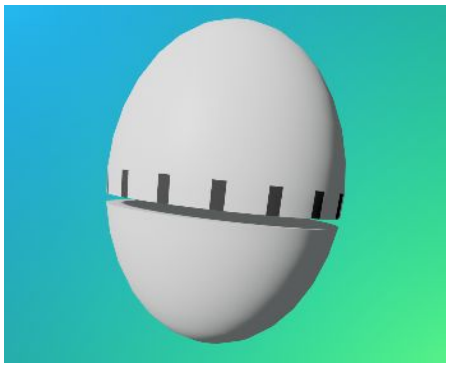
Cikoria



Egg Timer

Documentation

Getting Started

This section will help you get up and running quickly. The Egg Timer interface is really simple but can also be really powerful and help keep your code clean and readable.

Executing Actions for a Duration

You can execute an action by calling *EggTimer.Execute*. Here is an example code snippet that prints “Hello World” every frame for one second:

```
EggTimer.Instance.Execute(() =>
{
    Debug.Log(“Hello World”);
})
.ForDuration(1f);
```

The first argument we passed in is a *Lambda Action* and has the following syntax:

```
() => {series-of-statements}
```

Inside the *series-of-statements* block, you execute different statements. In this case, we are just printing “Hello World” to the console.

Notice that the *ForDuration* method is chained to the *Execute* method. This type of method chaining is known as the *Named Parameter Idiom.* This allows you to chain methods in any order you like.

You can also pass in an optional second argument for the time style. There are four different time styles that will affect the way the duration is interpreted by the egg timer:

- TimeStyle.Scaled - The duration will be multiplied by the application’s time scale and the action will update every frame.
- TimeStyle.Unscaled - The duration will be unaffected by the application’s time scale and the action will update every frame.

- TimeStyle.FixedScaled - The duration will be multiplied by the application's time scale and the action will update every fixed frame (physics update). This is useful when you want to execute actions that call physics methods such as Rigidbody.AddForce.
- TimeStyle.FixedUnscaled - The duration will be unaffected by the application's time scale and the action will update every fixed frame (physics update).

Executing an Action with a Delay

What if you do not want to execute an action immediately? In this case, you can specify a delay for your action. Here is an example code snippet that waits for one second and then prints "Hello World" to the console once:

```
EggTimer.Instance.Execute(() =>
{
    Debug.Log("Hello World");
})
.WithDelay(1f);
```

You can also chain this method with the previously used method in any order you like:

```
EggTimer.Instance.Execute(() =>
{
    Debug.Log("Hello World");
})
.WithDelay(1f)
.ForDuration(1f);
```

Interrupting Actions

If you want to interrupt an action, you first need to have a reference to the action that has been executed and then call the *EggTimer.Remove* method:

```
var action = EggTimer.Instance.Execute(() => { /* … */
}).WithDuration(1f);

EggTimer.Instance.Remove(action);
```

The last line removes the action from being executed. This will also remove the action even if it has not started invoking yet. So if you execute an action with a delay of one second and then immediately remove the action, then the action will never be invoked.

Execute Logic when an Action Finishes

If you want to execute some logic when an action has finished, you can make use of the *OnFinish* method. Here is an example code snippet:

```
EggTimer.Instance.Execute(() =>
{
    /* … */
})
.WithDuration(5f)
.OnFinish(() =>
{
    Debug.Log(“This action is now finished.”);
});
```

Like we did in the *Execute* method, the first argument we passed is a Lambda action. The statements inside this action will get executed when first action is finished. In this case, a message to the console will be printed after five seconds.

Contact

If you notice any bugs or have any feature requests, you can reach us on: studiocikoria@gmail.com.

Bug Reporting

When reporting a bug, you can send us a message describing the bug and where it occurred. It would also be really helpful if you also included the editor log. You can access the editor log by right-clicking on the *Console* tab and clicking *Open Editor Log*.