Game Audio Scripting Essentials Documentation

Table of Contents

Overview

The Game Audio Scripting Essentials are a collection of three core scripts that add to the functionality of Unity's base audio system. The primary uses of these scripts are to add randomized sound effects to objects that may have multiple sounds for the same actions, such as footsteps, weapon attacks, and UI sounds, as well as the ability to implement adaptive music systems all in the inspector. This overview will describe the usage of each of the three scripts.

AudioRandomizerContainer.cs

This is a scriptable object that acts as a container for multiple audio files and the settings associated with it. Through this object in the inspector, the user can set up the various settings normally seen in the Audio Source component (Audio Mixer Output Group, a Loop toggle, Priority, Volume, Stereo Pan, and Spatial Blend) as well as the additional settings provided by the tool (a list for the Audio Clips, a toggle for No Repeats, and Randomized Pitch toggle with a minimum and maximum pitch).

Within the object there are multiple functions that are public as well, primarily getters and setters for most of the variables, and functions that return the length of audio clips in the container as well.

AudioClipRandomizer.cs

This is the primary way for audio clips to be played through this system. This script is added as a component to a game object in the scene or through script, and when an Audio Randomizer Container object (or an array of audio clips) are added to the component, it can then play one when needed by the user. The majority of the fields on this script are carried over from the

Audio Randomizer Container object, with the option to override the former's settings through this component.

The functions that the user will primarily need to interact with are the PlaySFX() function, creating an audio source for a randomized clip to play from and destroying it when completed, the StopSFX() function, stopping but not destroying the current source, the DestroySFX() function, destroying the current source immediately, and the volume property (SFXVolume), allowing the user to input a float to adjust the volume. Each of these functions are public and are recommended to be accessed through Unity Events for ease of access.

AdaptiveMusicContainer.cs

This is the most complex script in the package, allowing the user to set up adaptive music, adaptive ambience, and all matters of adaptive sound effects. This script is added to a game object or dragged as the prefab into the scene, and it has quite the long list of settings and options. For a brief explanation of how this works, each part of the music is set up as sections, which are a collection of audio layers. Each layer of the section is played simultaneously and is looped, with the volume depending on the states. Each section also has transitions to different sections, and they can be customized through different trigger types, crossfading, and quantization options. Alongside the sections, there are also different states that can change the volumes of the different layers of the section, and can be used to fade in or out different layers depending on the gameplay situations.

The functions that are primarily called from Unity Events or through script would be the following: the RunContainer() function, acting as a manual start for the container itself; the SetState() function, changing the state of the container; and the TransitionSection() function, calling the index of the current section's transitions, transitioning to the region specified in the container.

Quick Start Guide

Here are a few use cases and how to easily set it up for your game.

Randomized Footstep Sounds

Once a character controller is set up, add a serialized field AudioClipRandomizer object to the script, and add an AudioRandomizerContainer with the necessary audio files to the character controller script in the inspector.

When the character is moving, start a coroutine dedicated to the footsteps, which also should only play if it is not already playing. Inside the coroutine, run the PlaySFX() function for the AudioClipRandomizer object, and put a "yield return new WaitForSeconds(x)" after playing, with x being the length of time you want between footsteps. I would recommend using the player's speed to adjust the variable to have it feel right depending on their speed.

Vertical Adaptive Music

Start by adding the AdaptiveMusicContainer prefab or script to the scene, and create a new section, then add the layers to the section. Once the container is ready, add the initial state and set the volumes for each of the layers, then add another state with different volumes according to what you need for the state.

After the states are set up, either set up a UnityEvent on a trigger volume or whatever you want to use to trigger the state change and run the SetState() function using the index of the state you want to switch to, or add the SetState() function to trigger it in script.

Horizontal Adaptive Music

Once again, add the AdaptiveMusicContainer prefab or script to the scene, and create a new section with the layers for that section, then make another section that will be transitioned into. With the sections created and filled out, add a section transition on the one you want to transition out of, putting the index of the target section as the value of the Transition Into field. After setting the trigger type, fade type, and quantization of your choice, you are ready to get it functional.

Much like the vertical adaptive music example, the best options are to either set up a UnityEvent on a trigger volume or whatever you choose as the transition point to the new section using the TransitionSection() function and inputting the index of the transition, or by triggering that function through script.