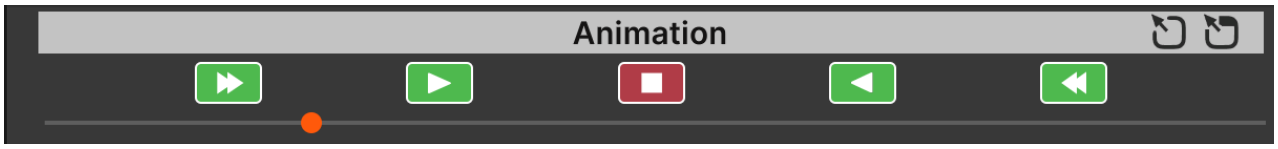
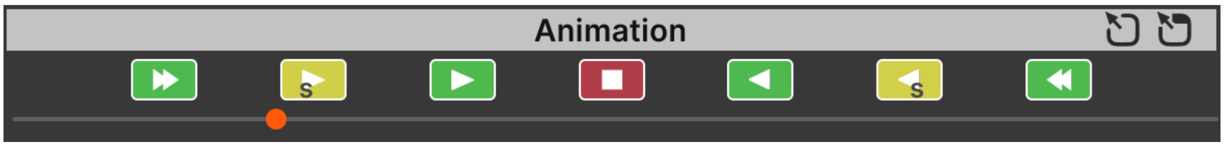
Easy Tweens Tutorial

Easy tweens package was created to simplify tween animation creation as much as possible. To create new animation start with adding a **TweenAnimation** component to any game object you like. Usually it is more convenient to create a separate GameObject and name it '…Animation' (e.g. ShowAnimation, HideAnimation, IdleAnimation etc.) but you're free to use it as you like, there are no restrictions.

You can create all animations in the edit mode and preview them without going into the play mode. At the very top-right you have 2 buttons to switch to window mode editing. First one will create a floating window that always will be on top of the Unity editor and could be dragged to any place on the screen. At the top of the **TweenAnimation** component you can see play control buttons.

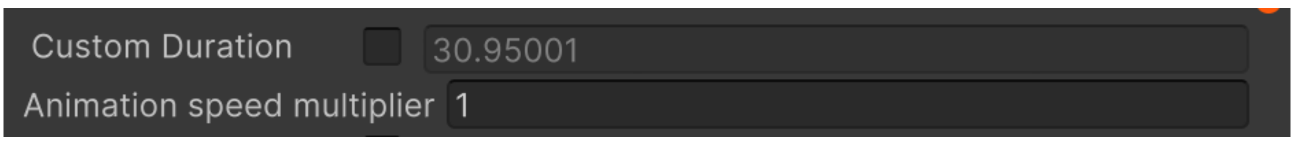


From the left to the right: Fast forward to end state(skip animation), play animated from start to end, stop currently playing animation, play animated from end to start, fast forward to the start state.



If you select any tween two additional buttons will appear. They will play only selected tweens so you can isolate and fine tune each part of your animation. That could be very handy if you have complex animation that consists of many different tweens.

Right below you can see an animation slider that you can manually drag to preview how animation evolves or just check certain time points in the animation or to just check how different tweens blend together in action.



Custom Duration 30.95001
Animation speed multiplier 1

Next up you have **Duration** and **animation speed multiplier** fields. By default duration is equal to the duration of the longest tween it contains(taking also delay into account). So if you have a tween with delay of 0.5 seconds and duration of 2 seconds minimal animation duration will be 2.5 seconds. But you can manually set duration to a higher value, you just have to toggle on Custom duration and the field will become editable. It can be useful in case of looped animation. If you want for example to play some accent animation on your button(for example bounce on purchase button) and you want to do it once in 3 seconds, you can set up a nice accent animation with a

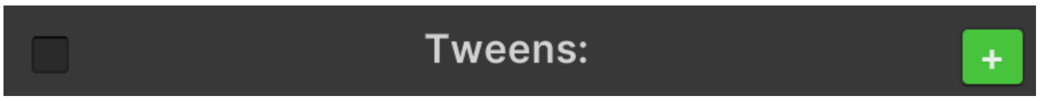
0.5 seconds duration and set total animation duration to 3.5. If you then loop this animation you will have you animation played with a 3 seconds delay between cycles.

If you toggle Custom duration off, animation duration will be automatically set to the minimum time needed for all tweens to be finished.

Speed multiplier field could be used in editor mode while testing your animation. Setting the multiplier to values below 1 allows you to see your animation in slow-mo and inspect if all transitions are well timed. Also sometimes it is very handy to speed-up/slow-down an animation in runtime depending on some game context(heart beating animation in the UI could be faster when a character has low HP for example).

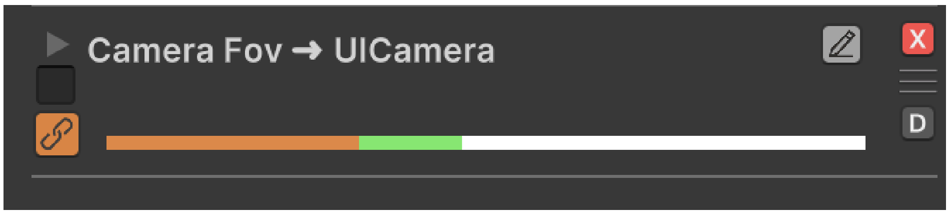
PlayOnAwake toggle speaks for itself, it sets if animation should be auto played when an object appears in the world or it should be triggered from the code.

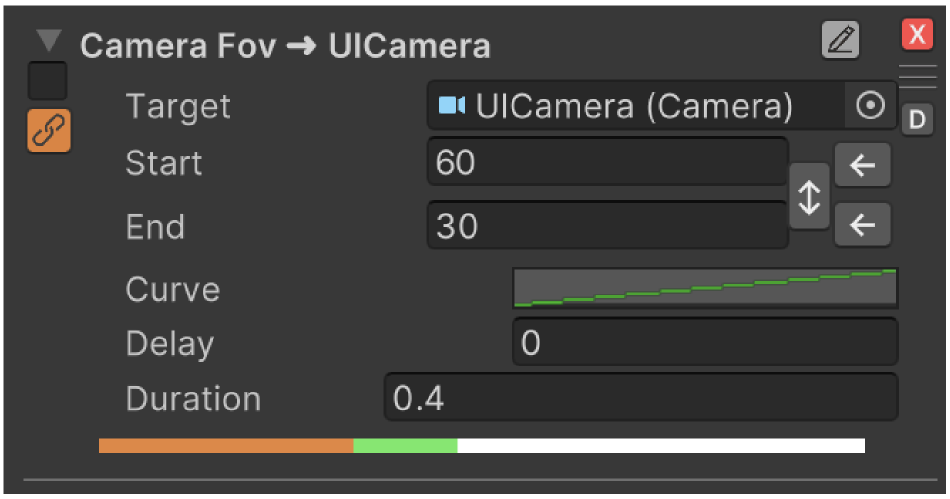
LoopType sets if animation should be looped or not. There is 3 option 'None' means one shot animation, Loop will restart animation from the start and PingPong will be playing animation backward and forward.



Next section contains real workers of the animation - tweens. At the right you have a button to add a new tween to the animation. Also you can just drag an object from the Hierarchy on top of 'Tweens' header and the menu will appear with all possible tweens for the dragged object. And at the left you can see a toggle selects/unselects all existing tweens. Each tween has its separate select/deselect toggle. That allows certain actions to be performed only on selected tweens.

All tweens have very similar editors. Lets see Camera Fov tween as an example as it is one of the most basic tweens. By default tween editor is in a folded state, if you click on the tween name it will unfold and you can inspect this tween properties. Grey button at the right from the name allows you to set a custom name to a tween, so you will navigate easier when the tween count grows.




Camera Fov ➔ UICamera

Target: UICamera (Camera)
Start: 60
End: 30
Curve
Delay: 0
Duration: 0.4

Now to the tween properties. At first it has a **Target**, that's the object which property we want to animate. Next we have **Start** and **End** values that sets property range along the animation. Easing is done through the animation **Curve**. Horizontal axis of the curve represents time(0 start of tween and 1 end of the tween) and vertical axis represents property value where 0 is the start value and 1 is the end value. But values on the animation curve are not clamped, so it could be higher than 1 and less than 0. If you have Start value 30 and End value 60 than value 2 on the curve vertical axis will set 90 to the camera for property (Value = 30 + (60 - 30) * 2).

Tip: You can set tween target even if tween editor is in folded state! Just drag and drop a target object from the Hierarchy window to the tween editor(it will be highlighted with green) and tween will automatically find component that fits and sets it as a target.

Buttons with arrows pointing at the **Start** and **End** property fields allow you to set the current object state as a start or end value. And a button with an arrow pointing up and down switches start and end values. So while animating any object you can modify it as it should be at the beginning of the animation, then go through all the tweens for this object and click the set current state button on Start property. Then modify the object as it should be at the end of the animation and click set current state buttons on the end property field. Then just add appropriate delay, duration and easing and you're done! (Ok, this depends on the animation and not always that easy..😜)

Also each tween has **Delay. Delay** normally calculated relative to the whole animation start or if tween linked to the other tween it waits for that tween to finish and only after this it starts its delay countdown. After that you have a **duration** field to set how much time it takes to play the tween from start to end. At the bottom you can see a custom progress bar that allows you to visualize how tweens are laid out in time. Orange part represents the time that tween waits until the linked tween will be finished(if tween is not linked to any other tween it won't be presented). White/green part represents tween active phase.

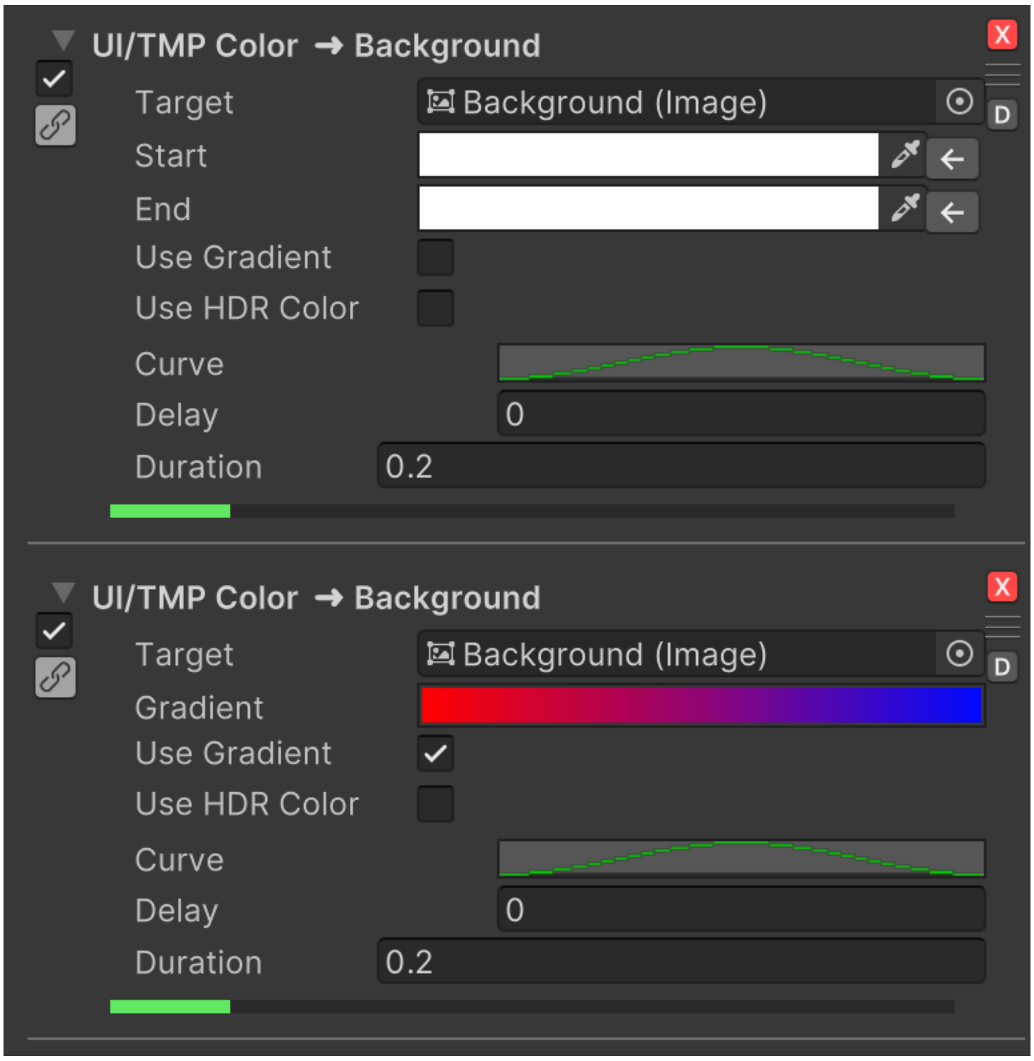
Control elements on the top right are(from top to bottom):

- Remove tween button
- Drag handle that allows you to rearrange tweens to more comfortably work on the big animations. Just press it and drag tween to the desired position
- Duplicate tween button(very often you have an objects with a similar animations that just need some minor adjustments)

Control elements on the left side are:

- Selection toggle that allows you to perform certain actions on the selected tweens only.
- Link control. If tween is not linked to any other tween it will have gray tint. To link the current tween to another one just press this control and drag it to the tween you want it to be linked to. When tween is linked to another one the link control will have orange tint and tween will wait until tween it linked to will be completely finished and only after that it starts its own animation. If you click on the control when it is linked a context menu will appear that allows you to check what tween it is linked to and also an option to unlink tween.

Some tweens have custom editors. For example any tween that changes the color has some additional properties.



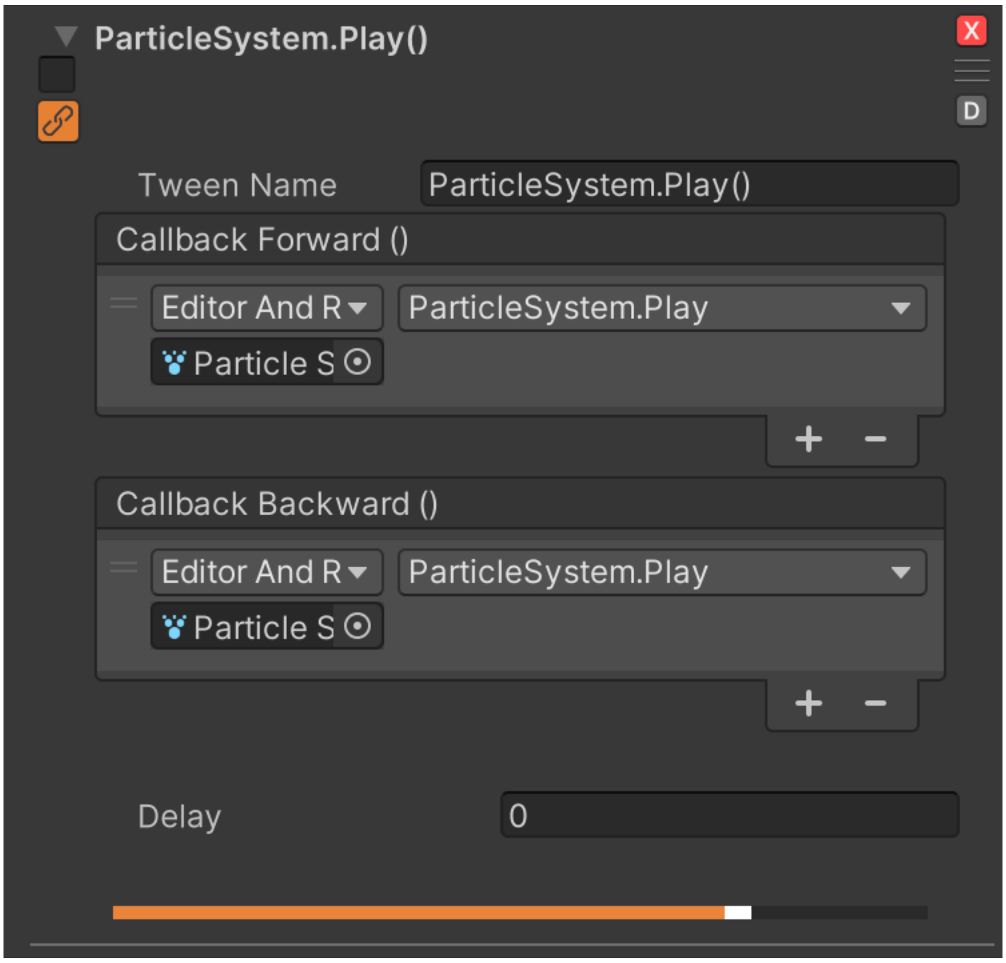
UI/TMP Color ➜ Background

Target	Background (Image)
Start	
End	
Use Gradient	
Use HDR Color	
Curve	
Delay	0
Duration	0.2

UI/TMP Color ➜ Background

Target	Background (Image)
Gradient	
Use Gradient	✓
Use HDR Color	
Curve	
Delay	0
Duration	0.2

You can choose to tween from the start to the end color which will be using Color.Lerp(). It could be perfectly fine when you change alpha value or tweening between shades of the same color. But if you change color HUE by a lot most likely you want to use a gradient for it, so in-between colors will look good. Also it is possible to use HDR colors and gradients, but of course target should support it.


ParticleSystem.Play()

Tween Name: ParticleSystem.Play()

Callback Forward ()

Editor And R | ParticleSystem.Play

Particle S

Callback Backward ()

Editor And R | ParticleSystem.Play

Particle S

Delay: 0

Now let's see a special one - **TweenCallback**. You can find it in Other->Callback.

Often during the animation we want certain callbacks to be called. Like playing a particle system when position tween finishes, or disable one game object and enable another. For this type of scenario you can add Callback tween to your animation.

In the tween name you can specify what this callback actually does to quickly distinguish different callbacks without unfolding and checking them. And you have 2 actions, Callback forward will be invoked when animation plays from start to end and its counterpart Callback backward will be invoked when animation is reversed. You can play animation reversed from code with the .PlayBackward() method call or it could be animation that is looped as PingPong so it goes full cycle forward and then full cycle backward. The reason behind this 2 calls nature is that this is not really tweening but direct one time method calls. So sometimes you want to perform some action on normal animation flow and when you go backward you want to undo that action. Also TweenCallback is a great candidate to be linked to another tween. Usually you want to call some function when some part of the animation finishes so you can just link callback tween to this animation end and you will be sure it will trigger exactly in time no matter how many changes you'll make to delays and durations during animation tuning.

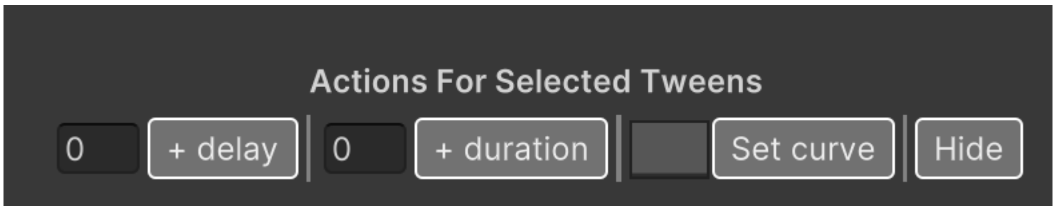
One issue that you can come across using **TweenCallback** is that not everything could be previewed in the edit mode. For example if you call Play() on the particle system in a callback you can only see particles spawning in play mode. Unity only shows particle preview when an object with a particle system is selected in the Hierarchy window. You can kind of hack this by having the Inspector window locked on the TweenAnimation component or open the animation editor in a separate window and then select Particle system in Hierarchy and hit play on

TweenAnimation. Also do not forget to select **Editor And Runtime** on a callback to see effect in edit mode.

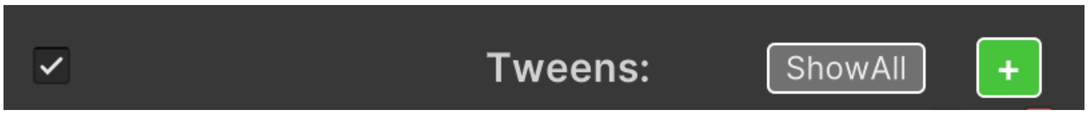
As this is literally method call and not animation per se duration is always 0, and you only have **Delay** to define when to perform this method or methods call relative to the other tweens.

Another special tween is **Child Animation tween**. You can find it in the Other->Child Animation menu. It allows you to effectively inject other TweenAnimations as part of current animations. For example you can create an animation for showing some UI screen as a combination of small animations that represents the appearance of each panel on this screen. That's handy in case of animations with a big amount of tweens and when some parts of the animation could be separated for convenience.

When you select one or more tweens with the toggles at the left, an additional menu at the bottom will appear.



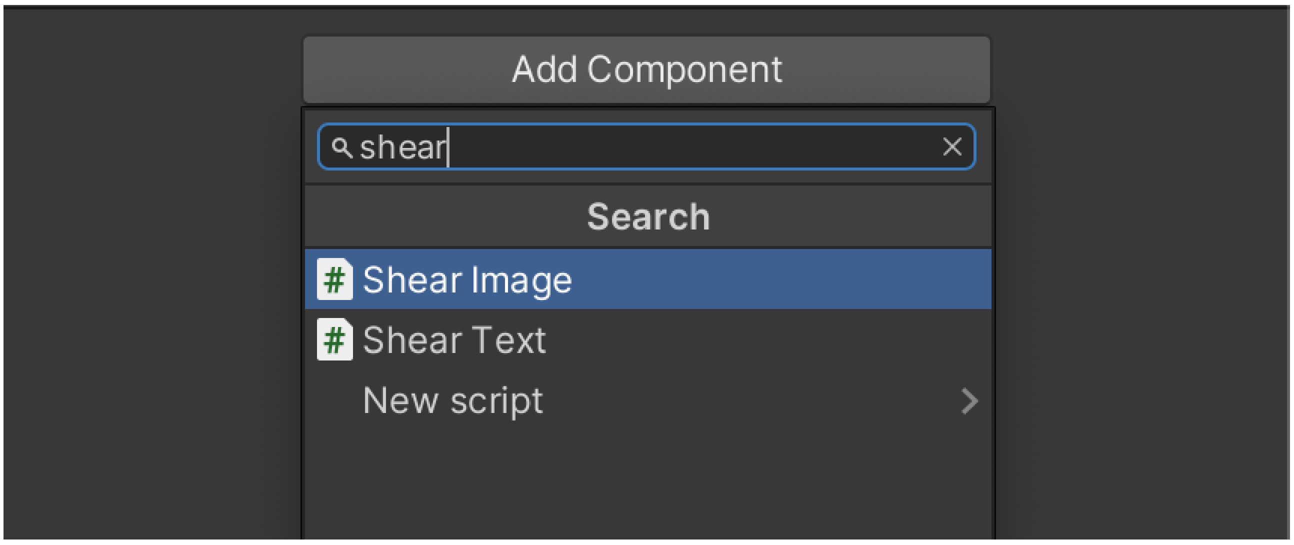
It allows you to perform certain actions only on selected tweens. Now you can add delay or duration to selected tweens(with negative values it will effectively subtract delay/duration), set a certain animation curve for easing and hide selected tweens. The last option could be handy if you have a lot of tweens in the animation and you're currently working only on a certain subset of them at the moment. If any of the tweens are hidden, the 'ShowAll' button will appear at the top of the tweens section so you can bring hidden tweens back.



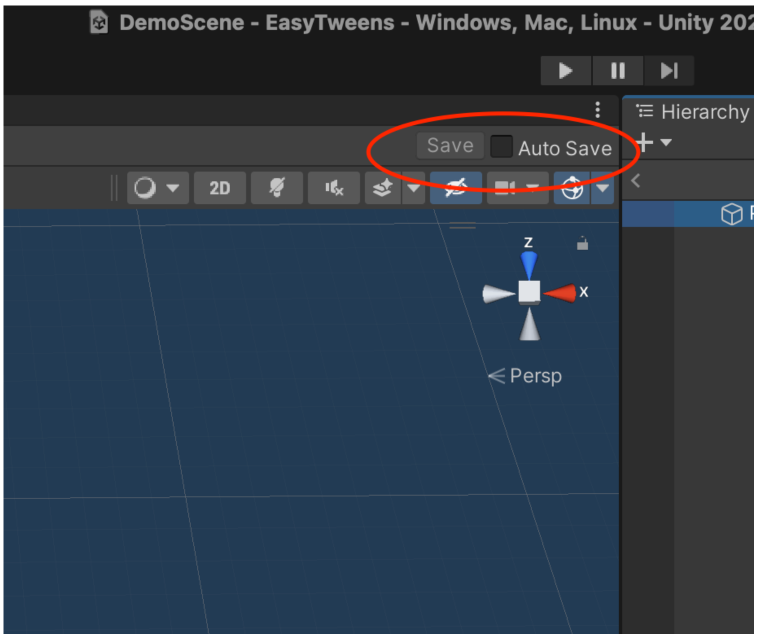
If you accidentally removed some tween just use Cmd-z/Ctrl-z.

Additional components

This asset contains some additional components to create some fancy animations. Specially for UI you can find **ShearImage** and **ShearText** components. They're just the usual UI Image and TextMeshProUGUI text that have 2 additional fields: Shear(in pixels) and Shear Pivot. If you use these components for your images or text you can add shear animation using respectively Shear Image or Shear Text tweens.



Also you can find a **BezierSpline** component that will allow you to easily create paths to follow in combination with **SplineMover** tween.



Important note: if you're creating an animation inside a prefab and you have 'Auto Save' on, you could make your editor very laggy during animation preview as any animation modify prefab properties every frame and saving prefab every frame could be a very resource heavy task.
That has nothing to do with this animation package itself but just how Unity editor works. So I'd recommend always turning it off at least for prefabs with animations.

Usage In Code

When you have a reference to your **TweenAnimation** in the script, you can just call **.PlayForward()** or **.PlayBackward()** to play the animation. You also can pass an optional boolean parameter to specify if you want to play animation normally or skip animation and just fast forward to start or end state. If you need some delay before the whole animation, call the **.SetDelay(delay)** method before calling play methods.

Advanced use

If you have minimal programming knowledge it's very easy to create custom tweens. Tweens aimed to change the value of a certain type bound to some Unity component. Like TweenLocalPosition changes Vector3 value that represents position on a Transform, and TweenCameraFov changes float value on a Camera. And depending on these 2 types you should inherit the generic class that represents this. In TweenLocalPosition case it will be:

```
public class TweenLocalPosition : Vector3Tween<Transform> {…}
```

And for TweenCameraFov:

```
public class TweenCameraFov : FloatTween<Camera>{…}
```

So as you see, base class type represents what type the value has that you want to animate and generic param specifies what type your target object will have.

And after this all you have to do is to override one property called Property 😅. That will be responsible to get/set required values to the target object. For example this is all the code you have to write for the Camera Fov tween:

```
public class TweenCameraFov : FloatTween<Camera>
 {
      protected override float Property
      {
         get
         {
            return target.fieldOfView;
         }
         set
         {
            target.fieldOfView = value;
         }
      }
}
```

And after this it should automatically appear in menu for adding tweens. Target object type will be folder name, and class name without “Tween” word will represent actual tween.

If for some reason you cannot create the tween you want or have any issue with the existing ones you can submit an issue here - https://github.com/timintal/TimintalAssetSupport . Or write a letter at timintal@gmail.com .