What is Audio?

Audio is most simply described as electrical energy (active or potential) that represents sound. Sound is defined as a vibration that typically propagates as an audible longitudinal wave of pressure through a transmission medium such as a gas, liquid or solid as longitudinal waves and also as a transverse wave in solids (see Longitudinal and transverse waves, below).

1.Audible sound, as we hear it, is within the frequency range of 20 Hz and 20,000Hz.

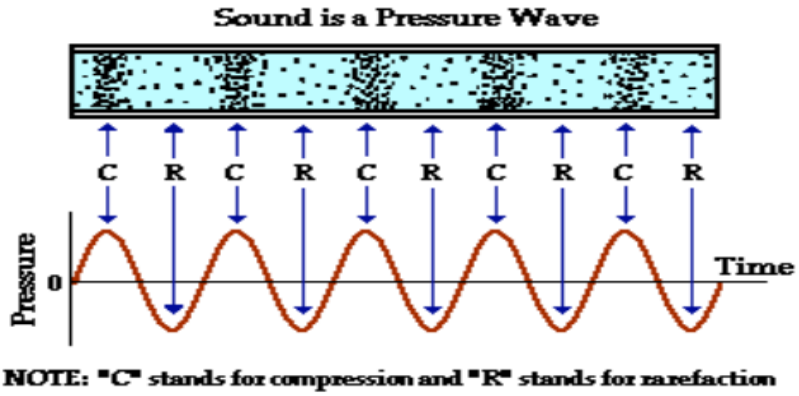
2.Infrasound is inaudible sound below 20 Hz.

3.Ultrasound is inaudible sound above 20,000 Hz.

Before we go ahead and understand the features and components of Unity Audio System, let us understand the nature of Sound: -

Nature of Sound:

The sound waves are generated by a sound source, such as the vibrating diaphragm of a stereo speaker. Sound is a longitudinal, mechanical wave. Sound can travel through any medium, but it cannot travel through a vacuum. There is no sound in outer space. Sound is a variation in pressure. A region of increased pressure on a sound wave is called compression (or condensation). A region of decreased pressure on a sound wave is called rarefaction (or dilation).



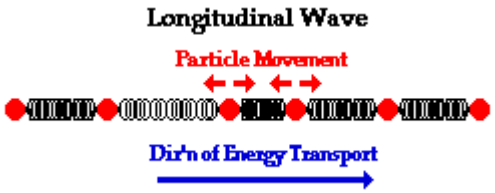
- Amplitude (Am) It is basically a measurement of the vertical distance of the trough(C) or crest(A) of a wave from the average(B) as illustrated in diagram above.
- Wavelength (λ) The distance between adjacent troughs or adjacent crests, measured in unit of length such as meters and expressed by symbol λ (lambda). For longitudinal wave, it will be distance between two successive rarefactions or compressions.
- Time Period (T) This defines the time it takes for one complete wave to pass a given point, measured in seconds (s) (WL = T).

- 4. Frequency (n) The number of complete waves that pass a point in one second, measured in Hertz (Hz). In simple words, the number of times the bob reaches appoint in (say A) in one second determines its frequency or repetitively.
- 5. Speed or velocity (v) The speed of the wave is proportional to the magnitude of the frequency. Wave speed is defined as the distance travelled by a wave disturbance in one second and is measured in meters/second ms–1 or m/s. Speed is scalar quantity while velocity is a vector quantity.

Not all of these properties are independent; one can relate some. Period is inversely related to the frequency. This means if the frequency is high, the period will be low. This is understandable because frequency is number of times a wave completes a set of up and down movements (or a set of crests and troughs) in 1 second. If these occur more frequently, it has to be done in very short time. Mathematically one may say period T = 1/n Where 'n' is frequency. We just said that wavelength is equal to the distance between two successive crusts or troughs. In one second this distance is covered a number of times given by frequency. So, Velocity = frequency × wavelength or V = n × λ.

1. Longitudinal Waves:

Longitudinal waves are waves in which the displacement of the medium is in the same direction as, or the opposite direction to, the direction of propagation of the wave. Mechanical longitudinal waves are also called *compressional* or compression waves, because they produce compression and rarefaction when traveling through a medium, and pressure waves, because they produce increases and decreases in pressure.



In the case of longitudinal harmonic sound waves, the frequency and wavelength can be described by the formula :-

$$y(x,t) = y_0 \cos\left(\omega\left(t - \frac{x}{c}\right)\right)$$

- *y* is the displacement of the point on the traveling sound wave.
- *x* is the distance the point has traveled from the wave's source.

- t is the time elapsed.
- y_0 is the amplitude of the oscillations,
- c is the speed of the wave.
- ω is the angular frequency of the wave.

The quantity x/c is the time that the wave takes to travel the distance x.

The ordinary frequency (f) of the wave is given by

$$f = \frac{\omega}{2\pi}.$$

The wavelength can be calculated as the relation between a wave's speed and ordinary frequency.

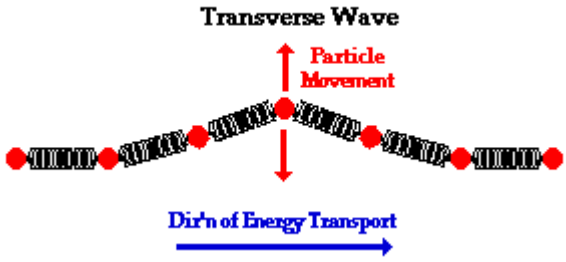
$$\lambda = \frac{c}{f}.$$

For sound waves, the amplitude of the wave is the difference between the pressure of the undisturbed air and the maximum pressure caused by the wave.

Sound's propagation speed depends on the type, temperature, and composition of the medium through which it propagates.

2. Transverse Waves:

In physics, a transverse wave is a moving wave whose oscillations are perpendicular to the direction of the wave or path of propagation. waves that are created on the membrane of a drum is an example of transverse wave.



The motion of such a wave can be expressed mathematically as follows. Let d be the direction of propagation (a Vector with unit length), and o any reference point in the medium. Let u be the direction of the oscillations (another unit-length vector perpendicular to d).

The displacement of a particle at any point *p* of the medium and any time *t* (seconds) will be

$$S(p,t) = Au \sin\left(\frac{t - (p-o)\cdot\frac{d}{v}}{T} + \phi\right)$$

where *A* is the wave's amplitude or strength, *T* is its period, *v* is the speed of propagation, and *φ* is its phase at *o*. All these parameters are real numbers. The symbol "•" denotes the inner product of two vectors.

By this equation, the wave travels in the direction *d* and the oscillations occur back and forth along the direction *u*. The wave is said to be linearly polarized in the direction *u*.

An observer that looks at a fixed-point *p* will see the particle there move in a simple harmonic(sinusoidal) motion with period *T* seconds, with maximum particle displacement *A* in each sense; that is, with a frequency of $f = 1/T$ full oscillation cycles every second. A snapshot of all particles at a fixed time *t* will show the same displacement for all particles on each plane perpendicular to *d*, with the displacements in successive planes forming a sinusoidal pattern, with each full cycle extending along *d* by the wavelength $\lambda = v\,T = v/f$. The whole pattern moves in the direction *d* with speed *V*.

Unity Audio Overview:

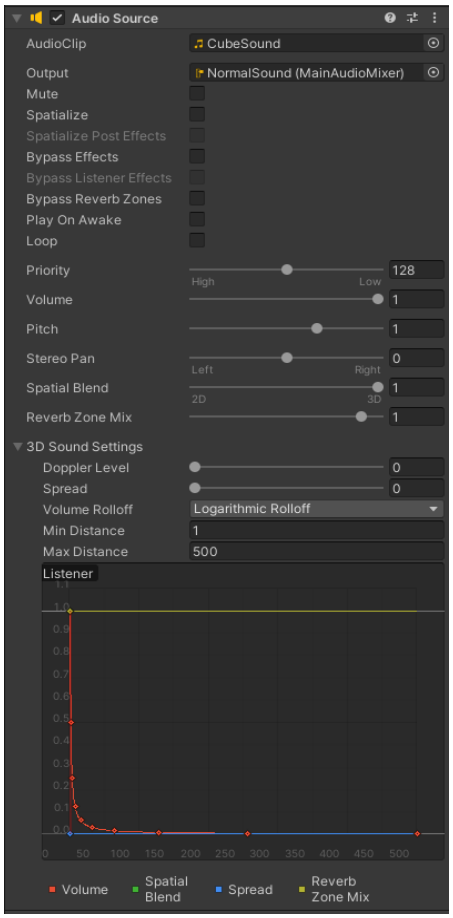
Unity's audio system is flexible and powerful. It can import most standard audio file formats and has sophisticated features for playing sounds in 3D space, optionally with effects like echo and filtering applied. Unity can also record audio from any available microphone on a user's machine for use during gameplay or for storage and transmission.

Audio Parts:

- Audio Source: To simulate the effects of position, Unity requires sounds to originate from Audio Sources.
- Audio Listener: The sounds emitted are then picked up by an Audio Listener attached to another object, most often the main camera.
- Audio Filters: Unity can't calculate echoes purely from scene geometry but you can simulate them by adding Audio Filters to objects.
- Audio Mixer: Audio Mixer allows you to mix various audio sources, apply effects to them, and perform mastering.
- Reverb Zones: Reverb Zones take an Audio Clip and distorts it depending where the audio listener is located inside the reverb zone.

1. Audio Source:

The Audio Source plays back an Audio Clip in the scene. The clip can be played to an audio listener or through an audio mixer. In the image Audio Source component is attached to Gameobject named AudioSource.



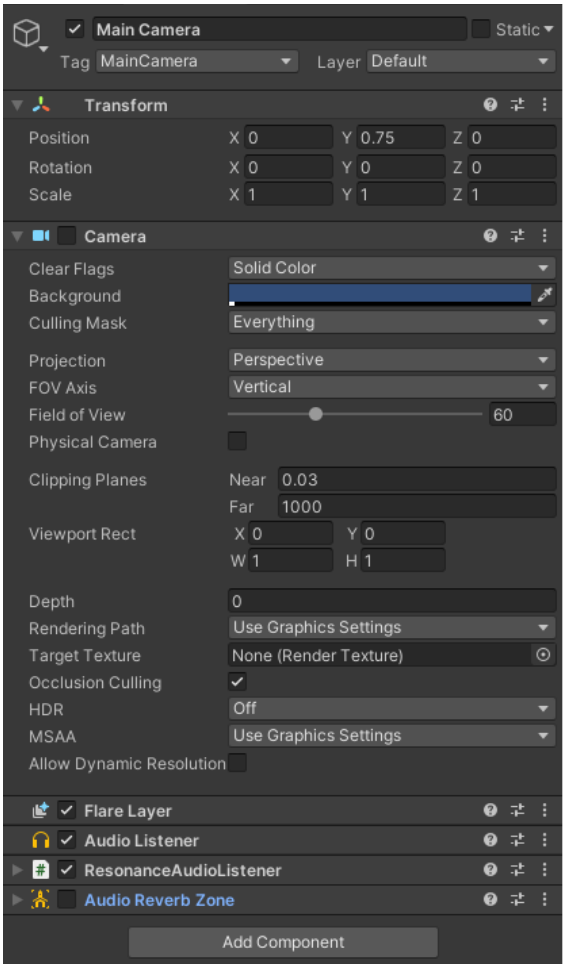
Property:	Function:
Audio Clip	Reference to the sound clip file that will be played.
Output	By default, the clip is output directly to the Audio Listener in the Scene. Use this property to output the clip to an Audio Mixer instead.

Mute	If enabled the sound will be playing but muted.
Bypass Effects	This is to quickly “by-pass” filter effects applied to the audio source. An easy way to turn all effects on/off.
Bypass Listener Effects	This is to quickly turn all Listener effects on/off.
Bypass Reverb Zones	This is to quickly turn all Reverb Zones on/off.
Play On Awake	If enabled, the sound will start playing the moment the scene launches. If disabled, you need to start it using the Play() command from scripting.
Loop	Enable this to make the Audio Clip loop when it reaches the end.
Priority	Determines the priority of this audio source among all the ones that coexist in the scene. (Priority: 0 = most important. 256 = least important. Default = 128.). Use 0 for music tracks to avoid it getting occasionally swapped out.
Volume	How loud the sound is at a distance of one world unit (one meter) from the Audio Listener .
Pitch	Amount of change in pitch due to slowdown/speed up of the Audio Clip. Value 1 is normal playback speed.
Stereo Pan	Sets the position in the stereo field of 2D sounds.
Spatial Blend	Sets how much the 3D engine has an effect on the audio source.
Reverb Zone Mix	Sets the amount of the output signal that gets routed to the reverb zones. The amount is linear in the (0 - 1) range, but allows for a 10 dB amplification in the (1 - 1.1) range which can be useful to achieve the effect of near-field and distant sounds.
3D Sound Settings	Settings that are applied proportionally to the Spatial Blend parameter.

Doppler Level	Determines how much doppler effect will be applied to this audio source (if is set to 0, then no effect is applied).
Spread	Sets the spread angle to 3D stereo or multichannel sound in speaker space.
Min Distance	Within the MinDistance, the sound will stay at loudest possible. Outside MinDistance it will begin to attenuate. Increase the MinDistance of a sound to make it 'louder' in a 3d world, and decrease it to make it 'quieter' in a 3d world.
Max Distance	The distance where the sound stops attenuating at. Beyond this point it will stay at the volume it would be at MaxDistance units from the listener and will not attenuate any more.
Rolloff Mode	How fast the sound fades. The higher the value, the closer the Listener has to be before hearing the sound. (This is determined by a Graph).
- Logarithmic Rolloff	The sound is loud when you are close to the audio source, but when you get away from the object it decreases significantly fast.
- Linear Rolloff	The further away from the audio source you go, the less you can hear it.
- Custom Rolloff	The sound from the audio source behaves accordingly to how you set the graph of roll offs.

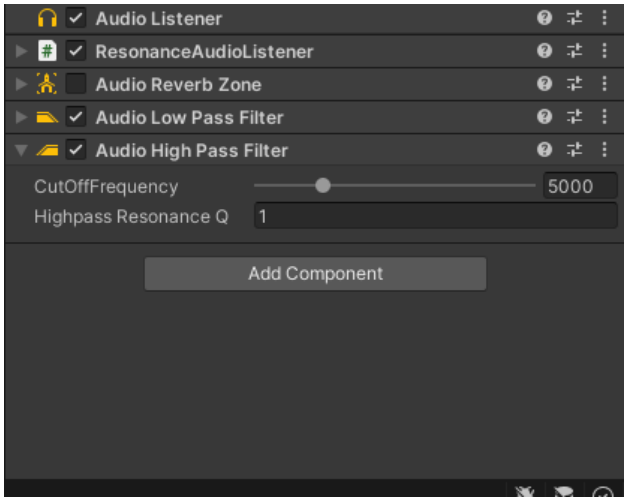
2. Audio Listener:

The Audio Listener acts as a microphone-like device. It receives input from any given Audio Source in the scene and plays sounds through the computer speakers. For most applications it makes the most sense to attach the listener to the Main Camera. If an audio listener is within the boundaries of a Reverb Zone reverberation is applied to all audible sounds in the scene. Furthermore, Audio Effects can be applied to the listener and it will be applied to all audible sounds in the scene. In the below image Audio Listener component is attached to the Gameobject camera.



3. Audio Filters:

You can modify the output of Audio Source and Audio Listener components by applying Audio Effects. These can filter the frequency ranges of the sound or apply reverb and other effects. In the above image I have attached an Audio Reverb Zone to get take an Audio Clip and distorts it depending where the audio listener is located inside the reverb zone. They are used when you want to gradually change from a point where there is no ambient effect to a place where there is one, for example when you are entering a cavern. Another example shown below for audio effect attached to audio listener Gameobject.



- ## Audio Low Pass Filter:

The Audio Low Pass Filter passes low frequencies of an AudioSource or all sound reaching an AudioListener while removing frequencies higher than the Cutoff Frequency

- ## Audio High Pass Filter:

The Audio High Pass Filter passes high frequencies of an AudioSource and cuts off signals with frequencies lower than the Cutoff Frequency.

- ## Audio Echo Filter

The Audio Echo Filter repeats a sound after a given Delay, attenuating the repetitions based on the Decay Ratio.

- ## Audio Distortion Filter

The Audio Distortion Filter distorts the sound from an AudioSource or sounds reaching the AudioListener.

- ## Audio Reverb Filter

The Audio Reverb Filter takes an Audio Clip and distorts it to create a custom reverb effect.

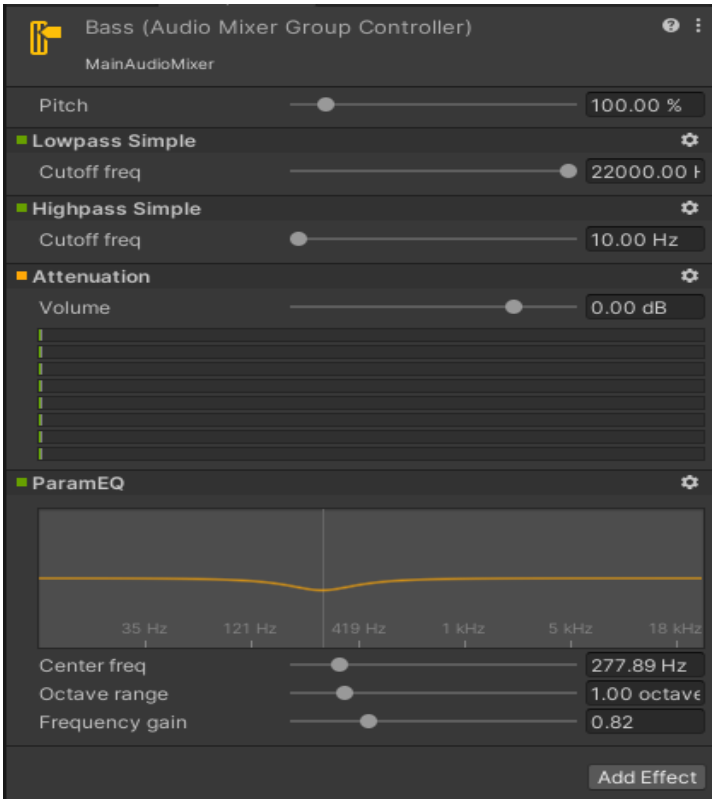
- ## Audio Chorus Filter

The Audio Chorus Filter takes an Audio Clip and processes it creating a chorus effect.

4. Audio Mixers:

The Unity Audio Mixer allows you to mix various audio sources, apply effects to them, and perform mastering.

The window displays the Audio Mixer which is basically a tree of Audio Mixer Groups. An Audio Mixer group is essentially a mix of audio, a signal chain which allows you to apply volume attenuation and pitch correction; it allows you to insert effects that process the audio signal and change the parameters of the effects. There is also a send and return mechanism to pass the results from one bus to another.



- The Asset - Containing all AudioGroups and AudioSnapshots as sub-assets.
- The Hierarchy view - This contains the entire mixing hierarchy of AudioGroups within the AudioMixer.
- The Mixer Views - This is a list of cached visibility settings of the mixer. Each view only shows a sub-set of the entire hierarchy in the main mixer window.
- Snapshots - This is a list of all the AudioSnapshots within the AudioMixer Asset. Snapshots capture the state of all the parameter settings within an AudioMixer, and can be transitioned between at runtime.
- The Output AudioMixer - AudioMixers can be routed into AudioGroups of other AudioMixers. This property field allows one to define the output AudioGroup to route this AudioMixer signal into.
- AudioGroup Strip View - This shows an overview of an AudioGroup, including the current VU levels, attenuation (volume) settings, Mute, Solo and Effect Bypass settings and the list of DSP effects within the AudioGroup.
- Edit In Play Mode - This is a toggle that allows you to edit the AudioMixer during play mode, or prevent edits and allow the game runtime to control the state of the AudioMixer.
- Exposed Parameters - This shows a list of Exposed Parameters (any parameter within an AudioMixer can be exposed to script via a string name) and corresponding string names.

Mixer Manipulation:

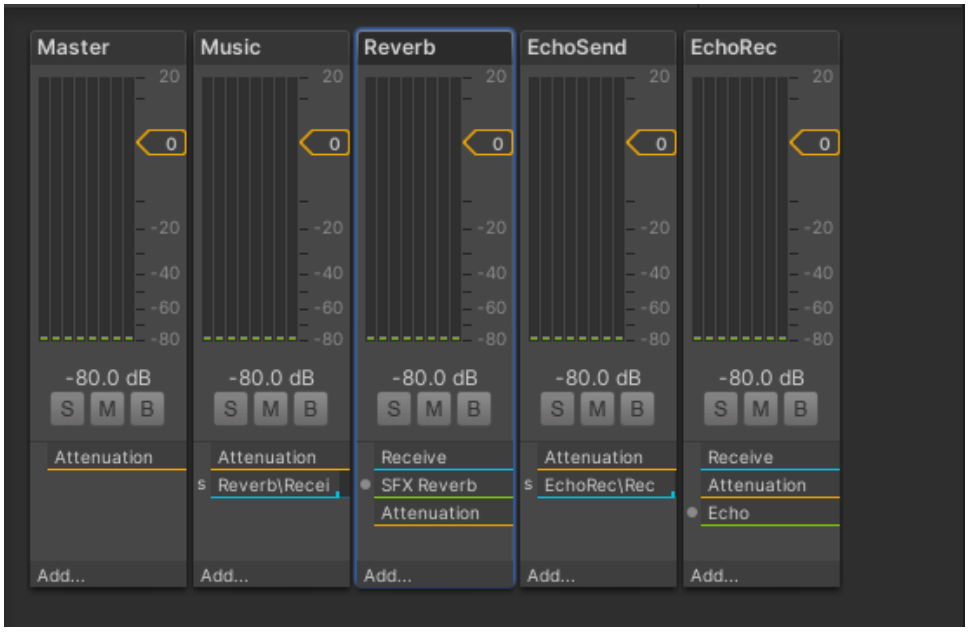
Below image shows an example of Send/Receive of audio signal and processing the output to the audio listener.

- Send - Sends allow you to diverge the audio signal flow and send a potentially attenuated copy of the signal to be used as a side-chain within another Effect Unit, for example, a side-chain compressor. You can insert Sends anywhere in the signal chain, allowing divergence of signal at any point. Initially, when Sends are added to an AudioGroup, they do not send to anything, and the Send Level is set to 80dB. To send to another Effect Unit, you must already have an Effect

Unit that can accept side-chain signals in the AudioMixer somewhere. Once the destination Effect Unit has been selected, the user needs to increase the Send Level to send signal to the destination.

1. To add a Send to an AudioGroup, click the ‘Add Effect’ button at the bottom of the AudioGroup Inspector and choose ‘Send’.
2. To connect a Send to another Effect Unit (capable of receiving signal), choose the destination from the drop-down menu in the Send Unit Inspector.
3. Set the level of signal sent to the destination with “Send Level”.

- Receive - Receives are the signal sinks of Sends, they simply take the audio signal that is sent to them from Sends and mix it with the current signal passing through their AudioGroup. There are no parameters to a Receive.



Sending normal sound from Music Group, receiving Reverb effect from Reverb Mixer Group

Processing reverb effect with the signal and routing it to audio listener with result output.

Script Reference:

- Simple Audio Manipulate:

1. BG Sound Audio Clip – This is the background music audio clip dragged to the slot.
2. Button Click Audio Clip – This is the button click sound audio clip.
3. Random Sounds – Array of audio Clips to played randomly dragged to each element from the project folder.
4. Bg Audio Source/SFX Audio Source/Random Audio Source – Used for background sound, Sfx Sound and Random sound for playing audio respectively.
5. Bg Music Toggle/SFX Music Toggle/Random Music Toggle – For toggling Music ON/OFF respectively.
6. Bg Music Vol_Slider/ Sfx Music Vol_Slider – For increasing and decreasing sound volume respectively.

```
[SerializeField] private AudioClip bgSoundAudioClip;
[SerializeField] private AudioClip ButtonClickAudioClip;
[SerializeField] private AudioClip[] randomSounds;
[SerializeField] private AudioSource bgaudioSource;
[SerializeField] private AudioSource sFXaudioSource;
[SerializeField] private AudioSource randomAudioSource;
[SerializeField] private Toggle bgMusicToggle;
[SerializeField] private Toggle sFXMusicToggle;
[SerializeField] private Toggle RandomMusicToggle;
[SerializeField] private Button randomMusicButton;
[SerializeField] private Slider bgMusicVol_Slider;
[SerializeField] private Slider sfxMusicVol_Slider;
```

Declaring all necessary variables to be filled in the editor

```
1 reference
private void BackgroundSoundTogggle(bool t_Bool)
{
    if (bgMusicVol_Slider.value != 0)
    {
        if (t_Bool)
            bgaudioSource.volume = 1;
        else bgaudioSource.volume = 0;
    }
    PlayOneShot(ButtonClickAudioClip, sFXaudioSource);
}
1 reference
private void SFXSoundTogggle(bool t_Bool)
{
    if (sfxMusicVol_Slider.value != 0)
    {
        if (t_Bool)
        {
            PlayOneShot(ButtonClickAudioClip, sFXaudioSource);
            sFXaudioSource.volume = 1;
        }
    }
    else sFXaudioSource.volume = 0;
}
1 reference
private void RandomSoundToggle(bool t_Bool)
{
    if (t_Bool)
    {
        PlayOneShot(ButtonClickAudioClip, sFXaudioSource);
        randomAudioSource.volume = 1;
    }
    else randomAudioSource.volume = 0;
}
```

Toggling Sound Methods

```
1 reference
private void ChangeRandomSounds()
{
    PlayOneShot(ButtonClickAudioClip, sFXaudioSource);
    randomAudioSource.Stop();
    randomAudioSource.clip = randomSounds[Random.Range(0,randomSounds.Length)];
    randomAudioSource.Play();
}
4 references
private void PlayOneShot(AudioClip t_Clip,AudioSource t_AudioSource)
{
    t_AudioSource.PlayOneShot(t_Clip);
}
1 reference
private void PlayInLoop(AudioClip t_Clip, AudioSource t_AudioSource)
{
    t_AudioSource.loop = true;
    t_AudioSource.clip = t_Clip;
    t_AudioSource.Play();
}
```

Changing Random Audio Clips, Playing Audio in Shot, And Playing Audio In loop.

```
Unity Message | 0 references
void Start()
{
    PlayInLoop(bgSoundAudioClip, bgaudioSource);
    bgMusicToggle.onValueChanged.AddListener(BackgroundSoundTogggle);
    sFXMusicToggle.onValueChanged.AddListener(SFXSoundTogggle);
    RandomMusicToggle.onValueChanged.AddListener(RandomSoundToggle);
    randomMusicButton.onClick.AddListener(ChangeRandomSounds);
    bgMusicVol_Slider.onValueChanged.AddListener(ControlVolBgMusic);
    sfxMusicVol_Slider.onValueChanged.AddListener(ControlVolSFXMusic);
    SetupUI();
}
1 reference
private void SetupUI()
{
    bgMusicVol_Slider.value = 1;
    sfxMusicVol_Slider.value = 1;
}
1 reference
private void ControlVolBgMusic(float t_Value)
{
   if(bgMusicToggle.isOn)
    bgaudioSource.volume = t_Value;
}
1 reference
private void ControlVolSFXMusic(float t_Value)
{
    if (sFXMusicToggle.isOn)
        sFXaudioSource.volume = t_Value;
}
```

Calling necessary methods in Start (), SetupUI for initial UI setup & controlling sound Volume by slider.

- Mixer Manipulation Code reference:

```
private void SetUpAudioPlayerAsModeSelected(AudioMixerGroup _audioMixerGroup, MusicType type)
{
    switch (type)
    {
        case MusicType.Normal:
            {
                SetUpAudioPlayeForNomal();
                break;
            }
        case MusicType.Surround:
            {
                SetUpAudioForSurround();
                break;
            }
    }
    audioSource.outputAudioMixerGroup = _audioMixerGroup;
}
```

Changing audio mixer from code.

```
private void SetUpAudioForSurround()
{
    isSurroundOn = true;
    audioSource.spatialize = true;
    audioSource.spatializePostEffects = true;
    audioSource.bypassReverbZones = false;
    audioSource.dopplerLevel = 1;
    audioSource.minDistance = 15;
    reverbZone.enabled = true;
    resonanceAudio.bypassRoomEffects = false;
    animator.SetBool("SurroundOn", true);
}
```

Another setup for surround sound effect

```
audioSource.spatialize = false;
audioSource.spatializePostEffects = false;
audioSource.bypassReverbZones = false;
resonanceAudio.bypassRoomEffects = true;
audioSource.dopplerLevel = 0;
audioSource.minDistance = 1;
reverbZone.enabled = false;
animator.SetBool("SurroundOn", false);
```

Another setup for Normal sound effect

```
Unity Script | 0 references
public class EngineControl : MonoBehaviour
{
    //getting the audio mixer
    public AudioMixer mixer;
    private float speed = 20.0f;

    Unity Message | 0 references
    void Update()
    {
        float acceleration = 100.0f * Time.deltaTime;
        //manipulating speed
        if (Input.GetMouseButton(0))
            speed += acceleration;
        else
            speed = Mathf.Max(14.0f, speed * Mathf.Pow(0.4f, Time.deltaTime));
        //controlling exposed properties of audio mixers from code
        mixer.SetFloat("PistonRate", speed);
        mixer.SetFloat("PistonSpeed", 0.6f + 0.002f * speed);
        mixer.SetFloat("PistonRandomOffset", 0.015f / (1.0f + 0.003f * speed));
        mixer.SetFloat("PistonLength", 0.05f / (1.0f + 0.003f * speed));
        mixer.SetFloat("PistonDistortion", 0.7f - 0.0002f * speed);
    }
```

Effect or filter variables which Is exposed from editor, controlling from code.

```
1 reference
public void SelectSongByName(string t_song)
{
    for (int i = 0; i < audioClips.Count; i++)
    {
        if (t_song.Equals(audioClips[i].name))
        {
            playMine(audioClips[i]);
            break;
        }
    }
}
1 reference
public void playMine(AudioClip t_Clip)
{
    Play_Button.SetActive(false);
    Pause_Button.SetActive(true);
    audioSource.Stop();
    audioSource.clip = t_Clip;
    playTime = (int)audioSource.time;
    audioSource.Play();
}
```

Changing and Playing Audio Clips from button Click

How to show frequency of audio listener or Spectrum Data:

```
/// <summary>
/// Mini "engine" for analyzing spectrum data
/// Feel free to get fancy in here for more accurate visualizations!
/// </summary>
Unity Script | 1 reference
public class AudioSpectrum : MonoBehaviour {

    Unity Message | 0 references
    private void Update()
    {
        // get the data
        AudioListener.GetSpectrumData(m_audioSpectrum, 0, FFTWindow.Hamming);

        // assign spectrum value
        // this "engine" focuses on the simplicity of other classes only..
        // ..needing to retrieve one value (spectrumValue)
        if (m_audioSpectrum != null && m_audioSpectrum.Length > 0)
        {
            spectrumValue = m_audioSpectrum[0] * 300;
        }
    }

    Unity Message | 0 references
    private void Start()
    {
        /// initialize buffer
        m_audioSpectrum = new float[128];
    }

    // This value served to AudioSyncer for beat extraction
    2 references
    public static float spectrumValue {get; private set;}

    // Unity fills this up for us
    private float[] m_audioSpectrum;
```

THE END