author: Igor Chebotar
contact: wolk.fo@gmail.com

COROUTINE EXTENSIONS

BY SIMPLE MAN

Часто во время написания кода возникает потребность отсрочить выполнение операции на какое-то время или до момента, пока некое условие не будет выполнено. Наши расширения призваны упростить эти операции до вызова одной команды (Delay, WaitUntil, RepeatUntil). Плагин не требует наследования вашего класса от некого специального и каких-либо других манипуляций. Вы спокойно можете загрузить coroutine extension в существующий проект и без пользоваться без ограничений уже сейчас!

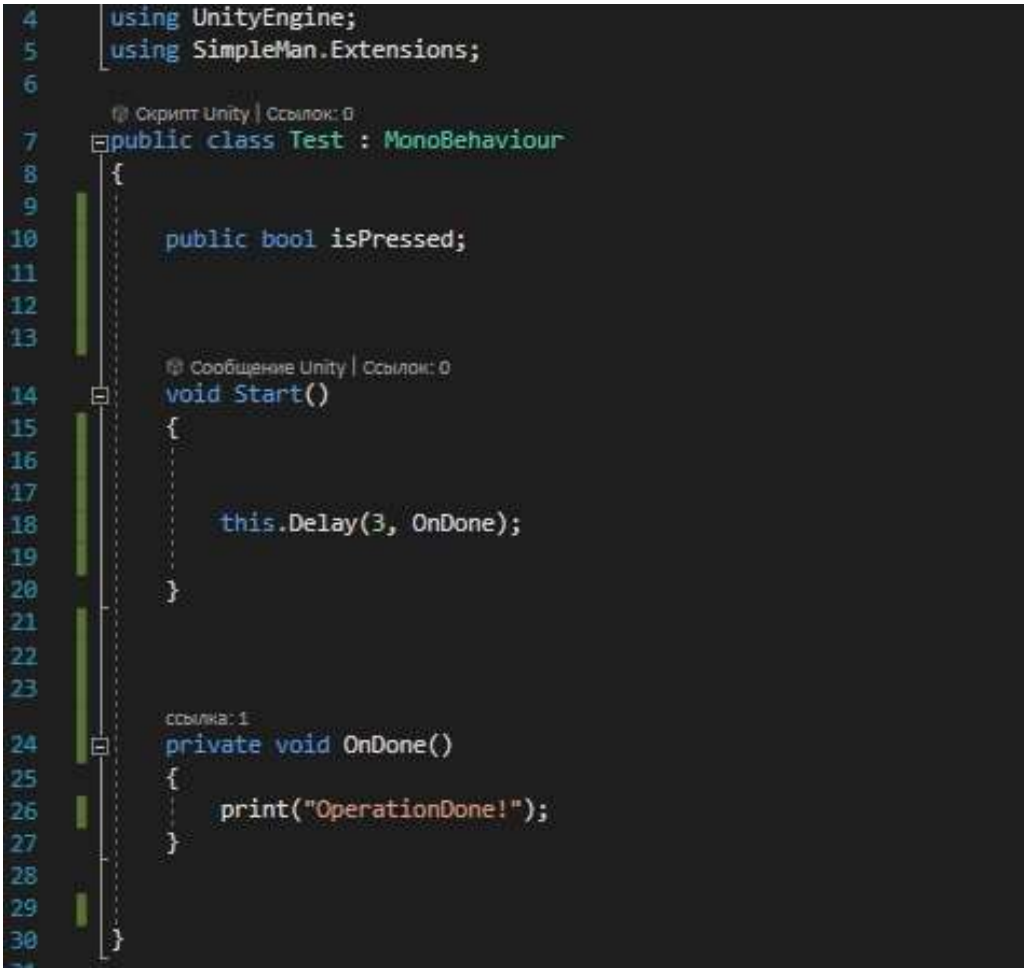
КАК ИСПОЛЬЗОВАТЬ DELAY?
Чтобы отсрочить выполнение операции на определенное время воспользуемся командой Delay.

Для примера создадим новый скрипт Test, в нем объявим метод типа void и назовем его OnDone (название роли не играет). Этот метод выводит сообщение в консоль.

Далее в методе Start вызовем команду Delay, где первый параметр отвечает за время задержки в секундах, а второй - делегат метода, который должен быть вызван по истечению времени таймера.
Детальный:

```
using UnityEngine;
using SimpleMan.Extensions;

Скрипт Unity | Ссылок: 0
public class Test : MonoBehaviour
{

    public bool isPressed;



    Сообщение Unity | Ссылок: 0
    void Start()
    {

        Action l_action = OnDone;

        this.Delay(3, l_action);

    }


    ссылка: 1
    private void OnDone()
    {
        print("OperationDone!");
    }


}
```

Быстрый:



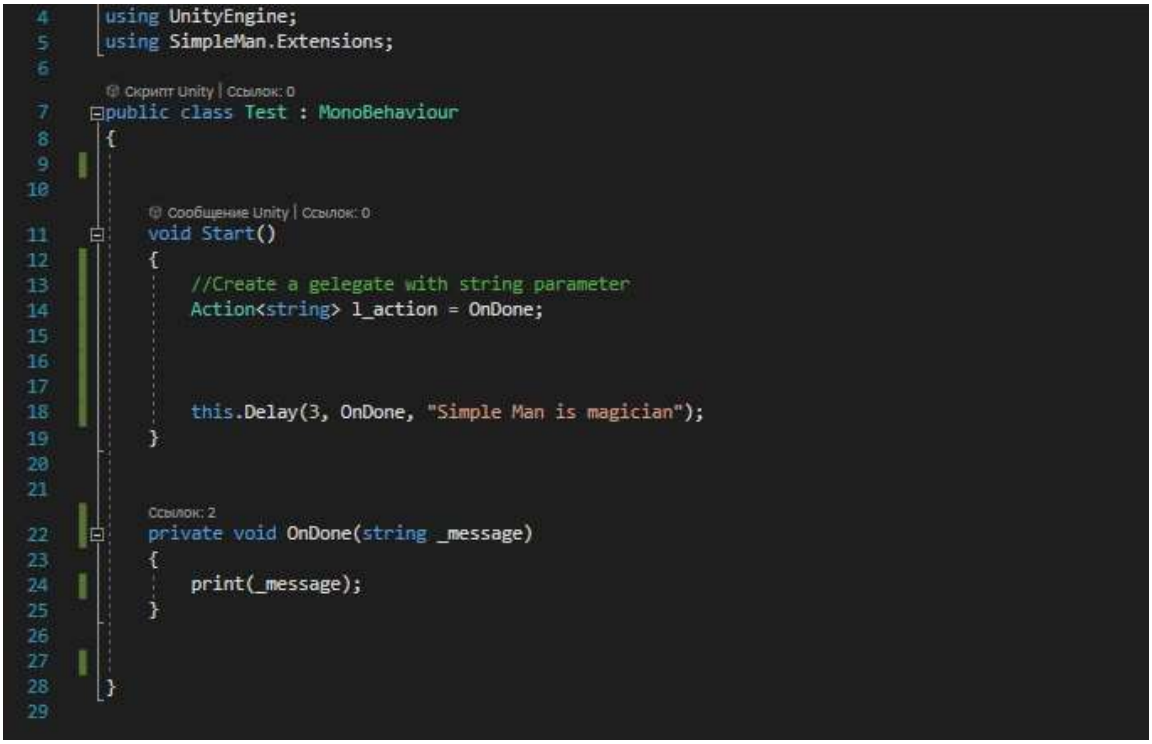
```
using UnityEngine;
using SimpleMan.Extensions;

Скрипт Unity | Ссылок: 0
public class Test : MonoBehaviour
{

    public bool isPressed;



    Сообщение Unity | Ссылок: 0
    void Start()
    {


        this.Delay(3, OnDone);

    }



    ссылка: 1
    private void OnDone()
    {
        print("OperationDone!");
    }


}
```

Итог: при нажатии на “Play” через 3 секунды метод OnDone выведет сообщение в консоль.

КАК ИСПОЛЬЗОВАТЬ DELAY С ПЕРЕДАЧЕЙ ПАРАМЕТРА?

Для передачи аргумента в метод OnDone(string _message) необходимо перед вызовом команды Delay создать соответствующий экземпляр делегата, отвечающий типу данных, который принимает конечный метод, в нашем случае это string.

Далее как обычно вызываем метод Delay и передаем ему наш делегат в качестве второго аргумента и строку в качестве третьего.



КАК ИСПОЛЬЗОВАТЬ WAIT UNTIL?

Иногда нам нужно отсрочить выполнение операции не на определенное время, а до того момента, пока не выполнится некое условие. В этом нам поможет Wait Until.

Детальный пример:

```
using UnityEngine;
using SimpleMan.Extensions;

Скрипт Unity | Ссылок: 0
public class Test : MonoBehaviour
{

    public bool isPressed;



    Сообщение Unity | Ссылок: 0
    void Start()
    {

        //Lambda expression that returning bool value
        Func<bool> l_condition = () => isPressed;

        //Delegate of OnDone method
        Action l_action = OnDone;


        this.WaitUntil(l_condition, l_action);
    }


    ссылка: 1
    private void OnDone()
    {
        print("OperationDone!");
    }


}

```

Быстрый:

```
using UnityEngine;
using SimpleMan.Extensions;

Скрипт Unity | Ссылок: 0
public class Test : MonoBehaviour
{

    public bool isPressed;



    Сообщение Unity | Ссылок: 0
    void Start()
    {

        this.WaitUntil(() => isPressed, OnDone);

    }


    ссылка: 1
    private void OnDone()
    {
        print("OperationDone!");
    }


}

```

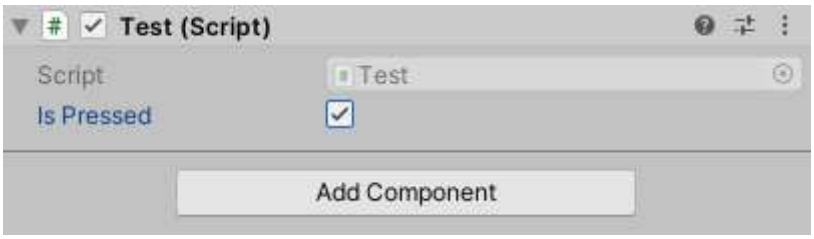
В данном примере условием выступит публичное поле IsPressed. Задача: вызвать метод OnDone, когда isPressed примет значение true.

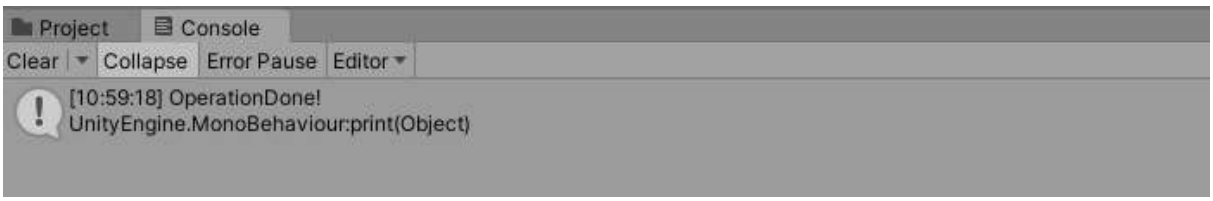
Для этого в метода Start создадим экземпляр делегата Func<bool> и через лямбда выражение передадим IsPressed в качестве условия.

Далее по аналогии с Delay вызываем команду WaitUntil и передаем делегат-условие в качестве первого аргумента, а делегат метода OnDone в качестве второго.

Также можно воспользоваться перегрузкой и передать параметр для конечного метода, аналогично предыдущему примеру.

Проверка:
По нажатию на IsPressed в консоли появляется сообщение





КАК ИСПОЛЬЗОВАТЬ WAIT UNTIL С ЗАДЕРЖКОЙ?

Встречаются также ситуации, которые требуют ресурсозатратных операций, которые могут выступать в качестве условий. Например если мы хотим взаимодействовать с компонентом объекта, но по какой-то причине этот компонент пока отсутствует на объекте. Известно, что GetComponent - дорогая операция и мы не имеем права проверять наличие компонента каждый кадр, пока ждем его появления. Однако можем проверять это условие каждые полсекунды, без особых потерь производительности.
В таких случаях нам поможет WaitUntil с задержкой.
Работает он так же как и обычный, но в качестве последнего аргумента при вызове метода можно выбрать время задержки в секундах.

Пример:

```
using UnityEngine;
using SimpleMan.Extensions;

Скрипт Unity | Ссылок: 0
public class Test : MonoBehaviour
{

    public bool isPressed;



    Сообщение Unity | Ссылок: 0
    void Start()
    {

        this.WaitUntil(() => isPressed, OnDone, 0.5f);

    }


    ссылка: 1
    private void OnDone()
    {
        print("OperationDone!");
    }


}

```

КАК ИСПОЛЬЗОВАТЬ REPEAT UNTIL?

Часто возникают ситуации, когда нужно повторять какое либо действие до определенного момента или же вовсе бесконечно. Примером может служить использование Raycast'а. Известно, что использование Raycast в каждом кадре серьезно снизит производительность игры и зачастую это делать вовсе необязательно, ведь того же эффекта можно добиться используя Raycast каждые 0.2 секунды или реже.

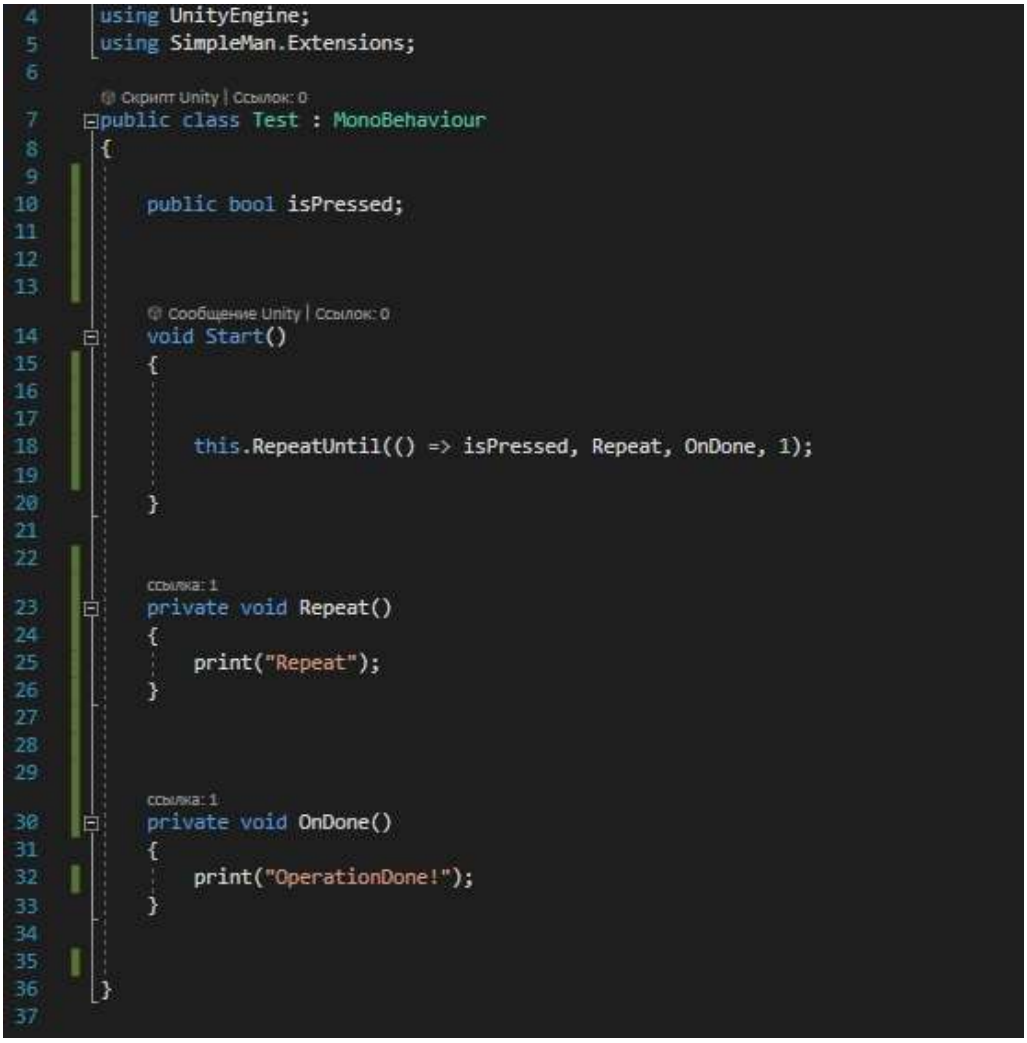
RepeatUntil позволяет реализовать такой подход путем написания одной строчки.

Задача: каждую секунду вызывать метод Repeat пока IsPressed не примет значение true, затем вызвать метод OnDone.

Детальный пример:

```
using UnityEngine;
using SimpleMan.Extensions;

Скрипт Unity | Ссылок: 0
public class Test : MonoBehaviour
{

    public bool isPressed;



    Сообщение Unity | Ссылок: 0
    void Start()
    {

        Func<bool> l_condition = () => isPressed;
        Action l_repeatAction = Repeat;
        Action l_doneAction = OnDone;
        float l_delay = 1;


        this.RepeatUntil(l_condition, l_repeatAction, l_doneAction, l_delay);

    }


    ссылка: 1
    private void Repeat()
    {
        print("Repeat");
    }



    ссылка: 1
    private void OnDone()
    {
        print("OperationDone!");
    }


}
```

Быстрый:



```
using UnityEngine;
using SimpleMan.Extensions;

Скрипт Unity | Ссылок: 0
public class Test : MonoBehaviour
{

    public bool isPressed;



    Сообщение Unity | Ссылок: 0
    void Start()
    {


        this.RepeatUntil(() => isPressed, Repeat, OnDone, 1);

    }


    ссылка: 1
    private void Repeat()
    {
        print("Repeat");
    }



    ссылка: 1
    private void OnDone()
    {
        print("OperationDone!");
    }


}

```

Задача: Бесконечно вызывать метод Repeat через каждую секунду

```
using UnityEngine;
using SimpleMan.Extensions;

Скрипт Unity | Ссылок: 0
public class Test : MonoBehaviour
{

    public bool isPressed;



    Сообщение Unity | Ссылок: 0
    void Start()
    {


        this.RepeatUntil(() => true, Repeat, null, 1);

    }


    ссылка: 1
    private void Repeat()
    {
        print("Repeat");
    }



    Ссылок: 0
    private void OnDone()
    {
        print("OperationDone!");
    }


}

```

КАК ОСТАНОВИТЬ ВЫПОЛНЕНИЕ ОПЕРАЦИИ?

Методы Delay, WaitUntil и RepeatUntil возвращают Coroutine. Остановить выполнение операции можно путем кэширования полученной корутины и ее последующей остановки стандартным Unity методом “StopCoroutine”.