AUDIO MANAGER (singleton class) meaning 🡪 only one will exist in a scene and all our scripts can access it!

// don’t forget to set the attenuation object in the audiolisterner probably the player if not FP

using System.collections;

using System.Collections.Generic;

using UnityEngine;

using FMODUnity; // for UI & Runtimemanager (= fmod components ?)

using FMOD.Studio; // for the event instances

public class AudioManager : MonoBehaviour

{

[Header (“Volume”)]

[Range(0, 1)]

Public float masterVolume = 1;

[Range(0, 1)]

Public float musicVolume = 1;

[Range(0, 1)]

Public float SFXVolume = 1;

[Range(0, 1)]

Public float ambienceVolume = 1;

[Range(0, 1)]

// these are the busses you need to connect to

Private Bus masterBus;

Private Bus musicBus;

Private Bus SFXBus;

Private Bus AmbienceBus;

Private List<EventInstance> eventInstances;

Private List<StudioEventEmitter> eventEmitters;

Private EventInstance ambianceEventInstance;

Private EventInstance musicEventInstance;

// 🡪 sets up the class as a singleton where access it via the instance variable

public static AudioManager instance { get; private set;}

// 🡪 this is an auto property ( a way variables can be declared . values can be accessed and changed from an external class

private void Awake() //this is an awake method

{

if (instance !=null)

{

Debug.LogError (“Found more than one Audio Manager in the scene.”);

}

instance = this;

eventInstances = new List<EventInstance>();

eventEmitters = new List<EventEmitter>();

masterBus = RuntimeManager.GetBus(“bus:/);

musicBus = RuntimeManager.GetBus (“bus:/music”);

SFXBus = RuntimeManager.GetBus(“bus: /SFX”);

AmbienceBus =RuntimeManager.GetBus(“bus:/Ambience”);

}

Private void start()

{

InitializeAmbiance(FMODEvents.instance.ambianceSound);

InitializeMusic(FMODEvents.instance.musicSound);

}

Private void Update()

{

masterBus.setVolume(masterVolume);

musicBus.setVolume(musicVolume);

SFXBus.setVolume(SFXVolume);

ambienceBus.setVolume(ambienceVolume);

}

Private void InitializeMusic(EventReference musicEventReference)

{

musicEventInstance = CreateInstance(musicEventReference);

musicEventInstance.start();

}

Private void InitializeAmbiance(EventReference ambienceEventReference)

{

ambienceEventInstance = CreateInstance(ambienceEventReference);

ambianceEventInstance.start();

}

Public void SetAmbienceParameter(string parameterName, float parameterValue)

{

ambienceEventInstance.setParameterByName(parameterName, parameterValue);

}

Public void SetMusicArea(MusicArea area)

{

musicEventInstance.setParameterByName(“area”, (float) area) //”area” is the exact name of the parameter in FMod

}

// new public method for OneShot

public void PlayOneShot (EventReference sound, Vector3 worldPos)

{

RuntimeManager.PlayOneshot (sound, worldPos);

}

// new public method that returns an event instance (play, stop, param control // variable name

Public EventInstance CreateEventInstance (EventReference eventReference) // declaring the eventInstance

{

EventInstance eventInstance = RuntimeManager.CreateInstance (eventReference);

eventInstances.Add(eventInstance);

Return eventInstance;

}

Public StudioEventEmitter InitializeEventEmitter( EventReference eventReference, GameObject emitterGameObject )

{

StudioEventEmitter emitter = emitterGameObject.GetComponent<StudioEventEmitter>();

Emitter.EventReference = eventReference;

eventEmitters.Add(emitter);

return emitter;

}

Private void CleanUp()

{

//stop and release any created instances // foreach is a loop

foreach (EventInstance eventInstance in eventInstances)

{

eventInstance.stop(FMOD.Studio.STOP\_MODE.IMMEDIATE);

eventInstance.release();

}

//stop all of the even emitters,

foreach (StudioEventEmitter emitter in eventEmitters)

{

emitter.stop();

}

}

// whenever the AudioManager is Destroyed all the instances are cleaned up

Private void OnDestroy()

{

CleanUp();

}

}