Smart Timers Manager Read me

What is a timer?

A timer is a function that is activated after a given period of time. The function can be executed as many times as you want.

What is Smart Timers Manager?

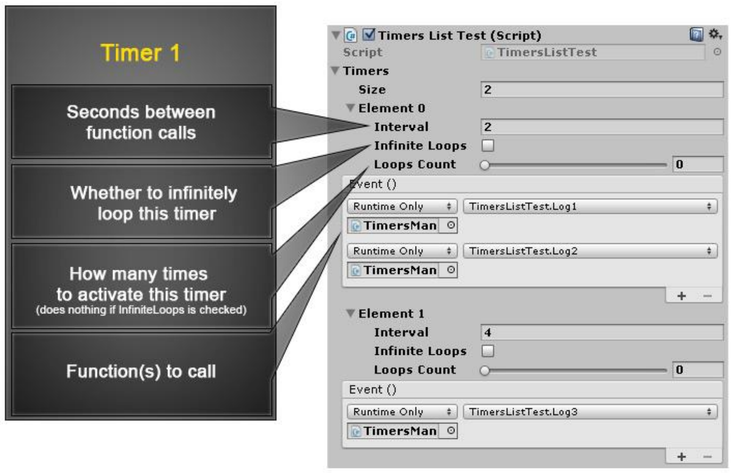
Smart Timers Manager is a C# package for UnityEngine that allows you to dynamically add, remove and monitor any active timer with ease.

Setup

No specific setup needed anymore.

Editor Usage

Smart Timers Manager comes with a `GUI_TimersList` component that allows you to add timers from inspector.


Timer 1
Seconds between function calls
Whether to infinitely loop this timer
How many times to activate this timer (does nothing if InfiniteLoops is checked)
Function(s) to call

```
Timers List Test (Script)
Script          TimersListTest
Timers
  Size          2
  Element 0
    Interval        2
    Infinite Loops  ☐
    Loops Count     0
  Event ()
  Runtime Only    TimersListTest.Log1
  TimersMan
  Runtime Only    TimersListTest.Log2
  TimersMan
  Element 1
    Interval        4
    Infinite Loops  ☐
    Loops Count     0
  Event ()
  Runtime Only    TimersListTest.Log3
  TimersMan
```

C# Usage

The **Smart Timers Manager**’s main classes are `Timer` and `TimersManager` which are in the `Timers` namespace. Before starting to add timers make sure you included the namespace `using Timers;` A timer can be added by accessing one of the SetTimer() static methods from `TimersManager` class.

IMPORTANT!
By default**,** setting the same timer method multiple times, will clear the previously running ones. If you want to stack timers, either use Lambda expression or set the optional param **overrideOld** to false. Note that if you are setting a timer by passing a Timer object, and it was previously started before and didn't finish, it will be overriden regardless of overrideOld's value

```
var timer = new Timer(this, 1f, 5, TimerFunc);
TimersManager.SetTimer(timer);
TimersManager.SetTimer(timer, overrideOld: false); // this will have no
effect and the timer will be overrdien regardless

// these will NOT stack as overrideOld is true by default
TimersManager.SetTimer(this, 5f, TimerFunc);
TimersManager.SetTimer(this, 3f, TimerFunc);


// these will stack
TimersManager.SetTimer(this, 5f, TimerFunc, overrideOld: false);
TimersManager.SetTimer(this, 3f, TimerFunc, overrideOld: false);


// these will stack too
TimersManager.SetTimer(this, 5f, () => TimerFunc);
TimersManager.SetTimer(this, 3f, () => TimerFunc);
```

Example:

```
using UnityEngine;
using Timers;

public class Test : MonoBehaviour
{
    void Timer1()
    {
        Debug.Log("Test");
    }

    void ClearTimer1()
    {
        // Remove Timer1
        TimersManager.ClearTimer(Timer1);
    }

    void ForgottenTimer()
    {}

    void Start()
    {
        // Log "Test" once every 2 seconds by calling Timer1()
        TimersManager.SetLoopableTimer(this, 2f, Timer1);

        // Call ClearTimer1() after 5 seconds
        TimersManager.SetTimer(this, 5f, ClearTimer1);

        // Call ForgottenTimer() once every second 50 times
        TimersManager.SetTimer(this, 1f, 50, ForgottenTimer);

        // Log "Remaining time: 50sec" which is 1f * 50
        Debug.Log("Remaining time: "+
                   TimersManager.RemainingTime(ForgottenTimer) +"sec");

        // Destroy this component after 10 seconds
        TimersManager.SetTimer(this, 10f, delegate { Destroy(this); });
    }
}
```

Notice that `ForgottenTimer` lasts 50 seconds but the object is destroyed after 10 seconds. Because we set `this` as the timer's owner, `TimersManager` can automatically remove `ForgottenTimer` as soon as the timer's owner is garbage collected and becomes `null`.

Don’t worry about adding timers with the same name from different classes, however, setting an active timer twice will override the previous one.

Example:

```
using Timers;

public class Class1
{
    public Class1()
    {
        // Call Timer1() from this class once every second
        Timer t1 = new Timer(1f, Timer.INFINITE, Timer1);
        TimersManager.SetTimer(this, t1);
    }

    void Timer1() { }
}

public class Class2
{
    public Class2()
    {
        // Call Timer1() from this class once every 2 seconds
        Timer t1 = new Timer(2f, Timer.INFINITE, Timer1);
        TimersManager.SetTimer(this, t1);



        // Set a timer that calls Timer2() once every second
        Timer t2 = new Timer(1f, Timer.INFINITE, Timer2);
        TimersManager.SetTimer(this, t2);

        // This will override the previous one,
        // calling Timer2() after 2 seconds ONLY ONCE!
        TimersManager.SetTimer(this, 2f, Timer2);
    }

    void Timer1() { }
    void Timer2() { }
}
```

Notice that you don’t have to extend `MonoBehaviour` in order to manage timers.

Timer comparison

A timer is greater than another timer if it has a higher frequency (calls per second), thus lower interval value.

Example:

```
// a timer that calls Timer1() every second
Timer A = new Timer(1f, Timer.INFINITE, Timer1);

// a timer that calls Timer2() 10 times per second
Timer B = new Timer(0.1f, Timer.INFINITE, Timer2);

if (A < B) // this is true
```

This is just a preview of how **Smart Timers Manager** works, but you can do much more. See the PDF documentation for complete interface.