Interpolactic

Generated by Doxygen 1.8.14

Contents

Chapter 1

Interpolactic

BOILERPLATE, BEGONE!

Interpolactic takes the pain out of Coroutine-based animations by bundling the boilerplate of a time-based I↩ Enumerator into a composable, reusable builder.

All Interpolactic needs is a function or closure that acts upon a time value ranging from 0 to 1.

Fading in a CanvasGroup's alpha can be done in just a few keystrokes:

```
new Interpolation(t => canvasGroup.alpha = t) //Set alpha to t at each time step
  .Duration(1) //1 second
  .Build(this) //Build a runner for this interpolation using this Monobehaviour
  .Play(); //Fire!
```

Animating a transform's position can be done similarly:

```
Vector3 start = transform.position;
Vector3 movement = target.transform.position - start;

new Interpolation(t => transform.position = start + movement * t)
  .Duration(1)
  .Build(this)
  .Play();
```

ANIMATE ANYTHING

The beauty of Interpolactic is that the caller declares the implementation by defining what happens at every time step. While many other plugins restrict interpolation to common types such as float and Color, Interpolactic simply needs an action to perform with respect to time.

For example, we can use caller-defined interpolation across a string to add a "typing" effect:

```
string str = "abcdefg";

new Interpolation(t =>
  {
     int substringLength = (int) t * str.Length;
     textField.text = str.Substring(0, substringLength));
  }
).Duration(1).Build(this).Play();
```

PAIN-FREE PLAYBACK

The actual Coroutine is then wrapped in an Runner object that offers a kit of utility methods to control playback.

```
void TogglePlaying()
{
     if(animation.playing)
         animation.Pause();
     else
         animation.Play();

}
```

Chapter 2

Namespace Documentation

2.1 Interpolactic Namespace Reference

Classes

- class Interpolation

Chapter 3

Class Documentation

3.1 Interpolactic.Interpolation Class Reference

Classes

- class Runner

Public Member Functions

- Interpolation ()
- Interpolation (Action< float > action)
- Interpolation Duration (float duration)
- Interpolation Delay (float delay)
- Interpolation Completion (Action onComplete)
- Interpolation OnStop (Action< float > onStop)
- Interpolation EasingFunction (Func< float, float, float, float > easingFunction)
- Interpolation RealTime (bool realTime)
- Interpolation FirstStepBeforeDelay (bool firstStepBeforeDelay)
- Interpolation Repeats (bool repeats)
- Interpolation AddAction (Action< float > stepAction)
- Runner **Build** (MonoBehaviour monoBehavior)

Properties

- float duration `[get]`
- float delay `[get]`
- bool realTime `[get]`
- bool firstStepBeforeDelay `[get]`
- bool repeats `[get]`

3.1.1 Detailed Description

The IPInterpolation class acts as a template for Coroutines to act on a callback with a time value from 0 to 1.

The IPInterpolation doesn't generate a IPRunner until Build() is called on it. Therefore, for repeated animations an IPInterpolation can be kept and tweaked, generating a new IPRunner with every Build.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Interpolation() [1/2]

```
Interpolactic.Interpolation.Interpolation ( )
```

Create a new Interpolator with no actions.

3.1.2.2 Interpolation() [2/2]

```
Interpolactic.Interpolation.Interpolation (
            Action< float > action )
```

Convenience initializer for a new Interpolator with a single action.

3.1.3 Member Function Documentation

3.1.3.1 AddAction()

```
Interpolation Interpolactic.Interpolation.AddAction (
            Action< float > stepAction )
```

Create a clone of Interpolation object, with an additional action to be called each time step. All existing actions registered to the Interpolation will be preserved.

Parameters

stepAction	Action to be called at each time step in the interpolation.

3.1.3.2 Completion()

```
Interpolation Interpolactic.Interpolation.Completion (
            Action onComplete )
```

Create a clone of Interpolation object with a callback to be invoked when the interpolation has run to completion. The completion action will not be called if the interpolation is stopped manually.

Parameters

onComplete	Action to be called when interpolation completes.

3.1.3.3 Delay()

```
Interpolation Interpolactic.Interpolation.Delay (
            float delay )
```

Create a clone of Interpolation object with a specified delay before execution.

Parameters

delay	Delay of interpolation in seconds.

3.1.3.4 Duration()

```
Interpolation Interpolactic.Interpolation.Duration (
            float duration )
```

Create a clone of Interpolation object with a specified duration.

Parameters

duration	Duration of interpolation in seconds.

3.1.3.5 EasingFunction()

```
Interpolation Interpolactic.Interpolation.EasingFunction (
            Func< float, float, float, float > easingFunction )
```

Create a clone of Interpolation object with an easing function.

Parameters

easingFunction	Easing function, such as `Mathf.SmoothStep`

3.1.3.6 FirstStepBeforeDelay()

```
Interpolation Interpolactic.Interpolation.FirstStepBeforeDelay (
            bool firstStepBeforeDelay )
```

Create a clone of Interpolation object with firstStepBeforeDelay enabled or disabled.

Parameters

firstStepBeforeDelay	Should the interpolation act on t=0 before performing the delay?

3.1.3.7 OnStop()

```
Interpolation Interpolactic.Interpolation.OnStop (
            Action< float > onStop )
```

Create a clone of Interpolation object will a callback to be invoked when the interpolation has been stopped manually.

Parameters

onStop	Action to be called if interpolation is stopped.

3.1.3.8 RealTime()

```
Interpolation Interpolactic.Interpolation.RealTime (
            bool realTime )
```

Create a clone of Interpolation ojbect with real time enabled or disabled.

Parameters

realTime	Should the interpolation run in real time?

3.1.3.9 Repeats()

```
Interpolation Interpolactic.Interpolation.Repeats (
            bool repeats )
```

Create a clone of Interpolation with repeating enabled or disabled.

Parameters

repeats	Should the interpolation repeat?

3.1.4 Property Documentation

3.1.4.1 delay

```
float Interpolactic.Interpolation.delay [get]
```

Length in seconds that the Interpolation's runner will wait before beginning interpolation.

Defaults to 0.

3.1.4.2 duration

```
float Interpolactic.Interpolation.duration [get]
```

Length in seconds of the Interpolation over time.

Defaults to 0.

3.1.4.3 firstStepBeforeDelay

```
bool Interpolactic.Interpolation.firstStepBeforeDelay [get]
```

If true, the Interpolation's step at t=0 will be called before the delay is applied. Useful for holding an animation at its initial state while waiting for the delay to end.

Defaults to false.

3.1.4.4 realTime

```
bool Interpolactic.Interpolation.realTime [get]
```

If true, the interpolation will execute independently of Time.timeScale.

Defaults to false.

3.1.4.5 repeats

```
bool Interpolactic.Interpolation.repeats [get]
```

If true, the associated Runner will repeat until stopped.

Defaults to false.

The documentation for this class was generated from the following files:

- /Users/max/Dev/Interpolactic/Assets/Interpolactic/Source/Interpolation.cs
- /Users/max/Dev/Interpolactic/Assets/Interpolactic/Source/Interpolation.Runner.Coroutine.cs

3.2 Interpolactic.Interpolation.Runner Class Reference

Public Member Functions

- virtual void Stop ()
- virtual void Play ()
- virtual void Pause ()

Protected Member Functions

- abstract float DeltaTime (bool realTime)
- IEnumerator< float > PerformInterpolation ()

Protected Attributes

- Interpolation interpolation

Properties

- bool finished [get]
- bool playing [get]
- bool stopped [get]
- bool started [get]

3.2.1 Member Function Documentation

3.2.1.1 DeltaTime()

```
abstract float Interpolactic.Interpolation.Runner.DeltaTime (
            bool realTime ) [protected], [pure virtual]
```

The interval in seconds since the last frame.

3.2.1.2 Pause()

```
virtual void Interpolactic.Interpolation.Runner.Pause ( ) [virtual]
```

Suspends playback on the Runner. Resources are still allocated in a paused Runner, so be sure to only call Pause() if planning on resuming the animation.

3.2.1.3 PerformInterpolation()

```
IEnumerator<float> Interpolactic.Interpolation.Runner.PerformInterpolation ( ) [protected]
```

IEnumerator for the actual interpolation of t from 0 to 1. Will perform the Interpolation in its entirety then call the onComplete callback, if defined.

3.2.1.4 Play()

```
virtual void Interpolactic.Interpolation.Runner.Play ( )  [virtual]
```

Begins or resumes playback on the Runner.

Warning

Will throw an exception if the Runner has already been stopped.

3.2.1.5 Stop()

```
virtual void Interpolactic.Interpolation.Runner.Stop ( )  [virtual]
```

Stops the Runner and frees all of its resources.

Calls the interpolation's onStop action, if defined.

3.2.2 Member Data Documentation

3.2.2.1 interpolation

```
Interpolation Interpolactic.Interpolation.Runner.interpolation  [protected]
```

The Interpolation model for the Runner's behavior.

3.2.3 Property Documentation

3.2.3.1 finished

```
bool Interpolactic.Interpolation.Runner.finished  [get]
```

Whether the Runner has been allowed to run to completion. This will always be "false" for runners of repeating Interpolations.

3.2.3.2 playing

```
bool Interpolactic.Interpolation.Runner.playing  [get]
```

Whether the Runner is currently animating.

3.2.3.3 started

```
bool Interpolactic.Interpolation.Runner.started  [get]
```

Whether the Runner has begun to play.

3.2.3.4 stopped

```
bool Interpolactic.Interpolation.Runner.stopped  [get]
```

Whether the Runner was stopped manually via Stop().

The documentation for this class was generated from the following file:

- /Users/max/Dev/Interpolactic/Assets/Interpolactic/Source/Interpolation.Runner.cs

Index