Tweener Service

Intro

TpTileTweener is a TPT service that can be used for tweening TilePlus tile sprites. The Tweener also has support for sequences.

Generally it's for use within TilePlus tile code, ZoneActions, or EventActions. The main restriction is that a TPT tile reference must be provided when creating Tweens or Sequences.

You use it by first obtaining a service handle from `protected static TpTileTweener TweenerService` property within any TPT tile code.

For example:

```
TweenerService.CreateTween(this,
                    Vector3.zero,
                    Color.red,
                    Matrix4x4.identity,
                    TpEasingFunction.Ease.Linear,
                    TpTileTweener.EaseTarget.Color,
                    1,
                    0,
                    TpTileTweener.LoopType.PingPong,
                    -1,
                    OnFinished);
```

This creates and starts a Color tween (EaseTarget.Color) with the final color being Color.red.

Although the Linear easing function is specified here, it's ALWAYS linear (Lerp) for Color tweens (the value is ignored for Color tweens).

The duration is one second. The -1 means that this tween repeats a PingPong tween until killed. When the tween is killed then the OnFinished callback is invoked.

The first four parameters are a reference to the tile itself (this), a Vector3 value, a Color value and a Matrix value. For the Vector3, Color, and Matrix parameters you just use the one you need for a particular tween.

Here, the Vector3 field is set to Vector3.zero and the Matrix field is set to Matrix4x4.identity because this is a Color tween.

The Vector3 field is used for tweens that use Vector3 values like Position, Rotation, or Scale.

The Matrix parameter is used for Matrix tweens which are a special tween variety.

Why Another Tweener?

But why create another Tweener in the first place? The Unity Asset store is full of free Tweeners on the Unity Asset Store and DemiGiant's DOTween is really great.

Here's why: none of them support natively tweening Tile sprites.

- You can do it in DOTween with Getters and Setters but DOTween can't handle tiles suddenly becoming null (if they're deleted) in the same way as it does for GameObjects.
- TilePlus Toolkit has a 'DOTween Adapter' which handles some of that, but it's inconvenient (and deprecated).
- DOTween is great but is way more complex than needed just for tiles.

TpTileTweener is optimized for TilePlus tiles. It isn't for normal Unity tiles. But it does some things DOTween doesn't do. It's non-generic, tight, hard-coded and single-mindedly Tilemaps only.

Demos

TilePlus Extras has a TpTweener folder with a few example tiles and several scenes.

- TweenerFlexTile has fields for all possible types of tweens (except DELAY) and the fields which are shown change with what's being affected
  - I.E., if the tween target is Color then a color picker is available, but for, say, position, a Vector3 field is shown.
- TweenSpecTile uses a TpTweenSpec asset to run a tween from that asset.
- TweenSpecSequenceTile uses all the entries from a TpTweenSpec asset to create and run a sequence. You can also check a toggle for interactive use. This allows you to tweak the TweenSpec asset while the Editor is playing: each time the sequence ends it re-loads from the asset rather than internally repeating the cached sequence.

These three tiles are part of the normal distribution in the Plugins folder but for many uses, in a real app, you'd run a tween as the result of a Message being sent to a tile or because of a Zone entry or exit.

However, the stock 'Tweening' tiles are great for your experimentation with this feature; especially TweenSpecSequenceTile.

To use tweens in the code for Event or Zone actions, check out the TpTweenerSubObject. SubObjects are scriptable objects which are attached as references to TpTileZoneActions or TpTileEventActions.

TpTweenerSubObject can be used by a TileAction to easily tween the tile's sprite when an event is handled (EventAction) or when a Zone is entered or exited (ZoneAction.)

This approach is used in the TileFabDemos/LayoutSystem demo.

Tweens and Sequences

Tweens

TpTileTweener supports tweening Position, Rotation, Scale, and Color of Tile sprites.

For color tweens you can also specify 'Constant A', which means that the alpha value doesn't change from the value found when the tween begins.

You can also tween the Tile's Matrix, or the Matrix with Color, or the Matrix with Color-Constant-A. These let you create a single tween that can change Position, Rotation, Scale and/or Color/Color-Constant-A, with optional fine-grained control over which parts of the transform change. For example, one could specify a Matrix tween where only position and scale are changed.

- Matrix tweens also allow specifying different Ease functions for Position, Rotation, and Scale.
- MatrixColor and ColorConstantA use Lerp for the Color tween.
- Color Tweens always Lerp

It's incredibly flexible, albeit a bit more work to set up.

Don't get confused: if you use one of the Matrix varieties, including those with Color, it's one tween. When the tween updates it can affect Postion, and/or Rotation, and/or Scale, and/or Color in one operation. It's not three or four tweens in parallel.

To that end, the TweenSpecSequenceTile has an interactive mode. In this mode, the sequence is forced to not loop. When the sequence completes, it restarts with a fresh set of value fom the Tween Spec asset. Hence, if you change values of this asset while the editor is Playing, such changes will be seen when the sequence restarts. As is true for project assets, the changes will remain after you exit play mode. It's a great way to play with sequences and Matrix-style tweens.

The tweener supports one-shot, looping, and ping-pong tweens. For looping tweens there's a loop count and as usual if the loop count is -1 the loop continues forever until cancelled or the tile is deleted.

Delay tweens can also be created, but they're only for sequences.

Sequences

A sequence can be created and tweens can be created with automatic adding to a sequence. Sequences do what you'd expect: run all the tweens in a list of tweens. Delay tweens can be added to a sequence. They don't tween anything, just time out. Since a callback can be issued each time the sequence changes to the next tween, a sequence of just Delay tweens can be used to create a simple sequencer.

Callbacks

Individual tweens have callbacks for each Update, on completion, and when a looping tween loops. Tweens are auto-deleted when their parent tile becomes null. Since a callback is issued to a tile and the tile is null in this case, the tween-ending callback is not issued.

Note that callbacks for each update can get extremely processor-intensive depending on how many tweens are running and what is done during execution of the callback. For example, if you have 100 tweens running then you'd get 100 callbacks every frame. Hence, these are recommended ONLY during development but there's no explicit restriction on this.

Note that while each callback provides a reference to the specific tween that caused the callback, the values in the instance are read-only properties so

1. Do not hold a reference outside the local scope. These are pooled at at the end of the tween: it is returned to the pool and reset.
2. You can't change anything except a "Diagnostics" property.
3. You can't copy a tween and inject it back into the system.

Sequences have callbacks when complete and when the next tween in the sequence begins.

Sequences intercept the individual tween callbacks for the sequence's internal use but don't relay them to tiles.

Create a tween

There are four ways to create and run a tween.

- Use the Create methods
  - CreateTween: create any sort of tween.
  - CreateDelayTween: Don't tween anything, just wait. Sequences only.

    - CreateTweenFromSpec: Use the TpTweenSpec asset to create tweens.
        - TpTweenSpec.Tween are the individual specs in a list of specs in the asset.

Create methods return a long integer which is the ID of the tween.

Note that tweens are relative: for example, if you tween position, the value provided as an endpoint is the CHANGE that you want and NOT the absolute position.

For example: Creating a tween that changes Position with an EndValue of Vector3(2.2,3.5,0) doesn't move the Tile's sprite to (2.2,3.5,0). It *offsets* the position by 2.2 units in the X direction and 3.5 units in the Y direction from the tile's position in the tilemap: it's relative to the tile.

So if the tile is at (0,0,0) for this example, the tile's sprite would move to 2.2,3.5,0. If the tile were at (10,15,0) then the tile's sprite moves to (10+2.2, 15+3.5, 0+0) = (12.2,18.5,0).

And that makes a lot of sense in this context: the tile itself doesn't move while you're tweening its color or transform. You're actually affecting the *Tilemap* and how it displays the tile's sprite. Nothing in the tile's data changes at all.

Similarly, if affecting rotation, the change is relative to the existing rotation of the tile's sprite, and if affecting scale, the change is relative to the existing scale of the tile's sprite.

Color tweens are a little different: you're tweening between the color when the tween is launched and an absolute end color specified in the CreateTween method call.

In a practical application, CreateTweenFromSpec is the most useful. Most likely you'll have several tweens that you'll use repeatedly for many tiles.

Rather than have individual tiles have all the fields for specifying tweens (as in TweenTileExample) you use a Tween specification from a TpTweenSpec asset; this can be seen in TweenTileExample2.

You can also use specs from this asset when creating sequences.

Ignoring CreateDelayTween, the Create methods launch the tween immediately. If you need a delay, create a two-element sequence with a DELAY as the first tween in the sequence.

Operations on Tweens

- A running tween can be cancelled with KillTween.
- Get a reference to a running tween with GetTween. However, since these are pooled items, do not hold this reference outside a local scope or you WILL get memory leaks.
- Get an unpooled copy of a running tween with CopyTweener. Note that this is NOT pooled but is a COPY of the running tween at that point in time.
- Get the ToString of a running tween without affecting the instance with GetTweenInfo.
- Tweens are auto-deleted if their parent tile or the parent tile's Tilemap become null.

Create a sequence.

- Use CreateSequence, add tweens using a Create method, and use PlaySequence to run the sequence.
- Use CreateSequenceFromSpec and use PlaySequence to run the sequence.

CreateSequenceFromSpec uses the tween specifications in a TpTweenSpec asset and makes a sequence out of the tweens.

- An array of indices into the List of Tweens in a TpTweenSpec asset can be used if you want only specific tweens from the list to be used.
  - Don't do this unless really needed: if you change the TpTweenSpec asset later you have to update the array.
- If the array is null (the default) then the entire list of Tweens is used.

CreateSequence and CreateSequenceFromSpec return a long integer which is the ID of the sequence.

If you're not using CreateSequenceFromSpec:

- Create a sequence using CreateSequence.
- Use CreateTween or CreateDelayTween and supply the ID of the sequence in the method call.
  - When the sequence ID is supplied, the created tween is NOT immediately run, but its data structure (a TpTween instance) is added to the sequence.
- Use PlaySequence to start the sequence.

Operations on Sequences

- Delete an unused sequence with DeletePendingSequence.
- Kill a running sequence with (wait for it...) KillRunningSequence.
- Sequences are auto-deleted if their parent tile or the parent tile's Tilemap become null.

Reset

Reset the tweener with ResetTweener and the Sequencer with ResetSequencer. Both release all active tweens and those which are included in the sequences' lists of tweens.

Observe

- Use the Tween Monitor (menu item). This opens an Editor window which displays all of the information about running tweens and sequences.

    - If you have many tweens running this can slow down your game.
- Use the Services Inspector (menu item). This shows all running services (including this one).

Revision #20
Created 22 June 2025 20:10:43 by Vonchor
Updated 16 July 2025 15:00:55 by Vonchor