Coroutines and Awaitables

Coroutines

TilePlus generally uses Awaitables rather than coroutines. However, it's easy to make a coroutine wait for a Tween or Sequence to complete:

```
var tId = TweenerService.CreateTween(.....);


yield return new WaitUntil(() => !TweenerService.IsRunningTween(tId));
```

Here we just poll TweenerService to see if the tween we created is still running. The Coroutine waits until IsRunningTween returns false.

However, if all you want is to know when the tween or sequence completes, just use one of the 'Finished' callbacks and save the coroutine overhead.

Important: You'll note below that Awaitable tweens and sequences can't have infinite loop counts and will fail if the loop count value passed to the create method is < 0. A similar situation exists for using Coroutines: if the loop count is infinite then the coroutine won't ever terminate. However, you need to handle this yourself or wrap the create method so that that circumstance is avoided.

Awaitable

Tweens and Sequences can be wrapped into `Awaitable` methods. The main restriction is that one cannot have Awaitable tweens or sequences with infinite loop counts (i.e., < 0).

Examples of how to use this feature are part of the TweenerFlex and TweenSpecSequence tiles.

Here's a bit of the code for the TweenerFlex tile:

```
private async Awaitable WaitForCompletion(long tId)
{
    if (m_LoopCount < 0)
```

```
    {
        //NOTE: The TweenAsAwaitable call below will fail if the loop count is < 0
        Debug.Log("infinite loop count tweens should not be awaited!");
        return;
    }
    var at = TweenerService.TweenAsAwaitable(tId);
    if (at == null)
        return;
    //The elapsed time calculation is obviously not required.
    var now = Time.time;
    var s   = TweenerService.GetTweenInfo(tId);

    await at;
    Debug.Log($"Awaitable tween complete, ET: {Time.time-now}. Id = {tId}\n{s}");
}
```

A simpler version (untested) could be:

```
private async Awaitable WaitForCompletion(long tId)
{
    var at = TweenerService.TweenAsAwaitable(tId);
    if (at == null)
        return;
    await at;
}
```

Since it's almost impossible for the TweenerService to be null, one could do this:

```
private async Awaitable WaitForCompletion(long tId)
{
    await TweenerService.TweenAsAwaitable(tId);
}
```

Or inline that type of statement in any async method that needs it.

It's pretty simple to use Awaitable tweens. Awaitable sequences are very similar, but you'll note that the Awaitable method is cancelled if the Sequence can't launch for some reason.

TpLibTasks has this helper method for Awaitables:

`public static async Awaitable WhenAll(List<Awaitable> awaitables)`

Using this method one can run a number of tweens and Await for them ALL to complete.

Revision #13
Created 2025-09-05 14:21:59 UTC by Vonchor
Updated 2025-09-08 19:25:37 UTC by Vonchor