

Лабораторная работа №1

Знакомство с Unity3D. Создание простейшей игры на Unity3D

Введение

Unity — это инструмент для разработки двух- и трёхмерных приложений и игр, работающий под операционными системами Windows, Linux и OS X. Созданные с помощью Unity приложения работают под операционными системами Windows, OS X, Windows Phone, Android, Apple iOS, Linux, а также на игровых приставках Wii, PlayStation 3, PlayStation 4, Xbox 360, Xbox One и MotionParallax3D дисплеях (устройства для воспроизведения виртуальных голограмм), например, Nettlebox. Есть возможность создавать приложения для запуска в браузерах с помощью специального подключаемого модуля Unity(Unity Web Player), а также с помощью реализации технологии WebGL. Ранее была экспериментальная поддержка реализации проектов в рамках модуля Adobe Flash Player, но позже команда разработчиков Unity приняла сложное решение по отказу от этого.

Приложения, созданные с помощью Unity, поддерживают DirectX и OpenGL. Активно движок используется как крупными разработчиками (Blizzard[5], EA, QuartSoft, Ubisoft[6]), так и разработчиками Indie-игр (например, ремейк Мор. Утопия (Pathologic), Kerbal Space Program, Slender: The Eight Pages, Slender: The Arrival, Surgeon Simulator 2013, Baeklyse Apps: Guess the actor и т. п.) в силу наличия бесплатной версии, удобного интерфейса и простоты работы с движком.

Интерфейс Unity представлен на рисунке 1.

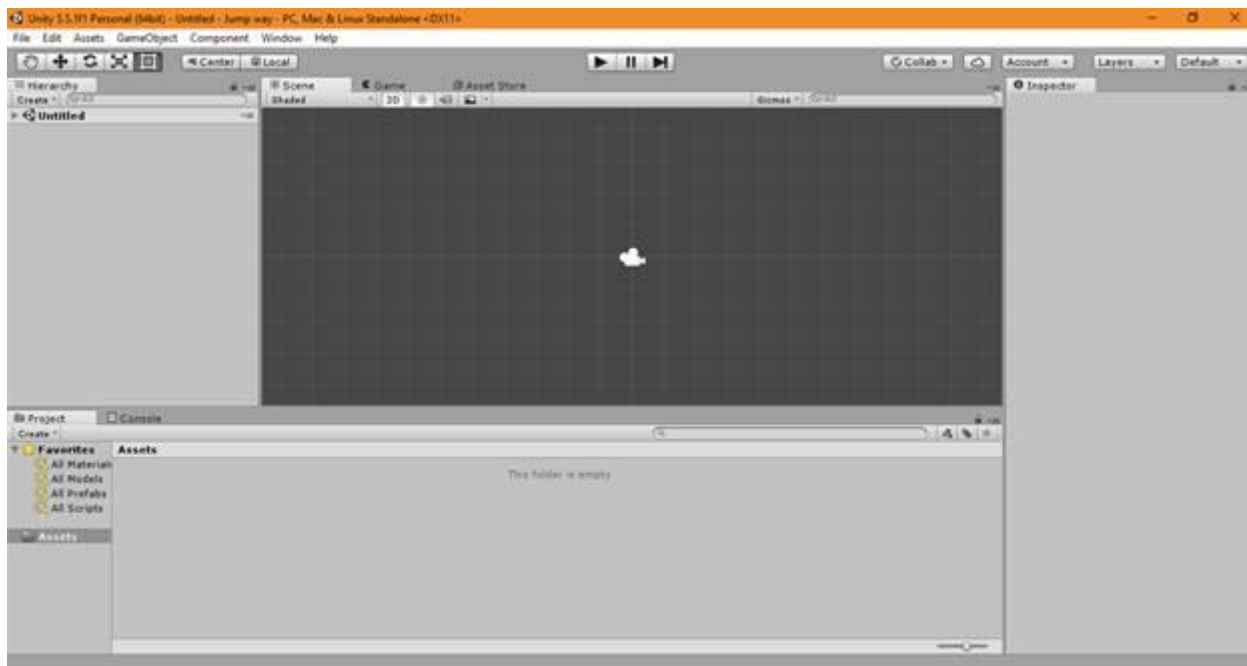


Рисунок 1 – Интерфейс Unity3D

Основные вкладки Unity

Вкладка Project представляет собой файловый менеджер для папки Assets. Папка Assets это основная папка проекта, в которой хранятся практически все файлы игры (аудио, изображения, скрипты).

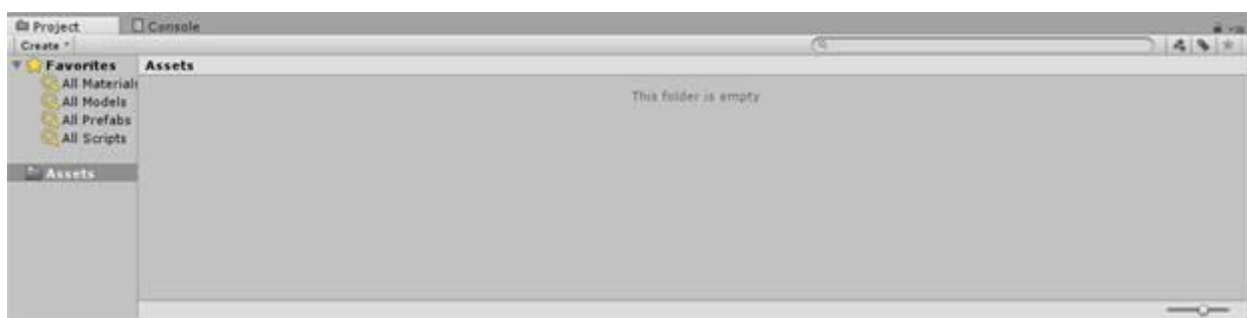


Рисунок 2 - Вкладка Project

Вкладка Console служит для вывода ошибок, или для вывода пользовательских сообщений из кода.

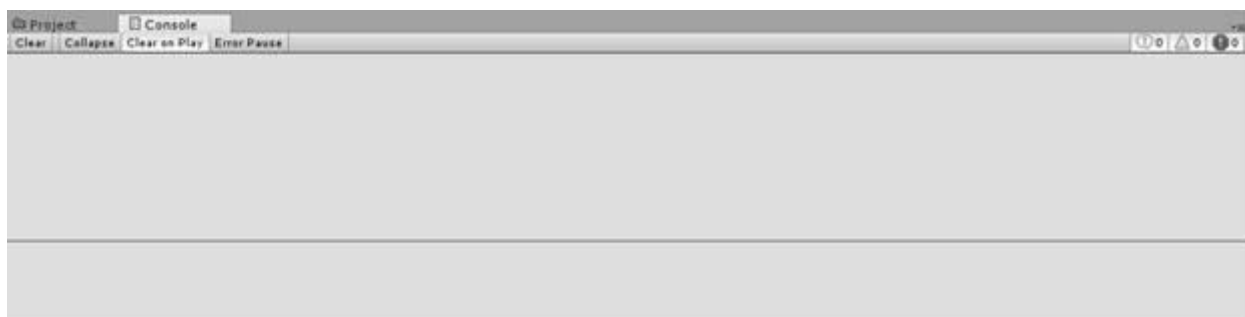


Рисунок 3 - Вкладка Console

Вкладка Hierarchy отображает все объекты, находящиеся в данный момент на данной сцене, кнопка create позволяет создавать объекты.



Рисунок 4 - Вкладка Hierarchy

Вкладка Inspector отображает характеристики выбранного объекта.



Рисунок 5 - Вкладка Inspector

Вкладка Scene отображает сцену, то есть все объекты как они будут отображаться в игре. По сути, это рабочая область. В ней можно передвигать объекты, изменять их размер и т.д. Вращать камеру можно с помощью правой клавиши мыши.

Вкладка Game отображает непосредственно игру. Она не позволяет проводить манипуляции с объектами и становится активной при запуске.



кнопка позволяет перемещаться по сцене



кнопка позволяет изменять размеры объектов



кнопка позволяет вращать объекты



кнопка позволяет изменять размеры объектов



кнопка позволяет проводить над объектами сразу все манипуляции (растягивать, перемещать и вращать)



кнопка запуска игры



кнопка паузы игры



кнопка служит для просмотра игры по кадрам

Кратко рассмотрим виды объектов:

- Клавиша Create Empty служит для созданий «Пустого объекта», обычно служит для группировки объектов.

- Клавиша Create Empty Child служит для созданий «Пустого дочернего объекта», про дочерние объекты будет описано ниже.

- 3D Object:

- Cube – куб
- Sphere – сфера
- Capsule – капсула
- Cylinder – цилиндр
- Plane – плоскость

- 2D Object – спрайты (то есть 2D текстуры)

- Light – объекты, издающие свет

- Audio – аудио компонент (например, музыка на заднем фоне)

- UI – компоненты пользовательского интерфейса, например, кнопки текст и другие.

- Camera – это камеры игры, точки с которых можно смотреть на игру, их можно установить несколько.

Дочерние объекты, наследуют свойство родительских объектов, например, при повороте родительского объекта дочерний так же будет поворачиваться, но дочерние объекты можно изменять без изменения родительских объектов.

Так же к объектам можно добавлять компоненты в панели Inspector. компоненты — это различные эффекты, физика, аудио и другие. Например,

компонент Box Collider позволяет указать что объект является твердым (он подходит для кубических объектов).

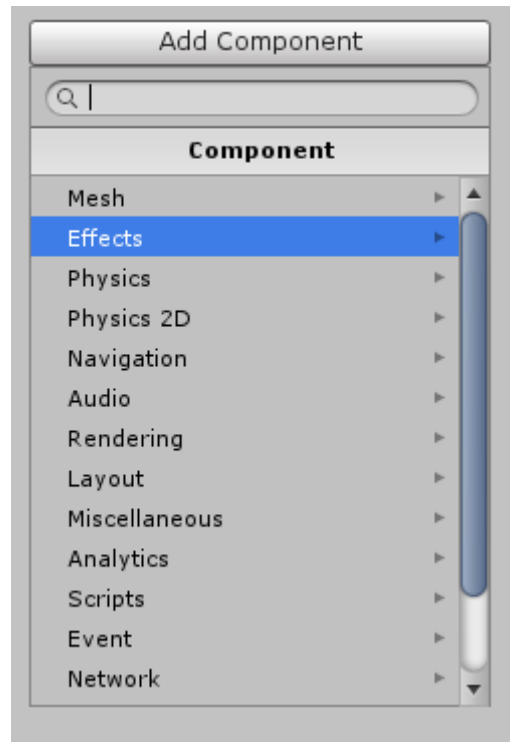


Рисунок 6 – Добавление компонентов к объекту


Последнее важное понятие — это понятие «Префаба» (Prefab). Префабы создаются из объектов. Для его создания нужно перетащить объект в папку Assets. После этого там появиться наш объект, который мы можем копировать на сцену. Префабы нужны для того что бы изменять свойства всех экземпляров нашего объекта. Например, мы перенесли 300 кубов на сцену, а потом решили изменить их цвет. Нам нужно изменить цвет каждого куба по-отдельности что не очень удобно. Для этого и служат префабы. Мы можем изменить цвет только префаба в папке assets и цвет измениться у всех его «экземпляров». Префаб это повторяющиеся вещи в игре, они являются клонами и занимают меньше памяти в игре.

Ссылка на официальную документацию Unity

(<https://docs.unity3d.com/ru/current/Manual/UnityManual.html>)

Практическая часть

Для создания тестовой игры будем использовать пример на официальном сайте Unity игру Roll-a-ball (<https://unity3d.com/ru/learn/tutorials/projects/roll-ball-tutorial>)

Запустите Unity и создайте новый проект нажатием на кнопку  , выберите путь, где будут храниться исходники игры. Введите имя игры. Выберем режим 3D и создайте проект.

После запуска проекта подготовьте файловую структуру проекта. В папке Assets создайте 4-ре папки:

- **_Scenes** – в этой папке будут храниться сцены.
- **Materials** – в этой папке будут храниться материалы.
- **Prefabs** – в этой папке будут храниться наши «префабы».
- **Scripts** – в этой папке будут храниться наши скрипты.

Сохраним нашу сцену, в папке сцен сочетанием клавиш Ctrl+S либо через меню File → Save Scene. Вводим любое название мы введем «Game».

Теперь создадим поле для игры. Нажмем кнопку Create в вкладке Hierarchy выберем 3D Object – Plane (плоскость).

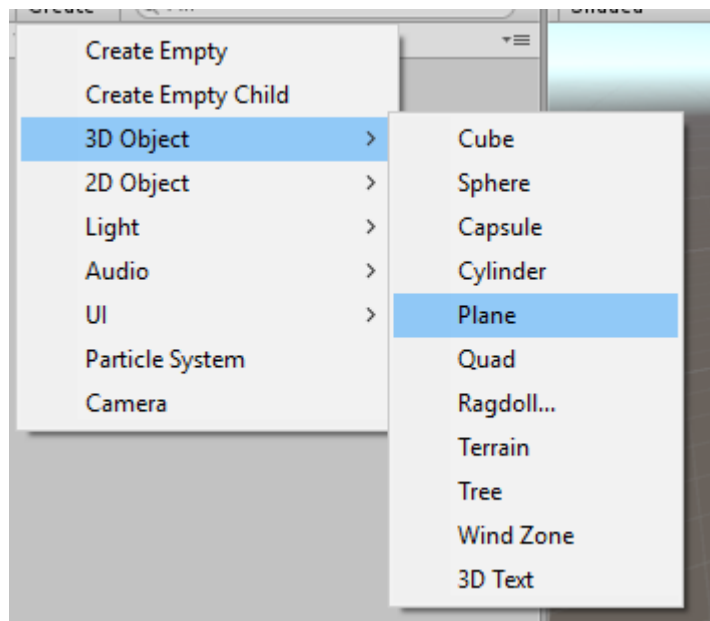


Рисунок 7 – Добавление плоскости на сцену

После этого должна появиться плоскость в окне Scene. Используя вкладку Inspector переименуйте область. Назовите ее «Ground» и нажмите клавишу Enter. Сбросим значение Transform используя кнопку Reset. Для отображения, выпадающего меню нажмите на значок «шестерёнки».

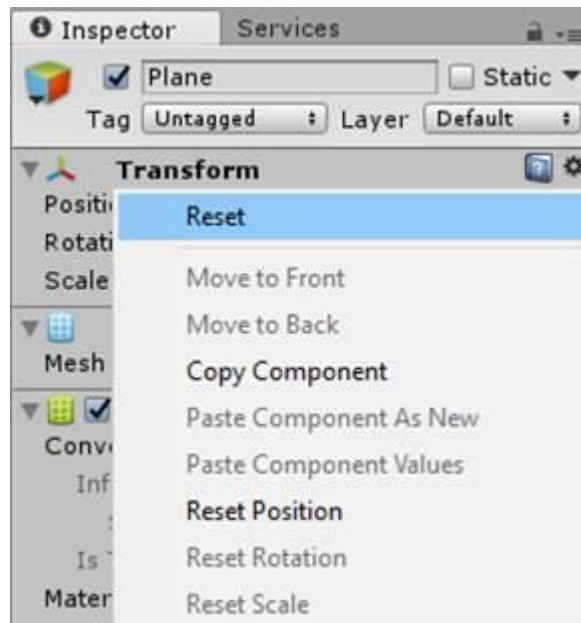


Рисунок 8 – Сброс пространственного размещения объекта

Рассмотрим вкладку Transform в инспекторе

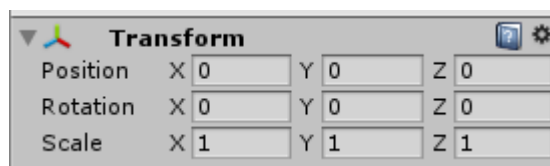


Рисунок 9 – Панель пространственного положения и ориентации объекта

- **Position** – отвечает за положение объекта в пространстве
- **Rotation** – отвечает за поворот объекта в пространстве
- **Scale** – отвечает за размер объекта

Теперь добавьте игрока на поле, в данном случае игроком будет выступать сфера. Аналогичным образом сбросьте значения transform для сферы. Переименуйте сферу в «Player» и установите свойство Position

значение 0.5 для того, чтобы наша сфера соприкасалась с плоскостью как на рисунке.

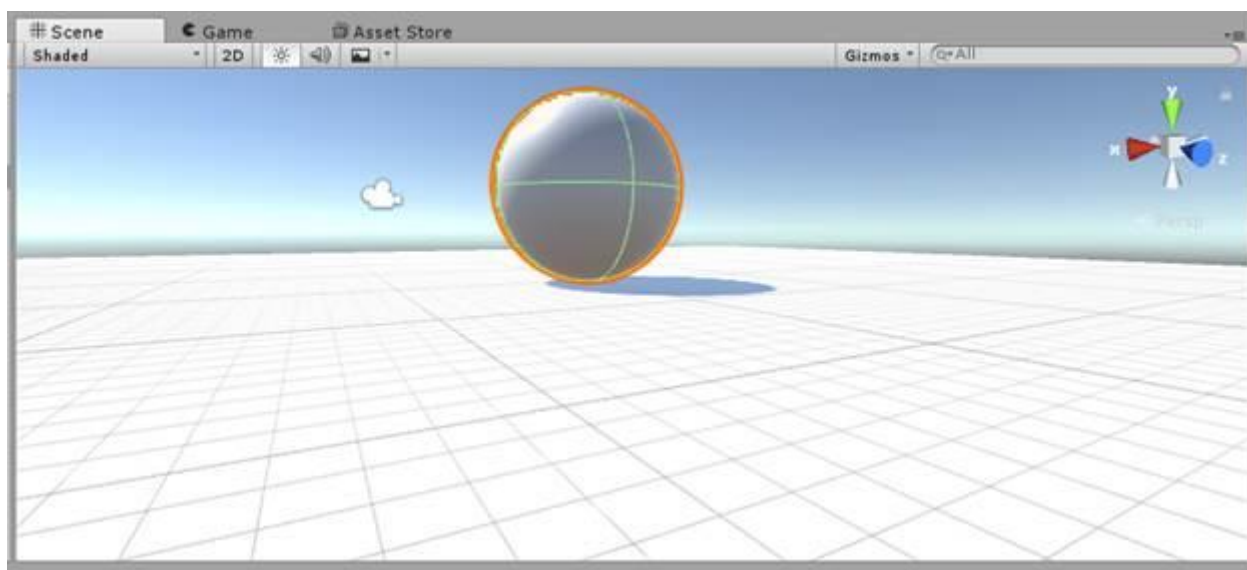


Рисунок 10 – Положение сферы (игрока) на сцене

Так же можно изменять свойства transform для объекта Directional Light, который отвечает за свет. Самостоятельно добейтесь результата, представленного на рисунке 11.

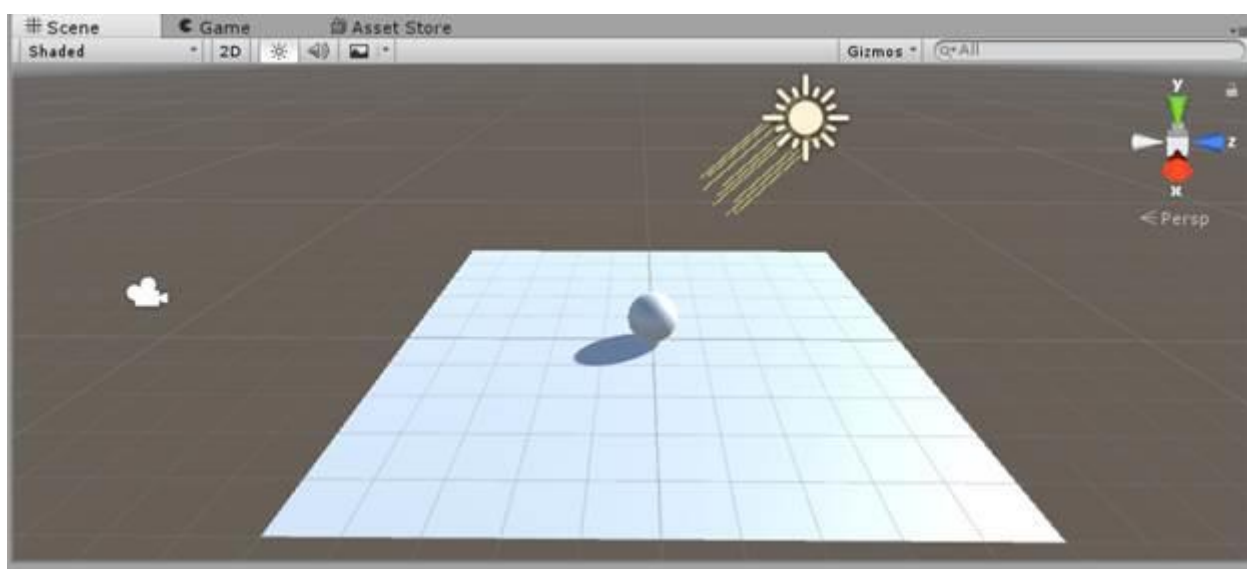


Рисунок 11 – Настройки освещения

Теперь измените цвет земли и сферы. Для этого в папке Materials добавьте новые материалы. В папке нажмите правой кнопкой мышки Create → Material.

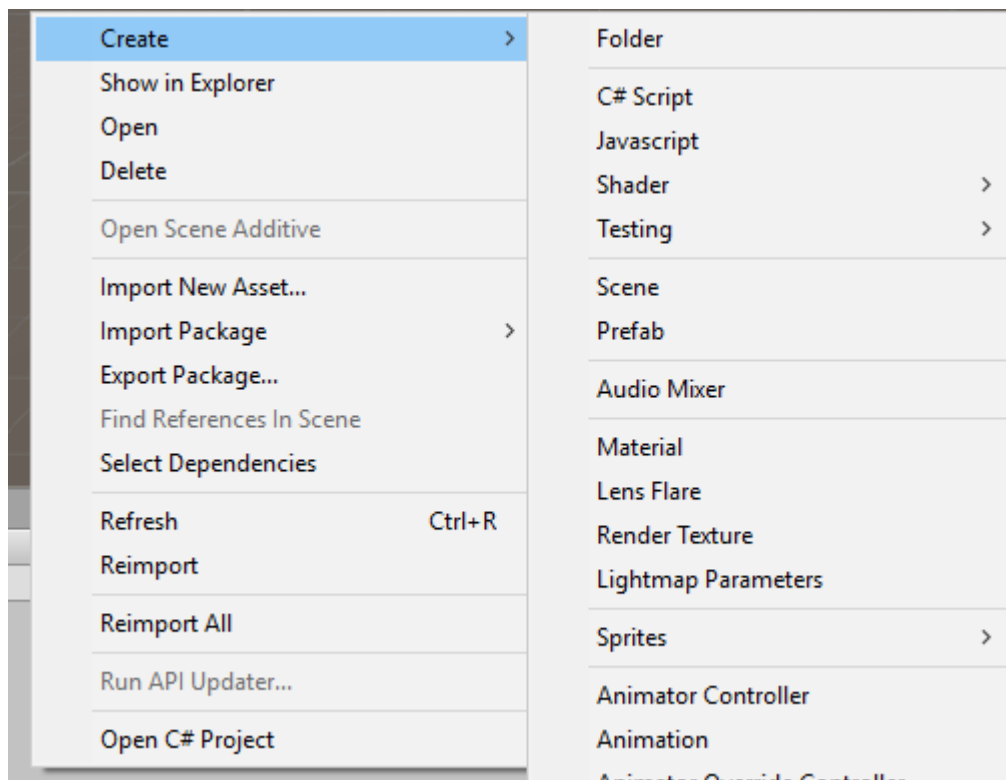


Рисунок 12 – Меню добавления материала

В папке появится новый материал. Введите для него любое имя.

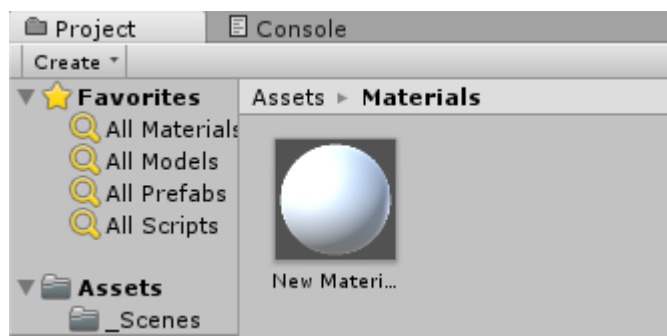


Рисунок 13 – Отображение добавленного материала

Теперь выберите для него цвет. Для этого выберите элемент и в вкладке инспектора установите свойству Shader значение Unlit → Color.

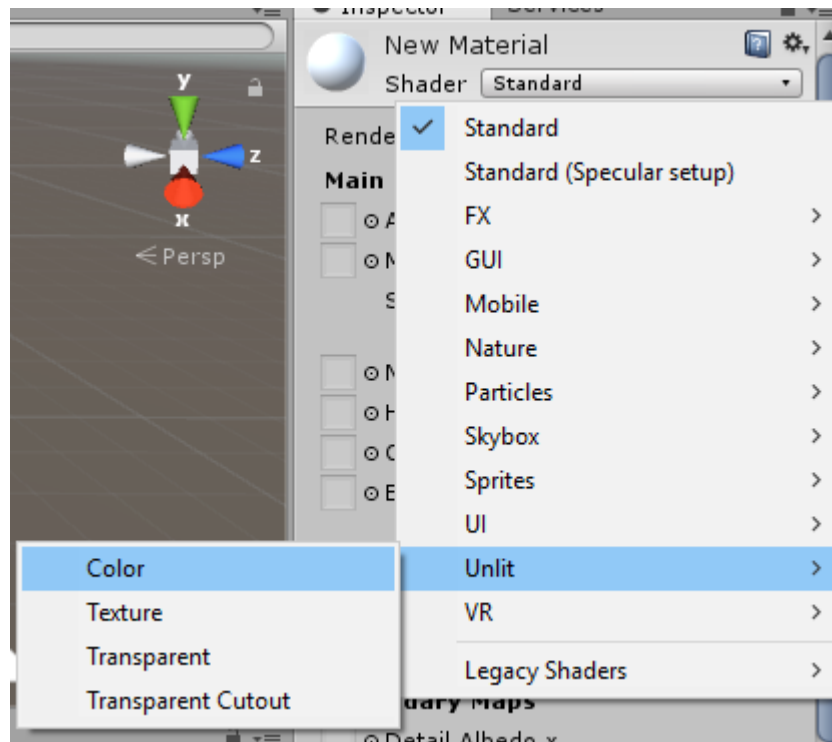


Рисунок 14 – Установка типа шейдера для нового материала

После этого в Main Color выберите нужный цвет. Теперь перетащите созданный материал на нужный нам объект во вкладке Hierarchy, либо выберите целевой объект и перетащите материал в инспектор. При этом в инспекторе целевого объекта появится добавленный материал

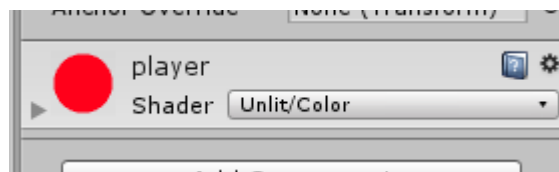


Рисунок 14 – Установленный материал (шейдер) у объекта

В результате должно получиться нечто подобное.

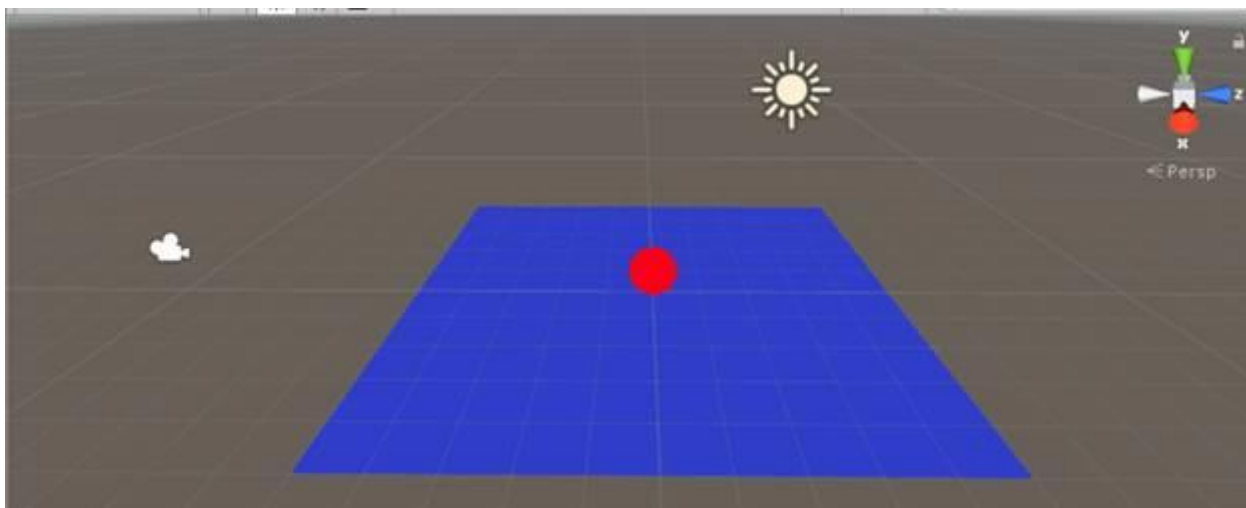


Рисунок 15 – сцена с установленными для объектов материалами

Проверьте чтобы у сферы и у земли были установлены свойства Collider (галочка рядом с именем должна быть включена). Эти свойства указывают что эти объекты твердые.

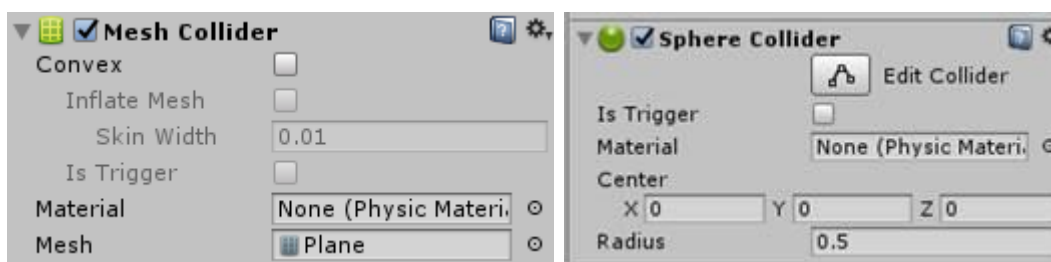


Рисунок 16 – Установленный компоненты Collider

Теперь сфере добавьте новый компонент Physics → Rigidbody. Этот компонент показывает, что элемент подчиняется законам физики. Это нужно для того чтобы объект мог двигаться по земле.

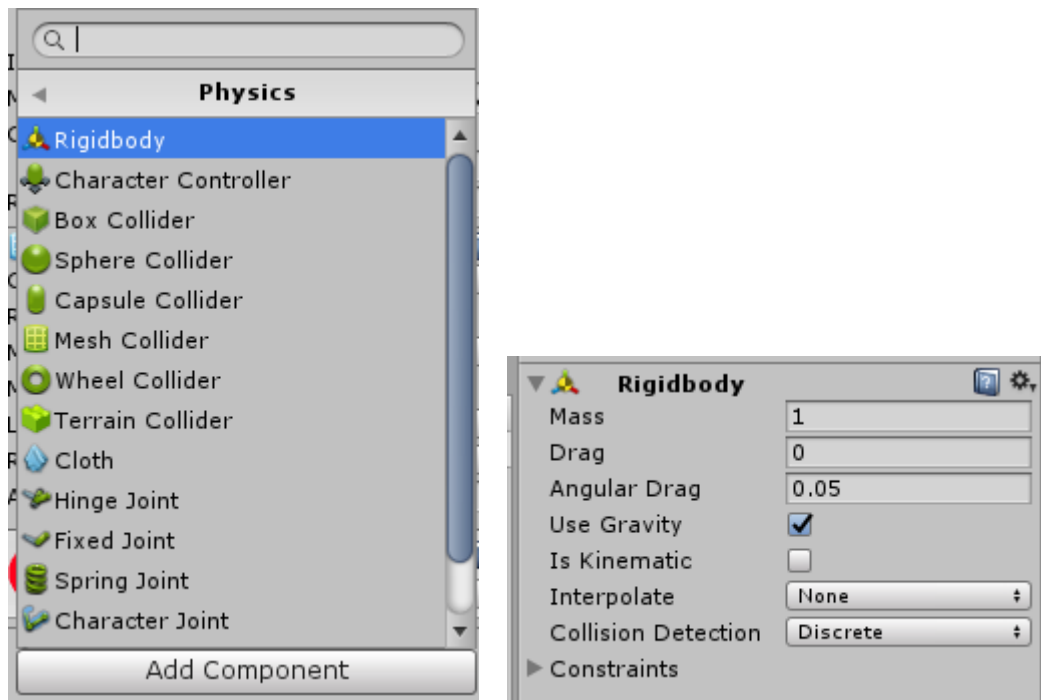


Рисунок 16 – Добавление компонента Rigidbody

Следующим шагом необходимо создать логику управления сферой. Для этого добавьте в папку Scripts скрипт C#. Назовите скрипт PlayerController. Привяжите скрипт к объекту (перетаскиванием на объект). Теперь откройте созданный скрипт двойным кликом.

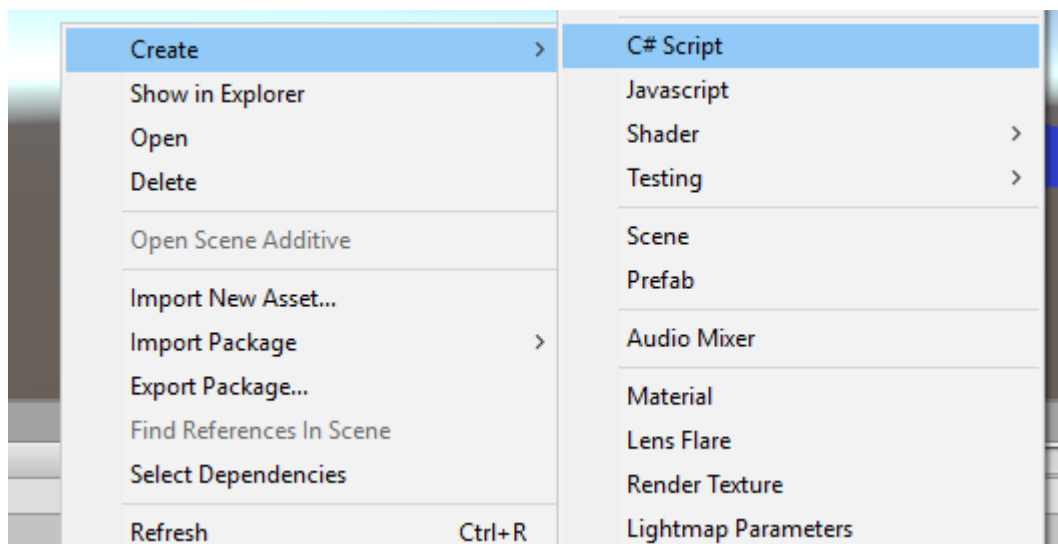


Рисунок 17 – Создание C# скрипта

Класс `PlayerController` наследуется от класса `MonoBehaviour` который является основным классом в Unity.

Функция `Start` вызывается один раз в начале вместе с первым кадром сцены.

Функция `Update` вызывается каждый кадр в сцены.

В Unity существует множество функций, но это самые популярные из них.

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

public class PlayerController : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }

}
```

Изменим код класса на следующий:

```
public float speed;
private Rigidbody rb;
void Start () {
    rb = GetComponent<Rigidbody>();
}
private void FixedUpdate()
{
    float moveHorizontal = Input.GetAxis("Horizontal");
    float moveVertical = Input.GetAxis("Vertical");
    Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);
    rb.AddForce(movement * speed);
}
```

Переменная «`speed`» отвечает за скорость перемещения. Переменная «`rb`» отвечает за хранение компонента «`Rigidbody`» данного объекта, рассмотренного ранее. Получение компонента происходит в методе `Start`.

Функция `FixedUpdate` аналогична функции `Update`, но вызывается не каждый кадр, а через определенное время.

`Input.GetAxis("Horizontal")` Получает от пользователя нажатии стрелки вправо или влево. Принимает значение от 1 до -1. Вправо 1, влево -1. Увеличение происходит постепенно.

`Input.GetAxis("Vertical")` Получает от пользователя нажатии стрелки вниз или вверх аналогично `Input.GetAxis("Horizontal")`

`Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);` В этой строке мы создаем вектор направления движения. `Vector3` содержит три координаты так как мы работаем в трехмерном пространстве, существует так же `Vector2`.

`rb.AddForce(movement * speed);` В этой строке нашему элементу добавляется сила вдоль направления вектора которая приводит к движению.

Теперь в Unity в инспекторе нашей сферы должно появиться поле для задания скорости (любые публичные переменные или поля будут отображаться в этой области). Введите значение 6.

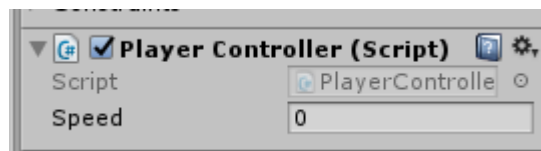


Рисунок 18 – Публичное поле класса (скрипта) в инспекторе объекта

После этого запустите игру и проверьте, если все сделано сфера должна перемещаться с помощью стрелок.

Теперь будем осуществлять работу с камерой, установим камеру чтобы область в окне Game выглядела как на рисунке 19.

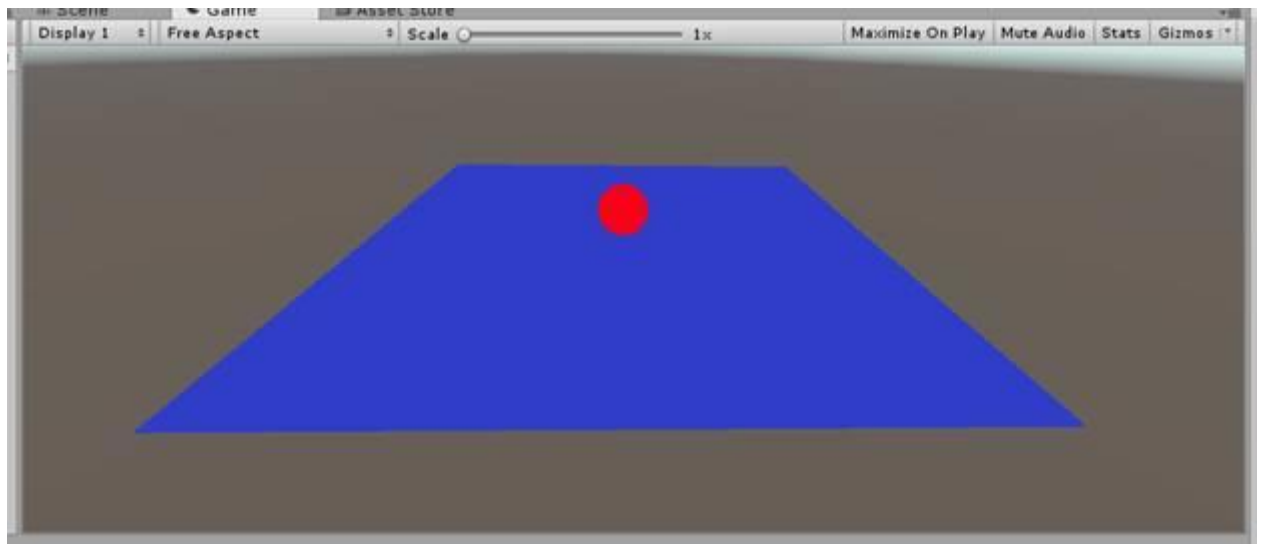


Рисунок 19 – Правильное положение камеры для этого проекта

Теперь привяжите нашу камеру к сфере. Для этого создайте новый скрипт и назовите его Camera и прикрепите его к нашей камере. Код скрипта следующий:

```
public GameObject player;
private Vector3 offset;
// Use this for initialization
void Start ()
{
    offset = transform.position - player.transform.position;
}
// Update is called once per frame
void LateUpdate ()
{
    transform.position = player.transform.position + offset;
}
```

`GameObject player;` хранит объект сферы

`offset = transform.position - player.transform.position;` получаем отступ от камеры до объекта отнимая позицию объекта от позиции камеры

`transform.position = player.transform.position + offset;` меняем позицию камеры устанавливая позицию объекта плюс отступ.

Функция LateUpdate вызывается в конце кадра после всех функций Update.

Не забудьте в инспекторе передать объект сферы классу камеры.

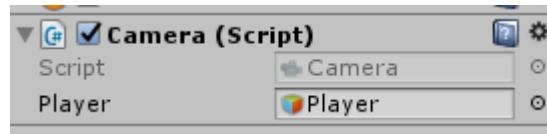


Рисунок 20 – Связанные объекты

Теперь самостоятельно создайте стены нашей области, как представлено на рисунке 21, используя элементы Cube для того чтобы сфера не выпадала за пределы области.

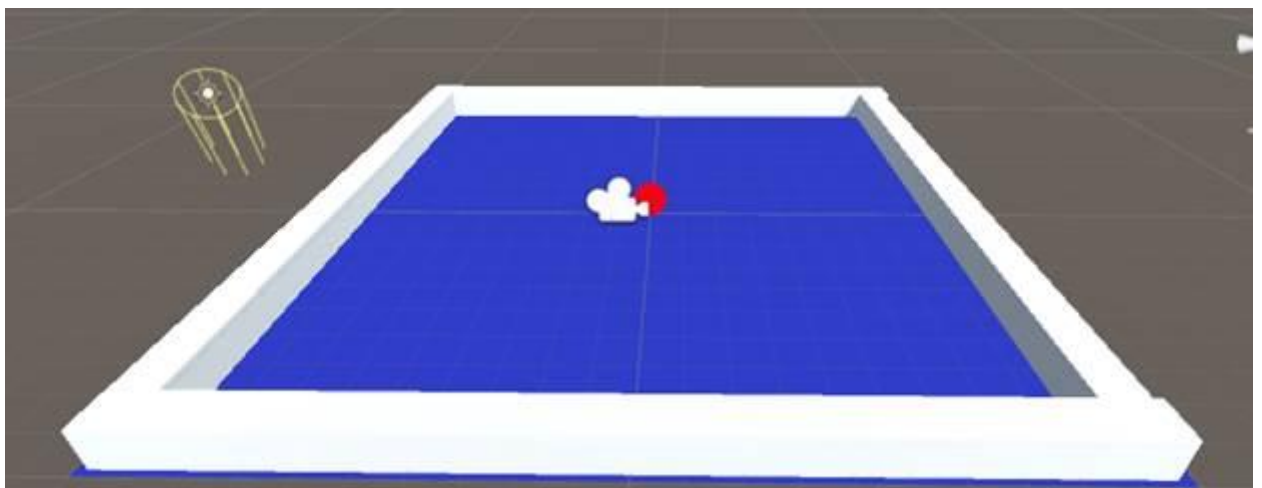


Рисунок 21 – Финальная сцена игры

Создайте новый куб, задайте ему размеры как представлено на рисунке 22.

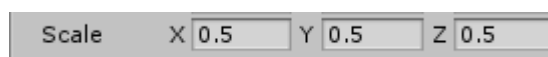


Рисунок 22 – настройки масштаба куба

Добавьте новый скрипт Rotate. Скрипт нужен для анимации кубиков. Перенесите скрипт на только что добавленный куб и откроем его. Добавьте код для вращения кубика.

```
void Update()
{
    transform.Rotate(new Vector3(15, 35, 28) * Time.deltaTime);
}
```

Функция Rotate производит вращение объекта по заданному вектору. Вектор умножается на время для того чтобы вращение происходило плавно. Делая вращение зависимым от времени, а не от кадров.

Сделайте из нашего кубика префаб, перетащив его в папку Prefabs. Создайте пустой объект и назовите его Cubes. Теперь используя созданный префаб добавьте в наш пустой еще несколько кубиков. А также уже созданный куб перенесите в пустой объект (рисунок23).

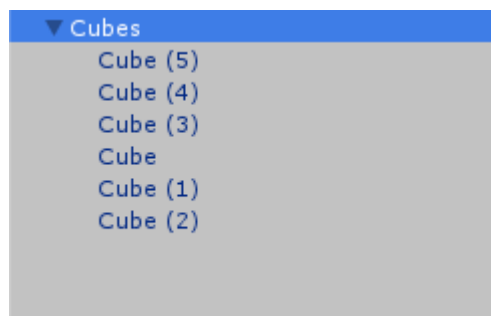


Рисунок 23 – Иерархия объектов (кубов)

Теперь создайте новый материал как мы это делали ранее. Выберите любой цвет. Сделайте активным наш префаб в вкладке project и добавьте ему материал, перетащив его в инспектор исходного куба для префаба. Если вы все сделали правильно все кубики изменят цвет (рисунок 24).

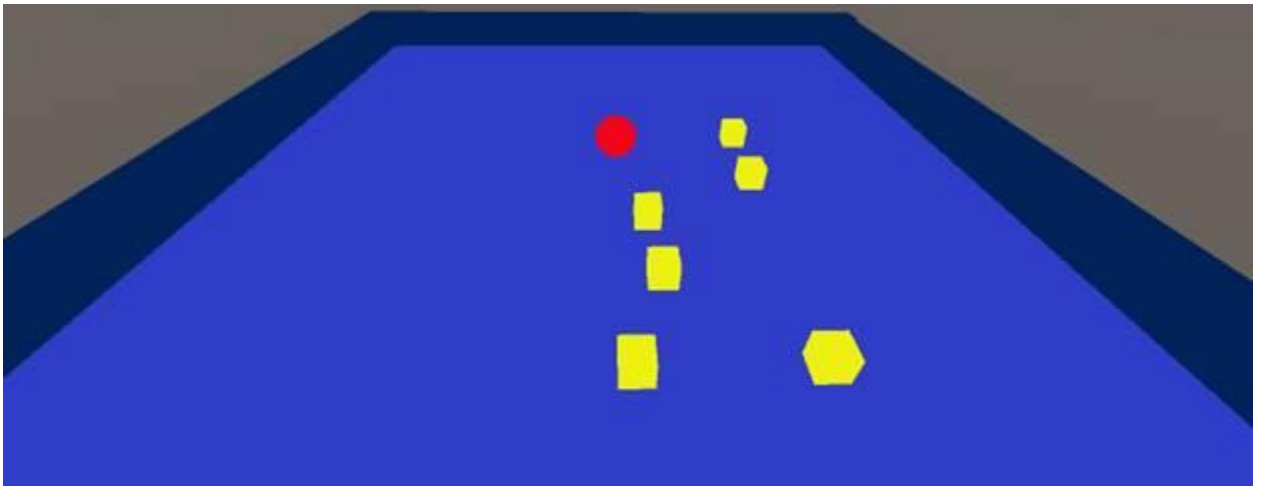


Рисунок 24 – готовые кубы

Нашему префабу в инспекторе установим галочку Is Trigger. Эта галочка позволяет другим элементам проходить сквозь этот элемент, но при этом отслеживать соприкосновения с ним (если просто снять галочку Box Collider, то мы так же сможем проходить сквозь элемент, но не сможет отследить соприкосновение)

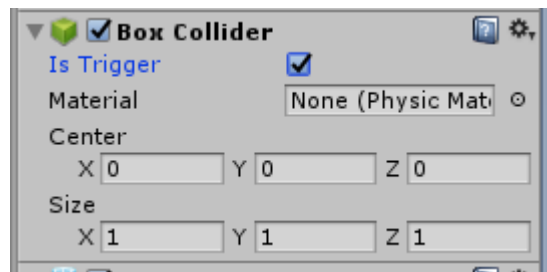


Рисунок 25 – Установленное свойство Is Trigger

Выберите наш префаб и в инспекторе нажмите Tag → add Tag

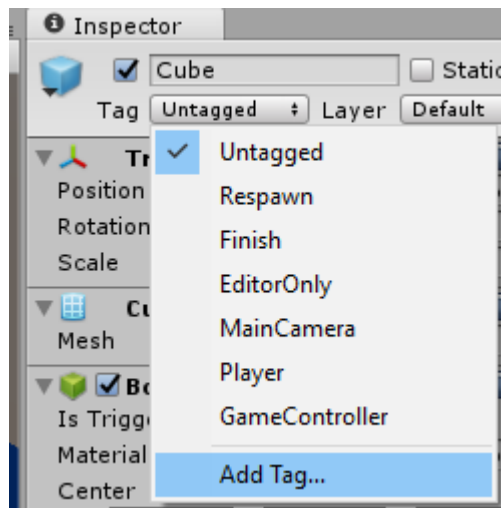


Рисунок 25 – Добавление нового тега объектов

В открывшемся окне нажмите плюс и введите имя тега «cubes» (без кавычек) и нажмите save снова выберите наш префаб и установите ему только что добавленный тег.

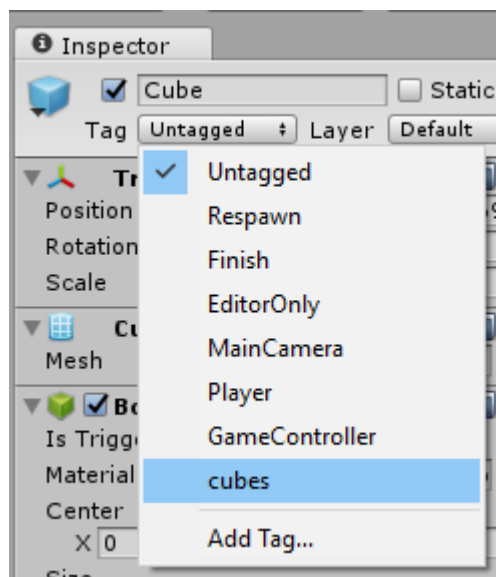


Рисунок 26 – Добавление тега объекту Куб

Откроем скрипт PlayerController и добавим следующий код

```
private void OnTriggerEnter(Collider other)
{
    if (other.tag == "cubes")
        Destroy(other.gameObject);
}
```

Функция `OnTriggerEnter` отслеживает прикосновение с объектом, в качестве параметра принимает объект, с которым произошло соприкосновение. Потом мы проверяем если это объект с тегом `cubes` то удаляем его. Теперь мы можем собирать кубики. Проверьте это, запустив игру.

Реализуем механизм подсчета собранных кубиков и вывод надписи после сбора все кубиков.

Создадим текстовое поле `Create` → `UI` → `Text`. Выбираем `Text` и изменим его имя на `CountText`, а затем нажимаем на квадрат в инспекторе.

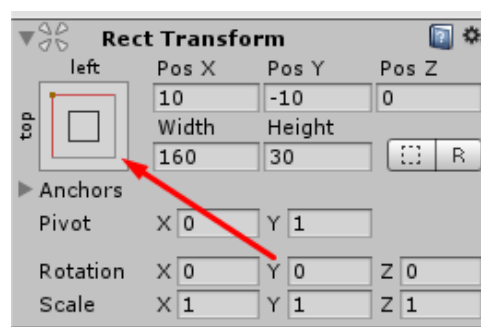


Рисунок 27 – Настройки положения объекта

После этого появиться окно выбора положения, зажимаем `Alt+Shift` и выбираем квадрат, указанный на рисунке 28. Текст должен закрепиться в левом верхнем углу экрана.



Рисунок 28 – Выбор места закрепления объекта

После этого введите значения в поля как на рисунке. Так же ниже вы можете изменить размер шрифта, цвет и стиль.

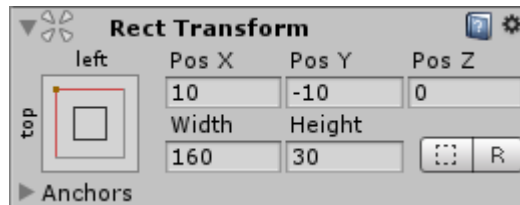


Рисунок 29 – Размеры надписи

Аналогичным образом добавьте еще один текст только сделайте выравнивание по центру экрана. Назовите его «WinText».

Затем откройте скрип PlayerController. В блок using добавьте

```
using UnityEngine.UI;
```

Так же в класс добавьте следующие поля

```
public Text countText, winText;
```

Вернемся в Unity и добавим ссылку на тексты в это текстовое поле, в инспекторе объекта сферы



Рисунок 30 – Связывание UI Объектов и параметров скрипта

Возвращаемся к скрипту PlayerController изменим код класса на следующий

```
public float speed;
private Rigidbody rb;
/// количество собранных кубов
private int count;
/// Изначальное количество кубов
private int countCoub;
/// текстовые поля
```

```

public Text countText, winText;
void Start () {
    rb = GetComponent<Rigidbody>();
    count = 0; /// по умолчанию собрано 0 кубов
    winText.text = ""; /// текст победы по умолчанию скрыт
    /// получаем количество элементов с тегом cubes
    countCoub = GameObject.FindGameObjectsWithTag("cubes").Length;
    setCount();
}
private void FixedUpdate()
{
    float moveHorizontal = Input.GetAxis("Horizontal");
    float moveVertical = Input.GetAxis("Vertical");
    Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);
    rb.AddForce(movement * speed);
}
private void OnTriggerEnter(Collider other)
{
    if (other.tag == "cubes")
    {
        Destroy(other.gameObject);
        count++;
        setCount();
    }
}
/// <summary>
/// Функция изменения текстов
/// </summary>
private void setCount()
{
    countText.text = "Количество: " + count.ToString();
    if(count >= countCoub)
        winText.text = "Победа!!!";
}

```

Самостоятельно попробуйте добавить текст, в котором будет написано сколько кубов еще нужно собрать как на рисунке.

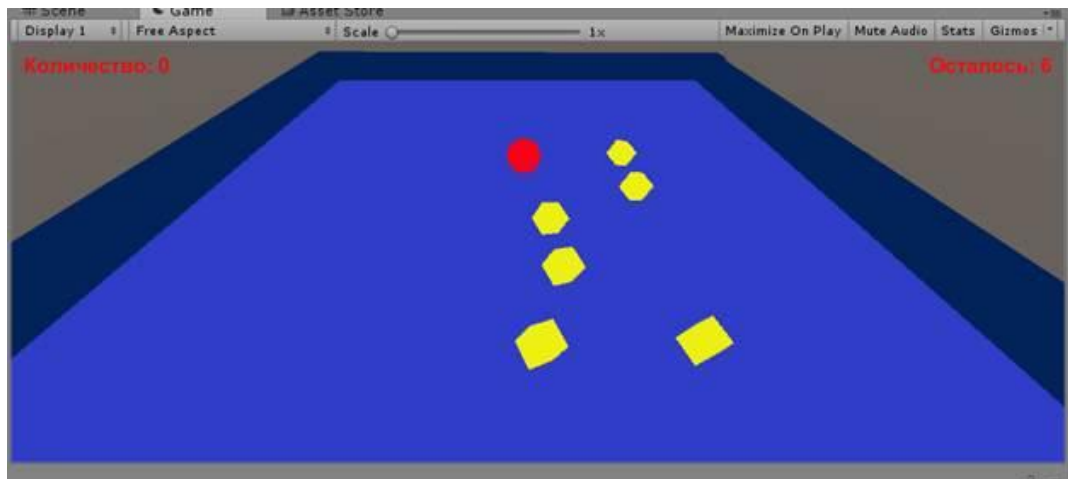


Рисунок 31 – Финальная сцена игры

Теперь сохраним сцену (Ctrl+S) и зайдем в File → Build Setting откроется окно (рисунок 32).

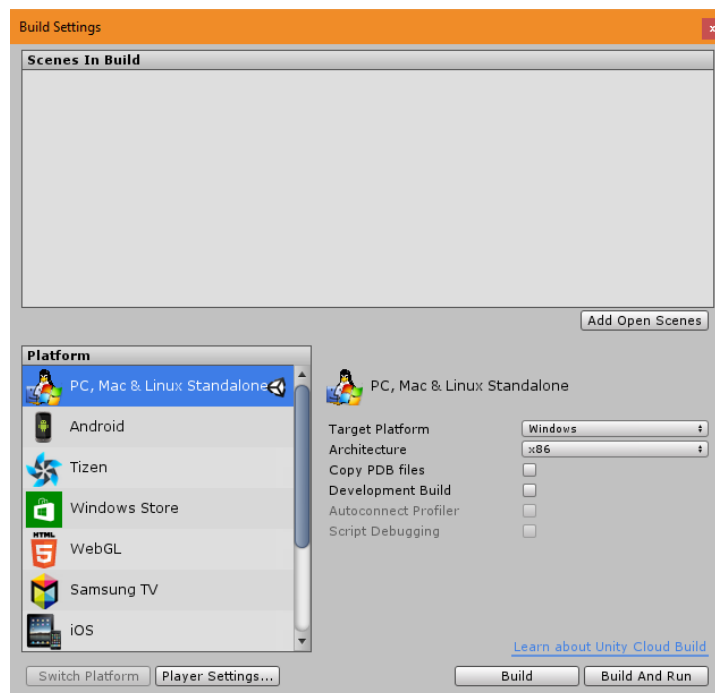


Рисунок 32 – Окно сборки проекта

Нажмите на кнопку Add Open Scenes чтобы добавить открытые сцены в игру. Выберите платформу Windows нажмите клавишу Build и выберите путь где будет сохранена наша игра. После этого вы можете запустить данную игру через exe файл.