Gestion du son 2D et 3D dans Unity

Formats de fichiers sonores acceptés dans Unity : .aif .wav .mp3 .ogg

Les fichiers mono, stéréo et multicanal (jusqu'à 8 pistes) sont supportés

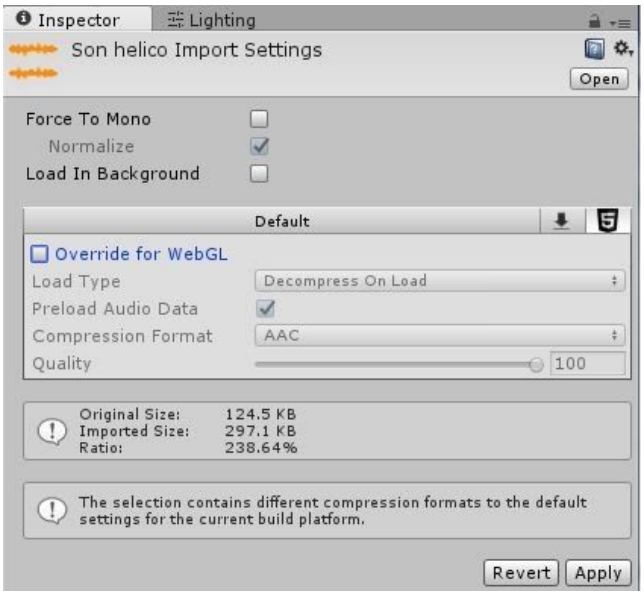
Unity gère également les "tracker modules" (Mod ou Module) en format .xm .mod .it (plus d'infos ici)

Propriétés des sons importés (*Audio Clip)*

Pour importer un son, le glisser directement dans l'onglet *project* (ou faire ***menu Assets-->Import New Asset...***)

Force To Mono : Converti un son stéréo ou multipiste en mono (1 piste). Si coché, l'option *normalize* deviendra accessible ce qui permet de normaliser le niveau du son converti (volume plus fort et plus égal).

Load In Background : Désactivé par défaut. Permet de commencer la lecture d'une scène sans que le son soit complètement chargé en mémoire. Par défaut, tous les sons sont chargés en mémoire avant que la scène ne commence. Attention : si on fait appel à un son alors qu'il n'est pas complètement chargé, il ne commencera pas à jouer immédiatement.



Default : sélectionnez les paramètres par défaut de la plateforme de publication**. Override for WebGl** lorsqu'il faut publier pour web**.**

Load Type

Decompress On Load : Les sons seront décompressés dès qu'ils sont chargés en mémoire. À choisir pour les sons courts où la synchronisation est importante. Attention : Un son encodé (ex. ogg Vorbis) décompressé après avoir été chargé en mémoire occupera environ 10 fois plus d'espace mémoire qu'un son compressé.

Compressed in memory: Les sons resteront compressés en mémoire jusqu'à leur lecture. Cela aura un léger impact sur les performances. À choisir pour les fichiers sonores plus volumineux.

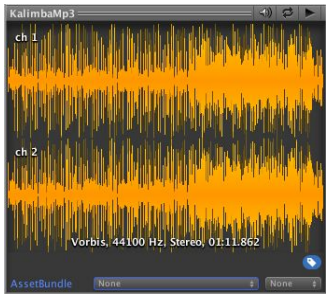
Streaming : Le son est décodé en temps réel, pendant sa lecture. Cette méthode n'a pas beaucoup d'impact sur l'utilisation de la mémoire. À choisir pour de la musique ou des ambiances.

Preload Audio Data : Activé par défaut. Le clip sera préchargé en mémoire avant l'activation de la scène.

Compression Format : Le format de compression utilisé pour compresser le son au moment de l'exécution du programme.

> **PCM** : Meilleure qualité, mais fichier plus volumineux en mémoire. À utiliser pour les sons très courts.
>
> **ADPCM** : Meilleur ratio de compression que PCM et moins gourmand en utilisation du processeur que Vorbis ou MP3. À utiliser principalement pour des sons qui se répètent très fréquemment (bruit de pas, bruit d'armes, etc.)
>
> **Vorbis/MP3** : Fichier plus petit, mais de qualité plus faible que PCM. La qualité de la compression peut-être ajustée avec le bouton glissière *Quality* qui apparaît lorsque ce format est choisi. À utiliser principalement pour des musiques ou des sons d'ambiance.

Dans la partie inférieure de l'inspecteur, vous pouvez voir une représentation du fichier sonore (ondes sonores). Différentes informations sont visibles (nombre de pistes, format sonore, échantillonnage, durée, etc.). Vous pouvez également écouter le son en utilisant le bouton de lecture.



Pour entendre le son : Audio Listener

Pour pouvoir entendre les sons, chaque scène devra inclure un "*Audio Listener*". Par défaut, ce composant est ajouté à **chaque caméra**.
Il est important de garder un seul « *Audio Listener* » actif en même temps, autrement un message d'avertissement s'affichera en bas de l'écran.

Propriétés

Propriété	Description	Exemple
volume	Contrôle le volume global du jeu (0.0 à 1.0)	AudioListener.volume = 0;
pause	Permet de mettre en pause le son global du jeu (true/false)	AudioListener.pause = true;

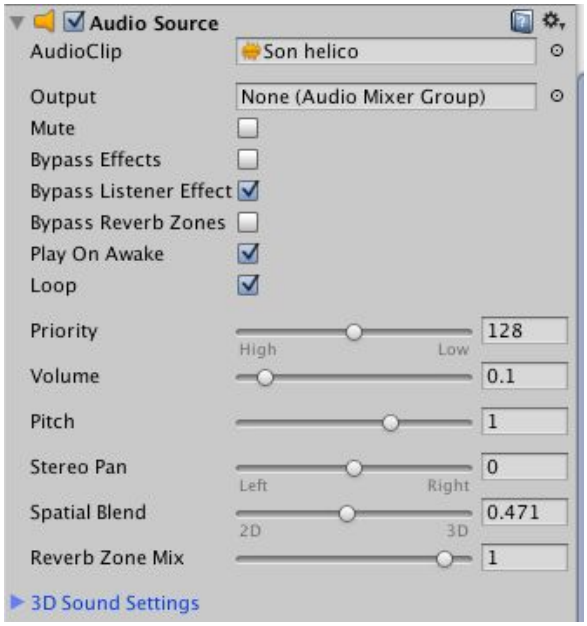
Ajouter un son à votre scène

Pour utiliser un son dans une scène, vous pouvez :

1- **Associez le son (*Audio Clip*) à un *gameObject* existant**.
Il suffit de prendre le son de l'onglet *project* et de le glisser sur un *gameObject* de la fenêtre *hierarchy*. Un composant *Audio Source* sera automatiquement ajouté à ce *gameObject*. Cette façon de faire permet d'associer à son à un objet du jeu. Si cet objet bouge, le son "bougera" avec lui puisque l'objet devient en quelque sorte l'émetteur du son.

2- **Associez le son (*Audio Clip*) à un *gameObject* vide** .
Il suffit de prendre le son de l'onglet *project* et de le glisser directement dans la *hierarchy*, sans le déposer sur un objet existant. Le son sera alors associé automatiquement à un *« gameobject »* vide qui se verra attribuer le composant *« Audio Source »*. À utiliser pour les sons qui ne sont pas directement associés à des objets, par exemple la musique du jeu ou les sons d'ambiance. **Note : Vous pouvez aussi créer un *gameObject* vide et lui associer un son par la suite. Le résultat sera le même.**

Propriétés du composant Audio Source



AudioClip: Le son (*audioClip)* qui sera joué par l'*Audio Source*. Vous pouvez "glisser/déposer" un son ici.

Output: Par défaut le son sera dirigé vers *l'Audio Listener* de la scène. Vous pouvez également diriger le son vers un *audio mixer* (plus d'infos sur les audio mixer ici)

Mute : Permet de mettre l'*Audio Source* en sourdine. S'il y a d'autres *Audio sources* dans la scène, elles continueront de jouer.

Ex. GetComponent<AudioSource>().mute = {true ou false}

Play On Awake: Si coché, le son sera lu dès le lancement de la scène ou dès que le gameObject est activé. S'il est décoché, le son sera en attente et pourra être déclenché avec la commande *Play().* Ex. GetComponent<AudioSource>().Play()

Loop: Détermine si le son joue en boucle ou non.

Priority : Détermine la priorité de cette *Audio Source*.
0 = Plus important, 256 = le moins important, 128 = valeur par défaut.
Ex. : Pour la musique d'un jeu, la priorité du son pourrait être fixée à 0 pour éviter toute coupure.

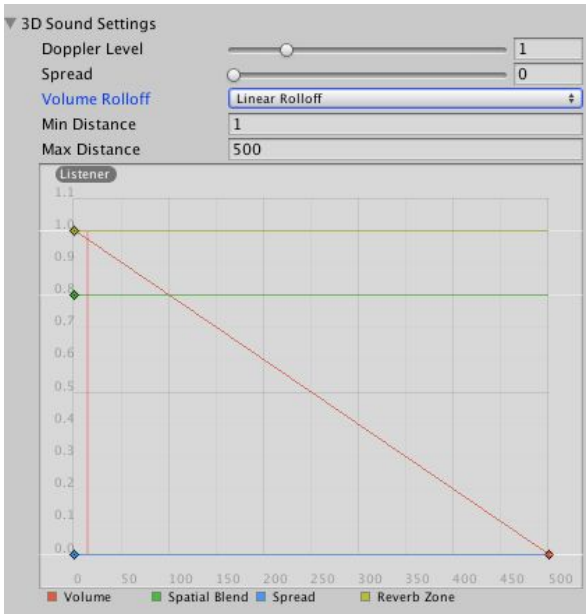
Volume: Contrôle le volume du son (0 à 1).

Pitch: Contrôle la vitesse de lecture du son. (Entre -3 et 3)
Ex. 1 = vitesse normale, 2 = double de la vitesse normale, 0.5 = la moitié de la vitesse normale, etc.

Stereo Pan : Permet de définir la spatialisation stéréo du son (effet gauche/droite). (Entre -1 et 1.)
Ex. 0 = son centré, -1 = seulement à gauche, 1 = seulement à droite, 0.75 = 75% à droite et 25% à gauche, etc.

Spatial blend : Entre 0 et 1. Permet de définir s'il s'agit d'un son 2D (valeur de 0) ou 3D (valeur de 1). Une valeur intermédiaire signifiera que le son sera en partie géré comme son 2D et comme son 3D.
Ex. Une valeur de 0.8 indique que le son est principalement géré (80%) par l'engin 3D. Le volume et la spatialisation de ce son seront calculés dynamiquement, en fonction de la distance entre l'émetteur (l'*AudioSource*) et le récepteur (l'*AudioListener)*. Le 20% restant sera géré comme un son 2D, c'est-à-dire audible en permanence et de façon égale au niveau du volume et de la spatialisation.

Propriétés des sons 3D



Doppler Level : (Entre 0 et 5) Permet de contrôler la modification de hauteur du son (pitch) qui est calculée par Unity pour les sons 3D. Pour éviter de drôles d'effets avec les fichiers musicaux, mettre cette valeur à 0.

Spread: Entre (0 et 360 degrés). Contrôle la façon dont le positionnement 3D du son affectera la panoramisation du son. Utile pour les sons d'ambiance dont la panoramisation doit être ajustée en fonction de la distance.
Ex : 0 = la panoramisation est déterminée complètement par la position 3D, 180 = le son utilise sa panoramisation originale, >180 = inverse la panoramisation, 360 = comme à 0, mais inversé.

Volume Rolloff: Permet d'ajuster le volume du son en fonction de la distance entre l'émetteur (*audioSource*) et le récepteur (*audioListener*).

- Logarithmic Rolloff : Le volume est plus fort lorsque *l'AudioListener* est près de *l'AudioSource*, mais il diminue rapidement lorsque la distance augmente.
- Linear Rolloff : Diminution plus progressive du volume lorsque la distance entre *l'AudioListener* et *l'AudioSource* augmente.
- Custom Rolloff : Ajustement du volume selon les paramètres que vous spécifiez.

Min Distance/Max distance : Entre 0 et la distance minimale, le son aura son volume le plus élevé. Entre la distance minimale et la distance maximale, le volume diminuera progressivement. Au-delà la distance maximale, le son ne diminuera pas davantage.

Graphique : Représentation graphique illustrant les propriétés mentionnées précédemment. L'axe horizontal représente la distance entre *l'AudioListener* et *l'AudioSource* et l'axe vertical représente le volume. La courbe peut être modifiée manuellement.

Quelques méthodes et propriétés du composant Audio Source (liste complète ici)

Propriétés

Propriété	Description	Exemple
isPlaying	Est-ce que le son joue? (true/false)	if (GetComponent<AudioSource>().isPlaying == true)
mute	Permet de mettre en sourdine (true/false)	GetComponent<AudioSource>().mute = true;
pitch	Permet de changer la vitesse de lecture (-3.0 à 3.0)	GetComponent<AudioSource>().pitch = 1;
volume	Permet de contrôler le volume (0.0 à 1.0)	GetComponent<AudioSource>().volume = 0.5;

Méthodes

Propriété	Description	Exemple
Pause	Pause de la lecture du son.	GetComponent<AudioSource>().Pause();
Play	Joue le clip spécifié dans l'inspecteur	GetComponent<AudioSource>().Play();
PlayDelayed	Joue le clip spécifié dans l'inspecteur après un délai en secondes.	GetComponent<AudioSource>().PlayDelayed(5);

PlayOneShot	Permet de jouer un son (audioClip)	AudioClip leSon ; // à définir dans l'inspecteur
		GetComponent<AudioSource>().PlayOneShot(leSon);
		GetComponent<AudioSource>().PlayOneShot(leSon,volume);

Stop	Arrête la lecture du son.	GetComponent<AudioSource>().Stop();
UnPause	Repart la lecture du son après une pause.	GetComponent<AudioSource>().UnPause();

Exemple de code

Script qui déclenche un son à chaque fois que la touche espace est appuyée

```
AudioSource laPiste; // pour garder en mémoire le composant AudioSource
AudioClip leSon ; // son à définir dans l'inspecteur

//Au lancement du jeu...
void Start ()
{
   laPiste = GetComponent<AudioSource>(); // on met dans la variable laPiste la référence au composant
                                          //  AudioSource de l'objet.
}

//À chaque frame...
void Update ()
{
   if(Input.GetKeyDown(KeyCode.Space)) // On vérifie si la touche espace est enfoncée
   {
      laPiste.PlayOneShot(leSon); // Si oui, on joue le son.
   }
}
```
