Tweening Library

Link to starting repo

Starting point

In the directory **tests/TestingScene.tscn** - A sprite is placed in the scene. It has a script **scripts/TestingScene.gd** attached where we can test our tweening library by interpolating e.g. position, rotation, scale and color modulation of the sprite. Some examples for each completed step are already there.

Goal

Functionality similar to `Tween.interpolate_property(...)` method.
Tween — Godot Engine (stable) documentation in English

Steps

In the directory **tweening_library/**.

Architecture overview

- **Tweens** - A tween manager responsible for handling all the tweens. Might be an autoload or a scene instantiated, where the tweens are needed.
    - Has an array of all the running tweens.
    - Has a function for creating a tween. Parameters will be:
        - the target object,
        - name of the tweened property (of the target object),
        - (initial and) target value,
        - duration,
        - easing function,
        - callback function on end.
    - Periodically updates all the running tweens (progresses them in time). If a tween finishes, its callback is called and the tween is removed from the array.
- **CustomTween** - A class representing a tween and all its properties including state.
    - Has fields for:
        - current progress (time)
    - Over this lab, it should gain fields for:
        - initial and target value,
        - total duration,
        - the target object,

    - name of the property (of the target object),
    - current progress (time),
    - easing function,
    - callback on end.
- Has a method which is called by the manager to progress the tween in time.
- Will have an assortment of static easing functions to choose from.

Tween manager

Tweens.gd - The tween manager has an array of tweens and in the `_process` function it goes through all of them, updates them and removes the finished ones. The tween class needs a function for update and a function to determine whether it has finished.

Tasks

For each task, you shall uncomment **only** the appropriate line(s) in **TestingScene.gd** and then make it work as the comment above it. The vast majority of the modifications should go into files **Tweens.gd** and **CustomTween.gd**, specific tasks will mention when you need to modify different files.

Task 1 - Tween class linearly interpolating between 0 and 1

Just output the numbers between 0 and 1 over 1 second and then finish.

All you need is a couple simple modifications to **CustomTween.gd** (add the counting functionality).
GDScript basics — Godot Engine (stable) documentation in English (classes)

Task 2 - Linear interpolation of numbers

Print numbers from an initial value to a target value over a given duration. Just modify the script to work with user-given values.

Task 3 - Linear interpolation of a specific property

Very simple modification. Let's apply the interpolated number to a specific property value of a given object - scale.

Task 4 - Linear interpolation of any number/vector property

We'll accept as parameter a string name of a property. It has to be either Number or Vector2/3 (both can be multiplied by a scalar value and so treated the same way, we'll handle more complex types later).

The name of the property is passed as an argument and, using a general setter, that property is interpolated.
Unfortunately in GDScript, we cannot use a general setter to access individual components of Vector2/3 (because these instances are immutable). Therefore we need to interpolate a whole Vector2/3 (we have access to them via the setter).
Object — Godot Engine (stable) documentation in English (`set` method)

Task 5 - Choosing the easing function as a parameter

Apart from linear interpolation, enable ease_in_quad, ease_out_quad and ease_in_out_quad.

We could implement this in four possible ways:

- Static functions - Every easing function has its own function, all the computation is nicely encapsulated.
    - However when we create a tween, we can pass a reference to a function (functions in GDScript are not first-class objects).
    - Or we can pass the name of the function as a string and then call the function based on the string using a general `call` function.
    - (In both cases we need to use a string with the name of the method which is not very convenient.)
- Enum - An enum contains all possible easing functions. Arguments for the tween creation are much cleaner (no need for creating a reference to a function, convenient intelli-sense) but then we need a large match statement (switch-case) to compute the correct function.
- Combination - We can use enum as a parameter and then use its values to index an array/dictionary of function names and call the function based on the string. This brings advantages of both approaches.

Unfortunately an enum cannot be used as a type for a type hint but we can still get a reasonable intelli-sense.

GDScript reference — Godot Engine (latest) documentation in English (match)
GDScript basics — Godot Engine (stable) documentation in English (referencing functions)
Object — Godot Engine (stable) documentation in English (calling function by string)
GDScript basics — Godot Engine (stable) documentation in English (enums)
Static typing in GDScript — Godot Engine (stable) documentation in English

Task 6 - Interpolation of color

Enable to interpolate color properties too.
Similarly to the Vector2/3, we cannot use a general setter to access individual components of Color, and therefore we need to interpolate a color as a whole.
Vector2 can be simply multiplied by a scalar, but for Color this is not possible and we need to handle it separately.

Color — Godot Engine (stable) documentation in English (`linear_interpolate` method)

Task 7 - On end callbacks

Functions in GDScript (from Godot 4) are now first-class objects! We can simply pass them as a parameter. Lambda functions function the same as referencing to class functions in GDScript, they do not need to be defined separately.

GDScript basics — Godot Engine (stable) documentation in English

Task 8 - Adding more easing functions

Add more easing functions:
ease_{in,out,in_out}_cubic
ease_{in,out,in_out}_sine
smoothstep3

To add smoothstep, either derive it (and use Horner's method!) or remember that it is a linear interpolation of ease_in_quad and ease_out_quad.

Easing Functions Cheat Sheet (easings.net)

Task 9 - Splines (with fluent syntax, method chaining) (AKA path following)

Enable running tweens across more than 2 points.

To accomplish this, add a function `then_to` on CustomTween which extends the spline with another part and returns the Tween object (fluent API).

The tween then remembers all the target values of each part and from the time computes which part is now active. Duration and an easing function given in the tween constructor are used for every part of the spline (duration is divided evenly).

E.g. when running a "position" tween over 2 seconds between points A, B, C; it should travel from A to B over 1 second and from B to C also over 1 second.

Tweens.gd - The `tween` function should return the created tween so that it is possible to chain additional `add_curve` function calls.
Method chaining - Wikipedia

Useful resources

Easing Functions Cheat Sheet (easings.net)
Interpolation — Godot Engine (stable) documentation in English
Beziers, curves and paths — Godot Engine (stable) documentation in English

Assets sources

Kenney • Space Shooter Redux