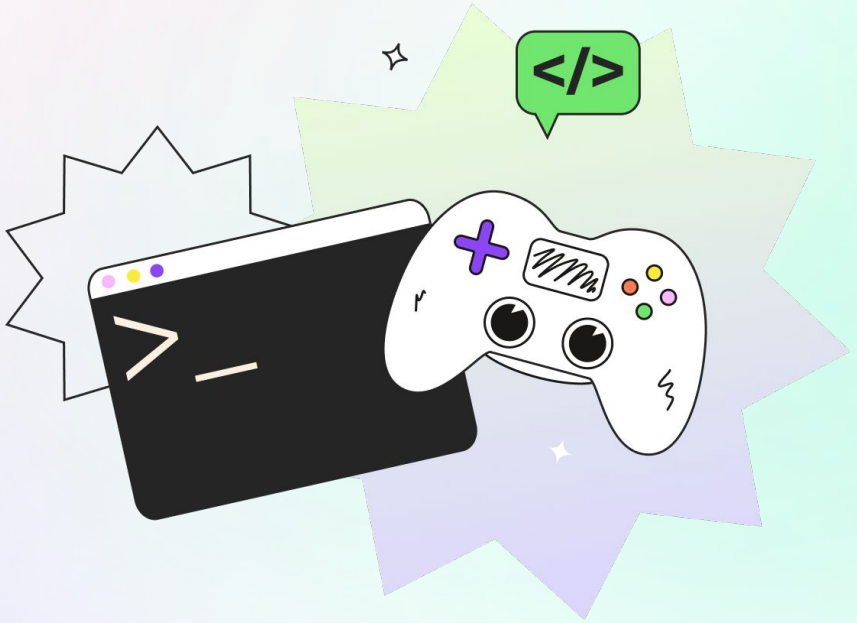
GeekSchool

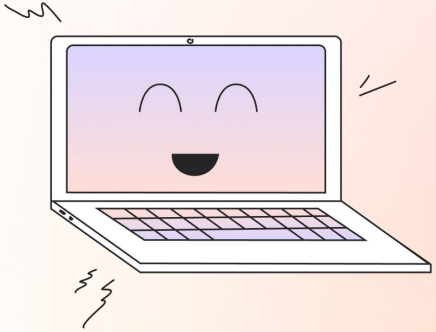


Разработка игр на Unity

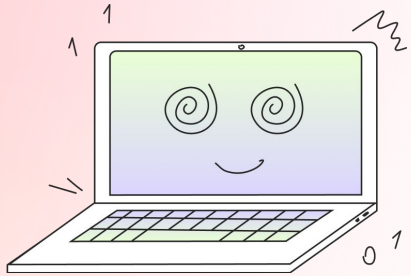
Урок 4
Создание игровой механики

План курса. Модуль 1 – Основы работы в Unity. Жанр «2D-платформер»

1. Знакомство и настройка программ
2. Игровые объекты
3. Основы программирования на языке C#
4. Создание игровой механики
5. Противники и анимация
6. Пользовательский интерфейс и монетки
7. Доработка игры



Ставь + в чат,
если хорошо видно и слышно



Какой программист ты сегодня? =)

1

2

3

4

5

6

Давайте вспомним предыдущий урок =)

Переходим на сайт с викториной по **ссылке**,
которую отправит преподаватель

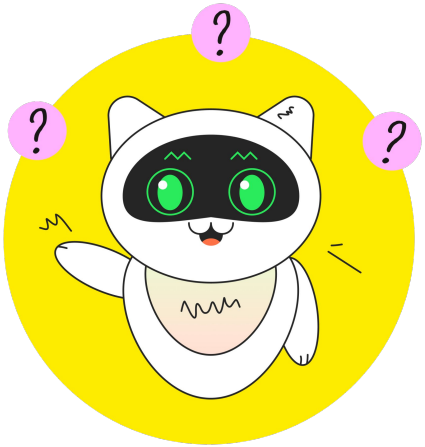
В прошлый раз мы начали изучать программирование на C#, разобрались с функциями, переменными и даже создали управление для персонажа. Сегодня еще больше углубимся в игровые механики!

Что будет на уроке сегодня

- Добавляем столкновение объектов
- Учим персонажа прыгать
- Создаём ловушки, которые наносят урон: лезвия, шипы

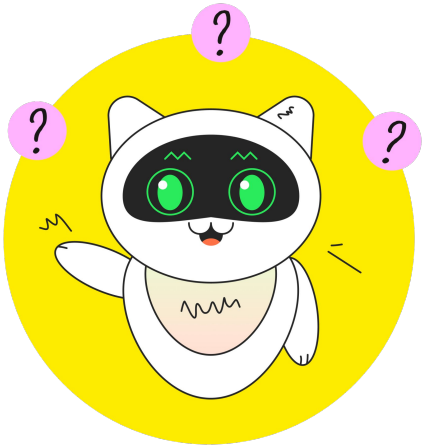
Вспомним некоторые термины

1. Что такое переменная?



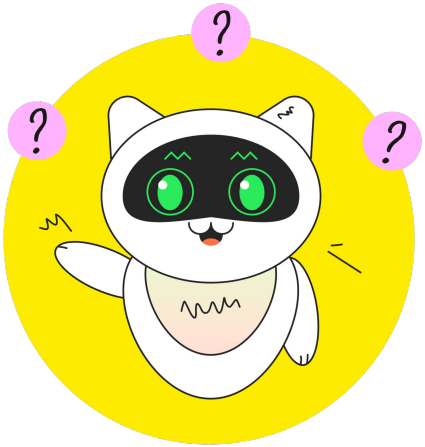
Вспомним некоторые термины

1. Что такое переменная?
2. Что такое функции? Событийная функция?



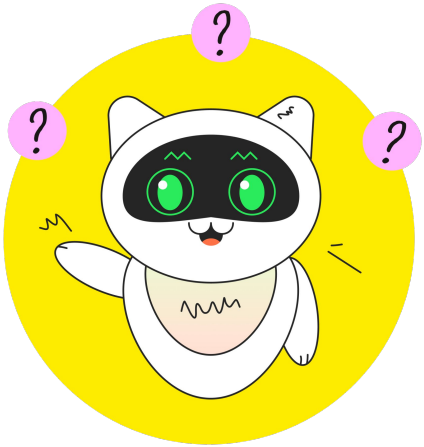
Вспомним некоторые термины

1. Что такое переменная?
2. Что такое функции? Событийная функция?
3. Что такое скрипт?



Вспомним некоторые термины

1. Что такое переменная?
2. Что такое функции? Событийная функция?
3. Что такое скрипт?
4. Какие есть правила создания скрипта и переменной?



Условия

Условия в жизни и условия в программировании очень похожи, но попробуем понять определение из программирования:

Условие – это логическое выражение, т.е. выражение, результатом которого является логическое значение true (истина) или false (ложь)

Примеры условий из жизни

1) Если идет дождь, то я возьму зонт
2) Если болит живот, то надо пойти к врачу
3) Если садится телефон, то поставить его на зарядку

Примеры условий из игры

1) Если у персонажа 0 здоровья, то закончить игру
2) Если у оружия 0 патронов, то включить анимацию перезарядки
3) Если нажат пробел, то выполнить прыжок

А какие другие примеры условий из игр и жизни ты помнишь?

Условные операторы — это группа символов (>,<, ==, ||, &&, <=,>=), которые помогают сравнивать значения внутри условий и, в зависимости от результата сравнения, выполнять конкретные действия

Условные операторы

Оператор	Значение	Пример	Пояснение
>	больше	health > 0	Если здоровье больше 0
>=	больше или равно	coins >= 100	Если монет ровно 100 или больше
<	меньше	health < 100	Если здоровье меньше 100
<=	меньше или равно	health <= 0	Если здоровье 0 или меньше 0
==	равно	health == 0	Если здоровье равно 0
!=	не равно	health != 100	Если здоровье не 100
||	или	health != 100 || health > 0	Если здоровье не 100 или если здоровье больше 0
&&	и то, и другое	health == 100 && name == “Dima”	Если здоровье равно 100 и имя Дима

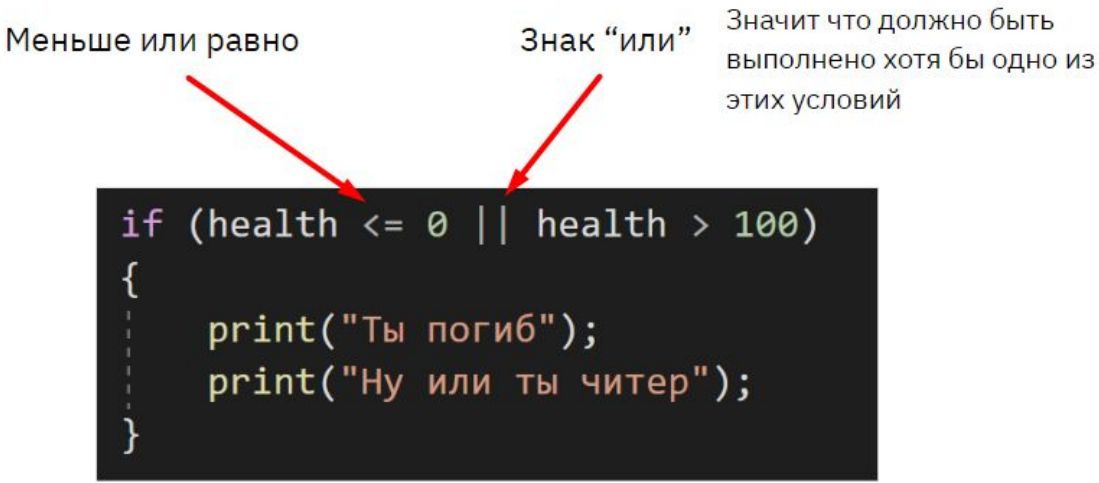
Содержание условия
(если здоровье меньше 0)

```
if (health < 0)
{
    print("У вас ноль здоровья");
}
```

Что будет, если условие будет верным?
(Пишется внутри фигурных скобок)

***не забудьте создать переменную health перед выполнением**

Символ || для обозначения или то, или другое условие



Символ && для обозначения выполнения сразу обоих условий

Значит что должно быть выполнено и то, и другое условие одновременно

Знак “и”

Знак равенства (используется только внутри условий)

```
if (health > 0 && name == "Ivan")
{
    print("Ты живой");
    print("А еще тебя зовут Иван");
}
```

Отслеживаем нажатие на кнопку

Конструкция **Input** (пример справа) отслеживает, нажали ли вы на определенную кнопку. И если нажали, то оно выдает **true**. Благодаря условию мы можем сделать так, что мы сможем всегда определять, нажата ли определенная кнопка.

Input помимо всего прочего переводится как “**ввод**, **ввести**”.

```
if (Input.GetKey(KeyCode.F))
{
    print("Вы нажали на кнопку F");
}
```

if (Input.GetKey(KeyCode.F))
{
print("Вы нажали на кнопку F");
}

Отслеживаем нажатие на кнопку

1) Input можно использовать с **условными операторами**

1) Input.GetKey – отслеживает **ЗАЖАТИЕ** на кнопку (длительное)

1) Input.GetKeyDown – отслеживает **НАЖАТИЕ** на кнопку (один раз)

```
if (Input.GetKey(KeyCode.F) || Input.GetKey(KeyCode.G))
{
    print("Вы нажали на кнопку F или на G");
}
```

```
if (Input.GetKey(KeyCode.Escape))
{
    print("Зажата кнопка Ecsape");
}
```

```
if (Input.GetKeyDown(KeyCode.Space))
{
    print("Нажали на пробел один раз");
}
```

А теперь вспомним скрипт с прошлого занятия. Там мы как раз таки благодаря условию отслеживали зажатие кнопки и, если она зажата, двигали игрока влево или вправо

```
if (Input.GetKey(KeyCode.A))
{
    transform.Translate(Vector2.left * speed * Time.deltaTime);
}
if (Input.GetKey(KeyCode.D))
{
    transform.Translate(Vector2.right * speed * Time.deltaTime);
}
```

Теперь вы знаете, что такое условия и как отслеживать нажатие кнопок в Unity! Это дверь в огромный мир созданиях любой механики, которую вы только придумаете

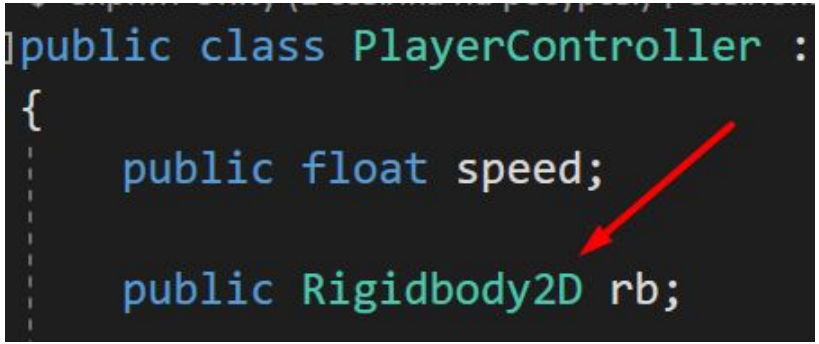
Прыжок персонажа

Теперь остановимся на том, как сделать прыжок. Функция **Translate двигает** объект по координатам, но для прыжка это не всегда самый лучший вариант. Существует очень популярная функция **AddForce**. Дословно она переводится как “добавить силы”. Она **толкает** объект в нужном направлении, используя физику.

Прыжок персонажа

Функция применяется не ко всему объекту, а к компоненту **Rigidbody**. То есть перед тем как ее применять, необходимо создать переменную, в которой будет этот компонент.

Ранее мы создавали переменные только с цифрами, буквами, но на самом деле можно практически с чем угодно



public Rigidbody2D rb;

Прыжок персонажа

Ранее мы умножали направление на переменную для того, чтобы прямо из Unity контролировать скорость. Здесь создадим еще одну переменную, чтобы **контролировать силу прыжка**.

Всегда старайтесь создавать переменные, которые влияют на механики. Так вам будет легче создавать разные комбинации из них

```
public class PlayerController :
{
    public float speed;

    public Rigidbody2D rb;

    public float jumpforce;
```

public float jumpforce;

Ну а чтобы у вас все заработало, надо написать следующий код внутри Update.

```
if (Input.GetKeyDown(KeyCode.Space))
{
    rb.AddForce(Vector2.up * jumpforce, ForceMode2D.Impulse);
}
```

Здесь, мы:
1) обращаемся к **переменной** rb
2) через точку применяем к ней **AddForce**
3) в скобках указываем, в каком **направлении** толкать.

Через запятую мы можем указать, что за вид толкания мы используем:
ForceMode2D.Force – давление (постепенное)
ForceMode2D.Impulse – резкий толчок (разовый)

if (Input.GetKeyDown(KeyCode.Space))
{
rb.AddForce(Vector2.up * jumpforce, ForceMode2D.Impulse);
}

Скрипт

Вот так выглядит наш скрипт **PlayerController** на данном этапе. Обратите внимание, что мы пока что используем только функцию Update.

Проверьте, чтобы ваш скрипт был точно такой же, как на скриншоте.

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Скрипт Unity (1 ссылка на ресурсы) | Ссылок: 0
public class PlayerController : MonoBehaviour
{
    public float speed;
    public Rigidbody2D rb;
    public float jumpforce;

    Сообщение Unity | Ссылок: 0
    void Update()
    {
        if (Input.GetKey(KeyCode.A))
        {
            transform.Translate(Vector2.left * speed * Time.deltaTime);
        }
        if (Input.GetKey(KeyCode.D))
        {
            transform.Translate(Vector2.right * speed * Time.deltaTime);
        }

        if (Input.GetKeyDown(KeyCode.Space))
        {
            rb.AddForce(Vector2.up * jumpforce, ForceMode2D.Impulse);
        }
    }
}
```

Прыжок персонажа

Последнее, что нужно сделать – сохранить изменения, открыть Unity и там проставить силу прыжка.

Поиграйтесь с Gravity Scale внутри **Rigidbody2D** и силой прыжка внутри **PlayerController**, чтобы сделать тот прыжок, который вам понравится

Поиграем!

Игра «Крокодил»

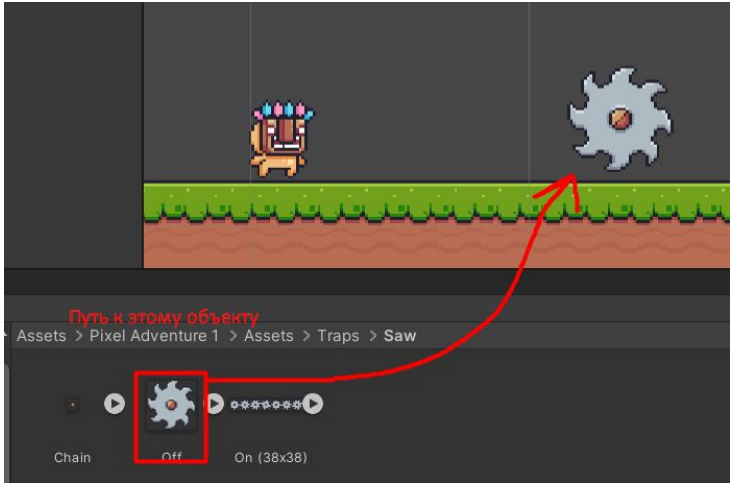
Столкновения – самый просто способ сделать взаимодействие в игре. Оно есть практически в каждом шутере или стратегии.

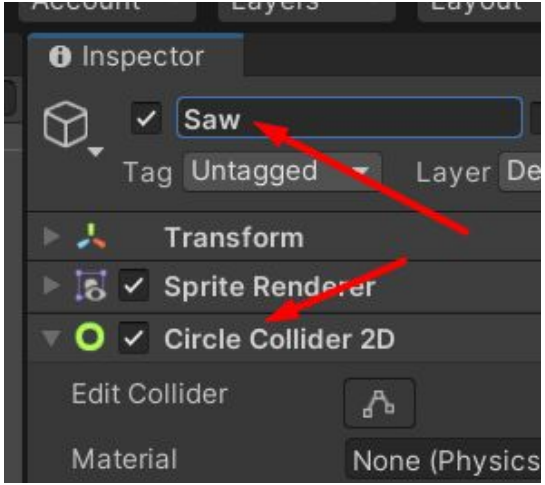
Collision (англ. столкновение) – термин, который используют профессионалы вместе слова “столкновение”

Ловушка-препятствие

Попробуем создать ловушку и сделать столкновение с ней.
В папке Saw перенесите объект **Off** на сцену.

Добавьте **Circle Collider 2D** и переименуйте объект в **Saw**.





Для того чтобы отслеживать соприкосновение с другими объектами, понадобится событийная функция **OnCollisionEnter2D**. Если на двух объектах есть коллайдер и хотя бы на одном есть Rigidbody2D, то произойдет столкновение и сработает функция.

```
Сообщение Unity | Ссылок: 0
private void OnCollisionEnter2D(Collision2D collision)
{
    print("Столкновение");
}
```

```
private void OnCollisionEnter2D(Collision2D collision)
    {
        print("Столкновение");
    }
```

```
Сообщение Unity | Ссылок: 0
private void OnCollisionEnter2D(Collision2D collision)
{
    print(collision.gameObject.name);
}
```

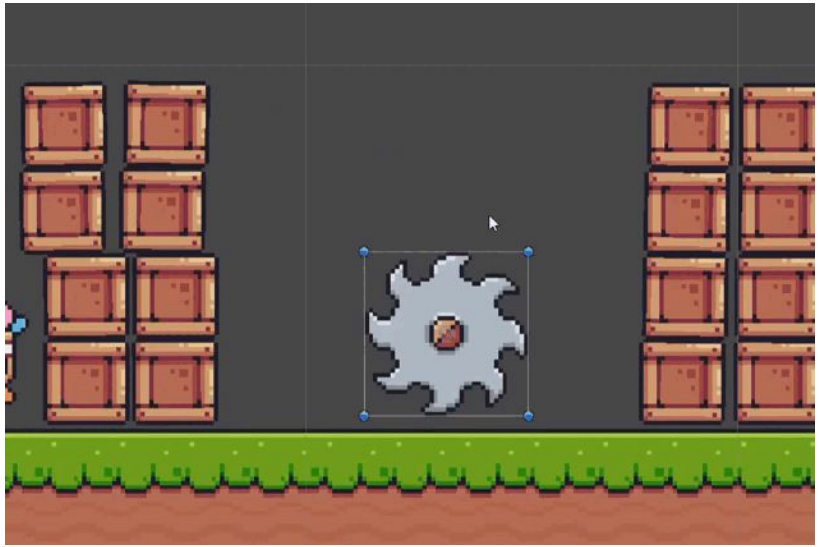
Collision2D collision – это создание новой переменной, где **Collision2D –** это тип данных, а **collision –** это название. В данной переменной содержится информация, с каким объектом произошло столкновение.

То есть в print мы можем обратиться к переменной, затем к объекту и затем к его имени. Таким образом мы выведем в консоль название объекта, с которым столкнулись.

Ловушка-препятствие

Мы сделаем так, что при соприкосновении ловушка уничтожит игрока. Для этого воспользуемся функцией **Destroy**, которая уничтожает тот объект, который мы укажем в скобках.

Нужно ли нам уничтожать все объекты, с которыми сталкивается ловушка? Тайлмап? Декорации? Чтобы не уничтожать все подряд используем условие. Оно будет спрашивать, кто же столкнулся с ловушкой, и если это игрок, то уничтожим его! В этом нам поможет **тег**



Тег – слово, которое можно закрепить на любом игровом объекте и в нужный момент обращаться к нему.

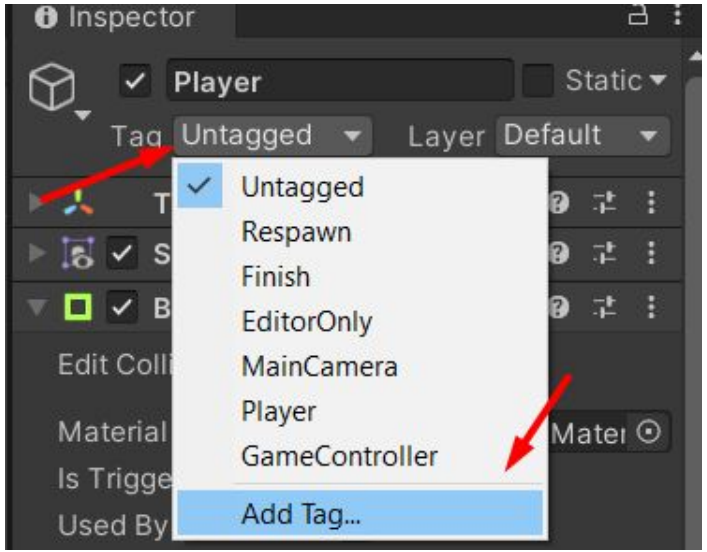
Используется чаще всего в функциях, отслеживающих **коллизии** (столкновения)

Похож на использование штрих-кодов из реальной жизни.

Тег можно выбрать в **Инспекторе** под полем названия объекта.

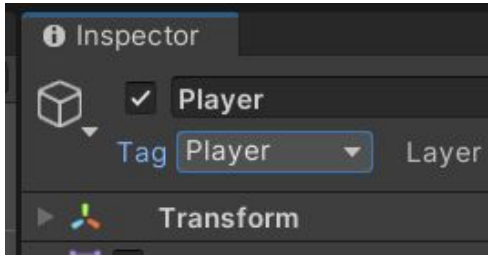
Изначально уже есть несколько созданных тегов. Например, **Player** или **MainCamera**

Кнопка **Add Tag** позволяет добавить свой собственный тег в этот проект



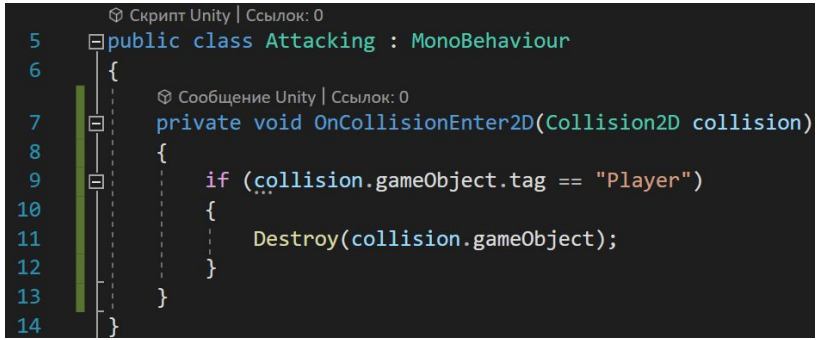
Ловушка-препятствие

Как мы видим, тег с игроком уже есть, поэтому пока можно не добавлять ничего. Нажмите на игрока и **выберите** ему тег **Player.**



Для ловушки создайте скрипт **Attacking** и вставьте код ниже:

```
private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.tag == "Player")
        {
            Destroy(collision.gameObject);
        }
    }
```



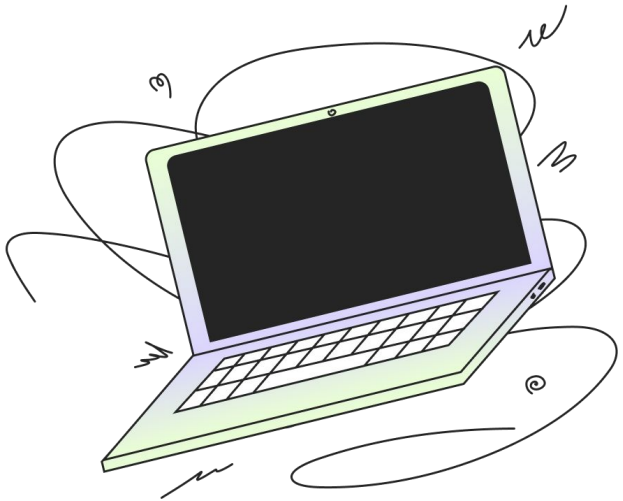
Эти конструкции позволяют спросить, такой ли тег у объекта, с которым столкнулась наша ловушка. Оба способа выполняют одинаковую функцию, если вдруг один из способов у вас не сработал, то попробуйте другой

```
private void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.gameObject.tag == "Player")
    {
        Destroy(collision.gameObject);
    }
}
```

```
private void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.gameObject.CompareTag("Player"))
    {
        Destroy(collision.gameObject);
    }
}
```

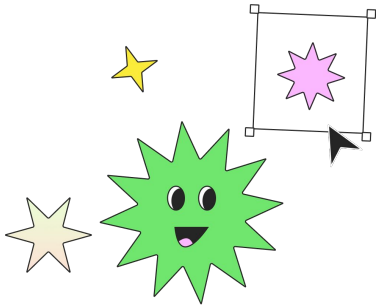
Практика

1. Создайте префаб каждой ловушки, который будет уничтожать ваш объект.
2. Создайте папку Prefabs и положите туда эти объекты.
3. Расставьте некоторые ловушки по вашей игре и попробуйте ее пройти!

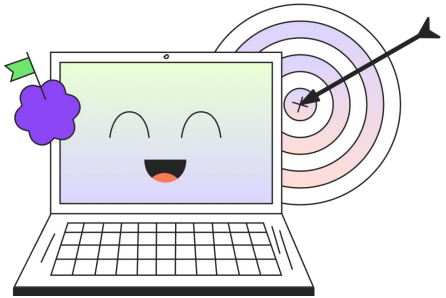


Рефлексия

Вам понравился урок?



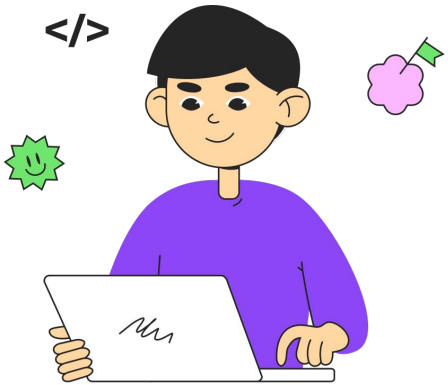
Сложно было?



Получилось справиться с заданиями?

Сегодня мы с вами

- 😎 Добавили столкновение объектов
- 😎 Научили персонажа прыгать
- 😎 Создали ловушки, которые наносят урон: лезвия, шипы



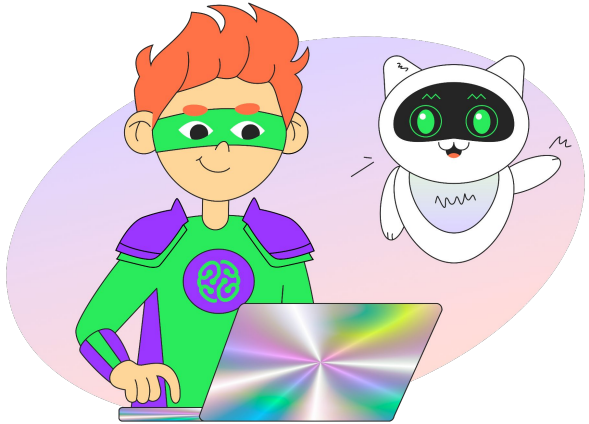
Время вопросов!

На следующем уроке

- 🎯 Учим противников передвигаться с помощью скрипта
- 🎯 Анимируем движения противников и препятствий
- 🎯 Изучаем поведение частиц огня

Домашнее задание

- Доделайте прыжок и ловушки, если не успели сделать на уроке
- Потренируйте скорость печати на английском языке на тренажере



Словарь терминов

📌 **Условие –** это логическое выражение, т.е. выражение, результатом которого является логическое значение true (истина) или false (ложь)

📌 **Условные операторы** — это группа символов (>,<, ==, ||, &&, <=,>=), которые помогают сравнивать значения внутри условий и в зависимости от результата сравнения выполнять конкретные действия

📌 **Тег** – слово, которое можно закрепить на любом игровом объекте и в нужный момент обращаться к нему.

Словарь английских слов

- 📌 **True** |truː| – истина
- 📌 **False** |fɔːls|– ложь
- 📌 **Input** |ˈɪnpʊt| – ввод, входной
- 📌 **Collision** |kəˈlɪʒn| – столкновение
- 📌 **Attacking** |əˈtækɪŋ| – атакующий

Заполни, пожалуйста,
форму обратной связи по уроку

Напоминание для преподавателя

- **Проверить заполнение Журнала**
- **Заполнить форму Т22**