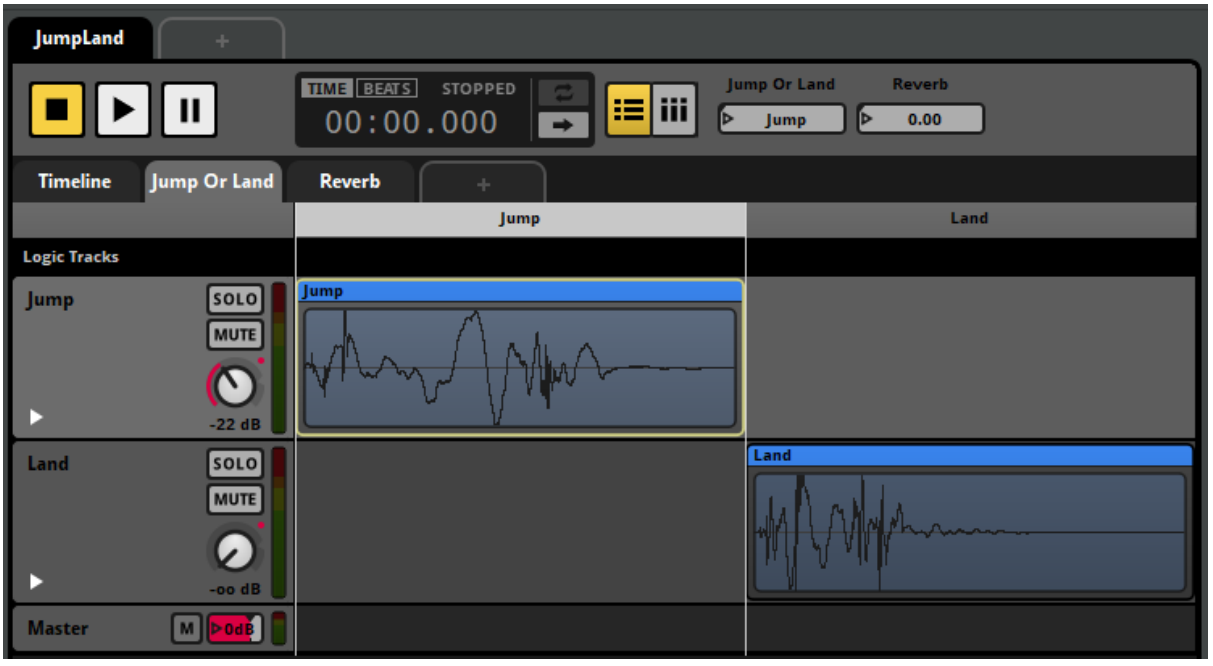
samples de escalada, podemos tener una pequeña variación haciendo más imperceptibles los bucles del sonido al escalar.



La variación viene dada por un modulador aleatorio del pitch. El parámetro de FMOD es un valor continuo que a medida que se acerque a “1”, el rango del modulador aleatorio aumentará.

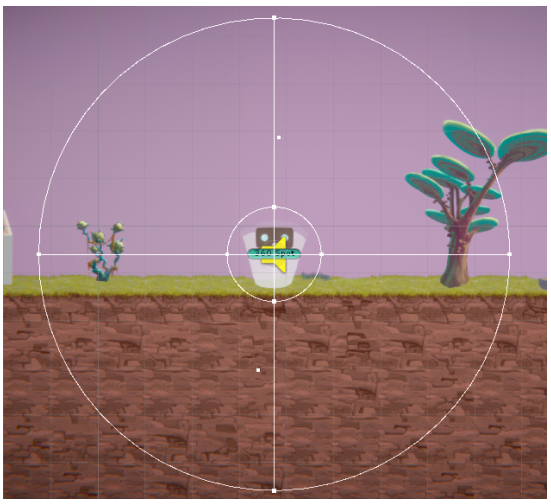
El evento **JumpLand** (sonido al saltar/aterrizar del personaje) cuenta con un parámetro por etiquetas, que dependiendo de su valor, hace que se reproduzca un sonido u otro.

Es decir, este parámetro actúa como booleano para decidir si se reproducirá el sonido del salto o del aterrizaje. Además, y de la misma forma que todos los efectos de sonido del personaje anteriores, cuenta con el parámetro *Reverb* para modificar el efecto de sonido en caso de que el personaje se encuentre dentro de la cueva.

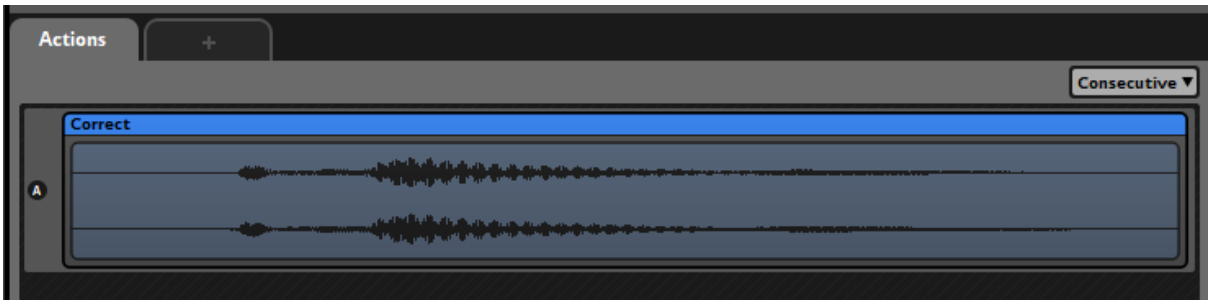


Los eventos 360Inner y 360Outer cuentan con un efecto de *Spatializer*. A medida que te acercas a estos objetos, se reproduce un sonido cada vez más fuerte (que resalta la importancia del objeto en el juego).

Además, contamos con dos eventos para estos objetos y, si el personaje se acerca a una determinada distancia, se escucharán los dos a la vez.



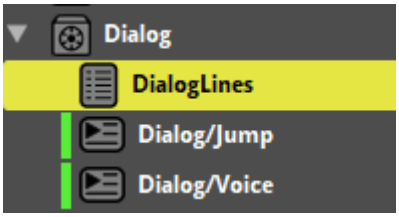
Los eventos de CorrectAnswer, IncorrectAnswer y Notification son *One-Shot Sounds.* Es decir, solo se reproducen una vez cuando se realiza una acción en concreto.



Por último, en el menú principal los botones emiten un One-Shot Sound al igual que los eventos anteriores.

4.4 Diálogos

Los diálogos son un tipo de sonido muy especial en videojuegos, pues pueden contar con un gran número de líneas de diálogo. Además, si el videojuego se dobla a otros idiomas, el número de ficheros de audio aumenta drásticamente y crear eventos individuales para cada línea se hace inviable. Lo más viable es implementar los diálogos utilizando **Programmer Instruments**, que sirven como sonido "placeholder" que el juego puede seleccionar en tiempo de ejecución para cada instancia de diálogo que se cree.



Para usar ficheros de audio que no se hayan asignado a ningún evento, se deben crear **tablas de audio**. La tabla *DialogLines* contiene todas las líneas de diálogo que dice el explorador. La voz del explorador fue grabada por Javier e interpretada por su compañero de instituto.

El evento de hablar es **Voice**, y éste incluye el *Programmer Instrument*. Desde Unity, una función PlayDialogue() crea una instancia de Voice, y selecciona el fichero de audio correspondiente.

```
27 references
public void PlayDialogue(string key) {
    dialogueInstance = RuntimeManager.CreateInstance(eventsDictionary["Dialog"]);

    GCHandle stringHandle = GCHandle.Alloc(key);
    dialogueInstance.setUserData(GCHandle.ToIntPtr(stringHandle));

    dialogueInstance.setCallback(dialogueCallback);
    dialogueInstance.start();
}
```

A los diálogos también se les aplica el efecto de **reverb** dentro de la cueva.

Como el juego fue desarrollado sin tener en cuenta esta ampliación sonora, sólo disponemos de una toma del sonido que hace el explorador cuando salta. Para disimular esta repetición, hemos creado un evento **Jump**, que reproduce el sonido con una variación aleatoria del pitch.

Finalmente, hemos limitado el número máximo de instancias de voz puesto que el protagonista no debería ser capaz de decir dos líneas o más simultáneamente.