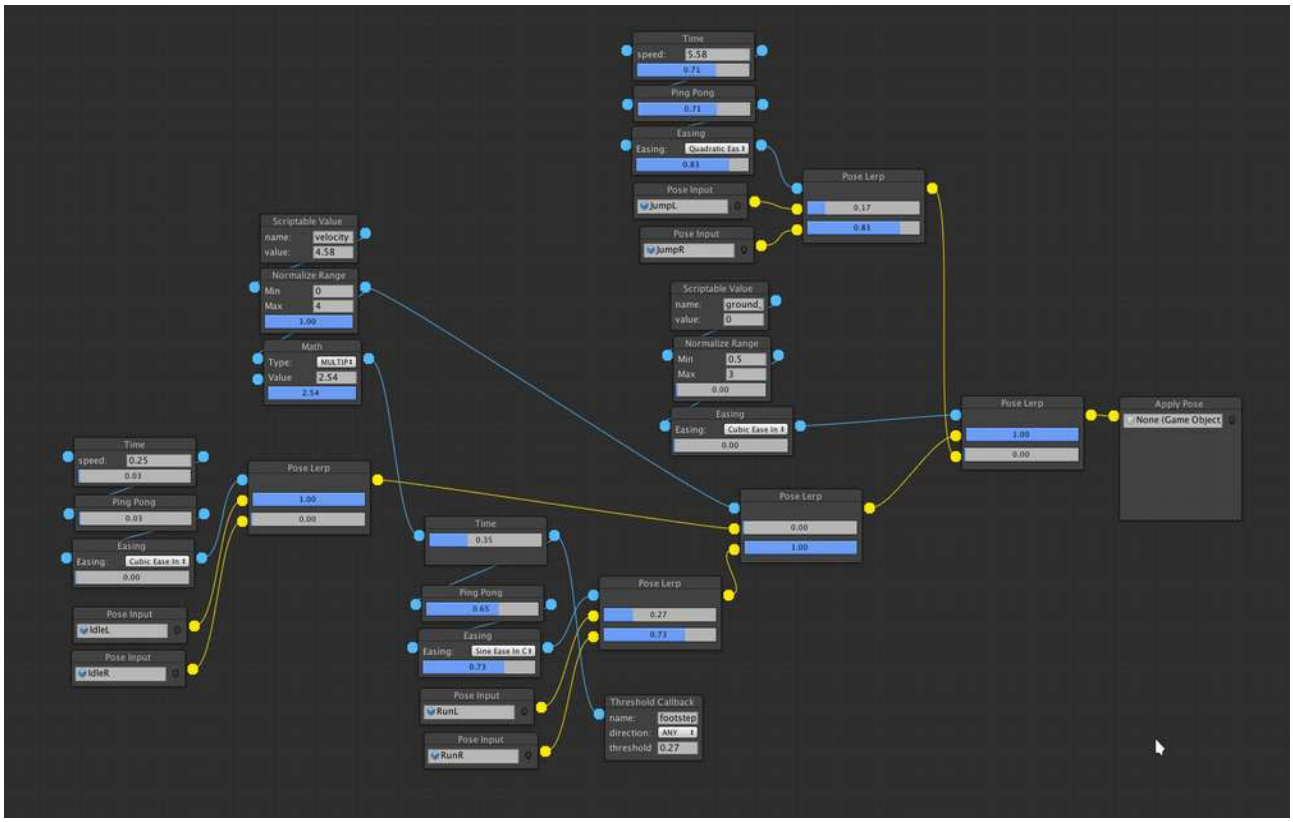
PANIMATION STUDIO



WELCOME TO PANIMATION STUDIO

Panimation Studio is a Unity3D tool for creating procedural animations.

Panimation Studio is a node-based editor which interpolates so called poses of your model to create various animations which seamlessly blend together. This allows you to create complex procedural animations in only a few minutes, directly in Unity3D, without the need of using a 3D modeling software.

BENEFITS

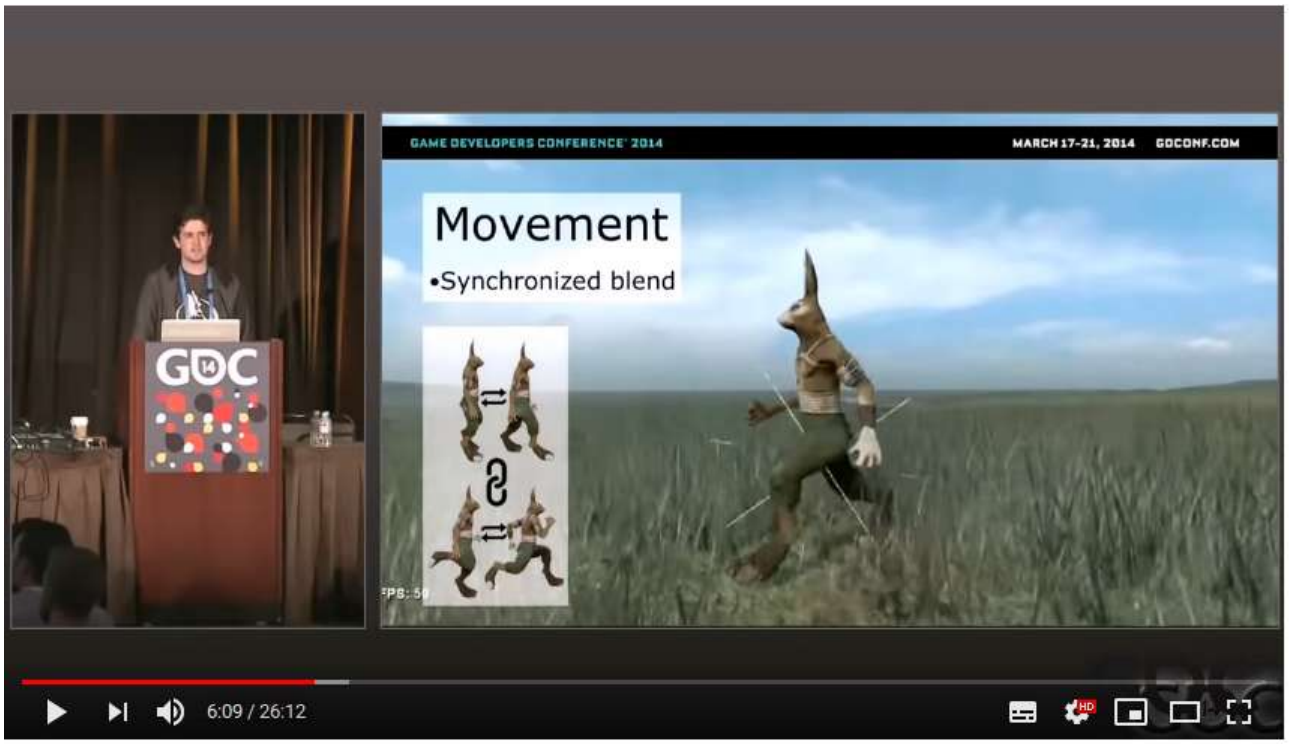
- create complex animations without the need of using a 3D modeling software
- no need for motion capturing or cumbersome key frame animation editing - only a few poses are needed to breathe life into your character
- procedural animations let your character dynamically react to the environment
- adjust your poses in real time – while the animation is running – to precisely fine tune your animations

WHERE DID THE INSPIRATION FOR PANIMATION STUDIO COME FROM?

The inspiration for Panimation Studio did come from David Rosen. He used this technique to create the amazing game Overgrowth.

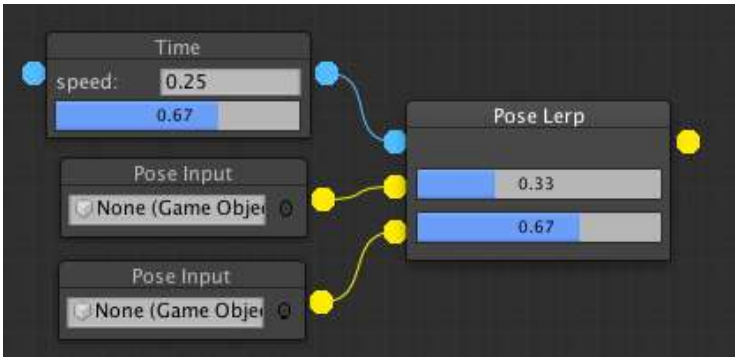
http://www.wolfire.com/overgrowth

Please have a look at his GDC talk to get some more ideas on what you can achieve by using Panimation Studio:



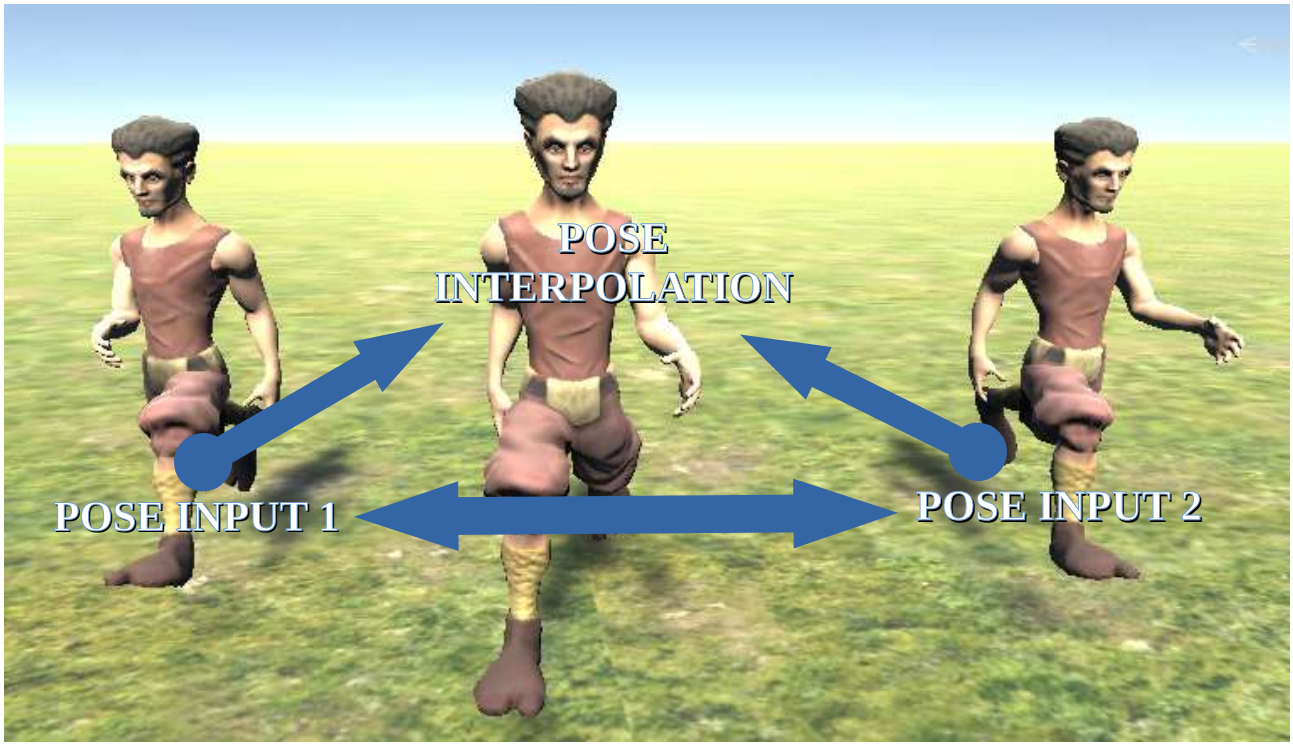
https://www.youtube.com/watch?v=LNidsMesxSE

MAIN CONCEPT



Yellow in- and outputs are poses.

Blue in- and outputs are signals.



A pose is like a static snapshot of your character at an extreme posture. In the example above, we have "pose input 1" where the left leg is in the front, and we have "pose input 2" where the right leg is in the front. Interpolating over time between the two poses leads to an animation where the character is running.

A signal is simply a single floating point number. Most of the time (but not always) a signal has a value between zero and one. In the example above we have two "Pose Input" nodes, which are interpolated in the "Pose Lerp" node, depending on the signal of the "Time" node. If the signal has a value of 0, then "pose input 1" will be used. A value of 1 means "pose input 2" will be used. A value of 0.5 leads to an exact interpolation between the two poses.

EDITOR OVERVIEW

- Right click your mouse to create a new node
- Right click your mouse on a node to delete it
- Left click your mouse on an in-/output, then left click again on another in-/output to create a connection
- Right click with your mouse on an input to delete a connection
- Click and hold your left mouse button to drag a node
- Click and hold your middle mouse button to move all nodes

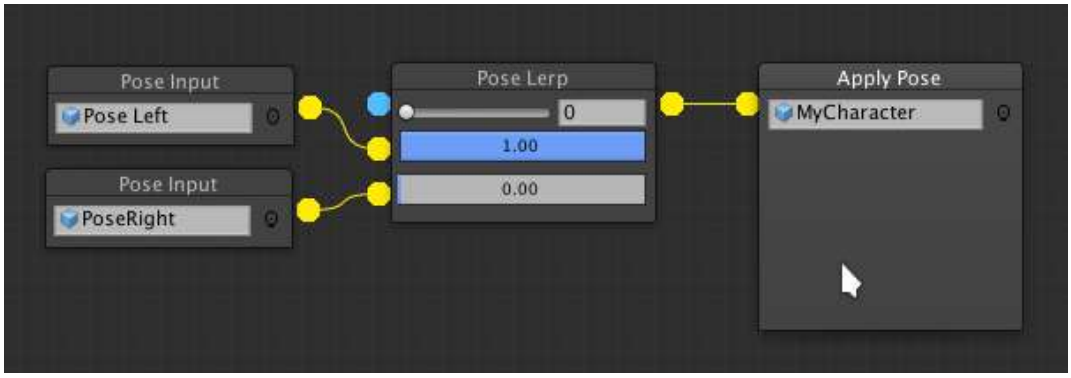
QUICK START

1. Import Panimation Studio and the model you want to animate into Unity
2. Create a new scene and drag your model into the scene, then name it "MyCharacter". If your model has an Animator component attached, you can safely remove it as it is not needed.
3. Drag and drop the "MyCharacter" game object from the scene into your project folder to create a prefab. Name the prefab "Standard Pose".
4. Duplicate the "Standard Pose" prefab and name the new prefab "Pose Left".
5. Duplicate the "Standard Pose" prefab and name the new prefab "Pose Right".
6. Double click "Pose Left" to open the prefab edit mode[1]. In it's hierarchy, search for the upper left leg transform and rotate it as you wish.
7. Double click "Pose Right" to open the other prefab in prefab edit mode. In it's hierarchy, search for the upper right leg transform and rotate it as you wish.
8. Ok, now we have our two poses set up and we are ready to create the first panimation. To do so, right click your project folder, then choose Create → Panimation
9. Double click the new panimation file to open the panimation editor.
10. In the editor, you will see a node called "Apply Pose". This node is always present and can not be deleted. Drag and drop the "MyCharacter" scene game object into the corresponding field of the "Apply Pose" node.

    Attention: *each time you restart unity (or reopen the panimation editor), the scene game object has to be reassigned to the "Apply Pose" node. This is not a bug and is caused by technical reasons on how unity handles serialized objects.*
11. Right click the editor and create a new "Pose Input" node. Drag and drop the "Pose Left" prefab into this node.
12. Right click the editor and once more to create a new "Pose Input" node. Drag and drop the "Pose Right" prefab into this node.
13. Right click the editor and create a "Pose Lerp" node.
14. Connect the yellow dot on the right side of each "Pose Input" node to the yellow dots on the left side of the "Pose Lerp" node.
15. Connect the yellow dot on the right side of the "Pose Lerp" node to the yellow dot on the left side of the "Apply Pose" node.

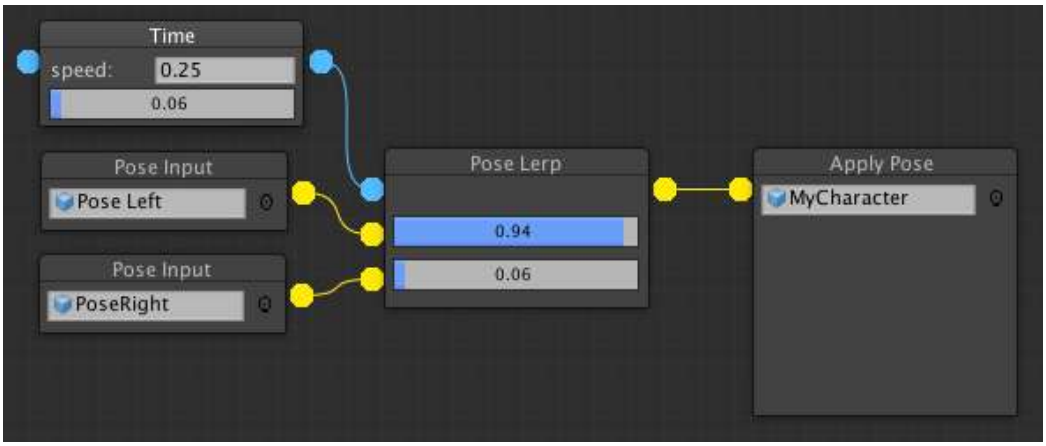
1 Prefab edit mode was introduced in Unity version 2018.3 (see https://docs.unity3d.com/Manual/EditingInPrefabMode.html). It has the huge advantage that you can manipulate your pose in real time, while the animation is running. If you are using a version below, you have to drag and drop your prefab into the scene, then manipulate the pose and then select assign to apply the changes to the animation.

16. It should now look the same as in the following image:



17. (OPTIONAL) Play around with the slider on the "Pose Lerp" node. You should then see the "MyCharacter" game object interpolating between the two poses in the scene view.
18. Right click the editor and create a "Time" node.
19. Connect the blue dot on the right side of the "Time" node to the blue dot of the "Pose Lerp" node. The "MyCharacter" scene game object should then continuously interpolate between the two poses. If you want, you can play around with the speed value of the "Time" node and see how it influences the animation.

    Here is how the final result should look like:



Congratulations, you have successfully created your first simple panimation!

Of course, the current animation doesn't look very pleasing. There are a lot of things we can do to improve it. Please continue with the next chapter to learn how to improve the animation.

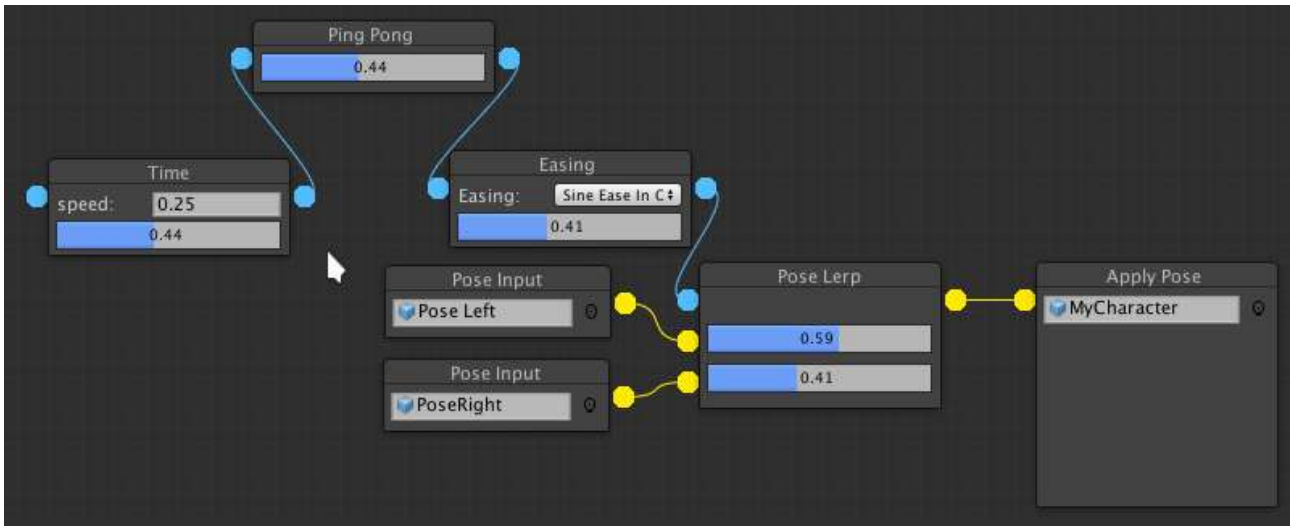
Furthermore, the animation is currently only running in the editor. As soon as you start the play mode, the "MyCharacter" scene game object doesn't animate anymore until you leave play mode again. To be able to run the animation in play mode, a few additional steps are required, which you will learn in the chapter "Quick Start - Play

Mode".

QUICK START – IMPROVING THE ANIMATION

1. The "Time" node creates a signal which smoothly transitions from 0 to 1 and then jumps back to 0. This jump is causing an ugly effect in our animation. To overcome this issue, create a "Ping Pong" node and put it between the "Time" and "Pose Lerp" node. The animation now continuously transitions back and forth.
2. The two poses are interpolated linearly. Linear interpolations don't look natural, so let's change this. Create an "Easing" node and put it between the "Ping Pong" and "Pose Lerp" node. The default value for the "Easing" node is linear, so this is why the animation still looks the same. Change the easing parameter of the new node from "Linear" to "Sine Ease In Out". Now it looks much better, right? Take your time and experiment with different easing values to see how they affect the animation.

This is it, well done! The final result should look like this:



QUICK START – PLAY MODE

If you start the play mode in Unity, the animation stops. To be able to run the animation in play mode, we have to attach a "Panimation Controller" component and write a little script. Don't be afraid, it's just a few lines of code. So let's get started:

1. Add the "Panimation Controller" component to your "MyCharacter" scene game object.
2. Drag and drop your panimation file (created at step 8 of the quick start guide) to the corresponding slot of the "Panimation Controller" component.
3. Create a new c# mono behaviour script, name it "MyCharacterBehaviour.cs" and add it as a component to your "MyCharacter" scene game object.
4. Copy the following script:

```
public class MyCharacterBehaviour : MonoBehaviour
{
    private PanimationController _panimationController;

    private void Awake()
    {
        _panimationController = GetComponent<PanimationController>();
    }

    private void Update()
    {
        _panimationController.UpdatePose(Time.deltaTime);
    }
}
```

   In the Awake() function, we get a reference to the "Panimation Controller" component we added in step 1. Then in the Update() function, we simply call the components UpdatePose() method and pass Time.deltaTime as an argument. This call will then update and apply the animation to your character in play mode.

Well done, you are now able to run your animation in play mode as well.

NODE TYPES

SCRIPTABLE VALUE

This node allows you to input a signal trough scripting.

E.g. if you have a running animation for your character, then you can use this node to input the characters rigidbody velocity magnitude. This allows you to then adapt the speed of the running animation to the velocity. Please have a look at the Demo #2 in the Panimation Studio _DEMOS folder to see how it's done.

To assign a value, you must first assign a unique name to the name field of the node. Then in your MyCharacterBehaviour component (see chapter Quick Start – Play Mode), you have to call the SetScriptableInput() method and pass the name and the value as arguments (right before calling UpdatePose()) as follows:

```
private void Update()
{
    _panimationController.SetScriptableInput( nodeName: "velocity",  value: 42f);

    _panimationController.UpdatePose(Time.deltaTime);
}
```

NORMALIZE RANGE

This node manipulates an incoming signal and then outputs the manipulated signal. The manipulation is identical to Unity's Mathf.InverseLerp() method (see https://docs.unity3d.com/ScriptReference/Mathf.InverseLerp.html).

E.g. you have a running animation for you character and a "Scriptable Value" node which inputs it's velocity, where the velocity may go from 0 to a value of let's say 5 (which is the maximum velocity your character can reach). Having a signal between 0 and 5 is not preferable to be used to interpolate poses. By attaching a "Normalize Range" node and set it's min parameter to 0 and it's max parameter to 5, it will output a signal between 0 and 1, which can then be used to interpolate an idle and a running pose, or set the speed of the running animation and so on.

Pro tip: attach two "Normalize Range" nodes to a "Time" node. Set the min max parameters of the first node to 0 and 0.5, and the ones of the second node to 0.5 and 1. This will then lead to the first node signal going from 0 to 1, and as soon as it reaches 1, the signal of the second node will start going from 0 to 1. This trick allows you to create sequential animations.

THRESHOLD CALLBACK

This node invokes an action (assigned by a script) each time the incoming signal crosses a threshold. You can set the action to be invoked only when the signal is increasing or decreasing or in both cases.

You can add a listener in the Start() function (never do this in the Awake() function, as it will throw an error) as follows:

```
private void Start()
{
    _panimationController.AddThresholdCallbackListener( nodeName: "footstep",  action: () => FootstepAudioSource.Play());
}
```

E.g. if you are interpolating two running poses, you can attach this node to the signal and set a threshold of 0.5. Then in your script, assign the Play() function of a footstep AudioSource as a listener to the node (see image above). Your running animation will then play the footstep audio each time the feet touches the ground.

TRIGGER

A trigger node can be de-/activated by script. It outputs a signal, which when activated linearly increases from 0 to a value of 1 and then stays at 1 until the node is deactivated. Vice versa for deactivating the node, where the signal decreases from 1 to 0. You can also adjust the transition speed. To de-/activate the node, give it a unique name and then call the following method by script:

```
_panimationController.SetTrigger( nodeName: "crouch",  isActive: true);
```

The trigger node is useful to change the state of your character. E.g. it can be used to enter a crouching mode, where your character then blends from a standing/idle animation to a crouching animation. Please have a look at the DEMO#2 example to see this in action.

POSE INPUT

You can assign a pose to this node which will then be output. The pose is a game object prefab of your model.

Important: The assigned game object or one of it's children must have a SkinnedMeshRenderer component attached. Also, please keep in mind you can only assign prefabs from your project folder. Scene objects are not allowed.

POSE LERP

This node interpolates/blends two incoming poses depending on an incoming signal and then outputs the result.

APPLY POSE

This is the final node of every animation. It applies the final pose to your model. This node must exist exactly once, therefore it can't be created or deleted.

In play mode, the "Panimation Controller" component will automatically assign the scene object - which the component is attached to - to this node.

In editor mode, you have to manually assign a scene object of your model to this node. This will then allow you to preview your animation in the editor. In contrast to the "Pose Input" node, only scene objects are allowed.

Important: Each time you change the scene, restart Unity or reopen the Panimation editor, a scene object of your model has to be reassigned to this node - if you still want to be able to preview your animation in editor mode.

TIME

This node produces a linear signal that goes from 0 to 1 and then jumps back again to 0. You can either set a constant speed value in the editor or you can attach another signal which then acts as the speed value.

EASING

This node manipulates an incoming signal and then outputs the manipulated signal. This is useful to change a linear signal into a more natural looking signal like e.g. sinus curves. Please have a look here to see how the different easing types manipulate your signal: https://wiki.sparrow-framework.org/_media/manual/transitions.png

CURVE

This node let's you create an animation curve, which manipulates an incoming signal and then outputs the mapped value.

PING PONG

This node changes a signal which permanently increases and then jumps back again to a lower value (like e.g. the "Time" node), into a signal which constantly increases and then decreases instead of jumping.

MATH

This node adds or multiplies a constant value (or another signal) to an incoming signal and then outputs the result.