





# **C# в Unity. Жизненный цикл объектов, классы**

Почему Update «дорогой», классы, наследование, бонус



# Обычный C# vs C# в Unity

## Unity < 2018

- Mono с .NET 3.5 (старая)
- Отсутствовали async/await, LINQ расширенный, Span
- Пространства имён урезаны: System.Drawing, System.Web, System.Reflection, System.Threading



# Обычный C# vs C# в Unity

## Unity 2021-6000

- .NET Standard 2.1 (Unity 6 ближе к .NET 8)
- Доступны почти все базовые классы (System, Collections, Linq, IO и т. д.).
- Недоступно всё, что связано с UI, WinForms, WPF, ASP.NET, ADO.NET, System.Drawing, System.Web
- Поток есть (Thread, Task), но работать напрямую с Unity API в них нельзя



# Namespace, которых **нет** в Unity C#

**Интерфейсы и графика Windows:** System.Windows.Forms, System.Windows (WPF), System.Drawing (GDI+)

**Веб и серверные технологии:** System.Web, System.Web.UI, System.Web.Services

**Базы данных:** System.Data, System.Data.SqlClient, System.Data.OleDb

**Потоки:** System.Threading.ThreadPool (ограничено), System.Threading.Timer (нестабильно)

**Технологии:** System.EnterpriseServices, System.Transactions, System.Messaging (MSMQ)

**Сеть:** System.Net.Sockets (ограничено)



## Особенности Unity C#

- C# интегрирован в движок, код работает через Unity API (MonoBehaviour, GameObject, Transform и т.д.)
- Нет главной точки входа, есть сцена и MonoBehaviour методы (Awake, Start, Update...)
- Многие стандартные библиотеки заменены на аналоги Unity (Vector3, Quaternion, Color, Debug.Log)
- Из потоков нельзя работать напрямую с Unity API (вызовы GameObject, Transform, MonoBehaviour должны быть из главного потока)
- Свой сборщик мусора



# Классы

```
public class Car
{
    public string model;
    public float speed;

    public void Drive()
    {
        Debug.Log(model + " едет со скоростью " + speed);
    }
}
```



# Классы MonoBehaviour

```
using UnityEngine;

public class Player : MonoBehaviour
{
    public float speed = 5f;

    void Update()
    {
        transform.Translate(Vector3.forward * speed * Time.deltaTime);
    }
}

public class GameController : MonoBehaviour
{
    void Start()
    {
        GameObject obj = new GameObject("PlayerObject");
        Player player = obj.AddComponent<Player>();
        player.speed = 10f;
    }
}
```





# Классы MonoBehaviour

```
using UnityEngine;
```

```
public class Enemy : MonoBehaviour
{
    public float health = 100f;
}
```

```
public class Spawner : MonoBehaviour
{
    public GameObject enemyPrefab;

    void Start()
    {
        GameObject enemyObj = Instantiate(enemyPrefab, Vector3.zero, Quaternion.identity);
        Enemy enemy = enemyObj.GetComponent<Enemy>();
        enemy.health = 200f;
    }
}
```



## Классы MonoBehaviour

**AddComponent<T>()** → навешиваем новый скрипт/компонент на объект

**Instantiate(prefab)** → создаём копию уже готового объекта с его настройками



# Инкапсуляция

```
using UnityEngine;

public class Player : MonoBehaviour
{
    [SerializeField] private int health = 100;

    public int Health
    {
        get { return health; }
        private set { health = Mathf.Clamp(value, 0, 100); }
    }

    public void TakeDamage(int damage)
    {
        Health -= damage;
        Debug.Log("Player получил урон. Текущее здоровье: " + Health);
    }

    public void Heal(int amount)
    {
        Health += amount;
        Debug.Log("Player восстановил здоровье. Текущее здоровье: " + Health);
    }
}
```



# Инкапсуляция

```
using UnityEngine;

public class GameController : MonoBehaviour
{
    void Start()
    {
        GameObject playerObj = new GameObject("PlayerObject");
        Player player = playerObj.AddComponent<Player>();

        player.TakeDamage(30);
        player.Heal(20);
        player.TakeDamage(100);
    }
}
```



## Наследование

```
public class Character : MonoBehaviour
{
    public float health;
}

public class Enemy : Character
{
    public void Attack() { Debug.Log("Enemy атакует!"); }
}

public class Player : Character
{
    public void Heal() { Debug.Log("Player лечится!"); }
}
```



# Полиморфизм

```
public class Character : MonoBehaviour
{
    public virtual void Die()
    {
        Debug.Log("Персонаж умер");
    }
}

public class Enemy : Character
{
    public override void Die()
    {
        Debug.Log("Враг исчезает с криком");
    }
}
```



# Полиморфизм

```
public class Character
{
    public virtual int health = 100;
}
```

НЕЛЬЗЯ

---

```
public class Character
{
    public virtual int Health { get; set; } = 100;
}

public class Boss : Character
{
    public override int Health { get; set; } = 1000;
}
```

МОЖНО



# Интерфейсы

```
public interface IDamageable
{
    void TakeDamage(int amount);
}

public class Enemy : MonoBehaviour, IDamageable
{
    private int health = 100;

    public void TakeDamage(int amount)
    {
        health -= amount;
        if (health <= 0) Destroy(gameObject);
    }
}

public class Player : MonoBehaviour
{
    void Attack(IDamageable target)
    {
        target.TakeDamage(25);
    }
}
```





## Снова про MonoBehaviour

- **Awake()** — вызывается один раз при создании объекта (до старта сцены)
- **Start()** — вызывается при первом кадре, когда объект активен
- **Update()** — вызывается каждый кадр, подходит для логики 🤖
- **FixedUpdate()** — вызывается с фиксированным шагом времени (для физики)
- **LateUpdate()** — вызывается после всех Update (для работы с камерой и анимациями)
- **OnDestroy()** — вызывается при уничтожении объекта

Подробная схема в файле



## Порядок MonoBehaviour

Не гарантирован!

НО...

# Порядок MonoBehaviour

## Script Execution Order

Add scripts to the custom order and drag them to reorder.

Scripts in the custom order can execute before or after the default time and are executed from top to bottom. All other scripts execute at the default time in the order they are loaded.

(Changing the order of a script may modify the meta data for more than one script.)

IngameDebugConsole.DebugLogManager	-9869	-
UnityEngine.EventSystems.EventSystem	-1000	-
TMPro.TextContainer	-110	-
TMPro.TextMeshPro	-105	-
TMPro.TextMeshProUGUI	-100	-
Default Time		
UnityEngine.UI.ToggleGroup	10	-
		+ ▼

Revert Apply



## Особенности MonoBehaviour

- Нельзя создать через new
- Работает только если прикреплен к GameObject
- Поддерживает корутины
- Есть специальные методы: OnCollisionEnter, OnTriggerEnter, OnMouseDown и т.д.
- Можно включать/выключать: `this.enabled = false;`



## Полезные атрибуты

- `RequireComponent`
- `SerializeField`
- `HideInInspector`
- `Header`
- `TextArea`
- `Tooltip`

Подробный список в файле



## Почему Update «дорогой»

- Unity каждый кадр проверяет у каждого MonoBehaviour, есть ли у него метод Update
- Вызов идёт не напрямую, а через рефлексию
- Даже если метод пустой, Unity всё равно чекает и вызывает
- Если игра работает в 60 FPS → 60 раз в секунду на каждый скрипт:  
при 1000 объектов → 60 000 вызовов в секунду
- Внутри Update часто создают новые объекты (new, конкатенация строк, List.Add) → генерируется мусор, который потом убирает GC → это может внезапно «заморозить» игру



## Почему Update «дорогой»

Дорогой не потому, что сам метод тяжёлый, а потому что Unity вызывает его рефлексией на каждом кадре для всех активных скриптов



## Про deltaTime

`Time.deltaTime` — интервал времени (в секундах) между текущим и предыдущим кадром





## Про deltaTime

- `Time.deltaTime` — межкадровый интервал (изменяется с FPS)
- `Time.fixedDeltaTime` — фиксированный шаг физики (по умолчанию 0.02)
- `Time.unscaledDeltaTime` — время между кадрами игнорируя `Time.timeScale` (полезно для UI и пауз)



## Последний слайд

Что будет на практике:

- научитесь делать базовые классы и наследование (разные типы врагов)
- сделаете переопределение методов (virtual / override) атаки
- будете использовать генерацию объектов в сцене
- научитесь связывать классы с Unity объектами
- будете использовать интерфейсы и полиморфизм через реальный игровой пример

Будет удобный, обновляемый шаблон проекта с настройками и комментариями 🤖  
<https://github.com/mopsicus/unity-practice>



## Бонус. Про работу

- Github аккаунт
- Тестовые задания
- LinkedIn
- Личный блог/канал
- Резюме под вакансию
- Нетворкинг
- Публикации

Подробный список в файле



# README.md

<https://github.com/matiassingers/awesome-readme>



## Бонус. Телеграм-каналы

Рассылка: <https://gamedevdigest.com>

Подробный список в файле



# Что дальше

Архитектура, паттерны:

- компонентный подход
- singleton
- event bus
- scriptable object
- ecs
- fsm
- nav mesh
- поведенческие деревья
- solid
- etc