

Лабораторная работа №1: Катящийся шар.

Целью работы является проверка остаточных знаний.

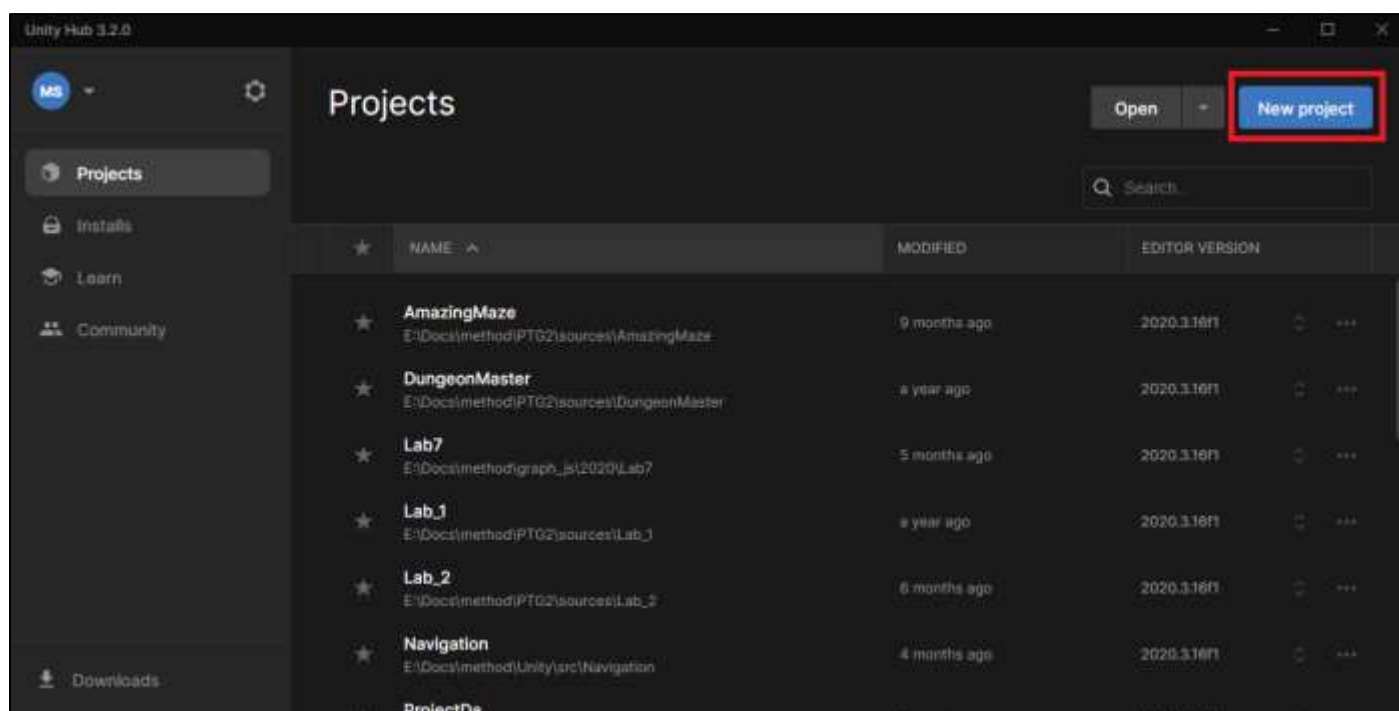
Примечание: используемая версия Unity Hub – 3.2.0, Unity – 2021.3.8f1

Оглавление

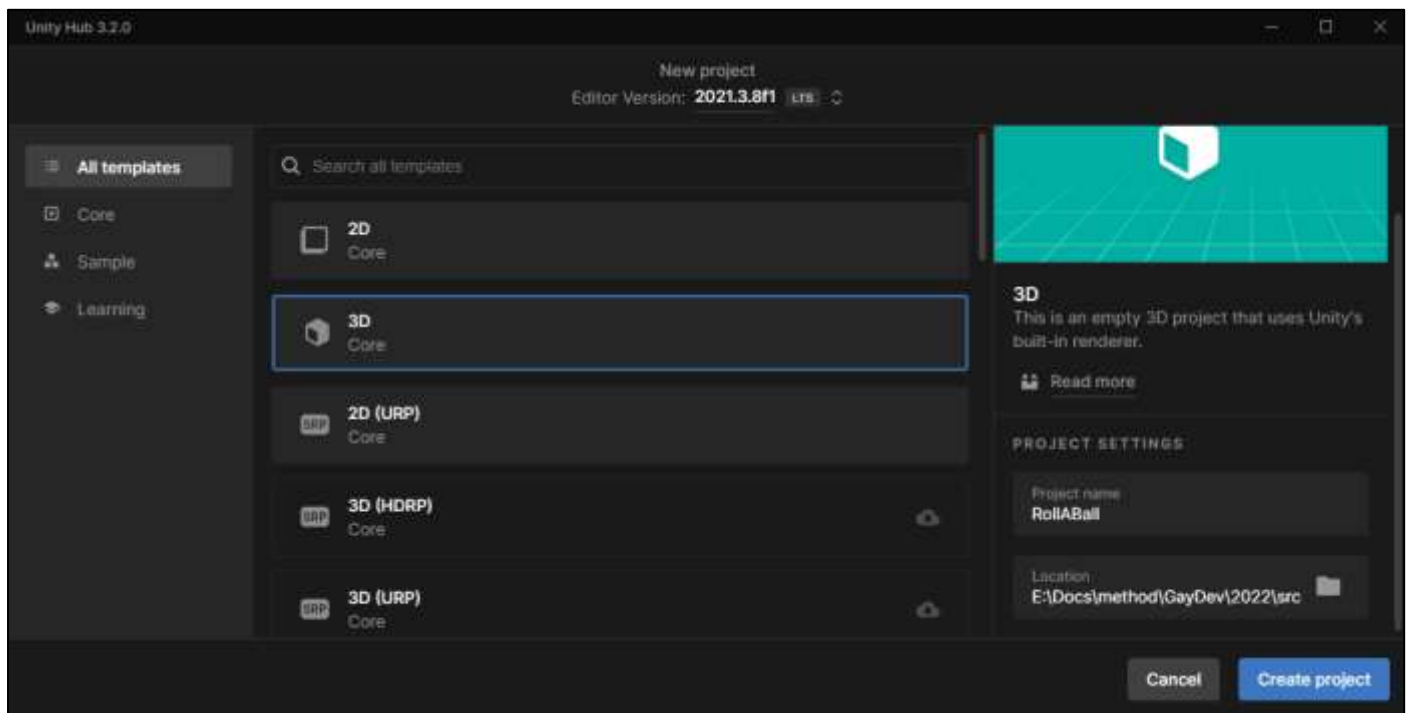
Создание проекта.	1
Начало работы.	2
Перемещение сферы.	5
Следящая камера.	8
Собираемые объекты.	10
Подсчёт собранных объектов и сообщение о победе.	14
Задания:.....	16
Отчёт.	16

Создание проекта.

В Unity Hub, перейдите в раздел Projects и нажмите New project:

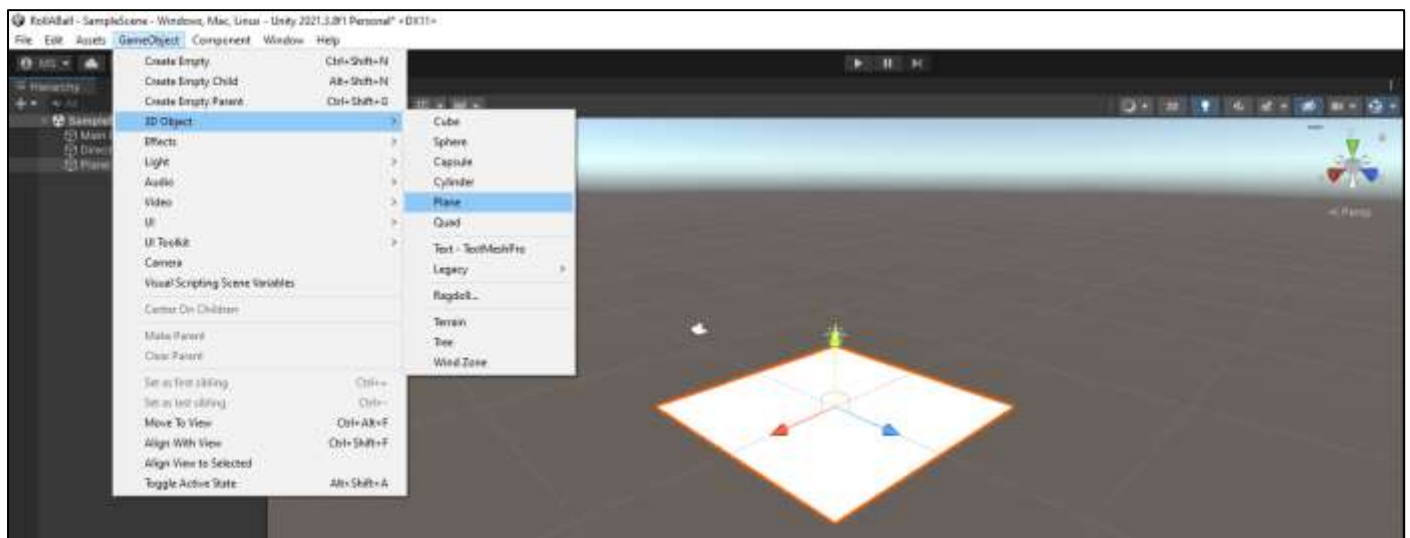


В появившемся окне, выберите тип проекта 3D, имя проекта и директорию сохранения:



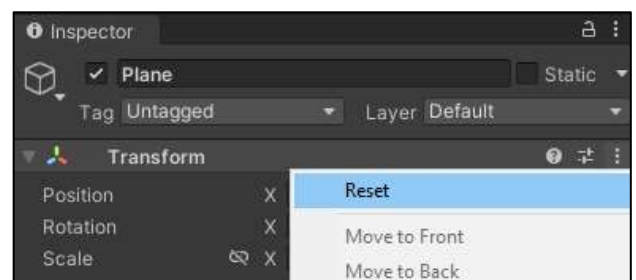
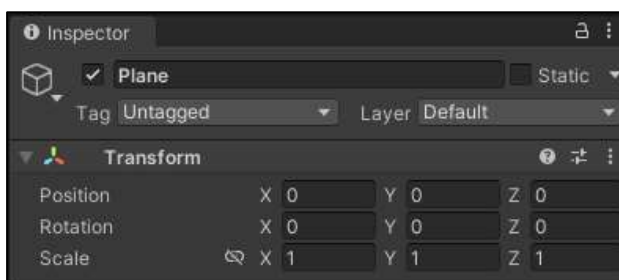
Начало работы.

Добавьте в сцену плоскость, выбрав GameObject -> 3D Object -> Plane (альтернативный вариант, кликнув правой кнопкой по области Hierarchy):



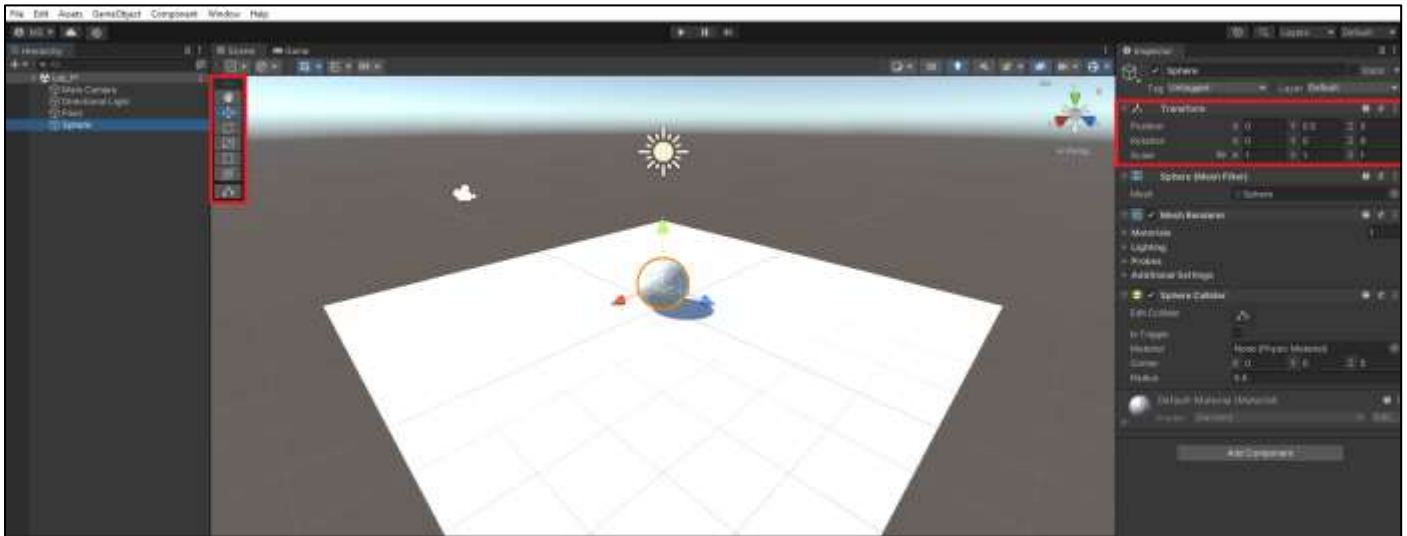
Примечание: по умолчанию, каждый проект имеет сцену с названием SampleScene. Для удобства дальнейшей работы с проектом, сцену имеет смысл переименовать. Плоскость имеет смысл назвать “Пол” или “Земля”.

Обратите внимание на координаты, масштаб и вращение плоскости при её появлении:

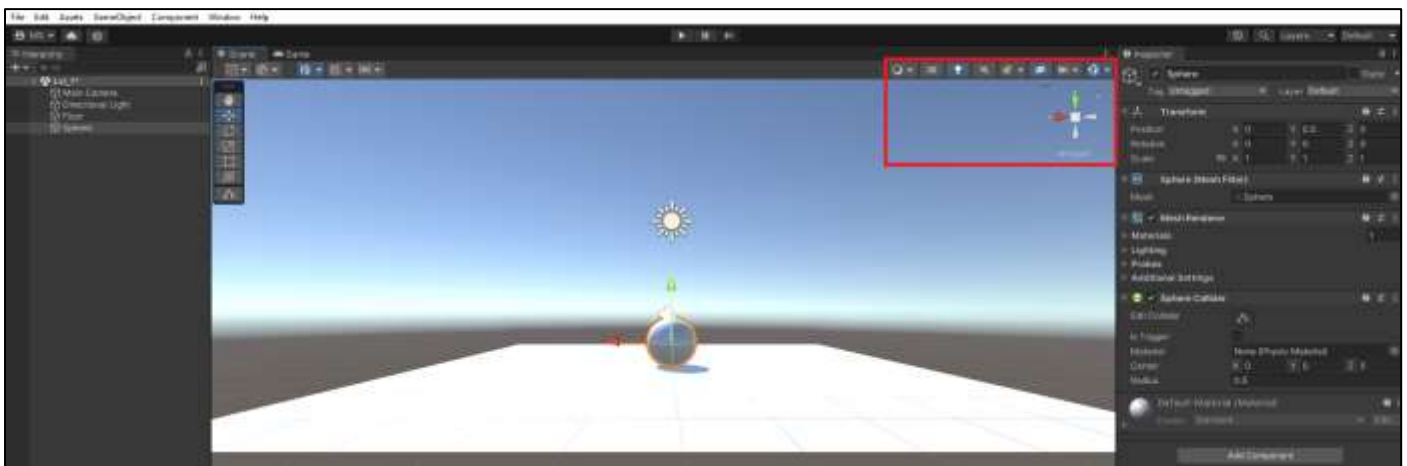


В случае, если необходимо вернуться к начальным значениям, всегда можно их сбросить, воспользовавшись командой Reset.

Добавьте в сцену сферу, и разместите её над плоскостью. Позицию, размер и поворот сферы можно задать, используя инструменты слева сверху, или параметры, расположенные в инспекторе:

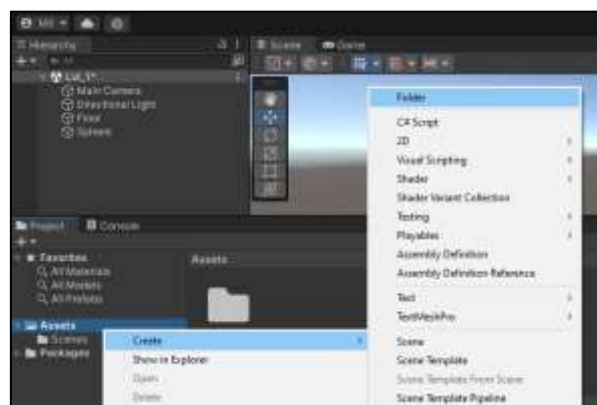


Для удобства позиционирования объектов, можно использовать инструменты работы со сценой:

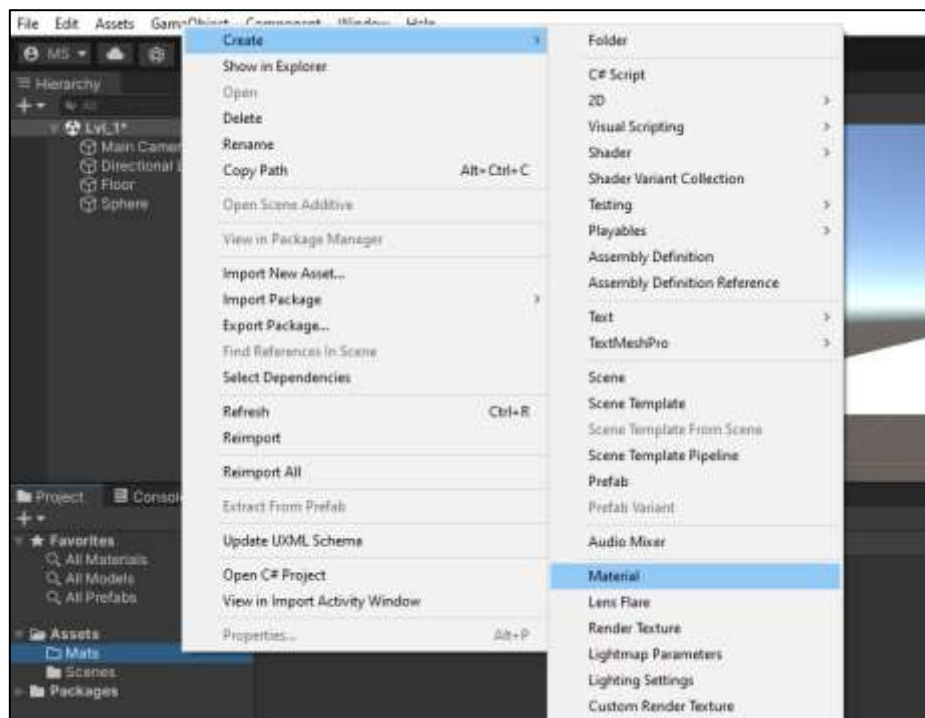


Примечание: для того, чтобы сфокусировать камеру на объекте, кликните по нему дважды в окне Hierarchy, либо выберите объект и нажмите F.

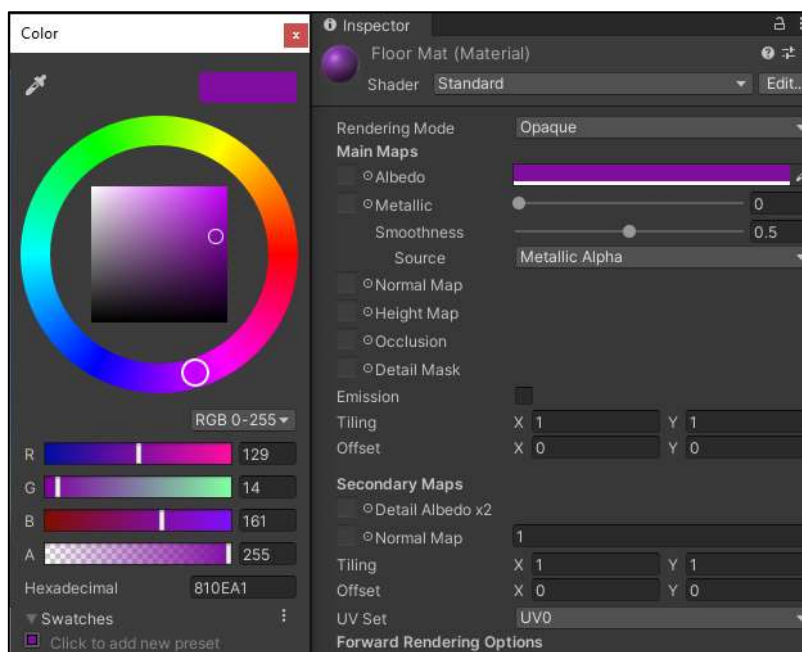
Для того, чтобы добавить объектам цвет, необходимо создать материалы. Создайте новую папку в разделе Project, кликнув правой кнопкой мыши на Assets и назовите её “Материалы”:



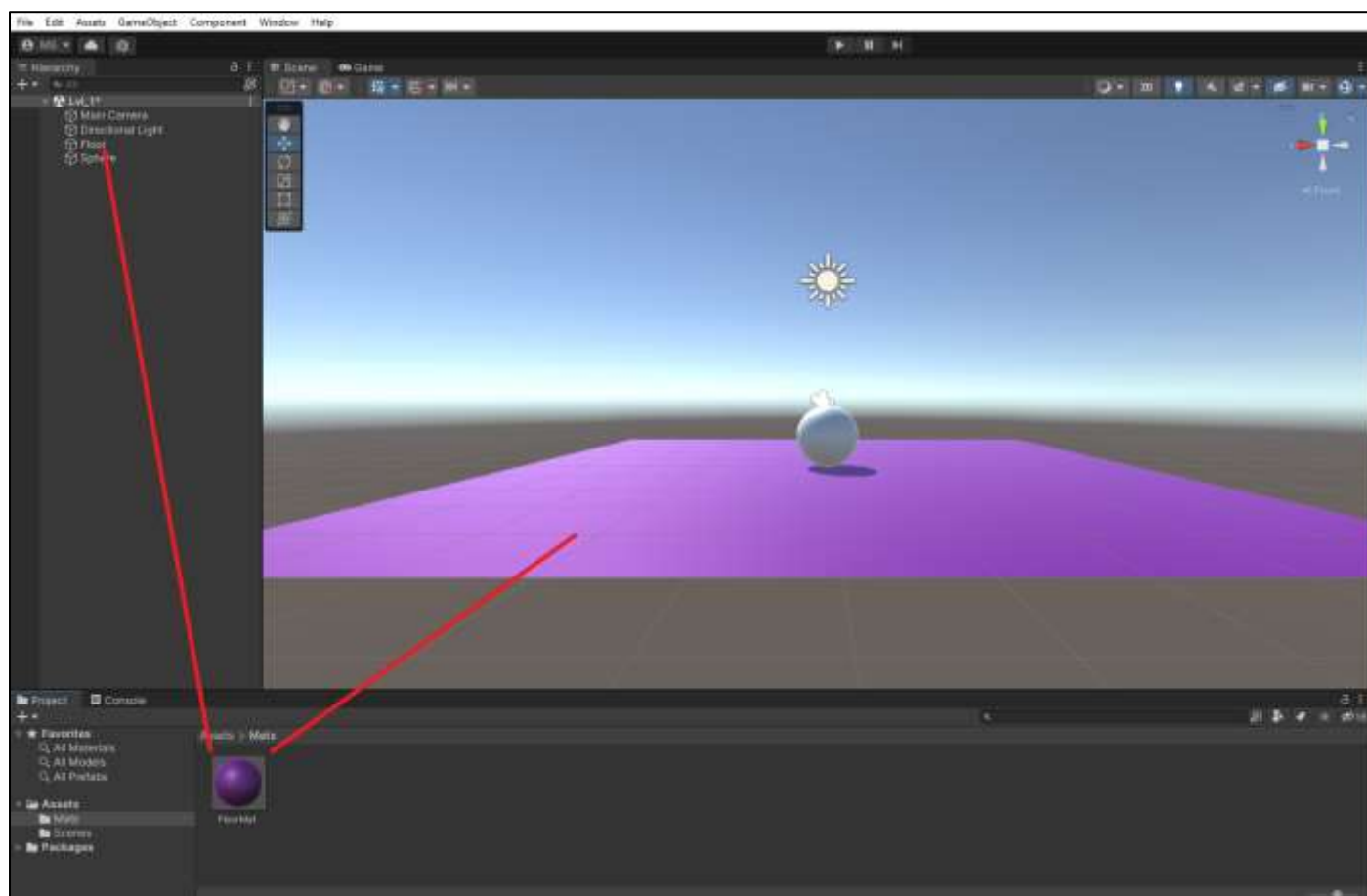
Перейдите в созданную папку и создайте новый материал, используя меню Create или правый клик в области папки:



Присвойте материалу осмысленное имя, после чего, в Inspector, выберите Albedo и назначьте материалу цвет:

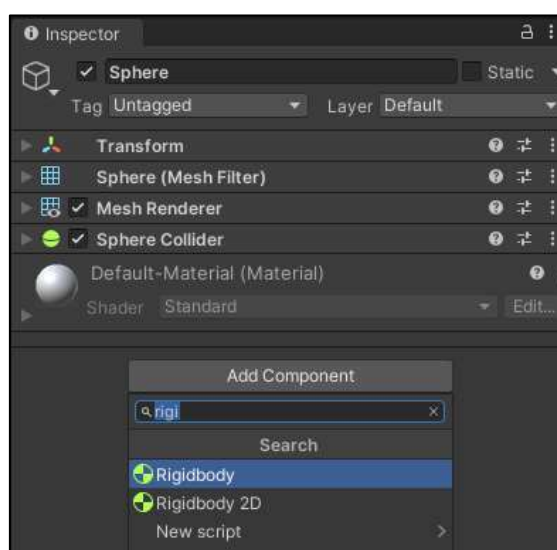


Назначить материал можно выбрав его в разделе Project и перетаскив на соответствующий объект либо в сцене, либо в разделе Hierarchy:



Перемещение сферы.

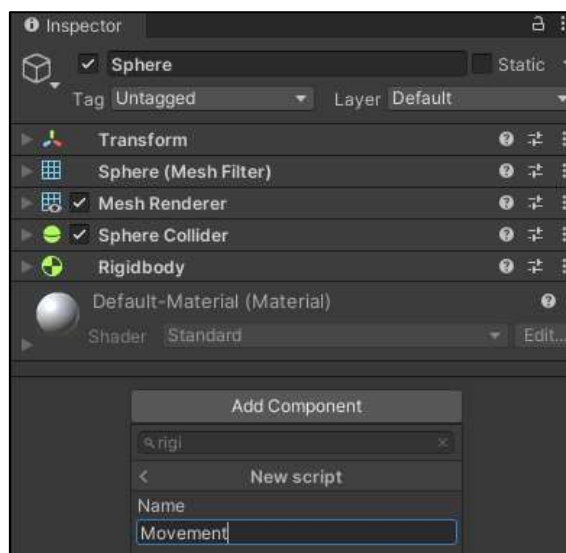
Существует пять способов перемещения объектов в Unity. В данном случае, предполагается использовать перемещение сферы при помощи задания воздействующей на неё силы. Для этого, необходимо выбрать сферу, затем в разделе Inspector, выбрать Add Component -> Physics -> Rigidbody (или воспользоваться поиском):



Если всё сделано правильно, сфера должна получить физическую модель.

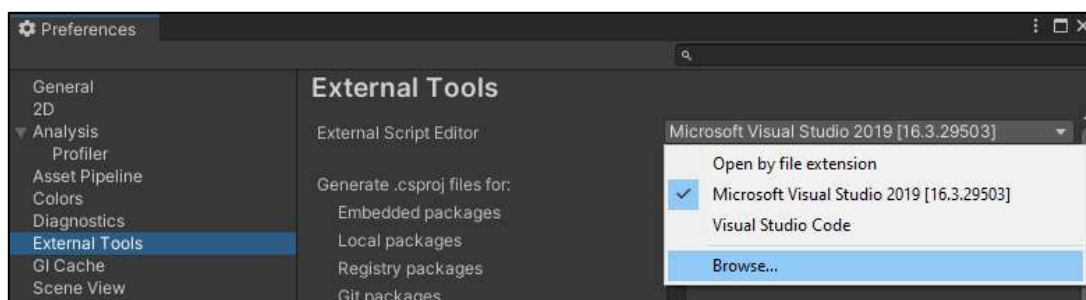
Перед переходом к следующему этапу, создайте новую папку в разделе Project и назовите её “Скрипты”

Для того, чтобы создать скрипт, связанный с сферой, необходимо выбрать сферу, затем в разделе Inspector, выбрать Add Component -> New Script -> ввести название скрипта и нажать Create and Add:



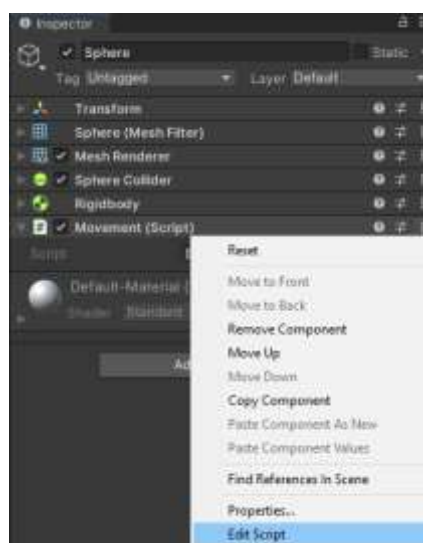
По умолчанию, файл скрипта будет создан в корневом каталоге Assets, переместите его в папку “Скрипты” что бы поддерживать раздел Projects в структурированном виде.

Для того, чтобы выбрать редактор скриптов по умолчанию, нажмите Edit -> Preferences, затем перейдите в раздел External Tools -> External Script Editor:



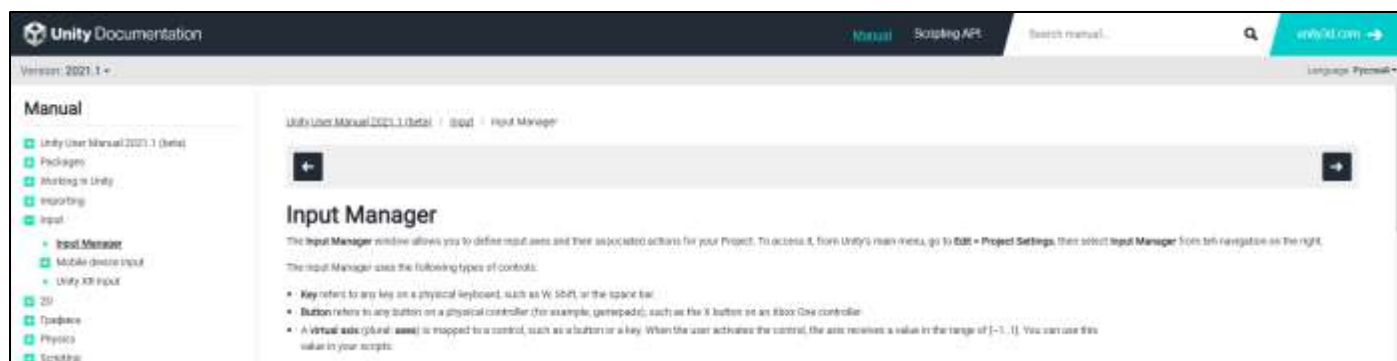
В случае, если наличие Visual Studio не было определено автоматически, можно указать путь к файлу запуска вручную. Как правило, исполняемый файл Visual Studio находится по адресу: Program Files (x86)\Microsoft Visual Studio\2019\Community\Common7\IDE\devnev.exe

Открыть файл скрипта можно дважды кликнув по нему в разделе Project, либо через меню в разделе Hierarchy объекта:



Справочную информацию по написанию скриптов и работе в Unity можно найти по адресу: <https://docs.unity3d.com/ru/2021.1/Manual/UnityManual.html>

Например, информация о методах ввода:

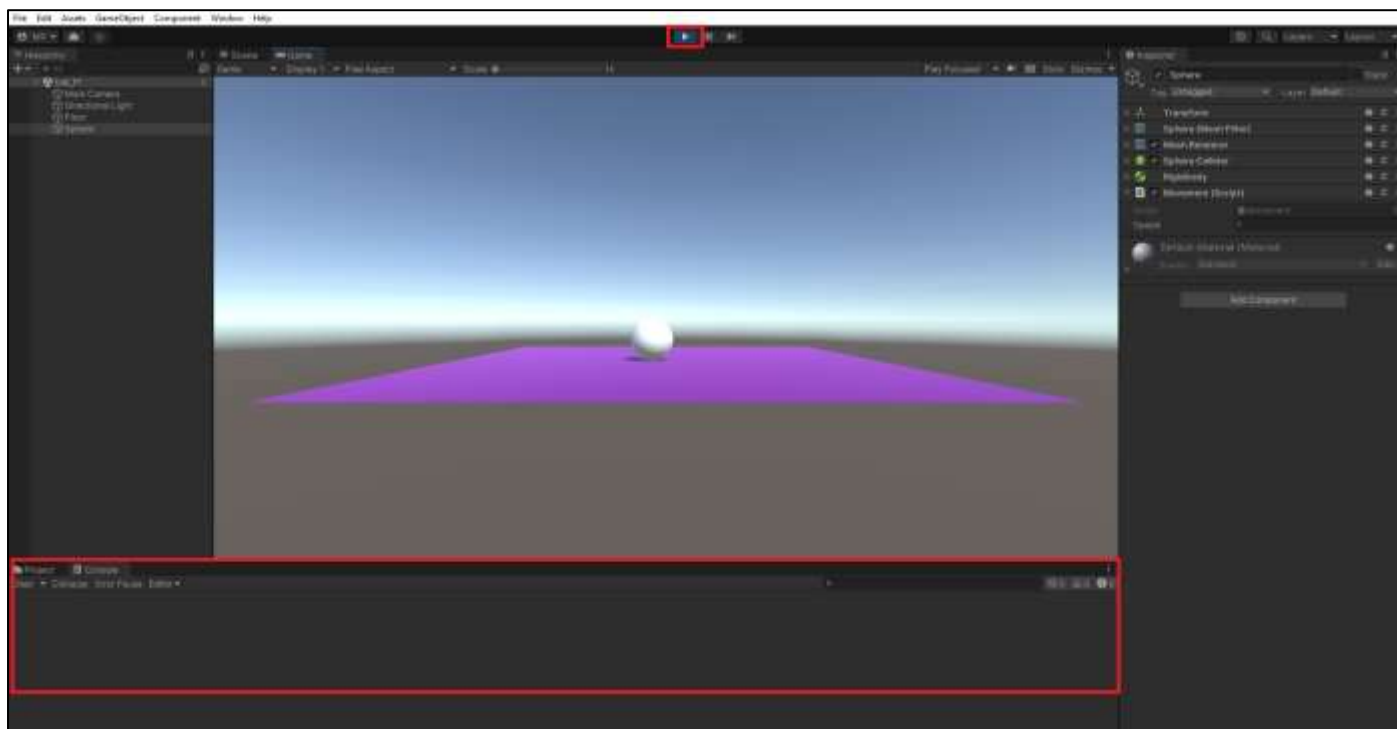


Скрипт описывающий перемещения сферы при помощи воздействия сил, будет выглядеть следующим образом:

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class Movement : MonoBehaviour // класс, описывающий поведение сферы
6 {
7     public float speed; // переменная, определяющая силу воздействия на сферу
8
9     private Rigidbody rb; // переменная для получения ссылки на физическую модель сферы
10
11     // вызывается один раз, в начале работы игры
12     void Start()
13     {
14         rb = GetComponent<Rigidbody>(); // получение ссылки на физическую модель сферы
15     }
16
17     // вызывается каждый кадр, до расчета физики
18     void FixedUpdate()
19     {
20         float moveHorizontal = Input.GetAxis("Horizontal"); // получение смещения по горизонтальной оси
21         float moveVertical = Input.GetAxis("Vertical"); // получение смещения по вертикальной оси
22
23         Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical); // создание направления воздействия силы исходя из параметров ввода
24
25         rb.AddForce(movement * speed); // применение силы в направлении заданном клавишами с величиной определяемой speed
26     }
27 }
```

Поскольку переменная speed публична, её значение можно задать в разделе Inspector при выборе объекта.

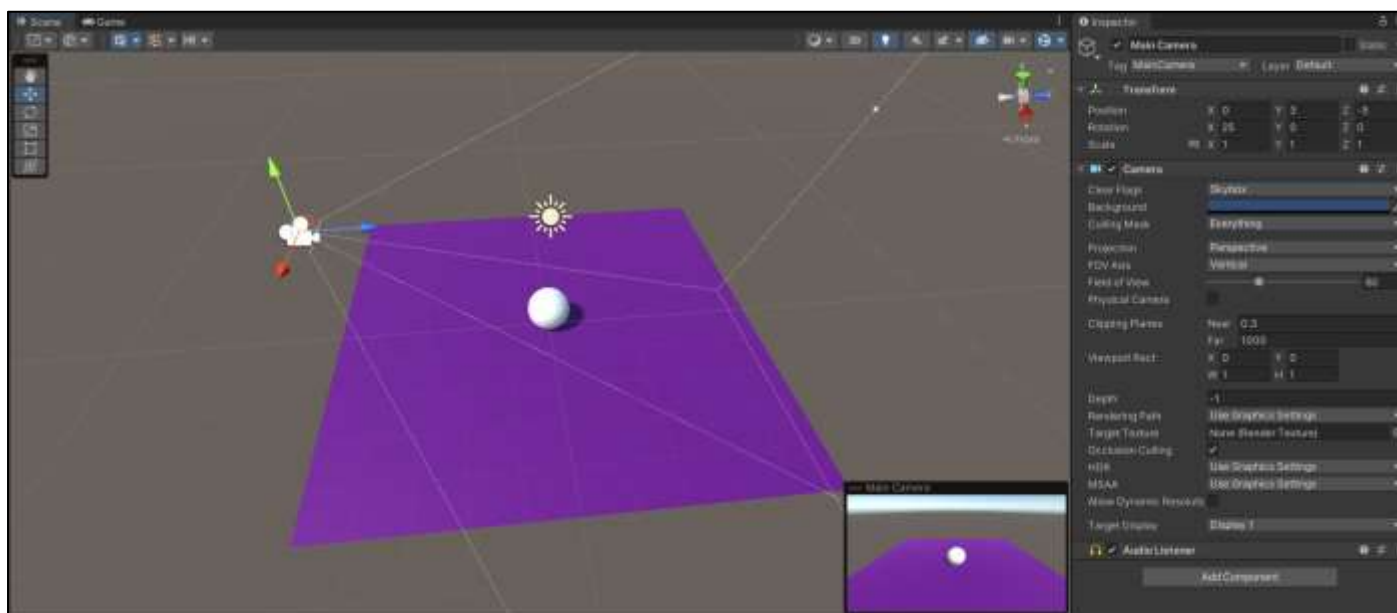
Для того, чтобы проверить работоспособность скрипта, нажмите кнопку Play. В случае возникновения ошибок, информация о проблеме будет отображаться разделе Console:



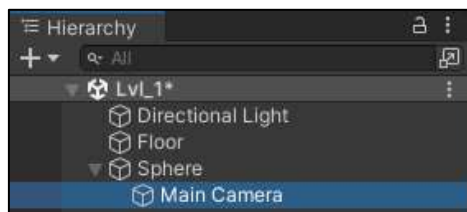
Если всё было сделано правильно, то при нажатии стрелок на клавиатуре, сфера должна начать перемещение в соответствующую сторону. (не забудьте указать параметр speed для объекта)

Следящая камера.

Для слежения за перемещающимся объектом, как правило, используется следящая камера. Реализовать её можно Выставив желаемую позицию камеры:



и перетащив объект Main Camera на объект “Сфера” в разделе Hierarchy:



В большинстве случаев этого достаточно, однако, поскольку объект “Сфера” при перемещении вращается, результат будет не то что бы хороший. Происходит это из-за того, что дочерний объект наследует все преобразования родительского объекта, такие как перемещение и вращение по всем трём осям.

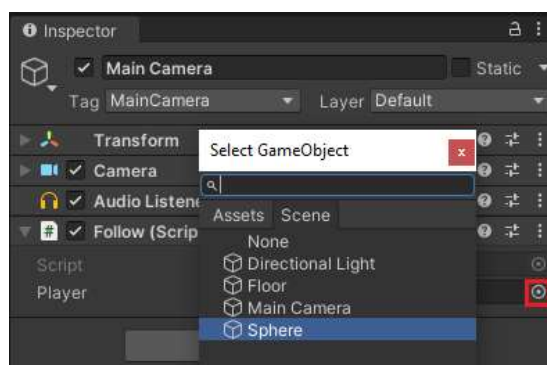
Решением данной проблемы будет описание отношений между камерой и сферой через отдельный скрипт.

Вынесите камеру из объекта “Сфера” в разделе Hierarchy и добавьте к ней новый скрипт в разделе Inspector так же, как вы делали это ранее для объекта “Сфера”.

Текст скрипта будет выглядеть следующим образом:

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class Follow : MonoBehaviour
6 {
7     public GameObject player; // ссылка на объект сфера
8
9     private Vector3 offset; // смещение камеры относительно игрока
10
11     // вызывается в начале работы только один раз
12     void Start()
13     {
14         offset = transform.position - player.transform.position; // вычисление смещения камеры относительно игрока
15     }
16
17     // вызывается каждый кадр, после внесения всех изменений
18     void LateUpdate()
19     {
20         transform.position = player.transform.position + offset; // смещение камеры в позицию игрока с учётом смещения
21     }
22 }
```

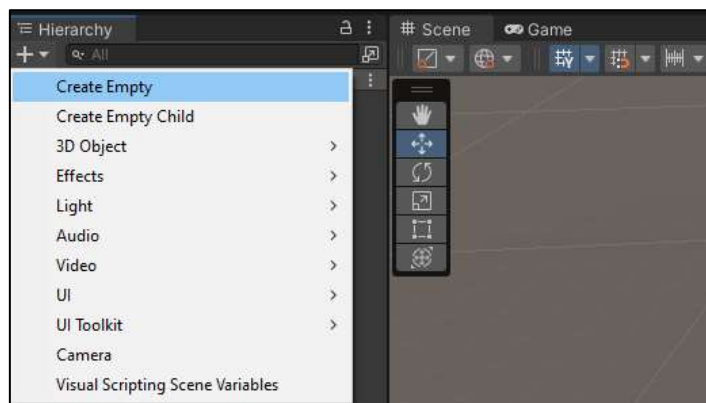
Передать ссылку на объект, за которым будет перемещаться камера можно следующим образом:



В случае, если всё было сделано правильно, камера должна перемещаться за сферой, при этом не вращаясь вокруг неё.

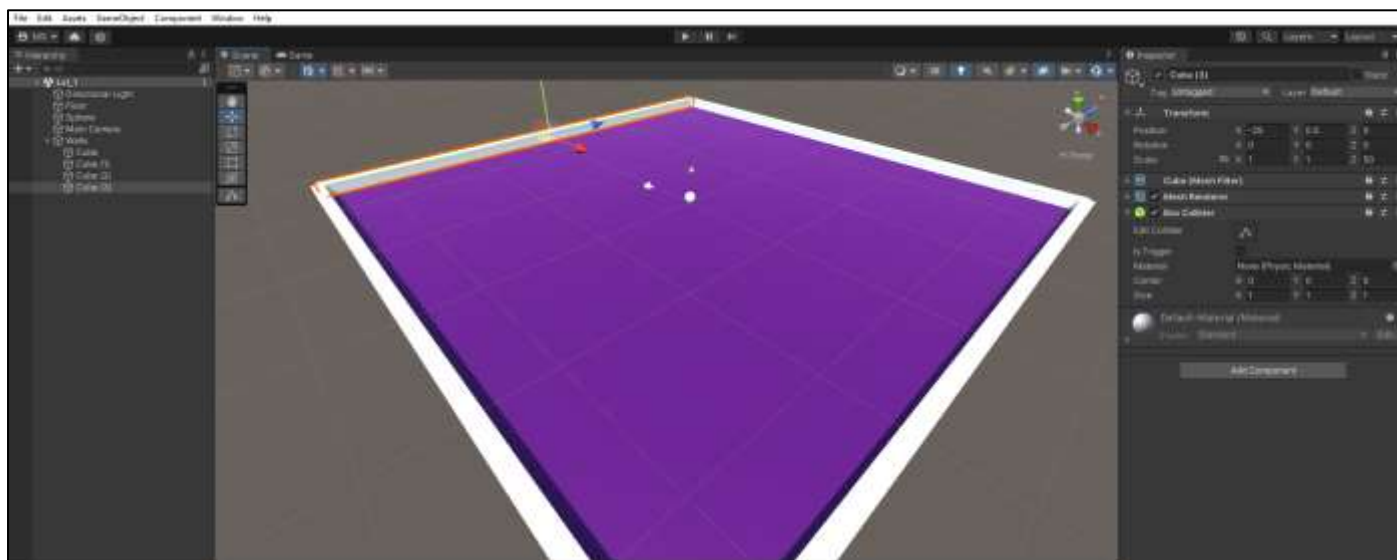
Для того, чтобы сфера перестала выкатываться за границы игрового поля, необходимо увеличить размер игрового поля задав параметр Scale для плоскости по осям X и Z, а затем добавить стены.

Поскольку стены являются частью одной структуры, имеет смысл добавить их как части одного объекта. Для этого, используя меню **GameObjects**, добавьте пустой объект в раздел **Hierarchy**:



Переименуйте его в “Стены”, а затем, добавьте к нему 4 куба в качестве дочерних элементов. Изменить размеры и позицию кубов можно в разделе Inspector, либо при помощи визуальных инструментов.

После изменения размеров и позиций кубов, должна получиться сцена, огороженная стенами:



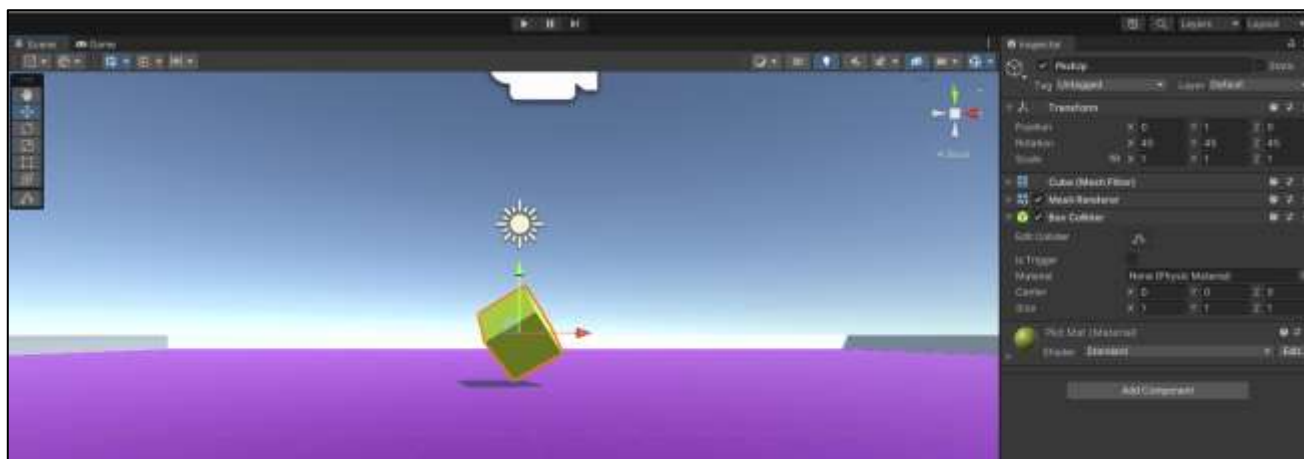
Собираемые объекты.

Следующий этап разработки, создание объектов, которые необходимо собрать для выполнения условия победы.

Примечание: скрыть объекты, находящиеся в сцене, можно сняв флажок рядом с их названием:



Добавьте в сцену новый объект “Куб”, задайте ему позицию, масштаб и повороты как на изображении ниже, а также, создайте для него новый материал:

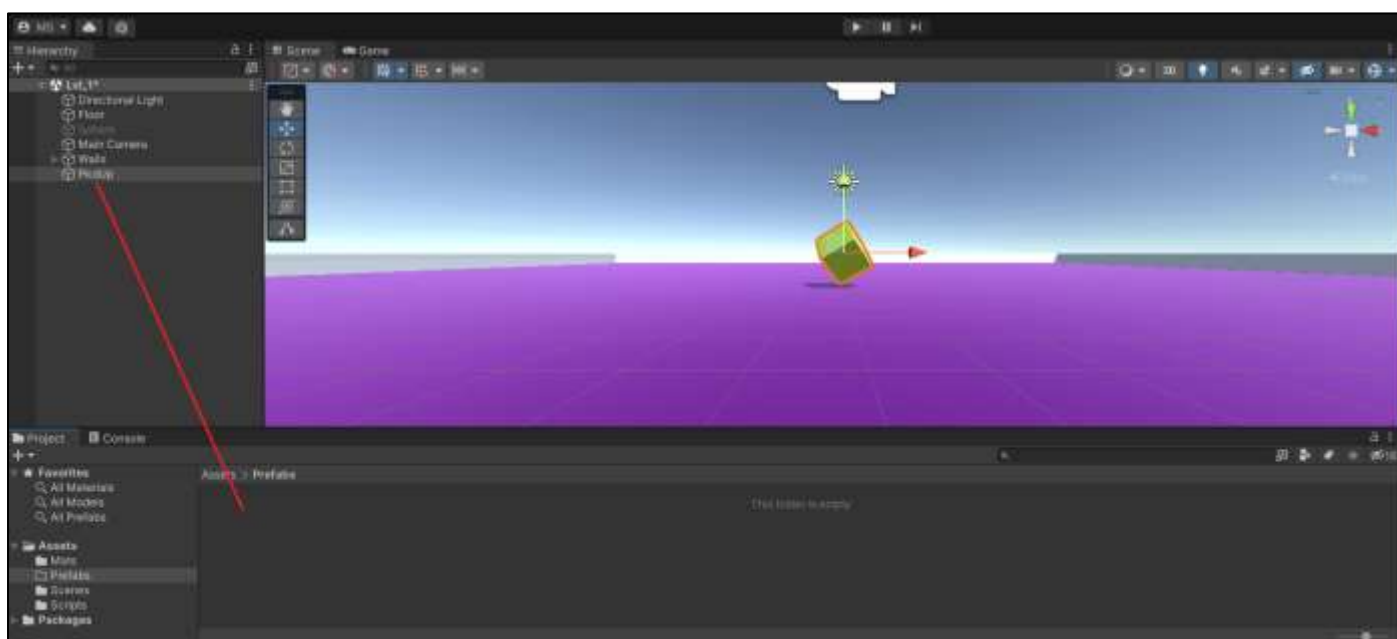


Переименуйте объект в “Подбираемый” и добавьте к нему скрипт следующего вида:

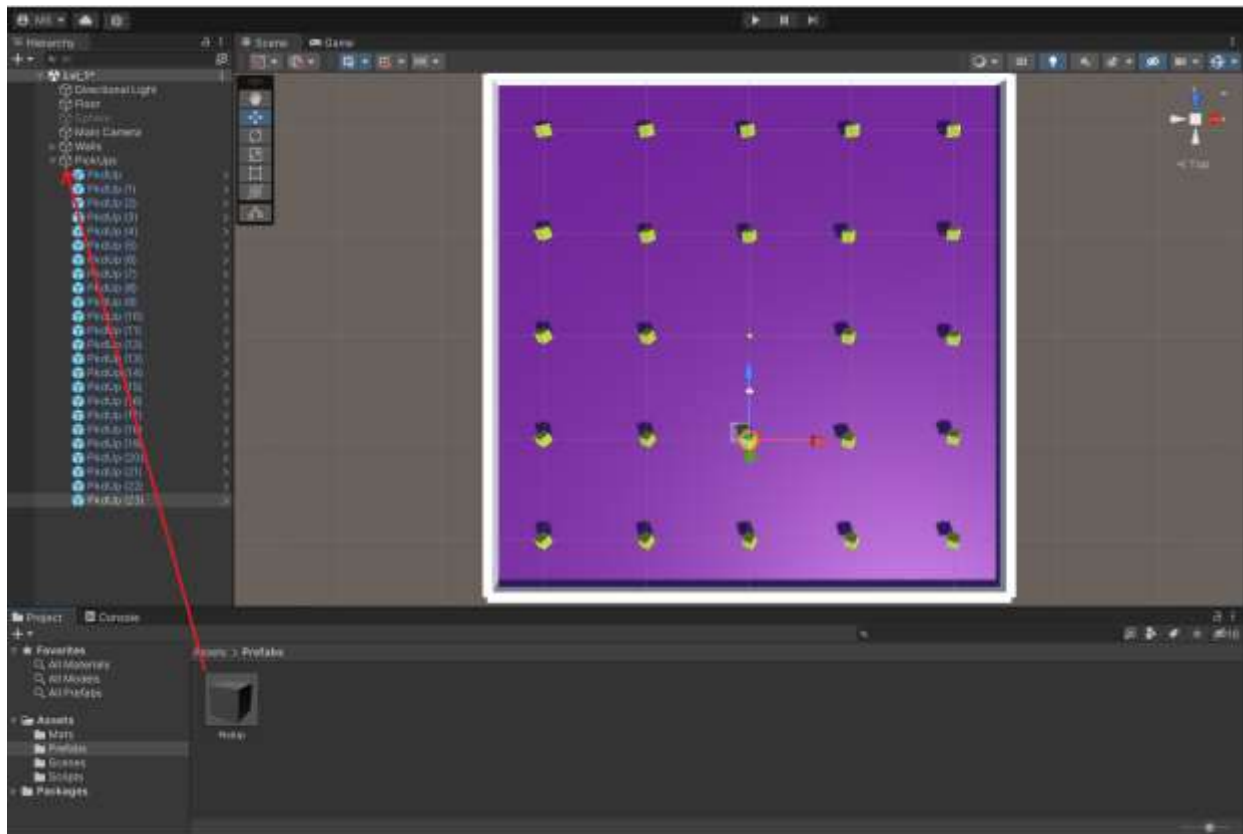
```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class Rotate : MonoBehaviour
6 {
7     // вызывается каждый кадр
8     void Update()
9     {
10         transform.Rotate(new Vector3(15, 30, 45) * Time.deltaTime); // вращение объекта по трём осям
11     }
12 }
```

Данный скрипт, будет поворачивать объект, для которого он назначен на указанную величину по указанным осям каждый кадр, в зависимости от прошедшего за это время числа тиков системного таймера. Собираемых объектов будет больше одного, однако все они будут работать по одному принципу. Для того, чтобы можно было повторно использовать созданный объект, имеет смысл сделать из него Prefab (шаблон объекта).

Для того, чтобы сделать Prefab, создайте соответствующую папку в разделе Project, а затем перетащите туда подбираемый объект из раздела Hierarchy:



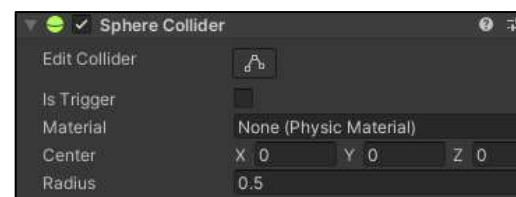
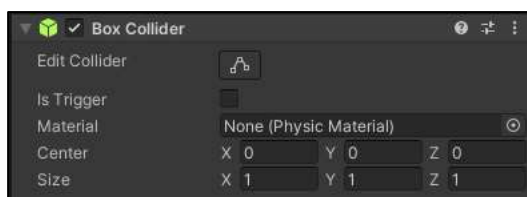
Для хранения собираемых объектов, имеет смысл использовать тот же подход что и для стен уровня. Создайте пустой объект, назовите его “Собираемые объекты”, а затем, добавьте в него несколько шаблонов собираемых объектов из раздела Project и разместив их в сцене:



Примечание: в случае, если необходимо расставить множество копий одного объекта, можно использовать сочетание клавиш Ctrl + D для создания дубликата.

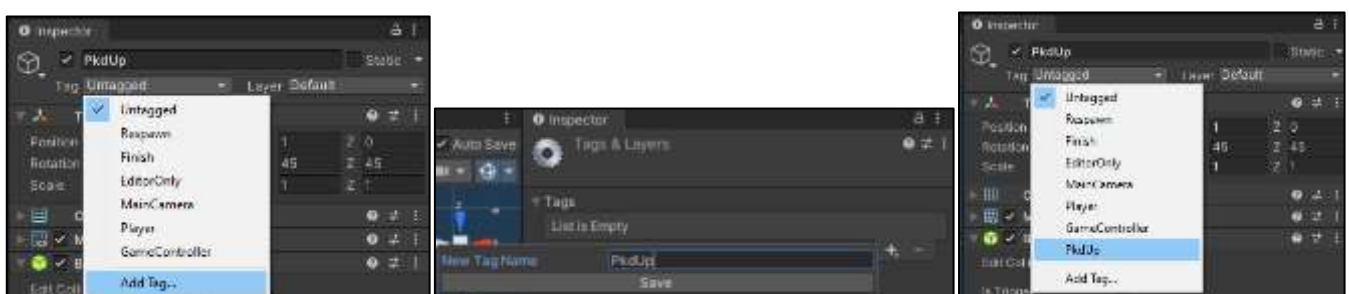
Следующий этап, модифицировать скрипт, закреплённый за сферой, добавив обработку столкновения между сферой и собираемыми объектами.

Примечание: обработка столкновений возможна только между объектами, имеющими структуру Collider.



На практике, сфера может сталкиваться не только с собираемыми объектами, но и другими объектами сцены, поэтому необходимо определять с каким именно типом объекта произошло столкновение. Сделать это можно используя механизм ярлыков (Tag).

Выберите шаблон собираемого объекта в разделе Project и задайте для него ярлык “Собираемый объект”:



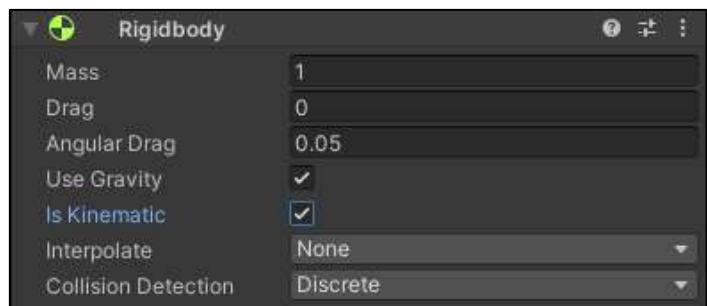
После чего, отметьте флаг Is Trigger в структуре Box Collider шаблона подбираемого объекта. Это нужно для того, чтобы отключить расчёт физики столкновения сферы и собираемого объекта. То есть будет регистрироваться факт столкновения, но не реакция на него с точки зрения физического взаимодействия.

Изменённый скрипт назначенный для сферы будет выглядеть следующим образом:

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class Movement : MonoBehaviour // класс, описывающий поведение сферы
6 {
7     public float speed; // переменная, определяющая силу воздействия на сферу
8
9     private Rigidbody rb; // переменная для получения ссылки на физическую модель сферы
10
11     // вызывается один раз, в начале работы игры
12     void Start()
13     {
14         rb = GetComponent<Rigidbody>(); // получение ссылки на физическую модель сферы
15     }
16
17     // вызывается каждый кадр, до расчёта физики
18     void FixedUpdate()
19     {
20         float moveHorizontal = Input.GetAxis("Horizontal"); // получение смещения по горизонтальной оси
21         float moveVertical = Input.GetAxis("Vertical"); // получение смещения по вертикальной оси
22
23         Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical); // создание направления воздействия силы исходя из параметров ввода
24
25         rb.AddForce(movement * speed); // применение силы в направлении заданном клавишами с величиной определенной speed
26     }
27
28     // вызывается при столкновении сферы с объектами, чей коллайдер был отмечен как Is Trigger
29     void OnTriggerEnter(Collider other)
30     {
31         if (other.gameObject.CompareTag("PickUp")) // если объект имеет ярлык "собираемый объект"
32         {
33             other.gameObject.SetActive(false); // отключить отображение собираемого объекта с которым произошло столкновение
34         }
35     }
36 }
```

В случае, если всё было сделано корректно, в режиме проигрывания, при столкновении сферы и собираемого объекта, собираемый объект должен исчезать.

В связи с особенностями реализации обработки столкновений Unity, реализованная схема обнаружения столкновений объектов является излишне затратной по ресурсам. Для того, чтобы оптимизировать проект, нужно добавить к шаблону собираемого объекта компонент Rigidbody (физическое тело), и обозначить его как Kinematic (не подвержено влиянию сил):



Подсчёт собранных объектов и сообщение о победе.

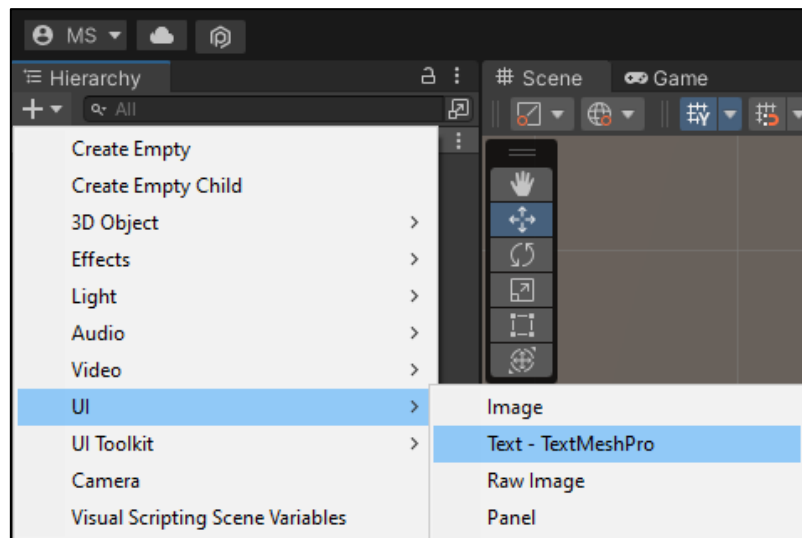
Для подсчёта собранных объектов, к скрипту сферы нужно добавить соответствующую переменную и присвоить ей начальное значение:

```
7 public float speed; // переменная, определяющая силу воздействия на сферу
8
9 private Rigidbody rb; // переменная для получения ссылки на физическую модель сферы
10 private int count; // счётчик собранных объектов
11
12 // вызывается один раз, в начале работы игры
13 void Start()
14 {
15     rb = GetComponent<Rigidbody>(); // получение ссылки на физическую модель сферы
16     count = 0;
17 }
```

Само изменение числа собранных объектов будет расположено в обработчике события столкновения:

```
30 // вызывается при столкновении сферы с объектами, чей коллайдер был отмечен как Is Trigger
31 void OnTriggerEnter(Collider other)
32 {
33     if (other.gameObject.CompareTag("PickUp")) // если объект имеет ярлык "собираемый объект"
34     {
35         other.gameObject.SetActive(false); // отключить отображение собираемого объекта с которым произошло столкновение
36         count = count + 1; // при столкновении с собираемым объектом, счётчик собранных объектов увеличивается
37     }
38 }
```

Для отображения числа собранных объектов, необходимо добавить элемент интерфейса Text:



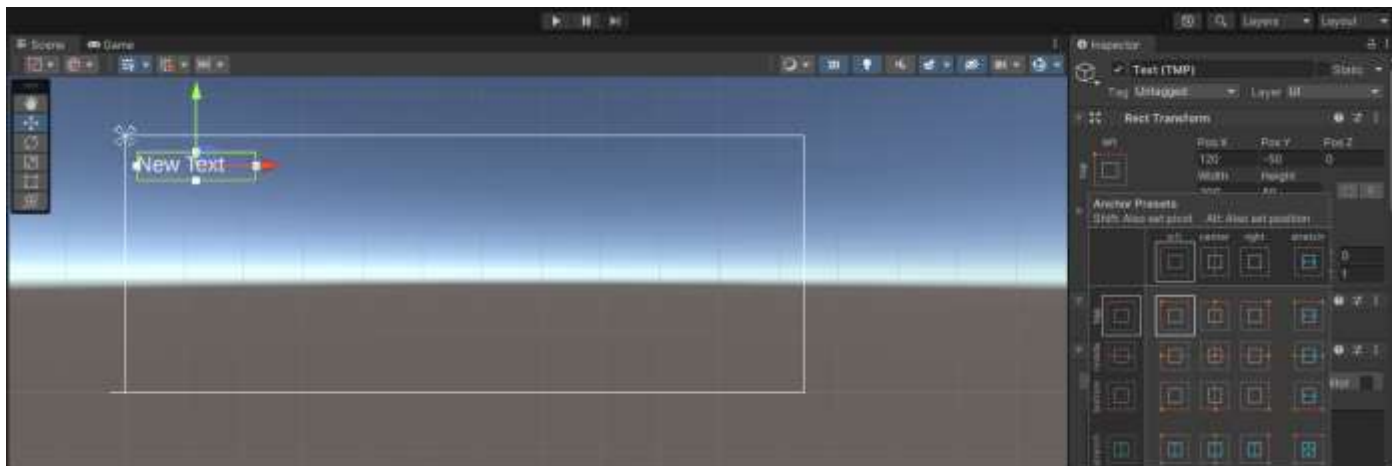
После добавления необходимых ресурсов, вместе с компонентом Text, в сцену будут добавлены компоненты Canvas и EventSystem. Это стандартные компоненты системы интерфейса Unity, необходимые для его корректной работы.

Что бы увидеть добавленный компонент, переключите сцену в режим 2D, дважды кликните по компоненту в разделе Hierarchy, а затем, задайте ему цвет:



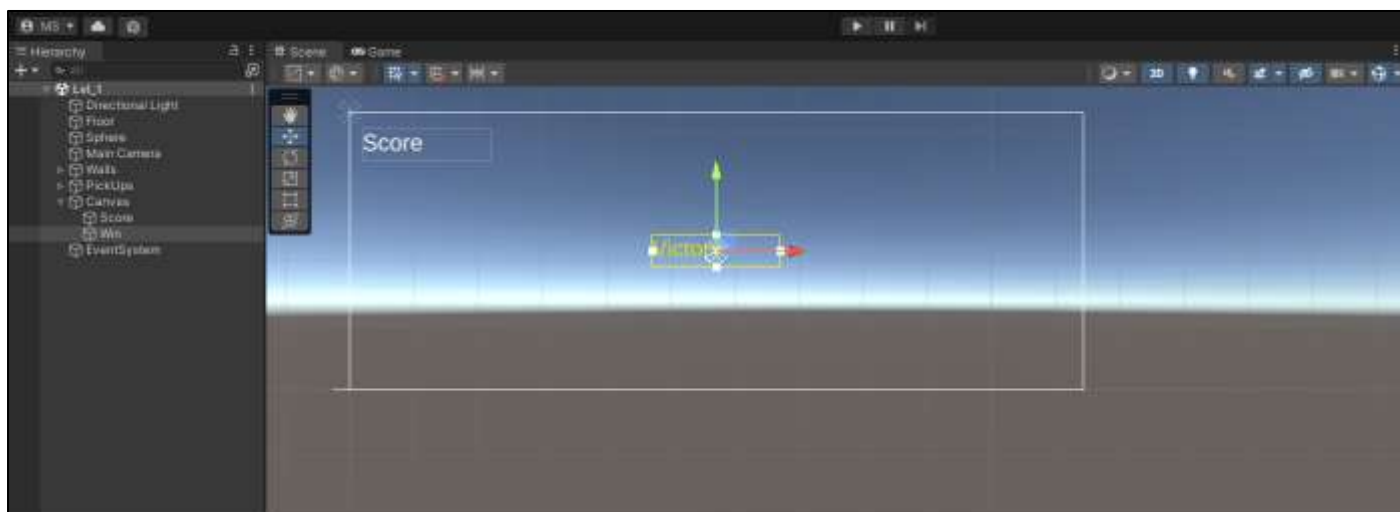
Примечание: позиция текста определяется точкой отсчёта. В большинстве случаев, имеет смысл переместить эту точку в угол кадра. Иначе на устройствах, с соотношением сторон отличным от использованного при разработке, часть интерфейса может оказаться за границей кадра.

Задать точку отсчёта можно при в разделе Inspector, либо перетащив курсором мыши в нужную позицию:



Помимо подсчёта очков, необходимо сообщать игроку о том, что он собрал все собираемые объекты (победил). Для этого понадобится второй элемент интерфейса типа Text.

В результате должен получиться интерфейс следующего вида:



После того, как элементы интерфейса были размещены на экране, можно переходить к их использованию для отображения соответствующей информации. Инициализация данных будет выглядеть следующим образом:

```
using TMPro; //подключение расширения для работы с текстом

Counters:0
public class Movement : MonoBehaviour
{
    public float speed; // переменная, определяющая силу воздействия на сферу
    public int objects; // количество собираемых объектов в сцене

    private Rigidbody rb; // переменная для получения ссылки на физическую модель сцены
    private int count; // счётчик собранных объектов

    public TMP_Text Score; // ссылка на элемент интерфейса для отображения числа собранных объектов
    public TMP_Text Victory; // ссылка на элемент интерфейса для отображения сообщения о победе

    // вызывается один раз, в начале работы игры
    Counters:0
    void Start()
    {
        rb = GetComponent<Rigidbody>(); //получение ссылки на физическую модель сферы
        count = 0;

        Victory.text = "";
        SetText(); //обновление текста
    }
}
```

Функция обновления текста будет вызываться в функции обработки столкновений:

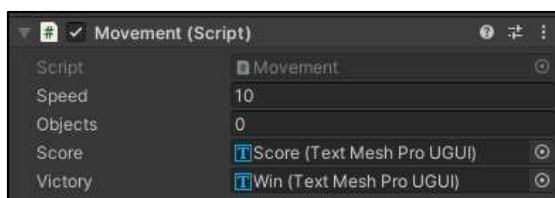
```
private void OnTriggerEnter(Collider other)
{
    if (other.gameObject.CompareTag("PkdUp")) // если объект имеет ярлык "собираемый объект"
    {
        other.gameObject.SetActive(false); // отключение собираемого объекта, с которым произошло столкновение
        count = count + 1; // при столкновении с объектом, счётчик собранных объектов увеличивается
        SetText(); // обновление текстовых полей
    }
}
```

И будет выглядеть следующим образом:

```
// обновление текстовых полей
Ссылка: 2
private void SetText()
{
    Score.text = "Score: " + count.ToString(); // обновление счётчика собранных объектов

    if (count >= objects) //если число собранных объектов равно числу собираемых объектов в сцене
    {
        Victory.text = "Victory!"; // сообщение о победе
    }
}
```

При этом в скрипт сферы необходимо передать скорость движения сферы, число собираемых объектов и ссылки на соответствующие элементы интерфейса:



Задания:

1. Создать три или более составных уровней (состоящих из нескольких частей), в которых сфера может упасть за пределы игрового поля. Можно использовать наклонные поверхности, узкие переходы между частями уровня, односторонние проходы с перепадом высот и т.д. При падении сферы за пределы уровня, должен происходить его (уровня) перезапуск.
2. В правом верхнем углу экрана выводить время, прошедшее с старта уровня (в секундах). При перезапуске уровня, время должно обнуляться.
3. Добавить дополнительную игровую механику. Например, ветер постоянно сдувающий сферу в каком-либо направлении, ускорение сферы с каждым собранным объектом, объекты/зоны, ускоряющие, либо замедляющие сферу и т.д.
4. При завершении уровня с победой, должен появляться 5 секундный таймер обратного отсчёта. При достижении 0, должен загружаться следующий уровень.

Отчёт.

В качестве отчёта вы должны предоставить документ, содержащий следующие пункты:

1. Цель и задачи лабораторной работы.
2. Краткое описание действий, которые вы выполняли для решения поставленных в лабораторной работе заданий с картинками, представляющими работу приложения.
3. Краткий вывод о проделанной вами работе
4. Ссылка на облачный диск/github с архивом в котором находится **написанные вами скрипты**.