
C#

C# — объектно-ориентированный язык программирования. Разработан в 1998—2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как язык разработки приложений для платформы Microsoft .NET Framework.

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Переняв многое от своих предшественников — языков C++, Pascal, Модула, Smalltalk и в особенности Java — C#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает множественное наследование классов (между тем допускается множественное наследование интерфейсов).

Чтобы мы могли программировать, нам нужно что-то, и это что-то называется переменными. Тоесть вся суть программирования заключается в оперировании некими изменяющимися (и не изминяющимися) данными. Этим данным мы задаем тип, который определяет, что это такое и что с этим можно делать.

Тип данных представляет собой классификацию информационных сущностей (например, таких как значения или выражения), определяющую возможность их использования в рамках заданной формальной системы. Наиболее принципиально различимых, хотя и не противоречащих друг другу, определений два:

декларативное — тип есть множество допустимых значений, которые могут принимать данные, принадлежащие к этому типу;

процедурное — тип определяется поведением, т.е. набором действий, которые можно осуществлять над данными, принадлежащими к этому типу.

Объяснение на ~~пальцах~~ коробках.

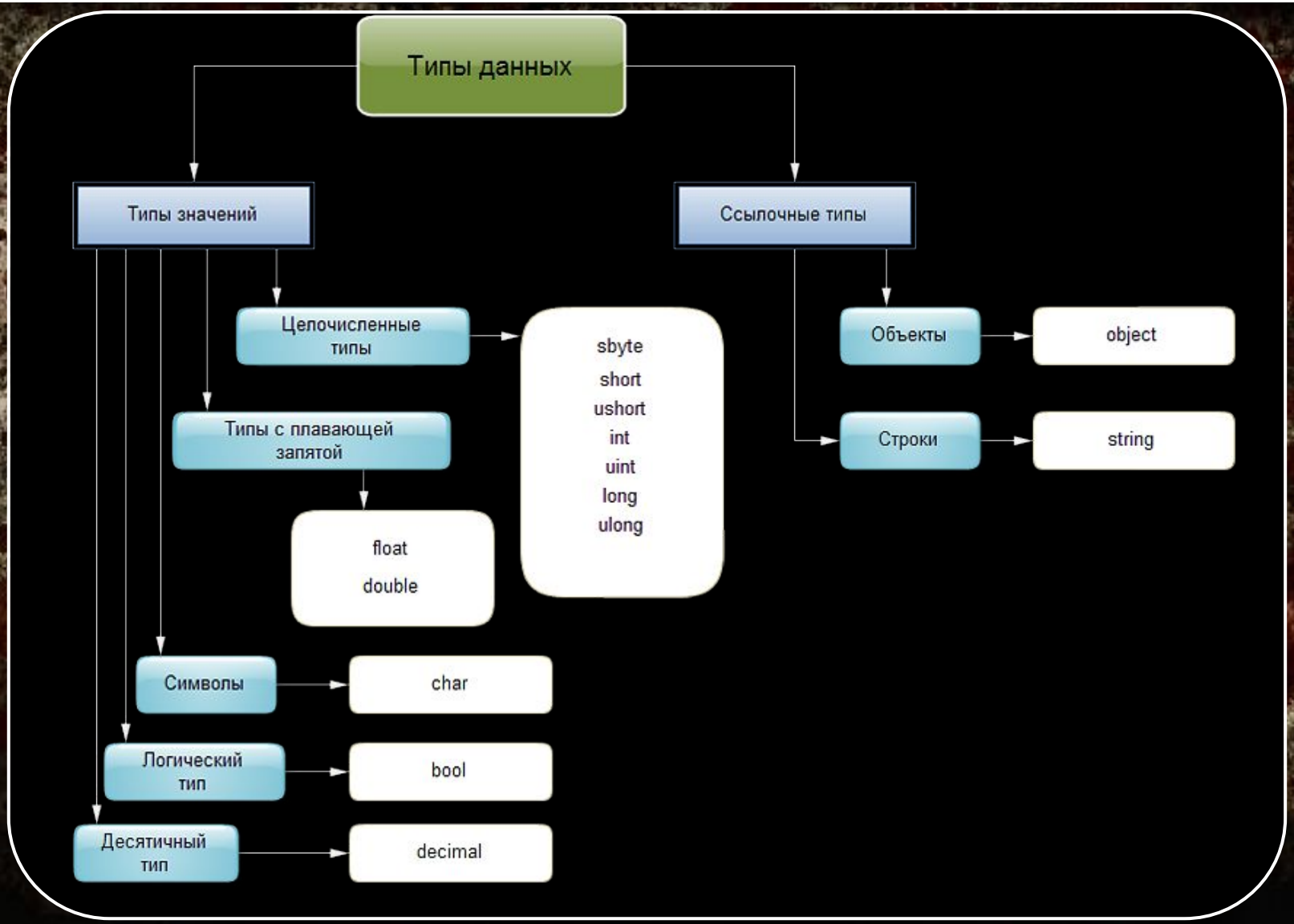
У нас есть 3 коробки:

1. Коробка слева может содержать в себе только целые предметы, например яблоки, которых может быть 1,2,3 и т.д.
2. Коробка по центру содержит семечки, которые меряются не по штучно, а по весу, например 2,5 кг. или 4,2 к.г. Причем семечки и яблоки смешивать нельзя, точнее можно, но для этого нужно произнести специальное заклинание, которое превратит, на наше усмотрение, либо семечки в яблоки, либо яблоки в семечки.
3. А третья коробка содержит в себе темную материю, которая ни как не сходится ни с яблоками ни с семечками, но может их принять как нечто целое, которое потеряет свойство яблок и семечек. Вот такие пироги.

Переменная – это именованная область памяти. В переменную можно записывать данные и считывать. Данные, записанные в переменной, называются значением переменной. Си-шарп – язык жесткой типизации. Каждая переменная должна быть определенного типа данных.

В C# имеются две общие категории встроенных типов данных: типы значений и ссылочные типы. Они отличаются по содержимому переменной. Концептуально разница между ними состоит в том, что тип значения (value type) хранит данные непосредственно, в то время как ссылочный тип (reference type) хранит ссылку на значение.

Эти типы сохраняются в разных местах памяти: типы значений сохраняются в области, известной как стек, а ссылочные типы — в области, называемой управляемой кучей.


Типы данных
Типы значений
Ссылочные типы
Целочисленные типы
sbyte
short
ushort
int
uint
long
ulong
Типы с плавающей запятой
float
double
Символы
char
Логический тип
bool
Десятичный тип
decimal
Объекты
object
Строки
string

Тип	Область значений	Размер
sbyte	-128 до 127	Знаковое 8-бит целое
byte	0 до 255	Беззнаковое 8-бит целое
char	U+0000 до U+ffff	16-битовый символ Unicode
bool	true или false	1 байт
short	-32768 до 32767	Знаковое 16-бит целое
ushort	0 до 65535	Беззнаковое 16-бит целое
int	-2147483648 до 2147483647	Знаковое 32-бит целое
uint	0 до 4294967295	Беззнаковое 32-бит целое
long	-92233720368547758О8 до 9223372036854775807	Знаковое 64-бит целое
ulong	0 до 184467440737095516l5	Беззнаковое 64-бит целое
float	±1,5*10-45 до ±3,4*1033	4 байта, точность — 7 разрядов
double	±5*10-324 до ±1,7*10306	8 байтов, точность — 16 разрядов
decimal		16 байт, точность — 28 разрядов

```
int a; // объявляем переменную a типа int
a = 5; // записываем в переменную a число 5
int b, c, m; // объявить можно сразу несколько переменных через запятую
bool d; // объявляем переменную d типа bool (true или false)
d = true; // записываем в переменную d значение true (истинна)
long e = 10; // при объявлении переменной можно сразу же задавать ей значение, это называется инициализацией
float f = 5.5f; // чтобы записать число с плавающей точкой типа float, нужно после значения добавлять суффикс f.
char g = 'g'; // объявление символьной переменой g с ее инициализацией значением символа 'g‘
string x= "azaza";
//Однострочный комментарий (используется для комментариев к коду или для скрытия на некоторое время строки кода, чтобы она не выполнялась)
```

- Не используйте русские буквы в именах переменных
- В имени переменной не должно быть пробелов (пробелы можем заменить знаком _)
- Имя переменной не должно начинаться с цифры
- Имя переменной должно быть адекватно, т.е. Дерево называйте – tree, машину – car и т.д. Старайтесь не использовать глупые и не отражающие сущность имена: a, b, x. Такие имена конечно можно использовать, но, например, в математических расчетах, циклах и т.д. А для сущьностей реального мира, лучше использовать понятные имена.
- Не используйте системные имена, т.е. Мы не можем написать int int и т.д.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            //Тащемта, комментарий, например
            Console.WriteLine("Hello, bro!");
        }
    }
}
```

```
Пространство имен
Пространство имен
Пространство имен
Пространство имен

Пространство имен для класса
{
    Наш новый класс
    {
        Точка входа в программу
        {
            //Тащемта, комментарий, например
            Вывод в консоль сообщения: Hello, bro!
        }
    }
}
```

using System; (и т.д.)

Эта строка означает, что в программе используется пространство имен System. В C# пространство имен определяет область объявлений. Благодаря пространству имен одно множество имен отделяется от других. По существу, имена, объявляемые в одном пространстве имен, не вступают в конфликт с именами, объявляемыми в другом пространстве имен. В анализируемой программе используется пространство имен System, которое зарезервировано для элементов, связанных с библиотекой классов среды .NET Framework, применяемой в C#. Ключевое слово using просто констатирует тот факт, что в программе используются имена в заданном пространстве имен.

namespace ConsoleApplication1

Объявляет пространство имен, с которым должен быть ассоциирован класс. Весь код в последующих фигурных скобках рассматривается как принадлежащий этому пространству имен. Оператор using специфицирует пространство имен, которое должен просматривать компилятор в поисках классов, упомянутых в коде, но не определенных в текущем пространстве имен. Это служит тем же целям, что оператор import в Java и using namespace в C++.

```
class Program
{
```

ключевое слово class служит для объявления вновь определяемого класса. Класс является основной единицей инкапсуляции в C#, а Program — это имя класса. Определение класса начинается с открывающей фигурной скобки "{" и оканчивается закрывающей фигурной скобкой "}". Элементы, заключенные в эти фигурные скобки, являются членами класса. Не вдаваясь пока что в подробности, достаточно сказать, что в C# большая часть действий, выполняемых в программе, происходит именно в классе.

```
static void Main(string[] args)
```

Метод Main() является точкой входа в программу. Формально класс, в котором определяется метод Main(), называется объектом приложения. Хотя в одном исполняемом приложении допускается иметь более одного такого объекта (это может быть удобно при проведении модульного тестирования). Обратите внимание, что в сигнатуре метода Main() присутствует ключевое слово static. Область действия статических (static) членов охватывает уровень всего класса (а не уровень отдельного объекта) и потому они могут вызываться без предварительного создания нового экземпляра класса. void используется тогда, когда функции не надо возвращать какое либо значение. Допустим ей надо только что нибудь посчитать и вывести результат нигде не сохраняя. Внутри метода Main() используется метод предопределенного класса Console, в частности WriteLine() - выводит на экран строку.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            //Тащемта, комментарий, например
            Console.WriteLine("Hello, bro!");
        }
    }
}
```

```
Пространство имен
Пространство имен
Пространство имен
Пространство имен

Пространство имен для класса
{
    Наш новый класс
    {
        Точка входа в программу
        {
            //Тащемта, комментарий, например
            Вывод в консоль сообщения: Hello, bro!
        }
    }
}
```

В юнити все намного проще и нам не нужно беспокоиться о, например классовых пространствах имен и точках входа.

```
using UnityEngine;
using System.Collections;

public class testone : MonoBehaviour {

	// Use this for initialization
	void Start () {
		Debug.Log ("Hello, bro!");
	}

	// Update is called once per frame
	void Update () {

	}
}
```

```
Пространство имен
Пространство имен

Наш класс (наследник базового MonoBehaviour){

	//Описание события
		функция события(){
		Вывод в консоль: Hello, bro!
	}

	//Описание события
	функция события(){

	}
}
```

Именно этим и отличается программирование на юньке от традиционного - событиями. У нас есть некоторые события, которые выполяются в определенном порядке. Этакие полочки, с которых можно что-то взять, только в свой черед.

Давайте же познакомимся с этими событиями и их порядком.

1. Editor - События в редакторе (наша основная рабочая область)

Reset: Reset вызывается как только скрипт присоединили к GameObject (Любой объект).

2. First Scene Load – Эти функции вызываются один раз для каждого объекта в момент старта сцены (уровня).

Awake: Эта функция всегда вызывается раньше функции Start, сразу после инстантиации префаба. (Если GameObject не активен на момент старта, Awake не вызывается до тех пор, пока он не станет активным, или пока ни одна функция из скриптов, прикрепленных к объекту, не будет вызвана.)

OnEnable: (Вызывается, только если объект активен): Вызывается сразу после enable (включения) объекта. Это происходит как только создан экземпляр MonoBehaviour, или когда загружена сцена, или GameObject с компонентом скрипт инстантиируется.

Заметка: эти функции полезны, когда у нас в игре что-то инстантиируется. На ходу, так сказать.

3. Before the first frame update – Вызывается до обновления первого кадра

Start: вызывается до первого обновления кадра, но только если экземпляр скрипта включен.Для всех объектов сцены Start будет вызвана до Update. Start не может быть вызвана во время создания экземпляра объекта (во время геймплея).

4. In between frames

OnApplicationPause: Вызывается в конце кадра, когда обнаружена пауза, эффективно между нормальным обновлением кадров. Один дополнительный кадр будет предоставлен в момент вызова OnApplicationPause, чтобы позволить игре отобразить состояние паузы.

5. Update Order

Большинство всех игровых ситуаций происходит в этих функциях. Самая часто используемая функция это Update, но есть и другие.

FixedUpdate: вызывается чаще, чем Update. Все физические расчеты и остальные Updateы выполняется сразу после этой функции. Вычисляя что-то динамическое (например движение объектов) в FixedUpdate, не нужно умножать это на Time.deltaTime. Это потому, что FixedUpdate вызывается по таймеру, зависящему от частоты кадров.

Update: Вызывается один раз за кадр. Это одна из самых основных функций, в которых мы будем работать.

LateUpdate: вызывается один раз за кадр, после завершения Update. Все вычисления производимые в Update, будут завершены в момент начала LateUpdate. Пример использования LateUpdate это камера, которая следует за игроком (от третьего лица). Все движение игрока можно сделать в Update, а следование камеры в LateUpdate. Это позволит не беспокоиться о том, что игрок, возможно, не до конца завершил движение.

6. Rendering

OnGUI: Вызываются много раз за кадр в ответ на события графического интерфейса GUI. Графический интерфейс GUI и GUILayout можно использовать только через эту функцию.
OnDrawGizmos Эта функция позволяет отображать Гизмо (визуальные обозначения).

7. Coroutines

Обычно сопрограммы запускаются после того, как Update возвратиться.
yield Сопрограмма продолжится после того, как все функции Update будут вызваны в следующем кадре.
yield WaitForSeconds Продолжится после указанного времени задержки
yield WaitForFixedUpdate Продолжится после вызова FixedUpdate во всех скриптах
yield WWW Продолжится после завершения загрузки WWW.

8. When the Object is Destroyed

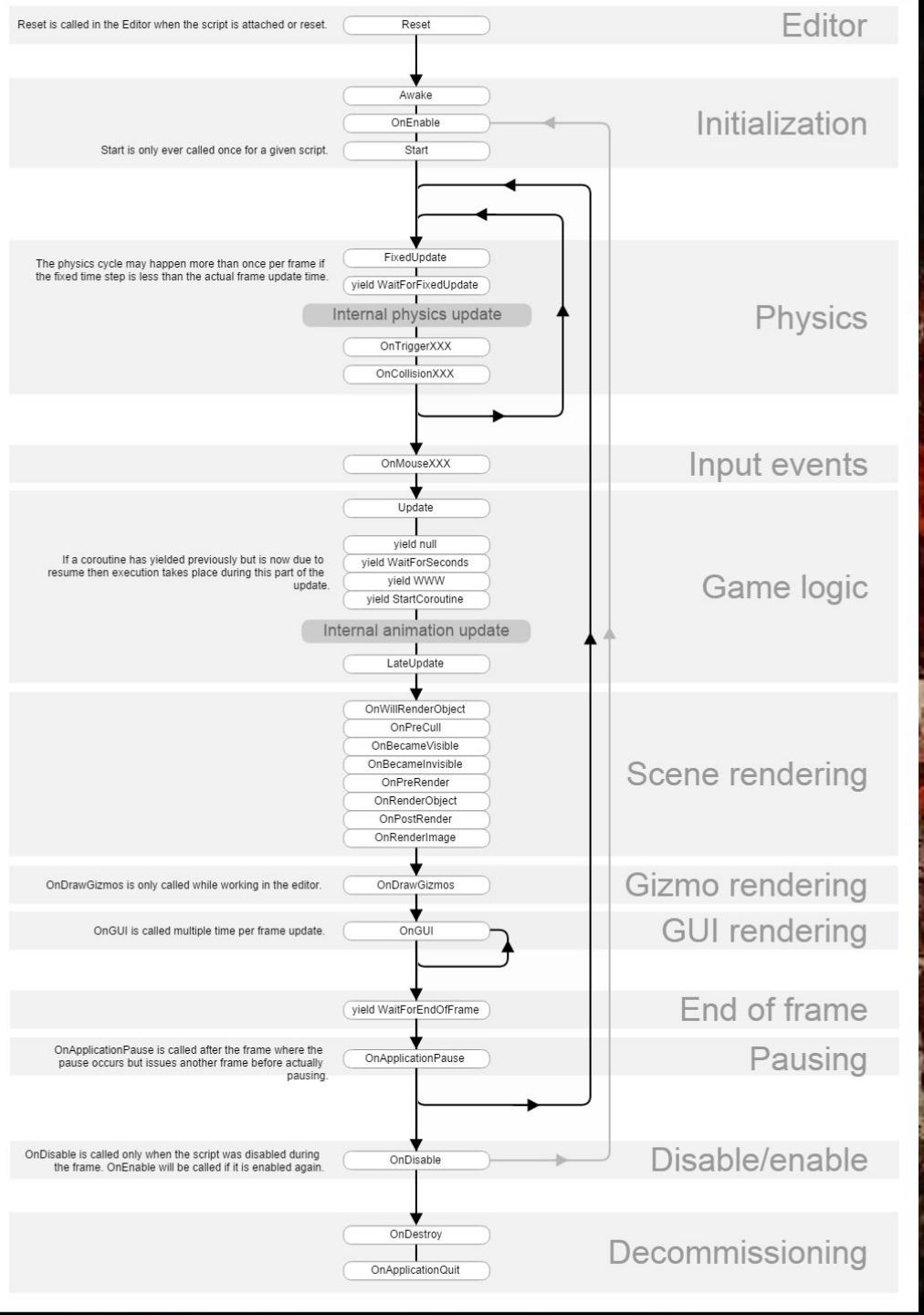
OnDestroy: Вызывается после всех Updateов последнего кадра, в котором существовал объект.

9. When Quitting

Все нижеследующие функции вызываются для всех активных объектов

OnApplicationQuit: Вызывается у всех активных объектов перед выходом из приложения (если в редакторе – то перед нажатием кнопки стоп).

OnDisable: Вызывается при деактивации.

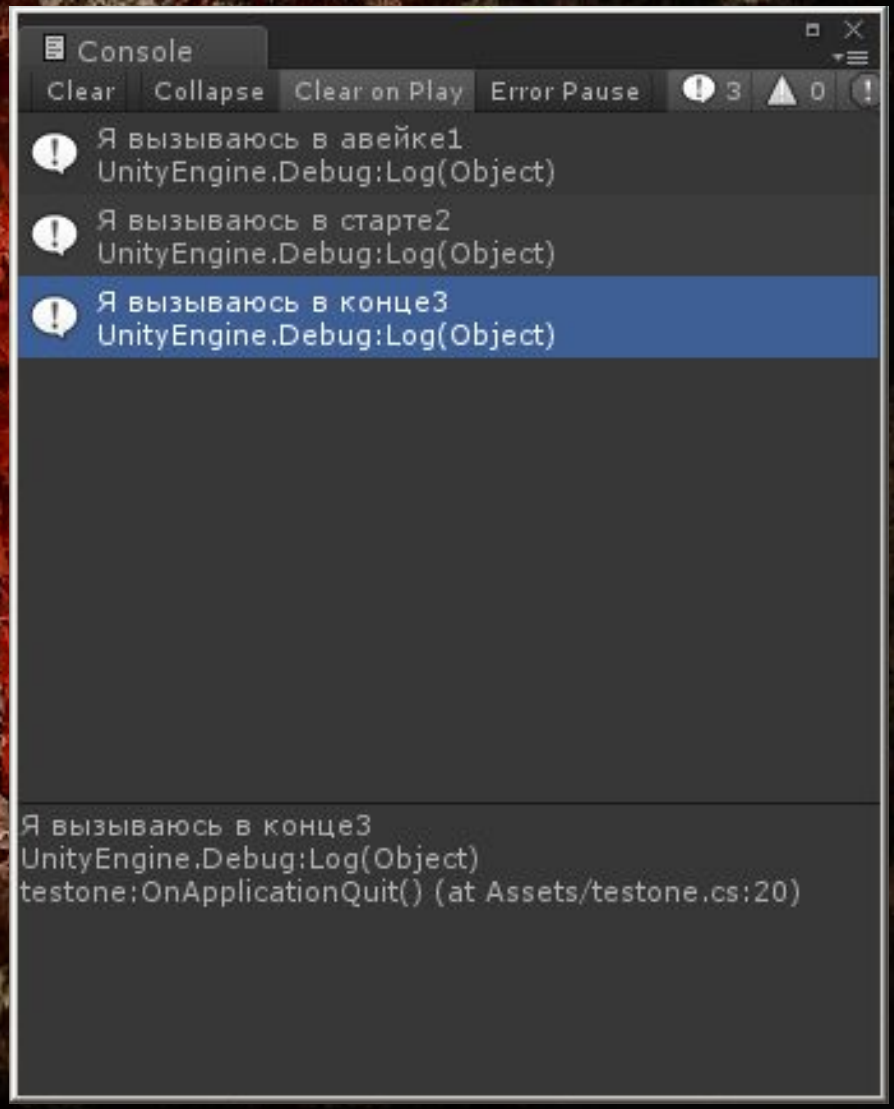

Editor
Reset is called in the Editor when the script is attached or reset.
Reset
Initialization
Awake
OnEnable
Start is only ever called once for a given script.
Start
Physics
The physics cycle may happen more than once per frame if the fixed time step is less than the actual frame update time.
FixedUpdate
yield WaitForFixedUpdate
Internal physics update
OnTriggerXXX
OnCollisionXXX
Input events
OnMouseXXX
Game logic
Update
If a coroutine has yielded previously but is now due to resume then execution takes place during this part of the update.
yield null
yield WaitForSeconds
yield WWW
yield StartCoroutine
Internal animation update
LateUpdate
Scene rendering
OnWillRenderObject
OnPreCull
OnBecameVisible
OnBecameInvisible
OnPreRender
OnRenderObject
OnPostRender
OnRenderImage
Gizmo rendering
OnDrawGizmos is only called while working in the editor.
OnDrawGizmos
GUI rendering
OnGUI is called multiple time per frame update.
OnGUI
End of frame
yield WaitForEndOfFrame
Pausing
OnApplicationPause is called after the frame where the pause occurs but issues another frame before actually pausing.
OnApplicationPause
Disable/enable
OnDisable is called only when the script was disabled during the frame. OnEnable will be called if it is enabled again.
OnDisable
Decommissioning
OnDestroy
OnApplicationQuit

Мы познакомились с языком Си шарп и его некоторыми особенностями, а так же с событиями и их порядками в юнити. Для закрепления материала, давайте взглянем на простой код написанный на юнити.

testone.cs

testone ▸ No selection

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class testone : MonoBehaviour {
5
6      int first=1;
7      int second=2;
8      int last=3;
9
10     void Awake(){
11         Debug.Log ("Я вызываюсь в авейке"+first);
12     }
13     void Start () {
14         Debug.Log ("Я вызываюсь в старте"+second);
15     }
16     void Update () {
17         //эту не используем, т.к. он она будет вечно что-то выводить
18     }
19     void OnApplicationQuit(){
20         Debug.Log("Я вызываюсь в конце"+last);
21     }
22 }
```



- Написать программу, которая бы выводила какие-то сообщения в разных событиях.
- Пообъявлять переменные

http://professorweb.ru/
http://mycsharp.ru/
https://ru.wikipedia.org
Моя голова и справка по юнити

Вы хотите еще больше качественных уроков?
Вы хотите все это абсолютно бесплатно?
Тогда вы можете помочь этому быть, подкинув копейку-другую в мой PayPal или WebMoney кошелек.

PayPal:
deatrocker@gmail.com

WebMoney:
WMZ: Z320863401836
WMR: R203258167795

Спасибо!