FMOD & UNITY / SCOTT

**Part 1 ONE SHOT & ANIMATION**

*Adding basic sound*

FMODUNITY. RuntimeManager. PlayOneShot (“eventpath”, GetComponent <transform>(),position);

* Good for ‘stable’ and simple events (hurt, impact…)

*Adding Sound to Animation*

Seek the animation clip. But you need the model that will be animated.

Most likely go to PREFAB and drag it inside inspector of animation clip.

Open animation tab in project view and add ‘event’ on timeline/frames

Function PlaySound

Float

Int

String Event path Fmod

Object

Create script with one shot 🡪

Go to player character, because it has the ANIMATOR COMPONENT attached and choose a script or add one

FUNCTION

**FMODPlayerScript**

Using UnityEngine;

String variable ‘path’ represent event in FMOD

public class *FmodPlayerScript* : MonoBehavior {

void PlaySound (string path)

{

FMODUNITY.RuntimeManager.PlayOneShot(path, GetComponent<transform>(),position);

}

}

Make sure this is on the **PREFAB** (for character and audiolistener) if there are different levels. Because Unity will destroy all of the assets for a new scene.

**Part 2 DYNAMIC GUN SHOT**

3D sounds

* Gun sound in the middle
  + Create One shot to gunshot event

FMODUnity. RuntimeManager.PlayOneShot (“event”, GetComponent<transform>().position);

To find the event path go to FMod tab in Unity 🡪 EventBrowser + copy

* Bullet 3D
  + Add a one shot to the bullet prefab
  + Add to the OnEnable function
    - Cfr. clone instances

*ISSUE*

*Same script for different objects*

* Create an Event reference in the Unity UI and not an event path on the script itself
* Using a string variable / declare it before the function/void
  + [FMODUnity.EventRef]

Public string NAME

FMODUnity. RuntimeManager.PlayOneShot (NAME, GetComponent<transform>().position);

**Part 3 FOOTSEPS & RAYCAST**

Event in FMOD 🡪 multi instrument

Create PARAMETER tab

* Volume automation in line with PARAM n°
* Choose Initial Value 🡪 right click dial

IF you have multiple PARAM maybe better to make a PRAM tab for each one and use them as an ON/OFF switch cfr. 1/0 for each tab param….

UNITY

Add code to script on PlayerCharachter Game Object

1. Create float variables
   1. 🡪 to control raycast distance (cfr, jump, not on ground)
   2. To control parameter switch and n°

public class *FmodScriptPlayer* : MonoBehaviour {

private float *distance = 0.1f*;

private float *material*;

Calling Functions

void FixedUpdate()

{

MaterialCheck();

Debug.DrawRay(transform.position, vector2.down\*distance, Color.Blue);

}

Another variable named “hit”. All info it collects is stored in this variable.

Void MaterialCheck()

Created in Unity, Edit; project settings, layers and tags

{

RaycastHit2D hit;

hit = Physics2D.Raycast(transform.position, vector2D.down, distance, 1 << 31);

// you want to access this variable (“hit”) and check it for the tags

if (hit.collider) 🡪 necessary or you get an ERROR (because if you jump no collide with game object)

{ if (hit.collider.tag == “Material: Earth”)

Layer Mask; pitch shit in binary

It’s the layer of the platform

Communicate this with Unity through an integer

Material = 1f;

else if (hit.collider.tag == “Material: Stone”)

Material = 2f;

else

Material = 1f;

}

// TAGS & LAYERS Both are used to **classify objects** of a scene, but Unity does use **layers in a functional way** where **tags are meant to be used by scripts only**. Functional how? For instance, layers can be used to specify a set (category) of objects to be rendered, layers can also be used to ignore collisions or raycasts and more. There is some overlap of usage between them, you can certainly get away with using layers only, but if you want to further categorize your game objects you can assign tags to them.What is the difference between layers and tags? For multiple objects in the same layer, you could use tags to **differentiate** between them while **deciding what to do in the result of a collision**, for example. Different from layers, **tags offer some helper methods** like FindGameObjectsWithTag which can be pretty useful to do initialization

To play sound we’ll attach events to the animation.

See PART 1 for details

But we don’t use One Shot, because we change a **PARAM** and you can’t do that with in the Event

* SO…..We need to create an Instance and tell it to find/set the PARAM VALUE 🡪 cfr private float variable in the beginning ‘Material’… and then tell it to START and STOP

Void *PlayFootstepsEvent* (string path)

Created an Event Instance where you can store your event into

{

FMOD.Studio.EventInstance *Footsteps*  = FMODUnity.RuntimeManager. CreateInstance(path);

Footsteps.setParameterValue (“Material”, Material);

Footsteps.start();

This is the float variable that controles that parameter; cfr material check

Footsteps.release();

}

If this changes then that changes

Exact name of the parameter in FMOD

}

**Part 4 PARAMETER & VELOCITY**

Sound depending on distances, intensity,…

Cfr**. Falling** from different heights

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Getting back to previous part…

* Removed the **string** in the animation event (= the path to FMOD event)
* So, not passing the event path in our script
* Only function is mentioned (cfr. “PlayEventName”)
* Why???

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

In FMOD

Landing sounds in timeline

* Hard and Soft landing tracks
* PARAM “Velocity”
  + Automate the volume between the two tracks depending on value of the param
  + - 0,60 to 0 ( 🡪 falling at slower speed 🡪 softer landing)

**C# FmodPlayerScript in Unity**

public class *FmodPlayerScript* : MonoBehaviour

Changing the way we play our footsteps

{

private float distance = 0.05f;

private float Material;

private bool wasGrounded = true;

private LayerMask LM = 1<<31;

private float Height;

Way to measure

private float old\_Height;

private float Height\_Difference;

FMOD.Studio.EventInstance Footsteps;

Variable that stores the event instance of our footstep sounds

Using the start function to store the event path of footsteps sounds into the footsteps variable… so not stored in the animation…? So you can use PARAM differently… which you release ondestroy

void Start ()

{

Footsteps = FMODUinty. RuntimeManager. CreateInstance (“EventNameFootsteps”);

}

stored in the animation

void PlayMeleeEvent (string path)

{

FMODUnity.RuntimeManager. PlayOneShot (path, transform.position);

}

Void FixedUpdate ()

{

MaterialCheck();

Debug.DrawRay (transform.position, vectro2D.down \* distance, Color.blue);

PlayerLanded();

Calling these functions

wasGrounded = IsGrounded();

IsGrounded();

PlayerFallingCheck();

Footsteps.setParameterValue (“Material”, Material);

}

Void MaterialCheck()

{

RaycastHit2D hit;

hit = Physics2D.Raycast(transform.position, vector2D.down, distance, LM);

if (hit.collider)

{ if (hit.collider.tag == “Material: Earth”)

Material = 1f;

else if (hit.collider.tag == “Material: Stone”)

Material = 2f;

else

Material = 1f;

}

Void PlayFootstepsEvent()

{

Footsteps.start();

2 Boolians // ! = false -🡪 checking to see if the player was touching the ground on the last frame🡪 if not, we know they just landed.

Check if player is currently touching ground

{

Void OnDestroy ()

{

Footstepts.release();

}

Void PlayerLanded ()

{

If (IsGrounded() && !wasGrounded)

{

FMOD.Studio.EventInstance Landing = FMODUnity. RuntimeManager. CreateInstance (“EventNameLand”);

Landing. setParameterValue (“Velocity”, Height\_Difference);

Footsteps. setParameterValue (“Footstepduck”, Height\_Difference);

Landing.start ();

Landing.release();

//Debug.log(Height\_Difference);

}

}

Bool IsGrounded ()

This is a Method; it can return a value; here you want to return a Boolean value. You want to know if something is true or false, using another raycast

{

return Physics2D.Raycast (transform.position, vector2D.down, distance, LM);

}

Void PlayerFallingCheck()

{

old\_Height = Height;

Height = transform.position.y;

Here also Checking the variable Last frame!!!!

Height\_Difference = Height – old\_Height;

If (Height\_Difference > 0)

Height\_Difference = Height\_Difference \* -1;

Mistake!

}

}

IN FMOD

* Click on the parameter name 🡪 VELOCITY : set a specific amount of time to return to the initial value
* “set as initial value”
* 1 unit per second

**Part 5 FEEDBACK**

Void OnDestroy Function

🡪

Also remember .release command comes with the FMOD API allows you to stop an event Instance or destroy it when you’re finished with it when it’s not playing , which is good because we don’t want to create to many instances 🡪 can affect our games’ performance

We create an instance in the void start function cfr*. FMODUnity.RuntimeManager.CreateInstance (“Name”);*

On the FIRST FRAME of EVERY SCENE 🡪 piling up a lot of instances we won’t be using

* That’s why 🡪 on destroy function (which is Unity specific) 🡪 runs on the very last frame before a game object is destroyed when we enter a new scene or level and re- introduced, depending on the transform of new level

Landing sound triggers TWICE

Use function or variable in character script

Protected variable 🡪 determines which classes can use this variable

Public (every script or class)

Private (only within class)

To find out variables use a Debug.Log (the variable you want);

In the fixed update

PLAYERCHARACTER SCRIPT

protected Vector2 m\_MoveVector;

* Make it public so FMODPLAYER script can use it
* Vector2 means two coordinates X and Y
  + Here we’re only interested in the Y
* Check the values
  + Using Debug.Log(m\_MoveVector)
  + Put it in the fixed update function
  + To see the range of values
  + (0.0 , -2.3) 🡪 X , Y
    - X 🡪 - = move left / positive = move right
    - Y 🡪 - = falling / positive = jump

Change script and reference it from the player character script

Remove all the info to calculate how fast character was falling/ moving

If we want to access this new variable you need to add a new **namespace**

**🡪** C# programs are organized using namespaces. Namespaces are used both as an “internal” organization system for a program, and as an “external” organization system—a way of presenting program elements that are exposed to other programs.

* As projects become larger and the number of **scripts**  
   increases, the likelihood of having clashes between script class names grows ever greater. This is especially true when several programmers are working on different aspects of the game separately and will eventually combine their efforts in one project. For example, one programmer may be writing the code to control the main player character while another writes the equivalent code for the enemy. Both programmers may choose to call their main script class *Controller*, but this will cause a clash when their projects are combined.

To some extent, this problem can be avoided by **adopting a naming convention** or by renaming classes whenever a clash is discovered (eg, the classes above could be given names like *PlayerController* and *EnemyController*). However, this is troublesome when there are several classes with clashing names or when variables are declared using those names - each mention of the old class name must be replaced for the code to compile.

The C# language offers a feature called **namespaces** that solves this problem in a robust way. A namespace is simply a collection of classes that are referred to using a chosen prefix on the class name. See Microsoft’s documentation on [namespaces](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/namespaces/) for more information.

Namespace 2DGamekit 🡪 put this above your FMOD script

* Get component and reference the playercharacter script
* In the void fixed update() of the FMOD script
  + Debug.Log(pc.m\_MoveVector.y);
    - Within that script (cfr. “pc”) you want to call the m\_MoveVector variable
      * Which is made public so we can access it
      * And within that variable we want to access the ‘Y’ value
* Now we know how to reference the value we’re after

**SO** Void PlayerLanded ()

{

If (IsGrounded() && !wasGrounded && pc.m\_VectorMove.y < 0 )

{

FMOD.Studio.EventInstance Landing = FMODUnity. RuntimeManager. CreateInstance (“EventNameLand”);

Landing. setParameterValue (“Velocity”, ~~Height\_Difference~~);

Footsteps. setParameterValue (“Footstepduck~~”, Height\_Difference);~~

Landing.start ();

Landing.release();

//Debug.log(Height\_Difference);

}

}

**Edit the velocity param so it’s to a new scale/value**

**Part 6 PLAYBACK STATES**

Woosh sound when falling form big height

Detect when player is falling fast/big height

* Looping the woosh rush event because we don’t know how long the player will be falling….
* Adding PARAM ‘velocity’ 🡪 volume automation in line with falling values in Unity

Adjust our script to call this in UNITY

FMODplayer script is attached to player script

* Make a new event instance “Falling”
  + public Fmod.Studio.EventInstance Falling (is this the API call instead on FMODUnity.RuntimeManager….
* Also made a new Boolean
  + public bool playerSubmerged = false;
  + Because you don’t want to hear air when player is in water

Void Start ()

{

Footsteps = FmodUnity.RuntimeManager.CreateInstance (“eventpath”);

Falling = FmodUnity.RuntimeManager.CreateInstance (“eventpath”);

pc = getComponent <PlayrerCharacter>();

}

This function is being called every frame and creates a PLAYBACK STATE….Something we can do with the FMOD API to detect what our events (instances) are doing … Detect if they are playing / sustain point (= pause )…

Take the event instance we created for the falling event and assign it to the playback state we created

Add it ( falling) to the fixed update

And also release it on void Destroy

Void PlayerFalling()

{

Fmod.Studio.PLAYBACK\_STATE PbState;

Falling.GetplaybackState (out PbState);

if (!IsGrounded () && pc.m\_MoveVector.y < -3 && !playerSubmerged)

{

The variable that controls the PARAM

Falling.setParameterValue (“Velocity”, pc.m\_MoveVector.y);

**if (PbState ! = Fmod.studio.PLAYBACK\_STATE.PLAYING)**

**Falling.start()**

}

}

We want to check if the event is NOT PLAYING … we can do that with the PLAYBACK STATE

If it is not playing then PLAY

SO the event does keep being triggered start and stopped

**Part 7 ACID SPLASHING WAVES**