Unity: 5.4.2f2
Date: 16.07.17
Author: NoiseCrime

Easing Equations & Visualisation Guide

This project provides a comprehensive easing class containing 10 different easing equations with 4 variations of each (out, in, out-in, in-out) based on the work of Robert Penner (see - http://robertpenner.com/easing).

Its main focus is to provides a front end visualisation of the easing equations as simple to understand graphs with real-time animated markers to illustrate the motion. You can change the view from a single graph to all the graphs as well as displaying a row of drawers (right side of image below) that as you roll-over with the mouse will ease in or out to see the motion in action.

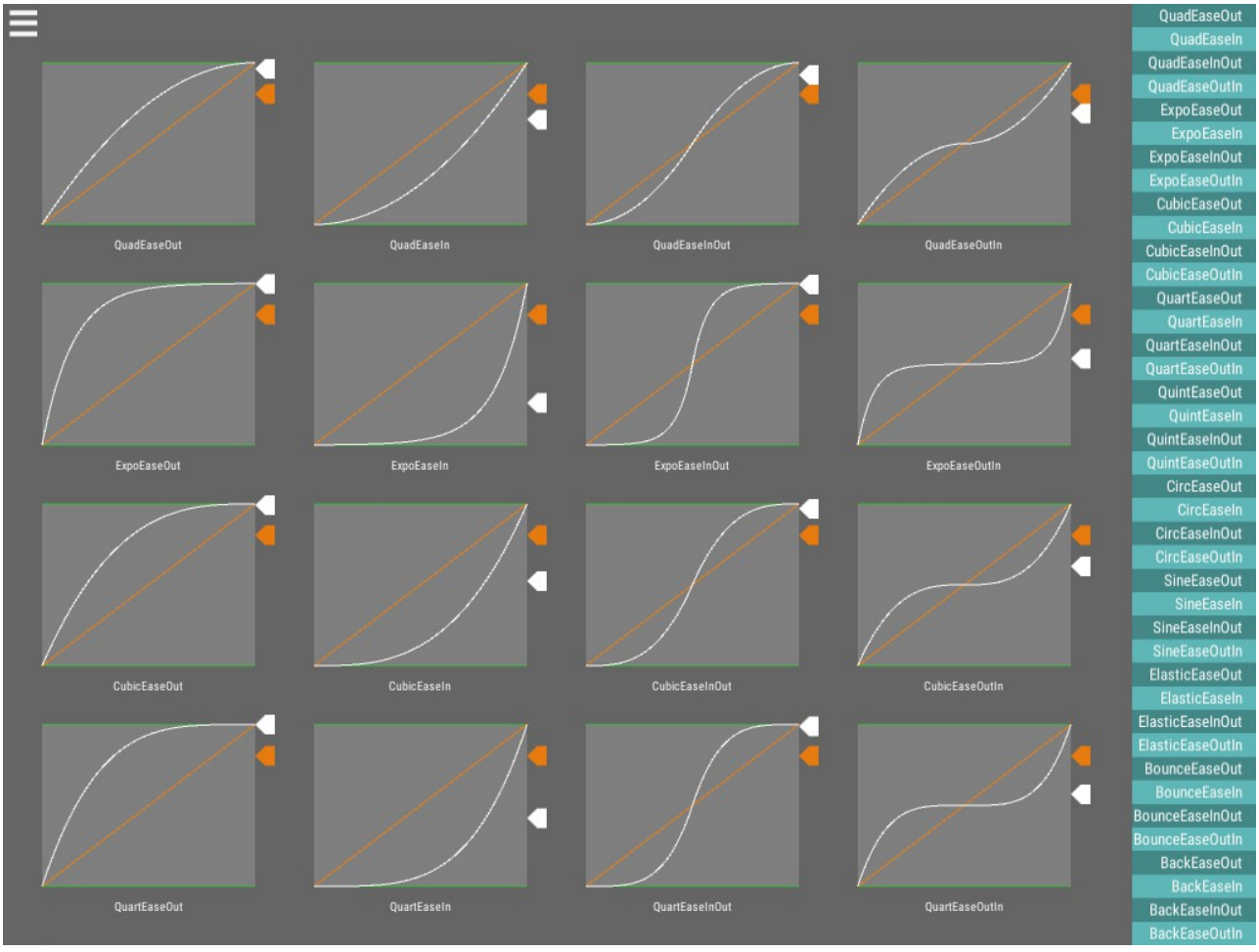


Table of Contents

1.0 Requirements & Known Issues 3

2.0 History 3

3.0 Easing Library Visualisation Demo 4

- 3.1 Ease Graphs 4
- 3.2 Controls 5
- 3.3 Settings Slide-Out Panel 5

4.0 Easing Equation Class 6

- 4.1 Using Delegates To Call An Ease Method 6
- 4.2 Dynamic Delegate Code Outline 7

1.0 Requirements & Known Issues

- Easing Demo requires Unity 5.4.2f2
- EasingEquationsDouble class can be used in any version.

2.0 History

16.07.17 - Unity 5.4.2f2

- Initial Release

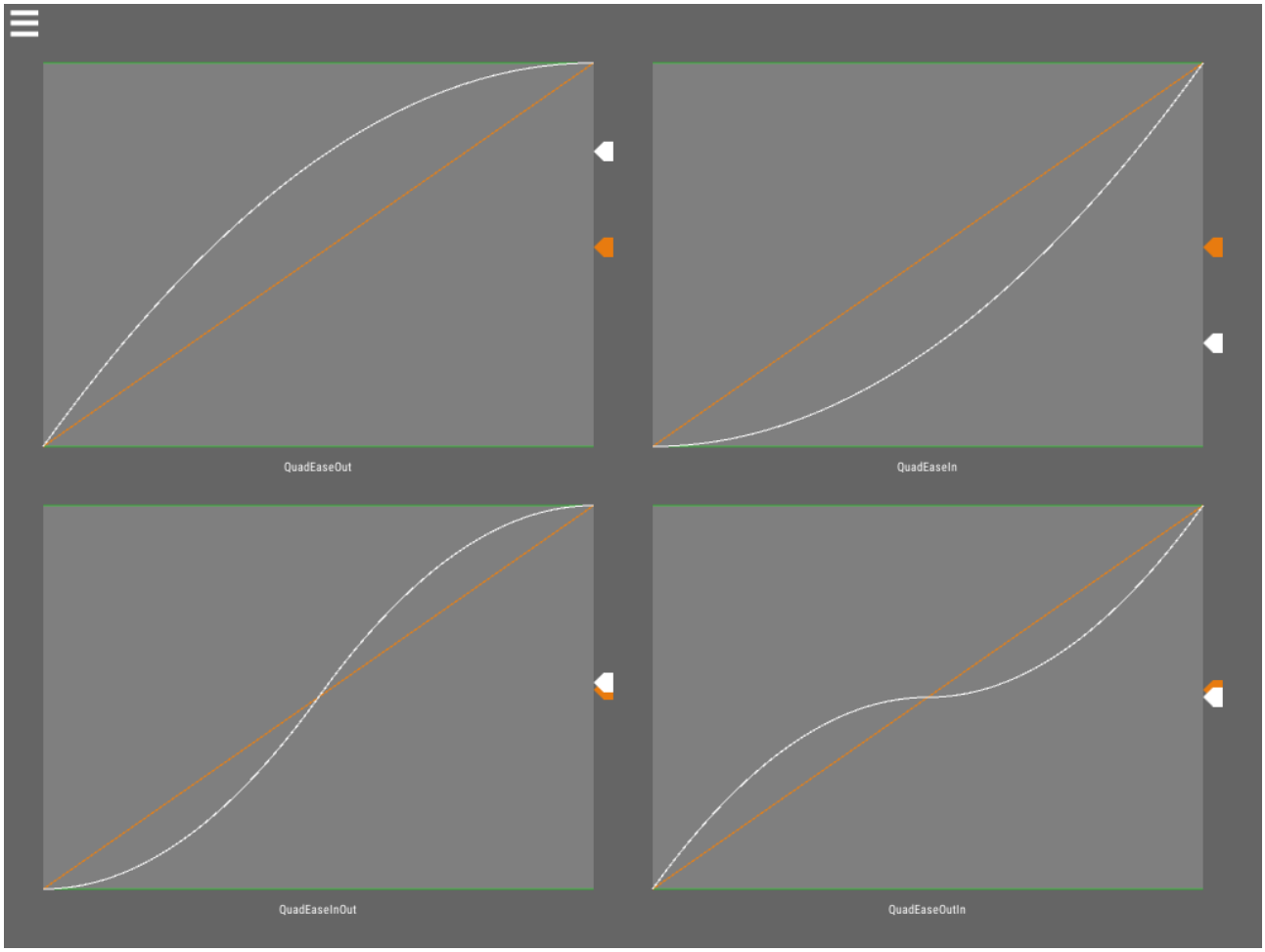
3.0 Easing Library Visualisation Demo

With so many different type of ease equations, each producing a unique motion I often found it difficult to know exactly what the results would be beyond some of the more obvious ones. In order to better understand the motion a specific ease equation I wrote this Visualisation Demo. Its not intended for the demo to be used in projects, though some parts of its code might be of interest, such as delegate creation to call ease methods.

3.1 Ease Graphs

Each graph illustrates a single easing equation method, showing both the shape and through the arrows to its side the motion the equation creates over time.

- Linear Line (orange) - represents no easing.
- Easing curve (white) - the curve of the ease equation used.
- Green lines show the limits of the graph (i.e. 0 – 1). Ease curves (white) that extend beyond these lines will have a motion that moves further than the start end values.

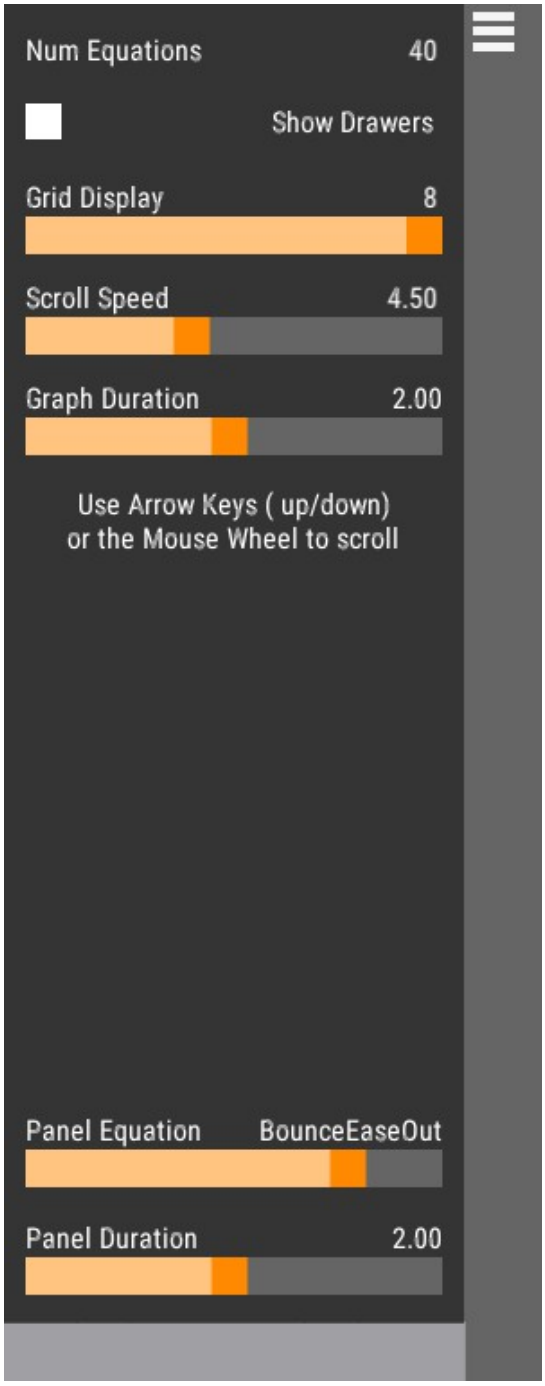


3.2 Controls

- Right click on a graph to make it full screen.
- Right click again to go back to grid view.
- Arrow/cursor keys or Mouse Wheel to scroll vertically.

3.3 Settings Slide-Out Panel

Click on the three bar icon to access the slide-out Setting Panel. The motion of the slide-out is controlled by its own user assigned ease equation, thus allowing the user to get a feel for a specific ease method animation.



Num Equations:
Total number of equations available.

Show Drawers:
When enabled will create a column of slide-out rectangles down the right of the screen, each one representing a specific ease method.

You can roll-over these drawers to open/close them and see the ease method motion .

Grid Display:
Number of rows and columns of ease equation graphs to display.

Scroll Speed:
Controls speed at which cursor keys scroll the view if graphs are off-screen.

Graph Duration:
The duration all graph motions take.

Panel Equation:
Slider to choose the ease equation used when opening/closing the Settings Panel. Can also be set by left-clicking on a visible graph.

Panel Duration:
Time taken to open or close the Settings Panel.

Light Gray Bar:
The bar at the bottom shows the panel motion if it were set to linear.

4.0 Easing Equation Class

You'll find the EasingEquationsDouble.cs class in the project., with each easing equation presented as its own static method to call.

All equations use the same four parameters, t, b, c, d.

- t is current time of the motion.
- d is the duration of the motion.
- b is starting value.
- c is the change in value.

Its important to note that 'c' is the change in value, not the ending value. This means if you want to animate a value from 1 to 0, you would use b = 1 and c = -1, not c = 0.

It should be noted that the class uses doubles, this is due to the .net framework Math methods that only uses doubles, so there seems little point writing a float class as some equations will require casting to/from double anyway, seems better to have the easing equations work exclusively in doubles, then simply cast the input/output values to float.

4.1 Using Delegates To Call An Ease Method

With so many easing equations to choose from it can be advantageous to allow code to dynamically choose the method instead of hard-coding like this,

```
float val = EasingEquationsDouble.BounceEaseInOut( t, b, c, d );
```

To accomplish this we can use delegates, however to make it even more useful its possible to create a delegate via an enum at runtime. This is used in the Easing Library Visualisation Demo to allow the user to dynamically change the easing behaviour of the slide in panel, as well as programmatically generate the 'drawers' that slide in from the right of the screen.

4.2 Dynamic Delegate Code Outline

All the motions in the project are performed through a simple class (TweenAction_RectTransformSlide.cs) that tweens a RectTransform base on the ease method that is defined via an enum and then called via a delegate through late binding.

First we declare the delegate, matching the ease method

```
public delegate double   Ease( double t, double b, double c, double d );
```

then create a field to hold a reference to the desired ease method

```
public Ease        m_EaseMethod;
```

to set the ease method we use createDelegate() so the reference to the method can be dynamically created at runtime, based on an enum value that is converted to a string.

```
m_EaseMethod = ( Ease )Delegate.CreateDelegate( typeof( Ease ),
      typeof( EasingEquationsDouble ).GetMethod( easeEquation.ToString() ) );
```

Finally the ease method field can be used directly and it will call which ever method from the easingEquations was assigned to it.

```
m_Position.x = ( float )m_EaseMethod( m_Timer, m_Start, m_Offset, m_Duration );
```