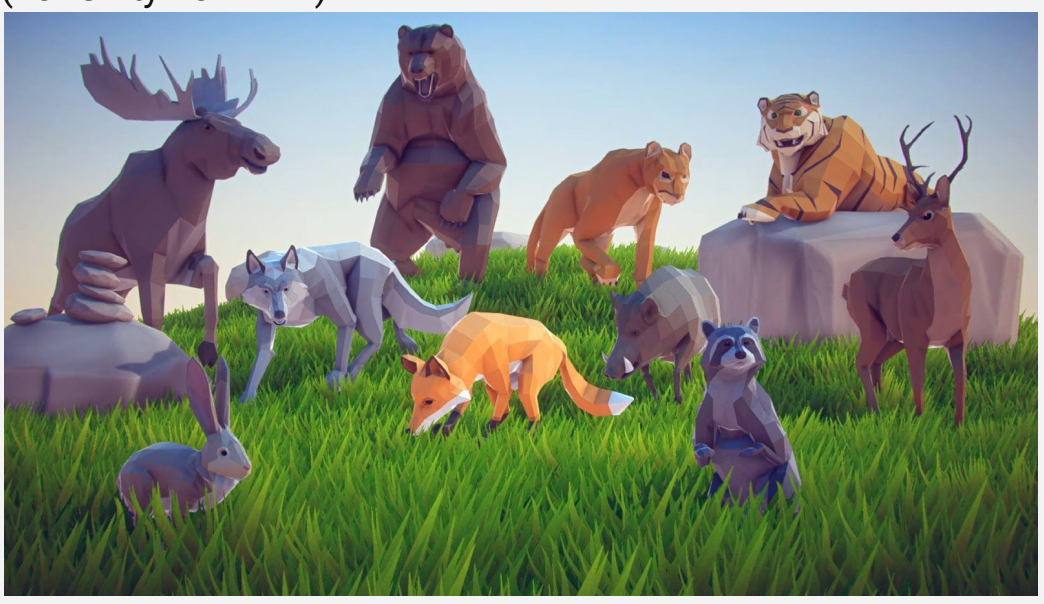
68% complete

Common Script for

MalberS Animations Unity Assets

Publisher: MalberS Animations

(For Unity 2017.1+)



INDEX

If you want, you can skip to [Frequently Asked Questions]

1. How can I know which actions to call using SetAction()?
2. My Animal/Creature just keep sliding and falling on the ground?
3. Why I get this error 'The type/namespace name *'CrossPlatformInput' does not exist.....'* while importing the asset?
4. How can I change the Blend Shapes using a UI Slider

Overview

Thanks for purchasing any of the Malbers Animations Asset.
All my assets have a tons of animations.
You can use to make your own controller if you don't want to use the Animal controller root-motion based I provide.

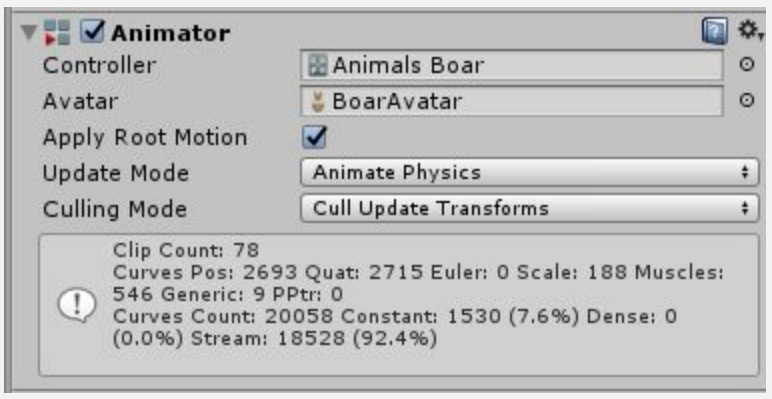
This Help will give you a better understanding of how the animal controller works.

Index

Main Principles

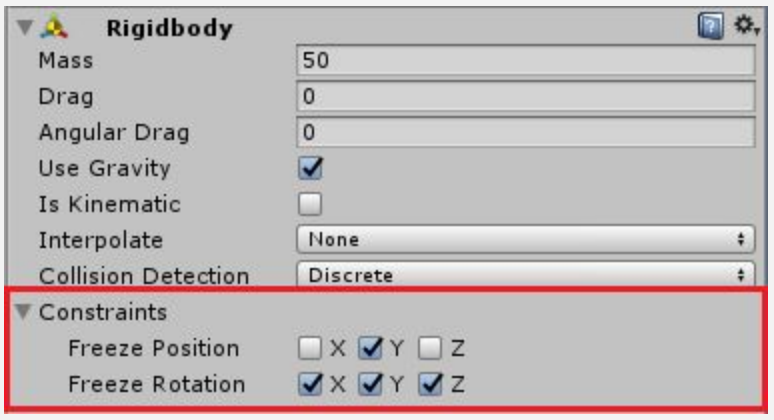
The Animal Controller needs an *Animator* and a *RigidBody* to work. This two components need to be at the same gameobject level than the **Animal** Script.

ANIMATOR
All animations are root motion based, so the *Animator* needs to have **applyRootMotion = true,** and because the we are using *RigidBody,* animation **Update Mode** must be setted to **Animate Physics.**



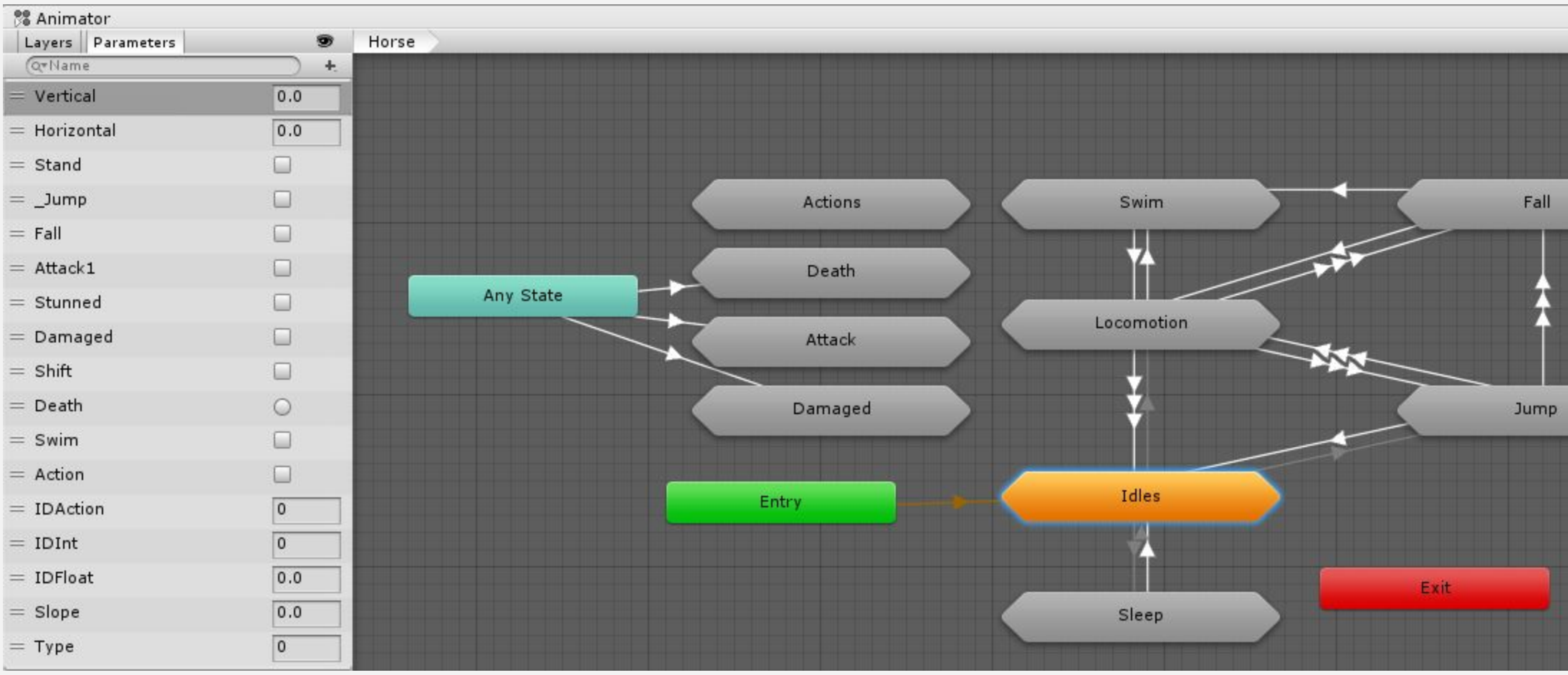
RIGID BODY

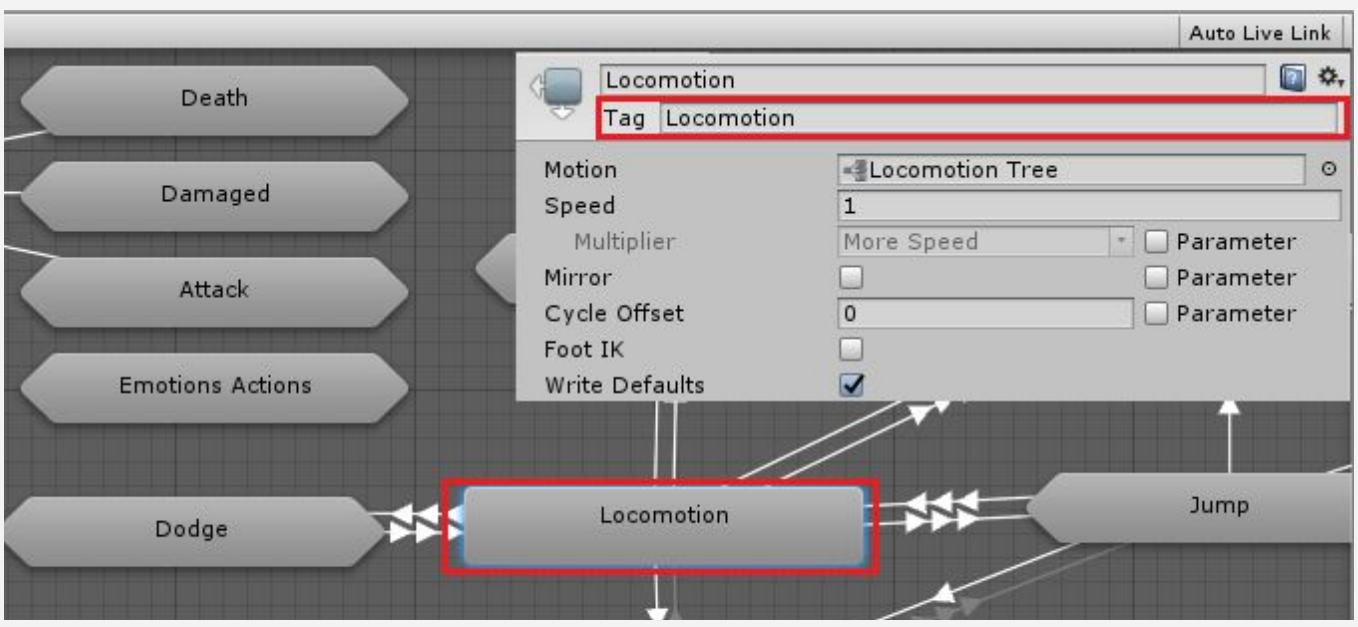
Since the *Colliders* for the animal are not on the same level as the rigid body, and these colliders are not touching the ground, we need to set the *RigidBody* **constraints** to :



This will avoid the animal to fall down, Also the rotations will be handled by the animations to we will freeze all the rotations.

The *Animator Controller* is the core for the Animal script.
Animation States are tagged with unique **tags** to find if check are on those main animation states.
Tags like (Locomotion, Jump, Attack, Recover, Idle, Action.. etc). those Tags are managed by the **Hash** script





(still filling it)

[*Index*](#)

Primary Scripts

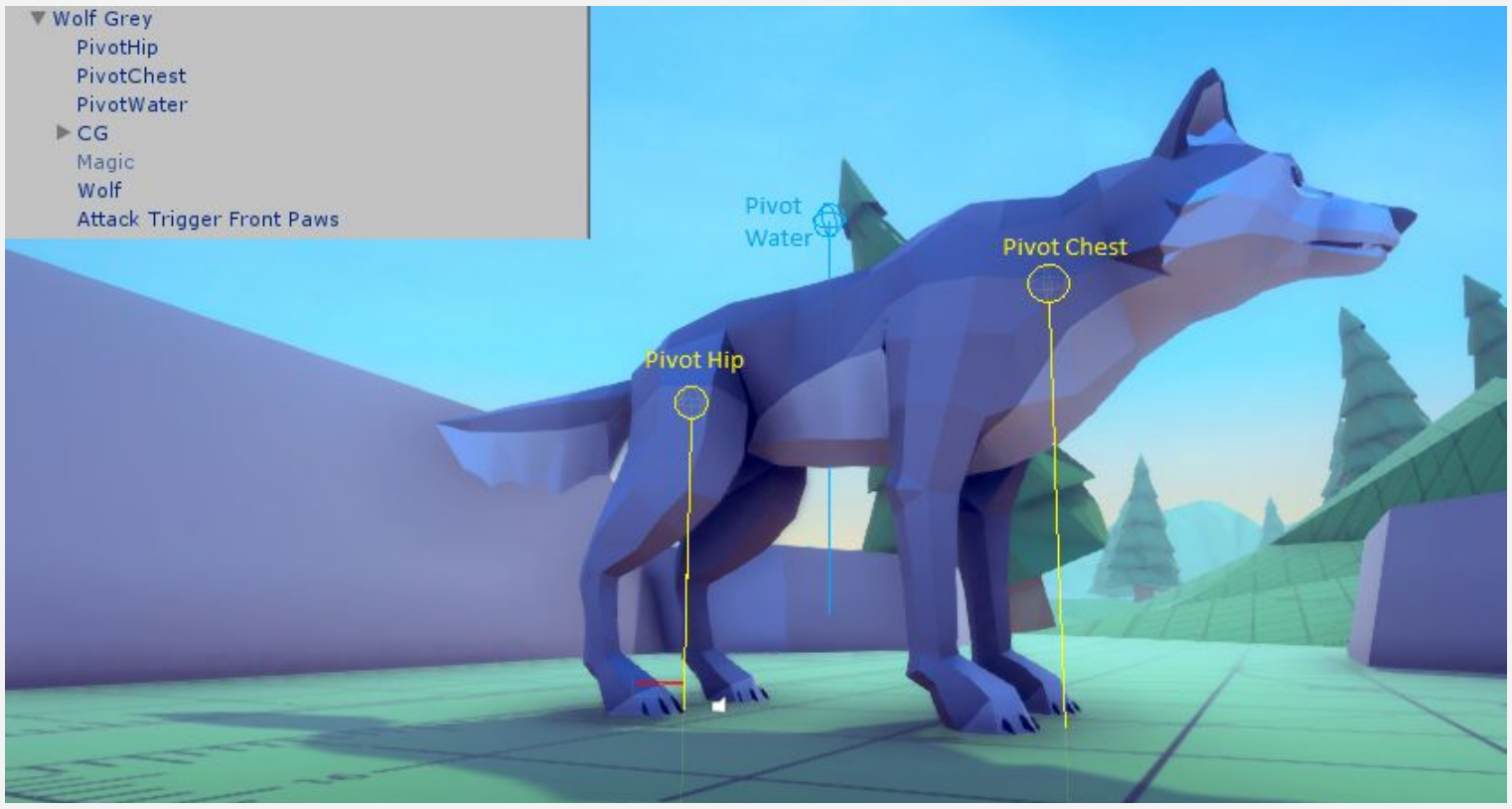
The Main Scripts are the core for the functioning of the creatures. The **Animal** Script is the main one but to have a better controller with complement it with the **Pivots** and the **Attack Triggers**.

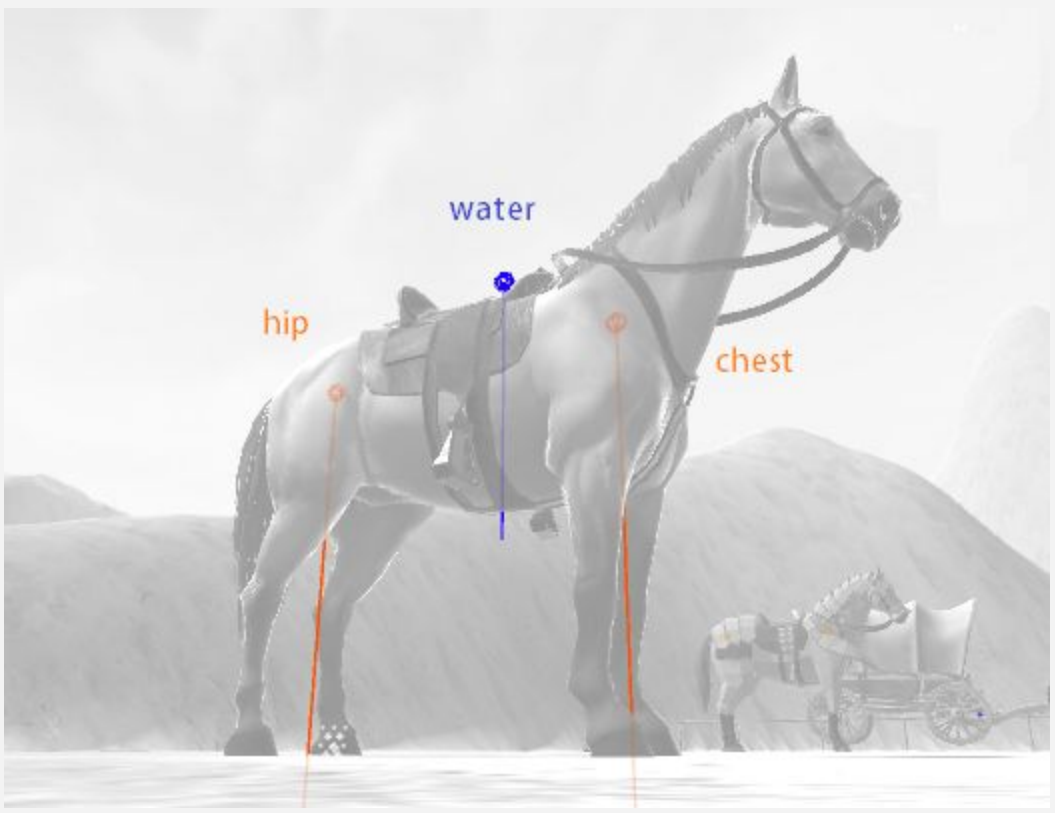
Index

Pivots

The pivots are identifier scripts to know where the to cast rays to calculate the alignment for the terrain(Chest and Hip Pivot) or just to find the water for the swimming and underwater logic(Water). Is used heavily for the ***Animal Script.***

All animal have 3 Pivots





The **Water Pivot** is responsible for activating the *Swim* ability. This transform is always casting a ray downwards to check if we find water (the Water game object needs to be set on the physical layer "*Water*"). If the water level is higher as the **Chest Pivot** Height (position.y), The swimming ability will be activated.

The **Hip** and **Chest** pivots are in responsible of aligning the animal to the ground. Works as a "fake IK".

The **Chest** pivot is also the base for the Fall Ray. When walking, trotting, running or jumping.. a front ray is cast to check if we are about to fall. This ray is combined with the ***Fall Ray Multiplier*** and ***Front Fall Ray*** on the Animal Inspector.

The length of each Pivots are very important, Short Rays will cause to always fall on big slopes, Long Ray will never fall on slopes and will execute the recover animation when falling too soon. You will need to find the right spot on the rays ;)

Tip 1:
If your animals do not need to swim you can remove the **Water Pivot** (it will skip all the ***Water logic***, increasing performance (No Water Raycast)).

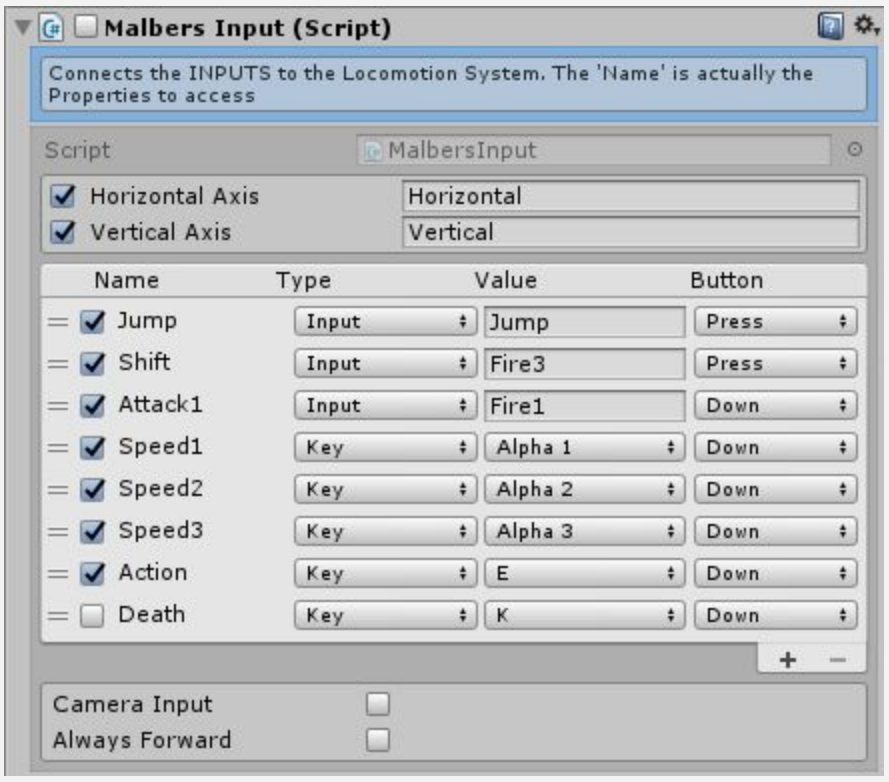
Tip 2:
If your animals will always walk on a flat terrain you can remove the **Hip** and **Chest Pivot** (it will skip all the ***Terrain Alignment logic,*** increasing performance (No Fix Position Raycast)).

Tip 3:
If your animal is two Legged (Raptor, Ostrich) you can remove the **Hip** and **Chest Pivot** (it will skip all the ***Terrain Alignment logic*** keeping the animal straight, no matter the terrain**,** increasing performance (No Fix Position Raycast)).

Index

Malbers Input

This is the Input source for all the animals and creatures. This will send the main actions to the Animal. When using the Animals for AI you don't need to add this script because the AI will handle the Inputs.



Tip 5:
When Using the *Riding System* from **HorseAnimsetPro**, this script should be Disabled, otherwise the animal will move as soon as you press any of the Inputs and we don't want to start moving when the rider is NOT mounted.

Parameters:

Horizontal/Vertical Axis:
This is the Input from the Unity Input System. Custom Advanced Third Person Controller(TPC) Uses their own Input values. You can easily change it here to match the inputs from those TPC.

Inputs:
This are the Inputs used to activate Abilities. They use the iMalbersInputs Interface to communicate with the Animal Script. If you are using a custom input system just change the type to Input and use the same Input Value on the Unity Input System.

Camera Input:
This will change from World Base Input to Camera Based Input (Meaning that if you press forward the animal will go towards the camera forward direction).

Always Forward:
The animal will always go forward, useful for Infinite Runners. This will always send to the `Animal.Move(Vector3 move)` with the `move.z = 1;`

Public Methods:

```
void EnableInput(string inputName, bool value)
```

Enable/disable the Inputs.
Example:

```
MalbersInput.EnableInput("Jump", false);
```

This will disable the Jump input and the animal will no longer be able to jump.
(An use for this method is for enable or disable the attacks of the animals when the rider has mounted the animal, and you want to use the attack inputs for the riders weapons, and not the animal attacks).

```
bool IsActive(string inputName)
```

Returns if an input is enable/disable.

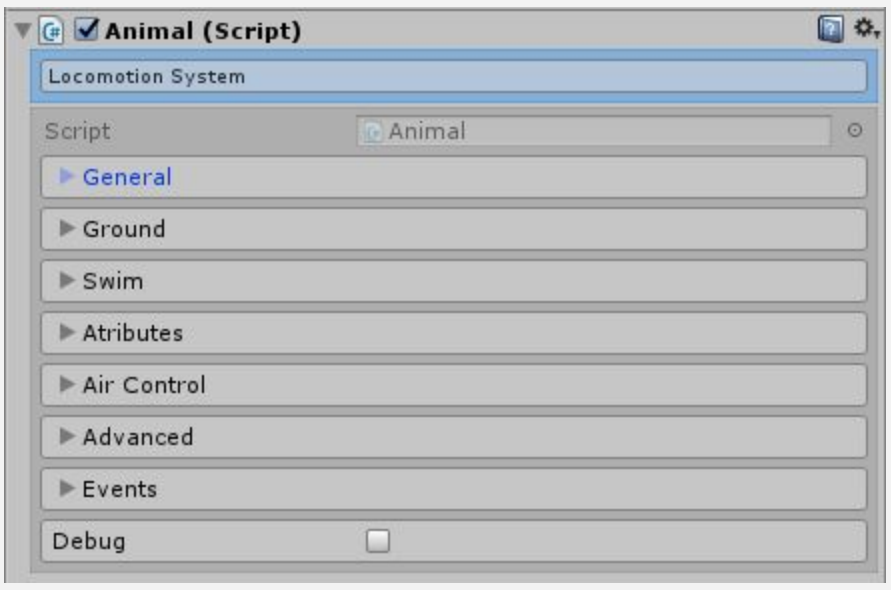
```
InputRow FindInput(string name)
```

Returns an Input by its name.

Index

Animal

The animal script is responsible to control all the movement logic for the animal to move and work properly. Manage all the Animator and the Rigid Body parameters.

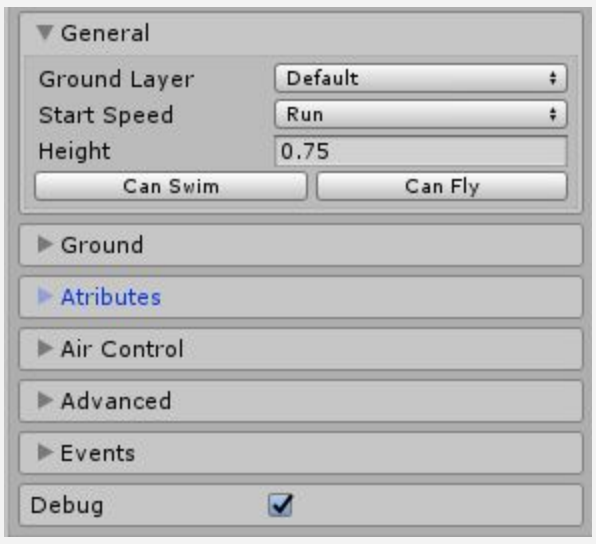


Parameters:

The script is grouped on 7 categories:

General
Ground
Swim
Fly
Attributes
Advanced
Events

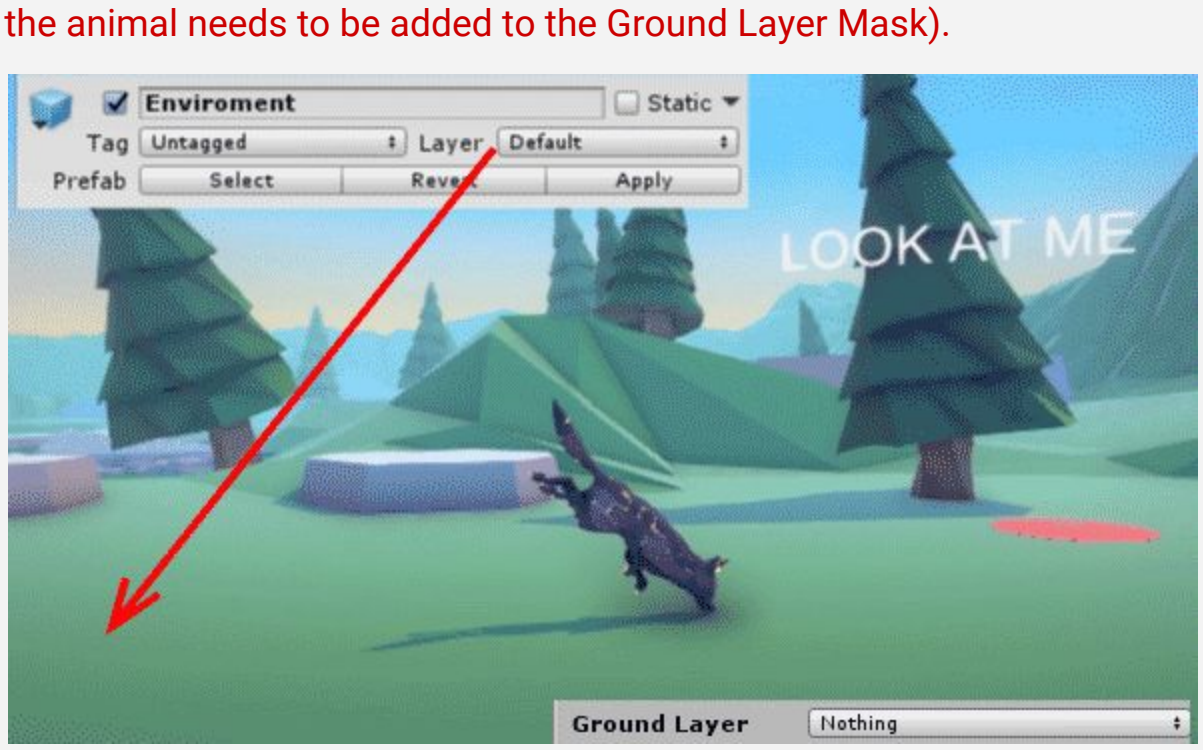
General



Ground Layer:

Layers the Animal will recognize as ground.

(If the animal start to fall for no reason, it should be because **Ground Layer** is empty, or the layer of the Game Objects beneath the animal needs to be added to the Ground Layer Mask).

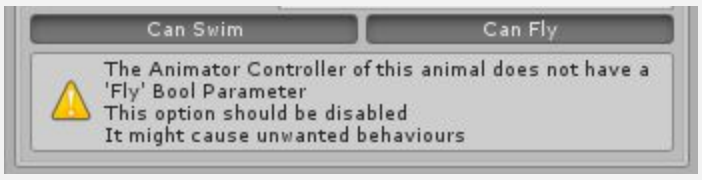


Start Speed:

Which speed the animals is starting at (Walk, Trot, Run).

Height:

The Height of the animal from the Center of Gravity Bone (CG) and the ground, (Lower Height can simulate moving on snow or on mud).



Can Swim:

Activate the Swim Logic and enables the Swim parameters.

If the Animator Controller of this animal does not have a **'Swim'** Bool Parameter. This option should be disabled. It might cause unwanted behaviours.

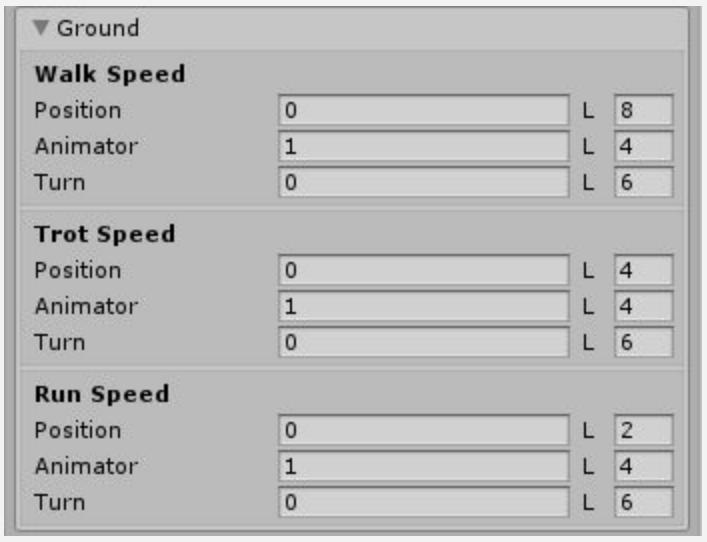
Can Fly:

Activate the Fly Logic and enables the Fly parameters

If the Animator Controller of this animal does not have a **'Fly'** Bool Parameter. This option should be disabled. It might cause unwanted behaviours

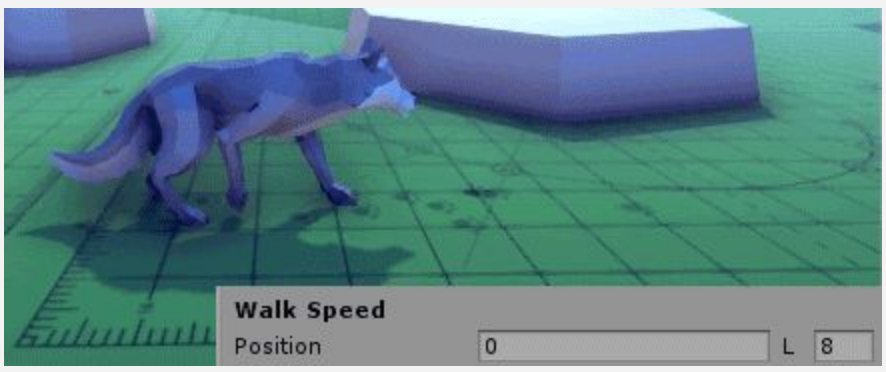
---

Ground



Walk, Trot and *Run* speeds are parameters from the ***Speed*** Class

Position:
Add more speed to the movement of the animal while Walking/Trotting/Running. This can cause sliding effect.

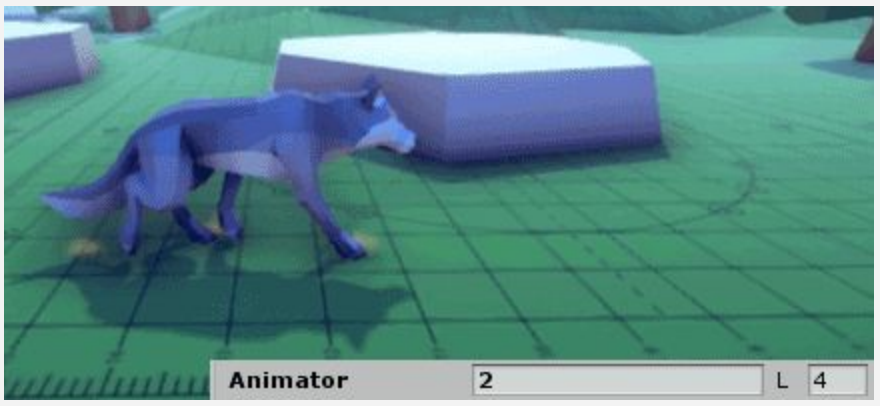


Left: Position = 0

Right: Position = 10

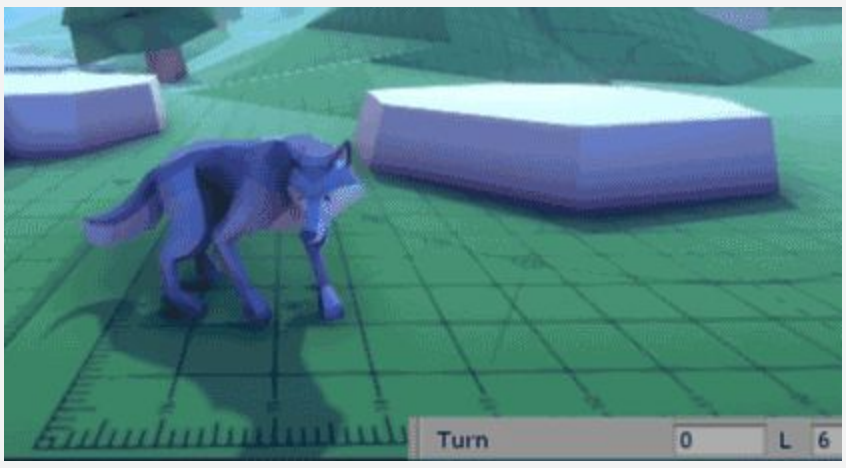
L (Lerp): Lerp value for changing between position speeds.
Higher values: more responsiveness

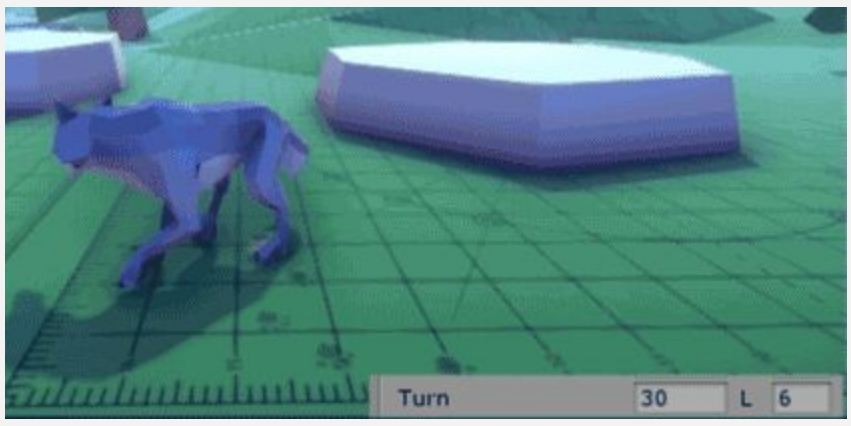
Animator:
Add more speed to the Animator while Walking/Trotting/Running. Default Value (1)



L (Lerp): Lerp value for changing between Animation Speeds.
Higher values: more responsiveness

Turn:
Add more speed to the rotation while Walking/Trotting/Running. Higher values will cause sliding

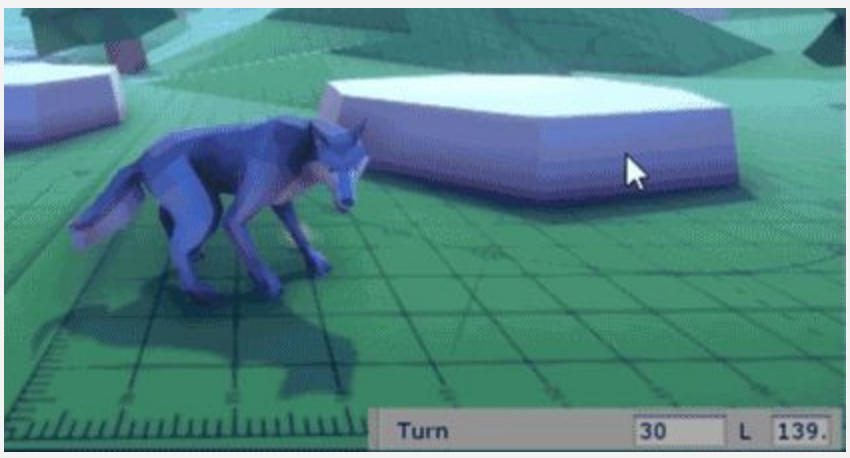




Left: Turn = 0

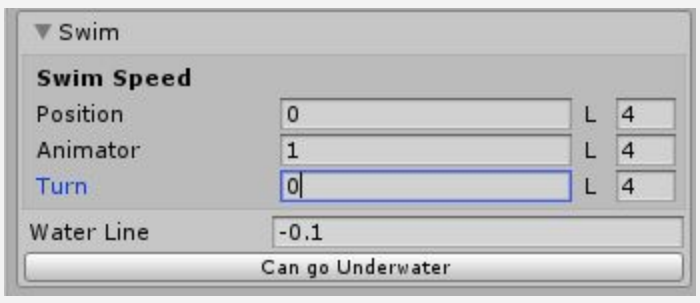
Right: Turn = 60

L (Lerp): Lerp value for changing between rotation Speeds.
Higher values: more responsiveness.



Swim

This will be enabled if **Can Swim** is active.

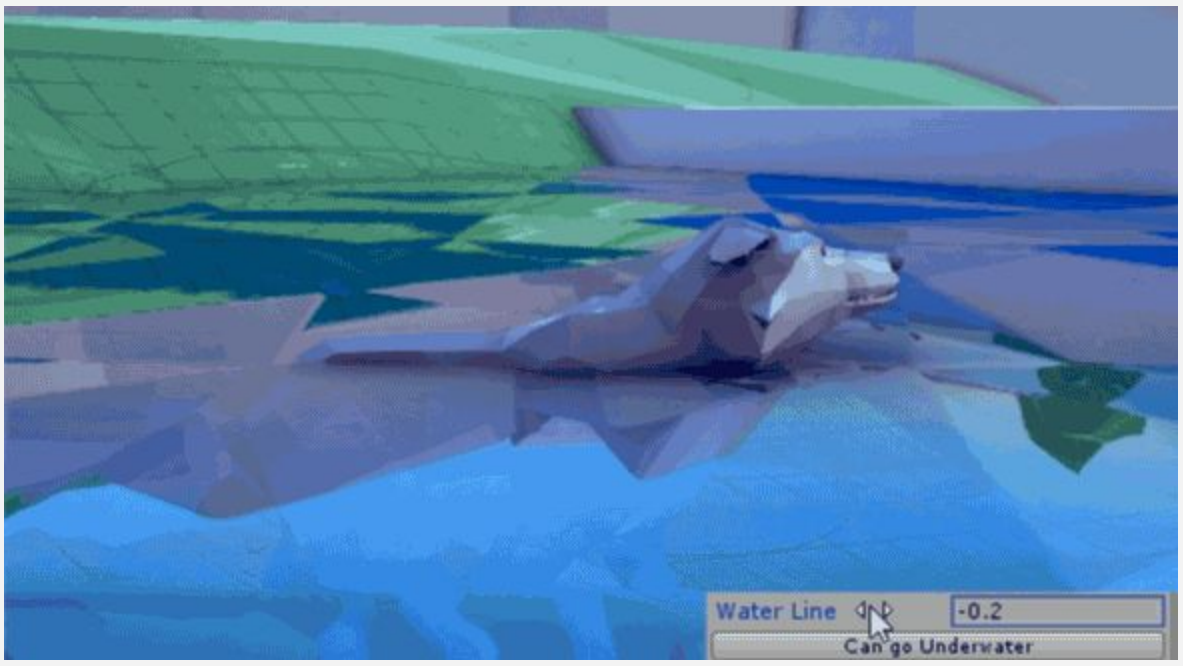


Position:
Add more speed to the movement of the animal while Swimming.
L (Lerp): Lerp value for changing between position speeds.
Higher values: more responsiveness

Animator:
Add more speed to the Animator while Swimming.
L (Lerp): Lerp value for changing between Animation Speeds.
Higher values: more responsiveness

Turn:
Add more rotation while Swimming.
L (Lerp): Lerp value for changing between rotation Speeds.
Higher values: more responsiveness.

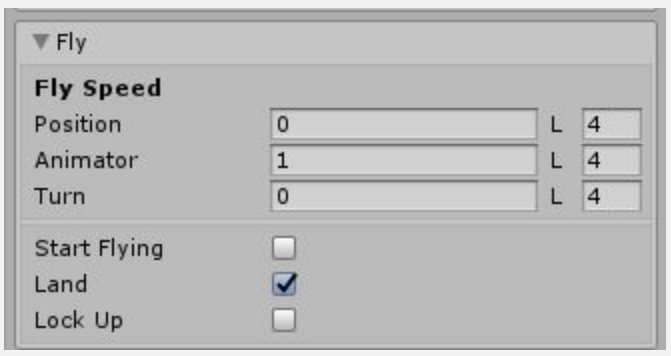
Water Line: Adjust the animal to the Water Surface.



Can go Underwater:
Enables the underwater logic and enables the underwater speeds.
If the animator Controller of the animal does not have a **'Underwater'** Bool Parameter. This option should be disabled, It might cause unwanted behaviours.
(The Underwater Ability is an Animator Behaviour component, it will exist only when the underwater animation is playing)

Fly

This will be enabled if **Can Fly** is Active.



Position:
Add more speed to the movement of the animal while Flying.
L (Lerp): Lerp value for changing between position speeds.
Higher values: more responsiveness

Animator:
Add more speed to the Animator while Flying.
L (Lerp): Lerp value for changing between Animation Speeds.
Higher values: more responsiveness

Turn:
Add more rotation while Flying.
L (Lerp): Lerp value for changing between rotation Speeds.
Higher values: more responsiveness.

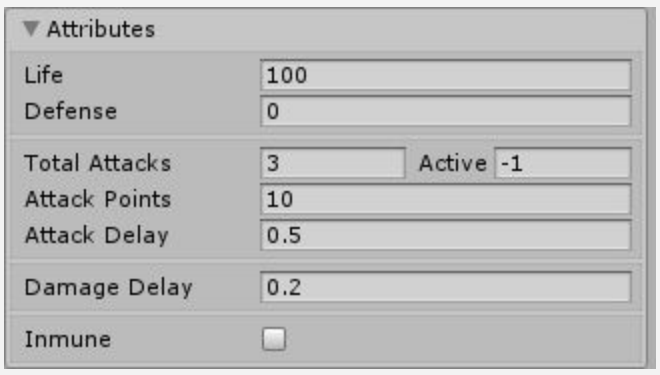
Start Flying:
When the animals starts, it will start on the Fly state.

Land:
If the animal is close to the ground it will automatically exit the Fly state.

Lock Up:
The Animal cannot go up… just forward and down. (Check what happens when is combined with SetToGlide(float))

Tip:
You can use *lockUp = true* for the little dragons babys to make them always glide and fall.

Attributes

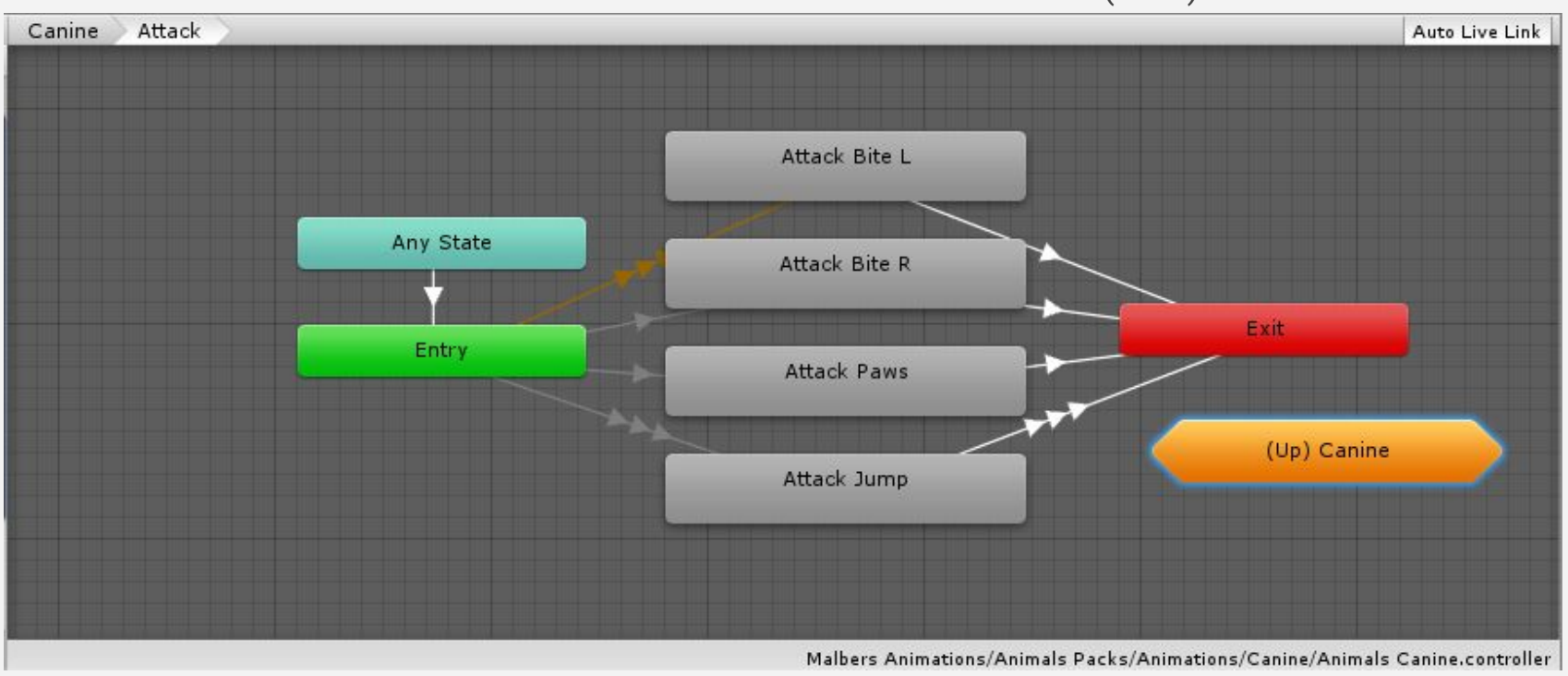


Life:

Health points of the Animal

Defense:
Defense points of the Animal

Total Attacks: Amount of Attack Animations of the Animator Controller. (Ex: 4)



Active:
The Current Animation State Attack. This will set the int **IntID** value of the animal script to the transition value for the attack..
If this parameter is set to -1 it will execute a random attack animation every time Attack is called.

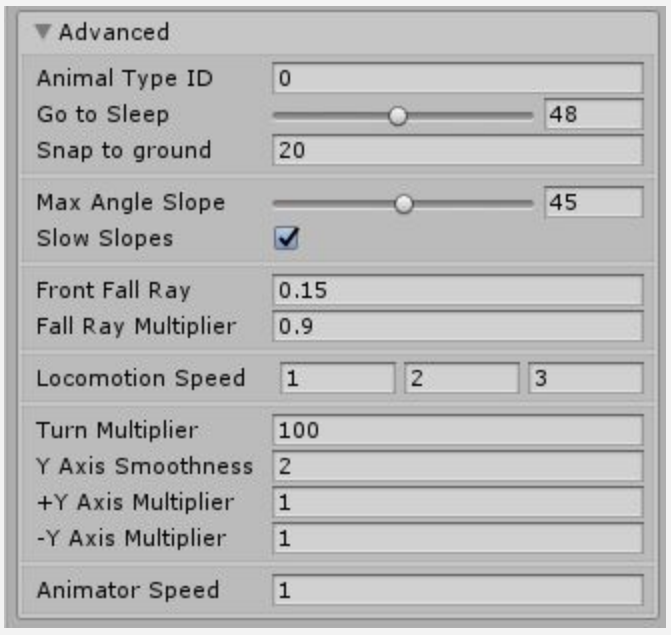
Attack Points:
Damage this animal can cause.

Attack Delay:
Time between attacks. (if this value is equal to zero the Attack time will be his animation Attack length)

Damage Delay:
Elapsed time for the animal to be damaged again.

Inmune:
The animal cannot receive damage and it will not play any damage animations.

Advanced



Animal Type ID:
Enable the Additive Pose Animation to offset the Bones.

Go to Sleep:
Number of Idles before going to sleep (Away From Keyboard)

Snap to Ground:
Smoothness to Snap to the ground (Position)

Align to Ground:
Smoothness to Align to the ground (Rotation)

Max Angle Slope:
Maximum Angle that the animal can walk

Slow Slopes:
If the animal is going uphill: slow it down one speed less

Front Fall Ray:
Multiplier to set the Fall Ray in front of the animal to predict a cliff and a fall.

Fall Ray Multiplier:
Multiplier for the Fall Ray, when falling the animal will detect if is falling or not depending of the amount of this multiplier and the *Chest Pivot Multiplier.*

Locomotion Speed:

This are the values for the Animator Locomotion Blend Tree when the velocity is changed.
(Horse for example have 5 Speeds (1 - Walk, 2 - Trot, 3 - Canter, 4 - Gallop, 5 - Sprint).

Turn Multiplier:
When using movement based on directions(AI) or (MalbersInput.CameraBasedInput) this will highly increase the rotation for the turn animations.

Y Axis Smoothness:
Smoothness (lerp value) of the UPDOWN axis. when pressing 'Jump' to go UP or 'Down' to go Down

+Y Axis Multiplier:
When using movement based on directions(AI) or (MalbersInput.CameraBasedInput) and the animal can Fly/swim UnderWater this will be multiplied by the UPDOWN axis when going Upwards.

-Y Axis Multiplier:
When using movement based on directions(AI) or (MalbersInput.CameraBasedInput) and the animal can Fly/swim UnderWater this will be multiplied by the UPDOWN axis when going Downwards.

Events

On Jump()
Invoked when the animals Jump.

On Attack()
Invoked when the animal attacks.

On Get Damaged(Single)
Invoked when the animal recieve damage. (Sends the damage amount)

On Death():
Invoked when the animal dies.

On Action()
Invoked when the animal makes an action.

On Fly(Boolean)
Invoked when the animal start(true) and stop(false) flying.

On Swim(Boolean)
Invoked when the animal start(true) and stop(false) swimming.

Index

Public Methods:

The public methods gives you more control over the animal script which it can be used to change dynamically the parameters of the animal. Useful for AI behaviours or changing inputs.

All public parameters/properties are available to make changes.

```
public void getDamaged(DamageValues DV)
```
Finds the direction vector and send it to the Damage Behavior with DamageValues.
This method is used mainly for the Attack Triggers when an animal hits another gameobject to any of this triggers.

```
public void AttackTrigger(int triggerIndex)
```
Enable/Disable Attack Triggers by its Index.
If triggerIndex *is set to -1 it will* ***activate*** *all the Attack Triggers.*
if triggerIndex *is set to 0 it will* ***deactivate*** *all the Attack Triggers.*

```
public void SetAttack()
```
Activate a random Attack.

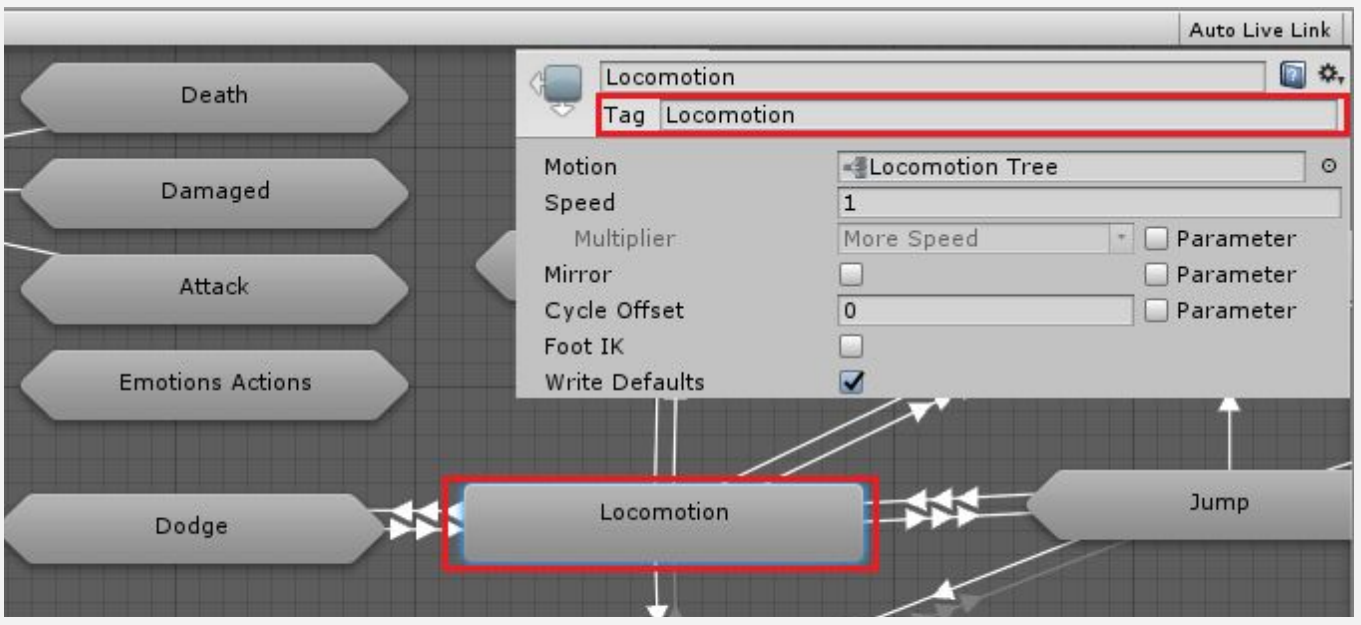
```
public void SetAttack(int attackID)
```
Activate an Attack by his Animation State ***IntID*** *Transition*

```
public void SetAttack(bool value)
```
Enable/Disable Attack1
if enable(value = true) it will activate a random Attack.
if disable(value = false) it will stop attacking.

```
public void SetSecondaryAttack()
```
Activate the secondary Attack. Used on the dragons to set the fireballs Attacks.

```
public bool RealAnimatorState(string tag, int layerIndex = 0)
```
Returns the if the Next or Current Animator State is tagged: ***tag***
layerIndex : *the Animator Layer to find the Animator State tagged:* ***tag*** *(Default: 0 (Base Layer))*

```
public bool RealAnimatorState(int tag, int layerIndex = 0)
```
Returns the if the Next or Current Animator State hashTag is ***tag***
layerIndex : *the Animator Layer to find the Animator State tagged:* ***tag*** *(Default: 0 (Base Layer))*



```
public void SetIntID(int value)
```
Set the Parameter Int ID to a value and update also the parameter on the Animator

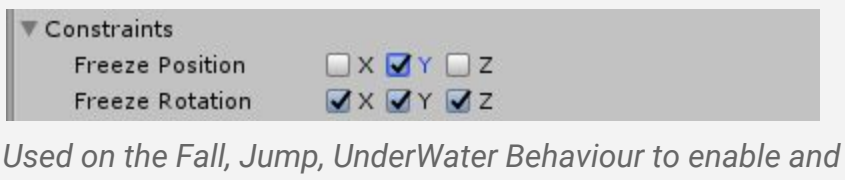
```
public void SetFloatID(float value)
```
Set the Parameter FloatID to a value and update also the parameter on the Animator

```
public void SetIntIDRandom(int range)
```
Set a Random number to ID Int , perfect for playing for random animations. Used for the Random Behaviour

```
public bool IsJumping()
```
True if the Animal is on the Jumping State

```
public void StillConstraints(bool value)
```
*Toggle the **RigidBody** Constraints.*
if value = false

if value = true



Used on the Fall, Jump, UnderWater Behaviour to enable and disable the Y position for the animal.

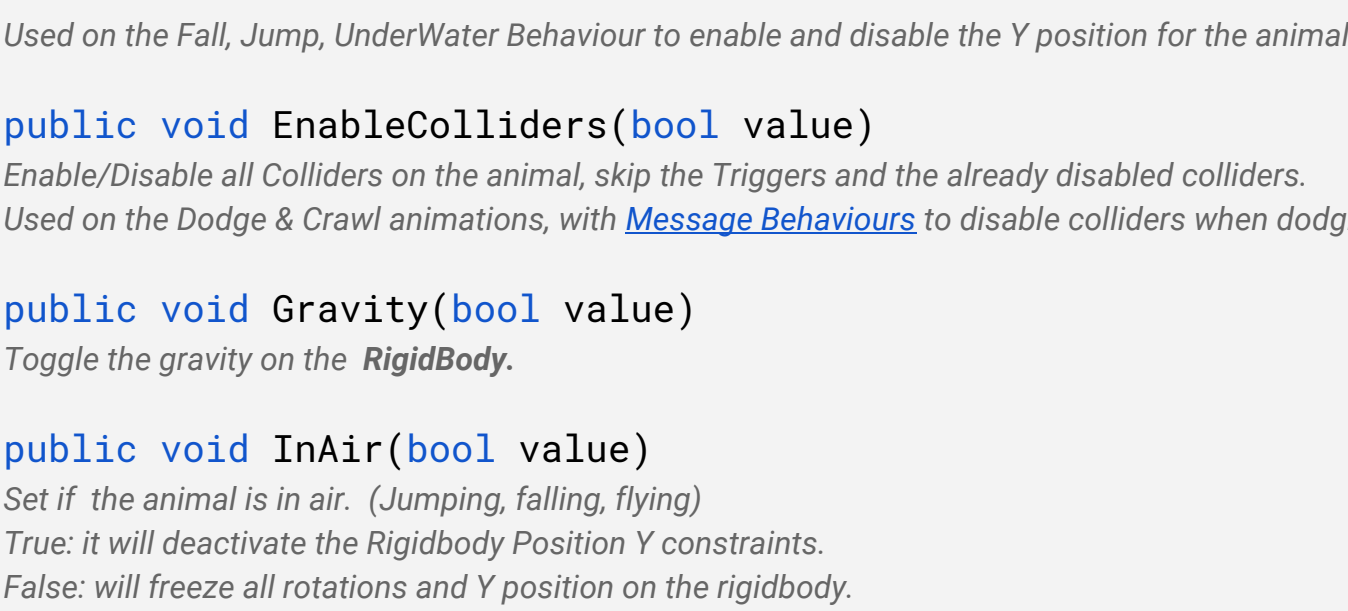
```
public void EnableColliders(bool value)
```
Enable/Disable all Colliders on the animal, skip the Triggers and the already disabled colliders.
Used on the Dodge & Crawl animations, with Message Behaviours to disable colliders when dodging/crawling.

```
public void Gravity(bool value)
```
*Toggle the gravity on the **RigidBody.***

```
public void InAir(bool value)
```
Set if the animal is in air. (Jumping, falling, flying)
True: it will deactivate the Rigidbody Position Y constraints.
False: will freeze all rotations and Y position on the rigidbody.

```
public void SetJump()
```
*Activate the Jump and deactivate it 3 frames later **(USEFUL for Button Inputs)***

```
public void SetAction(int ID)
```
*Set an Action using their Action ID (Find the **ID** on the Animator Actions Transitions)*




```
public void SetStun(float time)
```
Set the Stun to true for a given time (NOT ALL ANIMALS HAVE STUN ANIMATIONS) Check Animator Controller first.

```
public void DisableAnimal()
```
Disable the animal Script and MalbersInput Script if it has it. (Useful when the animals dies).

```
public void SetFly(bool value)
```
*This will deactivate or activate the fly mode directly, the Property "**Fly**" which when is set to true, it will toggle the fly on and off... this will set directly the fly to true or false.*

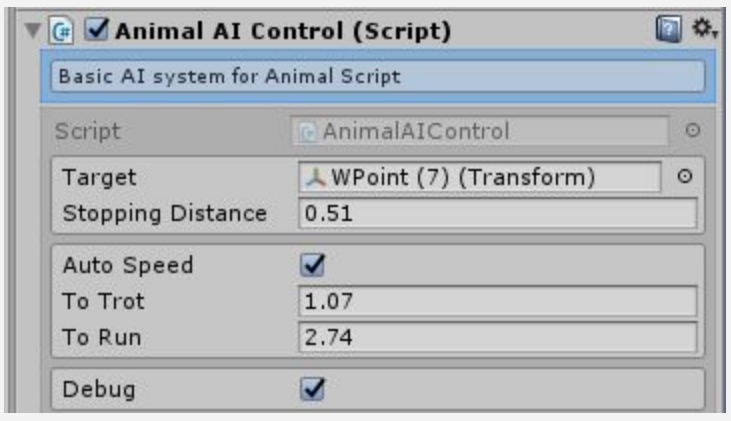
```
public void SetToGlide(float value)
```

*This will enable the gravity while flying and add some **drag** to the **Rigid Body.** (Check Little Dragons Baby Only Glide example).*

[Index](#)

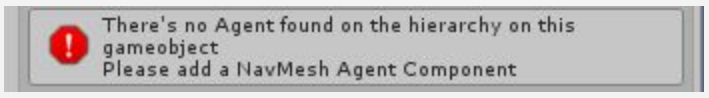
Animal AI Control

The Animal Ai Control script is the bridge between the Nav Mesh Agent and the Animal Script. it moves the animal by the direction that the agent gives when a target is set. using Animal.Move(direction)



This script Requires a NavMeshAgent

Usually the Agent is on the same level of this script or as a child on the hierarchy.. in not agent was found, you will get a error message:



The Agent is only used to give the direction to the animal to move on the Navigation mesh.
the animations moves the Agent not the other way around..
that is why Agent **updateRotation** and **updatePosition** are set to false;

- Make sure you have a navigation mesh Baked on the Navigation Tab.

Parameters:

Target:
Target to follow

Stopping Distance:
Agent Stopping distance

Auto Speed:

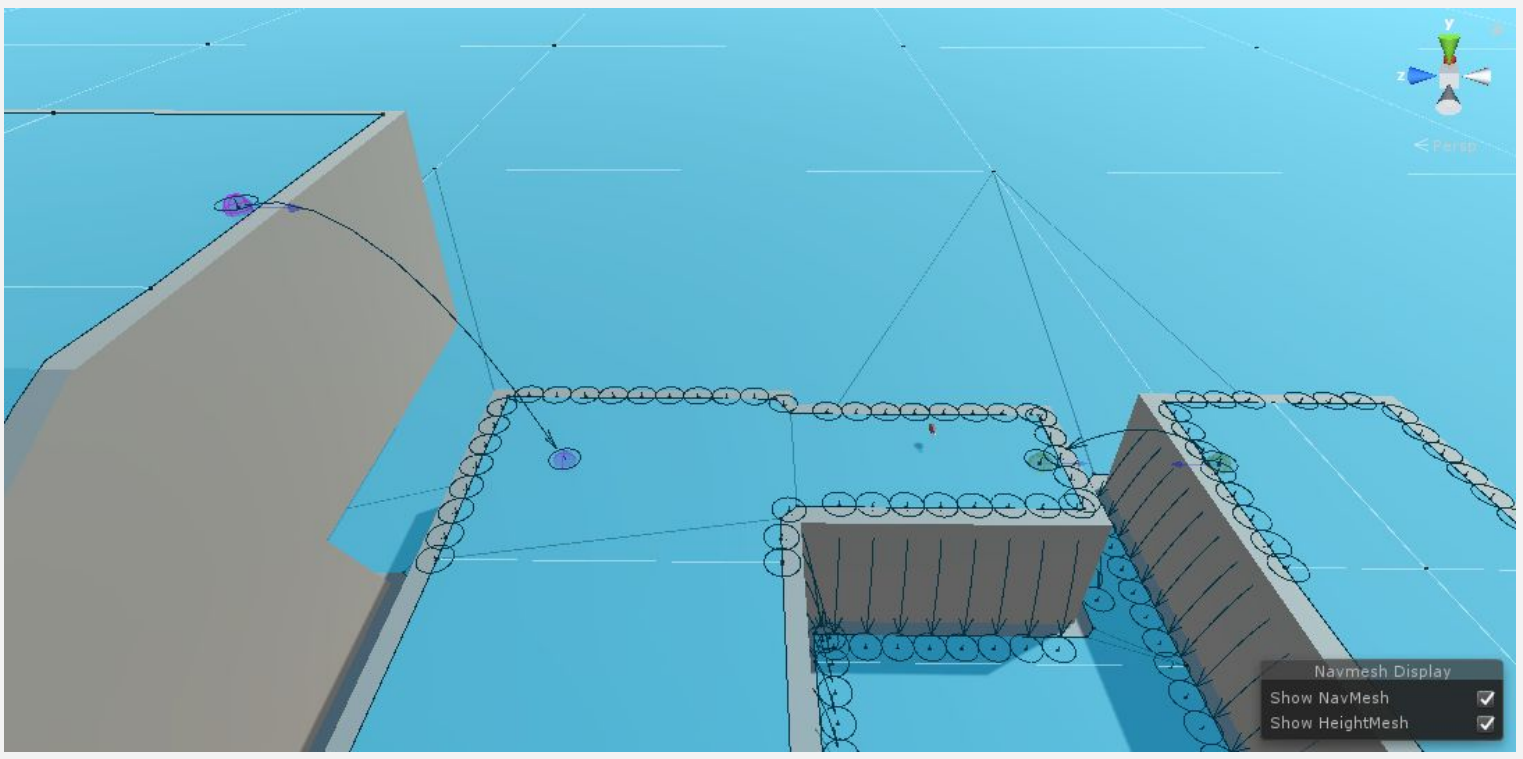
The speed changes are handled by this script. it will show **ToTrot** and **ToRun** parameters

To Trot:
Minimum distance to start trotting

To Run:
Minimum distance to start running

Debug:
It will show the Radius for the **Stopping Distance**, **To Trot** and **To Run** parameters

Animal AI Control will also handle OffMesh Links for ***Fall*** and ***Jump***.. (Check AI Scene)



Index

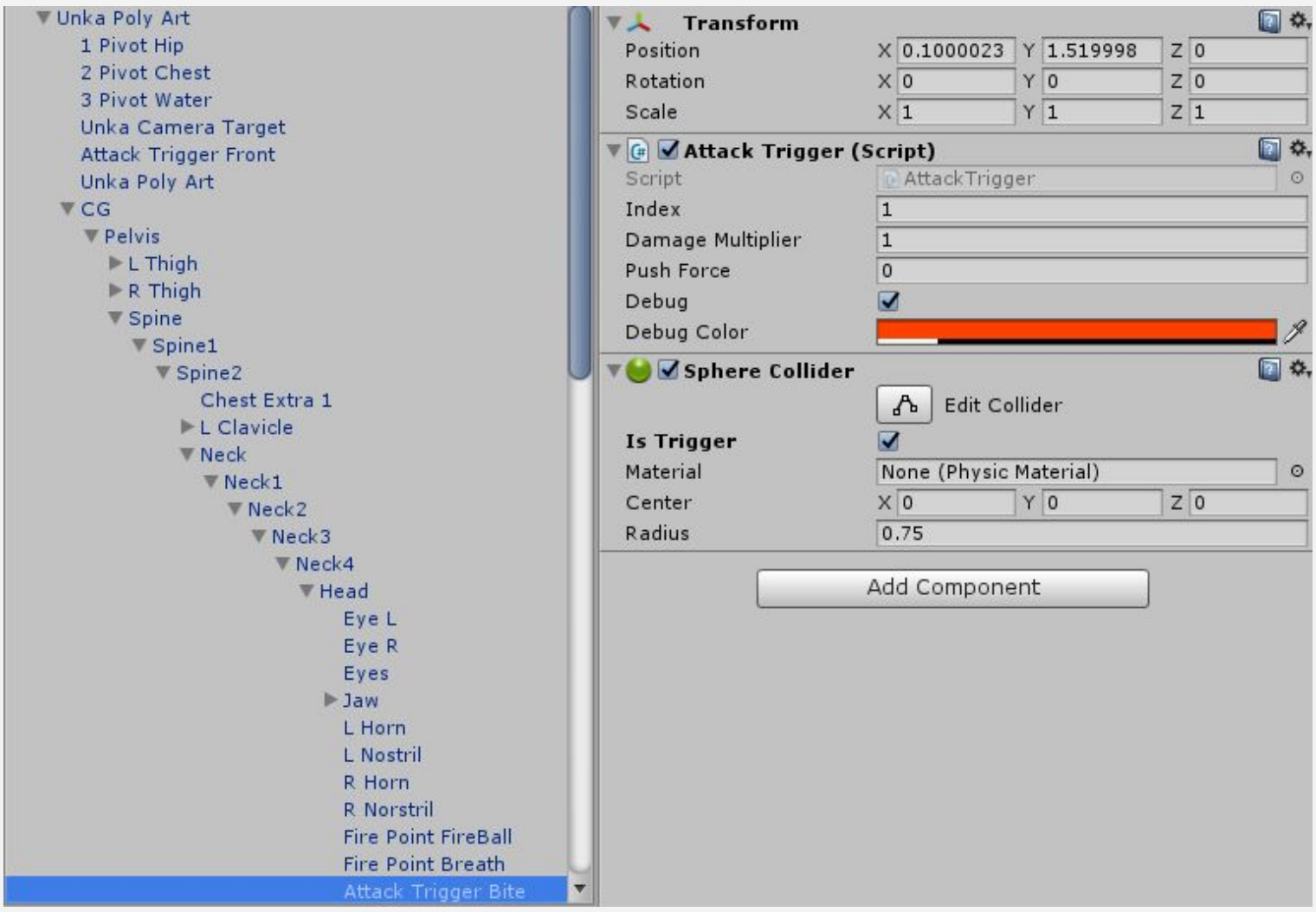
Action Zone

(STILL filling it)

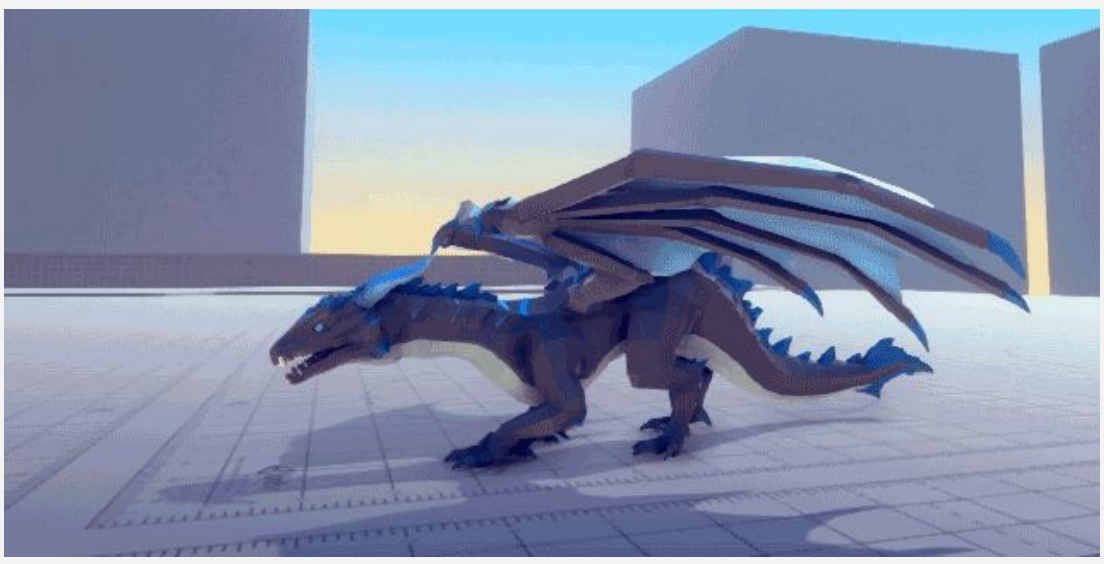
Index

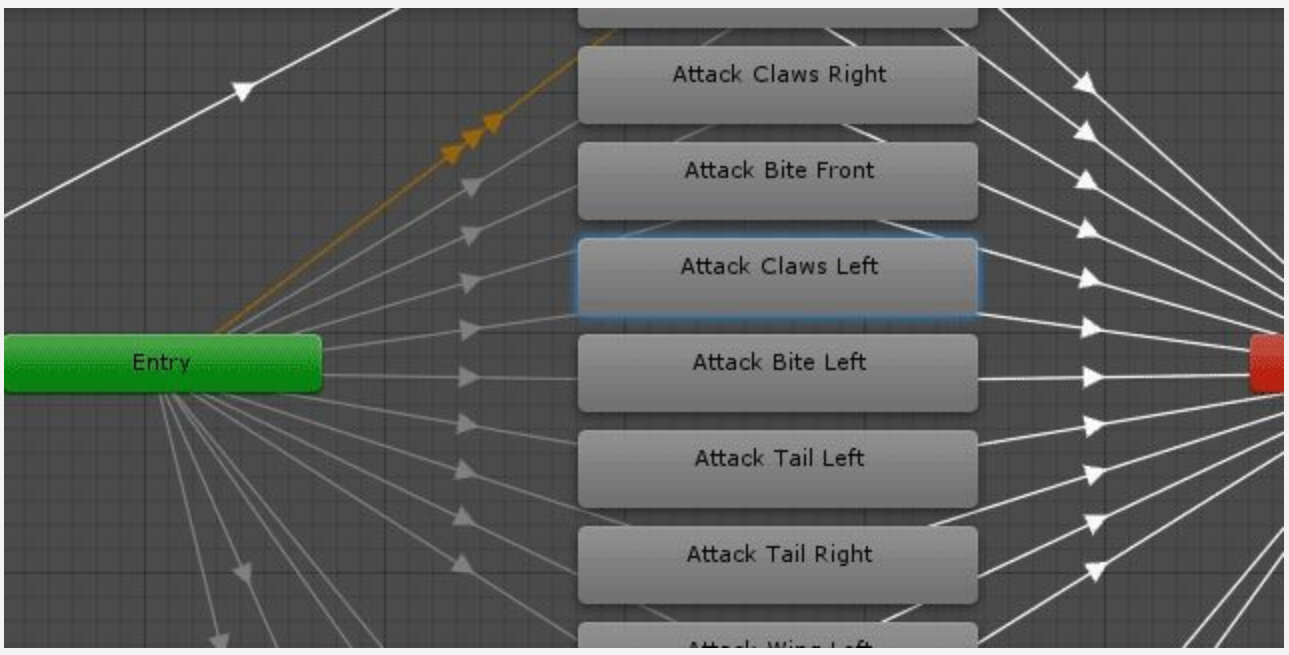
Attack Triggers

(Requires a Collider set to Trigger)
Attack Triggers managers for Triggers which can cause damage to other animals.
When other animal enters an active **Attack Trigger**, the *OnTriggerEnter()* is activated, inside the Direction of the damage and the Attack points are send to the Hitted Animal Using Animal.getDamage(DamageValues). which removes life and execute the damage animations.



These Attack Triggers are Enable/Disable by the **Attack Behaviour** attached to the Attack Animations.





Parameters:

Index:
The Index or ID of the Attack Trigger.

Damage Multiplier:
Multiplier for the Animal Attack. If you want the *Bite Attack Trigger* cause more damage than the *Tail Attack Trigger*, modify this parameter.

Push Force:
If the Object the attack trigger is hitting, has a RigidBody, add a push force to it.

Index

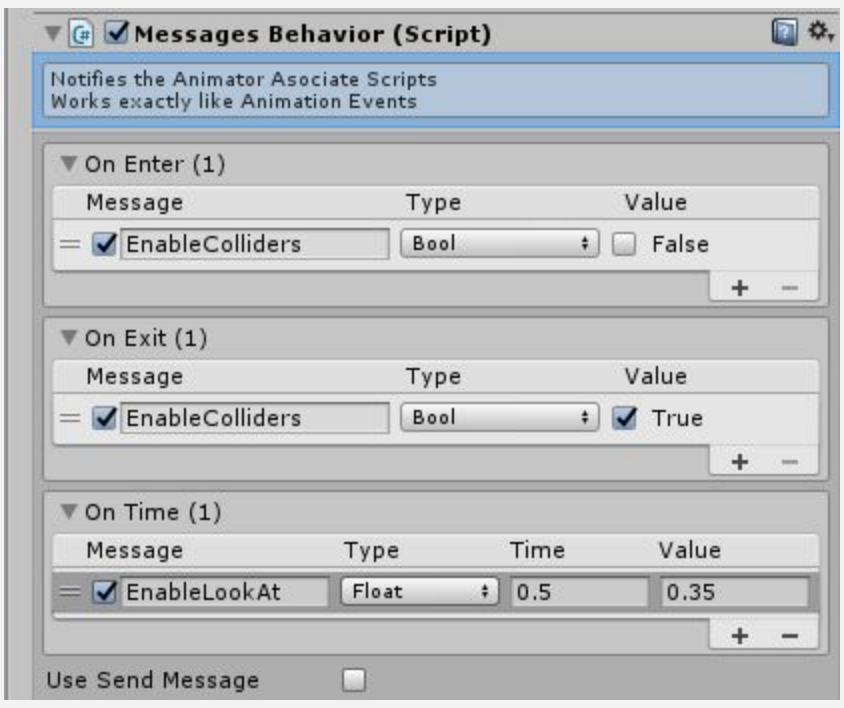
Animator Controller Behaviours

Sometimes you only need to execute code when an specific animation is playing, this is very useful to avoid repetitive code for asking on which animation are you currently in. the animator behaviours are perfect for this kind of tasks.

Index

Messages Behaviour

Notifies the Animator associate scripts using the ***IAnimatorListener*** interface. Works exactly as the Animations Events, the only difference is that animation events throws errors when they don't find a receiver. if MessageBehaviour does not find a receiver nothing happens ;) .
 Also Message behaviour are attached to the Animation States on the animator, not the Animation clip. This have the advantage that we can know exactly if we **enter** or **exit** an animation state.



Message Behaviour are extensively used on the Animal Script Controller to change attributes.

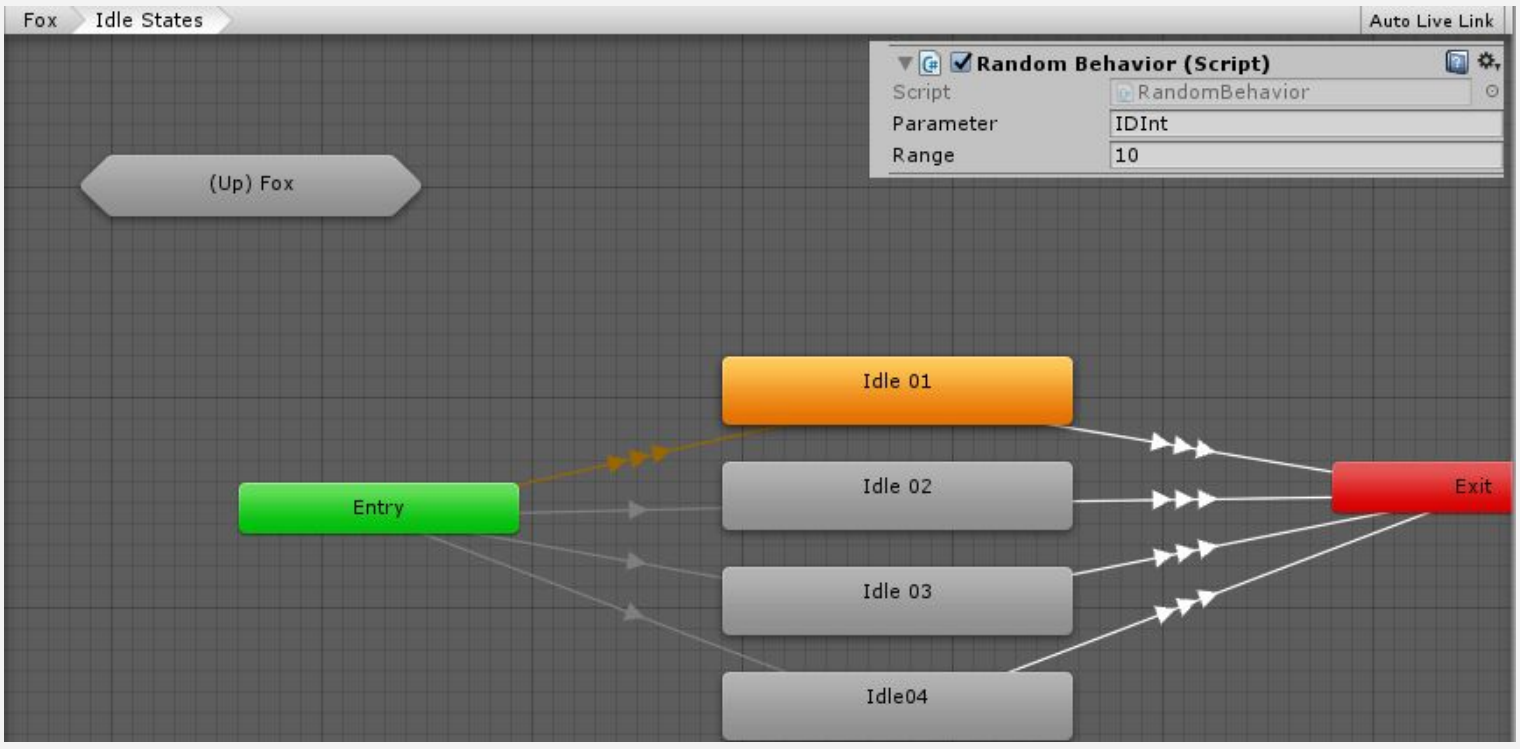
IAnimatorListener needs this method to exit which you can find inside the Message behaviour Script:

```
public virtual void OnAnimatorBehaviourMessage(string message, object value)
{
      this.InvokeWithParams(message, value);
}
```

If you don't want to use the ***IAnimatorListener*** interface,
you can activate ***Use Send Message*** instead.

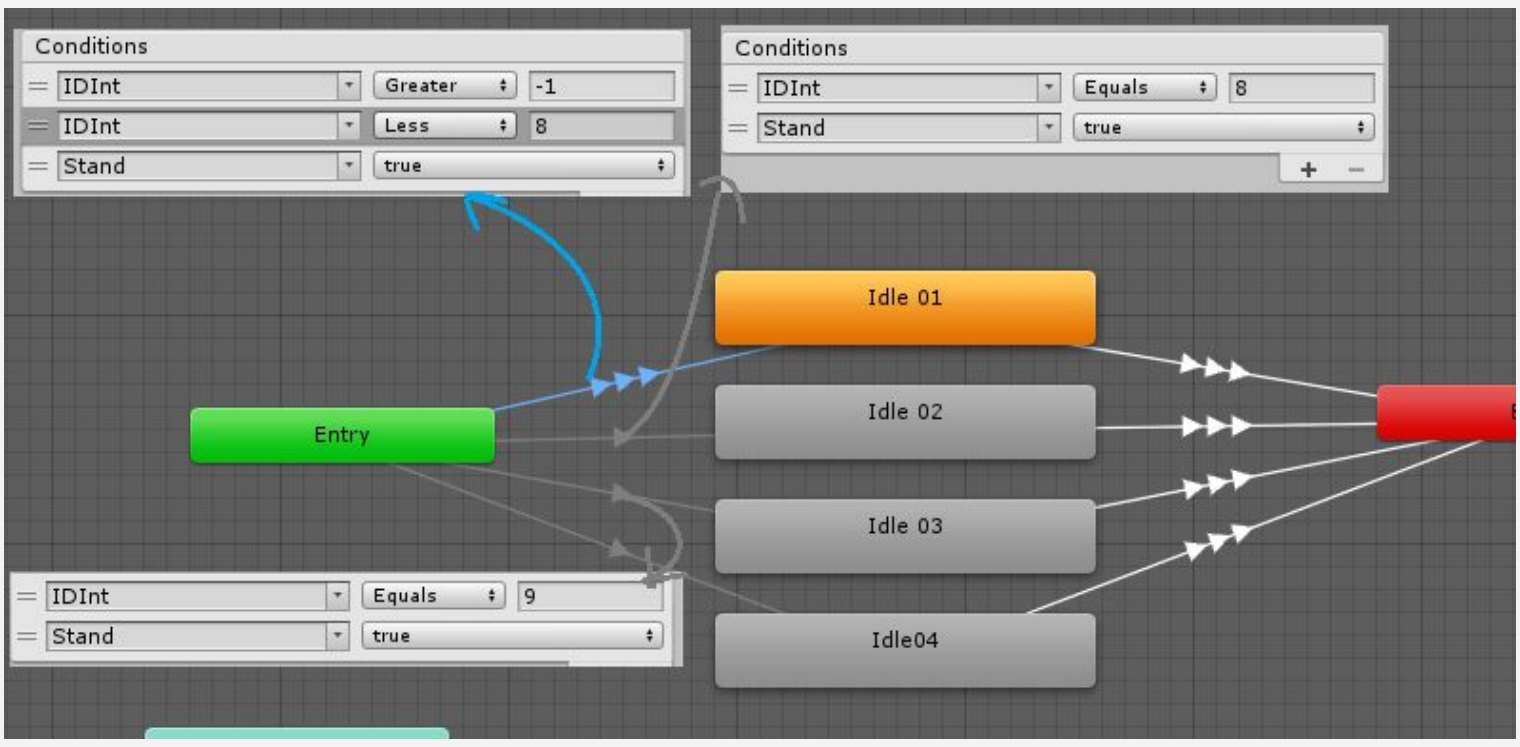
Index

Random



Is used for playing random animations inside an animation state machine using **IDInt** as the random parameter (Useful for play random idles, attacks, deaths)

Tip:
When you need an animation to be played more than the others, set a big Range on the Random behaviour, and to the principal animation state, set the Transition to have the most opportunities to be played.



Index

Damage

Almost all the animals have 6 Hit Animations so when an animal receive a hit I save the direction of the hit.
This script calculates the angle of the hit and use it to play the Right Animation. using the IntID animator parameter.

Index

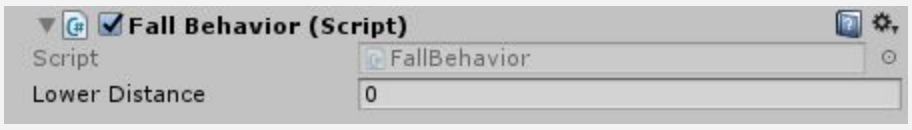
Recover

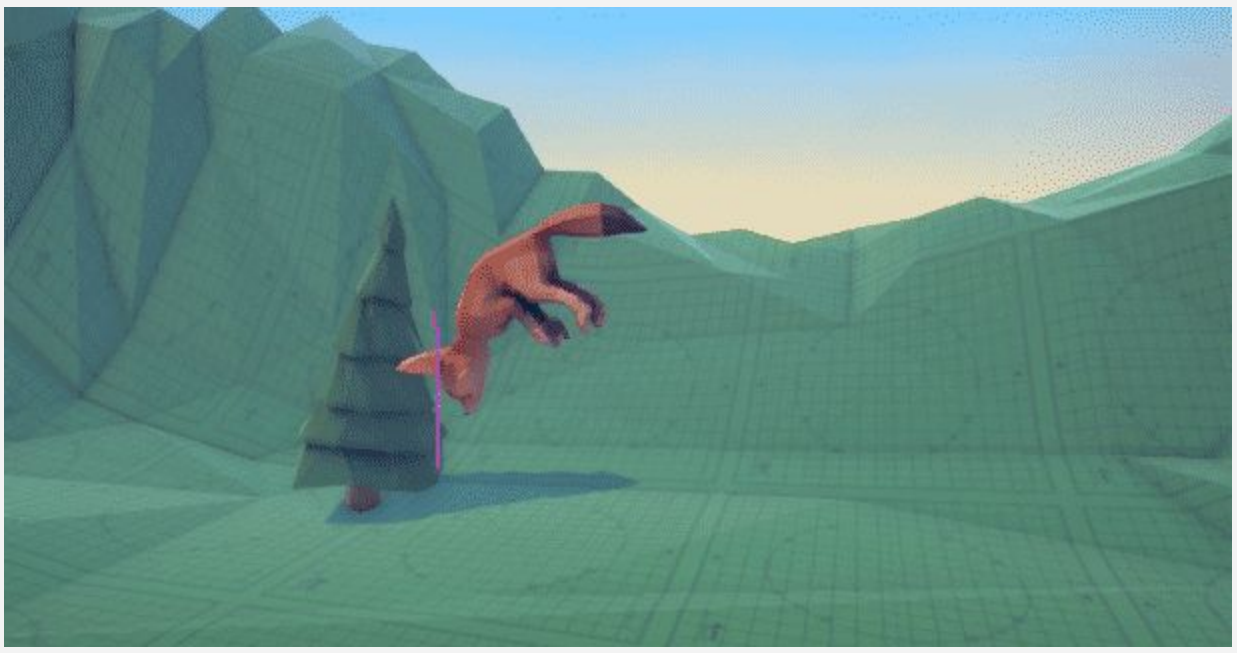
Is used on the recover animations (land and water). This script re-enable the **RootMotion** on the animator and restore the **Rigidbody** constraints to the default.

Index

Fall

Is used on the fall Animation State. This script disable the *rootmotion* on the **Animator** to create a free fall motion using the **Rigidbody** inertia accumulated from the **Animator** speed. Freeze all Rotation on the **RigidBody** and updates the *FloatID* on the **Animator** to create a smooth blend between the *High* and *Low* Fall Animation blend tree.





Lower Distance:
The Lower Fall animation will set to 1 if this distance is greater than the current distance to the ground.

Index

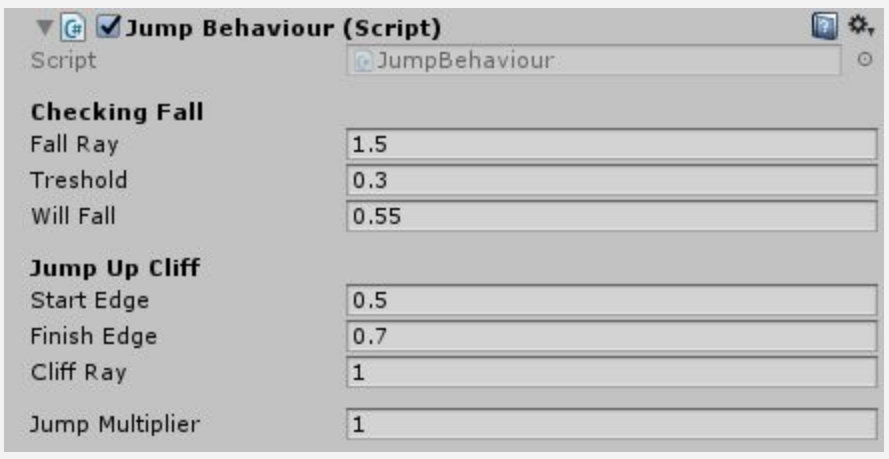
Sleep

Uses the **GotoSleep** parameter to count Idles cycles if the animal is not moving. if the cycles is greater than the **GotoSleep** then uses the **IntID** parameter to transition to the next animation (Sleep).

Index

Jump

Is used for calculate the perfect jumps and landing of the animals.



At the beginning of the Jump take off, the **IntID** parameter is set to **0**. then a Jump/Fall ray is casted during the jump animation, checking if the ground beneath the animal is still at the same height level from where the jump started.

The Jump/Fall Ray is represented by a **red** line.

If no ground is found the **IntID** parameter is set to **111**. Then, if the Jump animation reaches the fall transition, it will break the jump animation and it will start the falling.

Fall Ray:
Ray Length to check if the ground is at the same level all the time.

Threshold:
Terrain difference to be sure the animal will fall.

Will Fall:
Animation normalized time to change to the fall animation if the ray checks that the animal is falling.

Start Edge:
Animation normalized time start checking for higher terrain.

Finish Edge:
Animation normalized time finish checking for higher terrain.

Cliff Ray:
Cliff Ray Length to check for higher terrain.

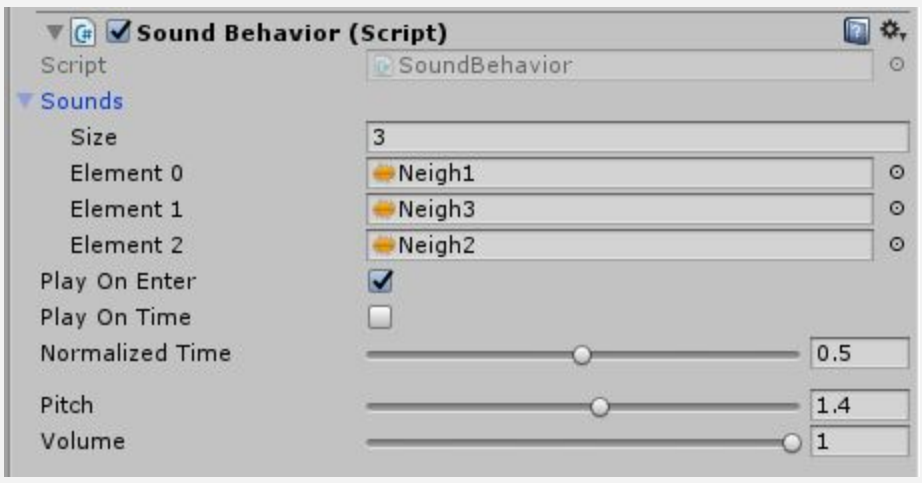
The Cliff Ray is represented by a **black** line (is visible from 0.6 and 0.8 normalized time of the Jump animation).
if that ray find a higher ground it will change the **IntID** parameter to **110** which will execute the transition to the **Jump to Cliff** Animation State.

Jump Multiplier:
Multiplier to add more length to the Jump/Fall Ray.

Index

Sound

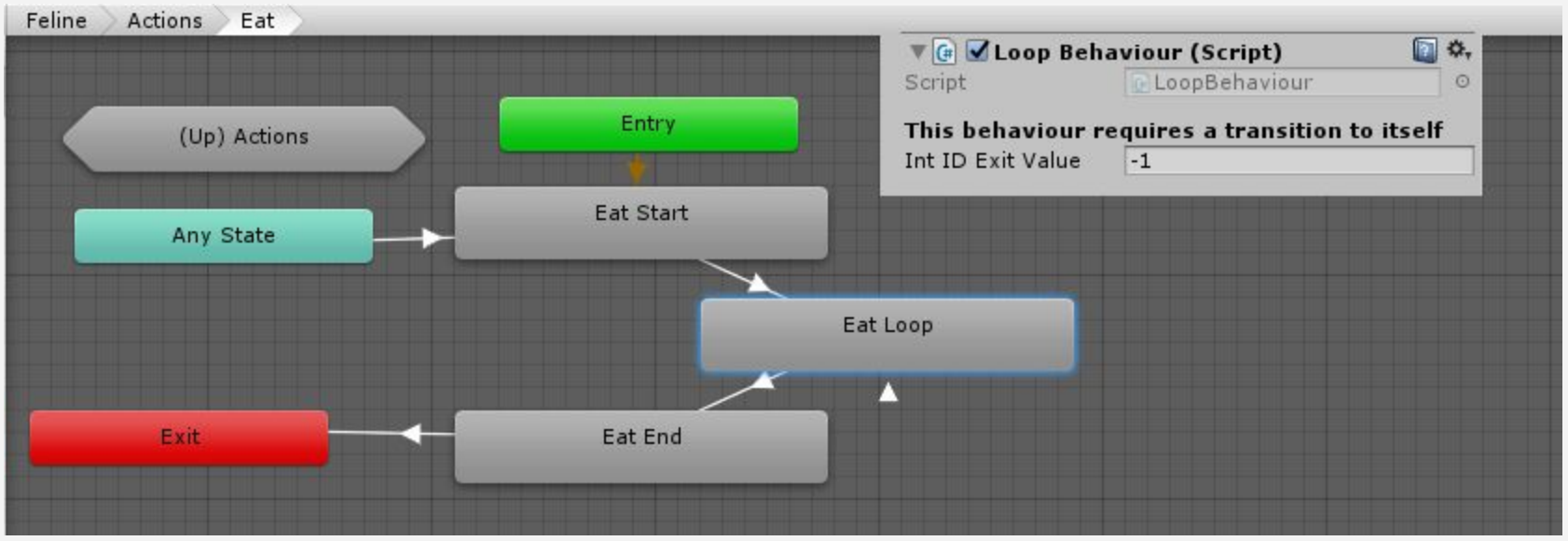
Plays a random sound from a list of audio clips when entering, or on an specific time of an Animation State.
It will create or use a AudioSource from the Animal. (Example the Horse Neigh Animations)



Index

Loop

This Animator behaviour is used to make loops on som animations depending the **`Animal.Loop`** property…



Useful for making Loop Eat,Drink or Attack animations using the **`Animal.Loop`** property...

Index

Utilities Scripts

Utilities Script are simple scripts to make life easier to the Animal controller. You can use them or not, or replace them for more advanced script or assets from the store. Is up to you ;)

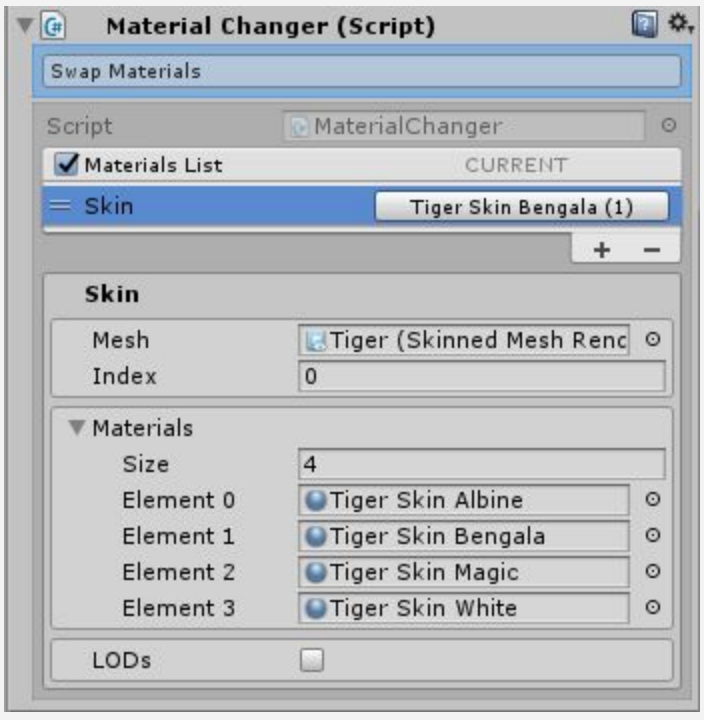
LookAt

Public Methods

Index

Material Changer

Uses a List of Material Items, which holds a mesh renderer and a list of materials to use as Active Material.



Material List:
Enable/Disable the Options on the selected Material Item.

Mesh:
Index:

Materials[]:

LODs:
LOD Mesh:

Public Methods

Index

Active Meshes

Public Methods:

Index

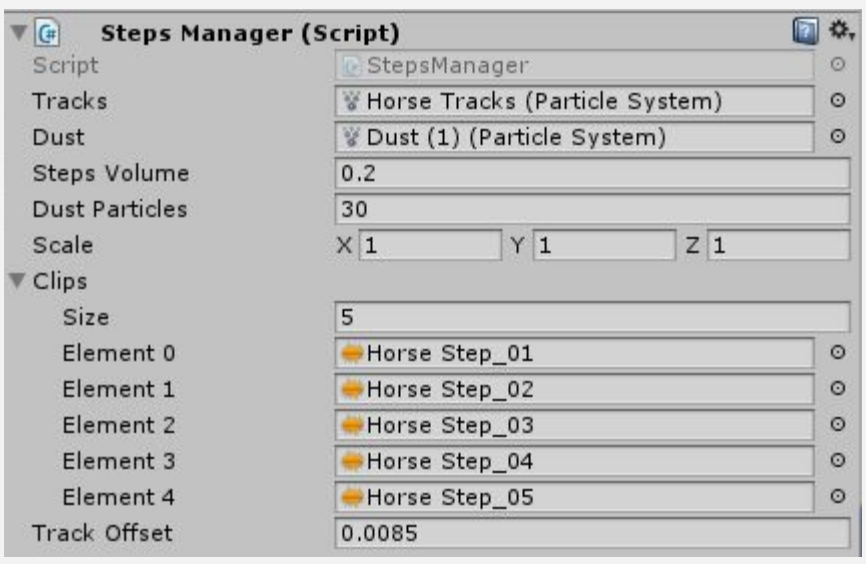
Blend Shapes

Public Methods:

Index

Steps Manager

This script receive from the **Step Trigger** all contacts with the ground and place the tracks and dust particles and play a step sound.



Tracks:
Gameobject which holds the Animal Footstep track particle effect.

Dust:
Gameobject which holds the dust particle effect.

Steps Volume:
The volume of the AudioSources located on each Step Trigger object.

Dust Particles:
How many particles to emit on a footstep.

Scale:
Add or reduce more scale to the track placement..

Clips:
Random sound to play on a footstep.

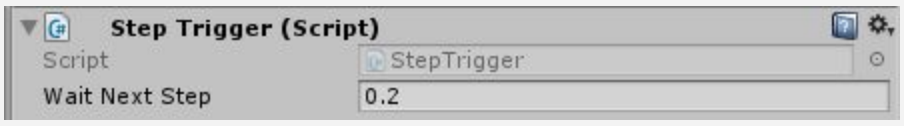
Track Offset:
Extra distance from the ground to avoid flickering over the ground.

Public Methods

Index

Step Trigger

This script manages the OntriggerEnter for the Trigger of the game object so every time this trigger makes contact with any collider. it will send upwards to the **Steps Manager** where the contact was made so the **Steps Manager** can place the tracks and dust particles and play a step sound.

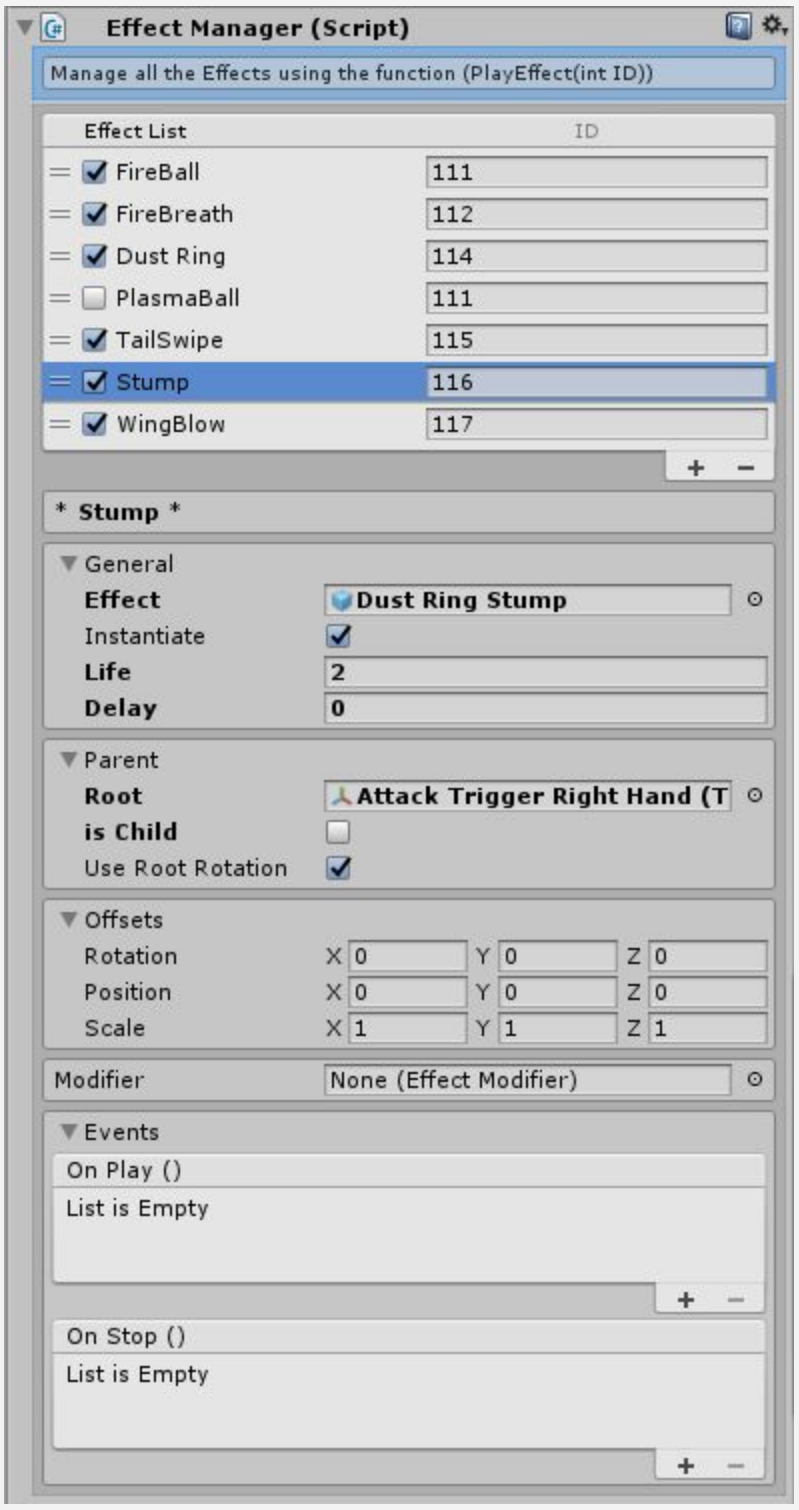


Wait Next Step:
Seconds to wait for a new step to be active. Perfect to avoid repetitive tracks.

Index

Effect Manager

Manage all the effects using the function: `PlayEffect(int ID)`



It has a List of **Effects** to manage several effects simultaneously..

Effect Parameters:

When you create a new Effect you will set the **name** and the **ID** parameters. when a Item of the list is selected its parameters will appear.

Effect:
The Prefab or Gameobject which holds the Effect(Particles, transforms).

Instantiate:
True you want to make a copy of the effect, or if the *effect* is a Prefab
if False the Toggleable parameter will show.

Toggleable:
The *effect* is already on the scene and you want to enable/disable it instead of instantiate/destroy it

Life:
Duration of the *effect* . The *effect* will be destroyed if **'Instantiate'** is set to true.
If **Life** < 0 then The *effect* will not be destroyed by the Effect Manager.

Delay:
Time before playing the Ef*effect* fect.

Root:
Uses the Root transform to position the *effect* .

isChild:
Set the *effect* as a child of the Root transform.

Use Root Rotation:
Orient the *effect* using the root rotation.

Position Offset:
Add additional offset to the *effect* position.

Rotation Offset:
Add additional offset to the *effect* rotation.

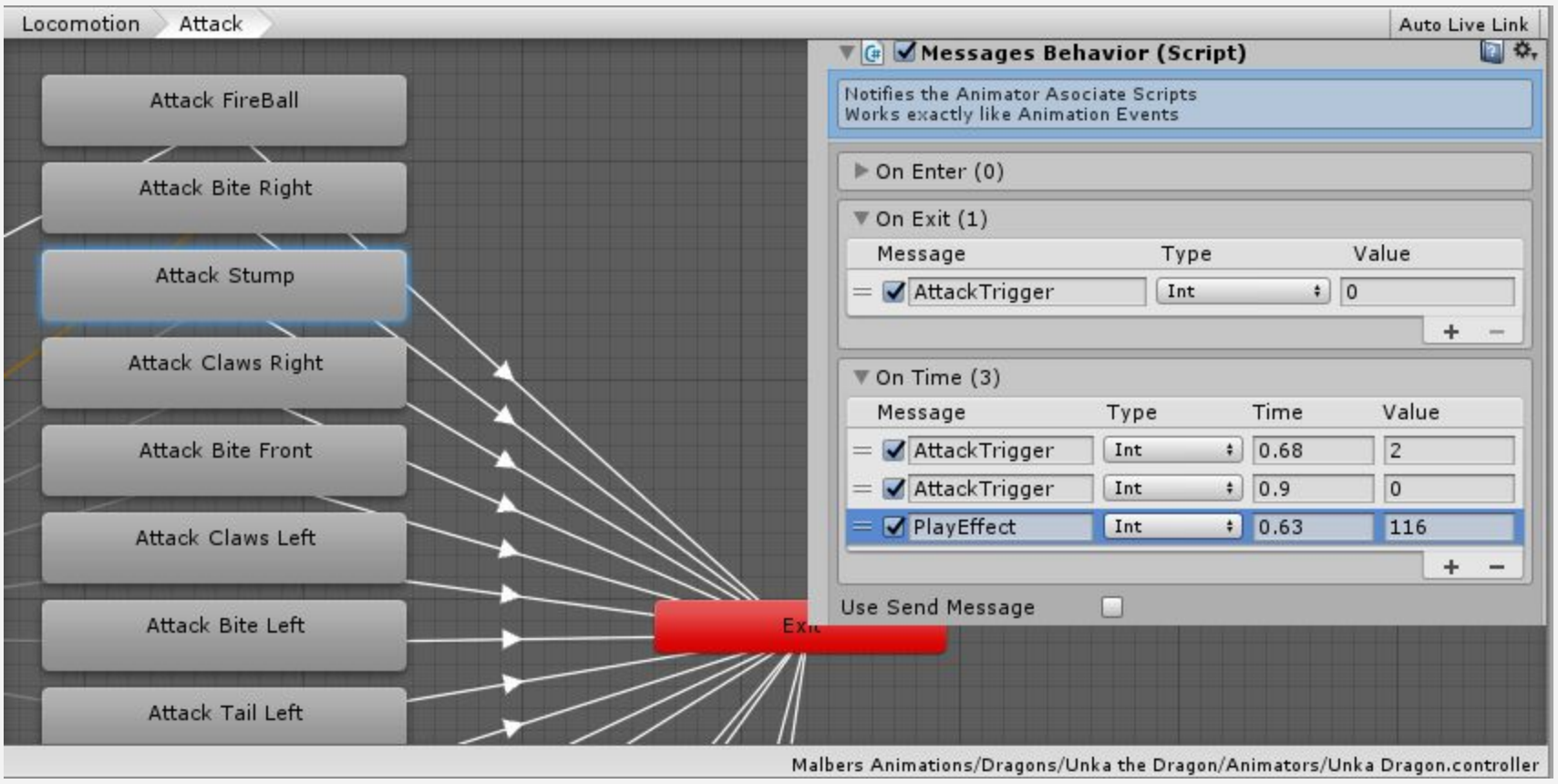
Scale Offset:
Add additional offset to the *effect* Scale.

Effect Modifier:
Scriptable object to modify the *effect* parameters before, during or after the effect plays.

On Play Event:
Invoked when the *effect* plays.

On Stop Event:
Invoked when the *effect* stops.

All effects are usually called by the Message Behaviour on the Animation States:



Public Methods

```
void _DisableEffect(string name)
```
Disable all effects using their name. If a disable effect is called it will be ignored.

```
void _DisableEffect(int ID)
```
Disable all effects using their ID. If a disable effect is called it will be ignored.

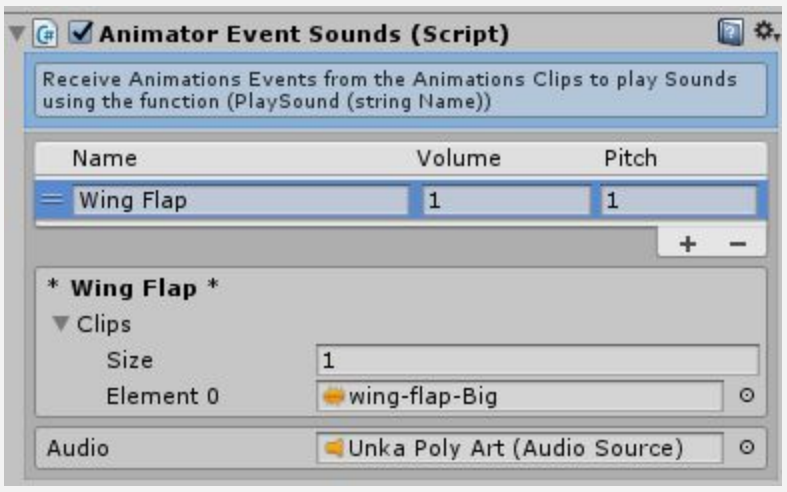
```
void _EnableEffect(string name)
```
Enable all effects using their name…

```
void _EnableEffect(int ID)
```
Enable all effects using their ID.

Index

Animator Event Sounds

Receive Animations Events from the Animations Clips to play Sounds using the function: **`PlaySound(string name)`**



The reason to use Animation Events for sounds is because sometimes Animation State on the animator controller are **Blend Trees**.
The animation clips are mixed (Example: Fly Animation State is a blend Tree and they have fly and glide).
Using Animation Events instead of Sound Behaviours ; we can know the weight of an Animation Clip on a blend tree , and use that weight as volume. so when the blend tree is on the Glide animation clip the volume for flapping wings is almost 0.

There's a List of **EventSound** to receive more than one animation event.

Name:
The value of the string parameter of the **`PlaySound(string name)`** function.

Volume:
This value is a multiplier for the *AudioSource* volume parameter.

Pitch:
This value is a multiplier for the *AudioSource* pitch parameter.

Public Methods

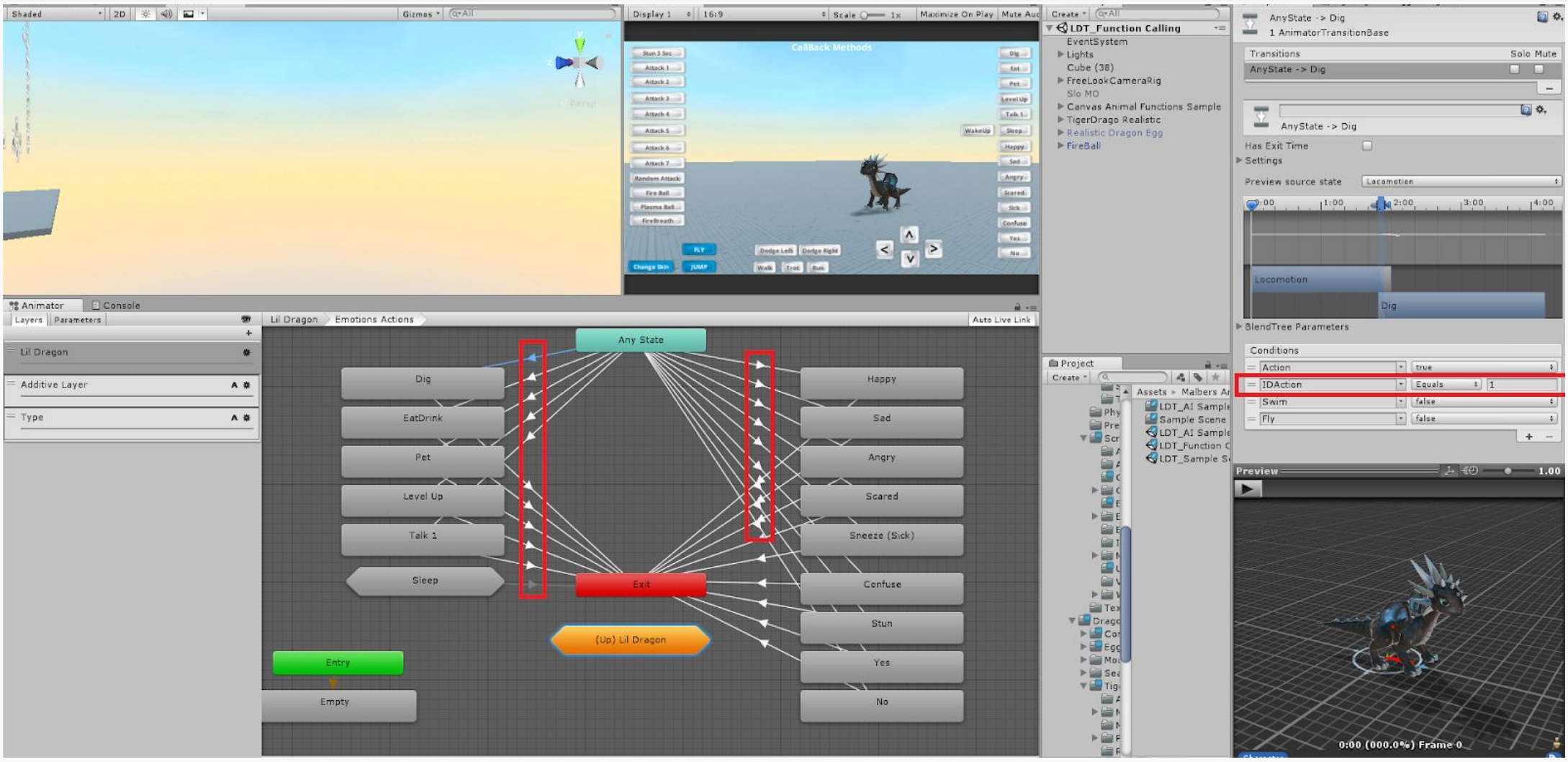
```
PlaySound(string name)
```
Apply to the audio source a random clip from the clip list given a EventSound name.

Index

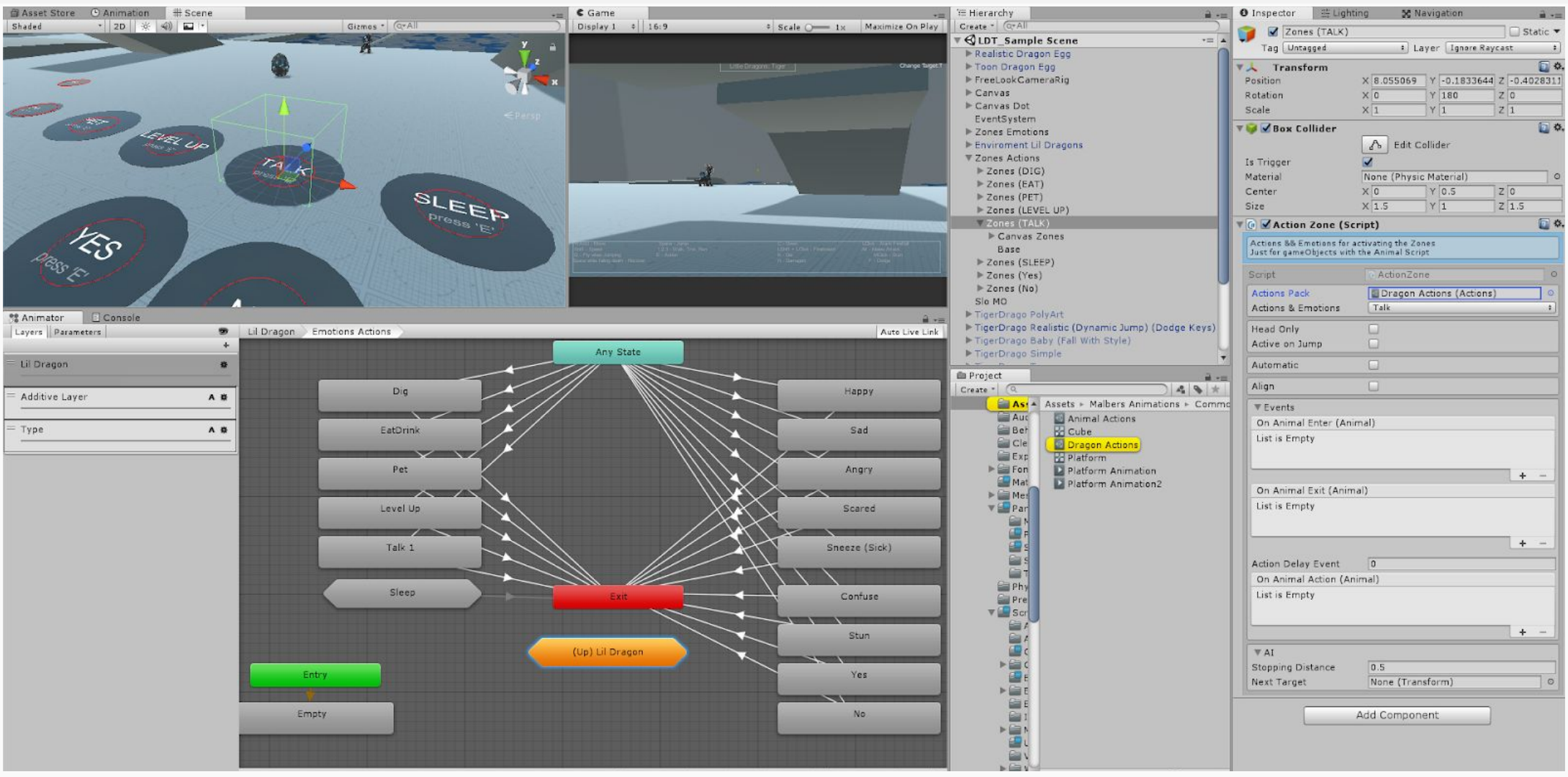
FAQ

1. How can I know which actions to call using SetAction() ?

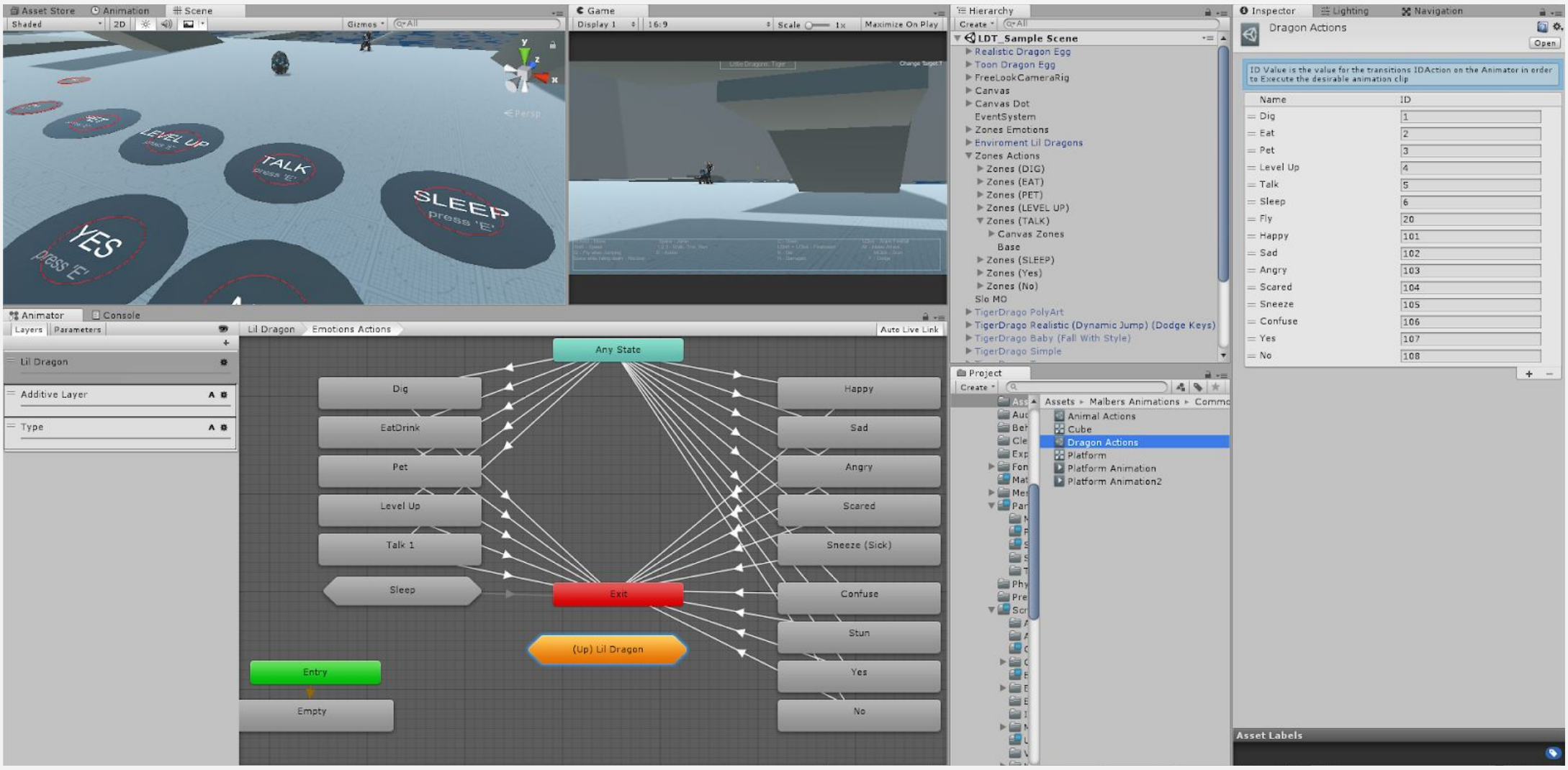
You can check the actions 2 ways:
Looking to the Animator Transitions conditions for the Actions to know their IDs...



OR on any Action Zone there is an action Pack required:

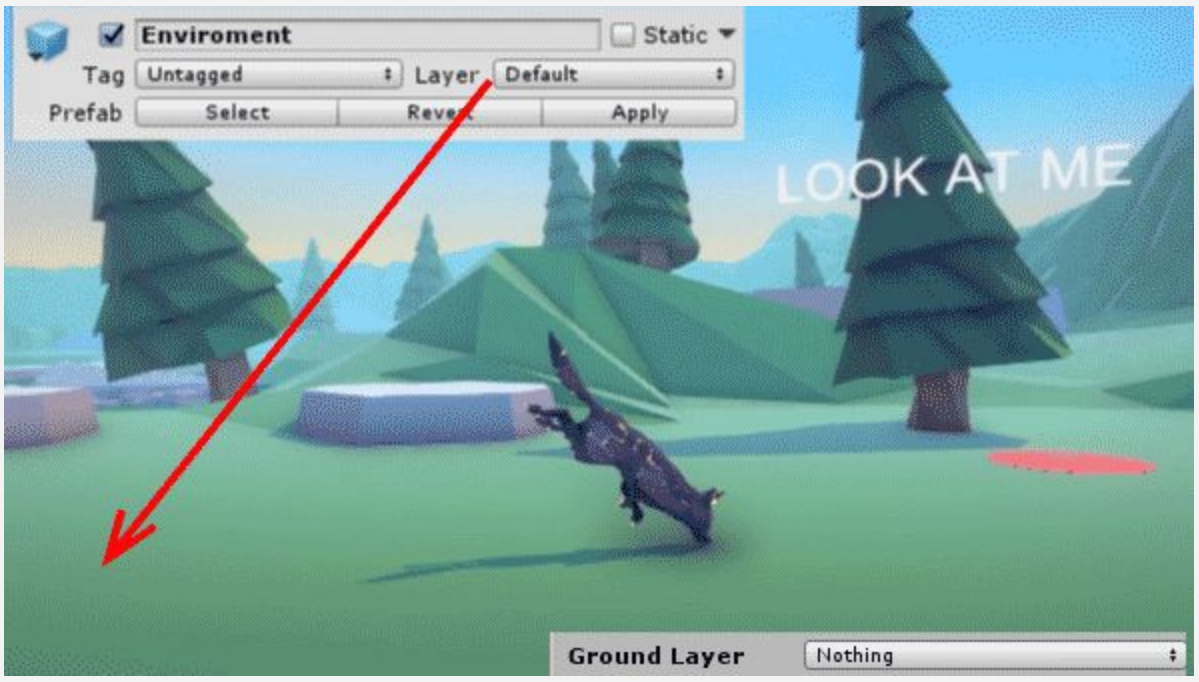


Which Holds the Actions IDs



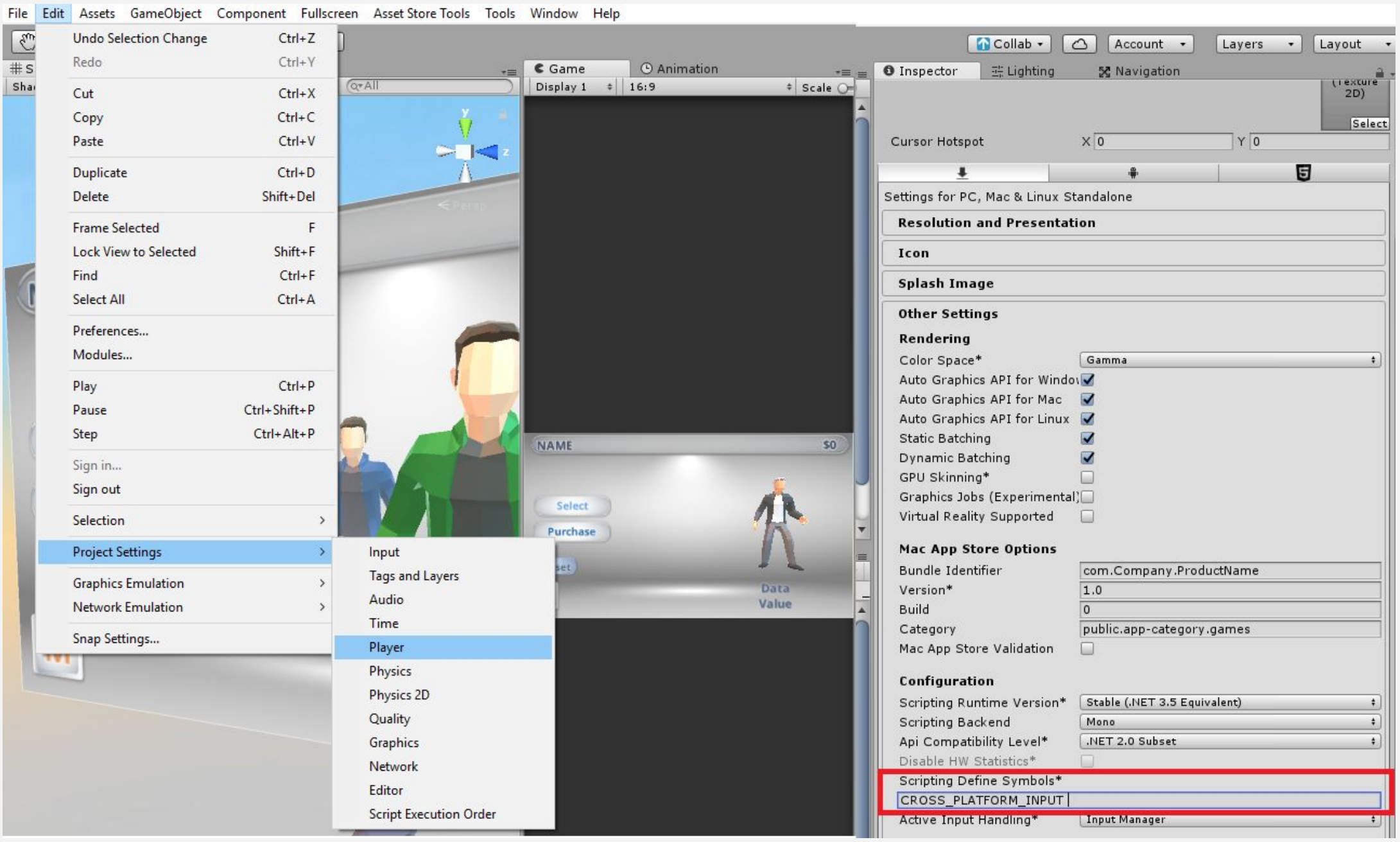
2. My Animal/Creature just keep sliding and falling on the ground?

If the animal start to fall for no reason, it should be because **Ground Layer** on the **Animal** Script is empty, or the layer of the Game Objects beneath the animal needs to be added to the Ground Layer Mask.



3. Why I get this error 'The type/namespace name *'CrossPlatformInput' does not exist.....'* while importing the asset?

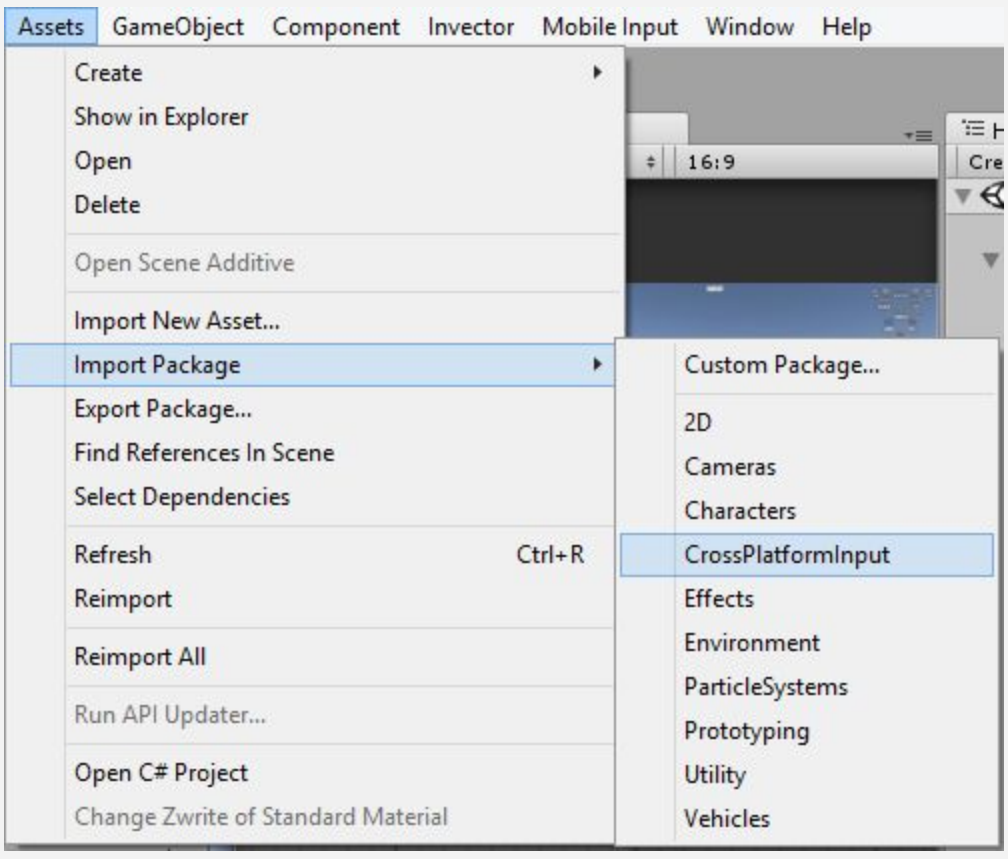
Unity AssetStore updated its [Submission Guidelines] Politics on the last quarter of 2016, and since then we, as a publishers, cannot include the Standard Assets.
So those errors come from the **CROSS_PLATFORM_INPUT** on the Player Settings....



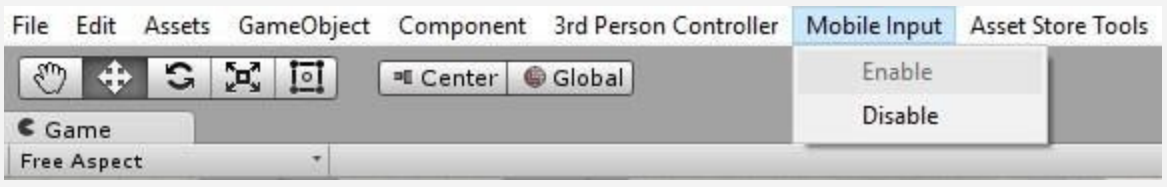
If you are NOT using this **Custom Scripting Define Symbol** which belongs to the **Standard Assets** then you should remove it...

This enables on the scripts the ability to use the animals with Mobile Input.... but it uses the **Cross Platform Input** Class from the Standard Assets..

So, for the mobile sample scene in order to work; you will need to import ***CrossPlatformInput*** from the Assets Menu.



After importing the package, change your platform to **Android** or **iOS** on the ***Build Settings*** and make sure you have the Android/IOS **SDK** installed and don't forget to ***Enable*** the Mobile Input after change the platform, it should work right on the Editor.

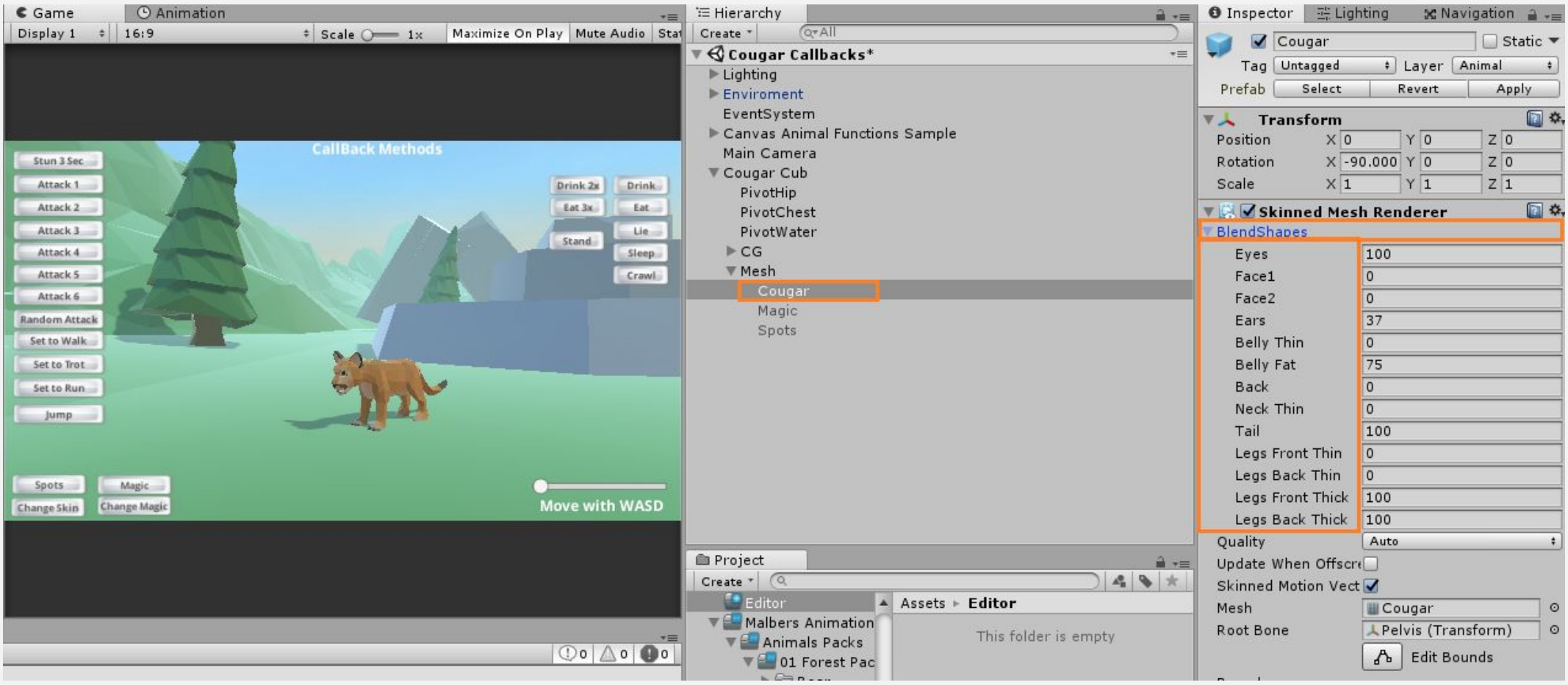


4. How can I change the Blend Shapes using a UI Slider

The ways of using the **BlendShape** script is using the methods:

```
SetBlendShape(string name, float value) or
SetBlendShape(int index, float value)
```

You can find the BlendShape names on the Cougar Mesh:



Make an extra script to use it as a bridge between a UI SLIDER and the BlendShapes script. (I called Pepe because why not 😁 but you should give it a proper name).

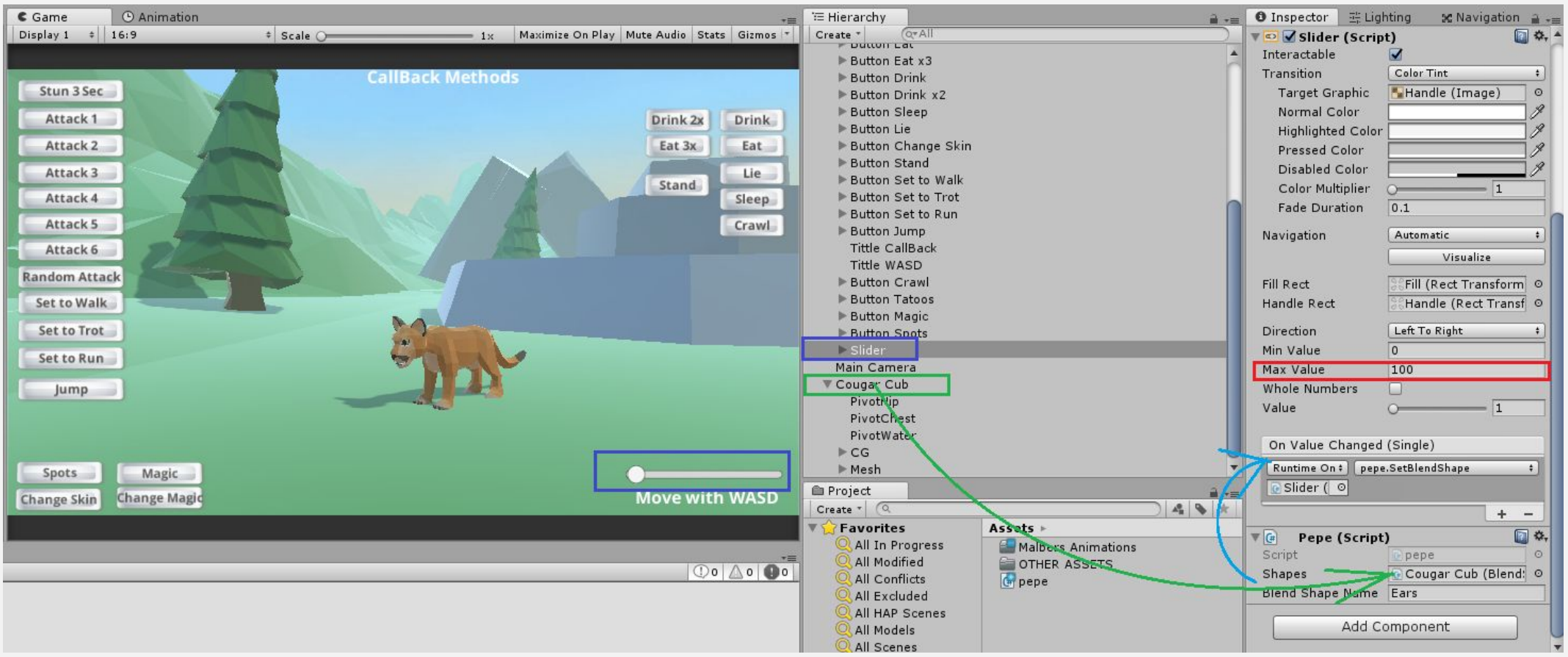
```
using MalbersAnimations.Utilities;
using UnityEngine;

public class pepe : MonoBehaviour
{

    public BlendShapes shapes;      //Rference of the BlendShape Script
    public string BlendShapeName;   //Name of the BlendShape you want to change

    //This method is the one you will link to the UI slider
    public virtual void SetBlendShape(float value)
    {
        shapes.SetBlendShape(BlendShapeName, value);
    }
}
```

Then use that script on the **UI SLIDER**



- Set the **Blend Shape Name** you want to change ....in this case I will use "Ears"
- Drag the **Animal** to the **Shapes** parameter on the **New Script** (Pepe)...
- On the **Slider** component set the **Max Value** to **100** (which is the Max value on the BlendShapes)
- Add new listener to the **OnValueChanged**(Single) Event with the method **SetBlendShape** created on the **New Script** .

and that's it!

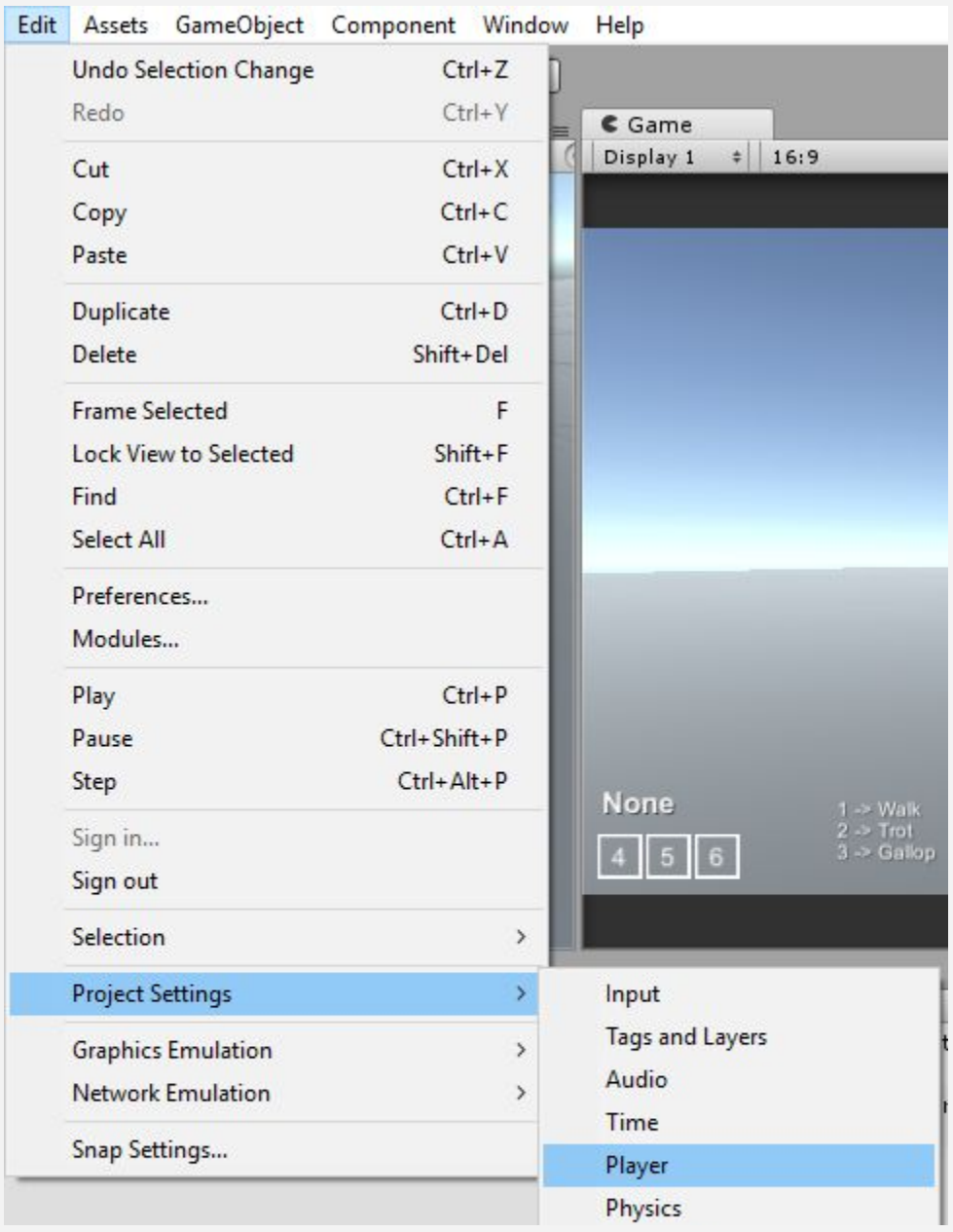
Integrations

Rewired

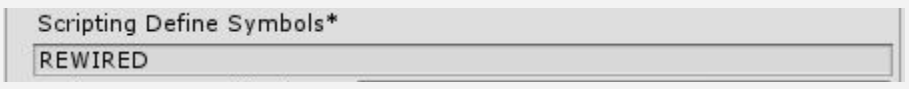
Asset location: https://www.assetstore.unity3d.com/en/#!/content/21676?aid=1100lHT6

To enable Rewired with Malbers Assets

1. Go to **Edit->Project Settings -> Player**



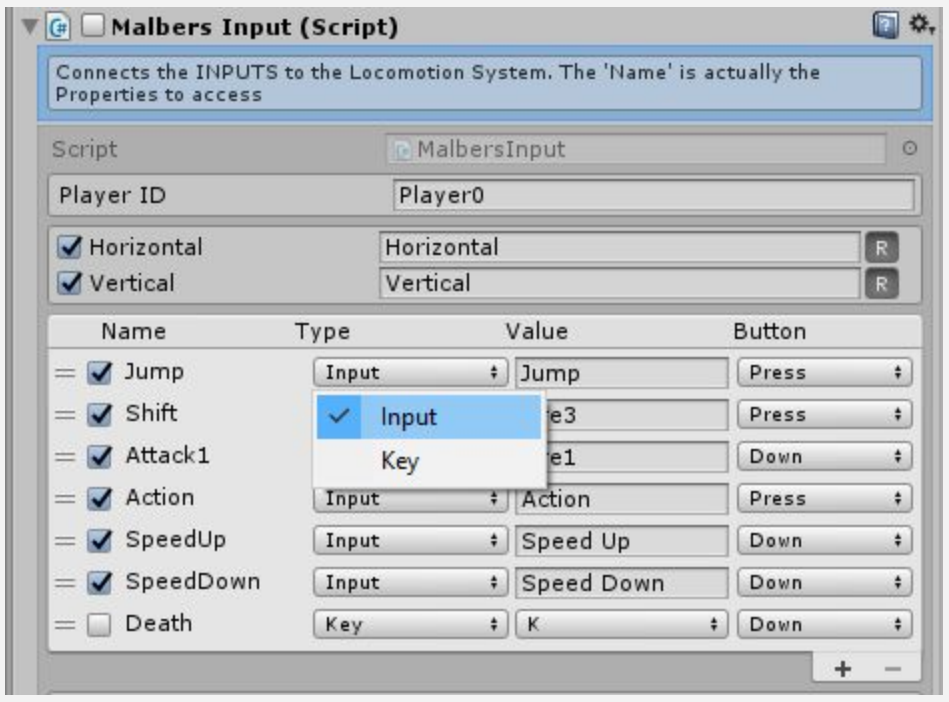
2. Add "**REWIRED**" to the Scripting Define Symbol*



Once done that a new **string** parameter will show up on every script that requires Input… such as:
MalbersInput, MFreeLookCamera, Rider3rdPerson… etc

This parameter is the **Player ID** for Rewired.

3. Change all the types from "**KEY**" to "**INPUT**" on all the Inputs Parameters on every script to complete the connection…



4. Download from the [Integration Drive Folder] the Rewired Input Source to have a Base Input Manager with all the mappings… (you can always change it with yours). you can find the prefab on this location:

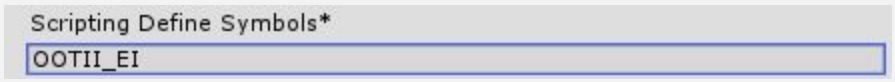
5. Drop the Prefab to your scene … and that's it.

   This prefab has the basic **Actions** already setted.. but you can always add/modify them.

Easy Input

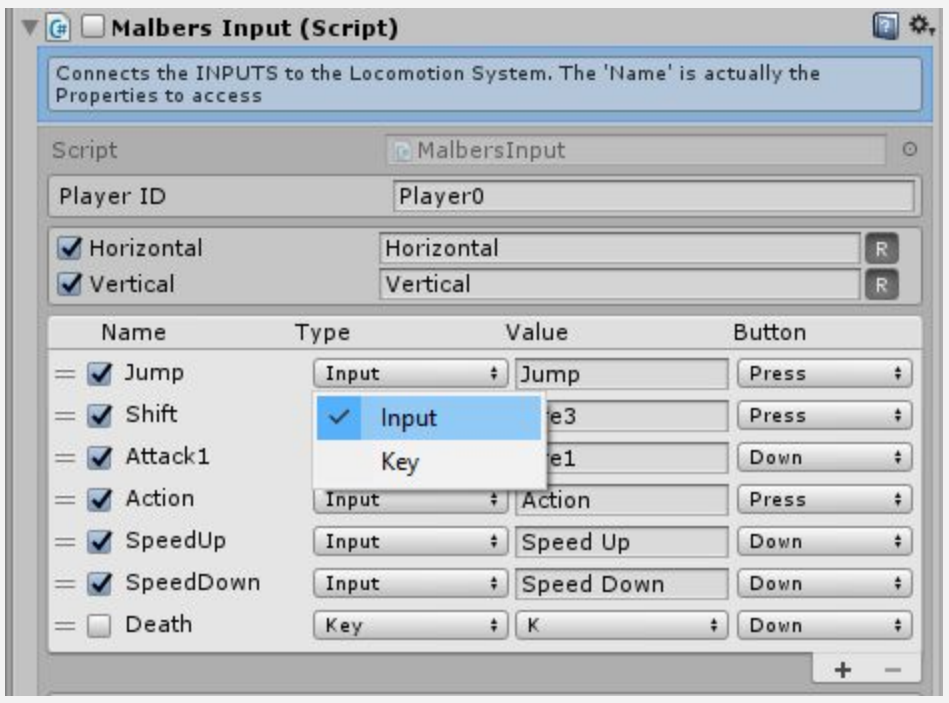
Asset location: https://www.assetstore.unity3d.com/en/#!/content/15296?aid=1100lHT6

By Installing Easy Input a custom Scripting Define Symbol* will be added automatically (**OOTII_EI)**

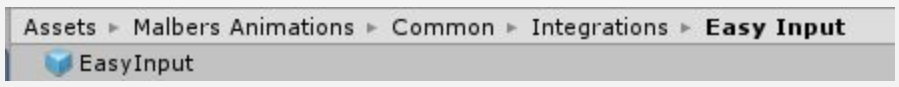


This will enable the code for using this asset.

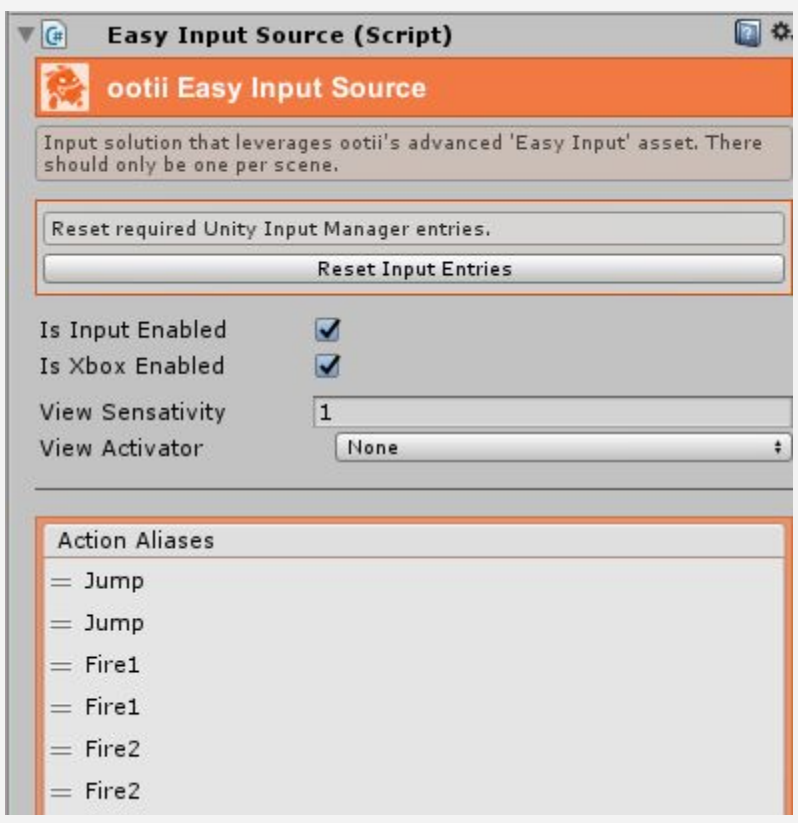
- Change all the types from "**KEY**" to "**INPUT**" on on every script that have Inputs Parameters.

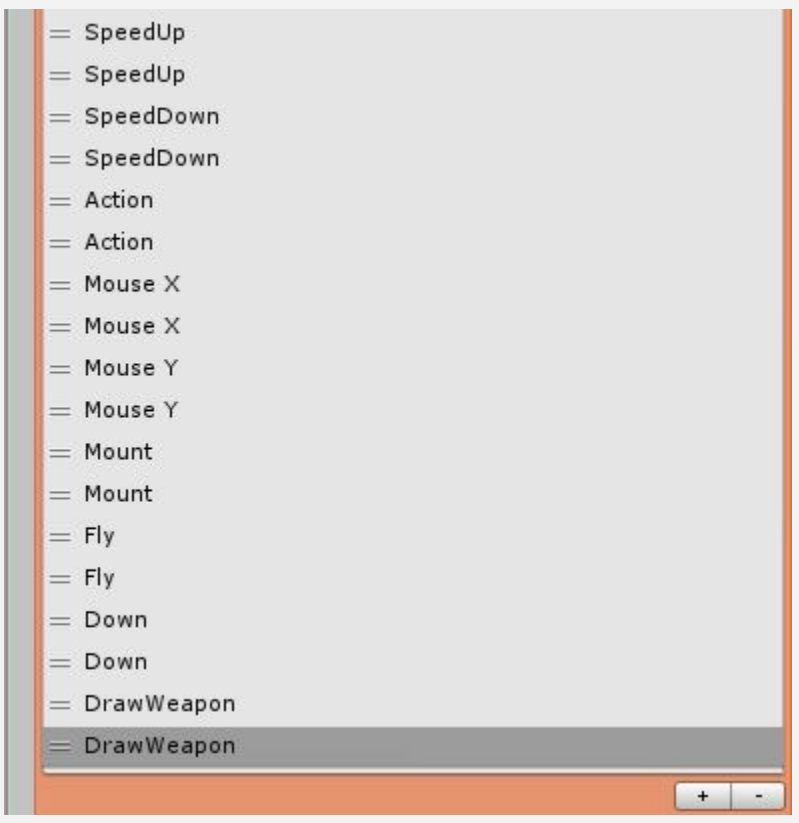


6. Download from the Integration Drive Folder the Easy Input Source to have a Input Manager with all the mappings… (you can always change it for yours). you can find the prefab on this location:



7. Drop the Prefab to your scene … and that's it.
   This prefab has the basic **Actions** already setted to work with my assets.. but you can always add/modify them.





ICE Creature Control (NOT DONE YET)

(Fill it)

Invector's Templates + HAP

Youtube:
[Invector + HAP 3 Part 1: Riding]
[Invector + HAP3 Part 2: Combat]

(Fill it)

Ootii's Motion Controller

(Fill it)

Opsive UFPS 1

(Fill it)

Opsive TCP 1

You can contact me via malbers.shark87@gmail.com. I’m always happy to help.

Cheers
Malbers!