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

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.

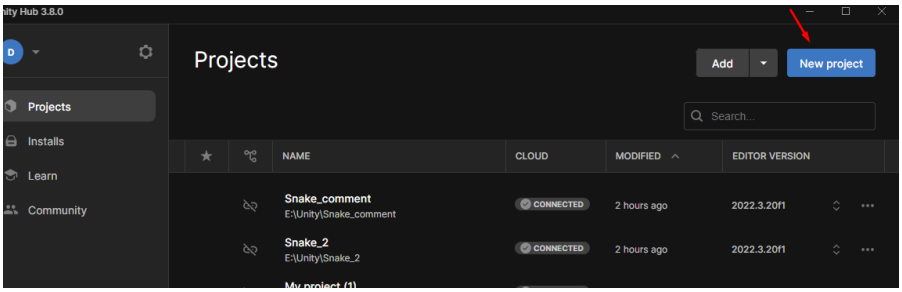
Задачи:

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 появление 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.

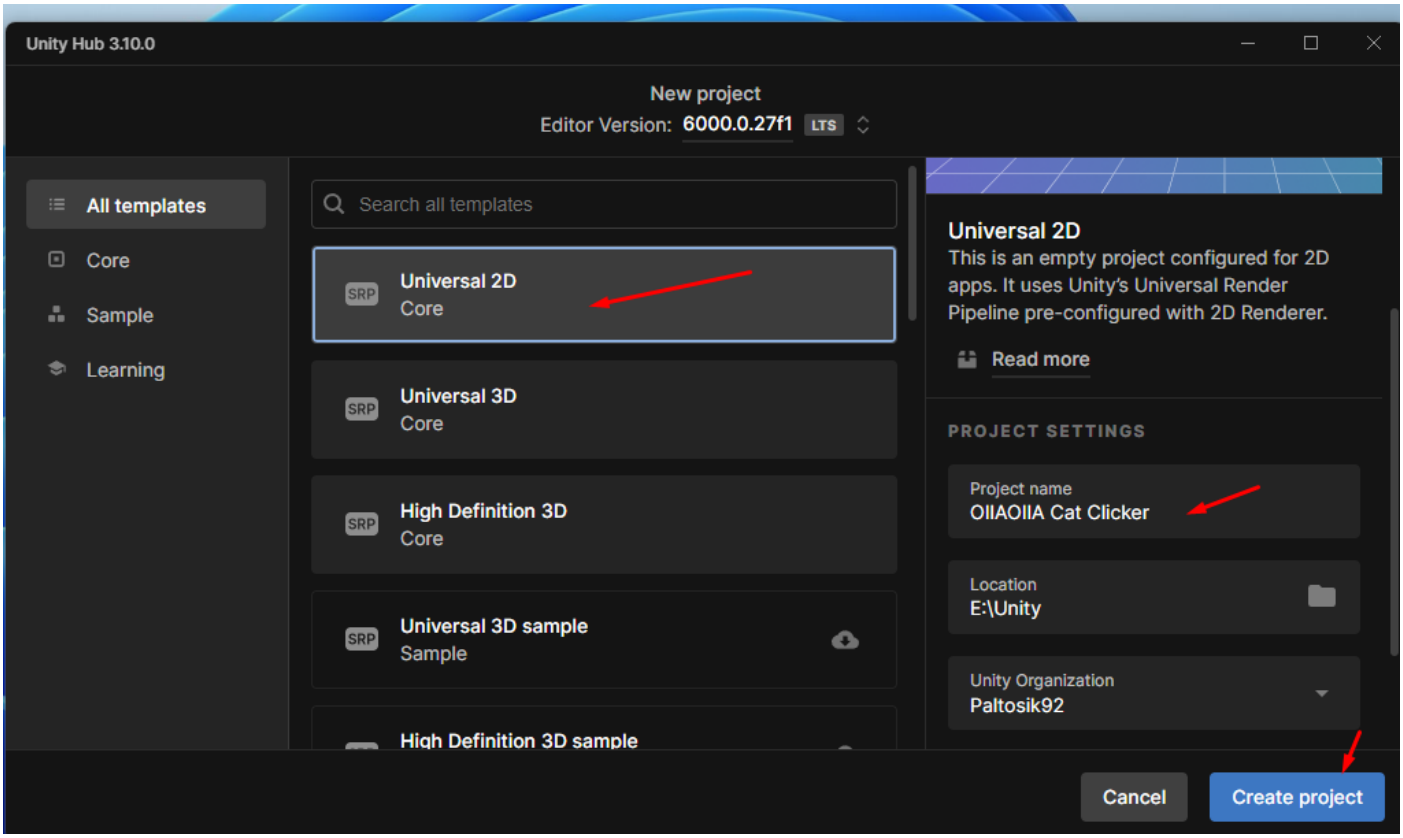
Ход работы:

1. Запускаем **Unity Hub**.

2. Создаём новый проект – **New project**:

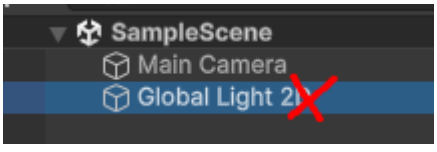


3. Выбираем **2D**. Вводим название проекта **OIIAOIIA Cat Clicker**, выбираем место расположения, и нажимаем **Create project**.

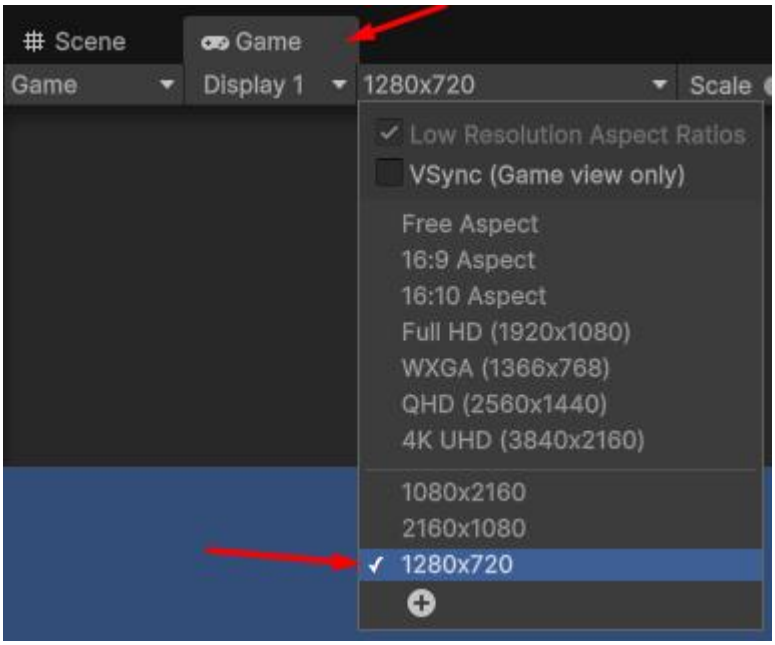


4. В папке **Scenes** меняем название сцены на **ClickerCat**:

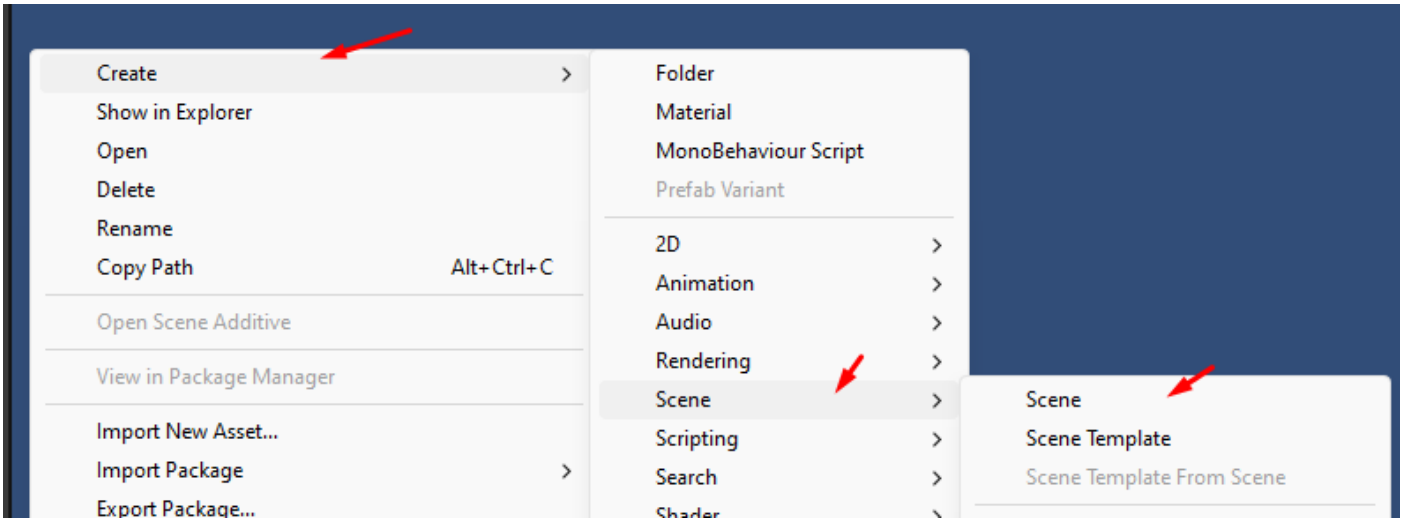
Удаляем объект **Global Light 2D,** он нам не понадобится:

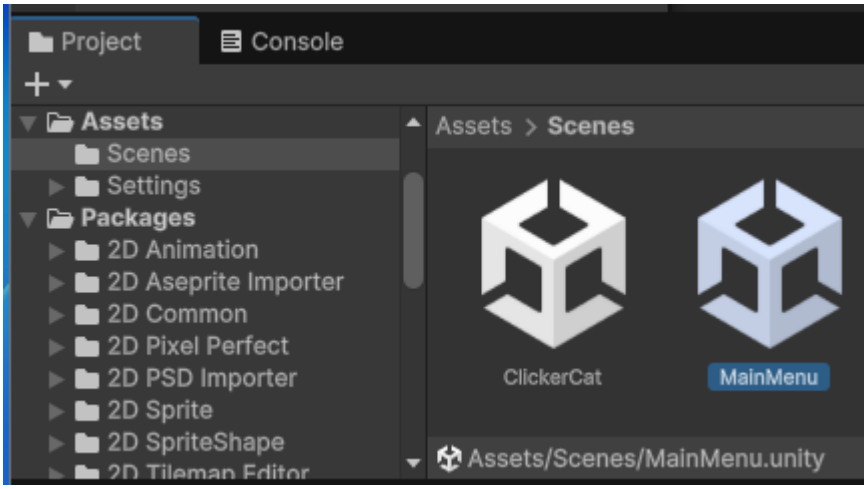


Во вкладке **Game** поменяем разрешение для нашей игры, в качестве примера выставим **1280x720**. Для этого нужно нажать на + и ввести вручную разрешение, после оно появится у вас в выборе:

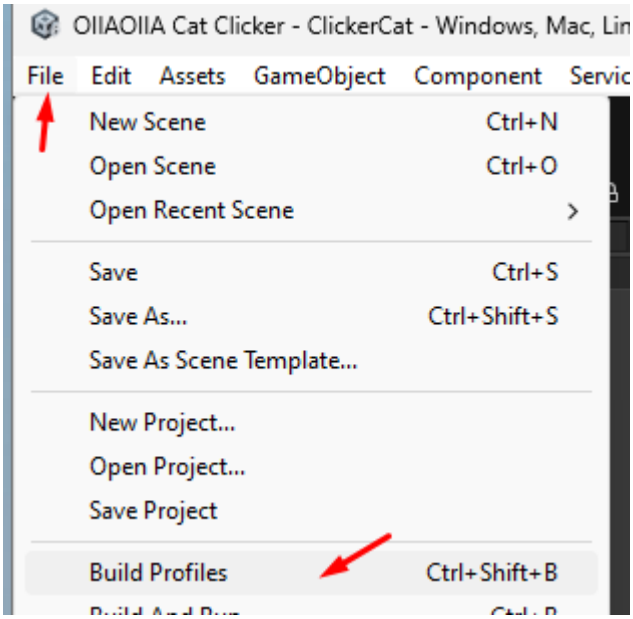


5. Добавим ещё одну сцену, назовём её **MainMenu:**

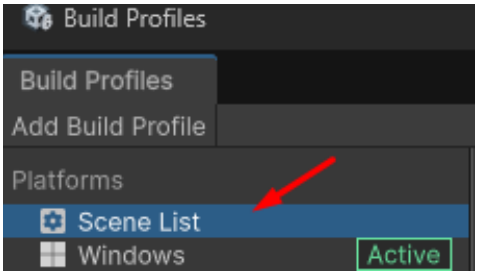




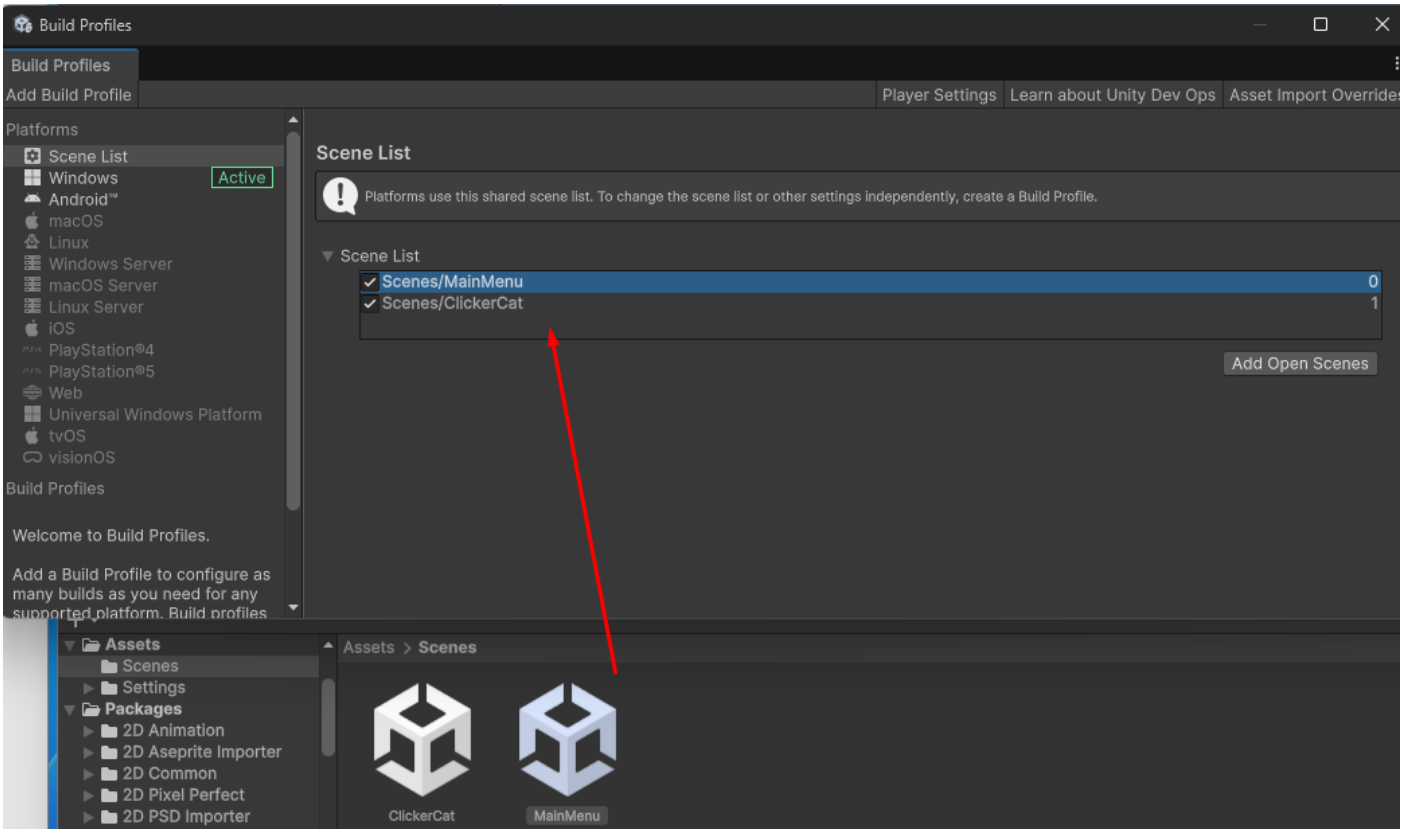
Теперь нам нужно подключить нашу сцену в проект. Для этого перейдите в **Build Profiles**:



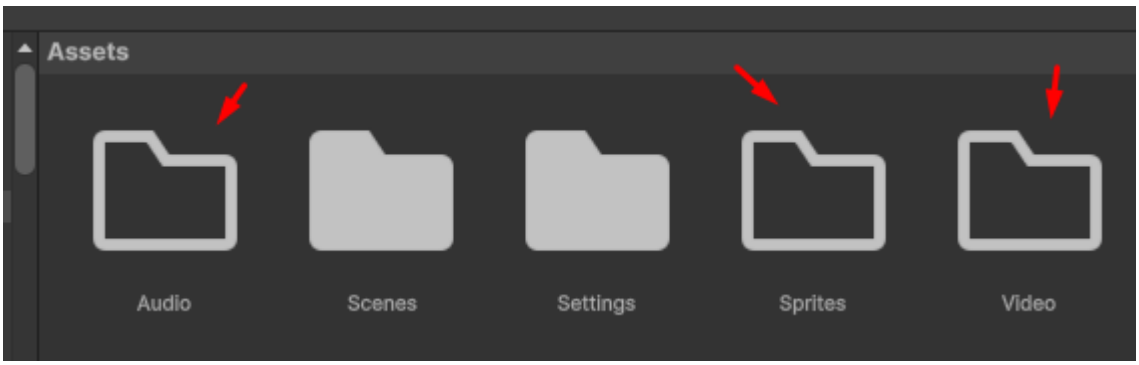
Откройте **Scene List:**



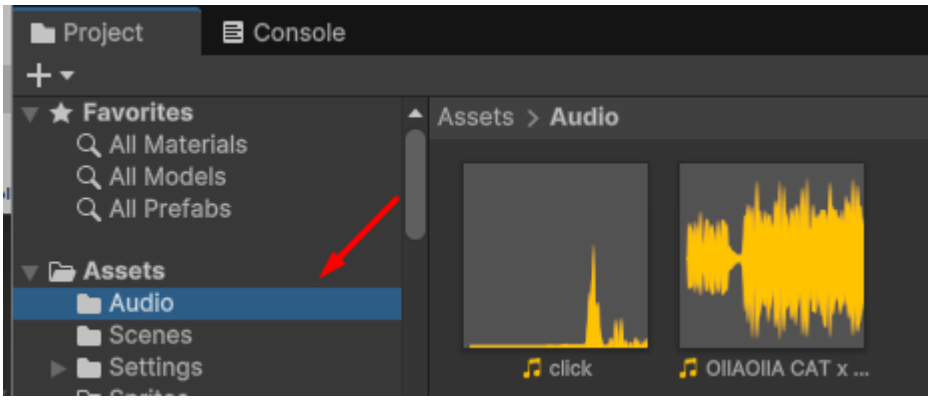
Перетащите и настройте правильный порядок ваших сцен:



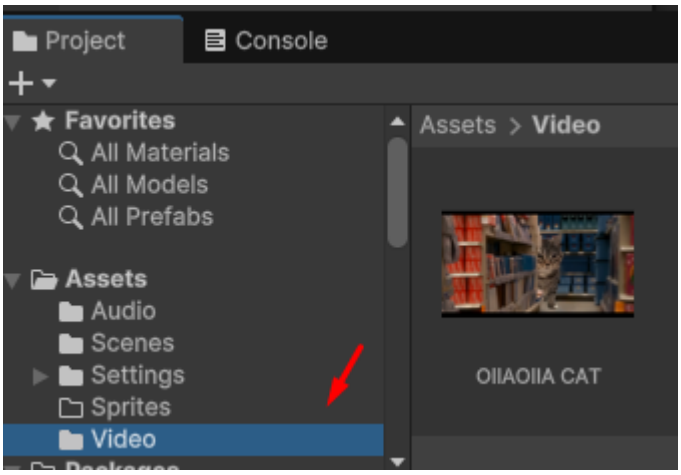
6. Создаём в нашем проекте папки **Video, Audio, Sprites:**



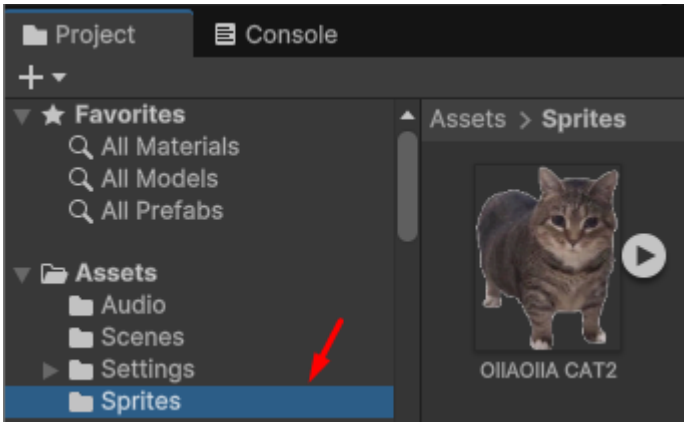
Из папки к лабораторной с ассетами переносим в папку **Audio два .mp3** звука:



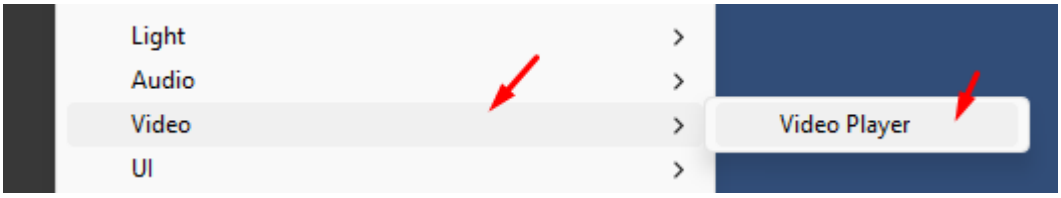
В папку **Video** перенесём наше фоновое видео:



В папку **Sprites** перенесём наше изображение:

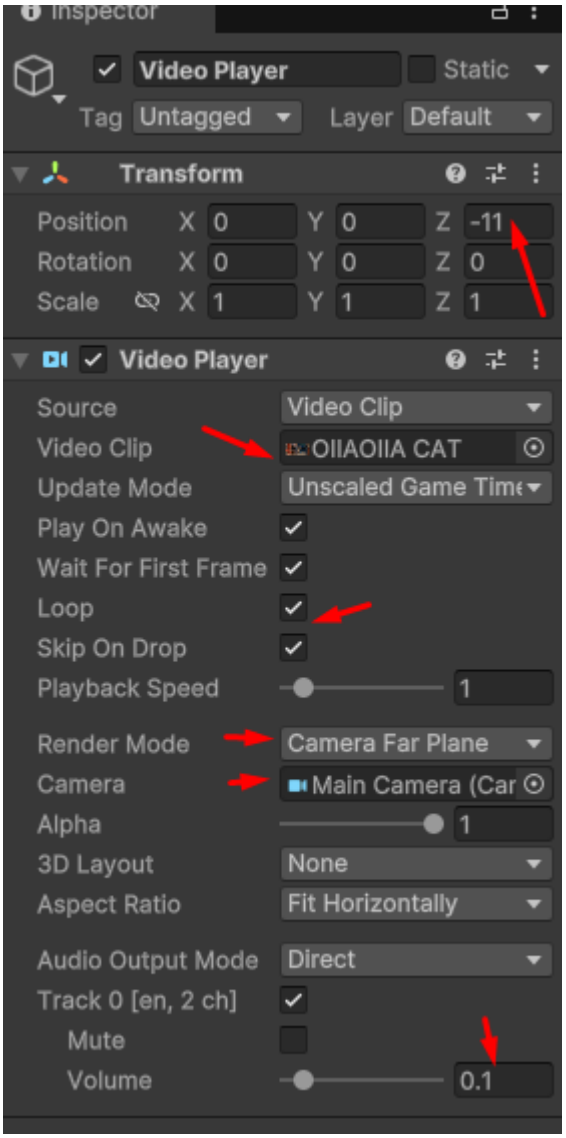


7. Перейдём к сцене **MainMenu.** Добавим фоновое видео. Для этого щёлкаем в нашей иерархии правой кнопкой мыши и выбираем **Video-Video Player:**



Произведём настройки:

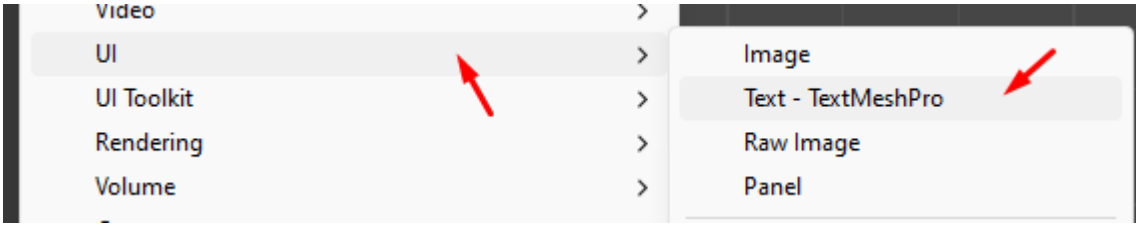
- сместим его по позиции **Z** на **-11**, чтобы он отображался на переднем плане в будущем корректно;
- в **Video Clip** переносим наш файл с видео из папки **Video**;
- ставим галочку на **Loop**, чтобы зацикливать видео;
- в **Render Mode** меняем на **Camera Far Plane** и переносим нашу **камеру**;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



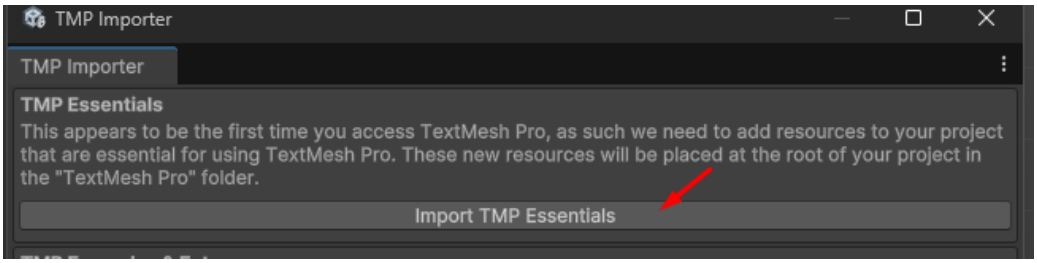
Теперь при запуске у вас будет идти фоновое видео.

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

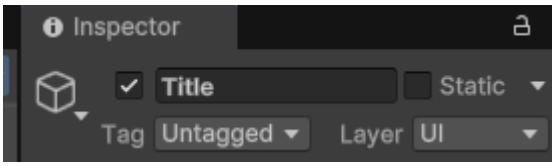
Щёлкаем правой кнопкой мыши – **UI – Text – TextMeshPro**:



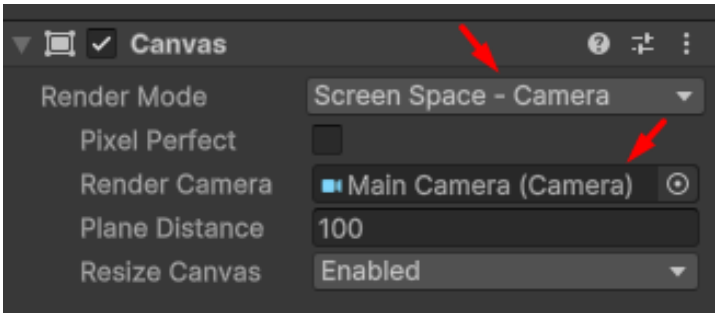
При появившемся окне нажимаем импортировать:



Меняем для текста название на **Title**:

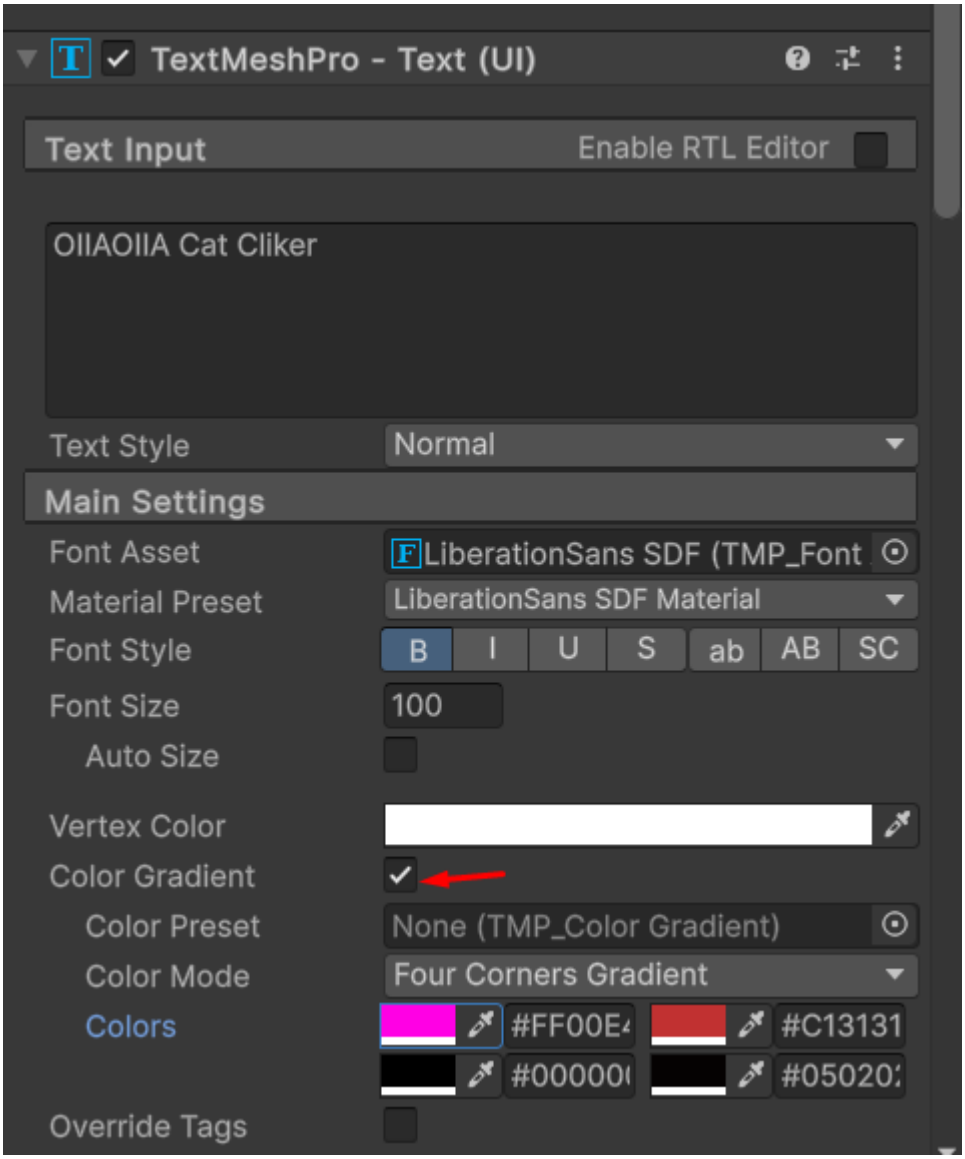


Для нашего Canvas меняем отображение на **Screen Space – Camera**, и добавляем нашу камеру:

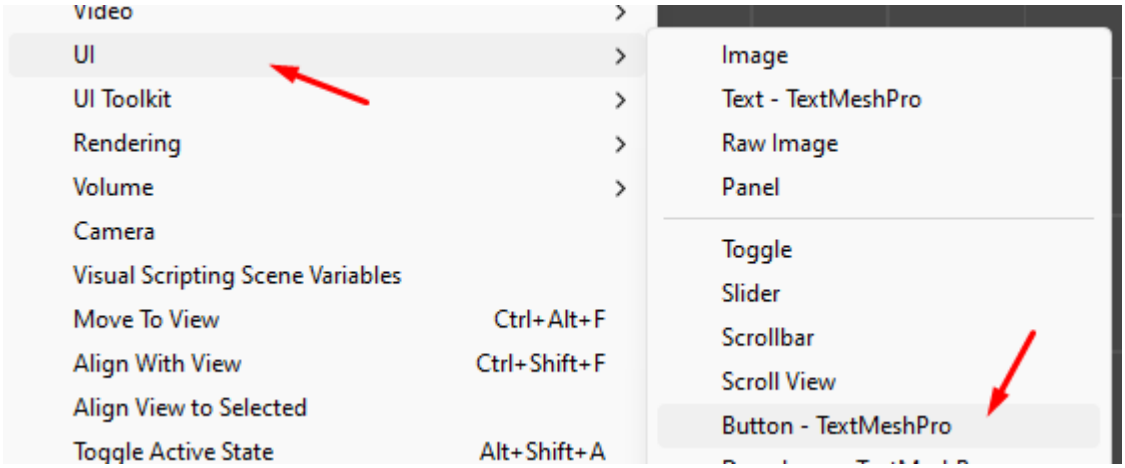


В поле текста пишем - **OIIAOIIA CAT Clicker.** Настройте его на свой вкус, пример:

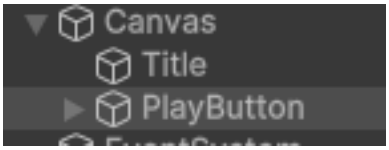
Для того чтобы сделать разноцветный текст, нужно поставить галочку в пункте **Color Gradient:**



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

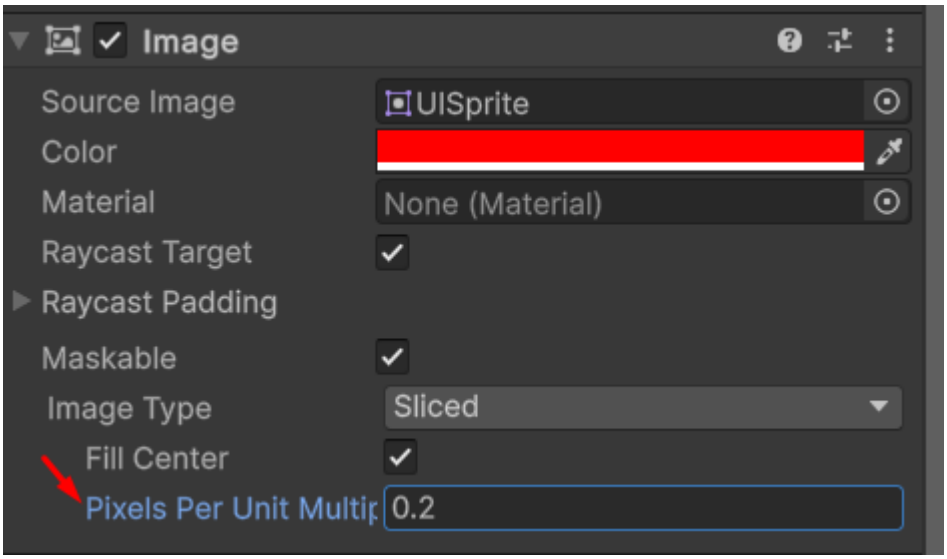


Называем её **PlayButton:**

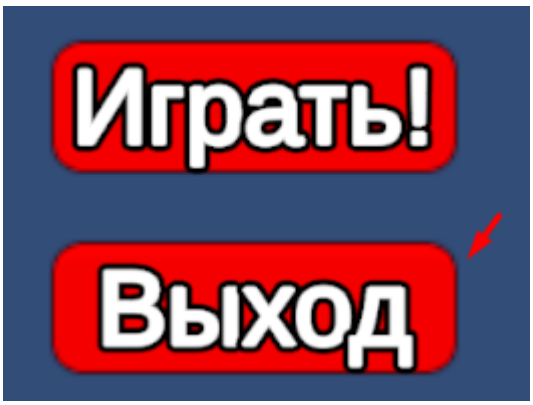


Меняем цвет текста и размер на своё усмотрение, пример:

Чтобы закруглить углы кнопки в Unity, можно настроить значение поля **Pixels Per Unit** (плотность пикселей) в настройках спрайта. По умолчанию одному пикселю на спрайте соответствует одна единица пользовательского интерфейса.

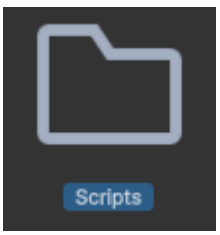


Создадим также кнопку для **выхода** из игры и назовём её **PlayExit**:

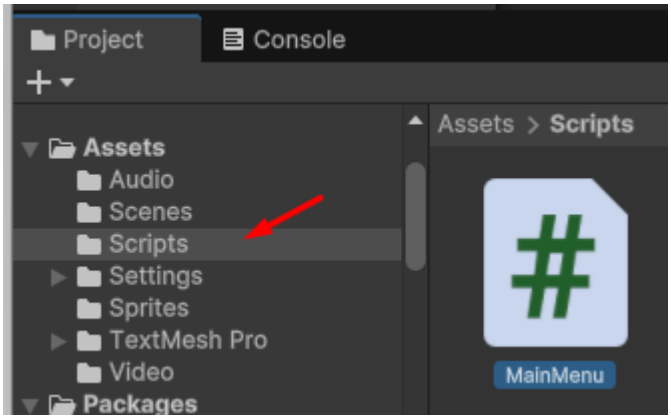


9. Теперь нам нужно создать скрипт для перехода на другую сцену и выхода из игры, и прикрепить его к объектам.

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.

Что такое директивы препроцессора?

Директивы препроцессора — это специальные команды, которые выполняются ещё до компиляции кода. Они позволяют изменять поведение программы в зависимости от условий.

1. ***#if UNITY_EDITOR** — проверяет, запущен ли код в редакторе Unity*
2. *Если игра запущена в редакторе, выполняется:*

UnityEditor.EditorApplication.isPlaying = false;

Это просто останавливает выполнение сцены в редакторе.

3. ***#else*** *— если игра запущена уже как готовое приложение (например, на ПК, телефоне и т. д.), выполняется:*

Application.Quit;

Этот метод закрывает игру.

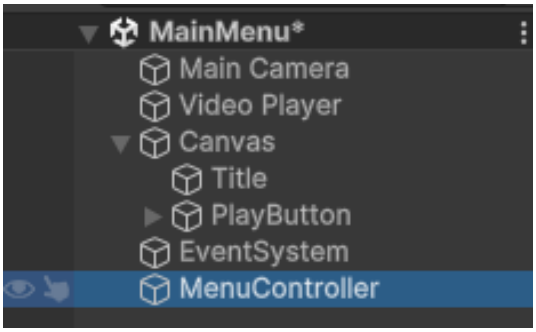
4. ***#endif*** *завершает блок условия.*

Зачем это нужно?

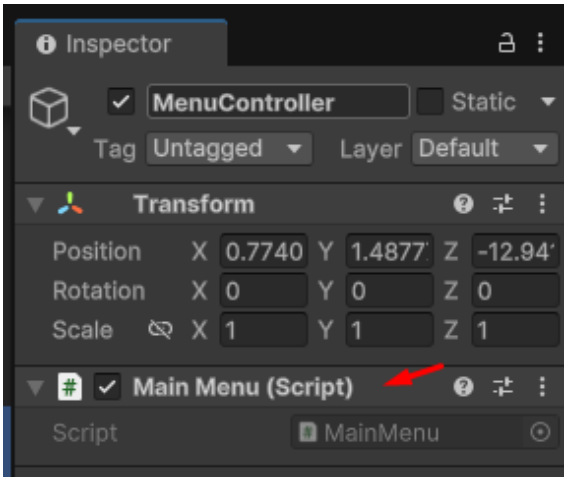
- *В* ***редакторе Unity*** *метод* ***Application.Quit();*** ***не работает****, потому что редактор сам не закрывается, когда ты нажимаешь «Выход».*
- *Поэтому* ***в редакторе*** *используется* ***UnityEditor.EditorApplication.isPlaying = false;****, чтобы просто остановить игру.*
- *А* ***в готовой сборке*** *(Application.Quit();) уже корректно закрывает приложение.*

Затем создаём пустой объект в иерархии:

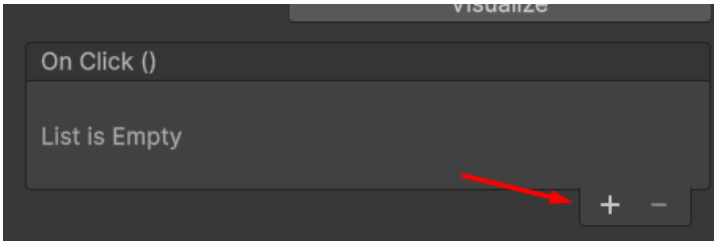
Назовём его **MenuController:**



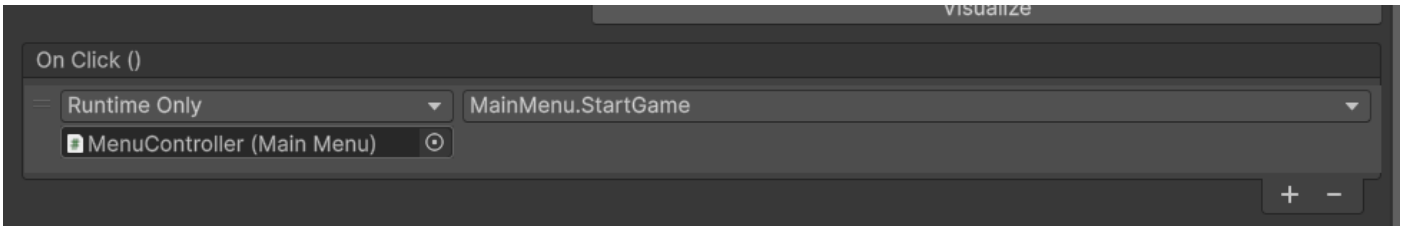
Добавляем к нему наш скрипт:



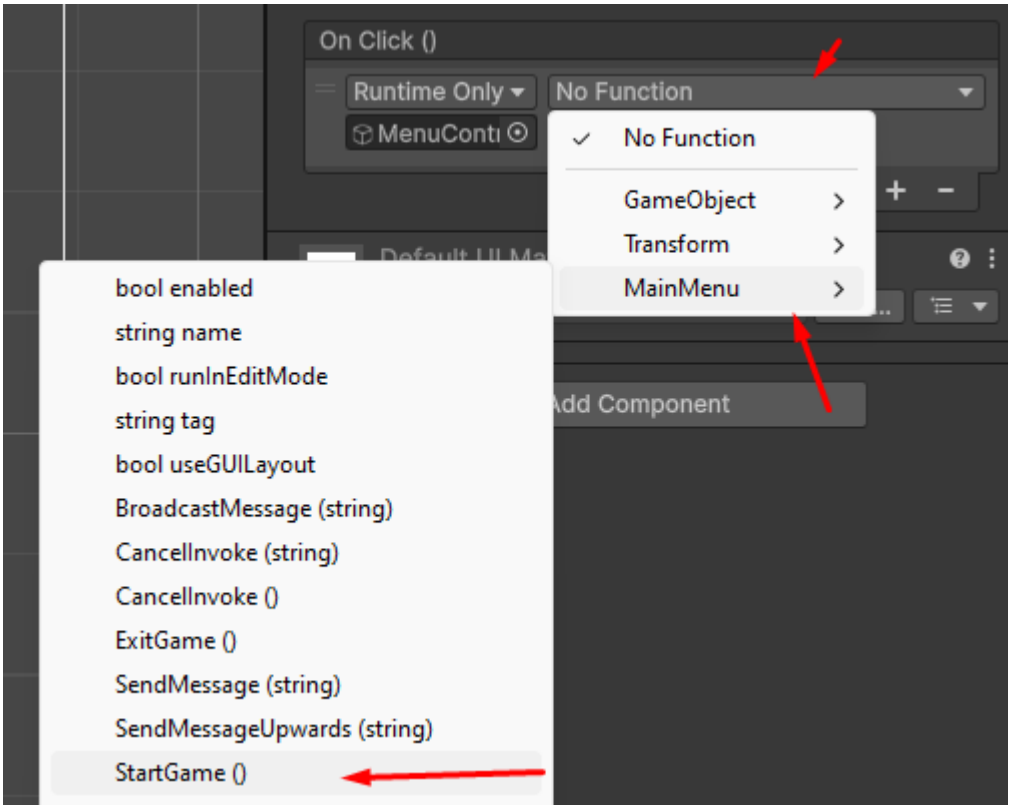
Далее нам нужно в нашу **кнопку PlayButton** добавить новое событие в **On Click()** нажав на +:



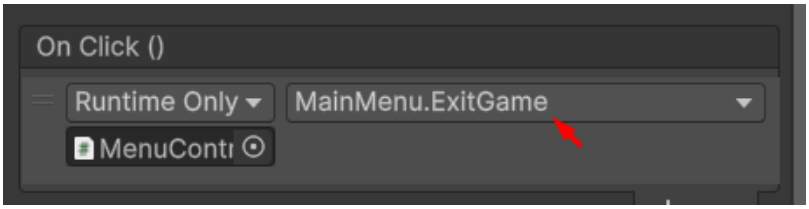
После нам нужно перенести наш объект **MenuController** в пустое поле:



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

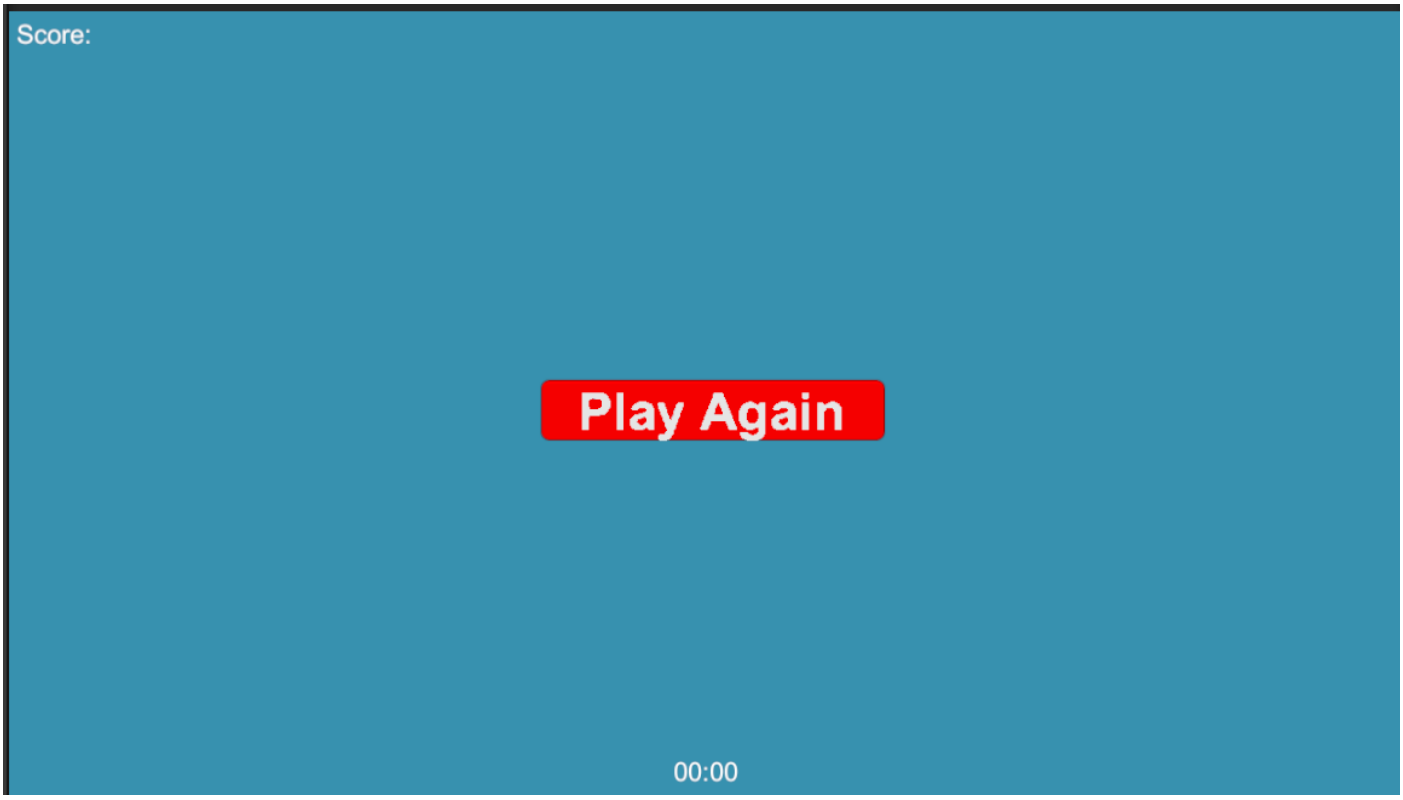


Аналогично делаем для кнопки **PlayExit**, но выбираем **ExitGame**():

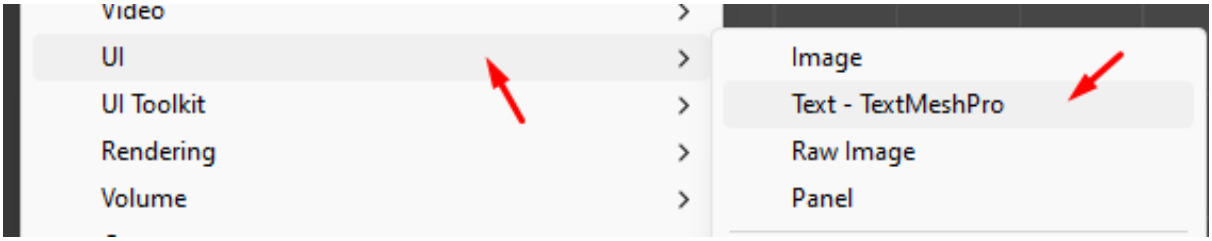


Проверьте что всё сделали правильно, запустив игру и нажав на кнопку, если вы перешли на вторую сцену, значит у вас всё получилось. А при выходе у вас закрывалась сцена.

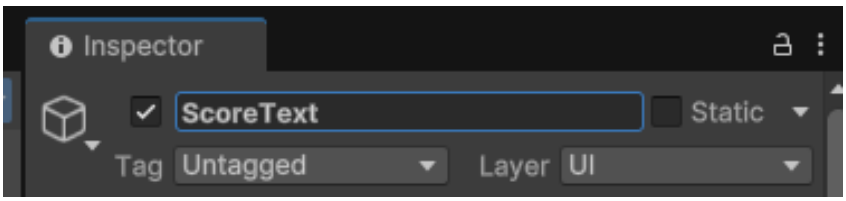
10. Переходим к основной сцене **CliclerCat,** дважды щёлкаем чтобы открыть её. Нам нужно будет оформить её приблизительно следующим образом:



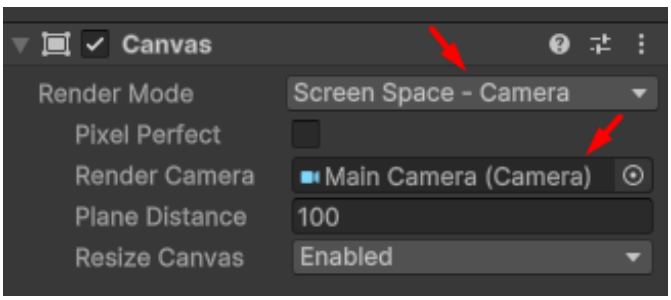
Щёлкаем правой кнопкой мыши – **UI – Text – TextMeshPro**:



Меняем для текста название на **ScoreText**:



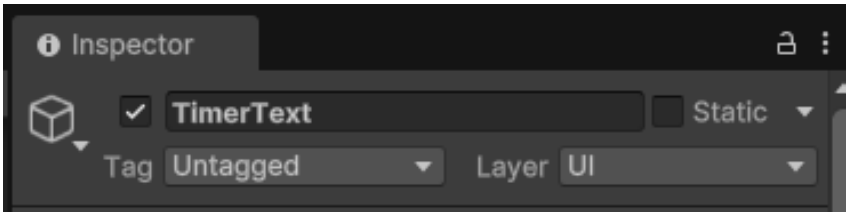
Для нашего **Canvas** меняем отображение на **Screen Space – Camera**, и добавляем нашу камеру:



В поле текста пишем - **Score:**

Настройте его на свой вкус, пример:

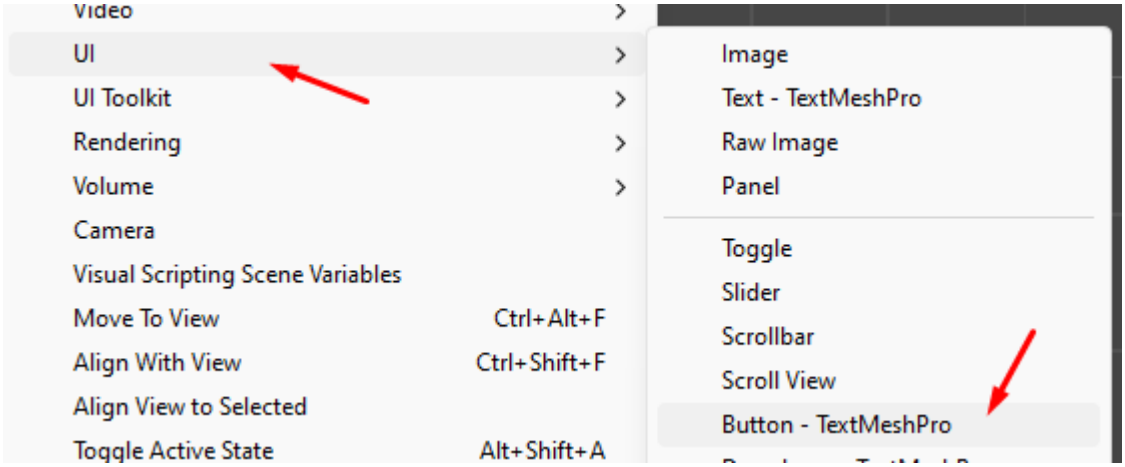
Теперь создадим аналогично текст для таймера, назовём его **TimerText:**



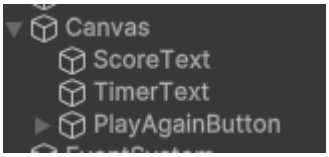
Пример оформления:

Теперь перейдём к созданию кнопки для продолжения начала игры.

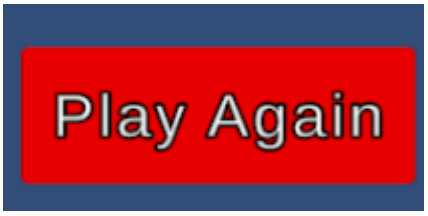
Внутри **Canvas** щёлкаем правой кнопкой мыши – **UI – Button – TextMeshPro**:



Называем её **PlayAgainButton:**

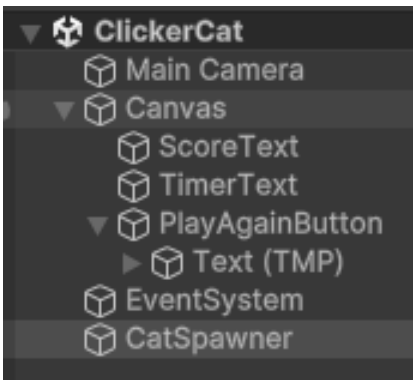


Меняем цвет текста и размер на своё усмотрение, пример:

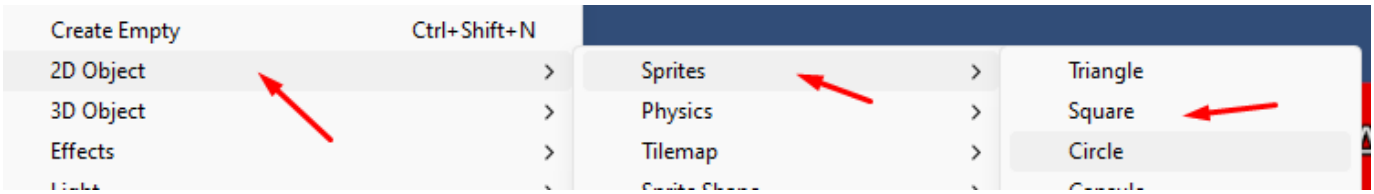


Также создадим пустой объект и назовём его **CatSpawner**

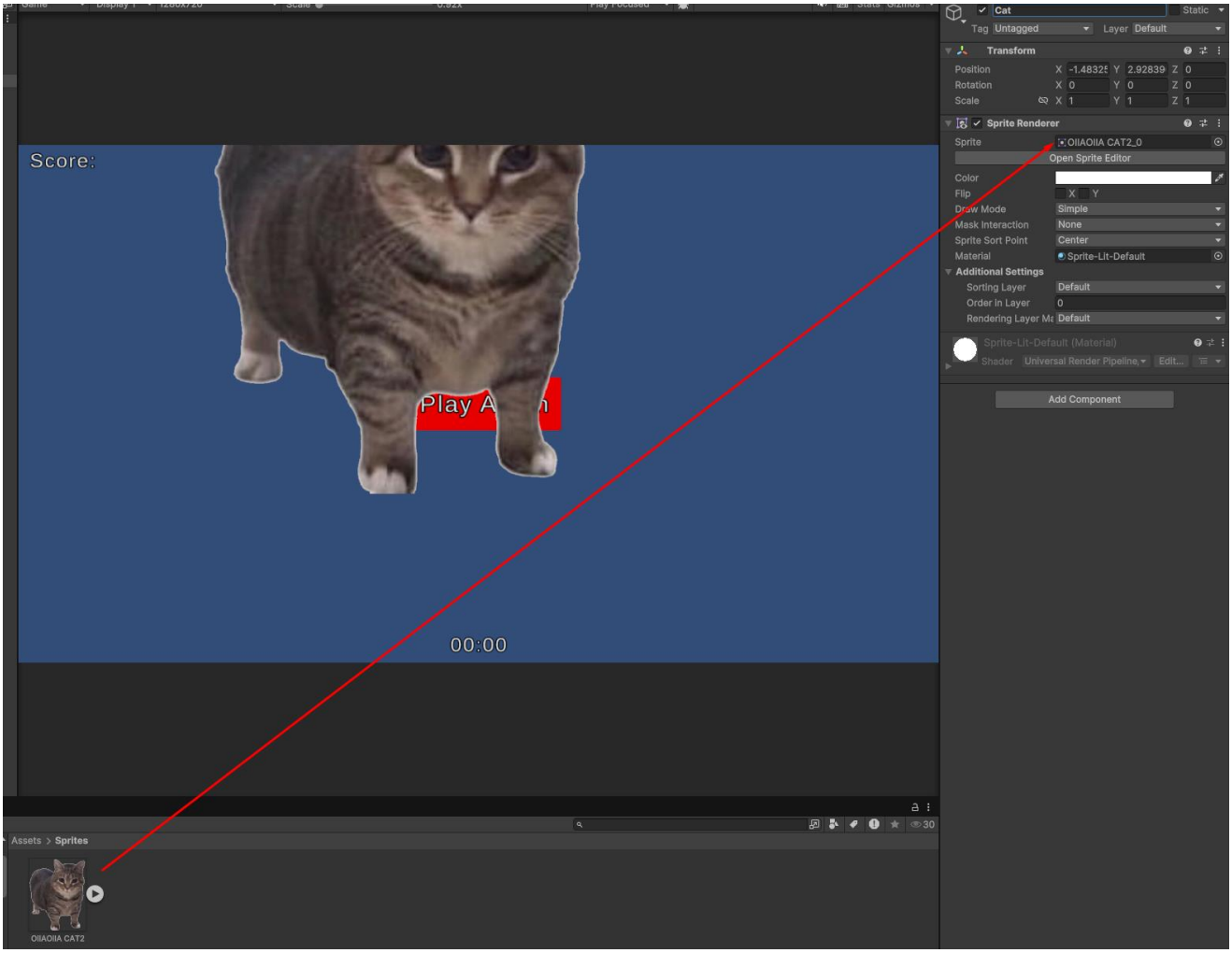
Итог:



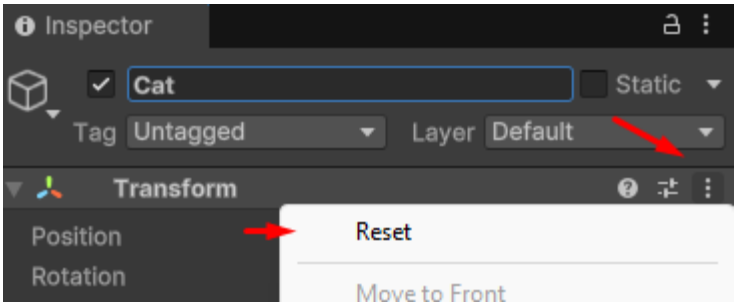
11. Создадим для нашего изображения основу, выберем **2D Object – Sprites – Circle:**



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

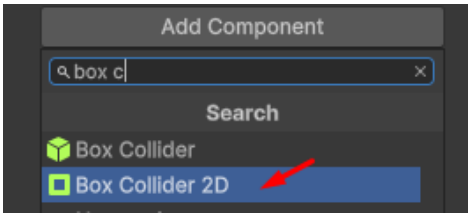


Сбросим позицию:

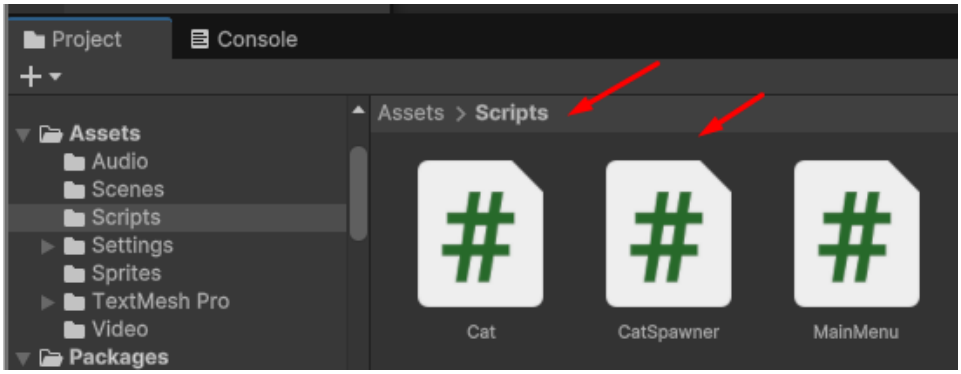


Уменьшим размер до **0.1** по **x,y,z**:

Добавим **Box Collider 2D:**



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

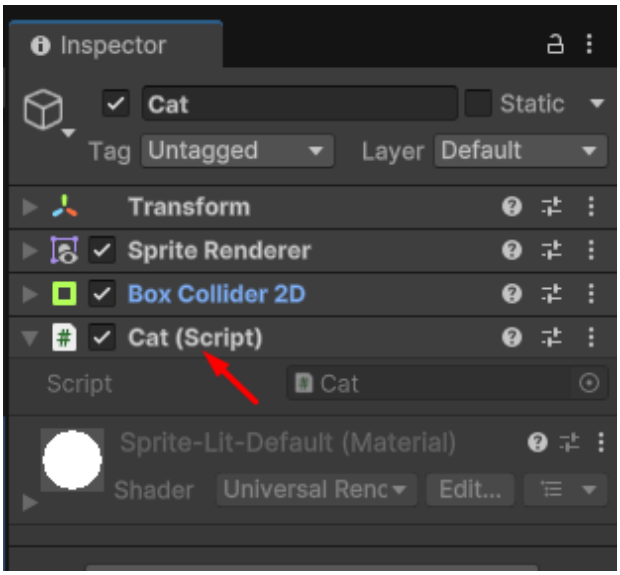


Переходим к скрипту **Cat**. Открываем его и пишем:

```
    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
        }
    }
```

Time.deltaTime *— это время в секундах, которое прошло с последнего кадра. Оно помогает делать движение, анимации и другие процессы независимыми от частоты кадров (FPS). Если мы не будем использовать, то объект будет двигаться быстрее на мощных компьютерах и медленнее на слабых.*

Добавим скрипт к нашему коту:



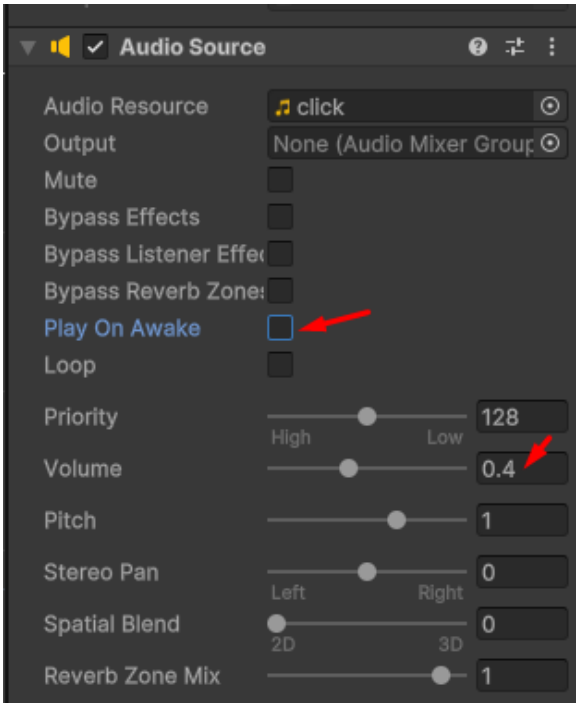
Теперь если мы запустим нашу игру, то увидим вращение у кота.

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

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

OnMouseDown вызывается, когда пользователь нажимает кнопку мыши над коллайдером. Это событие отправляется всем скриптам **GameObject** с **Collider** .

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 **Awake** получим ссылку на наш компонент:

```
private void Awake()
{
    _audioSource = GetComponent<AudioSource>();
}
```

В методе **OnMouseDown** добавим:

```
private void OnMouseDown()
{
    _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 звук. Самый простой способ добавить на камеру, которая есть на любой сцене, зациклить воспроизведение и настроить громкость. Но у этого подхода есть минусы.

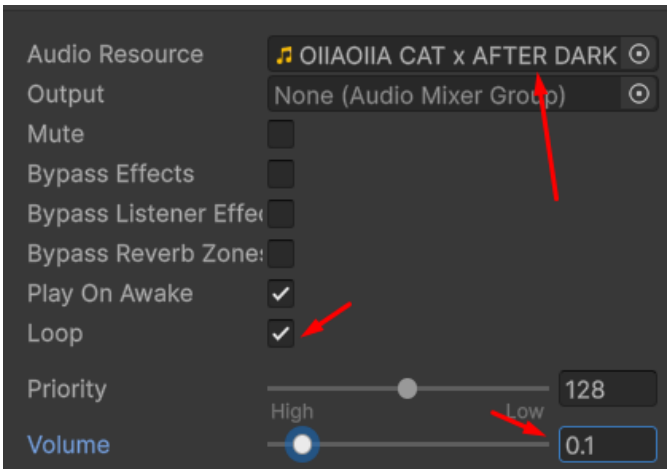
Почему НЕ стоит вешать AudioSource на камеру?

1. ***Музыка может прерваться при смене сцен***
    - *Если камера удаляется или заменяется при загрузке новой сцены,* ***AudioSource*** *пропадёт вместе с ней.*

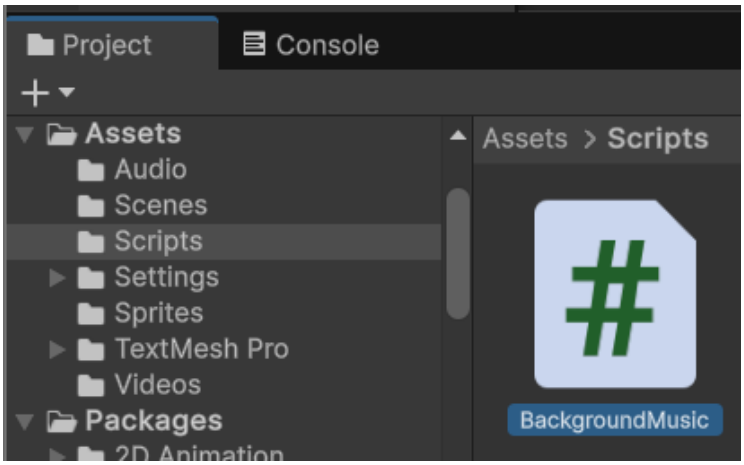
2. ***Звук будет зависеть от позиции камеры***
    - *В Unity AudioSource зависит от положения в пространстве (если spatialBlend = 1).*
    - *Если камера двигается, могут быть **нежелательные эффекты затухания**.*
3. ***Сложнее управлять***
    - *Если музыка **не привязана** к камере, её проще настраивать и менять.*

Создаём пустой объект в **Hierarchy**, называем его "**BackgroundMusic**":

Добавляем на наго **звук** и включаем **Loop,** настраиваем громкость**:**



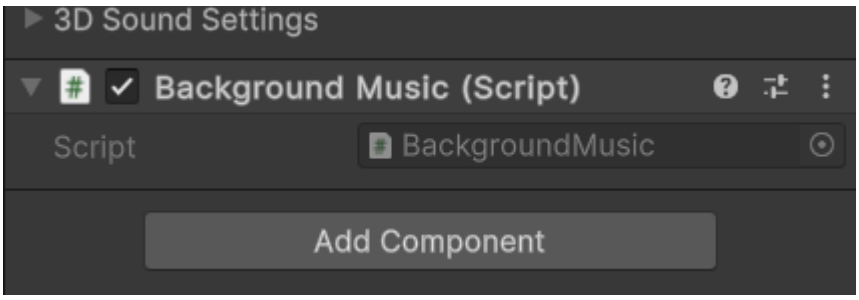
Создаём новый **скрипт BackgroundMusic** в папке **Scripts**:



Открываем его и пишем:

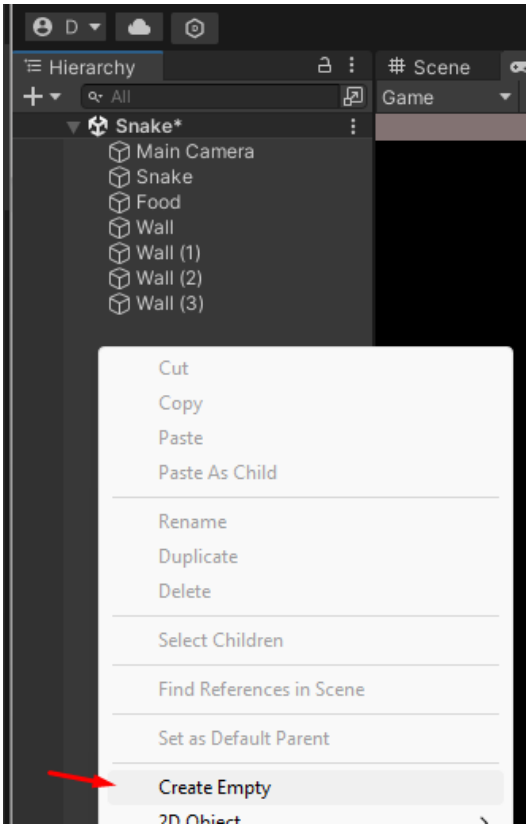
```
private void Awake()
{
      DontDestroyOnLoad(gameObject);  //  Не  уничтожать  при
загрузке новой сцены
}
```

Добавляем на наш объект **BackgroundMusic:**

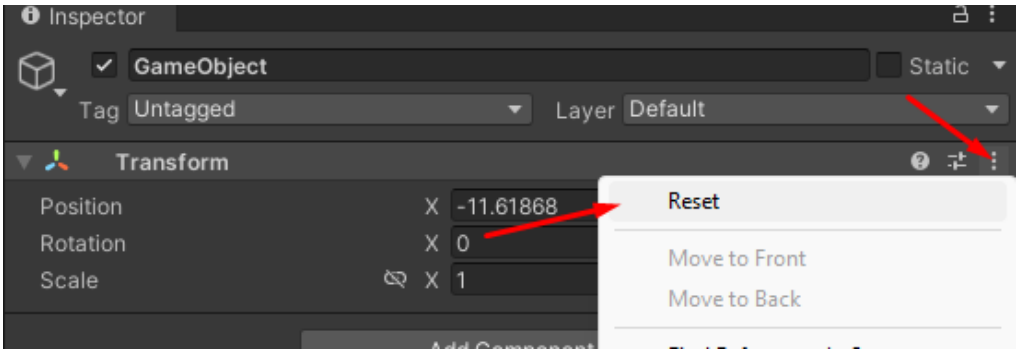


16. Теперь создадим возможность появления котов в случайной позиции на экране.

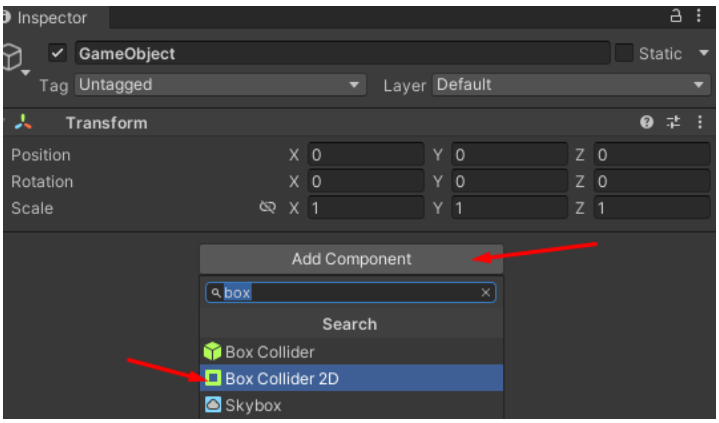
Нажимаем правой кнопкой мыши в нашей **Hierarchy → Create Empty:**



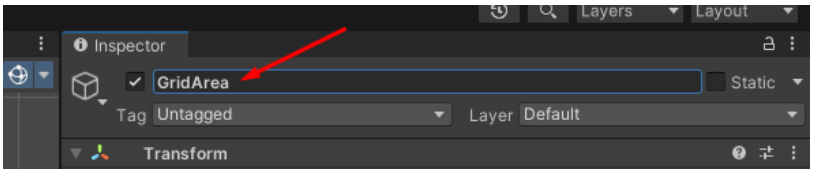
Reset:



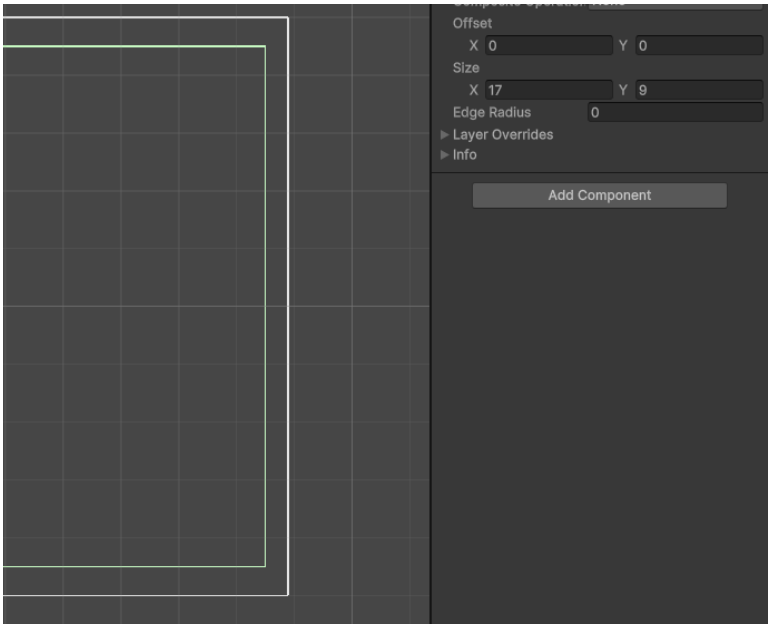
Добавляем **Box Collider 2D:**



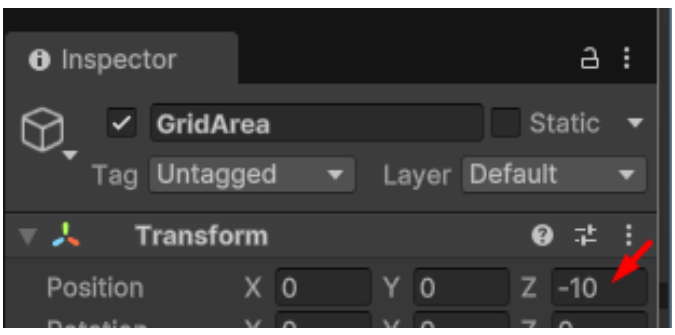
Называем **GridArea:**

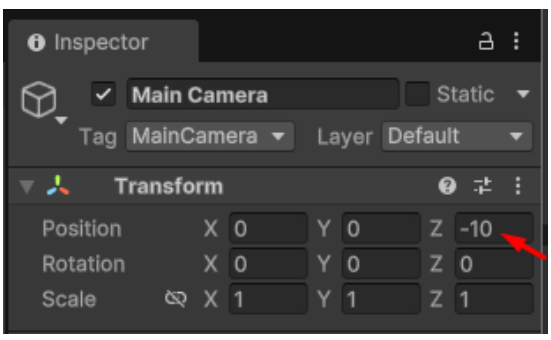


Настроим размер **Box Collider 2D** по вашему усмотрению (в моём случае это **x = 17** и **y = 9**):



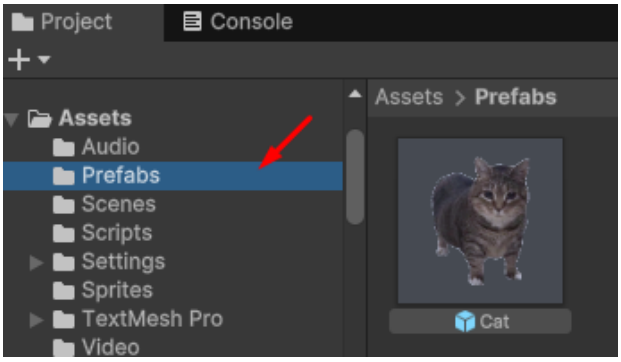
Обязательно проверьте, чтобы у вас позиция по оси Z у Camera и GridArea были одинаковы!



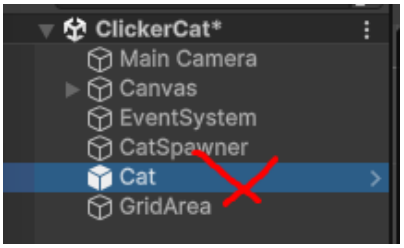


17. Создадим из нашего кота **Prefab.**

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



И удаляем его со сцены:



18. Теперь откроем скрипт **CatSpawner.**

Объявляем переменные:

```
    public GameObject catPrefab; // Ссылка на префаб объекта, который будет спавниться (кот).
    public GameObject playAgainButton; // Ссылка на кнопку "Play Again"
    public BoxCollider2D GridArea; // Границы области, в которой будут спавниться объекты.

    public float minSpawnInterval = 0.5f; // Минимальный интервал между спавнами
    public float maxSpawnInterval = 0.7f; // Максимальный интервал между спавнами
    private float nextSpawnTime; // Время для следующего спавна

    public float gameTime = 60f; // Время игры в секундах
    public bool isGameActive = true; // Флаг для отслеживания состояния игры
```

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.
Time.time - текущее время.
    }
```

После пропишем метод **Update()**:

```
    private void Update()
    {
        if (gameTime > 0) // Если время игры ещё не закончилось
        {
            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 выполнение скрипта
        }
    }
```

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.
    }
```

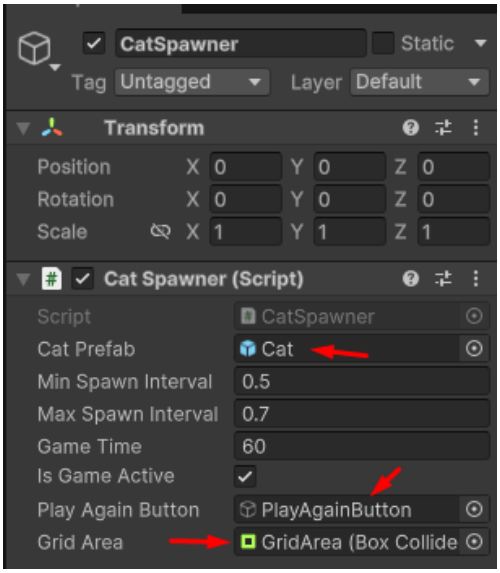
Что такое Quaternion?

Quaternion — это математическая структура, используемая в Unity для представления вращения объектов в 3D-пространстве. В отличие от углов Эйлера, кватернионы позволяют избежать проблем, таких как блокировка осе), и обеспечивают более плавные и стабильные вращения.

Quaternion.identity *— это специальное значение в Unity, которое представляет отсутствие вращения (или нейтральное вращение).*

В ***CatSpawner*** *коты создаются без начального вращения благодаря* ***Quaternion.identity****. В скрипте* ***Cat*** *каждому коту присваивается случайная скорость вращения (****rotationSpeed****), которая определяет, как быстро он будет вращаться. Таким образом, каждый созданный кот будет уникальным по скорости вращения, но начнёт свою жизнь без начального вращения.*

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



Обновим немного наш скрипт **Cat**:

```
using UnityEngine;

public class Cat : MonoBehaviour
{
    private CatSpawner _spawner; // получаем ссылку на
скрипт CatSpawner
    private AudioSource _audioSource;
    private float rotationSpeed;

    private void Awake()
    {
        _audioSource = GetComponent<AudioSource>();
        _spawner = FindAnyObjectByType<CatSpawner>(); //
ищем объект спаунера
    }
    private void Start()
    {
        rotationSpeed = Random.Range(200f, 800f);
    }
    private void Update()
    {
        transform.Rotate(0, 0, rotationSpeed *
Time.deltaTime);
    }
    private void OnMouseDown()
    {
        if(_spawner.isGameActive) // выполняем проверку
        {
            _audioSource.Play();
            Destroy(gameObject, _audioSource.clip.length);
        }
    }
}
```

Атрибуты в Unity *— это специальные метаданные, которые добавляются к классам, методам, полям или свойствам для изменения их поведения или отображения в редакторе Unity. Они не влияют на логику кода напрямую, но предоставляют дополнительные инструкции для Unity, например, как отображать поле в инспекторе или как обрабатывать определённые события.*

Небольшой рефакторинг:

```
[Header("Объекты")]
public GameObject catPrefab;
public GameObject playAgainButton;
public BoxCollider2D GridArea;

[Header("Настройки спавна")]
[SerializeField] private float minSpawnInterwal = 0.3f;
[SerializeField] private float maxSpawnInterwal = 0.7f;
private float nextSpwanTime;

[Header("Настройки игры")]
[SerializeField] private float gameTime = 60f;
public bool isGameActive = true;
```

[Header("Настройки спавна")] в Unity

*Это атрибут ([**Header**]), который используется для организации и улучшения читаемости элементов в Инспекторе Unity. Он позволяет сгруппировать переменные по категориям и добавить описания или заголовки, чтобы улучшить восприятие кода при работе с ним через редактор Unity. Когда добавляется атрибут **[Header ("Название группы")]** перед полем в классе, то в **Инспекторе** Unity появляется заголовок или раздел с этим названием.*

***[SerializeField]** — это атрибут, который позволяет **сериализовать** (сохраняет его значение в файле сцены или префаба), приватные поля, чтобы они **отображались в Инспекторе Unity**. Без этого атрибута приватные поля **не показываются** в Инспекторе, даже если они являются переменными публичного доступа.*

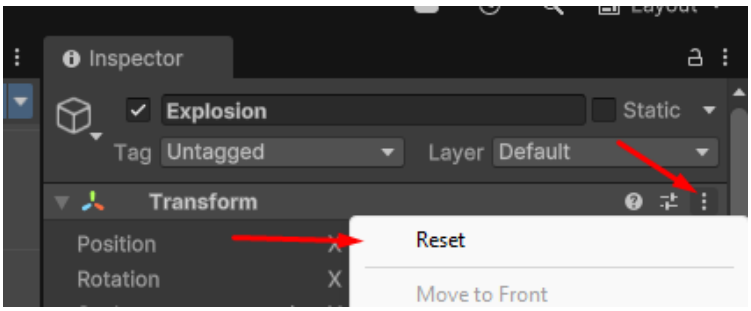
19. Теперь добавим простой эффект частиц при уничтожении котов.

***Система частиц (Particle System)** в Unity — это мощный инструмент для создания визуальных эффектов, таких как огонь, дым, взрывы, магические заклинания, дождь, снег и многое другое. Она позволяет управлять большим количеством мелких частиц, которые вместе создают сложные и динамичные эффекты.*

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

Назовём её **Explosion**:

Сбросим трансформацию (это важно, иначе будет себя вести не так):



Duration (продолжительность) - **1;**

Looping (зацикливание) уберём;

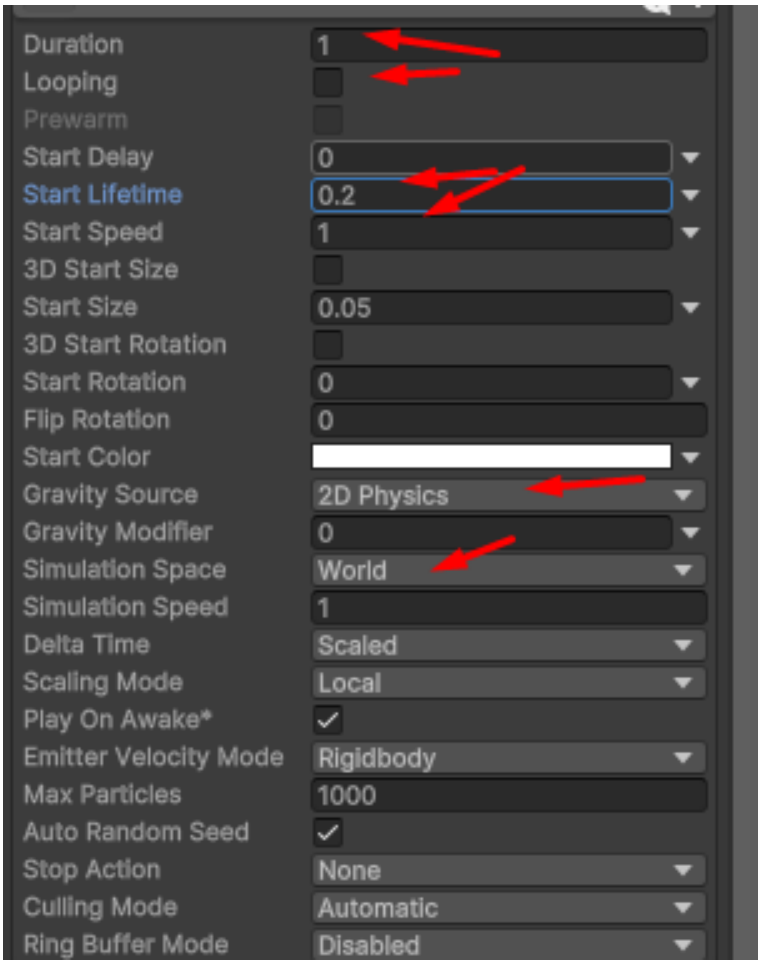
Start Lifetime (продолжительность жизни) – **0.2**;

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

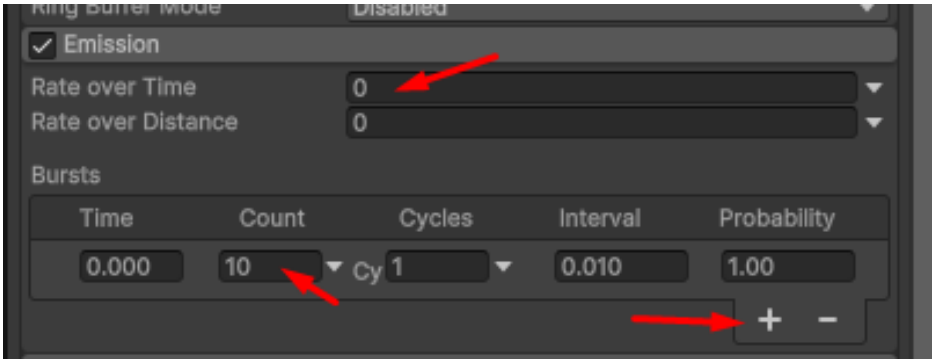
Start Size (начальный размер) – **0.05**;

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

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

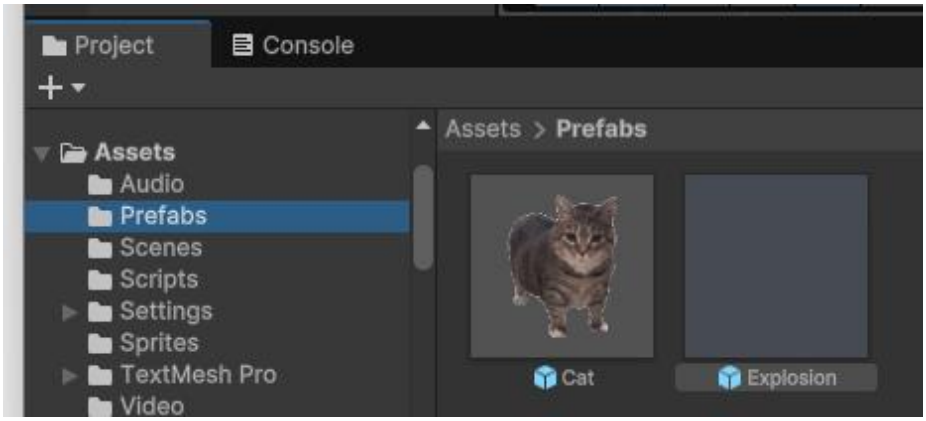


Раскроем меню **Emission** (контролирует, как частицы создаются (например, количество частиц в секунду) и установим **Rate over Time** на **0**, и добавим новый список, где установим количество наших частиц **10**:

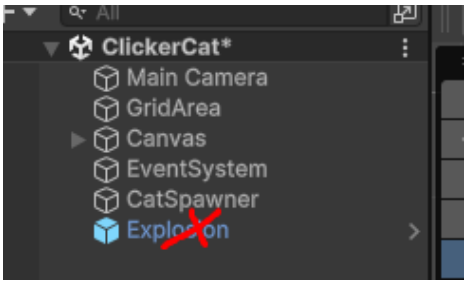


В меню **Shape** поменяем форму на **Circle** и установим минимальный радиус **0.0001:**

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



И удалите из иерархии:



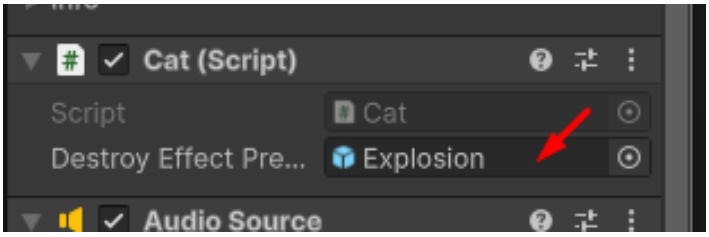
20. Обновим скрипт **Cat**. Добавим новую переменную:

```
public GameObject _destroyEffectPrefabs; // получаем ссылку на скрипт CatSpawner
```

Обновим метод **OnMouseDown**:

```
private void OnMouseDown()
{
    if(_spawner.isGameActive)
    {
        Instantiate(_destroyEffectPrefabs, transform.position,
          Quaternion.identity); // воспроизводим эффект
        _audioSource.Play();
        Destroy(gameObject, _audioSource.clip.length);
    }
}
```

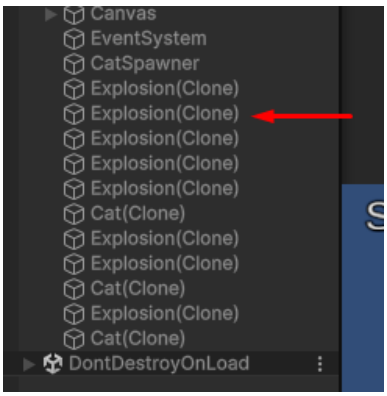
Затем добавим наш префаб системы частиц **Explosion** в скрипт **Cat** у префаба **Cat**:



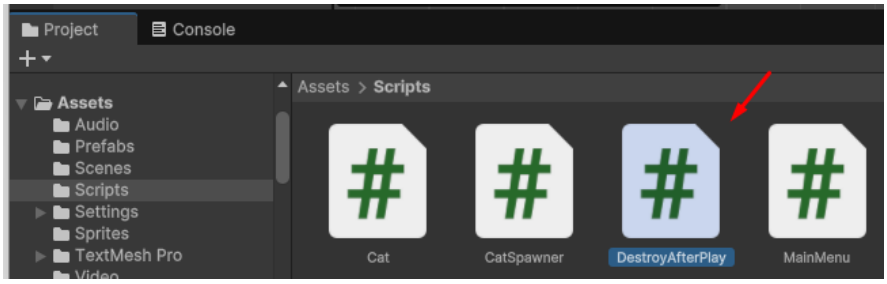
Сделаем небольшой рефакторинг:

```
[Header("Референсы")]
[SerializeField] private GameObject _destroyEffectPrefabs;
private AudioSource _audioSource;

[Header("Настройки")]
[SerializeField] private float _minRotationSpeed = 200f;
[SerializeField] private float _maxRotationSpeed = 800f;

private CatSpawner _spawner;
private float rotationSpeed;

private void Start()
{
    rotationSpeed = Random.Range(_minRotationSpeed,
_maxRotationSpeed);
}
```

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**:



Открываем его и напишем код:

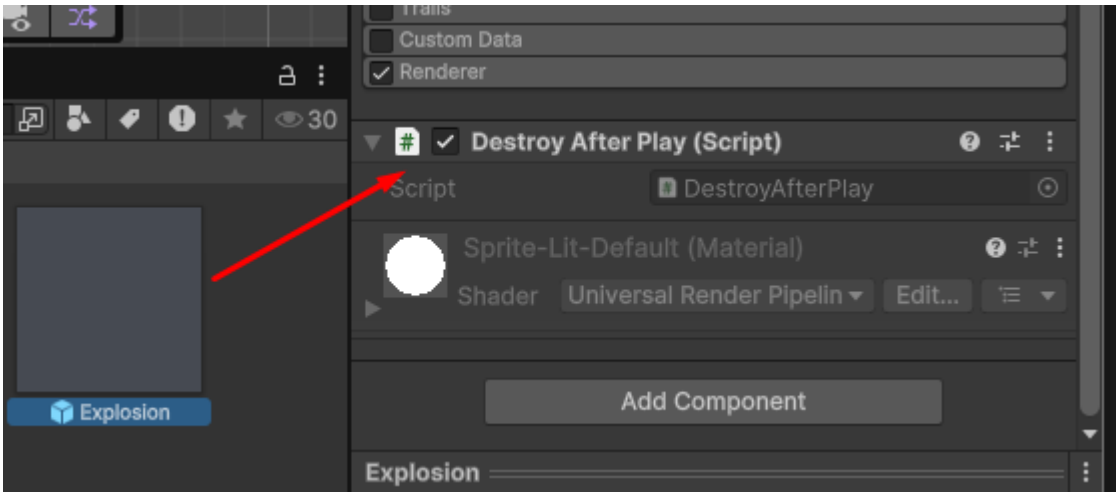
```
    using UnityEngine;

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

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

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

Эта часть условия проверяет, есть ли у объекта компонент ***ParticleSystem.*** *Если компонент* ***ParticleSystem*** *существует, метод вернет ссылку на него.* *Если компонент отсутствует, метод вернет* ***null.*** *Проверка* ***!= null*** *гарантирует, что дальнейший код не вызовет ошибку, если компонент* ***ParticleSystem*** *отсутствует.*

IsAlive() *— это метод компонента* ***ParticleSystem****, который возвращает* ***true****, если система частиц все еще активна (то есть хотя бы одна частица существует или система еще не завершила свою работу). Если система частиц завершила свою работу и больше не испускает частицы, метод вернет* ***false****.*

Оператор ***!*** *(логическое НЕ) инвертирует результат. Таким образом,* ***!IsAlive()*** *означает, что условие выполнится, если система частиц не активна (то есть завершила свою работу).*

Присоедините скрипт к префабу системы частиц:

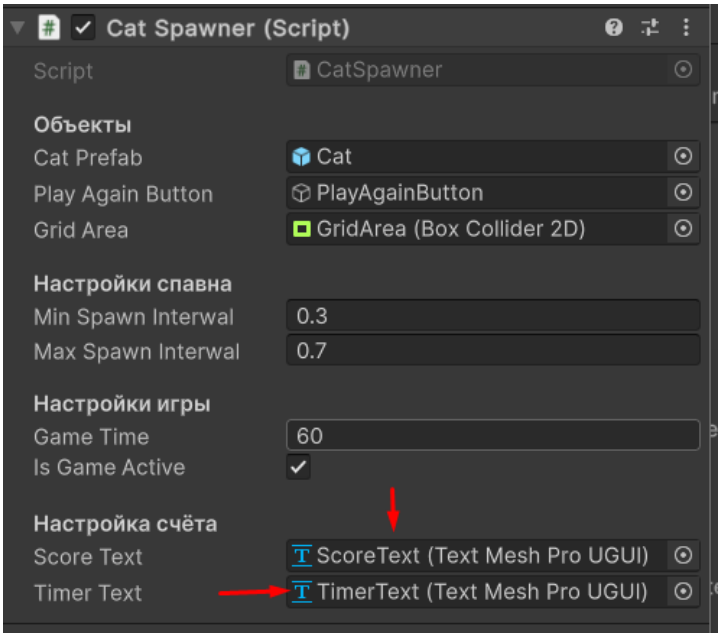


Теперь, когда будет создаваться система частиц с помощью **Instantiate**, она автоматически уничтожится после завершения анимации.

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

```
[Header("Настройка счёта")]
[SerializeField] private TextMeshProUGUI _scoreText;
[SerializeField] private TextMeshProUGUI timerText;
private int score = 0;
```

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);
    nextSpwanTime = Time.time + Random.Range(minSpawnInterwal, maxSpawnInterwal);
    UpdateScoreText();
}
```

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

```
        private void UpdateTimerText() // Метод для обновления текста таймера
        {
            timerText.text = $"Time: {Mathf.CeilToInt(gameTime)}"; // 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 напишем 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.isGameActive)
    {
        Instantiate(_destroyEffectPrefabs,
          transform.position, Quaternion.identity);
        _audioSource.Play();
        Destroy(gameObject, _audioSource.clip.length);
        _spawner.AddScore(1); // Добавляем очки при
          уничтожении
    }
}
```

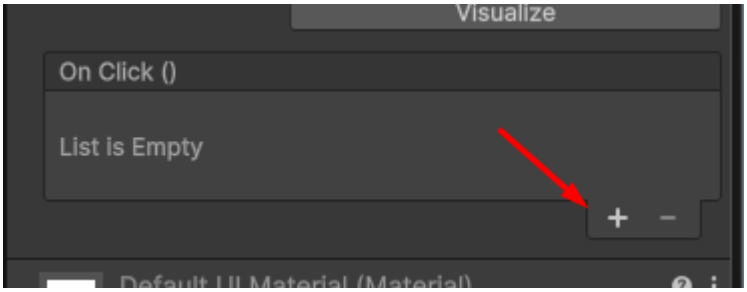
Возвращаемся в скрипт **CatSpawner,** и напишем метод **PlayAgain()** который будет по нажатию кнопки перезапускать игру:

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

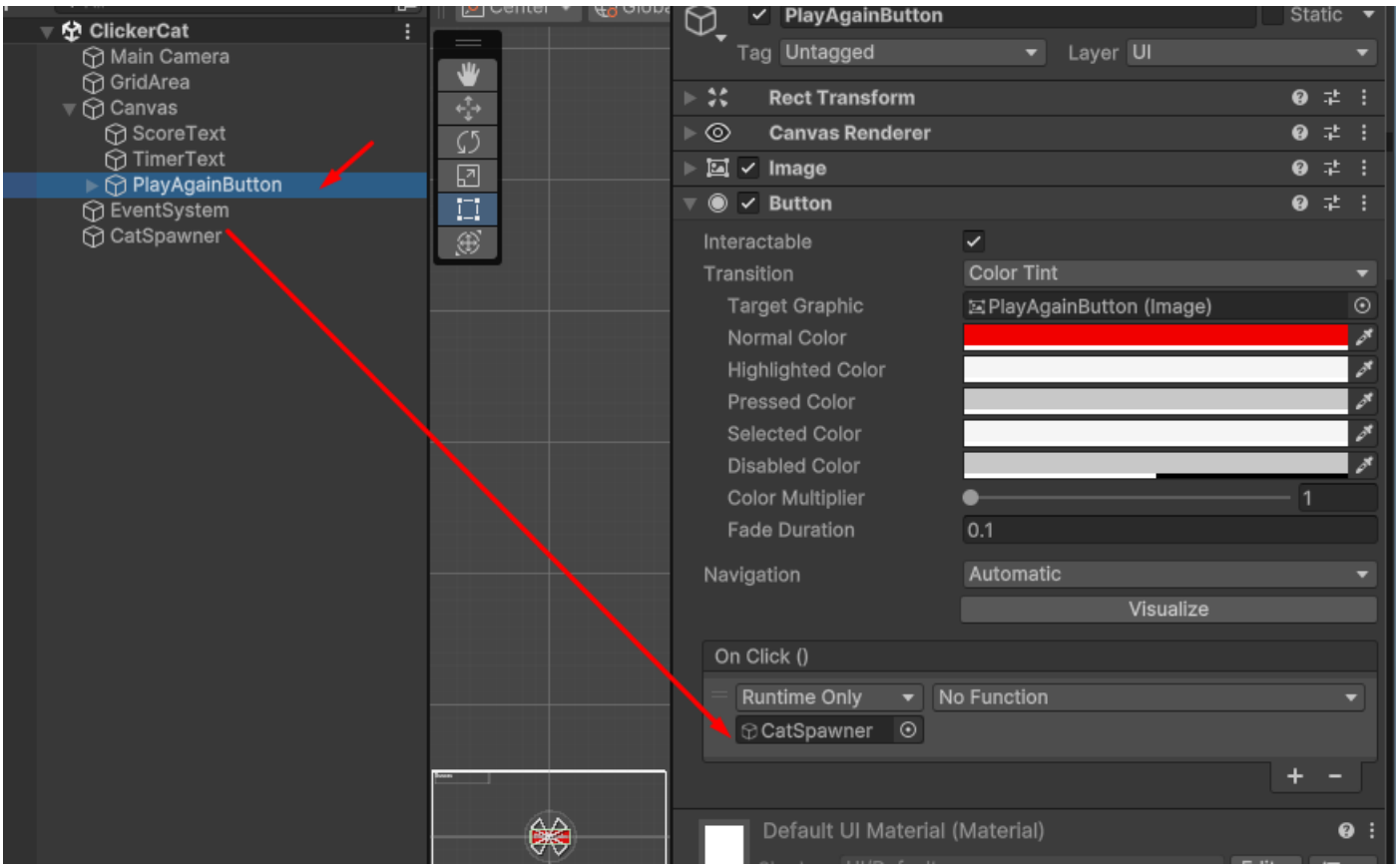
Обратите внимание, что у вас должно было подключиться в начале скрипта пространство имён **UnityEngine.SceneManagement:**

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

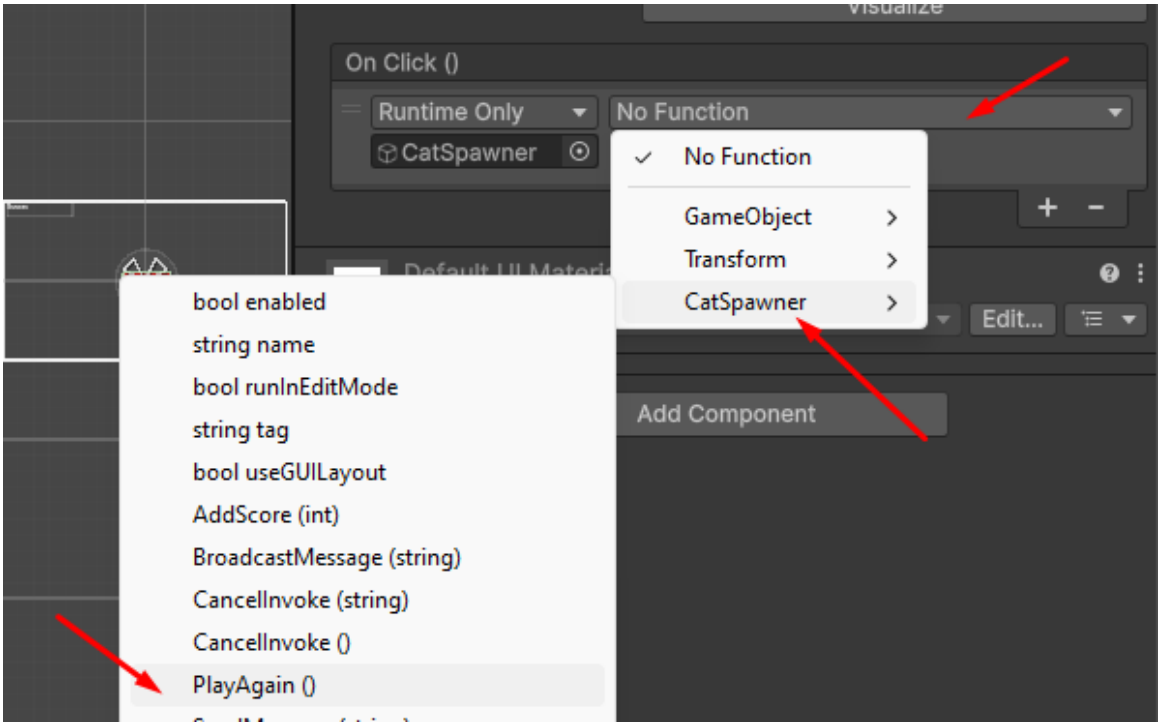
Теперь нам нужно в Unity для нашей кнопки добавить это событие. Находим у кнопки **PlayAgainButton** метод **On Click()** и нажимаем +:



Переносим объект **CatSpawner**:



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



Запускам игру, тестируем. И ловим **баг** – при окончании игры у нас продолжает играть звук, и воспроизводится новый.

Проблема в том, что при перезапуске сцены создаётся **новый дубликат объекта с фоновой музыкой**, потому что скрипт **BackgroundMusic** использует **DontDestroyOnLoad(gameObject)**.

Как исправить?

1-ый способ, продолжать игру музыку.

Нам нужно **убедиться, что при загрузке новой сцены не создаётся дубликат музыки**. Для этого можно немного изменить твой скрипт **BackgroundMusic**, добавив проверку перед **DontDestroyOnLoad**:

```
private void Awake()
{
    if (instance == null)
    {
        instance = this;
        DontDestroyOnLoad(gameObject);
    }
    else
    {
        Destroy(gameObject); // Если такой объект уже есть, удаляем новый
    }
}
```

Как это работает?

1. ***instance*** *— это* ***статическая переменная****, которая хранит ссылку на единственный объект* ***BackgroundMusic****.*
2. ***Первый раз*** *при запуске игры:*
    - ***instance == null****, поэтому объект* ***сохраняется*** *(****DontDestroyOnLoad****).*
3. ***При перезапуске сцены****:*
    - ***Awake****() снова вызывается у нового объекта.*
    - *Так как* ***instance*** *уже* ***не null****, новый объект* ***удаляется*** *(****Destroy(gameObject)****).*
    - *В результате в игре остаётся* ***только один объект*** *с музыкой, и звук не дублируется.*

2-ый способ, останавливать звук при перезапуске.

Если мы хотим, чтобы музыка перезапускалась вместе с игрой, нужно добавить в скрипт **BackgroundMusic** новый метод **StopMusic:**

```
public void StopMusic()
{
    GetComponent<AudioSource>().Stop();
}
```

А затем в скрите **CatSpawner** в метод **PlayAgain** вызвать его перед загрузкой сцены**:**

```
public void PlayAgain()
{
    FindObjectOfType<BackgroundMusic>()?.StopMusic(); //
Останавливаем музыку перед рестартом
    SceneManager.LoadScene(SceneManager.GetActiveScene().na
me);
}
```

** В Unity 6 можно использовать метод* ***FindFirstObjectByType***

В Unity 6 есть отличие между методами FindAnyObjectByType и FindFirstObjectByType*.*

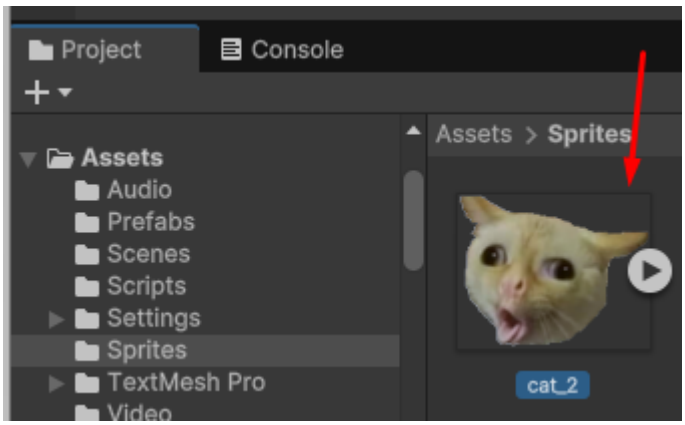
FindAnyObjectByType *возвращает произвольный активный загруженный объект указанного типа. Этот метод может быть более оптимизированным для многопоточного окружения.*

FindFirstObjectByType *возвращает первый объект указанного типа. Функционально он идентичен* ***FindObjectOfType****, но имеет новое название, чтобы отличить его от* ***FindAnyObjectByType****. Этот метод работает чуть быстрее, так как находит первый объект нужного типа.*

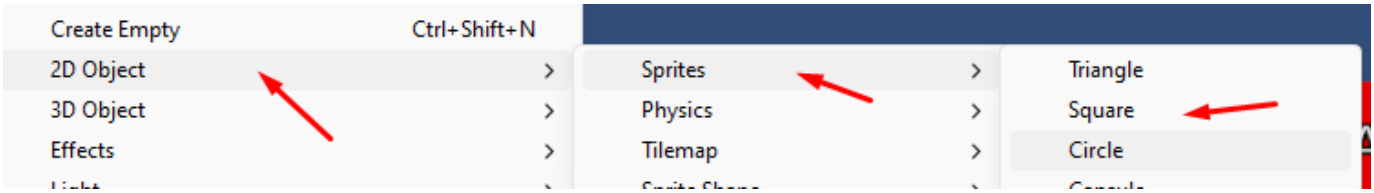
Поскольку у нас должен быть только один объект ***BackgroundMusic****, лучше использовать* ***FindFirstObjectByType****(), так как он чуть быстрее и логичнее для поиска единственного объекта.*

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

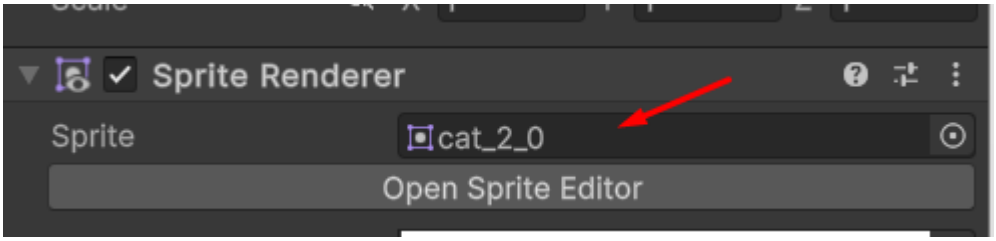
Для этого добавьте из папки с ассетами нового кота **cat_2** в папку **Sprites**:



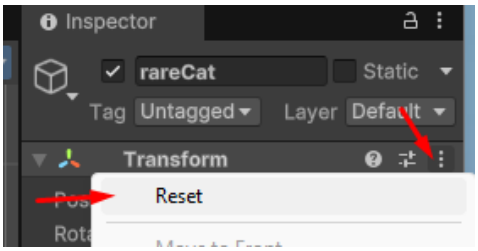
Создадим для нашего изображения основу, выберем **2D Object – Sprites – Circle:**



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

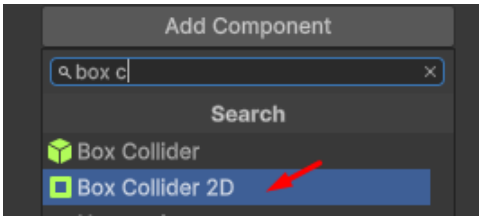


Сбросим позицию:

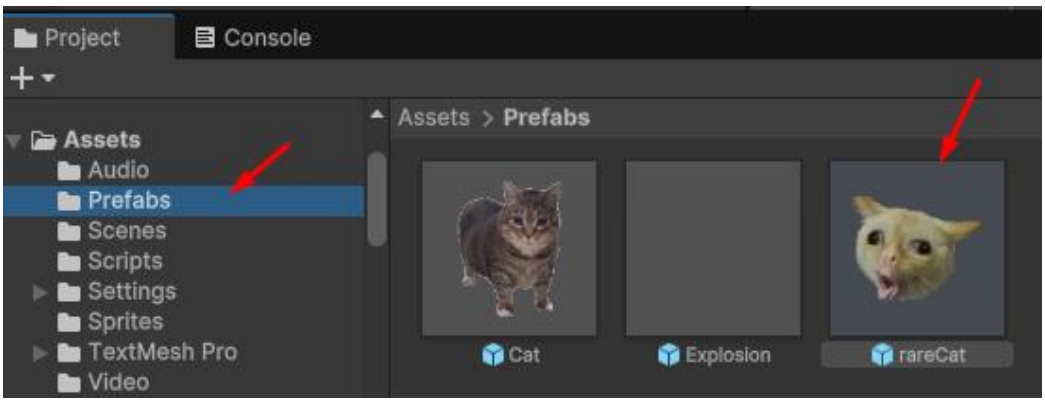


Уменьшим размер до **0.15** по **x,y**:

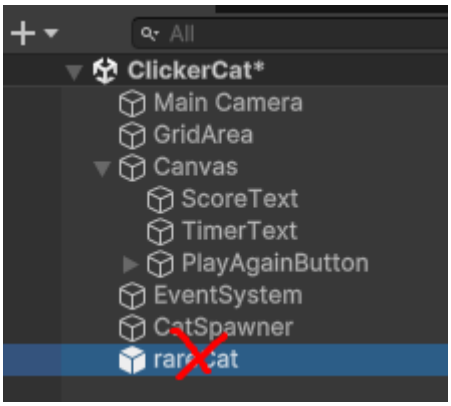
Добавим **Box Collider 2D:**



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:

```
[Header("Объекты")]
public GameObject rareCatPrefab; // префаб редкого кота
public GameObject catPrefab;
public GameObject playAgainButton;
public BoxCollider2D GridArea;

[Header("Настройки спавна")]
[SerializeField] private float rareCatChance = 0.1f; // Шанс появления
  редкого кота (10%)
[SerializeField] private float minSpawnInterwal = 0.3f;
[SerializeField] private float maxSpawnInterwal = 0.7f;
private float nextSpwanTime;
```

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

```
GameObject catToSpawn;

if (Random.value < rareCatChance)
{
    catToSpawn = rareCatPrefab; // Спавним редкого кота
}
else
{
    catToSpawn = catPrefab; // Спавним обычного кота
}

GameObject cat = Instantiate(catToSpawn, spawnPosition,
  Quaternion.identity);
```

Используя тернарную операцию, мы можем сократить написание блока **if/else**:

```
GameObject catToSpawn = Random.value < rareCatChance ?
  rareCatPrefab : catPrefab;

GameObject cat = Instantiate(catToSpawn, spawnPosition,
  Quaternion.identity);
```

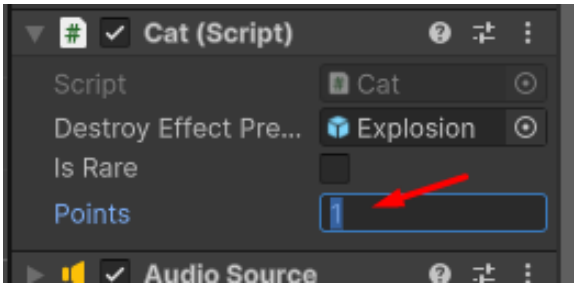
Открываем скрипт **Cat,** немного изменим структуру переменных, а также добавляем переменную для определения редкости кота и количество очков за его уничтожение:

```
[Header("Референсы")]
[SerializeField] private GameObject _destroyEffectPrefabs;
private AudioSource _audioSource;
private CatSpawner _spawner;

[Header("Настройки вращения")]
[SerializeField] private float _minRotationSpeed = 200f;
[SerializeField] private float _maxRotationSpeed = 800f;
private float rotationSpeed;

[Header("Очки")]
[SerializeField] private int _points = 5; // Количество
очков за уничтожение

public bool isRare = false; // Флаг для редкого кота
```

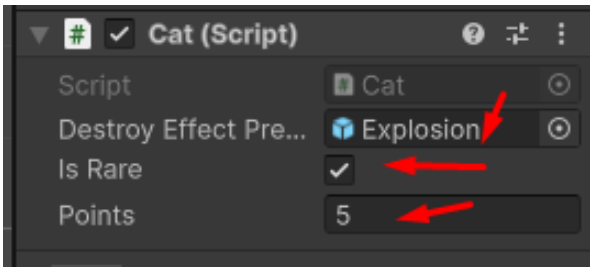
Изменяем метод **OnMouseDown**, чтобы добавлять разные очки в зависимости от редкости кота:

```
private void OnMouseDown()
{
    if(_spawner.isGameActive)
    {
        Instantiate(_destroyEffectPrefabs, transform.position,
          Quaternion.identity);
        _audioSource.Play();
        Destroy(gameObject, _audioSource.clip.length);
        _spawner.AddScore(_points); // Добавляем очки при уничтожении
    }
}
```

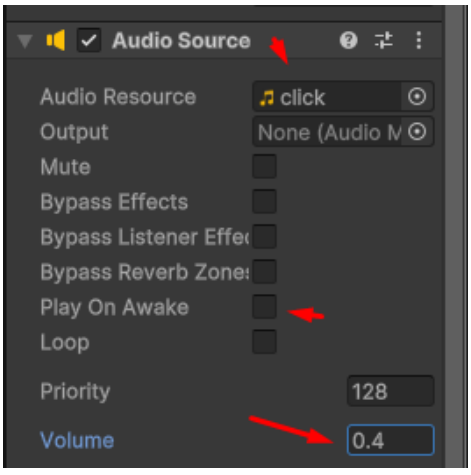
Теперь в Unity для префаба **Cat** в скрипте **Cat** ставим **1** очко:



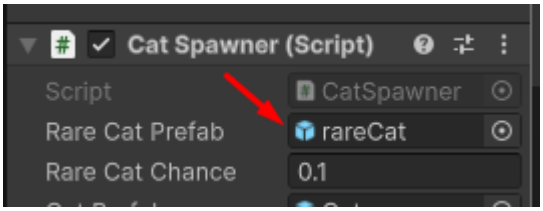
Для префаба **rareCat** добавляем скрипт **Cat,** ставим **5** очков и галочку **Is Rare**, и наш префаб **Explosion**:



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 **Awake** уничтожения нашего редкого кота по истечению времени:

```
private void Awake()
{
    _audioSource = GetComponent<AudioSource>();
    _spawner = FindAnyObjectByType<CatSpawner>();

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

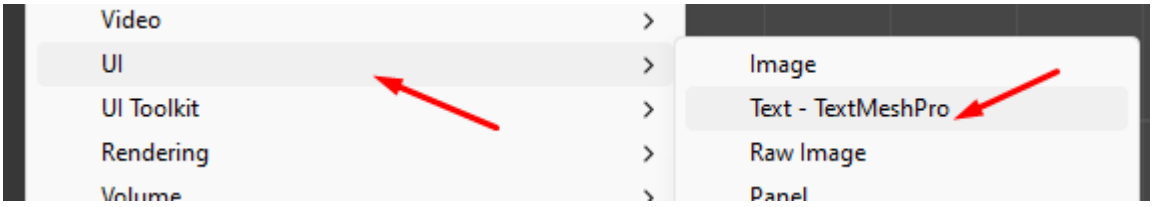
Какие проблемы есть в нашем коде?

Если мы забудем добавить эффекты, звук то у нас выдаст ошибку (**NullReferenceException**). Давайте решим эту проблему. В методе **OnMouseDown** внесём правки:

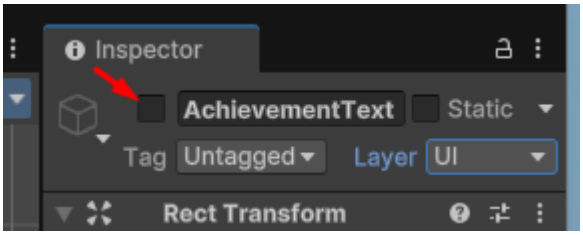
```
private void OnMouseDown()
{
    // Проверяем, существует ли объект CatSpawner и активна
ли игра
    if (_spawner == null || !_spawner.isGameActive)
        return; // Если нет, ничего не делаем

    // Если задан префаб эффекта уничтожения, создаём его в
позиции кота
    if (_destroyEffectPrefabs != null)
    {
```

```
        Instantiate(_destroyEffectPrefabs,
transform.position, Quaternion.identity);
    }

    // Проверяем, есть ли у кота аудиокомпонент и загружен
ли в него звуковой файл
    if (_audioSource != null && _audioSource.clip != null)
    {
        _audioSource.Play(); // Проигрываем звук
        Destroy(gameObject, _audioSource.clip.length); //
Удаляем кота после завершения звука
    }
    else
    {
        Debug.LogWarning("Отсутствует звук!"); // Выводим
предупреждение в консоль
        Destroy(gameObject); // Удаляем кота сразу, если
звука нет
    }

    // Добавляем очки за уничтожение кота
    _spawner.AddScore(_points);
}
```

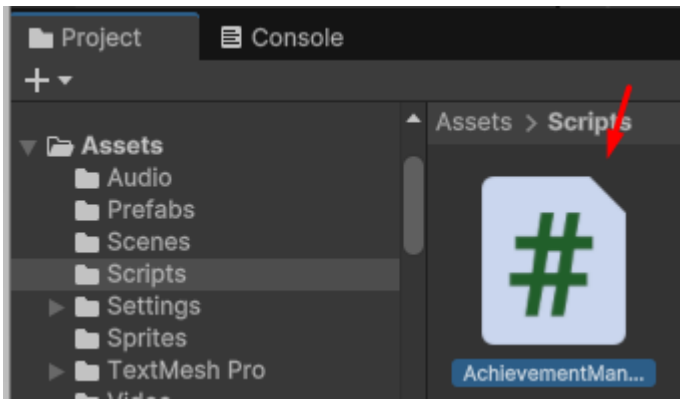
24. Ну и напоследок давайте с вами реализуем систему **Ачивок**, которые будут появляться в углу экрана. В нашем **Canvas** создаём новый текст **UI – Text – TextMeshPro**:



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:

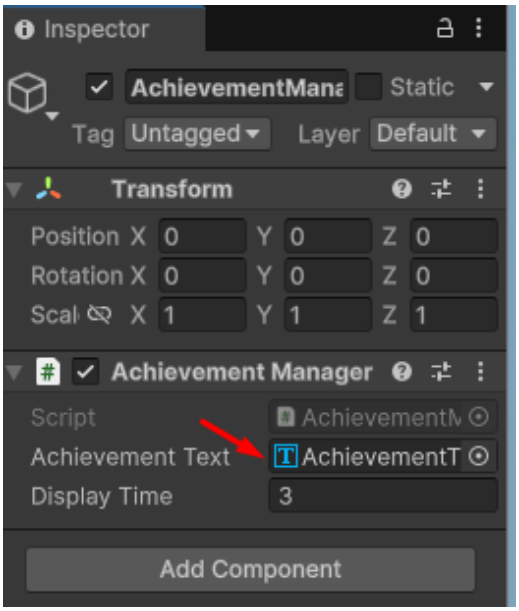
1. Запуск корутины: **StartCoroutine** используется для запуска корутины.

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
```

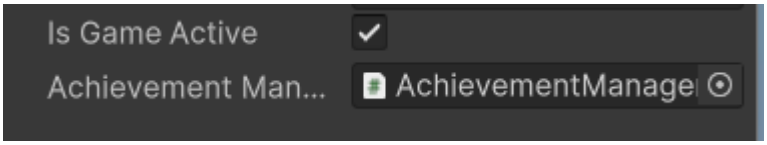
Немного изменим метод **AddScore:**

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

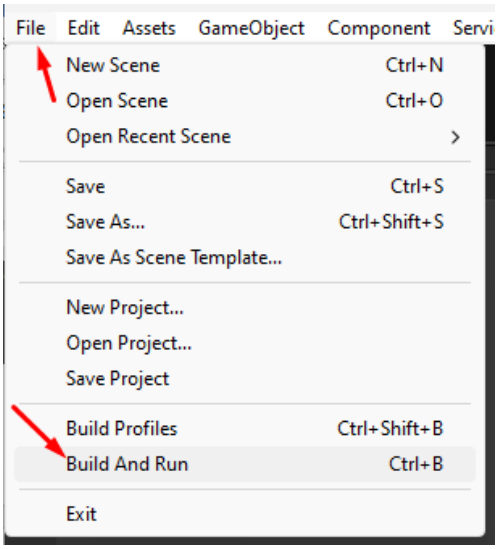
В скрипте **Cat** в методе **OnMouseDown** внесём изменения:

```
_spawner.AddScore(_points, isRare);
```

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.