Tweening GameObjects

Intro

TpTweener can also be used to tween GameObjects' Position, Rotation, Scale, or Color; or to tween a GameObject's position along a Bezier path.

Unlike tweens for TPT tiles, there's no equivalent 'Matrix' tween that can tween all of these at once.

CreateGoTween is the most basic entry point for GameObject tweening:

```
public long CreateGoTween(GameObject?            gameObject,
                          Action<TpTween,float>? gameObjectAction,
                          TpEasingFunction.Ease    ease,
                          float                    duration,
                          long                     sequenceId         = 0L,
                          LoopType                 loopType           = LoopType.None,
                          int                      numLoops           = -1,
                          Action<TpTween>?         onFinished         = null,
                          Action<TpTween, float>?  onRewindOrPingPong = null,
                          Action<TpTween>?         onLoopEnded        = null,
                          Action<TpTween, float>?  onUpdate           = null )
```

This is very flexible: provide a target GameObject and an Action to modify same. The Action gets the Tween instance and a 0 - 1F progress value. The remaining parameters are the same as for TPT tile tweens: ease function, duration, etc.

One can also create a sequence and add GameObject tweens to the sequence just like with TPT tile tweens. However, one cannot mix GameObject and TPT tile tweens in the same sequence: you'll get a 0L return value, indicating an error.

There are also two higher-level methods with preset Actions: GameObjectTrs and GameObjectColor

GameObjectTrs allows transform-related GameObject tweens for the most common use: supply an enum value indicating which component (position, rotation, scale) to affect.

The `startValue` parameter is nullable. If this is null when the method is invoked then the GameObject's transform.position, transform.rotation, or transform.scale is used for `startValue` and this obtained value is added to the `endValue`: that is, the tween is 'relative'.

For example, if:

- `startValue` == null
- `endValue` = Vector3(5,6,7)
- the GameObject's transform.position was (3,4,5)

Then internally:

- `startValue` becomes (3,4,5) (the tranform.position from the GameObject)
- `endValue` becomes (5,6,7) + (3,4,5).
- This can produce strange results when tweening rotation or scale.
- Another way to have greater control is to always supply the startValue but make the tween 'relative' by modifying endValue yourself before calling the method.

If `startValue` isn't null then the tween is absolute:

- If the current position, rotation, or scale doesn't match the value of `startValue` then the current position, rotation, or scale becomes `startValue`
  - For example, in the case of position, the GameObject's position will instantly change when the tween starts.
- `endValue` isn't changed.

```
public long GameObjectTrs(GoEaseTrsTarget trsEaseTarget,
                          GameObject              go,
                          Vector3?                startValue,
                          Vector3                 endValue,
                          float                   duration,
                          TpEasingFunction.Ease   ease,
                          long                    sequenceId         = 0L,
                          LoopType                loopType           = LoopType.None,
                          int                     numLoops           = -1,
                          Action<TpTween>?        onFinished         = null,
                          Action<TpTween, float>? onRewindOrPingPong = null,
                          Action<TpTween>?        onLoopEnded        = null,
                          Action<TpTween, float>? onUpdate           = null)
```

GameObjectColor is for color tweening. The `startValue` parameter is evaluated in the same way: use the value if provided but if null use the current color value.

This method accomodates SpriteRenderers and Renderers. It will use the first one it finds in the GameObject or its children.

```
public long GameObjectColor(GameObject             go,
                            Color?                 startValue,
                            Color                  endValue,
                            float                    duration,
                            TpEasingFunction.Ease    ease,
                            long                     sequenceId         = 0L,
                            LoopType                 loopType           = LoopType.None,
                            int                      numLoops           = -1,
                            Action<TpTween>?         onFinished         = null,
                            Action<TpTween, float>? onRewindOrPingPong = null,
                            Action<TpTween>?         onLoopEnded        = null,
                            Action<TpTween, float>? onUpdate           = null)
```

The Tween instance variables for GameObject-related tweens have public property accessors rather than internal so you can modify them just after creation (as seen in the code for these two methods). Modifying these during the execution of a tween is unsupported and may cause exceptions or other issues.

Like all other Tween instances, GameObject tween instances are pooled. If you hold a reference to a tween instance it will become invalid at some point and may interfere with the pooling operation.

Creating GameObject Tweens from an asset

Similar to TPT tile tweens, you can use an asset `TpGoTweenSpec` to create GameObject tweens or Sequences of same.

There are corresponding methods:

- CreatGoTweenFromSpec: use an individual GameObject tween spec from the list of these in the asset.
- CreateSequenceFromGoSpec: use all or some of the asset's GameObject tween specs to create a sequence.

NOTE: since the 'spec' is a project asset you can't modify its fields without affecting everything that uses this 'spec'. If you want to modify fields in a spec prior to sending it to the Create method you need to clone it first and then make changes to the cloned spec's fields. Then send the cloned

instance to the Create method.

NOTE: you can add Delay tweens to the 'spec'. However, as usual Delay tweens can *only* be used in sequences.

Other uses

One can create a tween using CreateGoTween with an Action that modifies other values besides the transform or color.

For example, when using GameObjectTrs to create a Position, Rotation, or Scale GameObject tween, the Action supplied to CreateGoTween actually performs the calculations and transform changes. Instead, you can use the progress value passed to the Action to change, say, the intensity of a Light.

Note that if you just want a periodic timer, use TpLib.InvokeRepeatingUntil instead. It's way more efficient.

An example custom tween is shown below. This is built-in to the Tweener.

There's an example of a custom Tween affecting the character size of a TextMesh. Find it in the Demos folder, and see the next page for a deeper dive.

Bezier Curve Position Tweening

```
public long GameObjectPositionBezierTween(GameObject
go,
                                          Vector3[] points,
                                          float
duration,
                                          long                    sequenceId             =
0L,
                                          LoopType                loopType               =
LoopType.None
                                          int                     numLoops               = -
1,
                                          Action<TpTween>?        onFinished             =
null,
                                          Action<TpTween, float>? onRewindOrPingPong     =
```

```
null,
                                        Action<TpTween>?        onLoopEnded           =
null,
                                        Action<TpTween, float>? onUpdate              =
null)
```

This is similar to GameObjectTrs. Provide a set of points and the GameObject's local position will tween between these points.

- Inclusion in sequences is allowed.
- points[0] is the start position.
  - If not the current position of the GameObject then the GameObject will immediately move there.
- points[N] is the end position.
- The size of the points array must be between 2 and 16.
- LoopType.Loop means that the GameObject will jump back from points[N] to points[0] before repeating.
- LoopType.PingPong means that the GameObject will tween back to points[0] from points[N] when repeating.

Revision #26
Created 2025-08-23 12:42:49 UTC by Vonchor
Updated 2025-09-27 16:51:50 UTC by Vonchor