
# Description

**FastTweener** - is a one more simple tweener, but **entirely without memory allocation**!

Will be released in AssetStore soon.

Source of inspiration - DoTween. DoTween is a really powerfull and user-friendly Tween Engine, we use it and love it. But when we faced with extra memory allocation problem we decide to make our own solution, because **DoTween allocate a lot of memory**. This is connected not with pooling (DoTween has a good recycling system), but with DoTween allocate memory while working.

To see source or Benchmarks go to github.

# How to Use

## Get Started

### Importing

You can download `FastTweener` from this repository or from AssetStore (coming soon). You can unzip it anywhere in your Unity Assets folder, except Editor folder. No additional setups needed, `FastTweener` is ready to use!

### Namespace

To use `FastTweener` you need to add namespace in each class where you want to use it.

```
using DG.Tweening;
```

### Initialize

You can initialize `FastTweener` to setup some global options:

```
FastTweenerSettings settings = new FastTweenerSettings();

//this ease will be used for each Tween if nothing another ease
settings.DefaultEase = Ease.Linear; //default - Ease.OutQuad

//size of the pool of the Transform extensions like transform.Tw
settings.TransformExtensionsPoolSize = 32; //default - 16

//size of the pool of the Rigidbody extensions like rigidbody.Tw
settings.RigidbodyExtensionsPoolSize = 32; //default - 16

//size of the pool of the general Tweens (common + extensions)
settings.TaskPoolSize = 32; //default - 16

//if true - FastTweener will write a name of the GameObject in E
settings.SaveGameObjectName = true; //default - false

//FastTweener will write Warnings if actual fps is lower then th
```

```
settings.CriticalFpsToLogWarning = 50; //default - 30

FastTweener.Init(settings);
```

If you don't do that `FastTweener` will be auto-initialized with the default settings. To get initialization status use `bool FastTweener.IsInitialized` property.

**WARNING: If you want to use manual initialization you need to do it before creating your first Tween!**

## Create a Tween

There is a several ways to create a new Tween:
* Using FastTweener class

```
FastTweener.Float(floatFrom, floatTo, duration, x => { /* Your l
FastTweener.Vector3(vectorFrom, vectorTo, duration, x => { /* Yo
FastTweener.Schedule(delay, () => { /* Your logic here */ });
```

- Using Extensions for Transform class

```
transform.TweenMove(vectorTo, duration);
transform.TweenMoveX(floatTo, duration);
transform.TweenMoveY(floatTo, duration);
transform.TweenMoveZ(floatTo, duration);

transform.TweenLocalMove(vectorTo, duration);
transform.TweenLocalMoveX(floatTo, duration);
transform.TweenLocalMoveY(floatTo, duration);
transform.TweenLocalMoveZ(floatTo, duration);

transform.TweenScale(vectorTo, duration);
transform.TweenScaleX(floatTo, duration);
transform.TweenScaleY(floatTo, duration);
transform.TweenScaleZ(floatTo, duration);

transform.TweenRotate(vectorTo, duration);
transform.TweenLocalRotate(vectorTo, duration);
```

- Using Extensions for Rigidbody class

```
rigidbody.TweenMove(vectorTo, duration);
rigidbody.TweenMoveX(floatTo, duration);
rigidbody.TweenMoveY(floatTo, duration);
rigidbody.TweenMoveZ(floatTo, duration);

rigidbody.TweenRotate(vectorTo, duration);
```

All extension methods made without closure and don't allocate memory too.

When you create a Tween it will start to play automatically.

Each method has required parameters:

```
T endValue              //finish value of Tween. Vector3 or floa
float duration          //duration of Tween in seconds

// Only for Tweens created via FastTweener class:

T startValue            //finish value of Tween. Vector3 or floa
Action<T> callback      //Action<Vector3> or Action<float> depen
```

And optional parameter:

```
Ease ease               //ease for tweening (default one you set
bool ignoreTimescale    //should Tween ignore timescale (default
Action onComplete       //the callback will be called when Tween
```

`FastTweener.Schedule` is the only exception. It contains only three parameters:

```
float delay             //delay before Action executing
Action callback         //action to execute after Delay
bool ignoreTimescale    //should Tween ignore timescale (default
```

Each method contains overloads for each combination of optional parameters. For example:

```
//No optional parameters
transform.TweenMove(vectorTo, duration);

//One parameter:
//ease
```

```
transform.TweenMove(vectorTo, duration, Ease.InElastic);
//ignoreTimescale
transform.TweenMove(vectorTo, duration, true);
//onComplete
transform.TweenMove(vectorTo, duration, OnComplete);

//Parameters combinations:
//ease & ignoreTimescale
transform.TweenMove(vectorTo, duration, Ease.InElastic, true);
//ease & onComplete
transform.TweenMove(vectorTo, duration, Ease.InElastic, OnComple
//ignoreTimescale & onComplete
transform.TweenMove(vectorTo, duration, true, OnComplete);
```

## Work with Tween

You can get or set Tween parameters after Tween creation. To make it you should save `FastTween` instance during Tween creation and call his methods.

```
FastTween tween = transform.TweenLocalMoveY(floatTo, duration);

uint id = tween.Id;

Ease ease = tween.GetEase();
tween.SetEase(Ease.InCirc);

bool ignoreTimeScale = tween.GetIgnoreTimeScale();
tween.SetIgnoreTimeScale(true);

tween.OnComplete(() => Debug.Log("Done!"));

bool isAlive = tween.IsAlive();
tween.Kill();
```

Also you can use chaining (Linq) style:

```
tween.SetEase(Ease.Linear).SetIgnoreTimeScale(true).OnComplete(d
```

Under the hood `FastTween` call static methods of `FastTweener` class, so you can use it too. It is the same.

```
FastTween tween = transform.TweenLocalMoveY(floatTo, duration);

Ease ease = FastTweener.GetEase(tween);
FastTweener.SetEase(tween, Ease.InCirc);

bool ignoreTimeScale = FastTweener.GetIgnoreTimeScale(tween);
FastTweener.SetIgnoreTimeScale(tween,true);

FastTweener.SetOnComplete(tween,() => Debug.Log("Done!"));

bool isAlive = FastTweener.IsAlive(tween);
FastTweener.Kill(tween);
```

**WARNING: Read Performance hints before using this methods for the best performance!**

## Ease Types

To set Default Ease that was set in the settings during Initialization use `tween.SetEase(Ease.Default)`.

You can use one of the next Eases:
*Linear*
InSine
*OutSine*
InOutSine
*InQuad*
OutQuad
*InOutQuad*
InCubic
*OutCubic*
InOutCubic
*InQuart*
OutQuart
*InOutQuart*
InQuint
*OutQuint*
InOutQuint
*InExpo*
OutExpo
*InOutExpo*

InCirc
*OutCirc*
InOutCirc
*InElastic*
OutElastic
*InOutElastic*
InBack
*OutBack*
InOutBack
*InBounce*
OutBounce
* InOutBounce

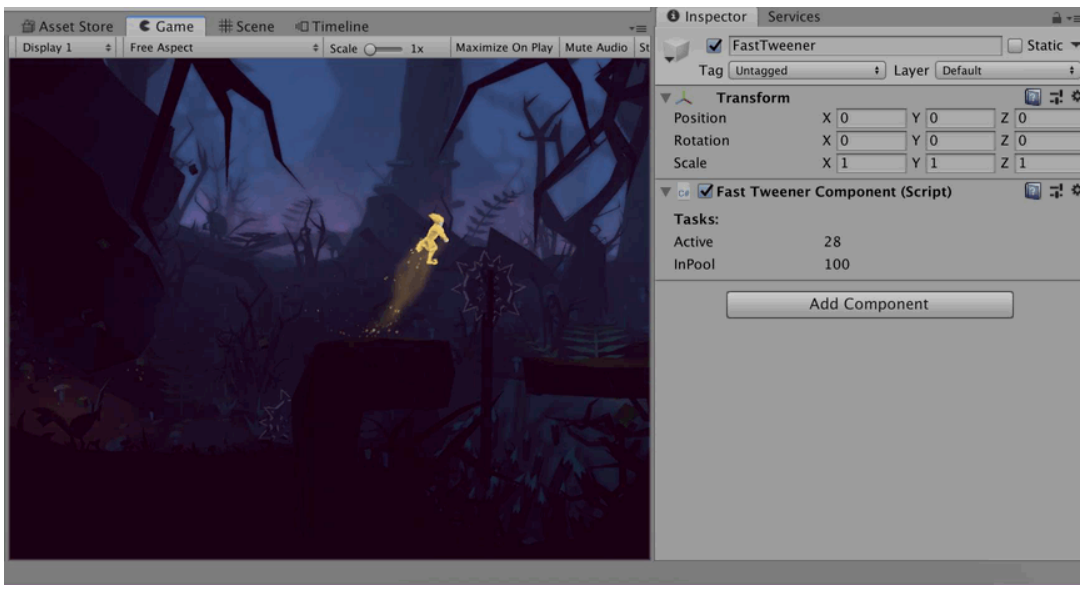
Formulas for simple eases were found at gizma.com (Action Script 3)

Bounce Eases from tweenman-as3 GitHub repository (Action Script 3)

Elastic and Back formulas were taken from here processing penner easing GitHub repository (Java)

# Monitoring

`Real-time Monitoring` allow you to see the actual count of alive Tweens and count of Tweens in the pool. To see this data find GameObject with name `FastTweener` in the root of `DontDestroyOnLoad` section in the `Hierarchy` window during the Play mode.



# Performance Hints

`FastTween` is just a struct with `Tween Task` id. We can't set instance of `Tween Task` to `FastTween` instance because in future this Tween will be used for another Tween. So all functions `IsActive`, `GetEase`, `SetEase`, `GetIgnoreTimeScale`, `SetIgnoreTimeScale`, and `OnComplete` required to find a `Tween Task` in the `Tween Tasks` list. But when you send these parameters during a Tween creating it won't take additional time.

For example, this code is faster:

```
FastTween tween = FastTweener.Float(-3, 3, 0.5f, value => DoSome
```

Than this code:

```
FastTween tween = FastTweener.Float(-3, 3, 0.5f, value => DoSome
tween.SetEase(Ease.OutBounce);
tween.OnComplete(OnComplete);
```

For the same reasons, receiving data from `FastTween` can be not so fast as we want. So, it will be better to cache Tween parameters if it’s possible.

For example, this code is faster:

```
private Ease tweenEase;
public void SomeMethod(FastTween someTween)
{
    tweenEase = someTween.GetEase();
}
public void Update()
{
    if (tweenEase == Ease.Linear)
    {
        //Do some logic
    }
}
```

Than this code:

```
private FastTween tween;
public void SomeMethod(FastTween someTween)
{
    tween = someTween;
}
public void Update()
{
    if (tween.GetEase() == Ease.Linear)
    {
        //Do some logic
    }
}
```