How to use the Simple Magic Effects generator kit

About SMEg

When I designed this kit, what I intended to have was a very simple and straight forward method of creating animated visual effects. I am not an artist, but a programmer, so for me to have fancy effects in my projects is quite a daunting task and I am forced to rely on third party effects packages.

Fortunately, in my mind, whenever I see an effect that has many moving parts, I tend to take notice, feel impressed and applaud the work that went into creating such an effect. In anime where the characters bash each other using increasingly powerful magical attacks, the attacks may look awesome simply by the sheer amount of space they take up, but when you break it down, it is just a couple of shining lights and rotating shapes.

So I set off to create the scripts I would need to recreate these effects inside a games engine.

My goal was to be able to create fancy effects using whatever art I could manage to create myself, knowing that in the hands of a decent 2D artist this kit would make it possible to create truly spectacular looking effects. In addition, the scripts used to do this are so basic that they can be applied to any Unity object...

This opens up the door to many interesting possibilities... Place a bunch of animation components on a camera and set their starting times to incremental values, then add a bunch of rotation components with incremental starting value and you now have yourself a cinematic camera for use in creating cut scenes...

Or, place the scaling prefab on a monster creature, set it to scale to 10 and then simply spawn that creature when you do a "summon" spell. Voila! Instant godlike creatures that can spawn behind your team and do great damage. Add in the camera you created before and you have a cinematic summon scene...

The possibilities are virtually endless...

How the scripts work

Most of the scripts derive from the same base class so most of them look identical but only behave differently. Again, I tried to make it functional and powerful but as simple as humanly possible... So let's look at the bodies of the scripts:

PLAY_MODE
The first thing you need to decide is how this effect will behave. You are presented with a drop down box containing 3 options:

1. **KeepGoing**

This will repeat forever.
It makes use of the Speed_up value so if you set the y to 50, for example, the object this is placed on will rotate at a speed of 50 on the y axis or move at a constant speed of 50 on the y axis etc, depending on the script, and just never stop.

2.**UnoDirectional**
This will animate from the Value_start value and gradually move it towards Value_end taking Time_up to get there.
Once the value reaches Value_end, the effect will stop and the component will be disabled to cut down on wasted processing power.

3. **BiDirectional**
This will do the same as UniDirectional but instead of disabling the component, it will wait for Pause seconds and then animate back to Value_start taking Time_down to do so.
Once it reaches the original value, the component will be disabled.

START_DELAY
This instigates a pause before the effect actually starts. Useful in complex effects where you might want to spawn a huge item and only after that item is visible do you want to spawn a couple of smaller items, for instance.

PAUSE
Only used by BiDirectional Play_mode
Measured in seconds, this is the delay between reaching the target value and starting the process of going back to the original value.

TIME_UP
Not used by KeepGoing Play_mode

The value is dampened; meaning it will start off slow, gradually move faster, then move slower again as it nears the end and, eventually, get to the end result.
This gives a much smoother and nicer look but it means that you can't think of the Time_up and Time_down values as seconds.

You need to just play around with this value until you find something that suits your taste and requirements.

More of a speed than a time, this represents how long you want the effect to take to go from Value_start to Value_end. Because this value is dampened, it takes longer in real world time but provides a much better looking result

TIME_DOWN
The same as Time_up except it goes from Value_end to Value_start

CLOSE_ENOUGH

Since we are dealing with dampened values, Value_start and Value_end will take unrealistic amounts of time to be reached so instead of animating to the exact value, we specify an offset that we consider to be close enough and then return the respective, intended result.

For example, we can say that when animating from scale 1 to scale 3, we are happy to accept 0.01 as Close_enough meaning that if the value reaches 2.99 we will consider it scaled to 3. Conversely, when scaling from 3 to 1, we will accept 1.01 as scale 1 and return 1 instead.

VALUE_START
This is the starting value and the final result of the BiDirectional Play_mode.
The meaning of this value is contextual to the script that uses it.
See "The Scripts" below

VALUE_END
This is the target result of the UnoDirectional Play_mode and the starting value of the BiDirectional Play_mode's reverse animation.
The meaning of this value is contextual to the script that uses it.
See "The Scripts" below

SPEED_UP
Only used by KeepGoing Play_mode
The meaning of this value is contextual to the script that uses it.
See "The Scripts" below

The Scripts

sfxBaseAction
This is the core of the various scripts. You need never worry about this

sfxRotate
In KeepGoing Play_mode, this will rotate the object it is on using the speed set by Speed_up. In all other cases, this will match the object's localRotation to whatever the current value is of the animation between Value_start and Value_end

sfxScale
In KeepGoing Play_mode, this will scale the object it is on using the speed set by Speed_up. In all other cases, this will match the object's localScale to whatever the current value is of the animation between Value_start and Value_end

sfxTint
This class does not use the sfxBaseAction so does not contain the fields explained above.

It does still have Time_delay and that still works the same, though.

Additionally it has two more fields for you to complete:

1. **Colors**

   Here you can select what color you want the object to be tinted to. If your object is entirely white then the color you select here is the color your object will be. If it is not completely white then it's color will only be tinted by the color you select.

   If you select more than 1 color, the script will fade between the different colors leaving the object's final color to the color you selected last.

   By changing the value of the alpha field to 0 you can make the object completely invisible so if you add this component to an object and then notice the object isn't in the scene, it is most likely that the color you chose has the alpha value set to 0 as this is what a new Color in Unity is set to by default.

2. **Available_time**

   Here you can tell the script how long it has to cycle between the various colors you have selected. If you chose only one color then the object is tinted to that color and the script is disabled.

   If you selected more than one color then the transition from one color to the other is done at such a speed that it goes from the first to the last across Available_time seconds.

   Example: You set Start_delay to 3, specify 5 colors and set Available_time to 5. This would mean that, upon starting, the effect will tint the object to the first color you specified then wait for 3 seconds.

   After 3 seconds it will cycle through the 5 different colors taking 1 second to go from the one to the other

This class requires a Renderer object so you can either specify one manually by dragging it into the This_renderer field provided or it will scan for the first Renderer it can find amongst the object's children. If no Renderer is found, the component is disabled.

sfxTransform

In KeepGoing Play_mode, this will move the object it is on using the speed set by Speed_up. In all other cases, this will match the object's localPosition to whatever the current value is of the animation between Value_start and

Value_end

sfxUVAnim

In KeepGoing Play_mode, this will adjust the object it's on's UVs using the speed set by Speed_up. In all other cases, this will set the object's UVs to whatever the current value is of the animation between Value_start and Value_end

This class requires a Renderer object so you can either specify one manually by dragging it into the This_renderer field provided or it will scan for the first Renderer it can find amongst the object's children.
If no Renderer is found, the component is disabled.

<u>How to use the scripts</u>

To create an effect is quite simple, really. Provided with the kit is a bunch of basic models and some sample images. The models come with a lower poly version also but the normal ones are already low enough poly that you might just want to use that anyway...

So the process is simply this:

1. Create an empty game object

2. Drag one or more of the basic models into your empty game object
   Optionally, drag in any other mesh / prefab you have in your project

3. For each of the basic models, drag in a relevant material setup to use one of the included sample images.

4. Add as many or as few of the SMEg SFX scripts as you want to each of these individual objects you added to your game object and configure each one as explained in "How the scripts work"

5. Give your game object a name and save it. You are done.

It really is as simple as that. Simply decide what you want to do and then just add the relevant component. Daisy chain as many or as few of them together.

Want to make your character grow 20% then return to normal size each time he level's up? Just add the sfxScale script.

Want to give an object a red tint when it receives damage? Add a sfxTint script.

Want to make a sword appear out of the ground, float 3 feet up into the air then rotate until the player collects it? Add sfxScale, sfxTransform and sfxRotate.

1. Set sfxScale to UnoDirectional and take 1 second to scale to 1

2. Set sfxTransform to UnoDirectional and set Start_delay to 1 so it only starts working after the object is fully scaled.
   Make it take 2 seconds to rise 3 feet.
3. Set sfxRotate to KeepGoing and give it a Start_delay of 3 seconds so it only starts to rotate after the object has reached 3 feet.

What else would you like to do? Simply add the scripts...

I hope you enjoy using this kit...

Jacco