СКРИПТИНГ

основы C#

Редактор кода

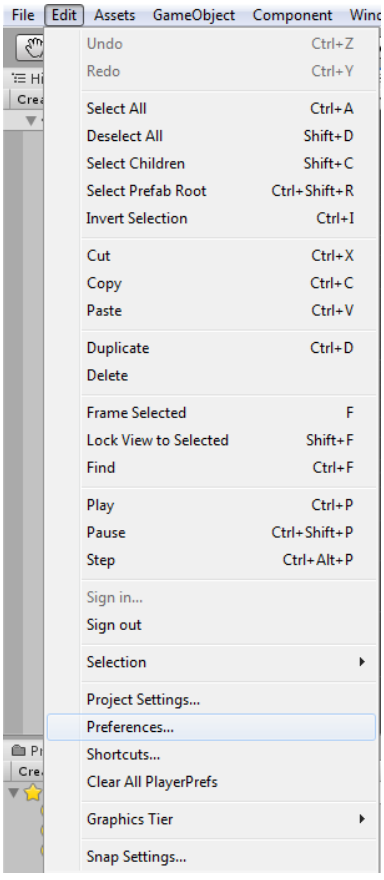
Integrated Development Environment(IDE)
Интегрированная Среда Разработки

Unity, при установке, обычно инсталлирует Visual Studio (VS).

Если VS не установлена, возможна отдельная установка и использование следующих IDE:

Visual Studio, Visual Studio Code, MonoDevelop, Jetbrains Rider.

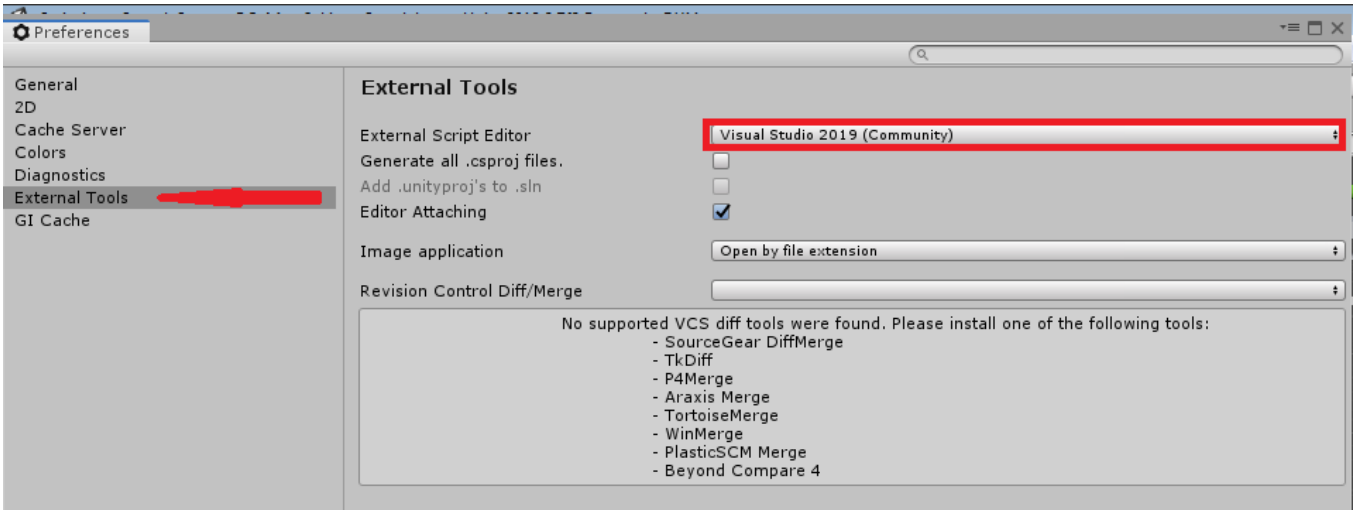
Интеграцию среды разработки (IDE) можно посмотреть или настроить вызовом пункта меню: Edit → Preferences, а затем выбором панели External Tools.



СКРИПТИНГ

основы C#

Выбор и настройка IDE

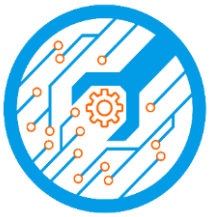


Понятие – скрипт

Скрипт — это нестандартный компонент, написанный программистом.

Unity поддерживает написание скриптов на языке C# (си-шарп).

Ранее была поддержка языка, похожего на Javascript и языка Boo, но, позднее, были удалены как непопулярные



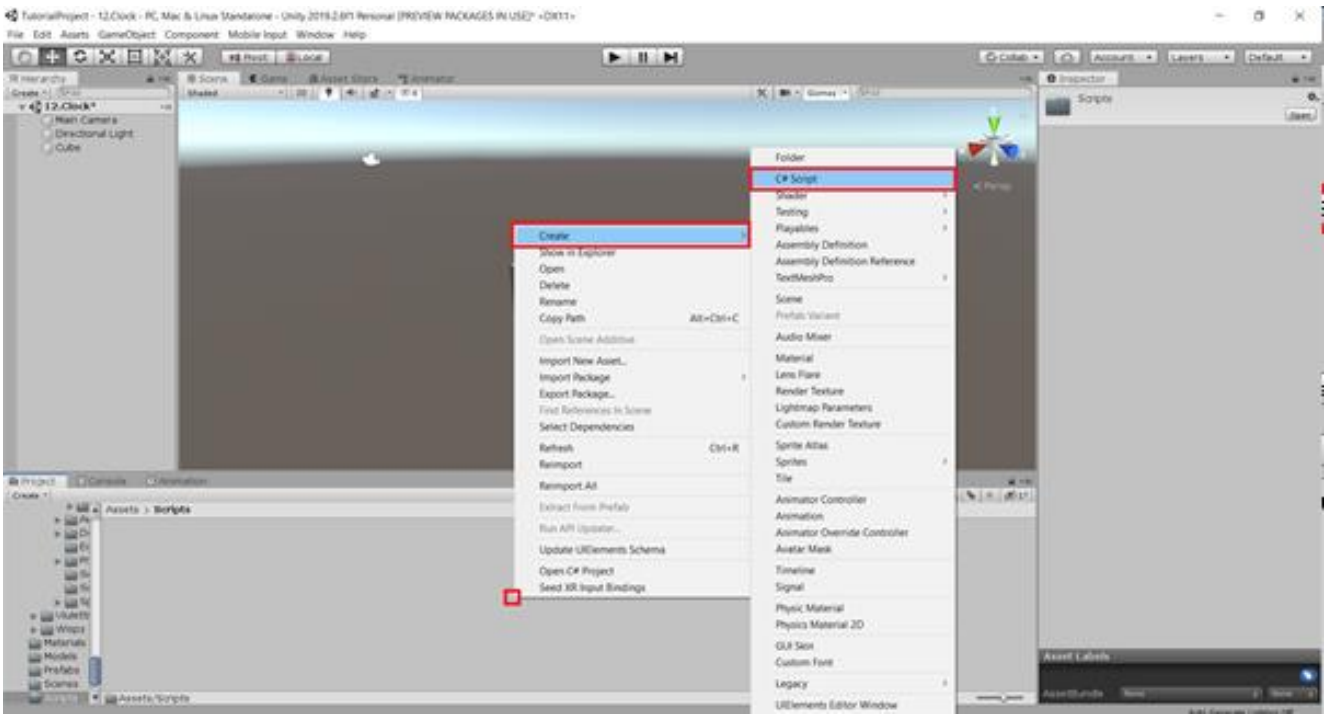
В коде скриптов на C# можно использовать как библиотеки Unity, так и стандартные (.NET) из установленного комплекта.

СКРИПТИНГ

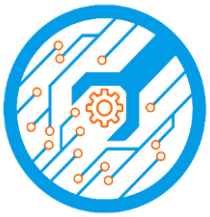
основы C#

Создание скрипта

В окне Assets нажатием правой кнопки мыши выбираем: Create → C# Script



Будет создан скрипт с именем NewBehaviourScript

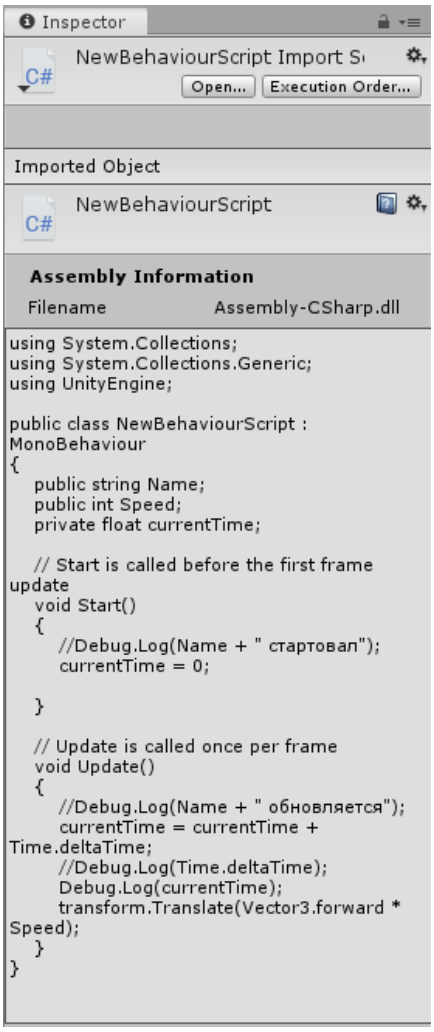


Желательно задать скрипту имя, соответствующее исполняемым функциям.

СКРИПТИНГ
основы C#

Вид стандартного скрипта

При этом в окне Inspector можно будет увидеть стандартный вид скрипта:



Из чего он состоит???

```
// Подключаемые библиотеки
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class NewBehaviourScript :
MonoBehaviour
{
   // Start is called before the first frame
update
    void Start()
    {
        // Выполняется при запуске
    }

    // Update is called once per frame
    void Update()
    {
        // Выполняется каждый кадр игры
    }
}
```

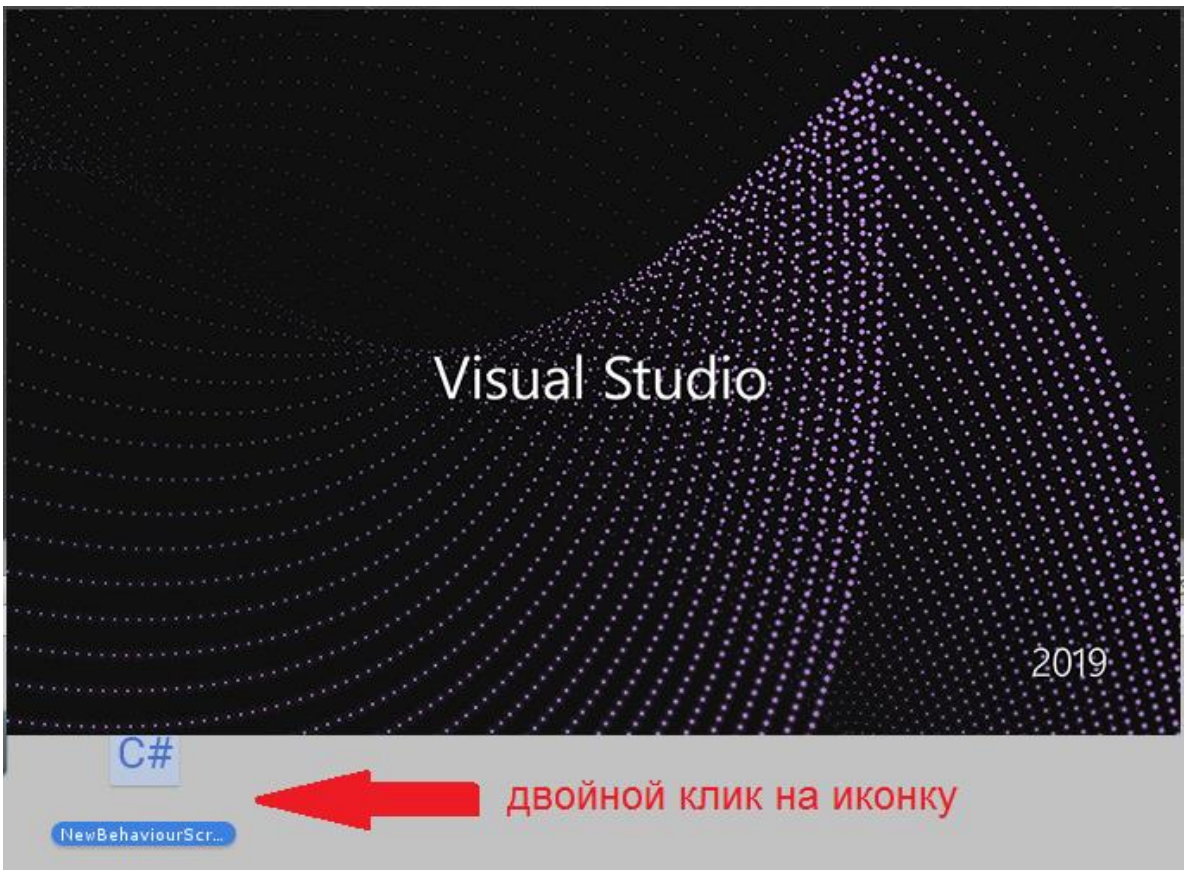
// - это комментарии. Они есть в коде программы, но не компилируются.
Об этом – чуть позже.

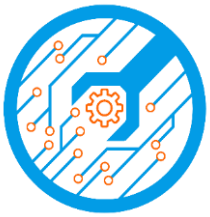
СКРИПТИНГ

основы C#

Редактор скрипта

При двойном щелчке на иконку скрипта в окне Assets, начнет открываться настроенный внешний редактор (см. стр. 2):





В нашем случае это Visual Studio 2019, загрузка которого может занять определенное время.

СКРИПТИНГ
основы C#

Редактирование скрипта

После внесения изменений в скрипт, его необходимо сохранить, а затем перейти назад в редактор Unity.

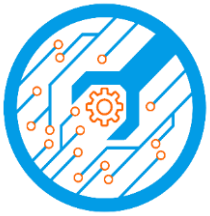
Unity начнет **компиляцию** скрипта.

Компиляция — это процесс преобразования набора команд, написанных на C# в машинный код.

Компилятор проверит код отредактированного скрипта на наличие ошибок и, при их отсутствии, откомпилирует его. В случае ошибок в скрипте компилятор выведет их список.

Список ошибок необходимо проанализировать и исправить во внешнем редакторе.

Исправление ошибок в тексте скрипта и его работе и есть основная задача программиста.



Работу скрипта можно увидеть только тогда, когда он будет применен к игровому объекту.

СКРИПТИНГ

основы C#

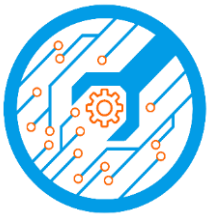
Первый скрипт

Код, который должен вызываться единожды, при старте игры, находится внутри тела функции **Start()**.

Код, который вызывается каждый кадр находится внутри тела функции **Update()**.

Допишем в тело функций следующее:

```
public class NewBehaviourScript : MonoBehaviour
{
    void Start()
    {
        Debug.Log("Я стартовал");
    }

void Update()
    {
        Debug.Log(“Я обновляюсь");
    }
}
```



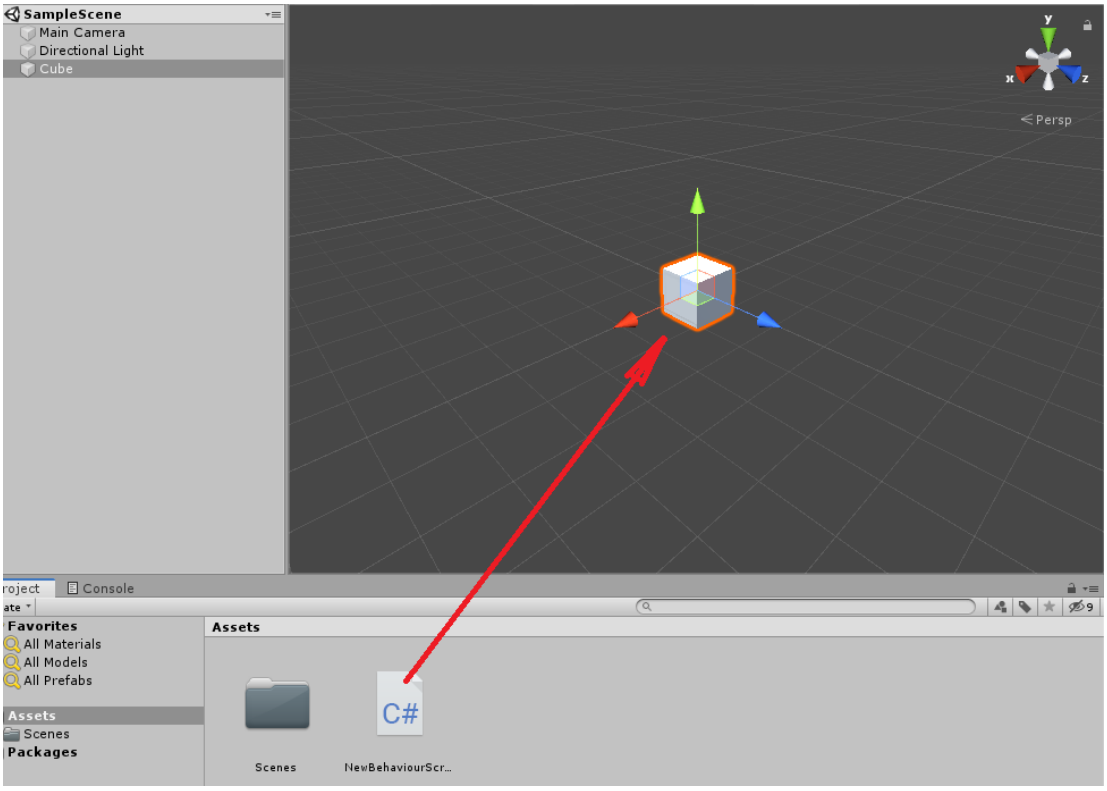
Сохраняем написанный код во внешнем редакторе и переходим в Unity.

СКРИПТИНГ

основы C#

Запуск скрипта

Для тестирования скрипта создаем любой объект, например Cube и применяем к нему скрипт – перетаскиванием.



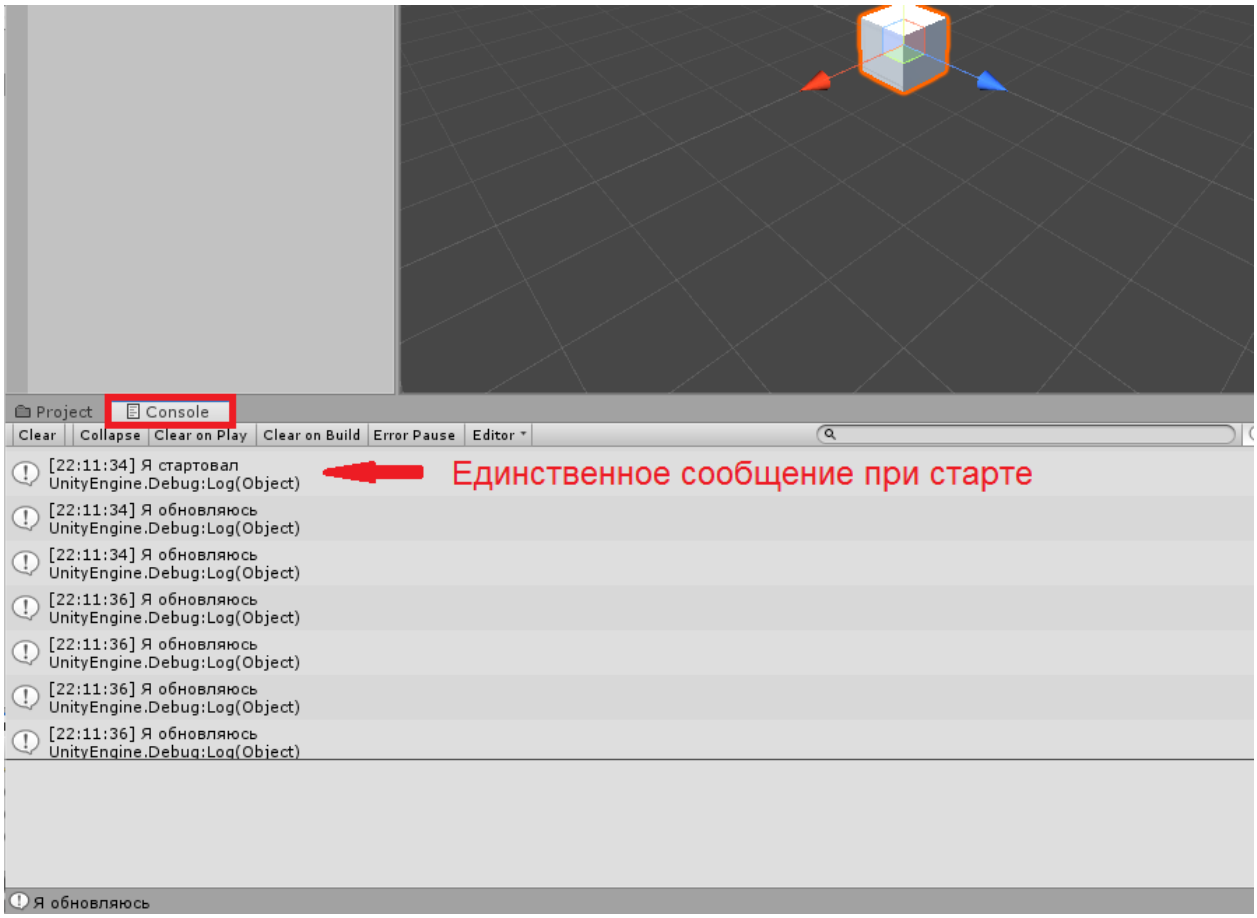
Запускаем игру. Результат можно увидеть в консоли.

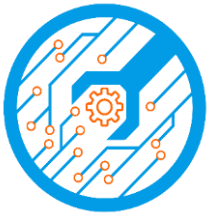
СКРИПТИНГ

основы C#

Результат работы скрипта

Как мы видим в консоли появились сообщения с написанным текстом: при старте и последующих обновлениях.





Это – отладочные сообщения, с помощью которых можно рассмотреть некоторые приемы программирования.

СКРИПТИНГ

основы C#

Переменные в скриптах

Введем переменную **Name**, которую запишем как:

```
public string Name;
```

- public – означает, что переменной можно управлять в среде Unity, в окне Inspector;
- string – переменная *строковая*.

Место объявления переменной и фрагмент нашего немного измененного скрипта – ниже:

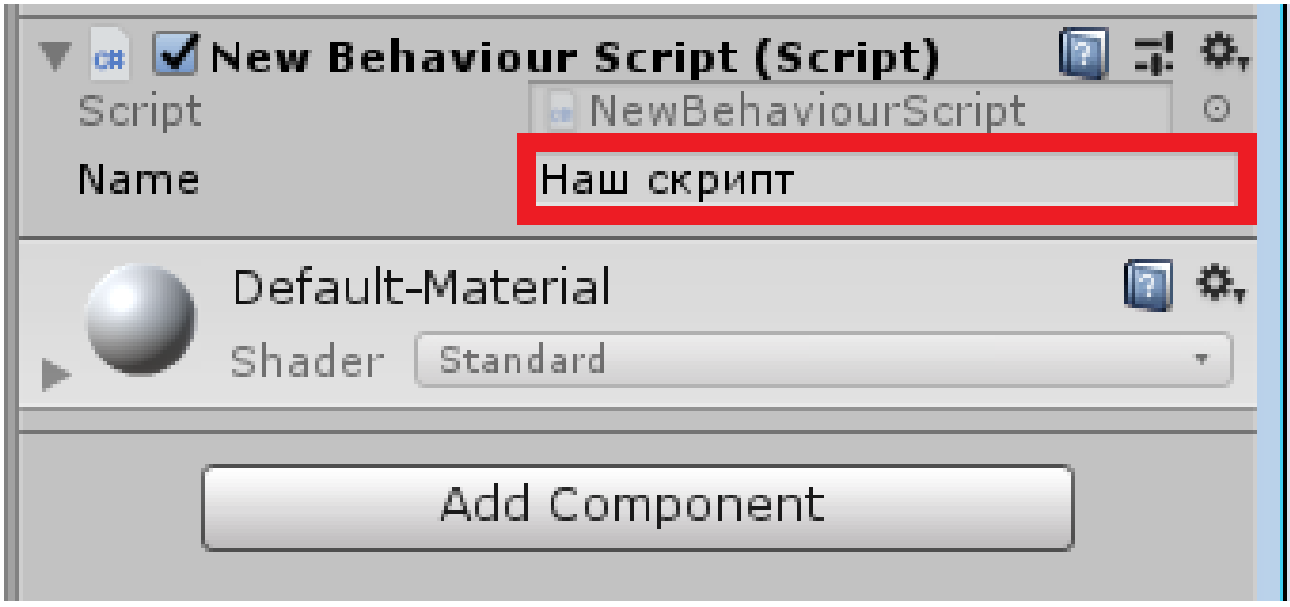
```
public class NewBehaviourScript : MonoBehaviour
{
    public string Name;

    void Start()
    {
        Debug.Log(Name + " стартовал");
    }

    void Update()
    {
        Debug.Log(Name + " обновляется");
    }
}
```

СКРИПТИНГ

основы C#

Управление переменной

После перехода в среду Unity, выбираем объект Cube, к которому применён скрипт и находим в Inspector следующее:



Изменяем в Inspector-е значение Name на «Наш скрипт». Запускаем!!!

В консоли увидим изменения. Вместо «Я стартовал» и «Я обновляюсь», будет:

«Наш скрипт стартовал»

и

«Наш скрипт обновляется»

СКРИПТИНГ

основы C#

Управление движением Cube

Введем целочисленную переменную, отвечающую за скорость:

```
public int Speed;
```

А также добавим строку в тело void Update():

```
transform.Translate(Vector3.forward * Speed);
```

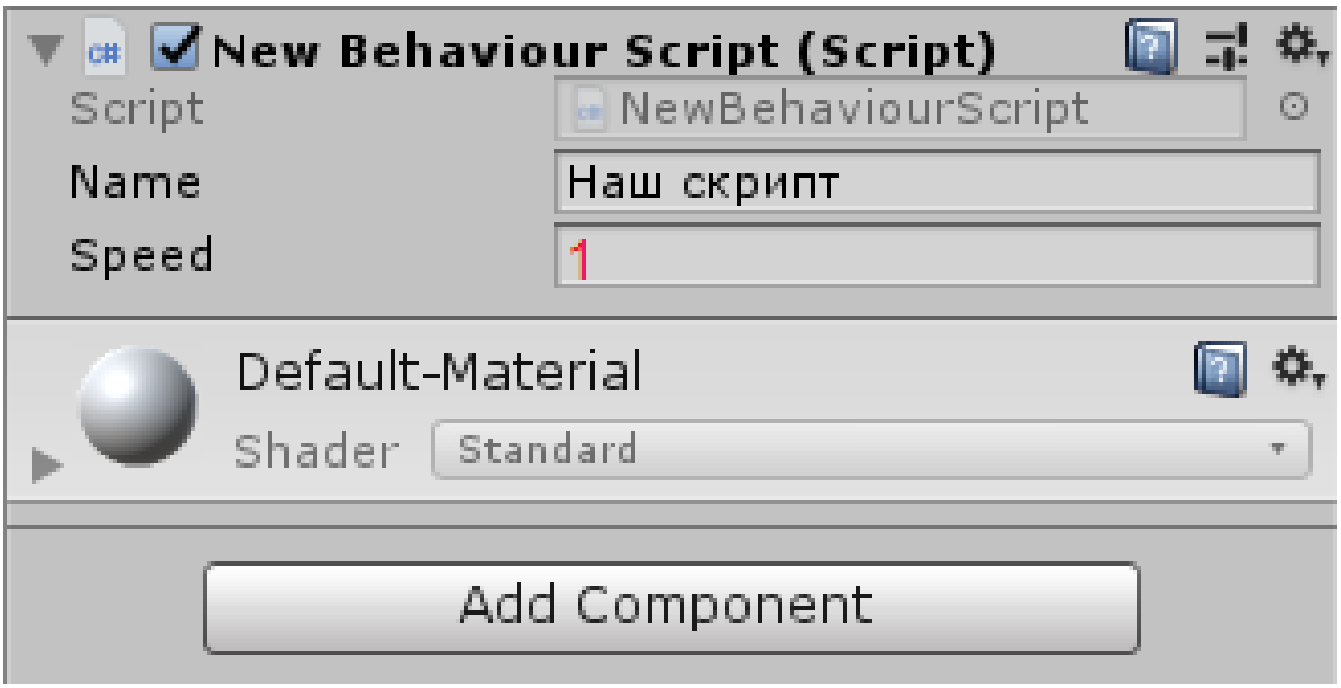
```
public class NewBehaviourScript : MonoBehaviour
{
    public string Name;
    public int Speed;


    void Start()
    {
        Debug.Log(Name + " стартовал");
    }


    void Update()
    {
        Debug.Log(Name + " обновляется");

        transform.Translate(Vector3.forward * Speed);
    }
}
```

СКРИПТИНГ

основы C#

Управление движением Cube

После перехода в среду Unity и компиляции, переменная появится в Inspector-е:



Изменим ее значение на «1». Запустим игру и наш Cube улетит вдаль.

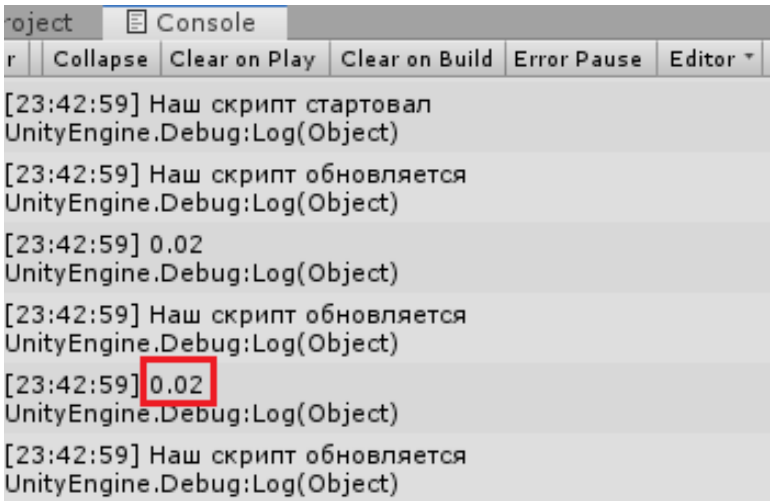
СКРИПТИНГ
основы C#

Привязка ко времени

Обновление кадров в движке Unity происходит примерно 60 раз в секунду. Точное количество определяется производительностью компьютера и загрузкой процессора на момент выполнения скрипта. Данное значение можно определять и регулировать с помощью метода deltaTime. Допишем в тело Update() следующее:

```
void Update()
    {
        Debug.Log(Time.deltaTime);
        . . . . . .
    }
```

Но не удаляем «полет» нашего Cube. Запускаем



На конкретном компьютере обновление происходит каждые 0,02 секунды или:

1/0,02 = 50 раз в секунду

СКРИПТИНГ
основы C#

Счетчик времени

Для того, чтобы сделать счетчик времени, нам необходимо добавить переменную с плавающей (float) точкой (десятичная дробь):

```
public class NewBehaviourScript : MonoBehaviour
{
    public string Name;
    public int Speed;
    private float currentTime;

void Start()
    {
        Debug.Log(Name + " стартовал");
        currentTime = 0;
    . . . . . .
```

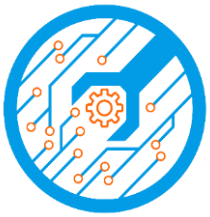
- private – означает, что переменной нельзя управлять в Inspector-е. Она там не появится.

СКРИПТИНГ

основы C#

Счетчик времени (вид скрипта)

```
public class NewBehaviourScript : MonoBehaviour
{
  public string Name;
  public int Speed;
  private float currentTime;

  void Start()
 {
   currentTime = 0;
 }

  void Update()
 {
   currentTime = currentTime + Time.deltaTime;
   Debug.Log(currentTime);// или Time.time
   transform.Translate(Vector3.forward * Speed);
 }
}
```



Запускаем и наблюдаем за временем в левом нижнем углу.

СКРИПТИНГ

основы C#

ТИПЫ ДАННЫХ И КЛАССЫ

Использованные типы данных:

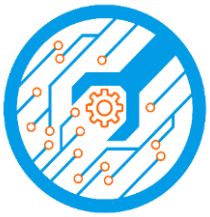
- string - строка, например, “привет” (в Inspector-е пишется без кавычек);
- int - целое число, например, 105;
- float – десятичная дробь, например, 18.995.

Данные и методы (функции) для работы с ними представляют собой **КЛАСС** (Class).

Возможно использование библиотечных классов.

Уже использованный нами библиотечный класс – Vector3 управляет перемещением объекта в трех координатах пространства (методы – left, right, up, down).

Также нами был использован класс Time (с его методом deltaTime). deltaTime – время, в долях секунды, которое потребовалось для отрисовки текущего кадра.



Классы может создавать и сам программист.

СКРИПТИНГ

основы C#

ЗАКЛЮЧЕНИЕ

Советую посетить:
https://3dgame-creator.ru/unity-5-metod-transform-translate

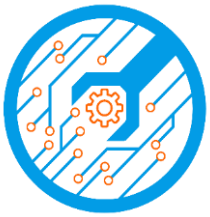
Используем еще такой класс, как Input, отвечающий за реакцию скрипта на нажатие клавиш клавиатуры, клики мышкой и т.д. Методы:

- GetAxis("Horizontal") – перемещение в горизонтальной плоскости с помощью стрелок или клавиш **A** и **D**;
- GetAxis("Vertical") – перемещение в вертикальной плоскости с помощью стрелок или клавиш **W** и **S**;

Метод Translate класса transform – "расшифровывает" способ перемещения объекта в игровом пространстве движка Unity.

Рассмотрим пример:

```
float LeftRight = Speed * Time.deltaTime * Input.GetAxis("Horizontal");
float UpDown = Speed * Time.deltaTime * Input.GetAxis("Vertical");
transform.Translate(Vector3.right * LeftRight);
transform.Translate(Vector3.up * UpDown);
```



Если этот код поместить в тело Update(), то станет возможным двигать наш Cube с помощью стрелок или WASD.