

Лабораторная работа. Создание 2D-игры кликера ОПАОПА САТ.

Цель работы: Создание 2D-игры кликера ОПАОПА Cat с использованием игрового движка Unity и инструментов разработки. Проект позволит получить практические навыки в программировании, проектировании игры и работе с графикой.

Задачи:

1. Разработка концепции игры:

- Определение основных механик и правил игры.
- Разработка схемы управления и взаимодействия пользователя.

2. Создание пользовательского интерфейса (UI):

- Разработка начального экрана с кнопкой "Играть".
- Создание экрана игры с отображением очков, таймера и системы частиц.

3. Программирование игрового процесса:

- Реализация спавна и вращения объектов (котов).
- Изменение интервала появления объектов (котов).
- Обработка кликов на объектах и начисление очков.
- Появление редкого кота, который начисляет большее количество очков.
- Добавление фоновой музыки и звуковых эффектов.
- Добавление системы частиц при уничтожении объектов.
- Реализация кнопки "Повторить игру" при окончании времени.
- Реализация системы ачивок.

4. Тестирование и отладка:

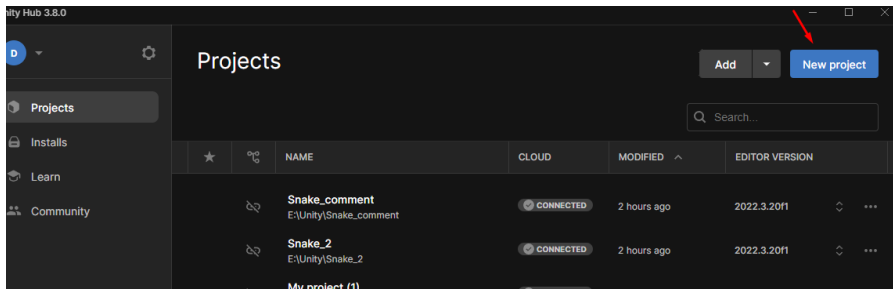
- Проведение тестирования для выявления багов и ошибок.
- Оптимизация игрового процесса для улучшения производительности.

5. Финальные настройки и сборка проекта:

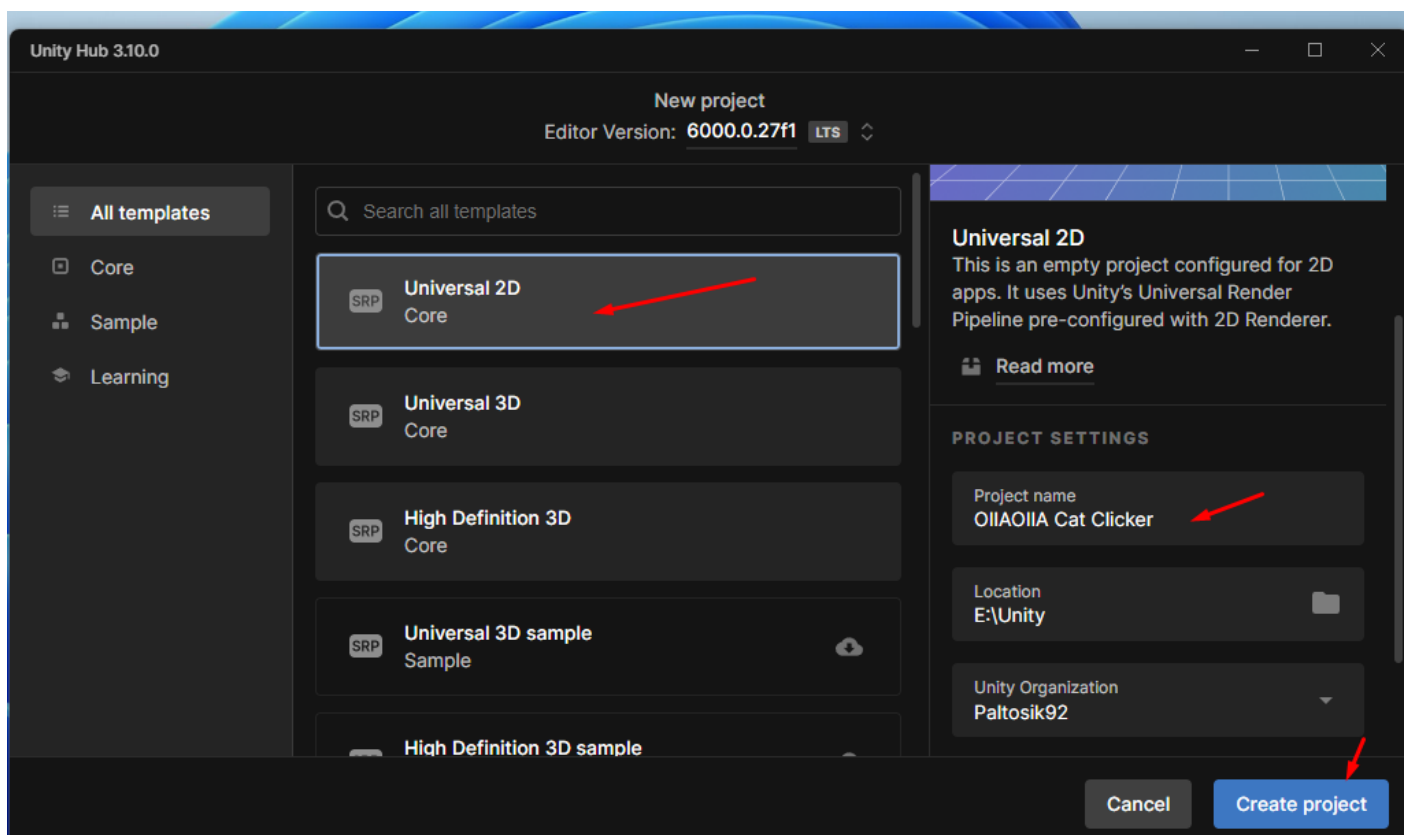
- Проведение рефакторинга кода для улучшения читаемости и производительности.
- Скомпилировать и сохранить финальную версию игры.

Ход работы:

1. Запускаем **Unity Hub**.
2. Создаём новый проект – **New project**:



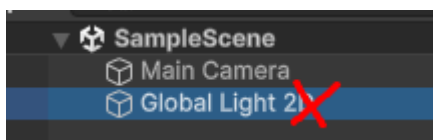
3. Выбираем **2D**. Вводим название проекта **ОПАОПА Cat Clicker**, выбираем место расположения, и нажимаем **Create project**.



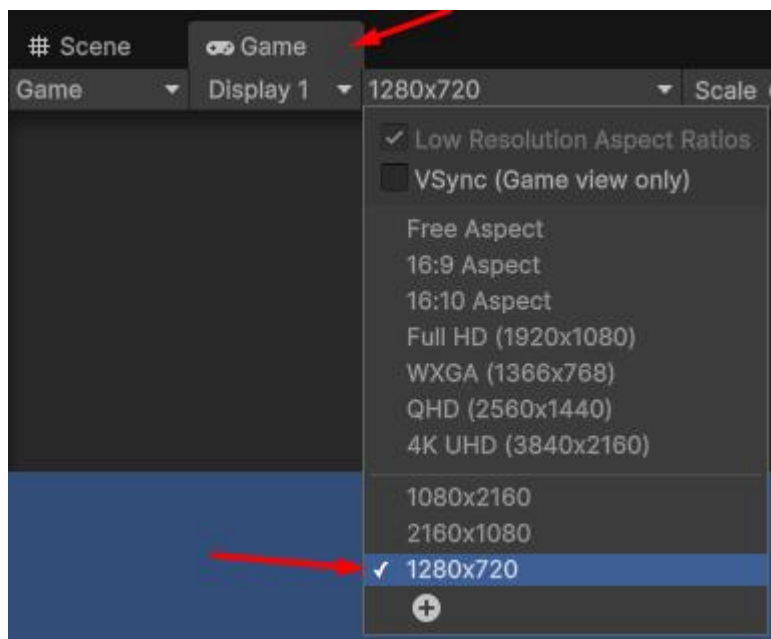
4. В папке **Scenes** меняем название сцены на **ClickerCat**:



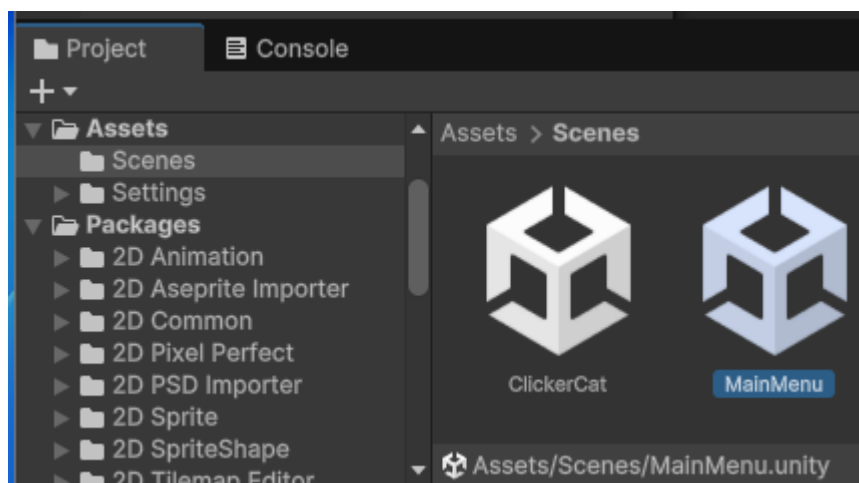
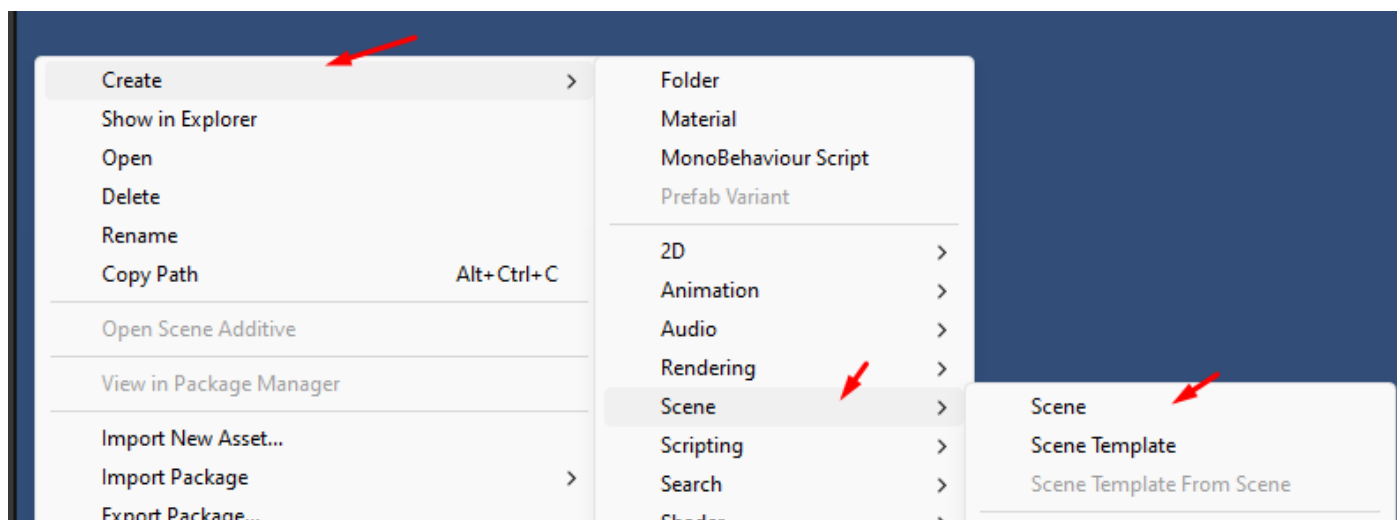
Удаляем объект **Global Light 2D**, он нам не понадобится:



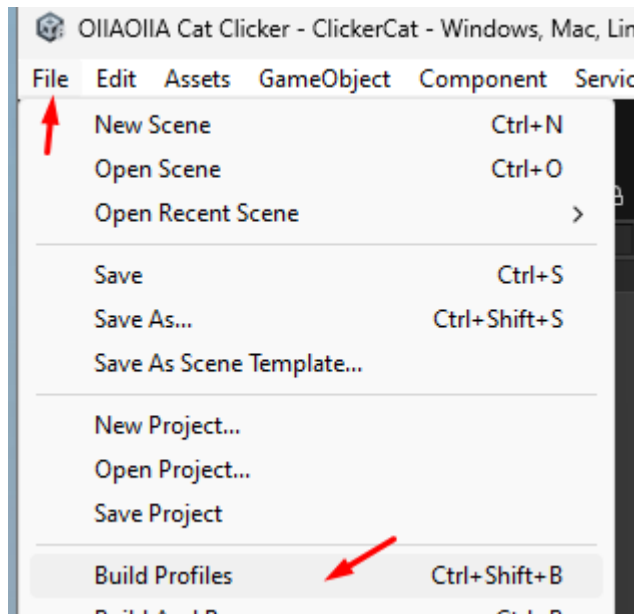
Во вкладке **Game** поменяем разрешение для нашей игры, в качестве примера выставим **1280x720**. Для этого нужно нажать на + и ввести вручную разрешение, после оно появится у вас в выборе:



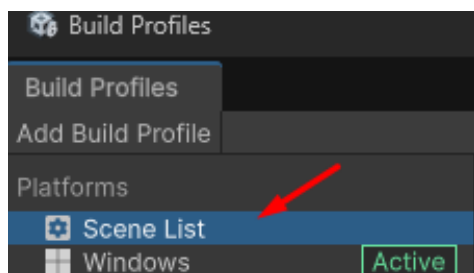
5. Добавим ещё одну сцену, назовём её **MainMenu**:



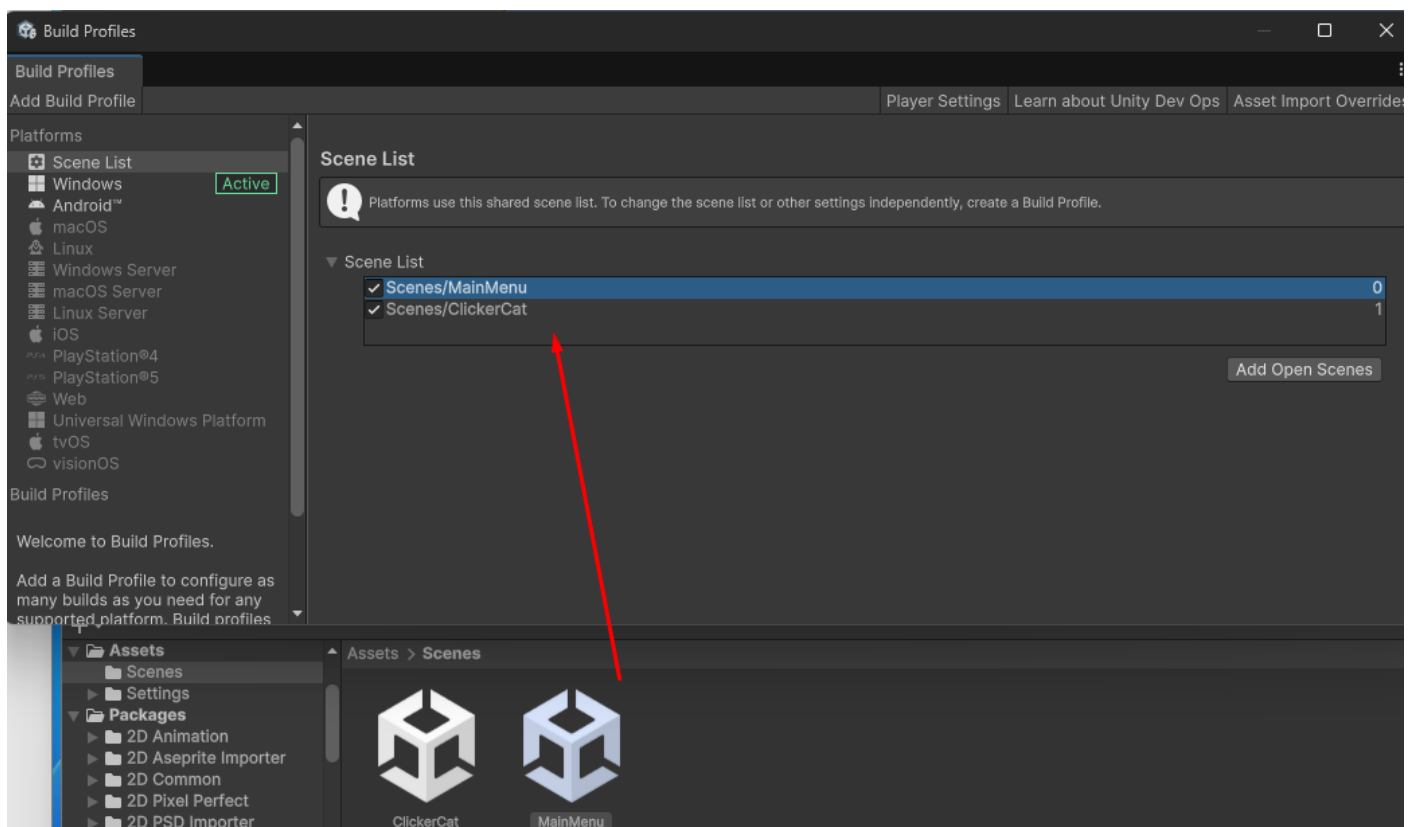
Теперь нам нужно подключить нашу сцену в проект. Для этого перейдите в **Build Profiles**:



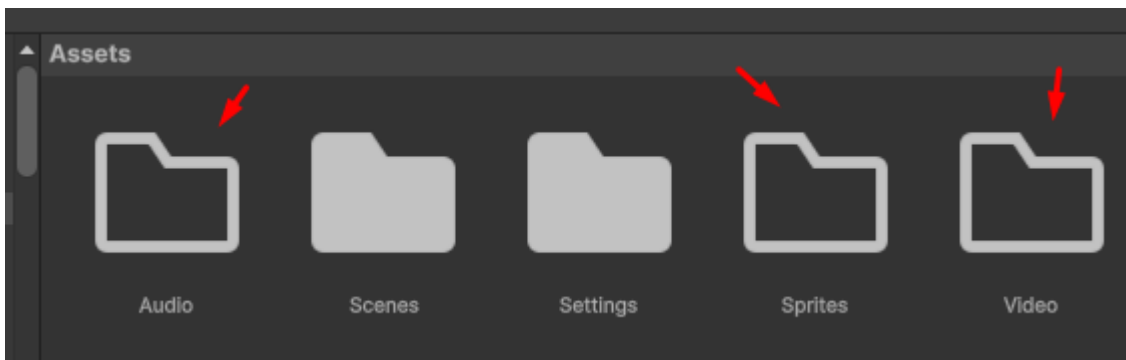
Откройте **Scene List**:



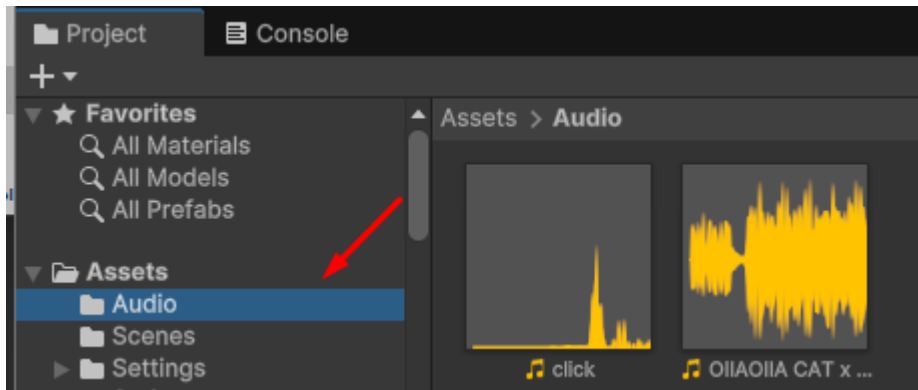
Перетащите и настройте правильный порядок ваших сцен:



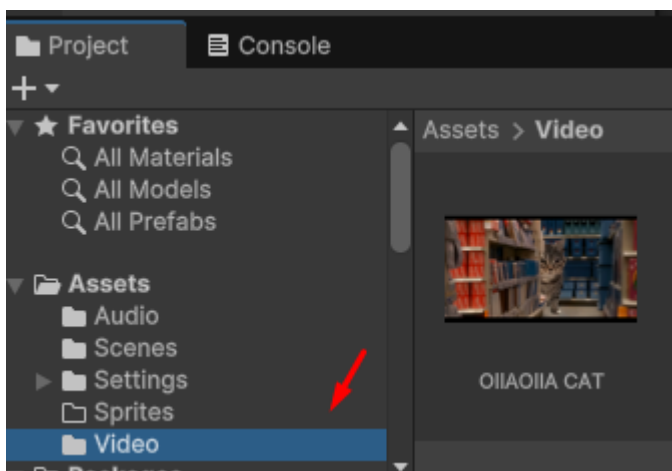
6. Создаём в нашем проекте папки **Video**, **Audio**, **Sprites**:



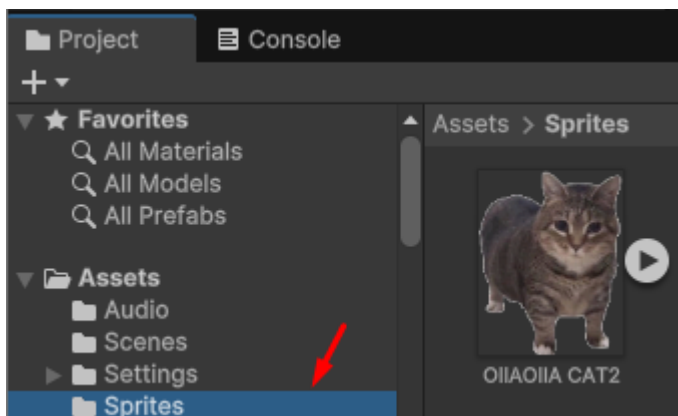
Из папки к лабораторной с ассетами переносим в папку **Audio** два **.mp3** звука:



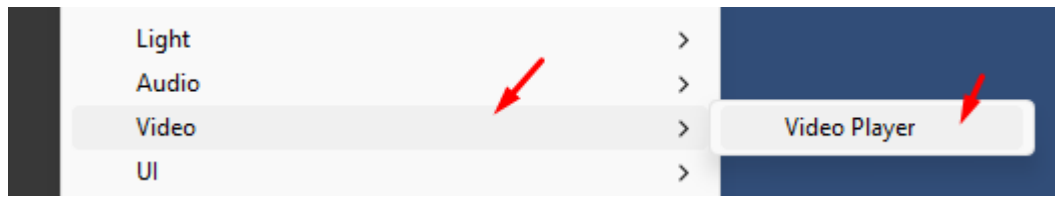
В папку **Video** перенесём наше фоновое видео:



В папку **Sprites** перенесём наше изображение:

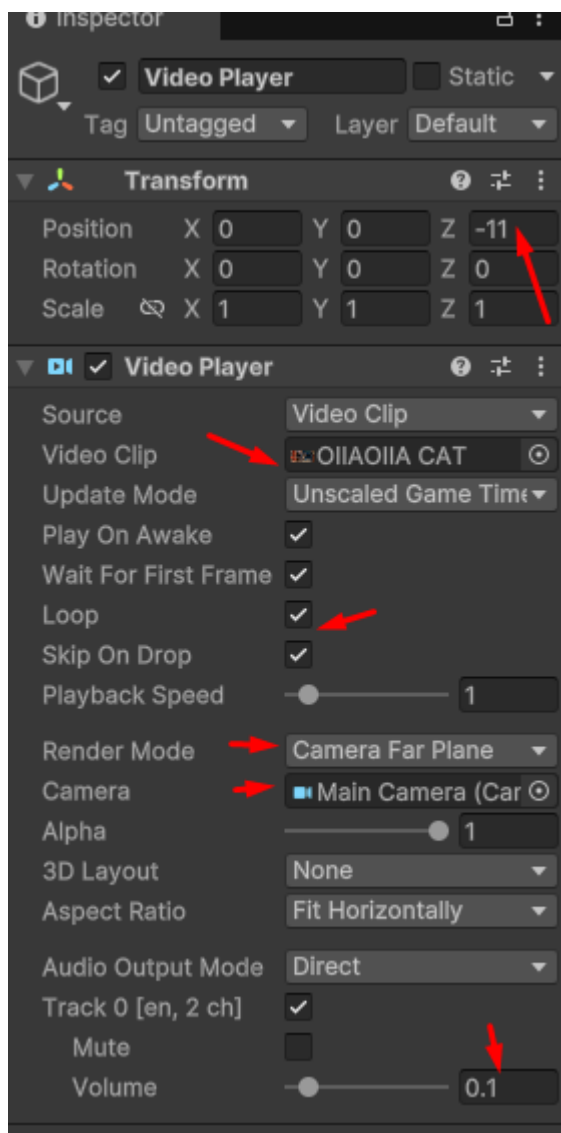


7. Перейдём к сцене **MainMenu**. Добавим фоновое видео. Для этого щёлкаем в нашей иерархии правой кнопкой мыши и выбираем **Video-Video Player**:



Произведём настройки:

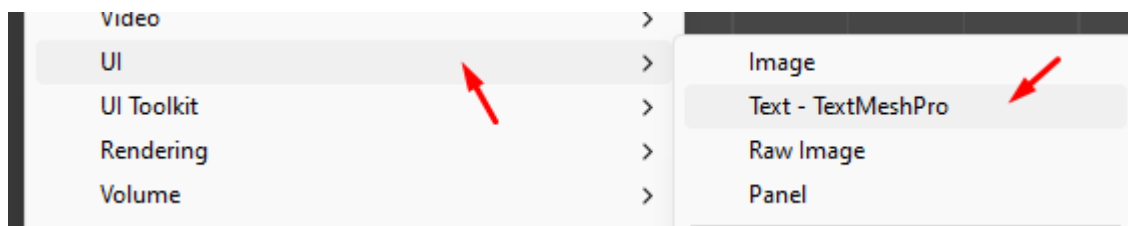
- сместим его по позиции **Z** на **-11**, чтобы он отображался на переднем плане в будущем корректно;
- в **Video Clip** переносим наш файл с видео из папки **Video**;
- ставим галочку на **Loop**, чтобы зацикливать видео;
- в **Render Mode** меняем на **Camera Far Plane** и переносим нашу камеру;
- громкость ставим потише на **0.1**.



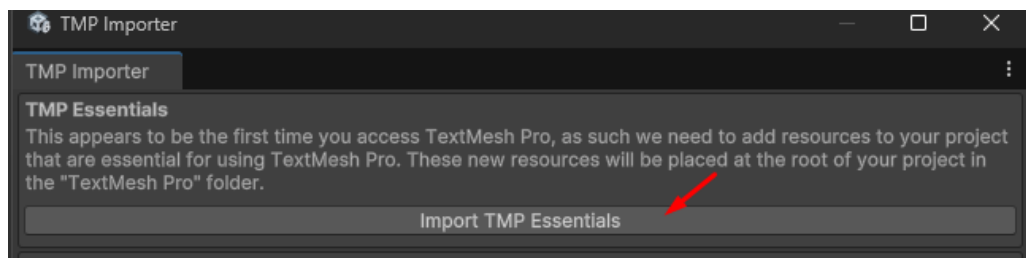
Теперь при запуске у вас будет идти фоновое видео.

8. Теперь создадим надпись с названием игры, и кнопку для перехода к следующей сцене.

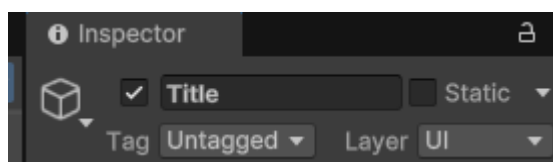
Щёлкаем правой кнопкой мыши – **UI – Text – TextMeshPro**:



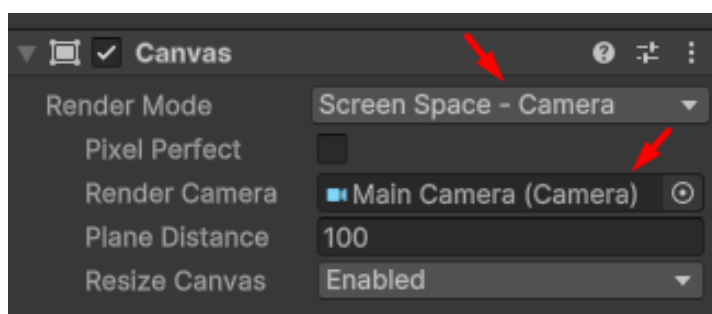
При появившемся окне нажимаем импортировать:



Меняем для текста название на **Title**:



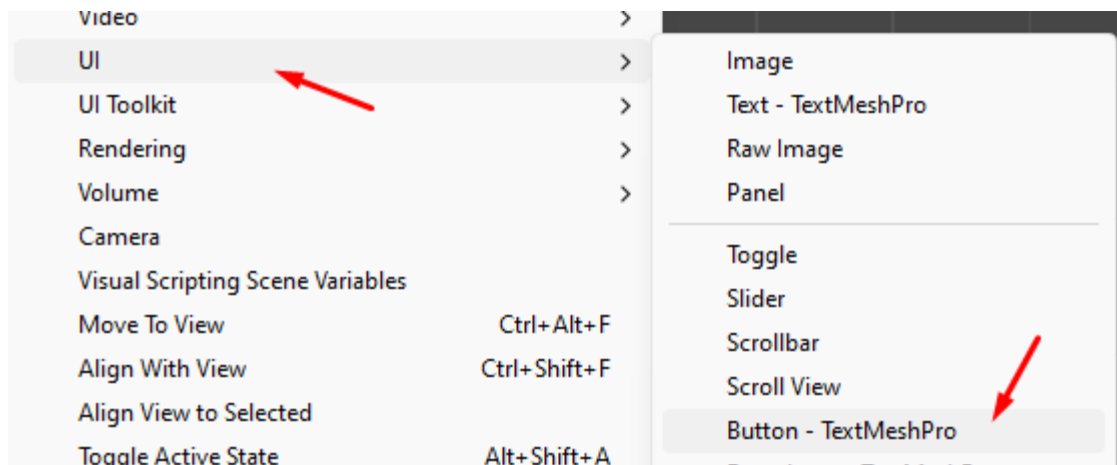
Для нашего Canvas меняем отображение на **Screen Space – Camera**, и добавляем нашу камеру:



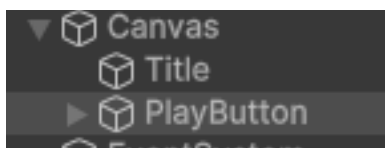
В поле текста пишем - **ОПАОПА CAT Clicker**. Настройте его на свой вкус, пример:



Внутри **Canvas** щёлкаем правой кнопкой мыши – **UI – Button – TextMeshPro**:



Называем её **PlayButton**:

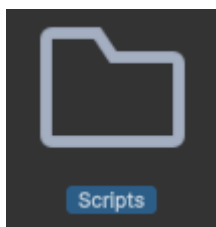


Меняем цвет текста и размер на своё усмотрение, пример:

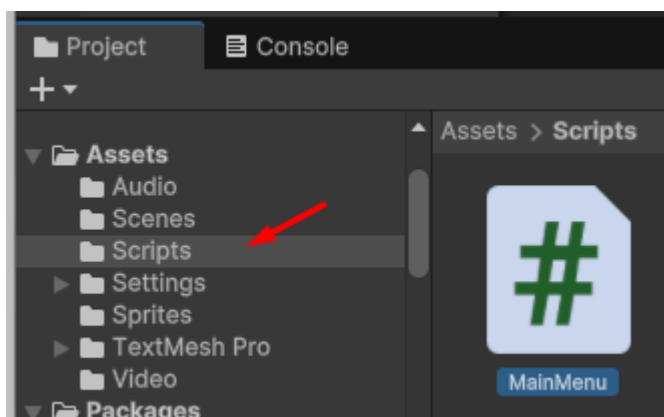


9. Теперь нам нужно создать скрипт для перехода на другую сцену, и прикрепить его к объекту.

Создаём папку **Scripts**:



В ней создаём скрипт **MainMenu**:



Открываем его и редактируем:


```

using UnityEngine;
using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour
{
    public void StartGame()
    {
        SceneManager.LoadScene("ClickerCat"); // Переход на
        новую сцену
    }

    public void ExitGame()
    {
#if UNITY_EDITOR
        UnityEditor.EditorApplication.isPlaying = false; //
        Для завершения игры в редакторе
    #else
        Application.Quit(); // Для завершения игры на
        устройстве
    #endif
    }
}

```

Мы дополнительно подключили библиотеку **UnityEngine.SceneManagement**, которая импортирует библиотеку для управления сценами в Unity.

Директивы препроцессора #:

1. В данном скрипте используются директивы препроцессора **#if** и **#else** для проверки условий компиляции. Это необходимо, чтобы различать поведение программы в зависимости от среды выполнения (редактор Unity или целевое устройство).

Объяснение директив:

2. **#if UNITY_EDITOR:**

- Эта директива проверяет, выполняется ли код в редакторе **Unity**. Если это так, выполняется следующий блок кода, вплоть до **#else**.

3. **UnityEditor.EditorApplication.isPlaying = false;**

- Эта строка завершает выполнение игры в редакторе **Unity**. Установка значения **false** останавливает режим игры.

4. **#else:**

- Если код не выполняется в редакторе **Unity**, то выполняется следующий блок кода, вплоть до **#endif**.

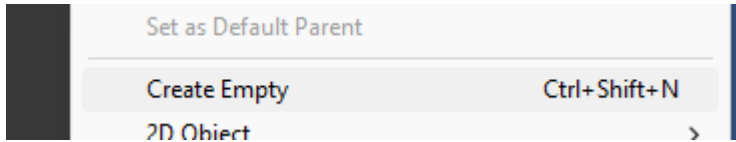
5. **Application.Quit();**

- Эта строка завершает выполнение игры на целевом устройстве. Метод **Application.Quit()** закрывает приложение.

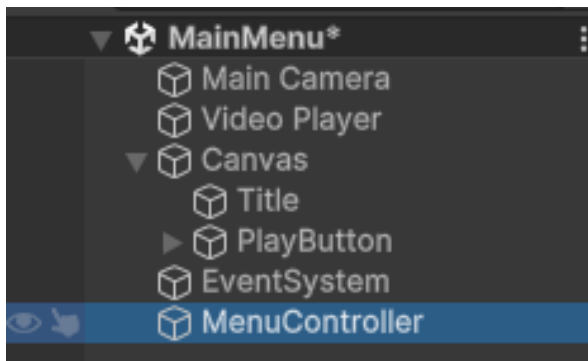
6. #endif:

- Конец блока директив препроцессора. Эта директива завершает условную компиляцию.

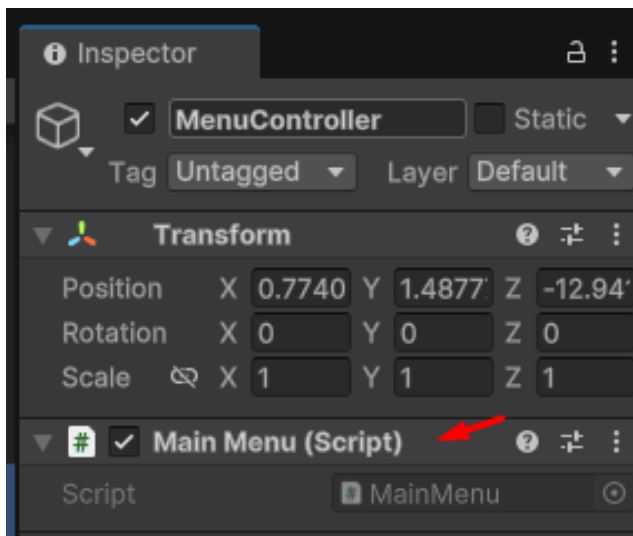
Затем создаём пустой объект в иерархии:



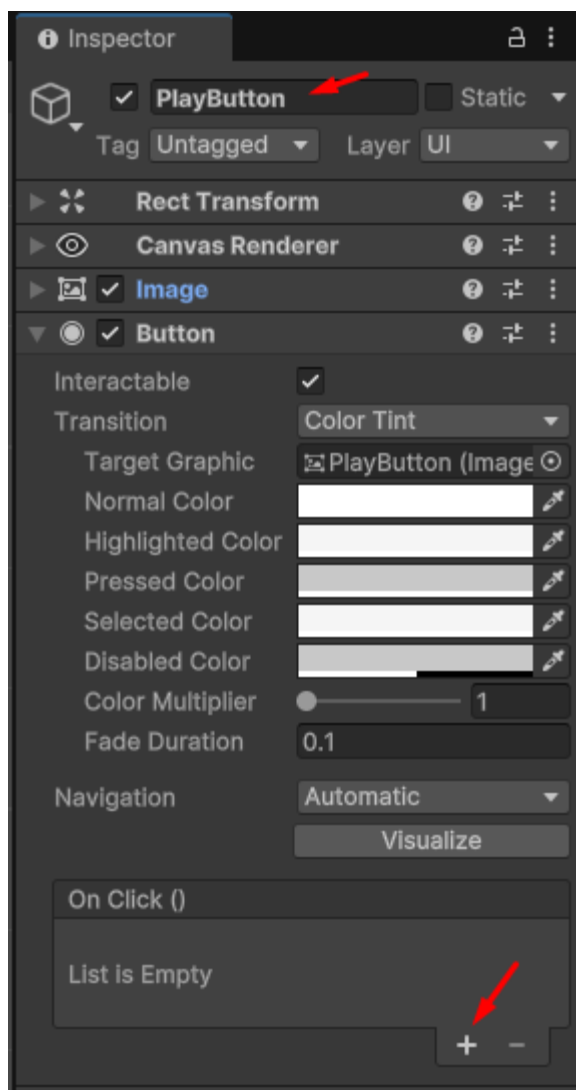
Назовём его **MenuController**:



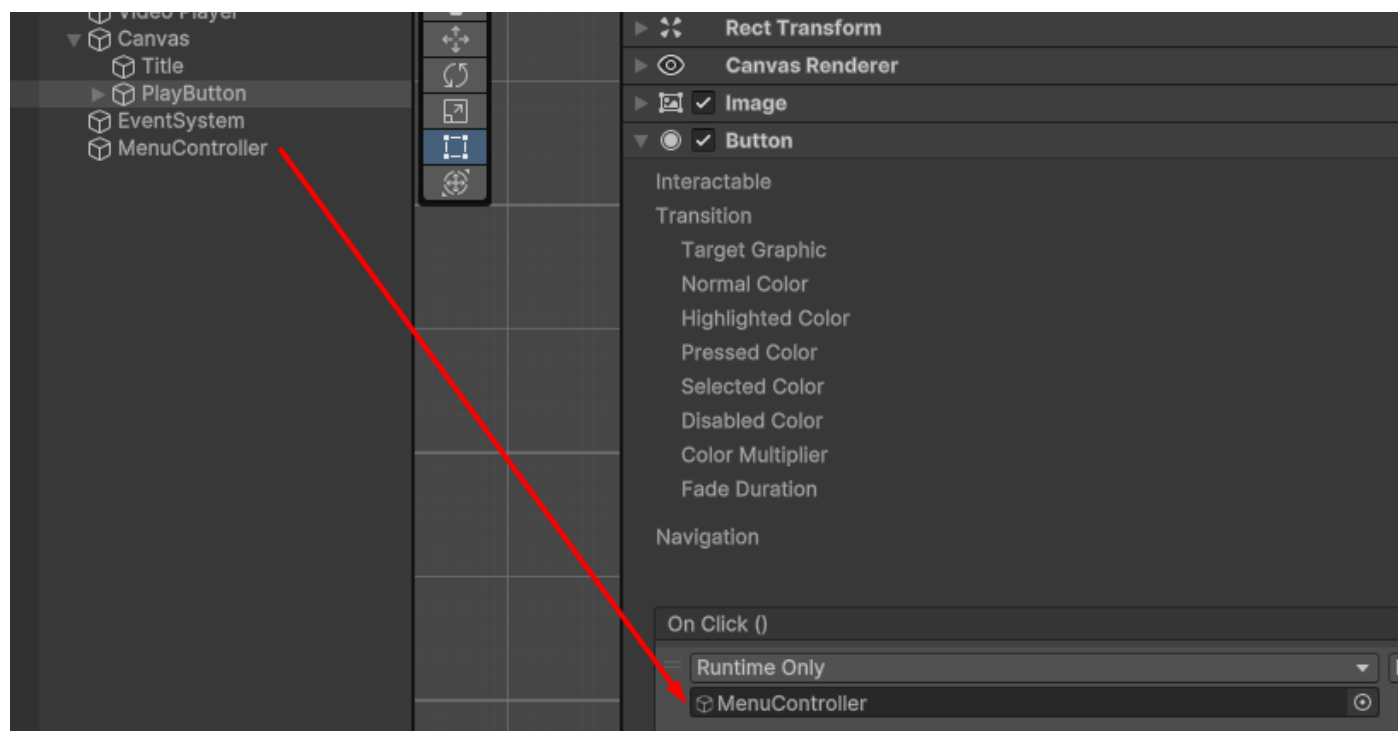
Добавляем к нему наш скрипт:



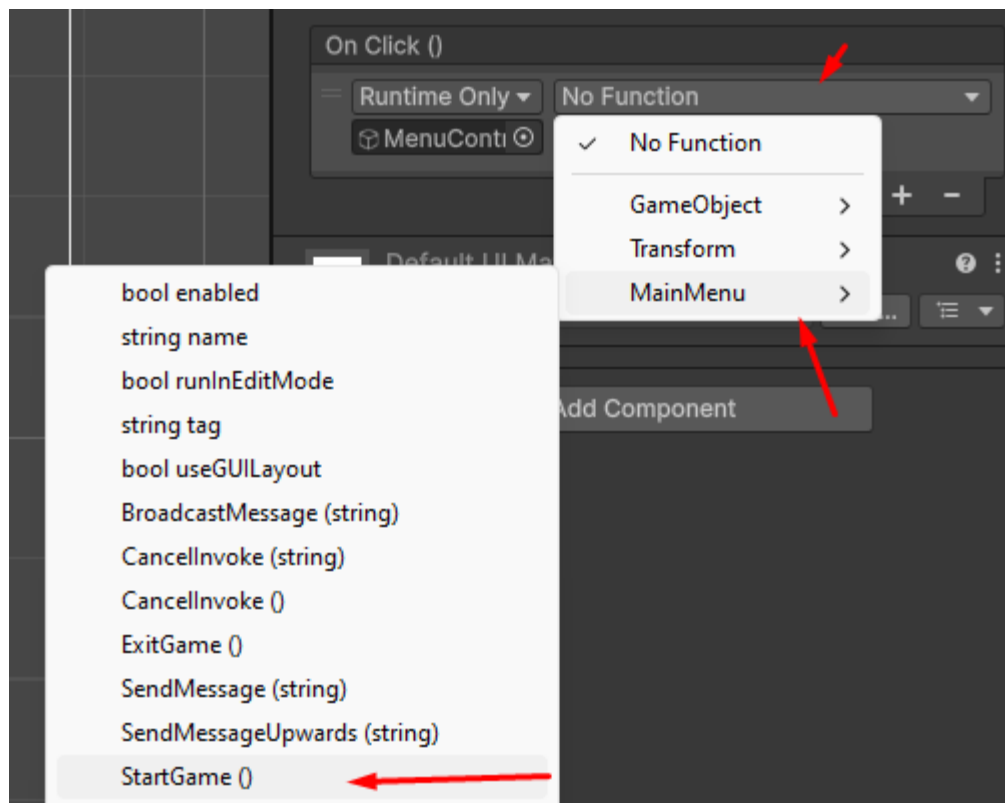
Далее нам нужно в нашу кнопку добавить новое событие в **On Click()** нажав на +:



После нам нужно перенести наш объект **MenuController** в пустое поле:

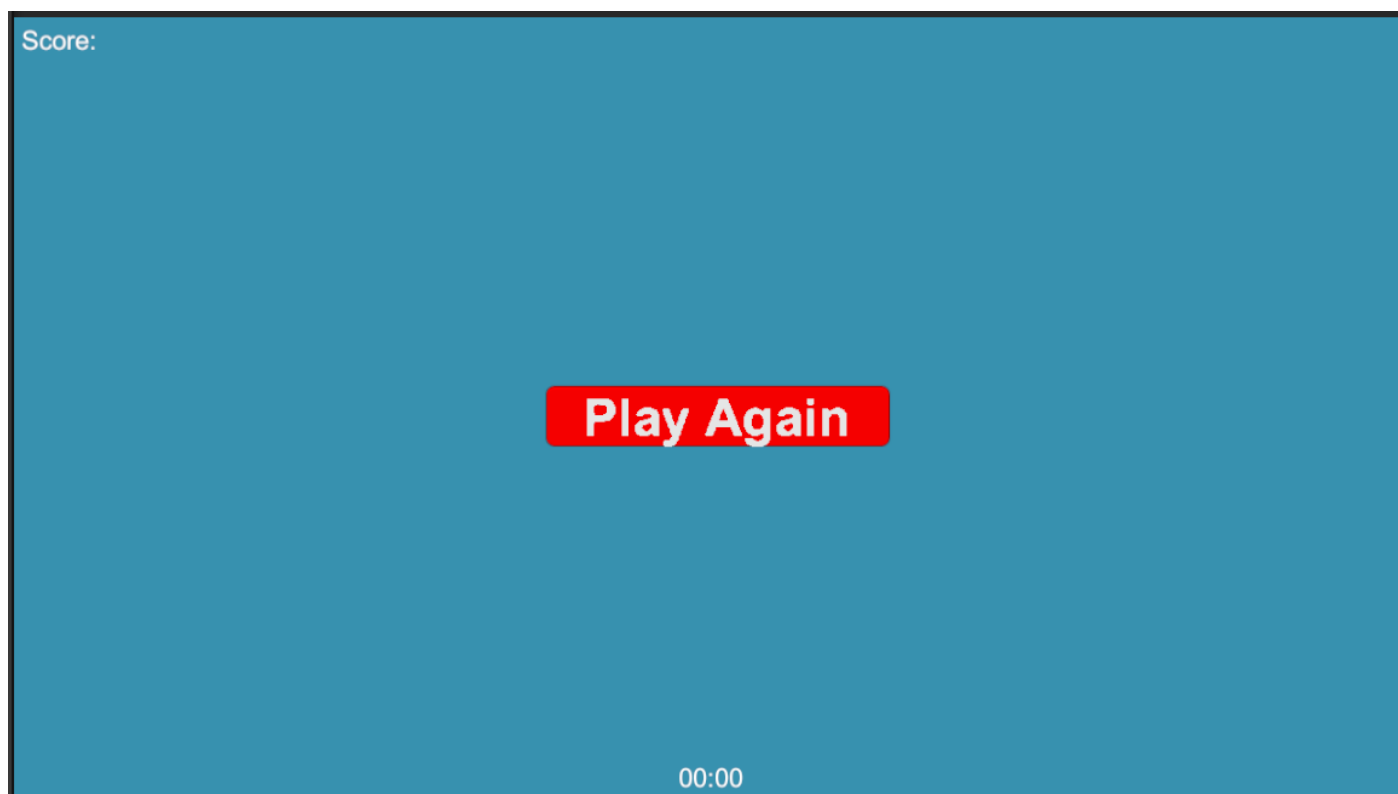


Далее щёлкаем на поле **Function**, и выбираем **MainMenu – StartGame()**:

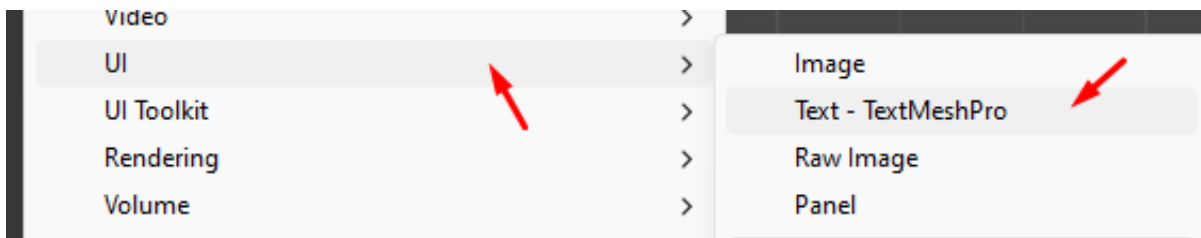


Проверьте что всё сделали правильно, запустив игру и нажав на кнопку, если вы перешли на вторую сцену, значит у вас всё получилось.

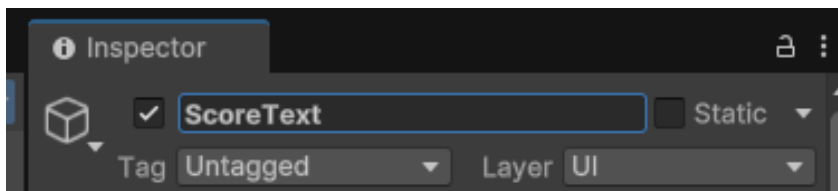
10. Переходим к основной сцене **CliclerCat**, дважды щёлкаем чтобы открыть её. Нам нужно будет оформить её приблизительно следующим образом:



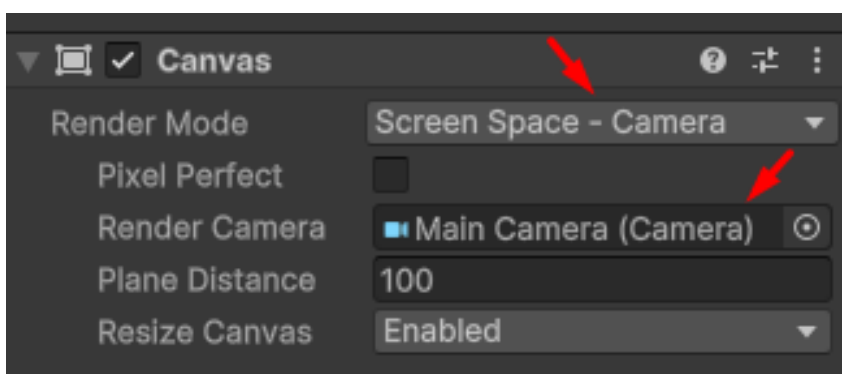
Щёлкаем правой кнопкой мыши – **UI – Text – TextMeshPro**:



Меняем для текста название на **ScoreText**:



Для нашего **Canvas** меняем отображение на **Screen Space – Camera**, и добавляем нашу камеру:

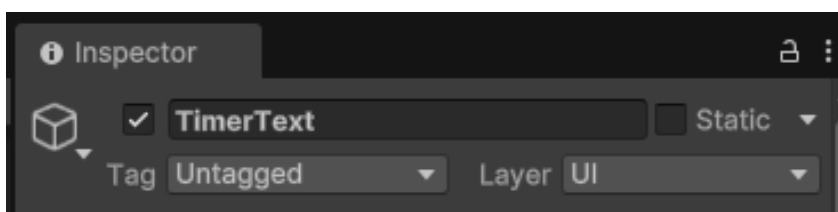


В поле текста пишем - **Score**:

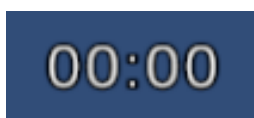
Настройте его на свой вкус, пример:



Теперь создадим аналогично текст для таймера, назовём его **TimerText**:

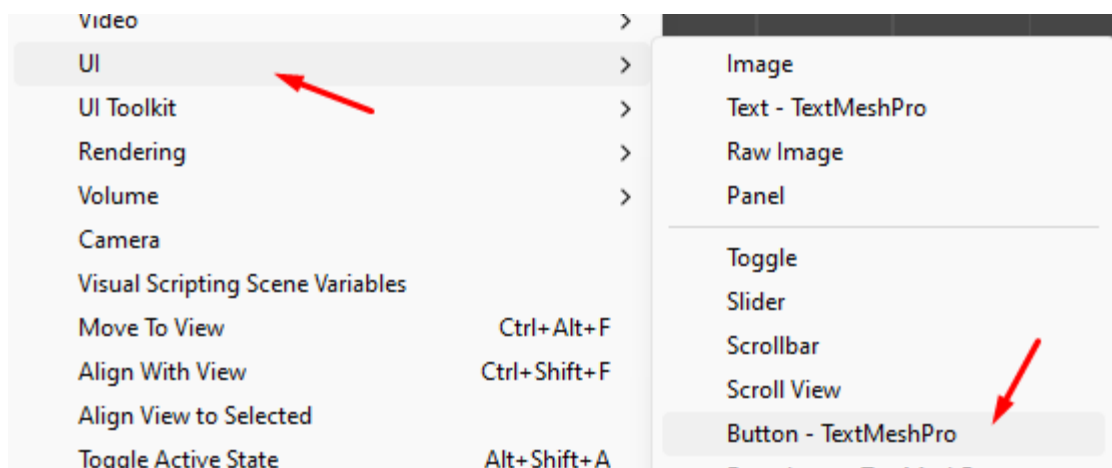


Пример оформления:

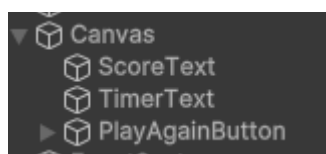


Теперь перейдём к созданию кнопки для продолжения начала игры.

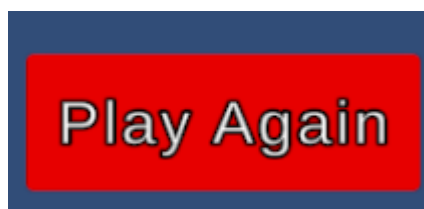
Внутри **Canvas** щёлкаем правой кнопкой мыши – **UI – Button – TextMeshPro**:



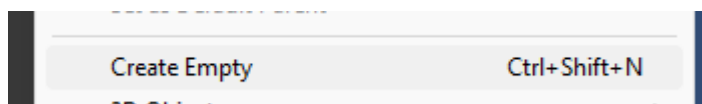
Называем её **PlayAgainButton**:



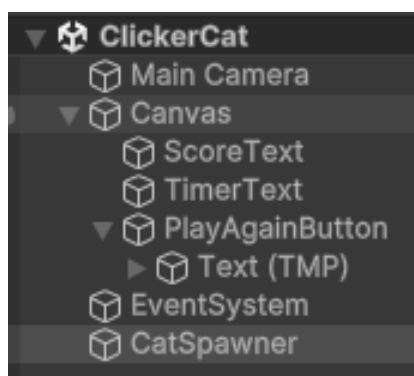
Меняем цвет текста и размер на своё усмотрение, пример:



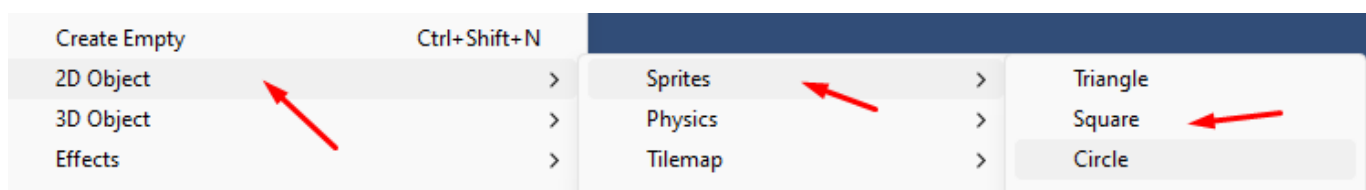
Также создадим пустой объект и назовём его **CatSpawner**



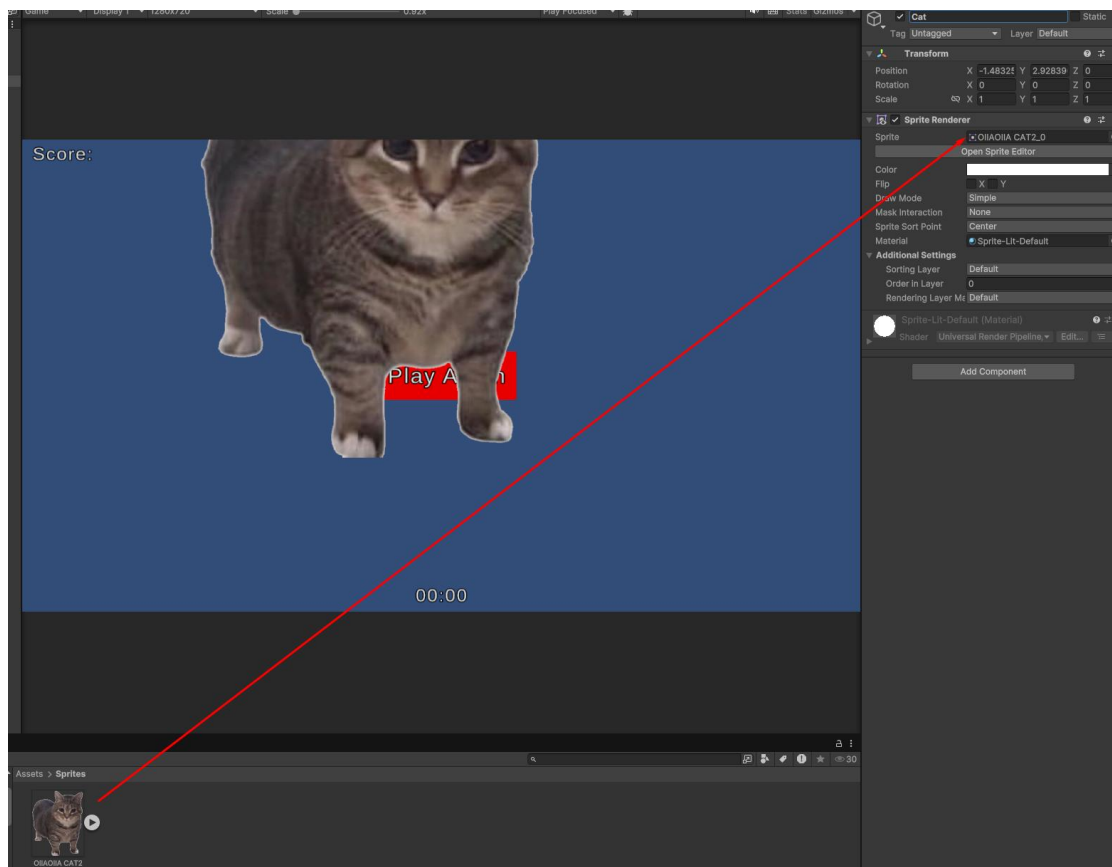
Итог:



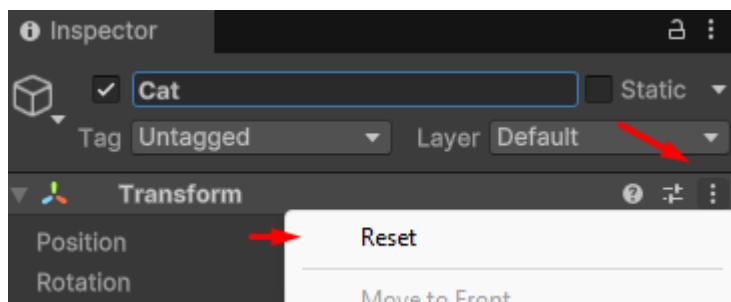
11. Создадим для нашего изображения основу, выберем **2D Object – Sprites – Circle**:



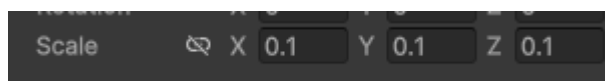
Назовём его **Cat**. Затем из папки **Sprites** перенесём нашего кота в поле **Sprite**:



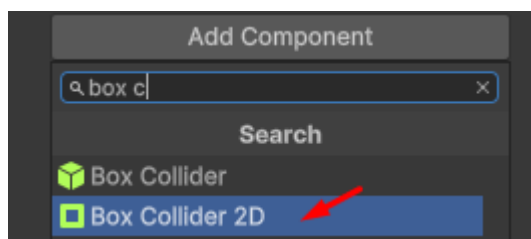
Сбросим позицию:



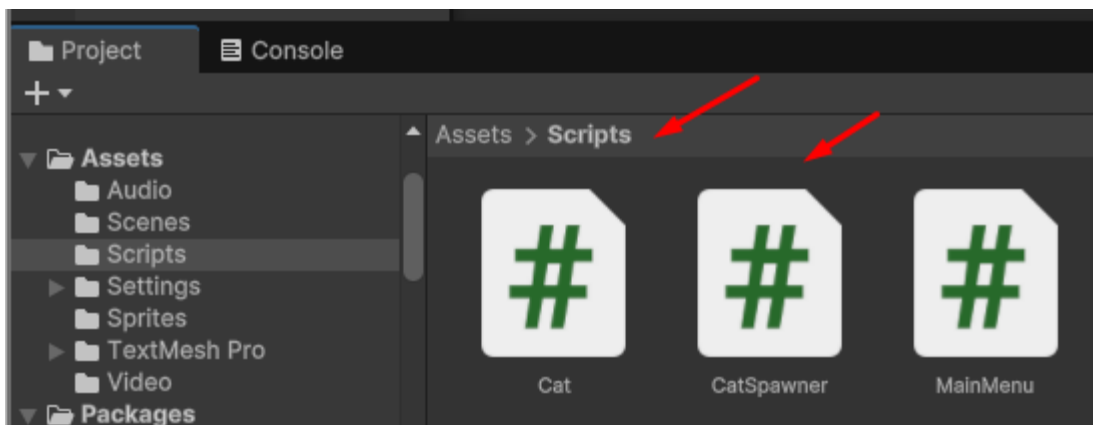
Уменьшим размер до **0.1** по **x,y,z**:



Добавим **Box Collider 2D**:



12. Создадим в папке **Scripts** два скрипта – **Cat** и **CatSpawner**:

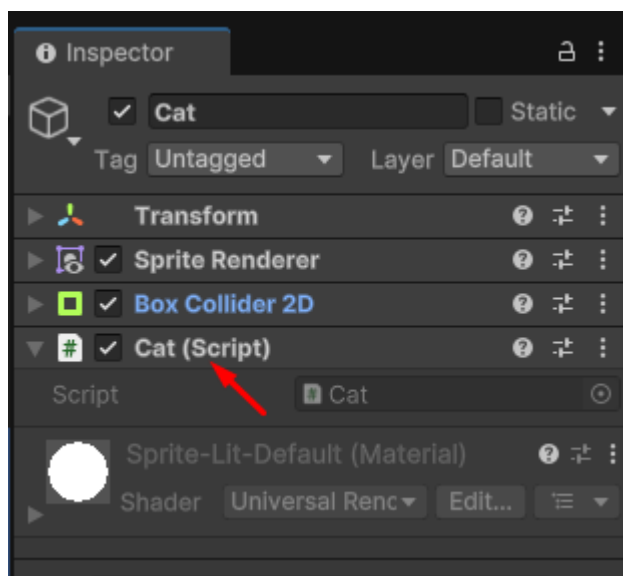


Переходим к скрипту **Cat**. Открываем его и пишем:

```
using UnityEngine;

public class Cat : MonoBehaviour
{
    private float rotationSpeed; // переменная для изменения
    вращения кота
    private void Start()
    {
        rotationSpeed = Random.Range(200f, 800f); // Генерируем
    случайную скорость вращения
    }
    private void Update()
    {
        transform.Rotate(0, 0, rotationSpeed * Time.deltaTime); //
    }
    Вращаем объект вокруг оси Z
}
```

Добавим скрипт к нашему коту:

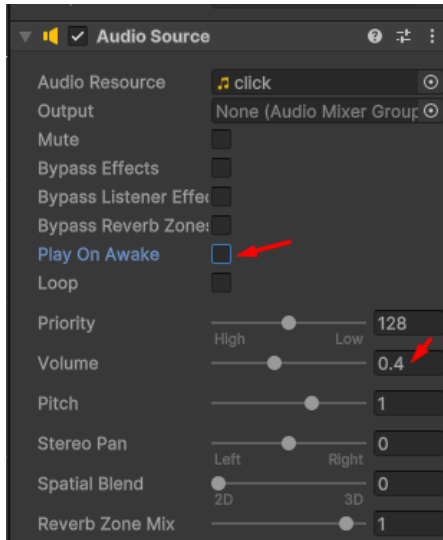


Теперь если мы запустим нашу игру, то увидим вращение у кота.

13. Далее добавим возможность удаления кота при нажатии на нём мышкой. Для этого добавим в скрипт метод **OnMouseDown**:

```
private void OnMouseDown()  
{  
    Destroy(gameObject);  
}
```

14. Добавим звук клика, при нажатии на кота. Для этого на наш объект кота добавляем компонент **Audio Source** и переносим на него звук клика. Сделаем также потише громкость, и уберём галочку с проигрывания при инициализации:



Добавим переменную для хранения звука:

```
private AudioSource audioSource;
```

В методе **Start** получим ссылку на наш компонент:

```
audioSource = GetComponent<AudioSource>();
```

В методе **OnMouseDown** добавим:

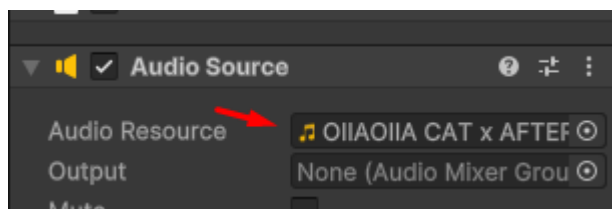
```
audioSource.Play();  
Destroy(gameObject, audioSource.clip.length);
```

audioSource.clip.length возвращает длину звукового клипа в секундах. Метод **Destroy** принимает два параметра: объект, который нужно уничтожить (**gameObject**), и задержку в секундах перед уничтожением (в данном случае, длительность звука). Поэтому, в нашем случае обеспечивается плавное воспроизведение звукового эффекта до конца, прежде чем объект исчезнет.

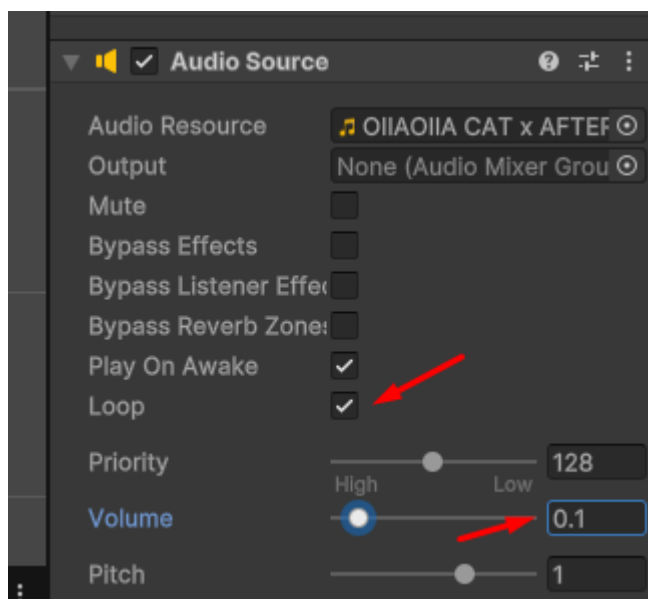
15. Добавим воспроизведения нашего трека при начале игры. Для этого выбираем **Main Camera – Add Component – Audio Source**:



Переносим из папки **Audio** в **Audio Resource** наш трек:

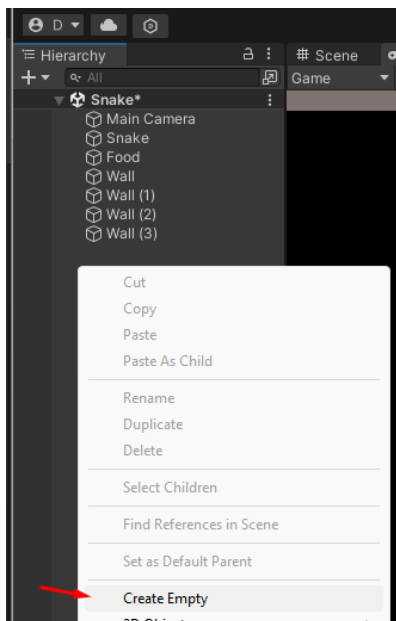


Сделаем звук немного потише, и включим зацикливание трека:

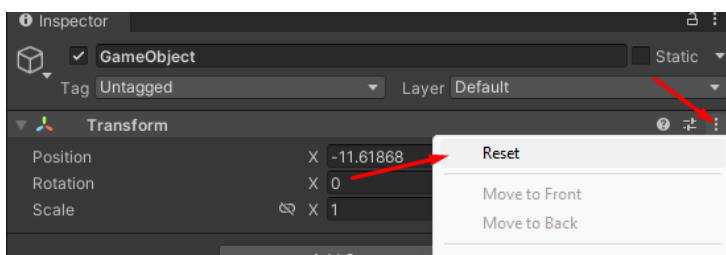


16. Теперь создадим возможность появления котов в случайной позиции на экране.

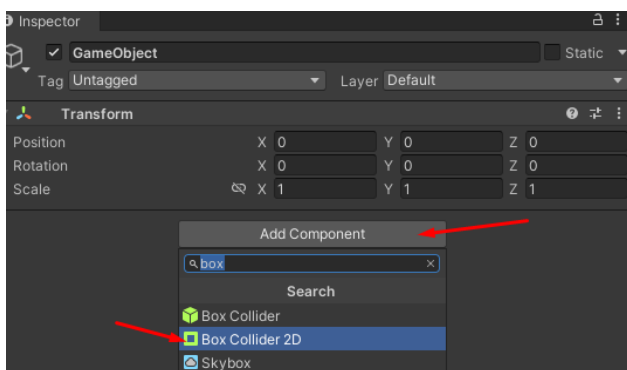
Нажимаем правой кнопкой мыши в нашей **Hierarchy** → **Create Empty**:



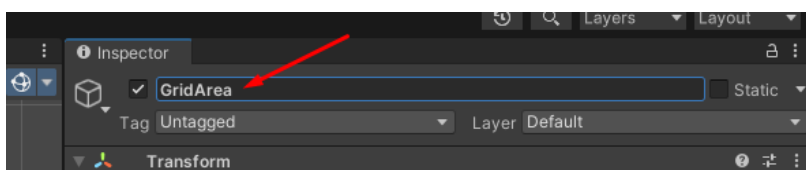
Reset:



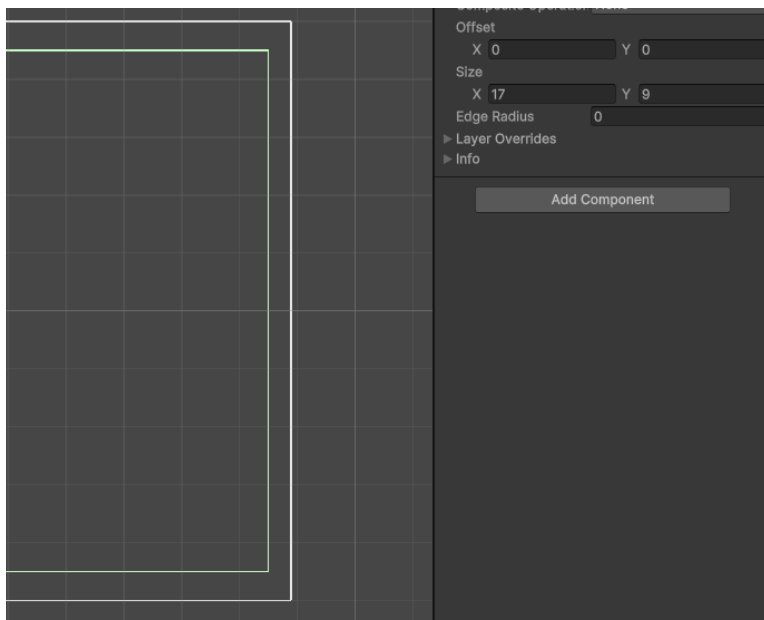
Добавляем **Box Collider 2D**:



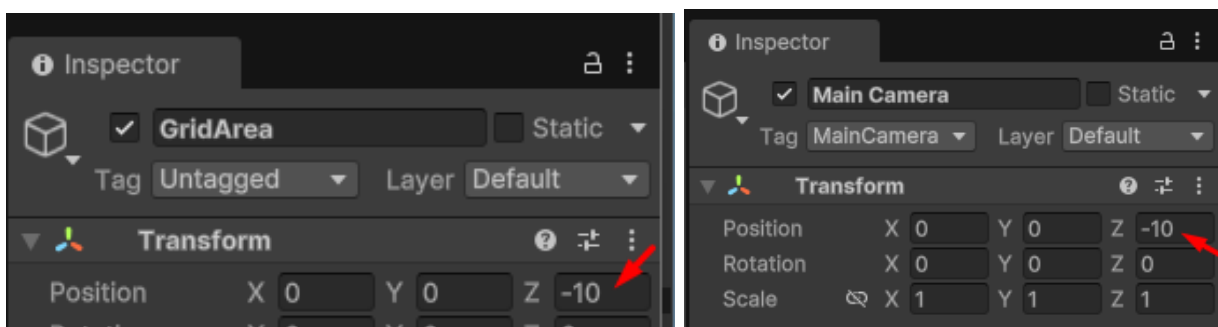
Называем **GridArea**:



Настроим размер **Box Collider 2D** по вашему усмотрению (в моём случае это **x = 17** и **y = 9**):

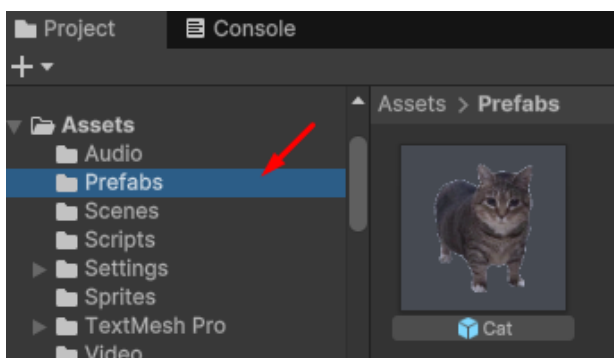


Обязательно проверьте, чтобы у вас позиция по оси Z у Camera и GridArea были одинаковы!

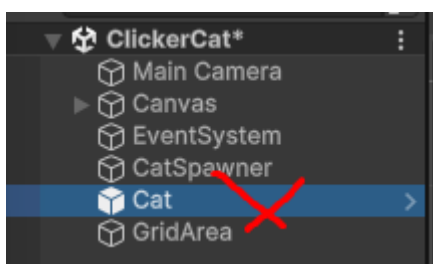


17. Создадим из нашего кота **Prefab**.

Создаём новую папку **Prefabs** и перетаскиваем наш объект **Cat** в неё:



И удаляем его со сцены:



18. Теперь откроем скрипт **CatSpawner**. Очищаем его от кода внутри класса:

```
using UnityEngine;

Скрипт Unity | Ссылок: 0
public class CatSpawner : MonoBehaviour
{
    //
}
```

Объявляем переменные:

```
public GameObject catPrefab; // Ссылка на префаб объекта,
который будет спавниться (кот).
public float minSpawnInterval = 0.5f; // Минимальный интервал
между спавнами
public float maxSpawnInterval = 0.7f; // Максимальный интервал
между спавнами
public float gameTime = 60f; // Время игры в секундах
public bool isActive = true; // Флаг для отслеживания
состояния игры
public GameObject playAgainButton; // Ссылка на кнопку "Play
Again"
public BoxCollider2D GridArea; // Границы области, в которой
будут спавниться объекты.
private float nextSpawnTime; // Время для следующего спавна
```

Далее добавим метод **Start()**:

```
private void Start()
{
    playAgainButton.SetActive(false); // Скрываем кнопку "Play
Again" при старте игры
    nextSpawnTime = Time.time + Random.Range(minSpawnInterval,
maxSpawnInterval); // Задаем случайное время для первого спавна
объекта после начала игры.
}
```

После пропишем метод **Update()**:

```
private void Update()
{
    if (gameTime > 0)
    {
        gameTime -= Time.deltaTime; // Уменьшаем общее время игры с
течением времени

        if (Time.time >= nextSpawnTime) //Проверяем, прошло ли
время для следующего спавна объекта
        {
            SpawnCat(); // Вызываем метод для спавна объекта.
        }
    }
}
```

```

        nextSpawnTime = Time.time +
Random.Range(minSpawnInterval, maxSpawnInterval); // Устанавливаем
случайный интервал времени до следующего спавна
    }
}
else if (isGameActive) // Если время игры истекло, отключаем
активное состояние игры и показываем кнопку "Play Again".
{
    isGameActive = false; // Останавливаем игру
    playAgainButton.SetActive(true); // Показываем кнопку "Play
Again"
    enabled = false; // Останавливаем спавн шариков
}
}

```

И последний метод **SpawnCat**:

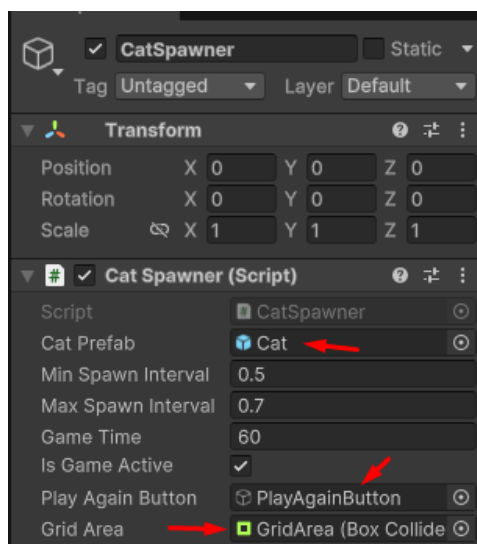
```

private void SpawnCat()
{
    Bounds bounds = GridArea.bounds; // Получаем границы области
для спавна объектов.
    float x = Random.Range(bounds.min.x, bounds.max.x); //
Определяем случайную позицию по оси X в пределах границ.
    float y = Random.Range(bounds.min.y, bounds.max.y); //
Определяем случайную позицию по оси Y в пределах границ.
    Vector3 spawnPosition = new Vector3(x, y, 0); // Формируем
вектор позиции для спавна объекта.

    GameObject cat = Instantiate(catPrefab, spawnPosition,
Quaternion.identity); // Создаем объект в случайной позиции внутри
области.
}

```

Переносим скрипт **CatSpawner** на игровой объект **CatSpawner** и добавляем к нему соответствующие компоненты:



Обновим немного наш скрипт **Cat**:

```
Скрипт Unity (1 ссылка на ресурсы) | Ссылки: 0
public class Cat : MonoBehaviour
{
    private CatSpawner spawner; // получаем ссылку на скрипт CatSpawner
    private AudioSource audioSource;
    private float rotationSpeed;

    Сообщение Unity | Ссылки: 0
    private void Start()
    {
        audioSource = GetComponent();
        spawner = FindAnyObjectByType<CatSpawner>(); // Найти объект спаунера
    }

    Сообщение Unity | Ссылки: 0
    private void Update()
    {
        transform.Rotate(0, 0, rotationSpeed * Time.deltaTime);
    }

    Сообщение Unity | Ссылки: 0
    private void OnMouseDown()
    {
        if (spawner.isActiveAndEnabled) // выполняем проверку
        {
            audioSource.Play();
            Destroy(gameObject, audioSource.clip.length);
        }
    }
}
```

19. Теперь добавим простой эффект частиц при уничтожении котов.

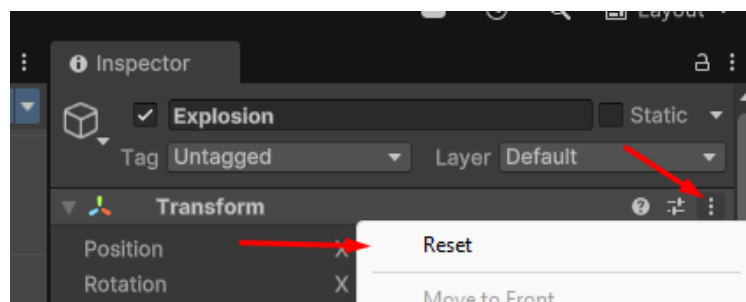
Создаём в **Hierarchy** систему частиц (**Effects – Particle System**)



Назовём её **Explosion**:



Сбросим трансформацию:



Duration (продолжительность) - **1**;

Looping (заикливание) уберём;

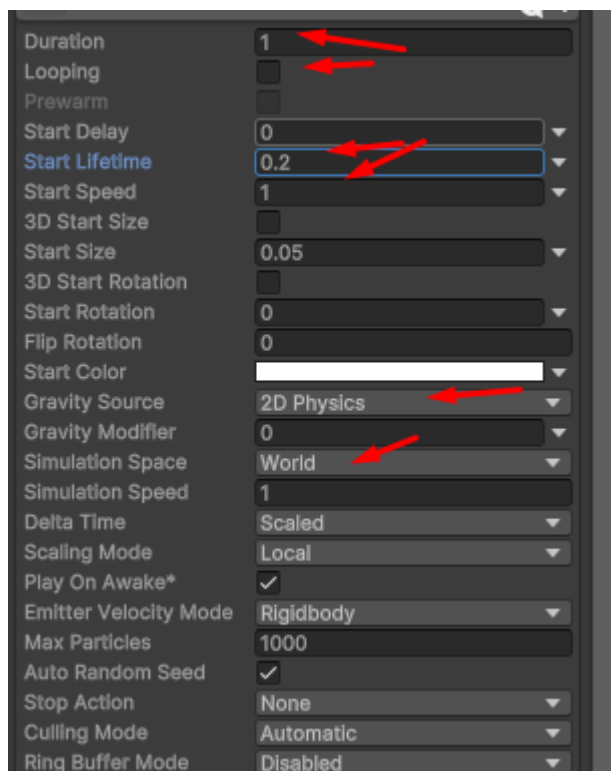
Start Lifetime (продолжительность жизни) – **0.2**;

Start Speed (начальная скорость) – **1**;

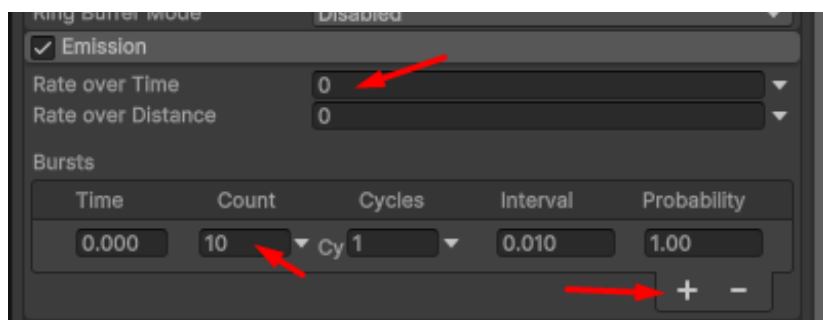
Start Size (начальный размер) – **0.05**;

Gravity Source (источник гравитации) – **2D Physics**;

Simulation Space (моделирование пространства) – **World**;



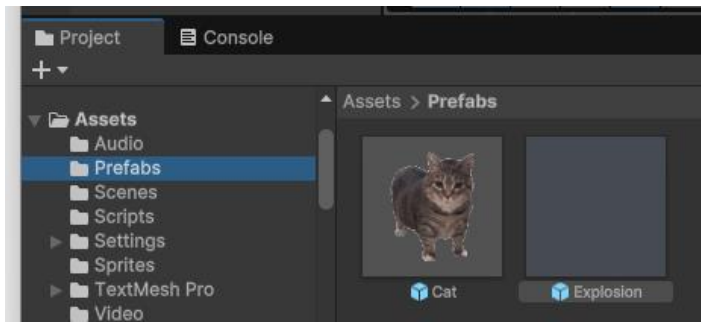
Раскроем меню **Emission** и установим **Rate over Time** на **0**, и добавим новый список, где установим количество наших частиц **10**:



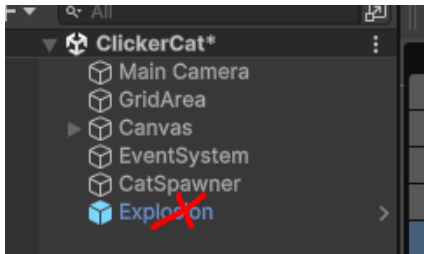
В меню **Shape** поменяем форму на **Circle** и установим минимальный радиус **0.0001**:



Перенесите систему частиц в папку с **префабами**:



И удалите из иерархии:



20. Обновим скрипт Cat:

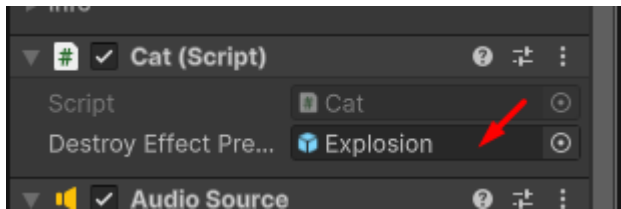
```
public class Cat : MonoBehaviour
{
    private CatSpawner spawner; // получаем ссылку на CatSpawner
    private AudioSource audioSource;
    private float rotationSpeed;
    public GameObject destroyEffectPrefab; // Ссылка на систему частиц

    [Unity] Сообщение Unity | Ссылки: 0
    private void Start()
    {
        audioSource = GetComponent<AudioSource>();
        spawner = FindAnyObjectByType<CatSpawner>(); // ищем объект спавна
        rotationSpeed = Random.Range(200f, 800f);
    }

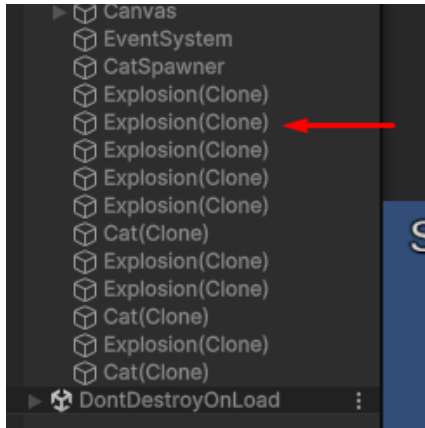
    [Unity] Сообщение Unity | Ссылки: 0
    private void Update()
    {
        transform.Rotate(0, 0, rotationSpeed * Time.deltaTime);
    }

    [Unity] Сообщение Unity | Ссылки: 0
    private void OnMouseDown()
    {
        if (spawner.isActive) // выполняем проверку
        {
            Instantiate(destroyEffectPrefab, transform.position,
                Quaternion.identity); // воспроизводим эффект
            audioSource.Play();
            Destroy(gameObject, audioSource.clip.length);
        }
    }
}
```

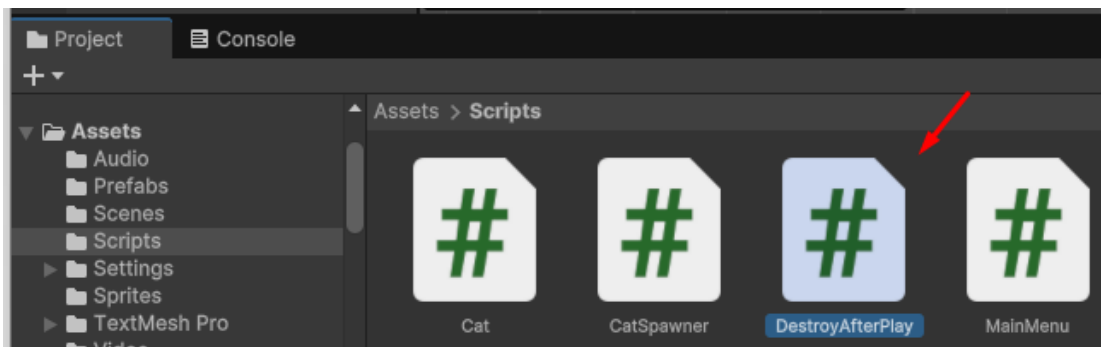
Затем добавим наш префаб системы частиц **Explosion** в скрипт **Cat** у префаба **Cat**:



При запуске игры мы можем обратить внимание, что при использовании **Instantiate**, у нас не уничтожается автоматически префаб после воспроизведения эффекта, они будут оставаться на сцене и накапливаться:



Чтобы устранить это, нужно добавить уничтожение системы частиц после завершения анимации. Добавляем новый скрипт **DestroyAfterPlay** в папке со скриптами:



Открываем его и напишем код:

```
using UnityEngine;

public class DestroyAfterPlay : MonoBehaviour
{
    private ParticleSystem _particleSystem; // Объявляем приватное поле для хранения ссылки на компонент ParticleSystem

    private void Start()
    {
```

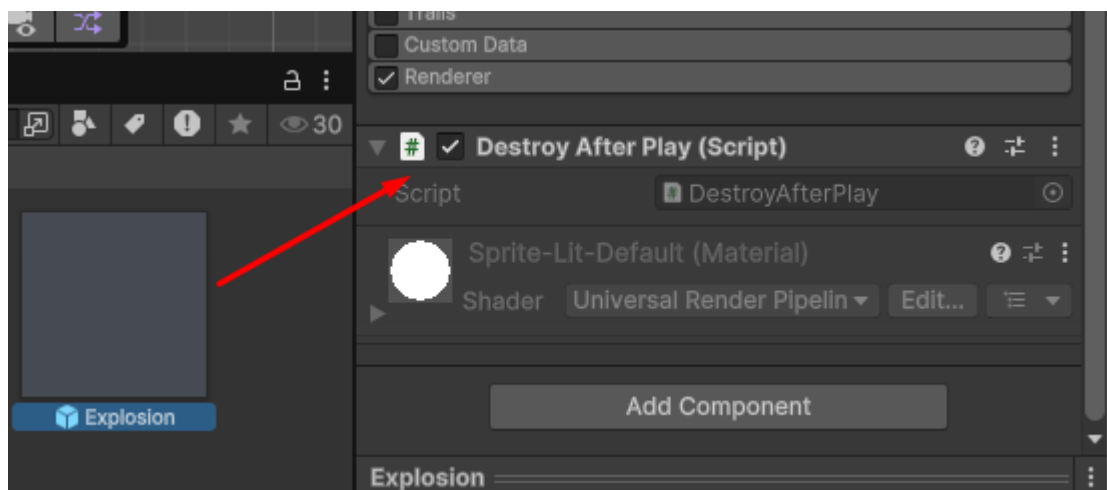
```

        _particleSystem = GetComponent<ParticleSystem>(); //
Инициализируем _particleSystem, находя компонент ParticleSystem на
том же объекте
    }

    private void Update()
    {
        if (GetComponent<ParticleSystem>() != null &&
!GetComponent<ParticleSystem>().IsActive()) // Проверяем, существует
ли компонент ParticleSystem и активен ли он
        {
            Destroy(gameObject); // если компонент ParticleSystem
не активен (IsActive() == false), уничтожаем объект
(Destroy(gameObject)).
        }
    }
}

```

Присоедините скрипт к префабу системы частиц:



Теперь, когда будет создаваться система частиц с помощью **Instantiate**, она автоматически уничтожится после завершения анимации.

21. Займёмся отображением подсчёта количества очков. Обновим скрипт **CatSpawner**, чтобы включить подсчет очков и создание UI. Добавляем в начало переменные:

```

private int score = 0; // начальное количество очков
public TextMeshProUGUI scoreText; // Ссылка на текст для
отображения счета
public TextMeshProUGUI timerText; // Ссылка на текст для
отображения таймера

```

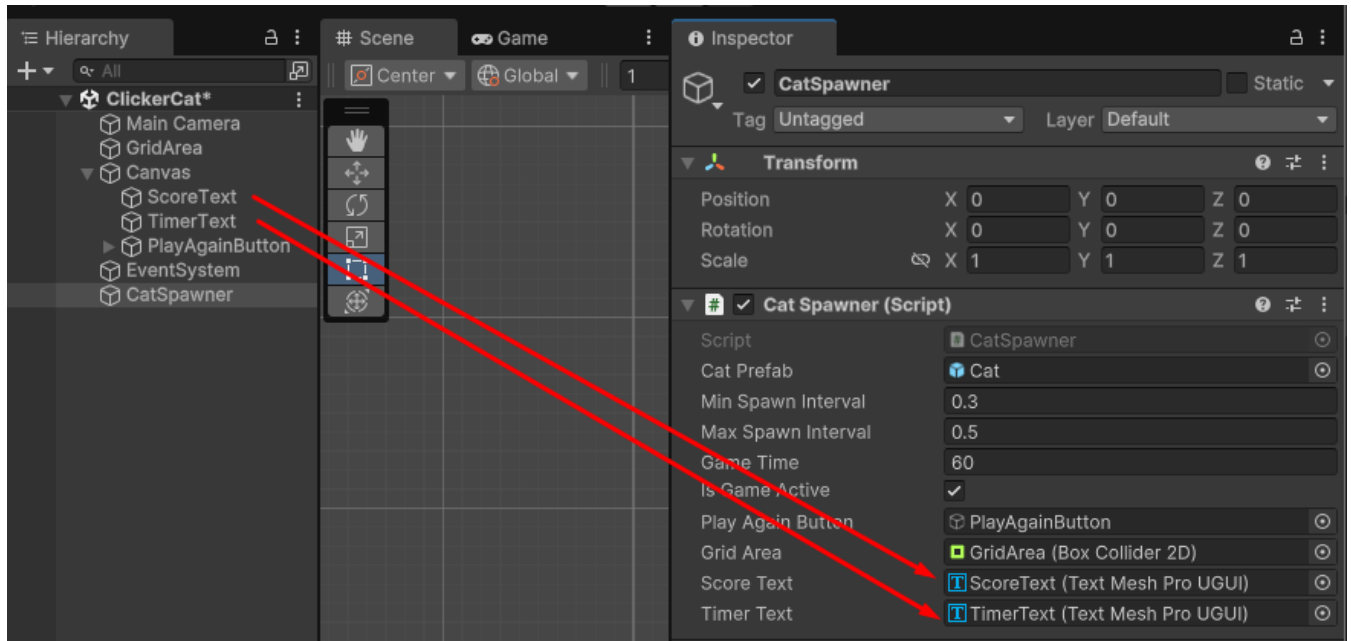
В начале скрипта у вас должно по умолчанию подключиться пространство имён **TMPPro**:

```

1  using UnityEngine;
2  using TMPro;
3

```

В Unity переносим соответствующие поля **ScoreText** и **TimerText** для объекта **CatSpawner**:



Напишем Метод для обновления текста счёта **UpdateScoreText()**:

```

private void UpdateScoreText() // Метод для обновления текста
счёта
{
    scoreText.text = $"Score: {score}"; // устанавливаем текст
счёта
}

```

Затем в методе **Start()** вызовем его:

```

private void Start()
{
    playAgainButton.SetActive(false);
    UpdateScoreText(); // Вызываем метод
}

```

Теперь напомним метод для обновления текста таймера **UpdateTimerText()**:

```

private void UpdateTimerText() // Метод для обновления текста
таймера
{
    timerText.text = $"Time:
{Mathf.CeilToInt(gameTime).ToString()}"; // устанавливаем текст
таймера
}

```

Mathf.CeilToInt возвращает наименьшее целое число, большее или равное f

И вызовем его в методе **Update()**:

```
private void Update()
{
    if (gameTime > 0)
    {
        gameTime -= Time.deltaTime;
        UpdateTimerText(); // Вызываем метод
        if (Time.time >= nextSpawnTime)
        {
            SpawnCat();
            nextSpawnTime = Time.time + Random
        }
    }
}
```

Далее напомним метод **AddScore()** для добавления очков к текущему счёту:

```
public void AddScore(int points) // Метод для добавления очков
к текущему счёту
{
    score += points; // увеличиваем счёт
    UpdateScoreText(); // вызываем метод обновления счёта
    текста
}
```

Затем в скрипте **Cat** вызовем его:

```
private void OnMouseDown()
{
    if (spawner.isActive)
    {
        Instantiate(destroyEffectPrefab, transform.position, Quaternion.identity);
        audioSource.Play();
        Destroy(gameObject, audioSource.clip.length);
        spawner.AddScore(1); // Добавляем очки при уничтожении
    }
}
```

Возвращаемся в скрипт **CatSpawner**, и напомним метод **PlayAgain()** который будет по нажатию кнопки перезапускать игру:

```
public void PlayAgain() // Метод для повторной игры
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    // Перезагружаем текущую сцену
}
```

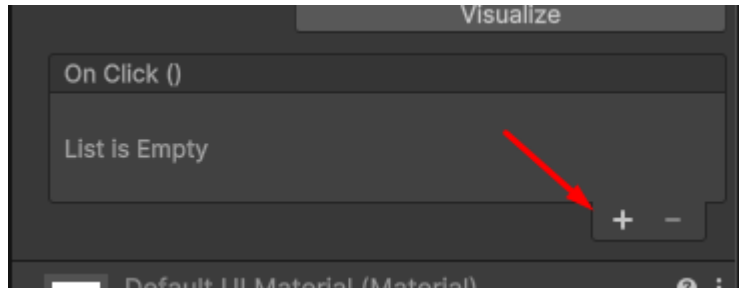
Обратите внимание, что у вас должно было подключиться в начале скрипта пространство имён **UnityEngine.SceneManagement**:

```

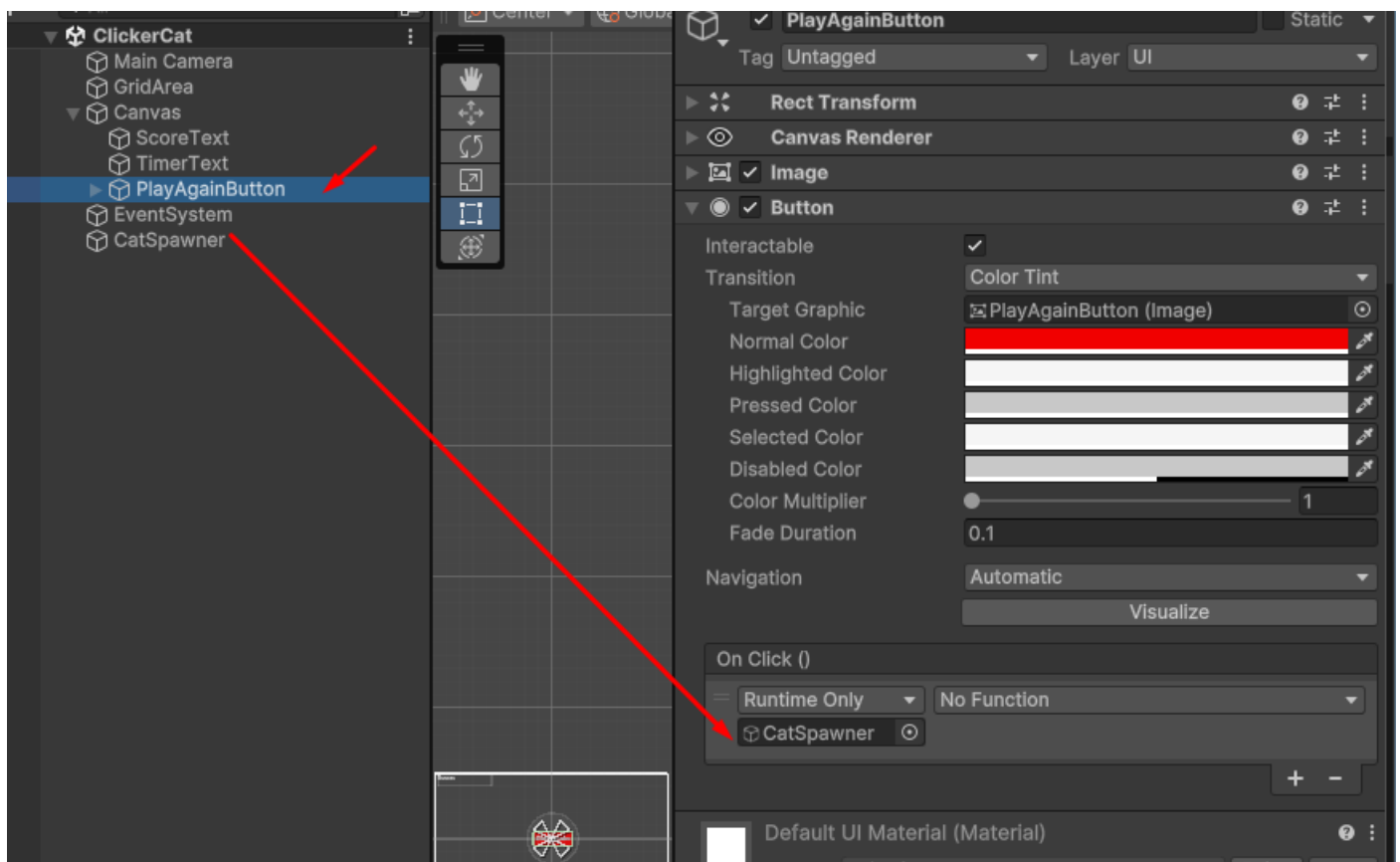
1  using UnityEngine;
2  using TMPro;
3  using UnityEngine.SceneManagement;
4

```

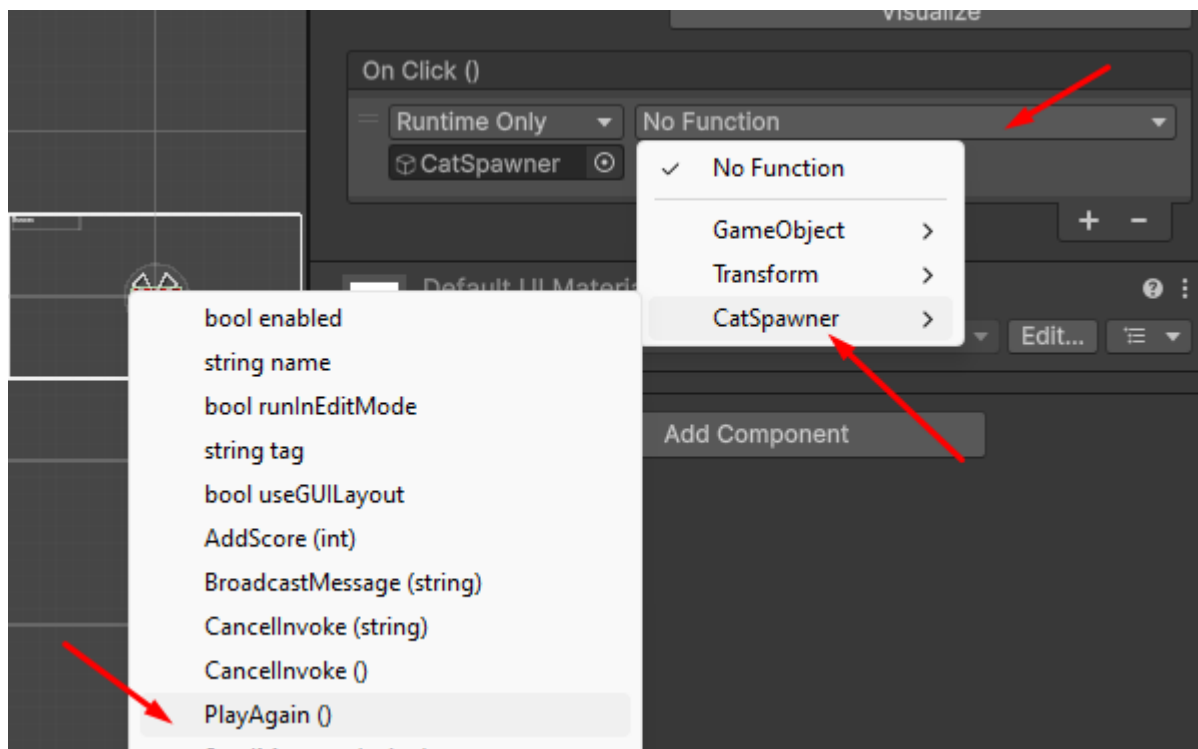
Теперь нам нужно в Unity для нашей кнопки добавить это событие. Находим у кнопки **On Click()** и нажимаем +:



Переносим объект **CatSpawner**:

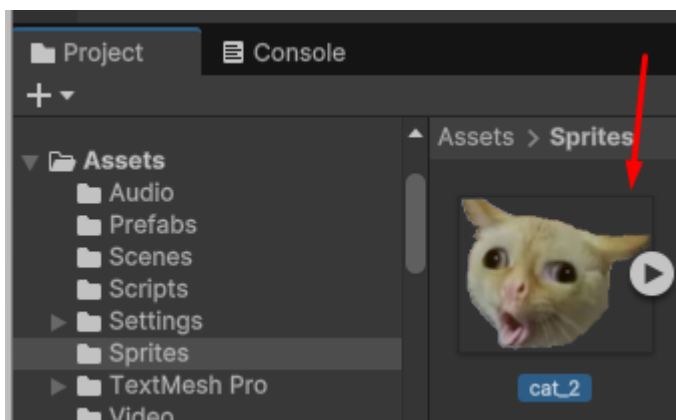


После нажимаем на функциях, выбираем **CatSpawner – PlayAgain()**:

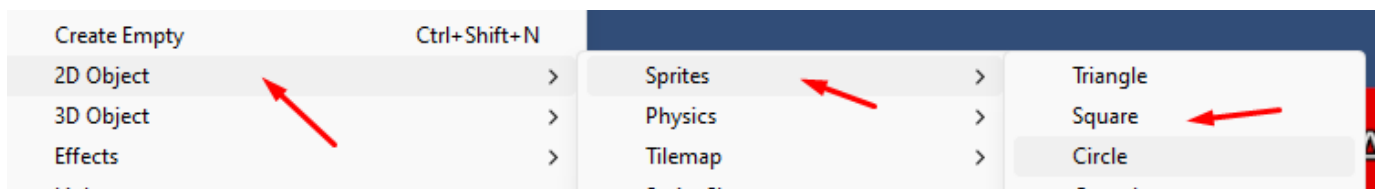


22. Чтобы сделать нашу игру немного интереснее давайте добавим появление редкого кота, за которого будет начисляться больше очков.

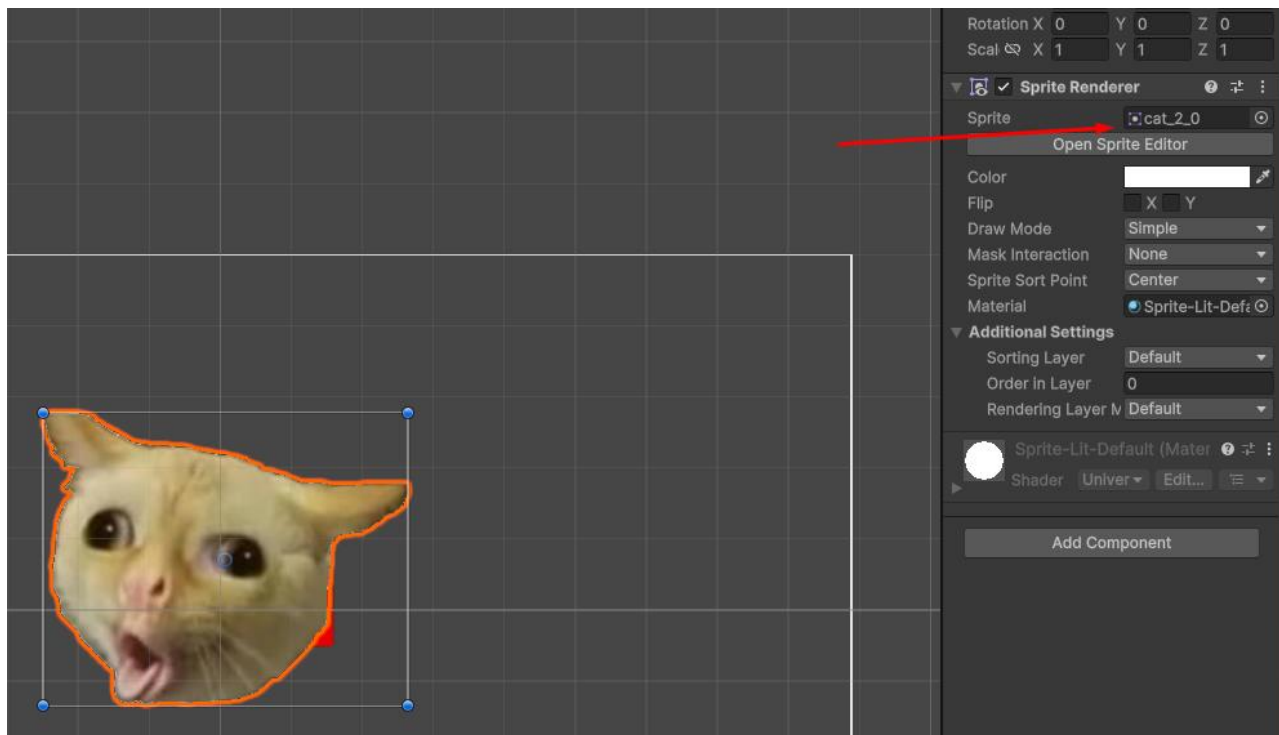
Для этого добавьте из папки с ассетами нового кота **cat_2** в папку **Sprites**:



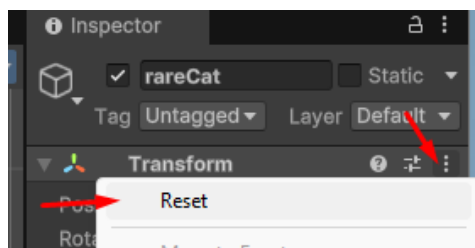
Создадим для нашего изображения основу, выберем **2D Object – Sprites – Circle**:



Назовём его **rareCat**. Затем из папки **Sprites** перенесём нашего кота в поле **Sprites**:



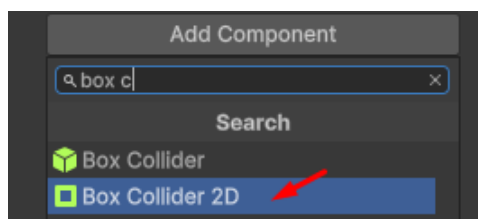
Сбросим позицию:



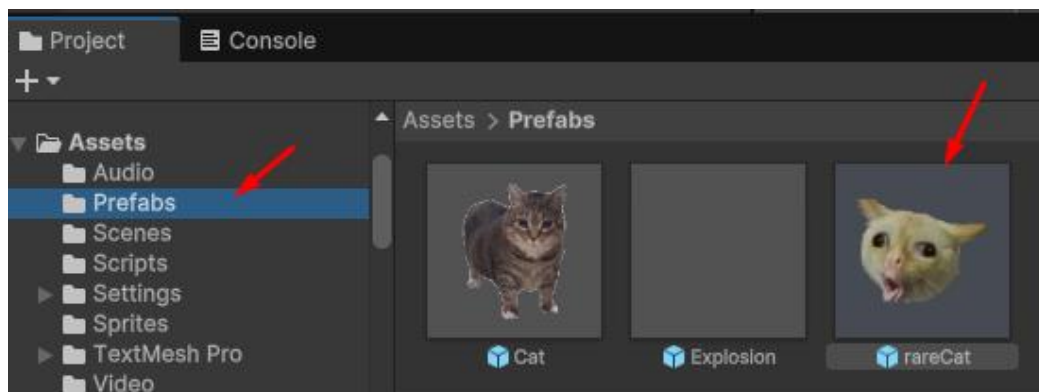
Уменьшим размер до **0.15** по x,y:



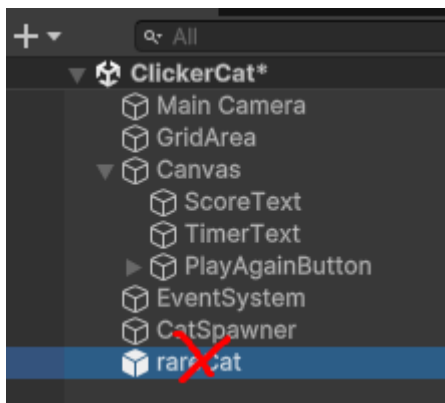
Добавим **Box Collider 2D**:



Далее переносим наш объект в папку с **префабами**:



Удаляем сам объект из иерархии:



23. Открываем скрипт **CatSpawner** и добавляем переменные для редкого кота и шанса его появления.

```
public GameObject rareCatPrefab; // Префаб редкого кота
public float rareCatChance = 0.1f; // Шанс появления редкого кота (10%)
```

В методе **SpawnCat**, добавим логику появления редкого кота:

```
private void SpawnCat()
{
    Bounds bounds = GridArea.bounds;
    float x = Random.Range(bounds.min.x, bounds.max.x);
    float y = Random.Range(bounds.min.y, bounds.max.y);
    Vector3 spawnPosition = new Vector3(x, y, 0);

    GameObject cat;
    if (Random.value <= rareCatChance)
        cat = Instantiate(rareCatPrefab, spawnPosition, Quaternion.identity);
    else
        cat = Instantiate(catPrefab, spawnPosition, Quaternion.identity);
}
```

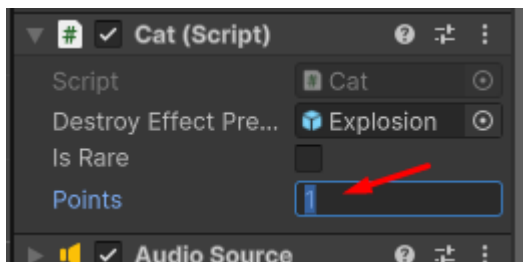
Открываем скрипт **Cat** и добавляем переменную для определения редкости кота и количество очков за его уничтожение:

```
public bool isRare = false; // Флаг для редкого кота
public int points = 5; // Количество очков за уничтожение
```

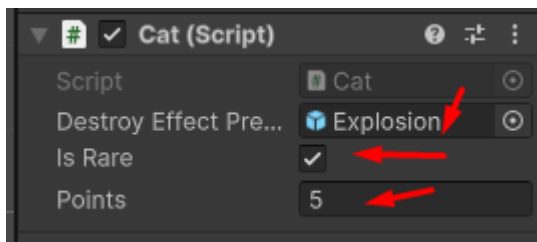
Изменяем метод **OnMouseDown**, чтобы добавлять разные очки в зависимости от редкости кота:

```
private void OnMouseDown()
{
    if (spawner.isGameActive)
    {
        Instantiate(destroyEffectPrefab, transform.position, Quaternion.identity);
        audioSource.Play();
        Destroy(gameObject, audioSource.clip.length);
        spawner.AddScore(points); // Добавляем очки в зависимости от редкости кота
    }
}
```

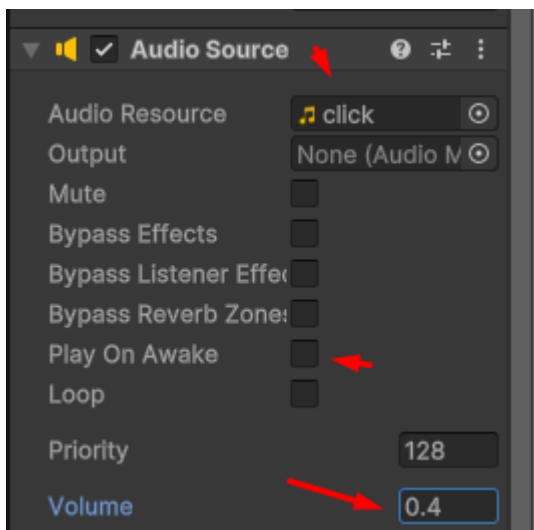
Теперь в Unity для префаба **Cat** в скрипте **Cat** ставим **1** очко:



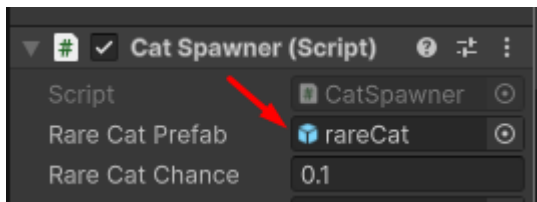
Для префаба **rareCat** добавляем скрипт **Cat**, ставим **5** очков и галочку **Is Rare**, и наш префаб **Explosion**:



Также добавляем ему звук клика (убираем галочку на **Play on Awake**, и меняем громкость на **0.4**):



Для объекта **CatSpawner**, в скрипте **CatSpawner**, добавляем префаб нашего кота в соответствующее поле:



Ну и чтобы было интереснее, сделаем так, чтобы кот исчезал через 1 секунду после появления. Для этого добавляем в начало скрипта **Cat** переменную его жизни:

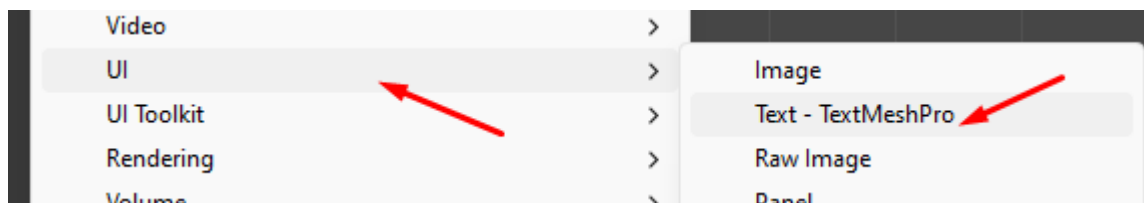
```
public float lifeTime = 1f; // Время жизни кота
```

И добавляем в методе **Start** уничтожения нашего редкого кота по истечению времени:

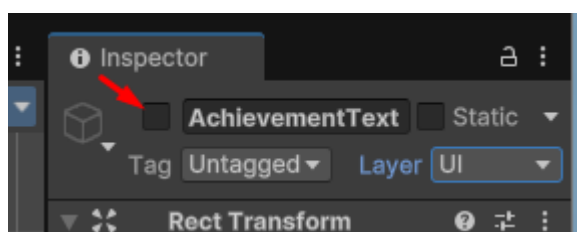
```
private void Start()
{
    audioSource = GetComponent();
    spawner = FindAnyObjectByType<CatSpawner>();
    rotationSpeed = Random.Range(200f, 800f);

    if (isRare) // применяем только для редкого кота
    {
        Destroy(gameObject, lifeTime); // Уничтожаем кота через
        определенное время
    }
}
```

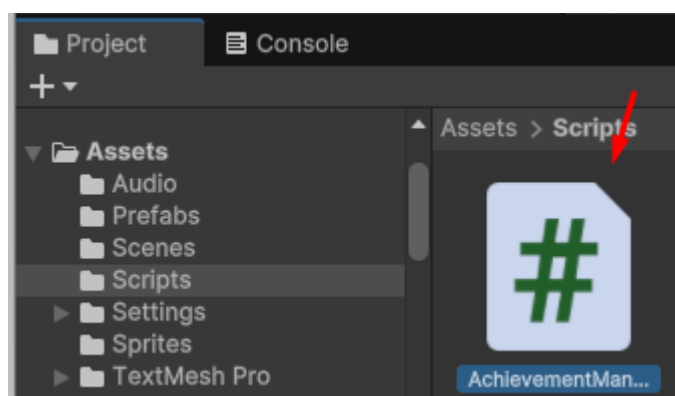
24. Ну и напоследок давайте с вами реализуем систему **Ачивок**, которые будут появляться в углу экрана. В нашем **Canvas** создаём новый текст **UI – Text – TextMeshPro**:



Называем его **AchievementText**, настраиваем размер, шрифт, цвет и т.д. Переносим в место на экране, где вы хотите, чтобы он появлялся. Также снимите галочку с его отображения:



Далее создаём скрипт **AchievementManager** в папке со скриптами:



Открываем его и пропишем:

```
using UnityEngine;
using System.Collections;
using TMPro;

public class AchievementManager : MonoBehaviour
{
    public TextMeshProUGUI achievementText;
    public float displayTime = 3f;

    // Логические переменные для отслеживания состояния ачивок
    private bool caughtRareCatAchieved = false;
    private bool collected20CatsAchieved = false;
    private bool collected50CatsAchieved = false;
    private bool catMasterAchieved = false;

    // Проверка и разблокировка достижений
    public void CheckAchievements(int score, bool
caughtRareCat)
    {
        if (caughtRareCat && !caughtRareCatAchieved)
        {
            caughtRareCatAchieved = true;
            UnlockAchievement("Поймай редкого кота!");
        }

        if (score >= 20 && !collected20CatsAchieved)
        {
            collected20CatsAchieved = true;
            UnlockAchievement("Собрал 20 котов!");
        }

        if (score >= 50 && !collected50CatsAchieved)
        {
            collected50CatsAchieved = true;
            UnlockAchievement("50 котов!");
        }

        if (score >= 100 && !catMasterAchieved)
        {
            catMasterAchieved = true;
            UnlockAchievement("Мастер котов!");
        }
    }
}
```

```

}

// Разблокировка достижения
private void UnlockAchievement(string achievement)
{
    StartCoroutine(DisplayAchievement(achievement));
}

// Отображение достижения на экране
private IEnumerator DisplayAchievement(string achievement)
{
    achievementText.text = "Achievement Unlocked: " +
achievement;
    achievementText.gameObject.SetActive(true);
    yield return new WaitForSeconds(displayTime);
    achievementText.gameObject.SetActive(false);
}
}

```

Разбор кода:

Переменные

1. public TextMeshProUGUI achievementText;

- Это ссылка на текстовый элемент, который будет отображать сообщение о достижении.

2. public float displayTime = 3f;

- Время, в течение которого достижение будет отображаться на экране.

3. Логические переменные

Эти переменные используются для отслеживания состояния каждого достижения:

4. Метод CheckAchievements

Этот метод проверяет текущий счет и флаг **caughtRareCat**, чтобы определить, разблокированы ли новые достижения. Метод принимает два параметра: текущий счет (score) и флаг, пойман ли редкий кот (caughtRareCat).

Если достижение еще не разблокировано, оно будет разблокировано и отображено на экране.

5. Метод UnlockAchievement

Этот метод запускает корутину для отображения достижения.

- Метод принимает строку с названием достижения.
- Запускает корутину DisplayAchievement.

6. Метод DisplayAchievement (Коррутина)

Коррутина отвечает за отображение достижения на экране в течение заданного времени:

- **IEnumerator** указывает, что метод является корутиной.
- Метод принимает строку с названием достижения.
- **achievementText.text** обновляется с сообщением о достижении.

- `achievementText.gameObject.SetActive(true);` активирует текстовый элемент, чтобы он стал видимым.
- `yield return new WaitForSeconds(displayTime);` заставляет корутину ждать заданное время (`displayTime`), прежде чем продолжить выполнение.
- После ожидания, текстовый элемент снова становится неактивным (`achievementText.gameObject.SetActive(false);`).

Что такое корутина?

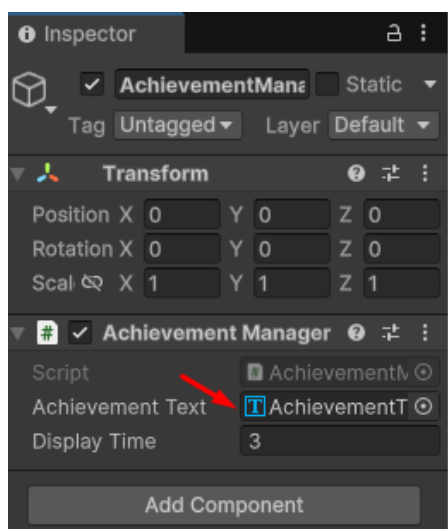
Коррутина (Coroutine) — это функция в Unity, которая позволяет вам приостановить выполнение на определенное время или до определенного события, не блокируя основной поток. Это полезно для выполнения длительных или повторяющихся действий.

Основные этапы работы корутины:

- 1. Запуск корутины:** `StartCoroutine` используется для запуска корутины.
- 2. Приостановка выполнения:** `yield return` используется для приостановки выполнения на определенное время или до выполнения определенного условия.
- 3. Продолжение выполнения:** После того как условие выполняется, выполнение корутины продолжается с того места, где оно было приостановлено.

Коррутины позволяют выполнять асинхронные операции, такие как ожидание времени или загрузка данных, без блокировки основного потока выполнения.

25. Далее создадим в Unity новый пустой объект и назовём его **AchievementManager**, назначим ему наш скрипт **AchievementManager**, в поле с текстом не забудем перенести наш текст:



В скрипте **CatSpawner** внесём правки. Создадим ссылку на наши ачивки:

```
public AchievementManager achievementManager; // Ссылка на AchievementManager
```

Немного изменим метод **AddScore**:

```

public void AddScore(int points, bool caughtRareCat) //
    добавляем параметры в вызов метода
{
    score += points;
    UpdateScoreText();
    if (achievementManager != null)
    {
        achievementManager.CheckAchievements(score,
            caughtRareCat); // Проверяем достижения
    }
}

```

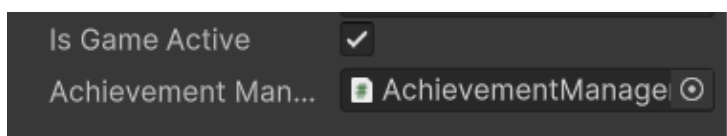
В скрипте **Cat** в методе **OnMouseDown** внесём изменения:

```

private void OnMouseDown()
{
    if (spawner.isGameActive)
    {
        Instantiate(destroyEffectPrefab, transform.position,
            Quaternion.identity);
        audioSource.Play();
        Destroy(gameObject, audioSource.clip.length);
        spawner.AddScore(points, isRare); // Передаем флаг
        редкого кота
    }
}

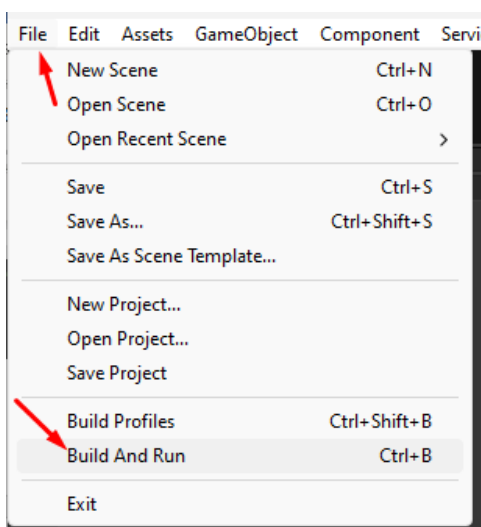
```

Добавляем в скрипт **CatSpawner** объект **AchievementManager**:



26. Осталось только скомпилировать нашу игру. Переходим в **File – Build And**

Run:



Выбираете любую папку куда хотите сохранить игру, или создаёте новую папку.

После можете запустить игру через **.exe**-файл.