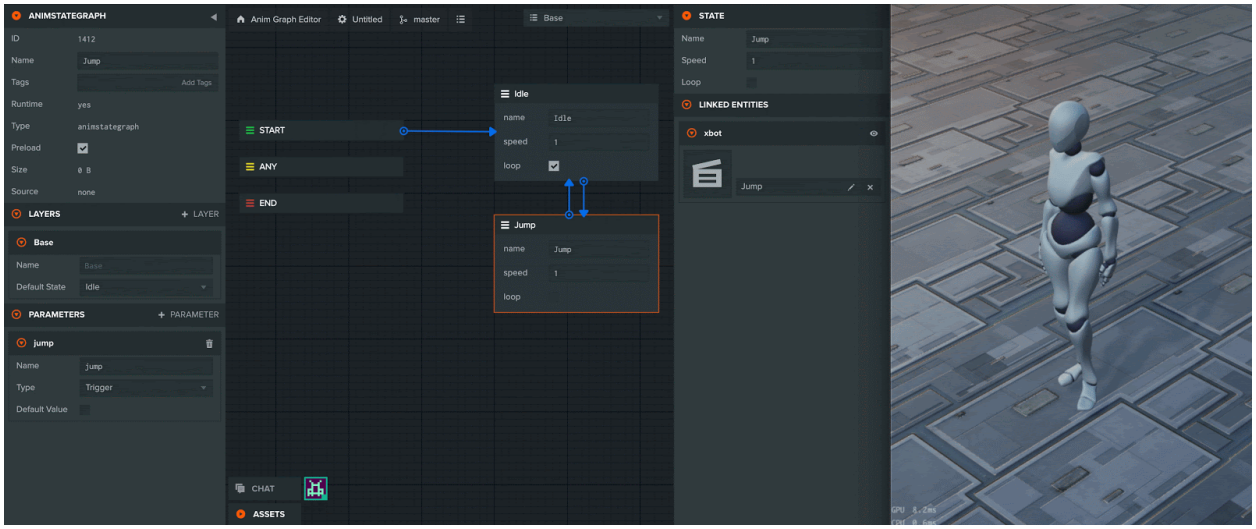
Animation - User Manual



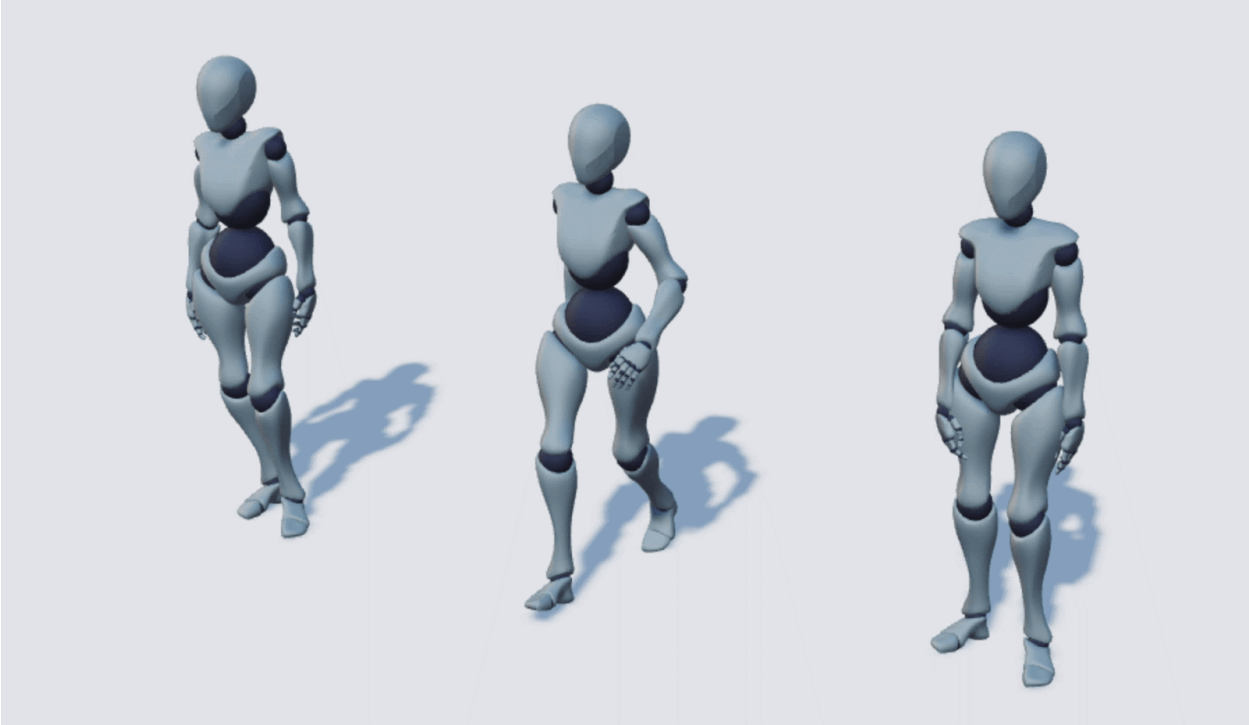
PlayCanvas provides a powerful state-based animation system which can be used to animate any entity model in the scene, including those of your characters. Users can work with any of their .FBX animation assets. These can be organised using animation state machines to easily control the animated behaviour of scene models at runtime.

System Overview

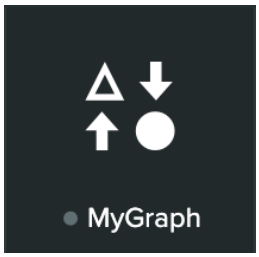
The animation system touches on three main areas of the PlayCanvas system. This section will walk through how these areas can be used together to create complex animation behaviour for your models. The following sections of the animation user manual then will explore each area in more detail.

Animating in PlayCanvas

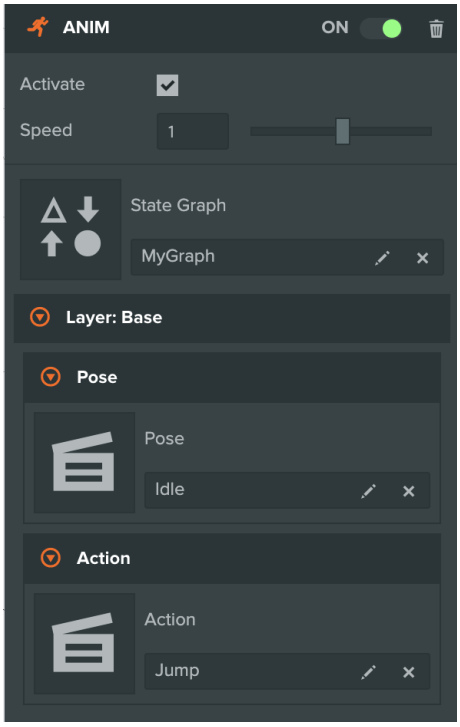
In order to begin animating a PlayCanvas entity, you must have a set of animation assets available and imported into your PlayCanvas project that will drive the animation of a given model you wish to animate. For example a humanoid character may have a set of animations; Idle, Walk, Jump.



Organising these animations into a single animation system will create a simple locomotion system for that character. The way in which this is achieved in PlayCanvas is through the use of an `animstategraph` asset. These assets can be thought of as state machines for an entity's animation behaviour. With each state in this asset relating to an animation, the state machine can be set up to define the complex animated behaviour of an entity's model. This includes defining when the system should stop one animation and start another and how the transition between these animations should be blended.

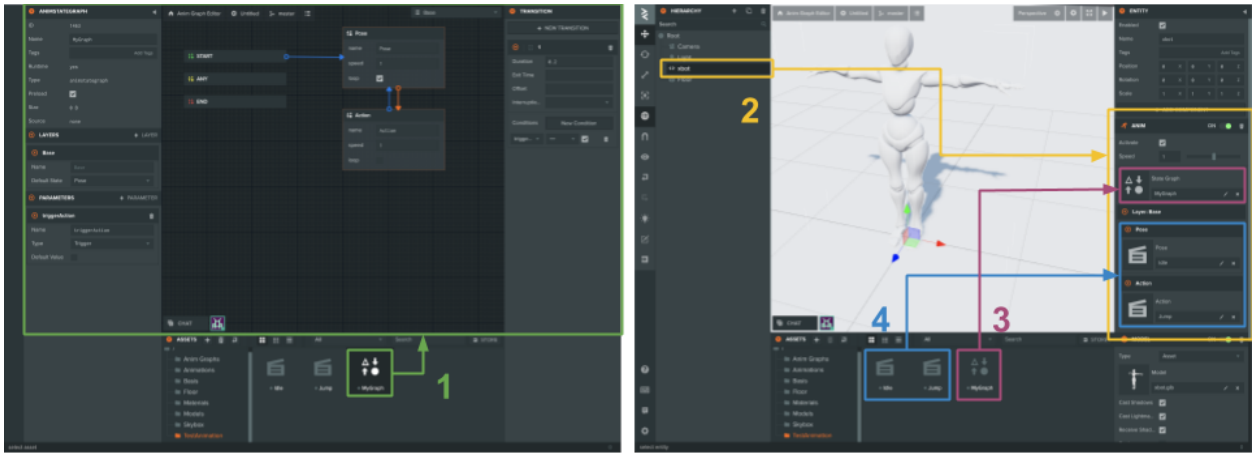


The anim component is then used to assign an `animstategraph` asset to a particular entity in your scene. Once an entity has been assigned an `animstategraph` asset, each state in the graph can have an actual animation asset assigned to it. Once all states have been assigned animations, the anim component will become playable. At this point the animation system is complete and the defined animation behaviour will be viewable in the PlayCanvas launcher.



Walkthrough

Follow these four steps to set up an animation on an entity in PlayCanvas:



1 - Create an animstategraph asset in the assets panel and edit your state graph
2 - Select an entity in the hierarchy and add an anim component to it
3 - Link the animstategraph asset you created to the anim component
4 - Link the desired animation assets to the component

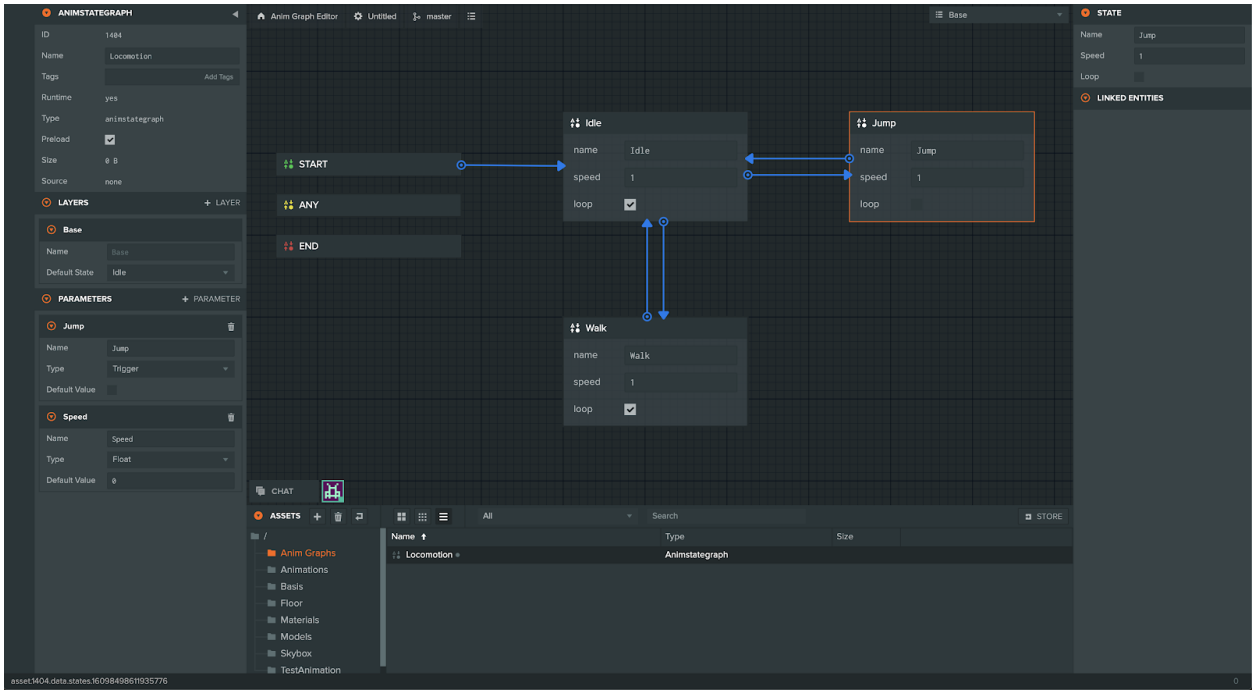
Animstategraph Assets

These assets are used to organise a set of different animation states, which are all the various ways in which a model might animate. This `animstategraph` asset is used to define the logic of how animations should behave, while the entity and it's anim component are responsible for the visuals of each state animation. It can be used to define each of these animation states, determine when each state should play and how states transition and therefore blend between one another. The `animstategraph` assets do not store or link to any real animation assets themselves, but rather act as a template for how animation assets should be organised. Actual animation assets are linked to the `animstategraph`'s animation states through the Anim Component[LINK].

The system was designed so that a single `animstategraph` can be used on many different entities, each with their own set of animation assets. An example being an `animstategraph` asset which manages the animations of humanoid character locomotion. This single asset could be used on a human entity, an elf entity and a dwarf entity. Each of these entities would be able to link their own character animation assets, all the while maintaining the same animation behaviour as each other.

These assets are therefore state machines for a model's animation behaviour and they control the flow of animation sequences over the lifecycle of an entity. A simple `animstategraph` asset used to define the behaviour of a wheel may define only two animation states; static and spinning. This asset can be defined to control when the wheel starts and stops spinning, for how long it will spin, the speed of the wheel spin and how sharply it starts / stops spinning. More advanced assets can be used to combine a multitude of animation states to create complex humanoid character animation behaviour.

When selecting an `animstategraph` asset in the editors asset panel, you'll open up the anim state graph editor view:



Within this view you can edit your animation state graph. The following sections will highlight how different elements of the `animstategraph` asset can be used to define specific animation behaviour.

States

In essence, states are used to specify which animations should play at a given point in time. An anim state graph can only be in one of these states at a given time.

There are four types of states present in state graphs. Animation states, along with the START state, END state and ANY state. Only animation states can be created and deleted by the user and only these will be linked to animation assets. The other states are used to control the flow through the state machine.

Animation states

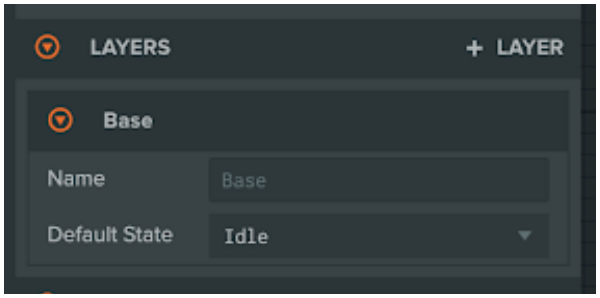
Animation states define a playable animation such as ‘Idle’, ‘Jump’ or ‘Walk’. New animation states can be created by right clicking on the blank canvas behind the state graph and selecting ‘Add new state’ from the menu. The editor will target your newly created state and show it’s inspector panel on the right hand side.

Within this inspector the following state variables can be modified:

Variable	**Description**
Name	The name that this state should be called by. This is used to find and edit and play states via script. Names must be unique per state graph layer.
Speed	The playback speed for animations that are linked to this state.
Loop	Whether animations linked to this state should loop during playback. If set to false the animation will pause on it's last keyframe until this state is exited.

START state

The START state is the entrypoint of every state graph. When an anim component begins playing its assigned anim state graph, it will first enter this state and transition directly to the default animation state it's connected to. This default animation state is called the 'Default State' and it can be assigned to an animation state via the layers panel here:



It is not possible to create any other transitions to or from the START state. It can only be entered again by transitioning to the END state.

END state

The end state marks an exit out of the current state graph. If your animation state is set up to transition to the END state, the system will move directly to the default animation state which is connected to the START state. This is useful to create cyclical flows through the graph while still laying out your graph in a linear fashion. It is not possible to create transitions from the END state to any other state. It will always transition directly to the START state.

ANY state

This state is used to create transitions which can be activated while the system is currently in any of the other animation states. Any transitions that trigger from this state will blend as if they had been connected directly from the currently active animation state. You can create transitions from the ANY state but not to it.

This is useful to set up transitions which you want to activate, no matter which state you're currently in. For example you could have a jump state which should be reachable from both an idle and walk state. Instead of setting up transitions from both the idle and walk states to the jump state, a transition can be set up between the ANY state and the jump state.

Checking State

It is likely that you'll want to check the currently active animation state of your graph at runtime. This is possible within a script through the use of the anim components api. A call to

`entity.anim.baseLayer.activeState` will return the name of the currently active state.

It is also possible to check the active states duration and elapsed time, as well as the previously active state. See the anim components [LINK] API for more information.

Transitions

Transitions define how the anim state graph can move from one animation state to another. They can be created by right clicking an animation state and selecting `Add transition` from the context menu.

By setting the variables of a given transition you can also control how the animations of the transitioning states will blend together.

The available transition variables are:

Variable	**Description**
Duration	The duration of the animation in seconds.
Exit Time	If provided, this transition will only be active for the exact frame during which the source states progress passes the time specified. Given as a normalised value of the source states duration. Values less than 1 will be checked every animation loop.
Offset	If provided, the destination state will begin playing its animation at this time. Given in normalised time* based on the destinations states duration. Must be between 0 and 1.
Interruption Source	Defines whether another transition can interrupt this one and which of the current or previous states transitions can do so.

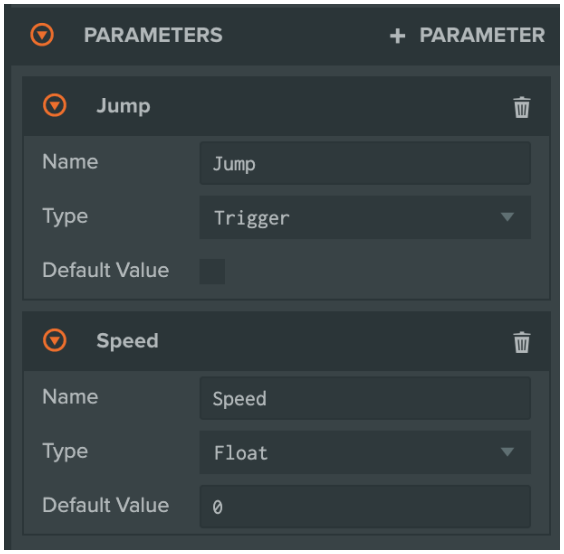
* Normalised time is a set of time values that have been rescaled so that all values within a given duration fall between 0 and 1.

It is possible to create multiple transitions between two animation states, which have different values and conditions set. The priority of these transitions can be reordered in the transition inspector after selecting a transitions arrow in the graph. The priority order determines which transition will be used by the state graph if multiple transitions have their conditions met.

Parameters

The parameters of an anim state graph are variables which are used to control the flow of animations during runtime. These variables can be accessed via scripts and set to new values at any time. They are then the way in which users can control the behaviour of an entity's animation during its lifecycle.

New parameters can be added to a state graph via the parameters panel on the left inspector:

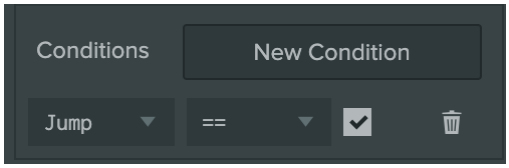


Each parameter has three variables which can be set:

Variable	**Description**
Name	The name that this parameter should be called by. This is used to find and set the parameter via script. Names must be unique per state graph.
Type	The type of variable that the parameter contains. One of; Boolean, Float, Integer or Trigger. The Trigger type acts as a Boolean but with the special property that it’s value is set back to false after it has been used to successfully activate a transition.
Default Value	The value of the parameters variable when the state graph launches.

The way in which they control the state graph is through the use of transition conditions. Each transition in the graph can have a list of conditions which define when a transition is usable by the system. A state will not be able to pass to another state through a given transition unless all of it’s conditions are met.

Each condition consists of a conditional statement which compares the current value of a parameters variable to the given value in the condition using the designated operator. For example the following condition:

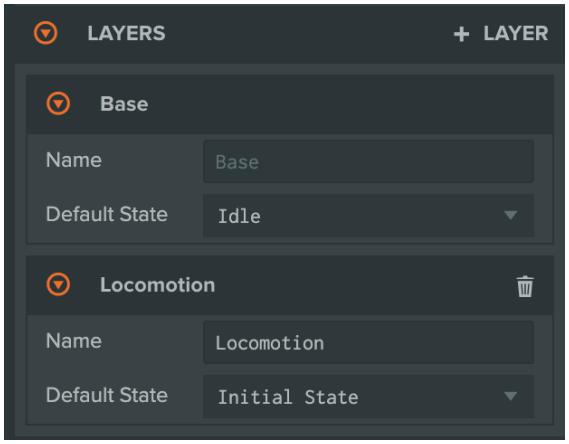


Can be used in the transition between the Idle and Jump animation states to ensure that a character only jumps when the ‘Jump’ parameter has been set to true via a script. This is made possible through the use of the anim component API [link], which provides getters and setters for all parameters that are associated with the current components state graph.

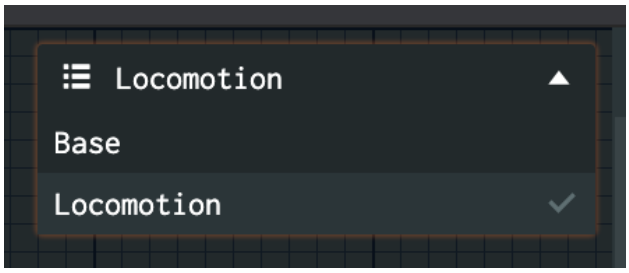
Layers

So far `animstategraph` assets have been discussed in the context of editing a single animation state graph. It may sometimes be necessary however to have the animations of a single model driven by multiple separate state graphs, each with their own defined behaviour. An example could be animating a main character's movement and locomotion on a single layer, which animating it’s facial expressions on a separate layer that’s driven by its own state graph and parameters.

When an `animstategraph` is created, it comes with a single base layer. This layer is not deletable and for many scenarios will be the only one necessary. However if you wish to create another layer you can do so by selecting the new layer button on the layers panel to the left of the state graph view:

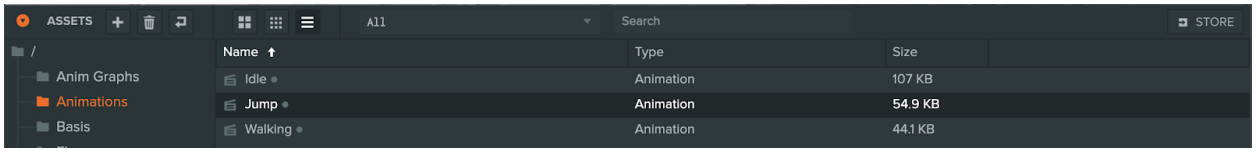


It is then possible to switch to editing this layer by selecting it from the layer select dropdown which is present at the top right of the graph view:



Tip: Layers animate a model in the order that they're created in the layers panel. Any animation values they set on a model will be overwritten by subsequent layers if they are animating the same bones.

Animation Assets



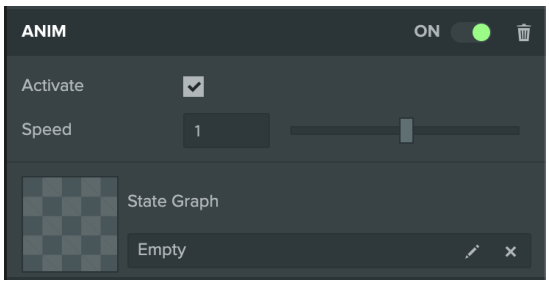
Animation assets are the animation keyframe data that's used to drive animations of a model in playcanvas. They are linked to an `animstategraph` asset via an entity's anim component.

The anim component currently supports animation assets that you import into a PlayCanvas project from .FBX files. It does not support legacy JSON animation assets. To use these animations with this system, the source .FBX files for these assets can be reimported into your project using the GLB asset import setting.

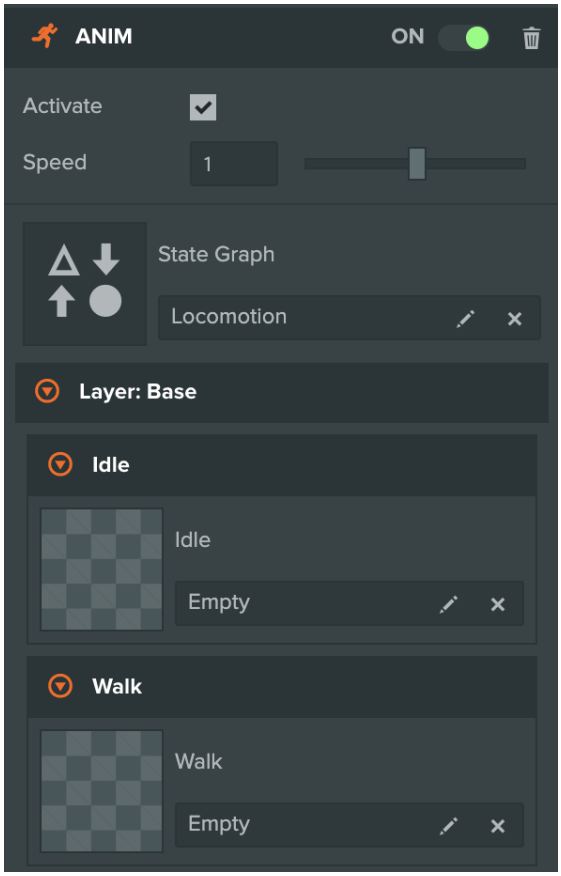
Anim Component

The anim component is used to connect an `animstategraph` asset and all of it's required animation assets to a single entity.

Below you can see the anim component after it has been added to a component. It shows an available slot for an `animstategraph` asset to be selected.



After selecting an `animstategraph` asset, the anim component will display a list of animation asset slots. There will be one slot for each animation state in every layer of the state graph asset. This is where actual animation data is connected to the previously created state graph. Multiple anim components can use the same `animstategraph` asset, each with their own set of animation assets.



After all animation state slots have been filled, the anim component will become playable. At this point the anim component can either be played via script by calling `entity.anim.play()` or if the `Activate` option is selected, it will play automatically upon the launch of the PlayCanvas project.

The anim component also offers the option to alter the speed of the animation playback. This speed will affect every animation within the state graph.