ADAPTIVE AUDIO MANAGER FOR UNITY3D (AAMU)

MANUAL

This manual will guide Unity game developers to use the Adaptive Audio Manager for Unity Package on their games.

Vertical Layering Technique: *Used to play one or more tracks simultaneously, controlling them individually by turning them on or off, fading in, fading out, cross fading etc. There are three important classes to understand the VerticalLayeringManager:*

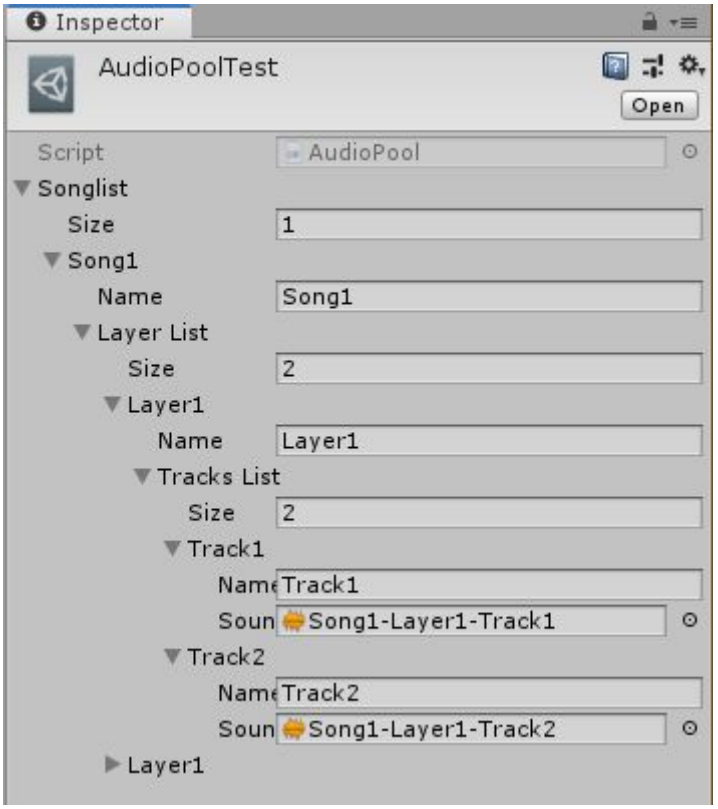
> ***Track:*** *Represents a* ***single*** *instrument/sound of a song. Playing a Track alone will only play the track itself.*
> ***Layer****: Represents a group of instruments that is part of a song and holds a* ***list of Tracks.*** *Playing a Layer will play all the tracks contained in that Layer. For example, if we have a Song called "Rock for Opera" we can divide his instruments into two Layers, one called "Orchestra Instruments", where we will put the Orchestra Tracks like Violin, Piano, Trumpets etc, and another Layer called "Rock Instruments" where we will put our Rock Tracks like Electric Guitar, Bass and Drums. That way we can control all rock instruments at once.*
> ***Song:*** *Represents a full song and holds a* ***list of Layers.*** *Playing a song will play all the layers contained on that layerList, which will result in playing all the tracks contained in all the layers.*

How to use:

1. ***Create a VerticalAudioManager Object on your scene: The VerticalAudioManager is a singleton which will be used to control your game audio (Can be accessed using VerticalAudioManager.Instance);***
2. ***Set a default AudioSource for the VerticalAudioManager (can use the AAMU prefab: Assets/Prefabs);***
3. ***Create an AudioPool Scriptable Object (Assets -> Create -> AudioPool) and reference him on the VerticalAudioManager editor: The AudioPool holds a list of Songs that will be used in your game;***
4. ***Set the name and the AudioClip of each Song, Layer and Track on your AudioPool;***
5. ***Use the VerticalAudioManager methods to control your game's audio;***

PS: It's recommended to use OnTriggerEnter2D and OnTriggerExit2D to trigger the VerticalAudioManager methods on the scene.

VerticalAudioManager ready for use



AudioPool example: This AudioPool has one Song divided into two Layers. The Layer1 has two Tracks: Track1 and Track2.

VerticalAudioManager Methods:

- Used to search and return a specific Song, Layer or Track by it's name:
  `SearchForSong(string songName)` (case insensitive)
  `SearchForLayer(string layerName)` (case insensitive)
  `SearchForTrack(string trackName)` (case insensitive)

- Used to play a specific Song, Layer or Track:
  `Play(Song song, float time = 0f, bool loop = true, float volume = 1)`
  `Play(Layer layer, float time = 0f, bool loop = true, bool stopOtherLayers = false, float volume = 1)`
  `Play(Track track, float time = 0f, bool loop = true, bool stopOtherTracks = false, float volume = 1)`

  `time = Defines the time of the tracks from where it will start playing (Can use the Song.GetTrackTime() method to get the time of a track from a Song that is being played. Works for Layer.GetTrackTime() and Track.GetTrackTime() too)`
  `stopOtherLayers = Defines if the Layer will play simultaneously with other layers or if it will stop other layers`
  `stopOtherTracks = Defines if the Track will play simultaneously with other tracks or if it will stop other tracks`

```
Example:
VerticalAudioManager.Instance.Play(audioManager.SearchForLayer("GuitarAndBass"),
```

```
time: audioManager.SearchForSong("TestSong").GetTrackTime(), volume: 0.8f, loop:
false);

Result:
 -  Play all tracks listed inside the layer “GuitarAndBass”;
 -  Start playing from the time returned from the TestSong;
 -  The volume will be 80% of the max volume;
 -  Stop playing at the end of the execution because loop is false;
```

- Used to stop all Songs, or a specific Song, Layer or Track:
  `Stop(Song song)`
  `Stop(Layer layer)`
  `Stop(Track track)`
  `StopAll()`

- Used to pause all Songs, or a specific Song, Layer or Track:
  `Pause(Song song)`
  `Pause(Layer layer)`
  `Pause(Track track)`
  `PauseAll()`

- Used to resume a specific Song, Layer or Track from the time where it has been paused:
  `Resume(Song song, bool loop = true)`
  `Resume(Layer layer, bool loop = true)`
  `Resume(Track track, bool loop = true)`

- Used to FadeIn (Play smoothly) a specific Song, Layer or Track:
  `FadeIn(Song song, AnimationCurve animationCurveType, bool loop = true, float fadeDuration = 3f, float volume = 1f, float time = 0f)`
  `FadeIn(Layer layer, AnimationCurve animationCurveType, bool loop = true, float fadeDuration = 3f, float volume = 1f, float time = 0f)`
  `FadeIn(Track track, AnimationCurve animationCurveType, bool loop = true, float fadeDuration = 3f, float volume = 1f, float time = 0f)`

  `animationCurveType = Defines the curve which the increasing volume will follow. You can create your own curve or use the VerticalAudioManager.defaultCurve`
  `time = Defines the time of the tracks from where the fade will start (Can use the Song.GetTrackTime() method to get the time of a track from a Song that is being played. Works for Layer.GetTrackTime() and Track.GetTrackTime() too)`

```
Example:
audioManager.FadeIn(audioManager.SearchForLayer("GuitarAndBass"),
audioManager.defaultCurve, time: audioManager.SearchForSong("TestSong").GetTrackTime(),
volume: 1, loop: true, fadeDuration: 5);

Result:
 -  Fade in all tracks listed inside the layer “GuitarAndBass”;
 -  Start from the time returned from the TestSong;
 -  Repeat playing when finished because loop is true;
 -  Fade in from 0 to 100% of the max volume;
 -  Progression of volume will follow the defaultCurve lasting 5 seconds;
```

- Used to FadeOut (Stop smoothly) a specific Song, Layer or Track:

```
FadeOut(Song song, AnimationCurve animationCurveType, float fadeDuration = 3f)
FadeOut(Layer layer, AnimationCurve animationCurveType, float fadeDuration = 3f)
FadeOut(Track track, AnimationCurve animationCurveType, float fadeDuration = 3f)
FadeOutAll(AnimationCurve animationCurveType, float fadeDuration = 3f)

animationCurveType = Defines the curve which the decreasing volume will follow. You can
create your own curve or use the VerticalAudioManager.defaultCurve
```

```
Example:
audioManager.FadeOutAll(audioManager.defaultCurve, fadeDuration: 2);

Result:
 -  Fade out all tracks listed inside the AudioPool;
 -  Fade out from current volume to 0;
 -  Progression of volume will follow the defaultCurve lasting 2 seconds;
```

- Used to CrossFade (Play smoothly and Stop smoothly) two different Songs, Layers or Tracks:

```
CrossFade(Track trackIn, Track trackOut, AnimationCurve animationCurveIn,
AnimationCurve animationCurveOut, bool loop = true, float fadeDuration = 3f, float
volume = 1f, float time = 0f)
CrossFade(Layer layerIn, Layer layerOut, AnimationCurve animationCurveIn,
AnimationCurve animationCurveOut, bool loop = true, float fadeDuration = 3f, float
volume = 1f, float time = 0f)
CrossFade(Song songIn, Song songOut, AnimationCurve animationCurveIn, AnimationCurve
animationCurveOut, bool loop = true, float fadeDuration = 3f, float volume = 1f, float
time = 0f)

animationCurveIn = Defines the curve which the increasing volume will follow. You can
create your own curve or use the VerticalAudioManager.defaultCurve
animationCurveOut = Defines the curve which the decreasing volume will follow. You can
create your own curve or use the VerticalAudioManager.defaultCurve
```

time = Defines the time of the tracks from where the fade will start (Can use the Song.GetTrackTime() method to get the time of a track from a Song that is being played. Works for Layer.GetTrackTime() and Track.GetTrackTime() too)

Example:

```
audioManager.CrossFade(audioManager.SearchForLayer("Layer1"),
audioManager.SearchForLayer("Layer2"), animationCurveIn: audioManager.defaultCurve,
animationCurveOut: audioManager.defaultCurve, time:
audioManager.SearchForSong("TestSong1").GetTrackTime(), volume: 1, loop: true,
fadeDuration: 4);
```

Result:

- Fade in all tracks listed inside the layer “Layer1”;
- Fade out all tracks listed inside the layer “Layer2”;
- Progression of fadeIn’s volume will follow the defaultCurve lasting 4 seconds;
- Progression of fadeOut’s volume will follow the defaultCurve lasting 4 seconds;
- Start from the time returned from the TestSong;
- Fade in from 0 to 100% of the max volume;
- Repeat playing when finished because loop is true;

Track, Layer and Song Methods:

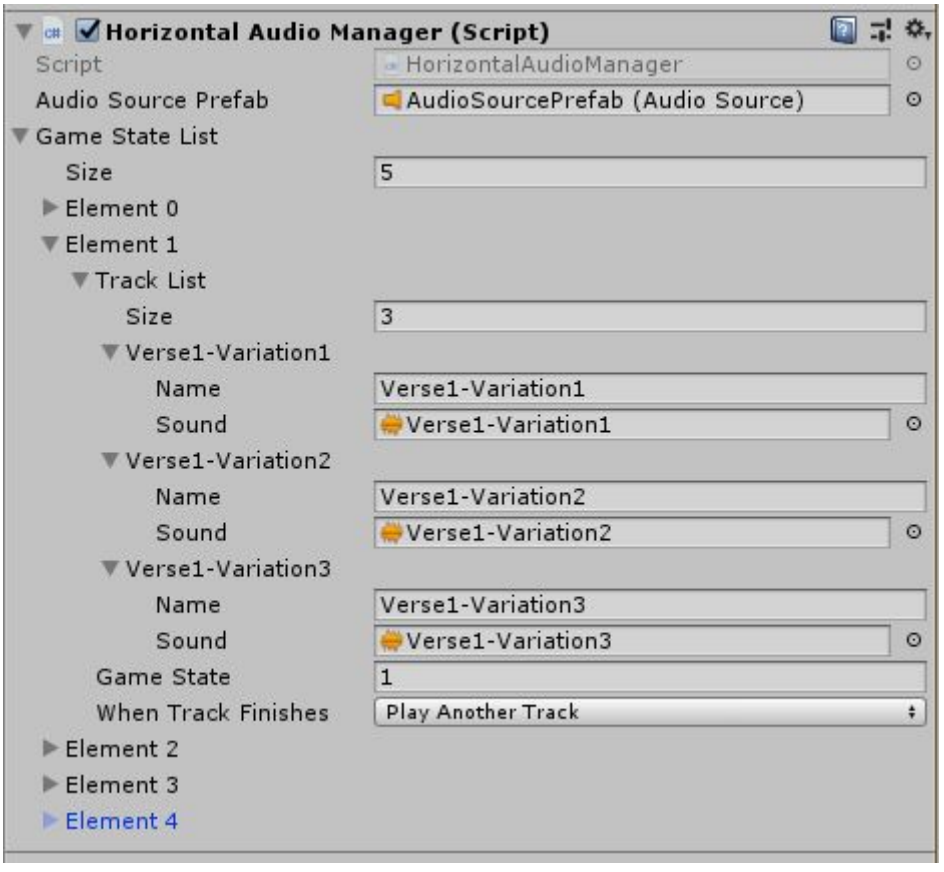
`SetVolume(float volume)`

`GetTrackTime()` - returns the current time of the Track, Layer or Song

Horizontal Resequencing Technique: *Used to play different track sequences according to the current state of the game. A Game State represents one State of the game that can change the audio dynamics. For example, an adventure game can have a state named "Calm", when there are no enemies around and another state called "Combat" when there is one or more enemies around. Each GameState has a list of Tracks that can be played if the game is currently at that Game State. If the current Game State has more than one track, the HorizontalAudioManager will randomly choose one of them to play each time.*

How to use:

1. ***Create a HorizontalAudioManager Object on your scene: The HorizontalAudioManager is an object which will be used to control your game audio and state;***
   Track: Represents a single instrument/sound
2. ***Set a default AudioSource for the HorizontalAudioManager (can use the AAMU prefab);***
3. ***Set the HorizontalAudioManager's GameStateList size according to the number of states you will be using (Starts from Game State 0);***
4. ***For each GameState on your GameStateList, set the GameState:***
   - ***Number of tracks that will be randomized on this Game State;***
   - ***Name of the track;***
   - ***AudioClip of the track;***
   - ***whenTrackFinishes - Determines what will happen when the AudioClip ends playing;***
     - *PlayAnotherTrack - Play a random track from the currentGameState,*
     - *PlayAndIncrement - Play one time, then increments 1 to the currentGameState variable,*
     - *PlayAndStop - Play one time then stop playing the track from the currentGameState*
5. ***Call the HorizontalAudioManager.GameStatePlay method to start playing and use your own code to control the HorizontalAudioManager's state;***



HorizontalAudioManager example: This HorizontalAudioManager has 5 States. As we can see, the Element1 represents Game State 1, that has 3 different Tracks that can be reproduced when the system is in that GameState. When Track Finishes, the System plays another Track without changing the GameState.

HorizontalAudioManager Methods:

- Used to search and return a specific Song, Layer or Track by it's name:
  `CurrentGameState - Variable used to directly set and get HorizontalAudioManager's GameState`
  `StartPlaying() - Start playing the HorizontalAudioPlayer from the CurrentGameState`
  `StopPlaying() - Stop the HorizontalAudioManager (after finish playing the current track)`