# Лабораторная работа №1: Основы Unity3d

**Цель работы**: познакомиться с основами работы в среде разработки интерактивных графических приложений Unity.

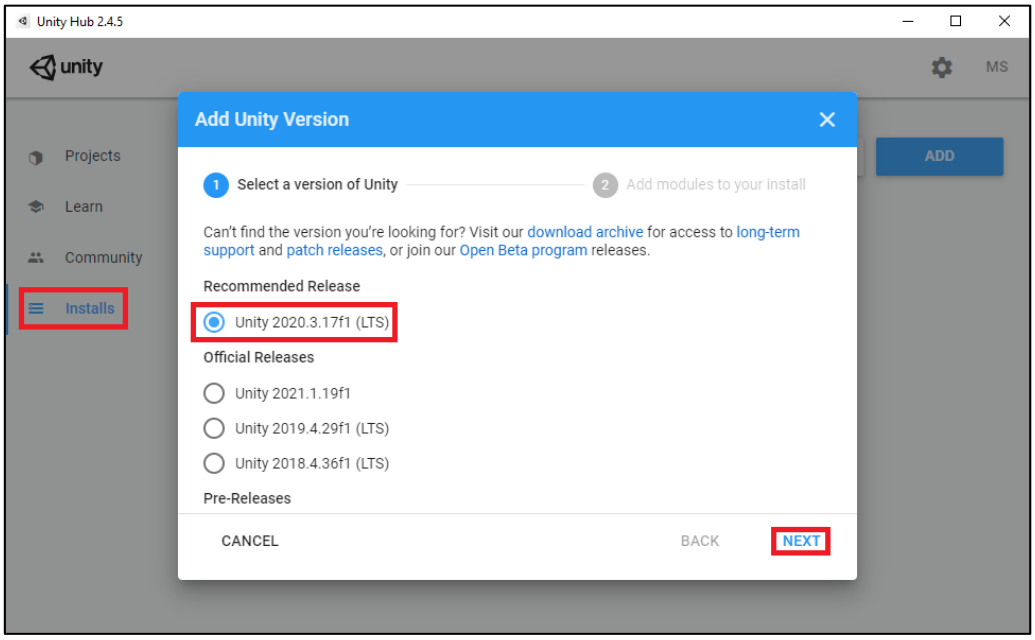
Получить Unity можно по ссылке: https://unity3d.com/ru/get-unity/download

Для загрузки, установки Unity и работы с Unity проектами, рекомендуется использовать Unity Hub.

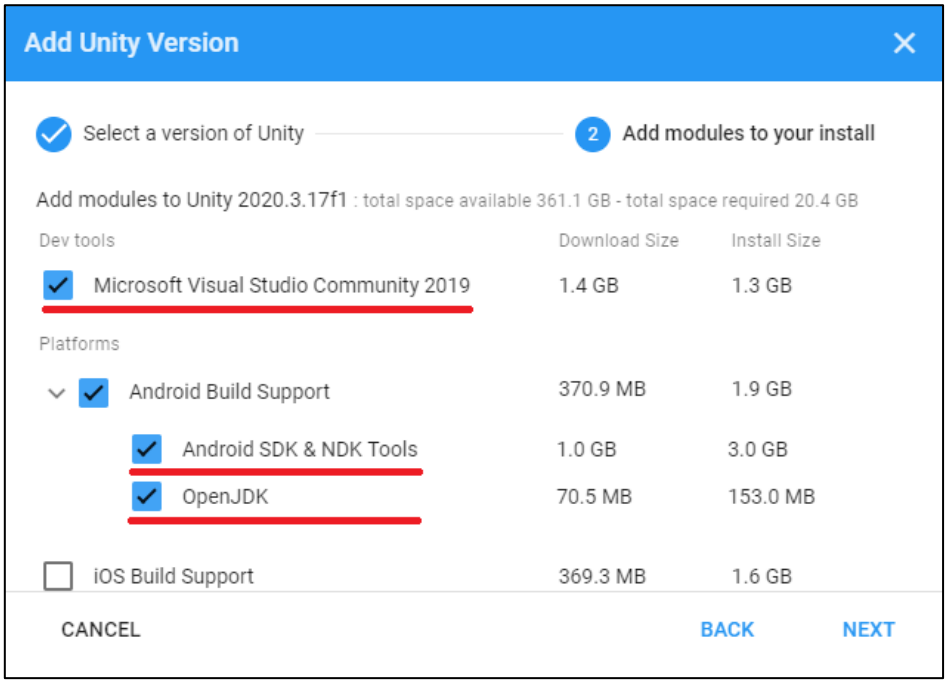
**Примечание**: получить лицензию Unity для не коммерческого использования и предприятий с доходом ниже 100 000$ в год можно бесплатно.

## Установка

Запустите Unity Hub, перейдите в раздел Installs и выберите желаемую версию Unity. (рекомендуется устанавливать версии с долгосрочной поддержкой (LTS)):

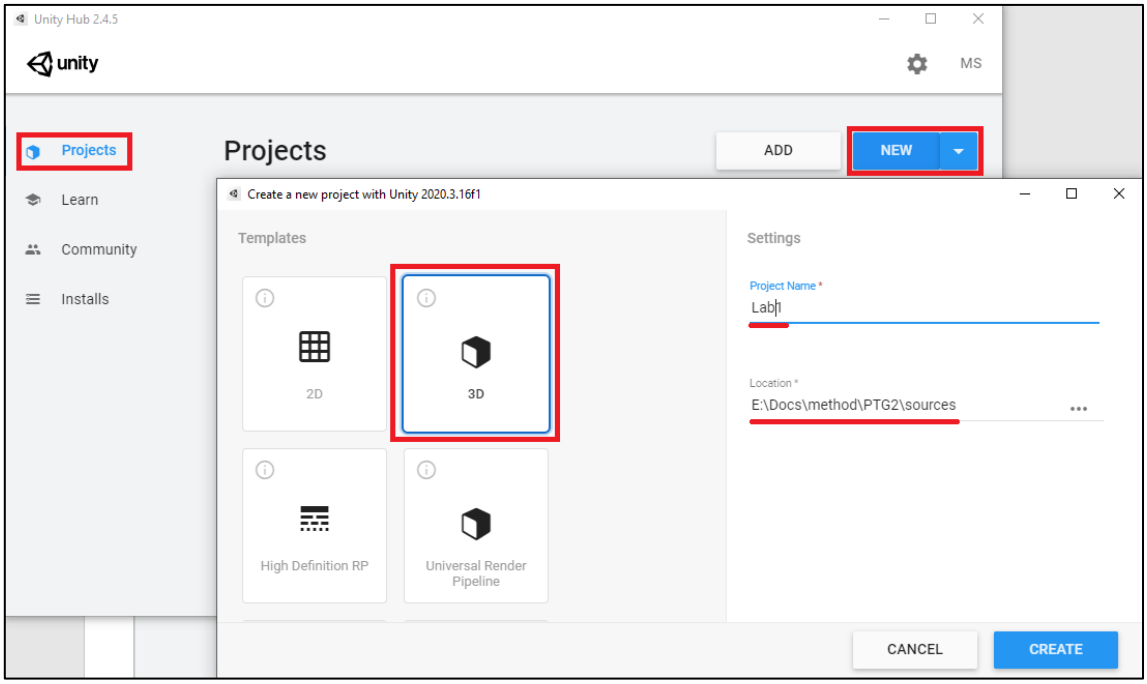



Укажите модули, которые необходимо установить (Поддержка Android, WebGL, и т.д.):



### Создание проекта

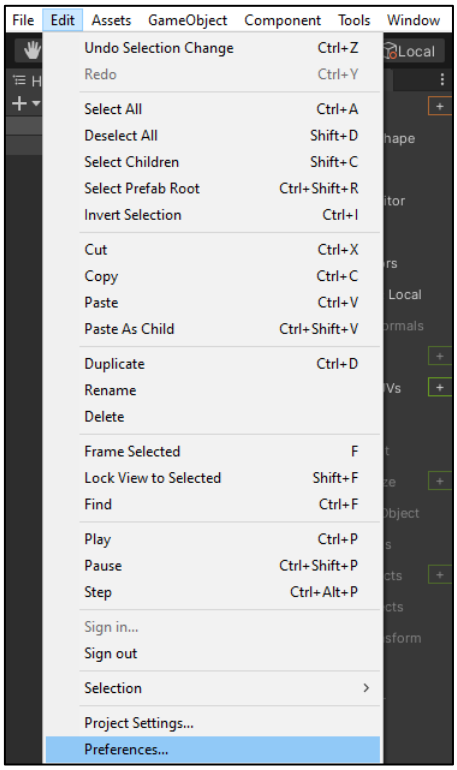
Перейдите в раздел Projects, кликните New, выберите тип проекта 3D, укажите название и директорию в которой будет создан проект:



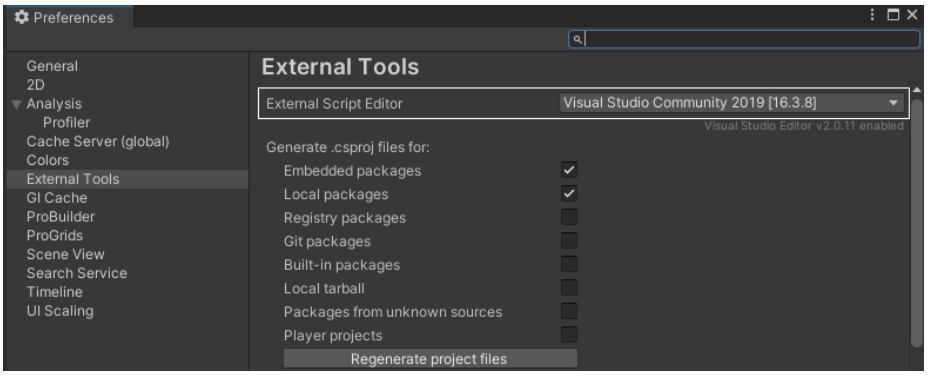

Рекомендуется сохранять проекты в отдельной папке в разделе Документы диска C: или на отдельном диске. Не рекомендуется открывать и работать с проектами, сохранёнными на внешних устройствах (флеш дисках).

### Интеграция с Visual Studio

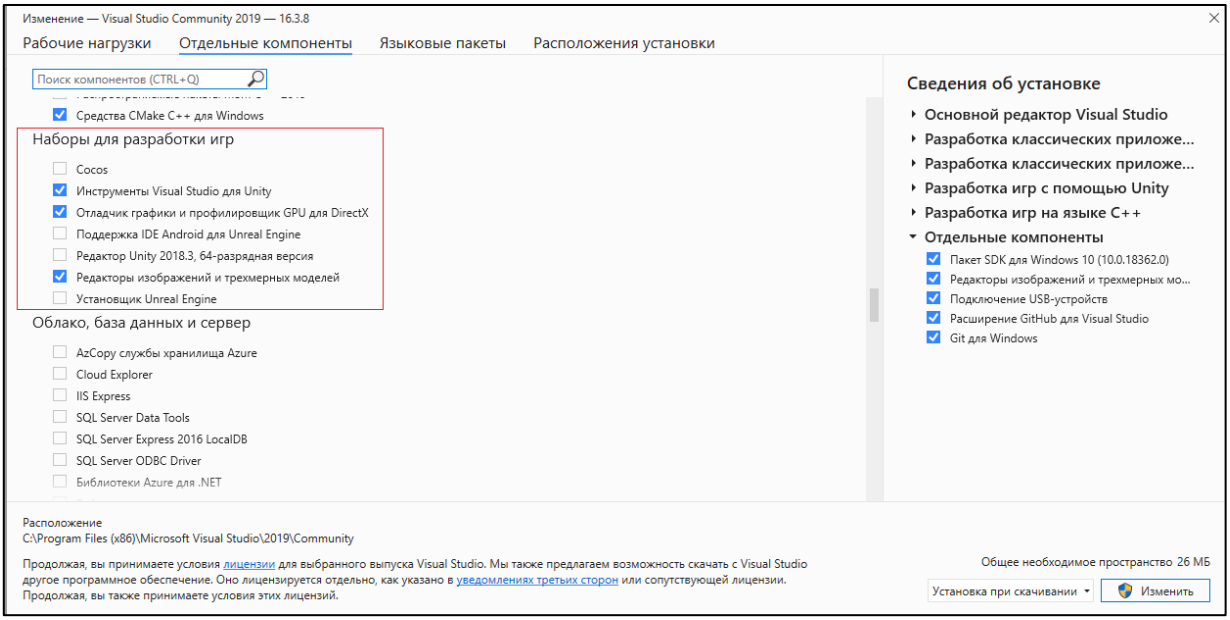
В качестве редактора кода скриптов C# можно выбрать Visual Studio. Для этого, перейдите в раздел Preferences:



И, в качестве внешнего редактора, выберите Visual Studio:

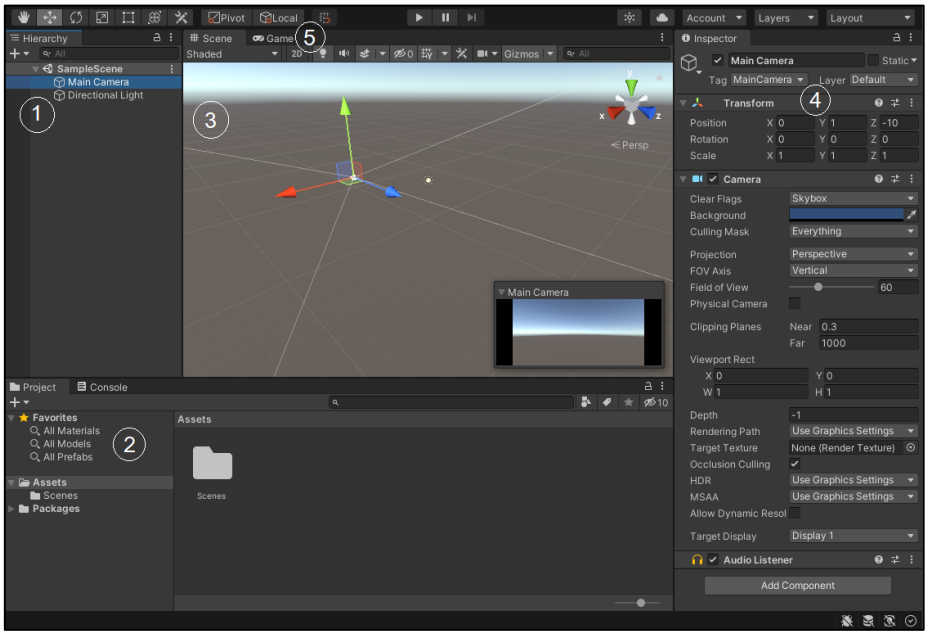



Убедитесь, что в пакете установки Visual Studio включены Инструменты для Unity. Для этого, запустите Visual Studio Installer, перейдите в раздел Отдельные компоненты и найдите список “Наборы для разработки игр”:




## Интерфейс

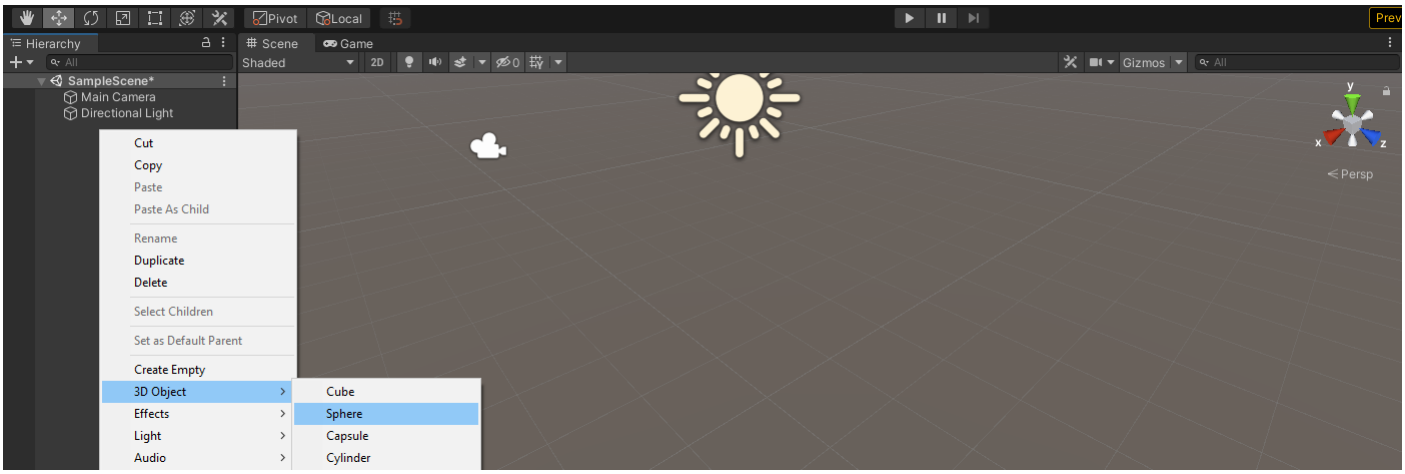
В целом, интерфейс Unity можно разделить на 5 блоков:



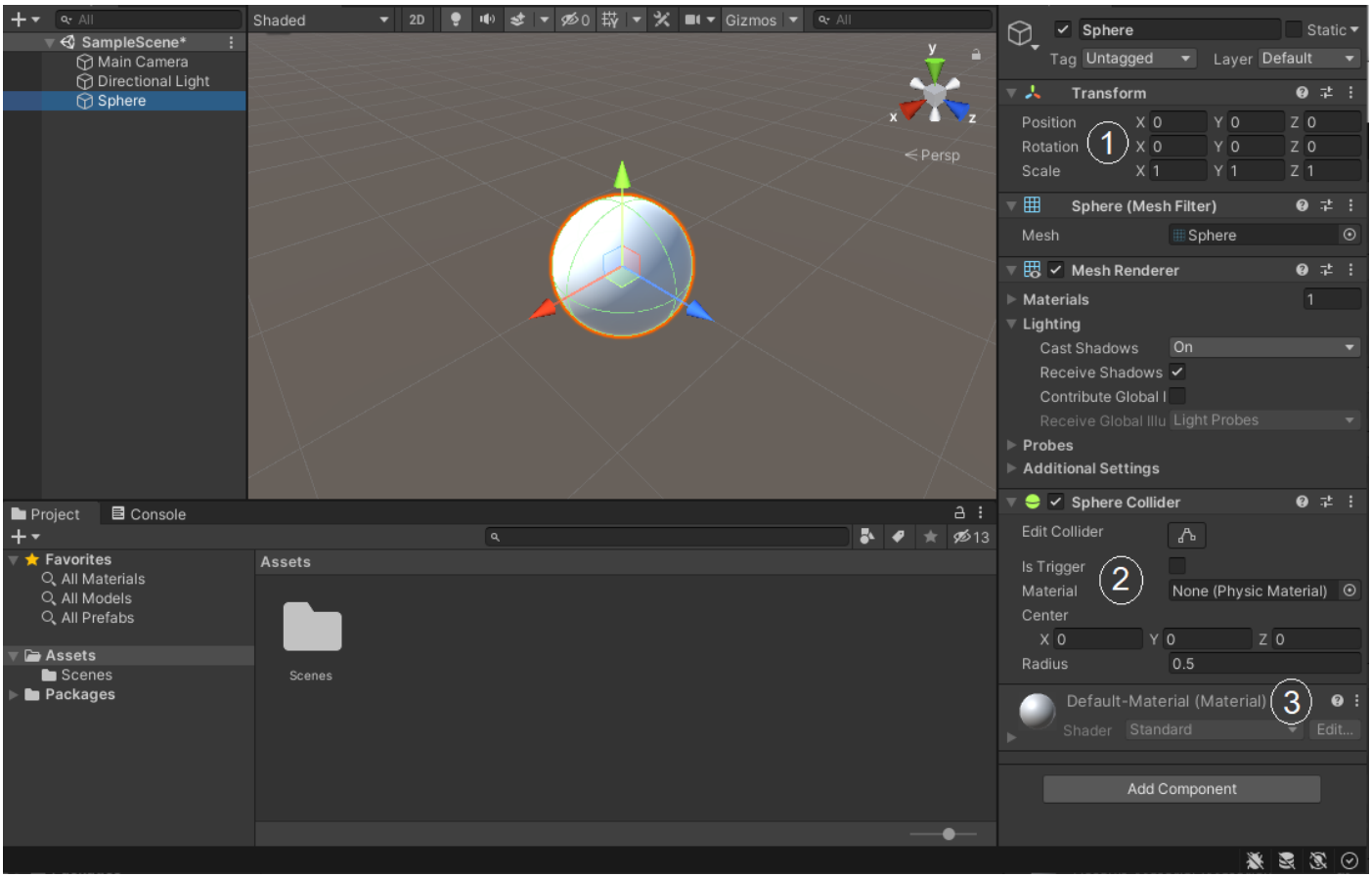
1. Иерархия объектов в сцене. В этом разделе находятся объекты, добавленные в сцену. По умолчанию в сцене присутствуют виртуальная камера и направленный источник освещения.
2. Структура проекта. В этом разделе отображаются папки и файлы, добавленные в проект.
3. Окно редактора сцены. В этом окне можно размещать и позиционировать объекты.
4. Инспектор объектов. В этом окне смотреть и редактировать параметры выбранного объекта сцены.
5. Панель вкладок. В этой строке можно переключаться между сценой, режимом вида из камеры и прочими открытыми вкладками.

**Добавление объекта**

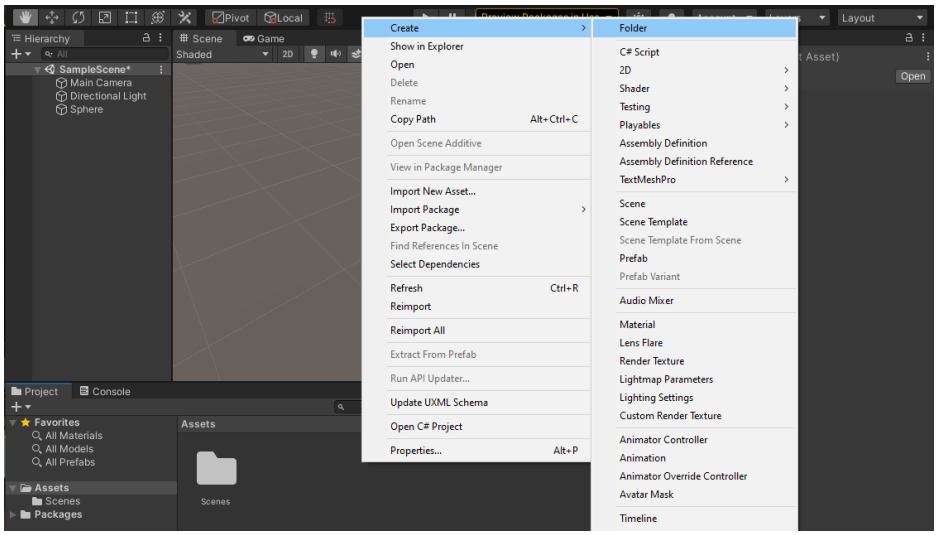
Для того, чтобы добавить новый трёхмерный объект в сцену, кликните правой кнопкой мыши в области Hierarchy, выберите 3D Objects, а затем, кликните на желаемый объект:



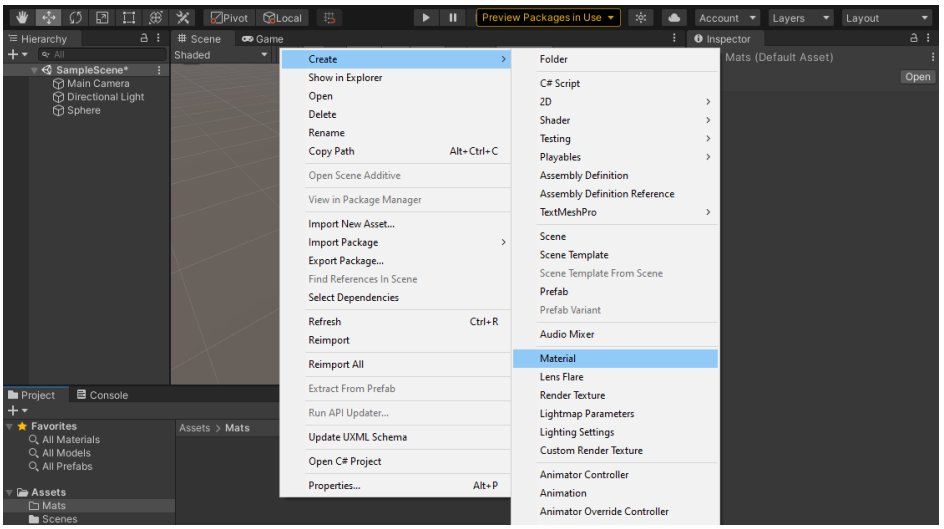

Выбрав объект, в инспекторе объектов вы увидите информацию о его позиции, повороте, масштабе (1), информацию о его ограничивающем объёме (2) и информацию о его материале (3):



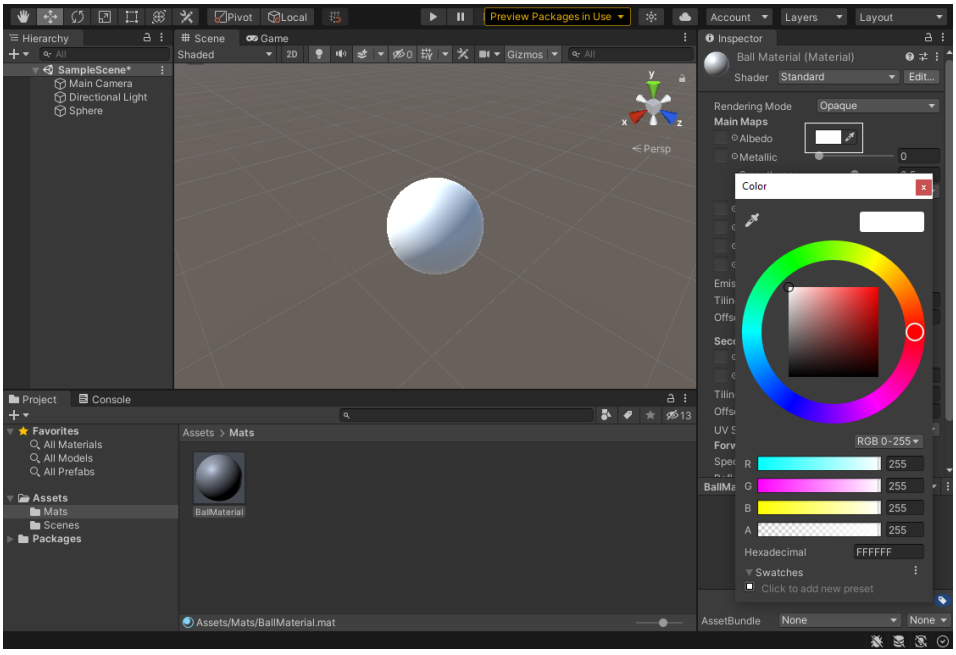
Чтобы создать материал для объекта, кликните в поле Assets и создайте новую папку:




Затем перейдите в созданную папку и кликнув правой кнопкой мыши, выберите создание материала:



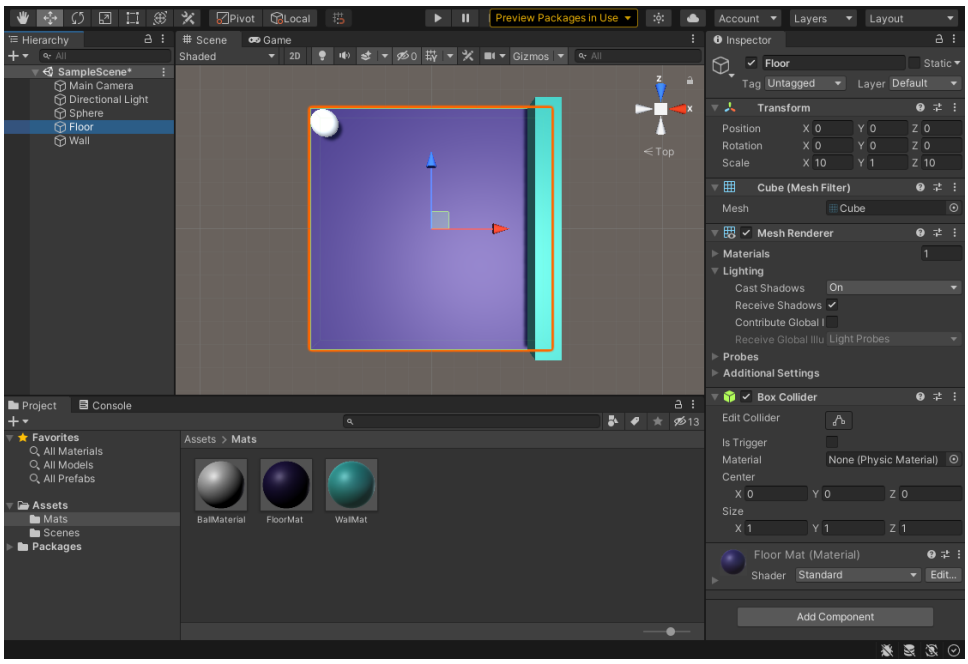

Кликнув на поле рядом с Albedo, вы можете настроить цвет созданного материала:



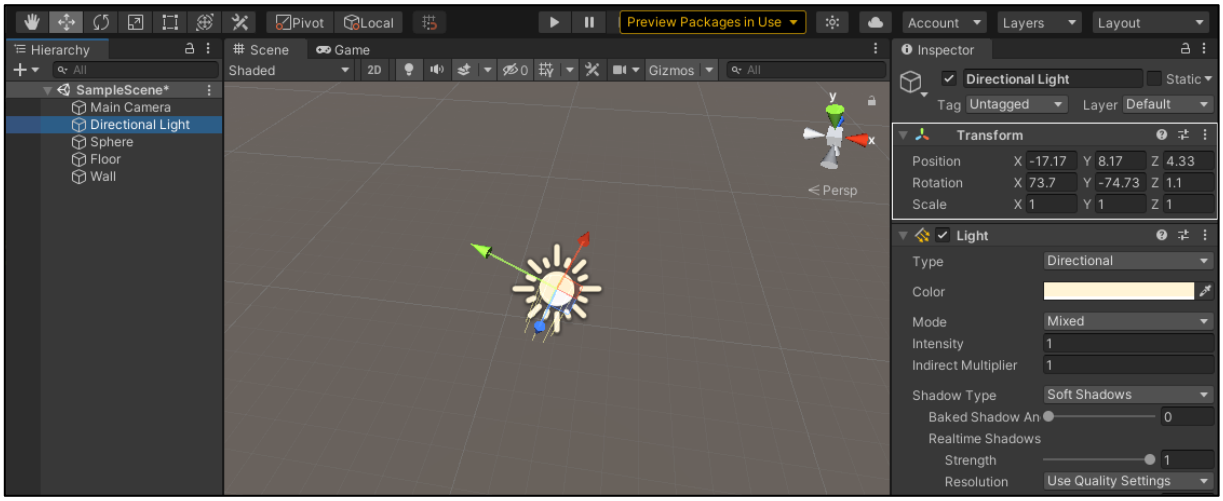

Назначить материал можно просто перетащив его на выбранный объект в сцене или иерархии.

## Перемещение объектов

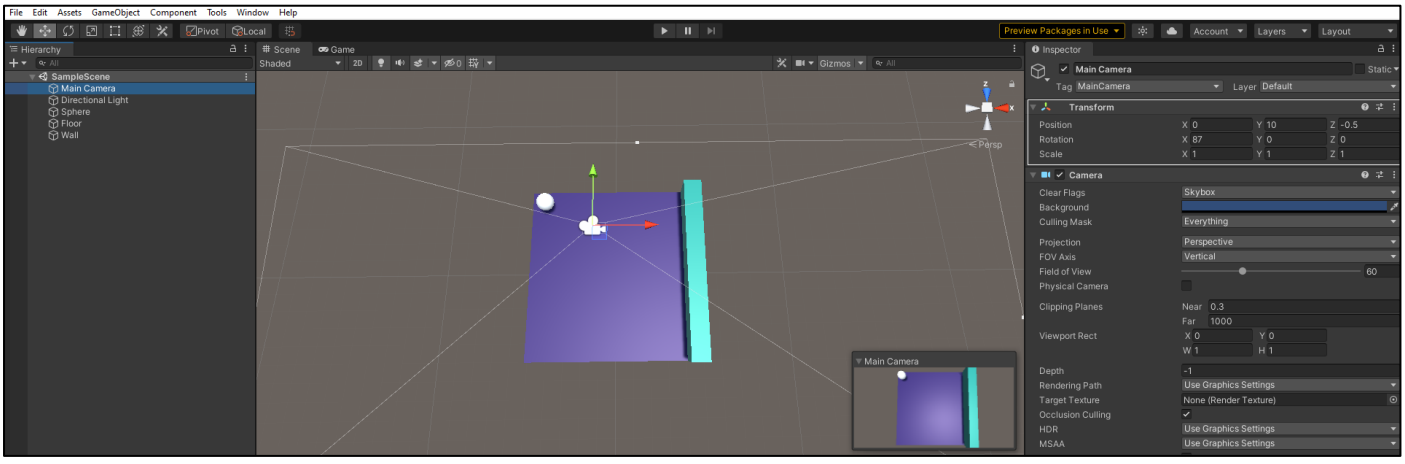
Перед тем как перейти к экспериментам с перемещением объектов, подготовьте сцену. Создайте сферу, создайте параллелепипед, на фоне которого будет происходить перемещение (пол), а также параллелепипед, который будет служить точкой остановки (стена):



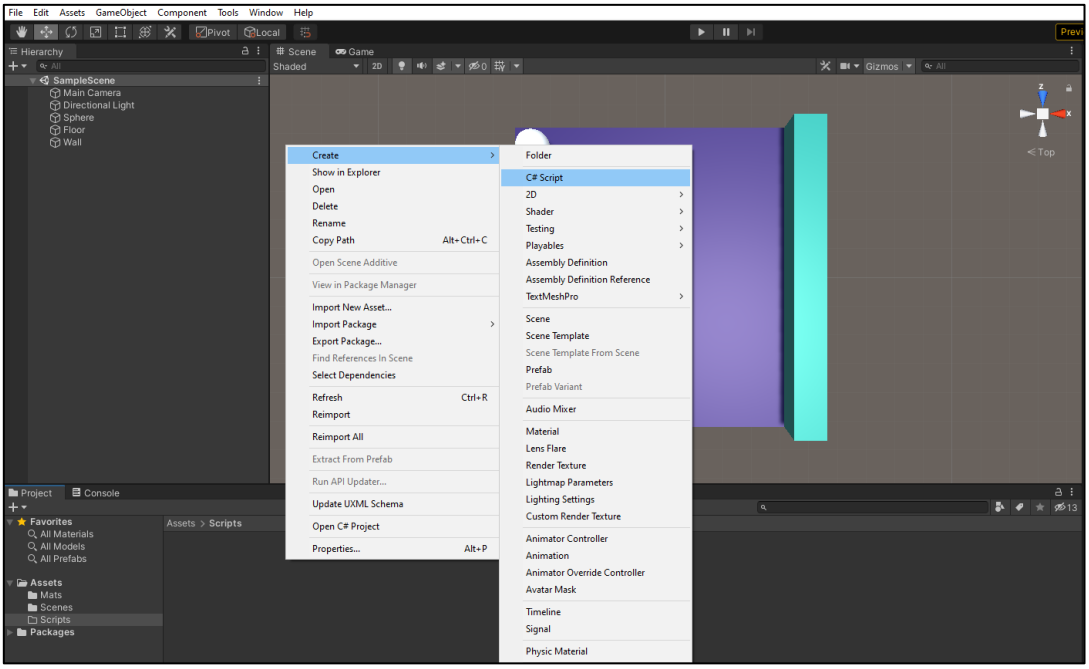

Настройте источник освещения, возможный вариант настройки приведён на изображении:




Настройте камеру так, чтобы она смотрела на получившуюся сцену сверху, под небольшим углом:



В целом, движение объектов в Unity описывается при помощи C# скриптов. Создайте новую папку для хранения скриптов, а затем создайте в ней C# скрипт, для описания поведения объекта Сфера:




По умолчанию, скрипт C# имеет следующий вид:

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

Ссылок: 0
public class BallScript : MonoBehaviour
{
    Ссылок: 0
    void Start() //вызывается один раз при запуске программы
    {

    }

    Ссылок: 0
    void Update() //вызывается каждый кадр
    {

    }
}
```

Для того, чтобы перемещать сферу вправо, необходимо каждый кадр изменять позицию объекта на определённое значение. Ниже приведён пример такого изменения:

```
void Update() //вызывается каждый кадр
{
    //изменение позиции объекта на 2 единицы в секунду
    transform.position += transform.right * Time.deltaTime * 2;
}
```

где:

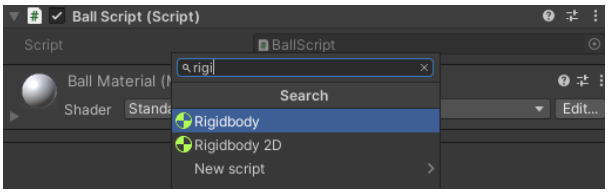
transform.position – текущая позиция объекта

transform.right – вектор “вправо” объекта, в локальной системе координат

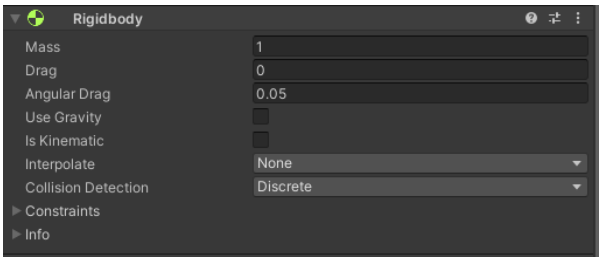
Time.deltaTime – время прошедшее с предыдущего кадра

Назначить скрипт можно перетащив его на объект, либо через меню Add Component:

При запуске проекта, объект должен начинать смещение вправо, однако, не будет останавливаться при достижении стены. Чтобы сфера останавливалась, необходимо назначить для неё “Физическое тело”, для этого, через меню Add Component, добавьте Rigidbody:




В рамках данного проекта, рекомендуется отключить параметр Use Gravity:



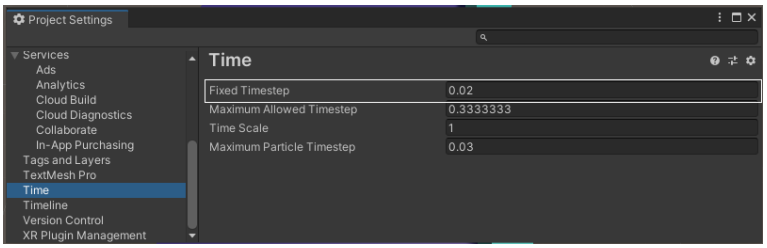

Узнать значение параметров данной структуры можно нажав знак вопроса справа сверху.

После запуска проекта, сфера должна смещаться вправо, до столкновения с стеной.

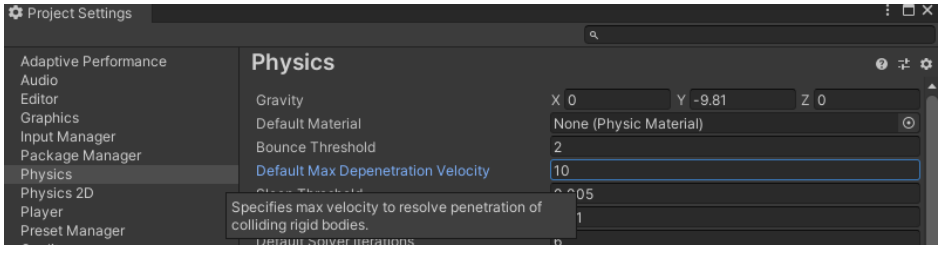
Обратите внимание, что при столкновении с стеной происходит “дрожание” сферы. Данный эффект обусловлен тем, что для предотвращения прохождения сферы сквозь стену используется алгоритм выталкивания. Поскольку расчёт перемещения происходит каждый кадр, а расчёт каждого кадра может занимать разное время, смещение сферы происходит не равномерно. Бороться с этим можно разными путями. Например, использовать более сложные алгоритмы Collision Detection, либо альтернативные методы Update:

```
Ссылок: 0
private void FixedUpdate() //вызывается через фиксированные промежутки времени
{

}

Ссылок: 0
private void LateUpdate() //вызывается после всех Update
{

}
```

Настроить период вызова Fixed Update можно в настройках проекта:



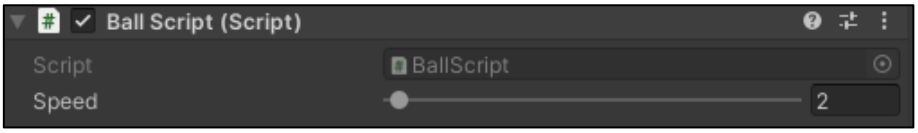

Кроме того, можно настроить силу выталкивания объектов при обнаружении столкновения:



Для удобства экспериментов с перемещением сферы, регулятор скорости перемещения можно вынести в инспектор объектов:

```
[Range(0.1f, 100f)] //параметр поля добавляющий в окно инспектора ползунок изменения значений
public float speed = 2;

Ссылок: 0
void Start() //вызывается один раз при запуске программы...

Ссылок: 0
void Update() //вызывается каждый кадр
{
    //изменение позиции объекта на 2 единицы в секунду
    transform.position += transform.right * Time.deltaTime * speed;
}
```

Если всё было сделано правильно, в инспекторе объектов должен появиться ползунок вида:



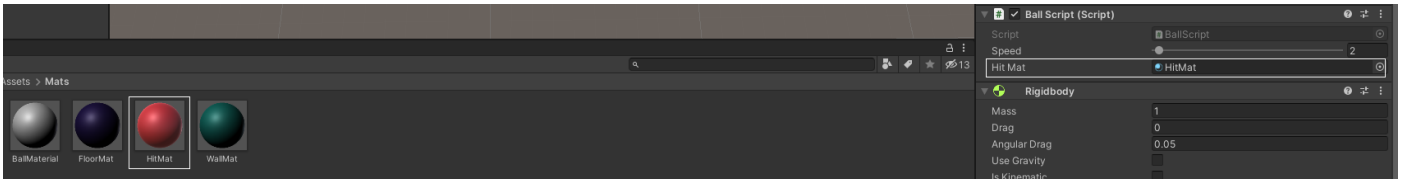

**Реакция на столкновения**

При обнаружении столкновения объектов, имеющих Collider (все участники столкновения) и Rigidbody (хотя бы один из участников столкновения), Unity автоматически генерирует событие OnCollisionEnter, параметром которого является ссылка на объект, с которым произошло столкновение.

Это событие можно использовать для обозначения столкновения сферы и стены. Для этого, добавьте новую публичную переменную типа материал, а также приватную переменную типа Render. Компонент Render отвечает за визуализацию объекта. В функции старт, можно, используя функцию поиска компонентов объекта GetComponent получить ссылку на Renderer и использовать в дальнейшем для смены материала. В заключении, в обработчике события OnCollisionEnter, задайте объекту новый материал:

```
public Material hitMat; //материал отображаемый при столкновении сферы

private Renderer ren;

Ссылок: 0
void Start() //вызывается один раз при запуске программы
{
    ren = GetComponent<Renderer>(); //получение ссылки на отрисовщик сферы
}

Ссылок: 0
private void OnCollisionEnter(Collision collision) //вызывается при столкновении сферы
{
    ren.material = hitMat; //смена материала
}
```

Не забудьте создать новый материал и передать его в качестве параметра в скрипт:




Если всё было сделано правильно, то при достижении стены, сфера должна менять свой цвет.

Следующим этапом развития проекта, будет начало движения сферы при клике по ней курсором мыши.

**Выделение объекта**

Для того, чтобы выделить объект, необходим добавить ещё две переменных, описывающих материалы: материал по умолчанию и материал выделения. Смена материала по умолчанию будет происходить при вызове публичного метода select(), возврат материала по умолчанию будет происходить каждый кадр:

```
[HideInInspector] //сокрытие переменной в инспекторе
public bool start = false; //переменная определяющая началось движение или нет

public Material defMat; //материал отображаемый при столкновении сферы
public Material selMat; //материал отображаемый при выборе сферы

ссылка: 1
public void select() //функция для переключения цвета при выборе сферы
{
    ren.material = selMat; //смена материала
}

Ссылок: 0
void Update() //вызывается каждый кадр
{
    if (start == true)
    {
        //изменение позиции объекта на 2 единицы в секунду
        transform.position += transform.right * Time.deltaTime * speed;
    }
}
```

Вызов функции select() будет осуществляться в скрипте, связанном с виртуальной камерой. Создайте новый скрипт следующего вида и прикрепите его к камере (не забудьте передать камеру в качестве параметра):

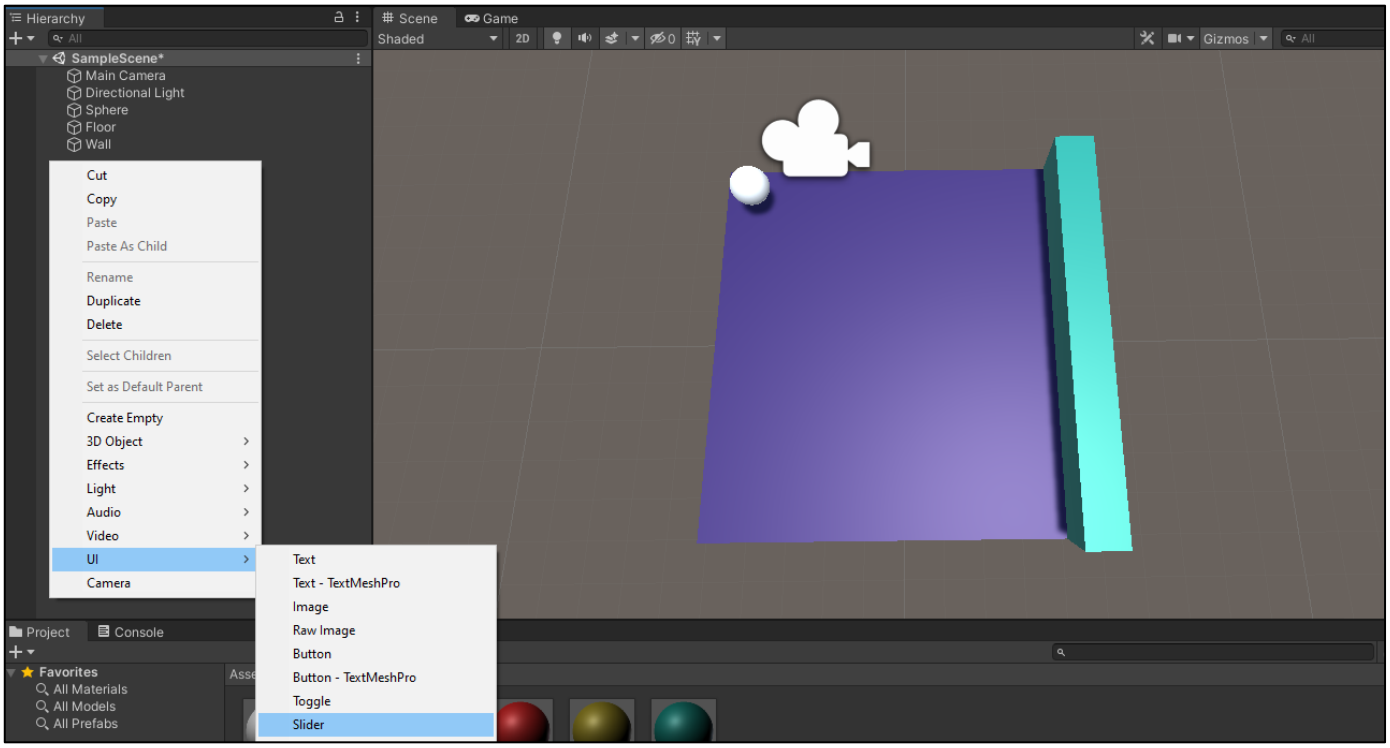
```
public class CameraScript : MonoBehaviour
{
    public Camera cam; //ссылка на камеру

    Ссылок: 0
    void LateUpdate()
    {
        RaycastHit hit; //структура для хранения информации о найденых пересечениях
        Ray ray = cam.ScreenPointToRay(Input.mousePosition); //проекция луча из позиции курсора мышы через сцену

        if (Physics.Raycast(ray, out hit)) //поиск пересечений луча и объектов сцены
        {
            Transform objectHit = hit.transform;

            BallScript bs;

            //если объект с которым произошло пересечение содержит BallScript
            if (objectHit.TryGetComponent<BallScript>(out bs) == true)
            {
                bs.select(); //вызов функции смены материала

                if (Input.GetAxis("Fire1") > 0) //если нажата левая кнопка мыши - запуск движения
                {
                    bs.start = true;
                }
            }
        }
    }
}
```

Обратите внимание, что действия скрипта происходят в LateUpdate, это необходимо для того, чтобы смена материала при выделении происходила после возврата исходного материала в скрипте сферы.

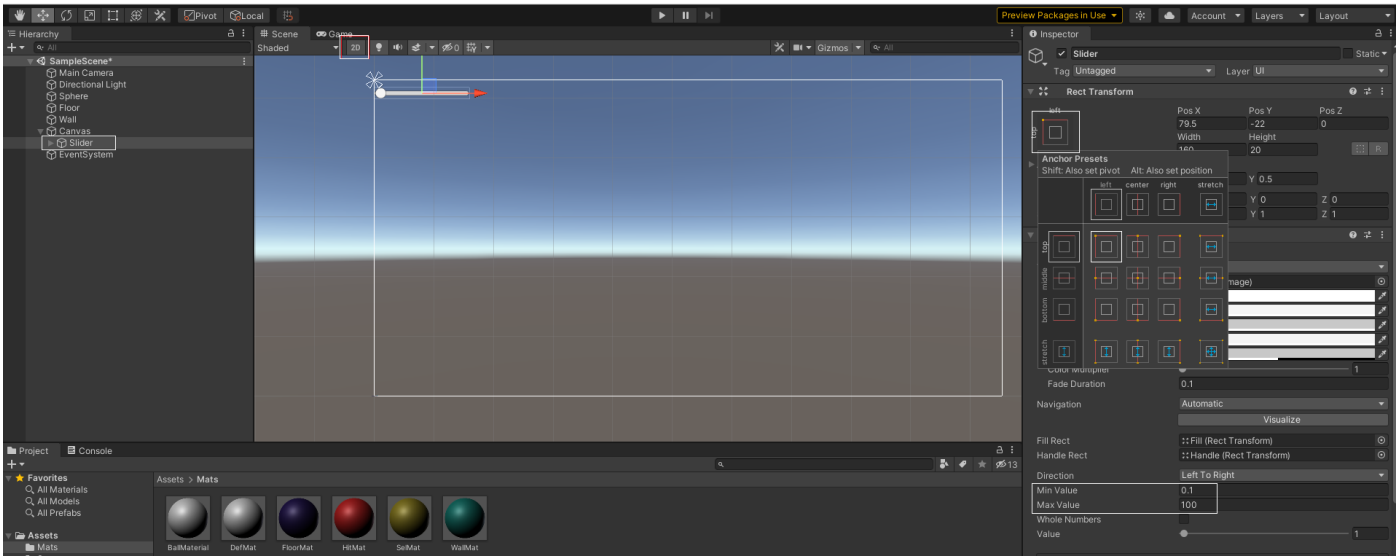
### Элементы интерфейса

Поскольку запуск сферы теперь управляется пользователем, имеет смысл вынести возможность изменять скорость сферы в пользовательский интерфейс.

Добавить элемент управления можно кликнув правой кнопкой в иерархии объектов сцены и выбрав элемент из списка UI:

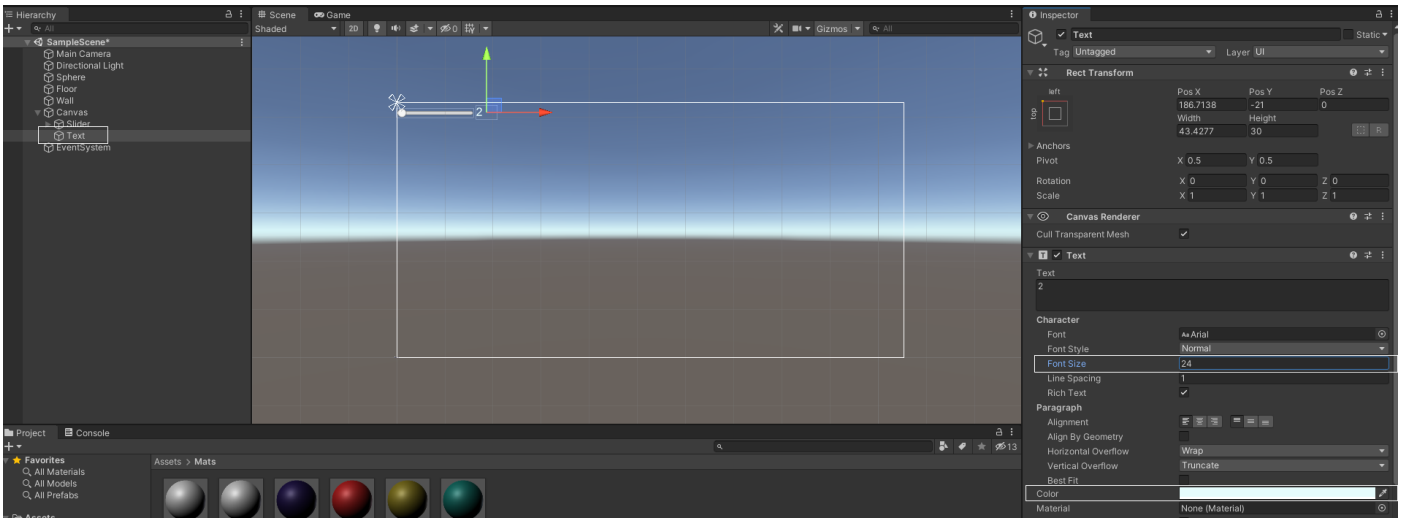



Позиционировать объект удобно в 2D режиме. Кликните на иконку 2D над окном сцены, а затем дважды кликните на Canvas в иерархии. Перемещать, поворачивать и масштабировать объекты интерфейса можно при помощи стандартных инструментов, либо в инспекторе объектов. Точкой привязки элементов интерфейса лучше всего использовать левый верхний угол экрана:




Обратите внимание, что для элемента Slider, необходимо указать диапазон значений изменения.

Помимо элемента Slider, для отображения параметра скорости объекта понадобится элемент Text:

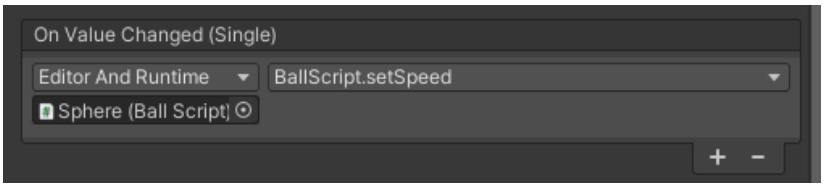


Добавьте в скрипт сферы публичную переменную текст, а также публичную функцию изменения скорости:

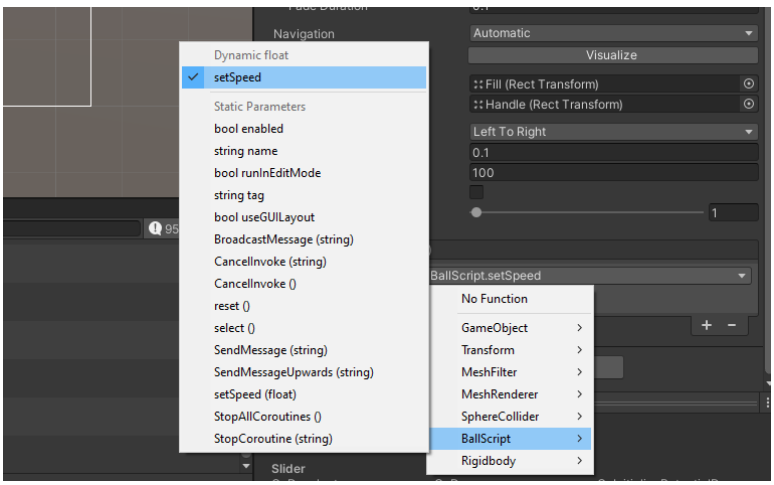
```
public Text val; //текстовое поле для отображения текущей скорости

Ссылок: 0
void Start() //вызывается один раз при запуске программы
{
    val.text = speed.ToString();
    ren = GetComponent<Renderer>(); //получение ссылки на отрисовщик сферы
}

Ссылок: 0
public void setSpeed(float value) //функция для изменения скорости при помощи полосы прокрутки
{
    Debug.Log(value.ToString());
    speed = value;
    val.text = speed.ToString();
}
```

где value – параметр, получаемый из элемента Slider.

Назначить функцию можно добавив новый обработчик события элементу Slider и указав в нём сферу:




Обратите внимание, что функцию изменения скорости нужно выбирать в разделе Dynamic float:



Если всё было сделано правильно, то теперь, при запуске проекта, вы сможете настроить скорость движения сферы при помощи полосы прокрутки.

Поскольку теперь можно изменять скорость движения сферы не перезапуская проект, имеет смысл добавить кнопку возврата сферы в исходную позицию.

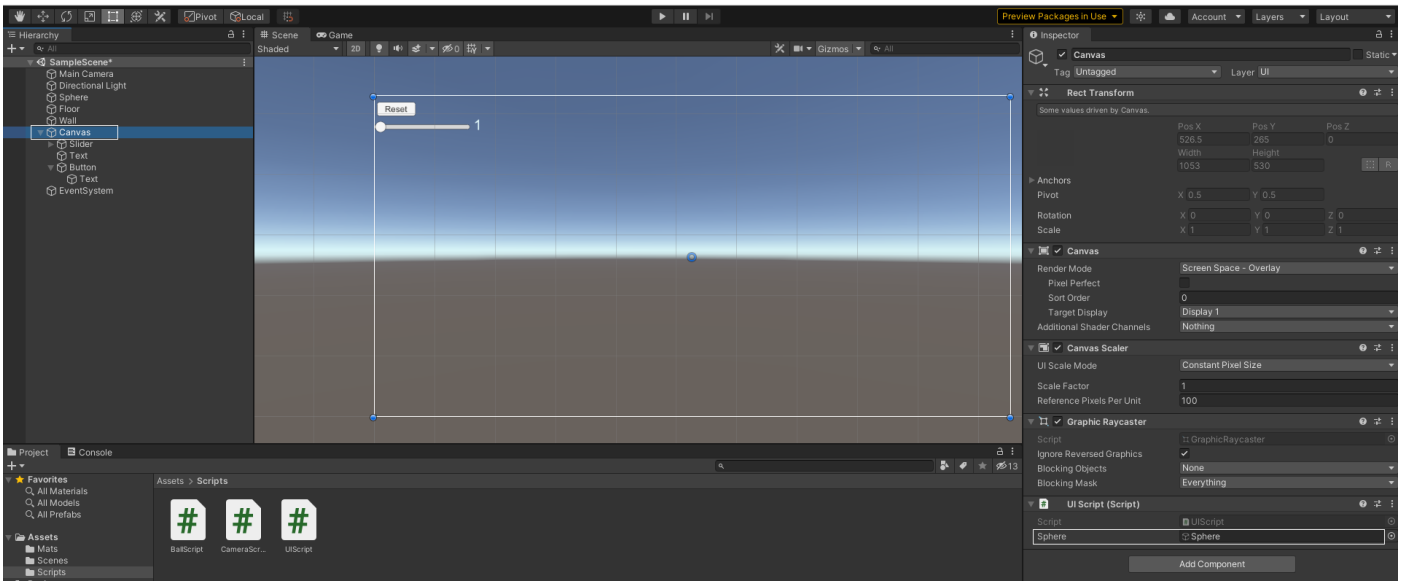
Добавьте в скрипт сферы приватную переменную для хранения исходной позиции, а также публичную функцию для сброса параметров:

```
Vector3 oldPos;
Ссылок: 0
void Start() //вызывается один раз при запуске программы
{
    oldPos = transform.position; //сохранение начальной позиции
    ren = GetComponent<Renderer>(); //получение ссылки на отрисовщик сферы
}

Ссылок: 0
public void reset()
{
    transform.position = oldPos; //возврат начальной позиции
    start = false;
    ren.material = defMat;
}
```

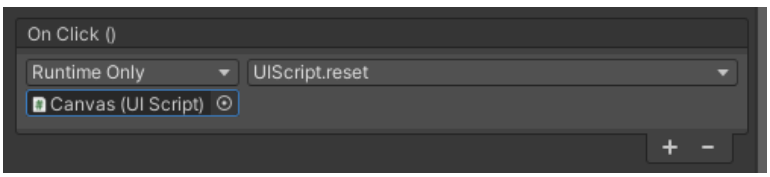
Создайте новый скрипт, который будет осуществлять вызов функции параметров у заданной сферы:

```
public class UIScript : MonoBehaviour
{
    public GameObject sphere; //ссылка на сферу
    Ссылок: 0
    public void reset()
    {
        sphere.GetComponent<BallScript>().reset(); //вызов метода reset сферы
    }
}
```

Назначьте новый скрипт объекту Canvas (не забудьте передать сферу в качестве параметра):




Добавьте в сцену кнопку и опишите для неё обработчик нажатия:




Теперь, при нажатии кнопки, сфера должна возвращаться в исходную позицию.

**Альтернативные методы перемещения**

Помимо изменения позиции рассмотренного ранее, в Unity, существует ещё четыре способа.

Использование метода Translate:

```
void Update()
{
    transform.Translate(Vector3.right * speed * Time.deltaTime);
}
```

Задание вектора скорости перемещения объекта:

```
Rigidbody rb;

Ссылок: 0
void Start()
{
    rb = GetComponent<Rigidbody>();
}

Ссылок: 0
void FixedUpdate()
{
    rb.velocity = Vector3.right * speed;
}
```

Придание ускорения объекту в определённом направлении:

```
Rigidbody rb; //ссылка на физическое тело

Ссылок: 0
void Start()
{
    rb = GetComponent<Rigidbody>(); //получение ссылки на физическое тело
}

Ссылок: 0
void FixedUpdate()
{
    rb.AddForce(Vector3.right * speed); //физическому телу задаётся ускорение в виде вектора
}
```

Изменение позиции “физического тела”:

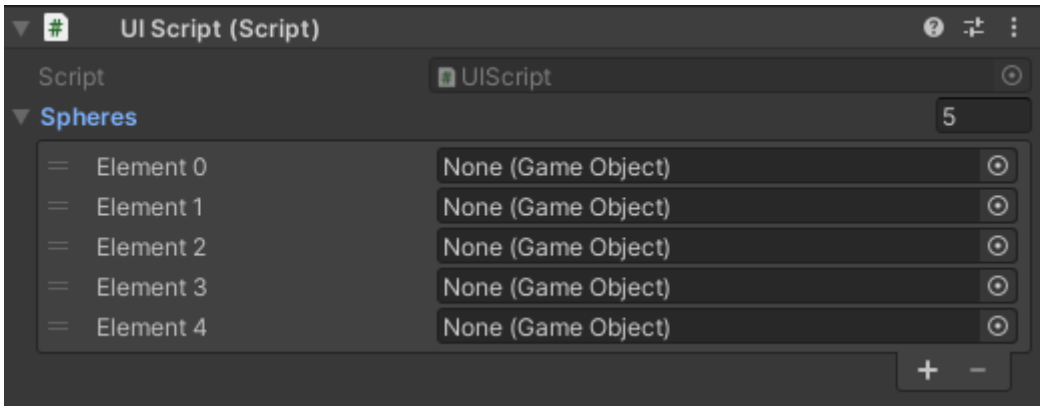
```
Rigidbody rb; //ссылка на физическое тело

Ссылок: 0
void Start()
{
    rb = GetComponent<Rigidbody>(); //получение ссылки на физическое тело
}

Ссылок: 0
void FixedUpdate()
{
    //перемещение физического тела на определённое значение
    rb.MovePosition(transform.position + (transform.right * Time.fixedDeltaTime * speed));
}
```

**Передача нескольких параметров**

Массивы могут являться публичными параметрами. Например, функцию reset можно изменить:

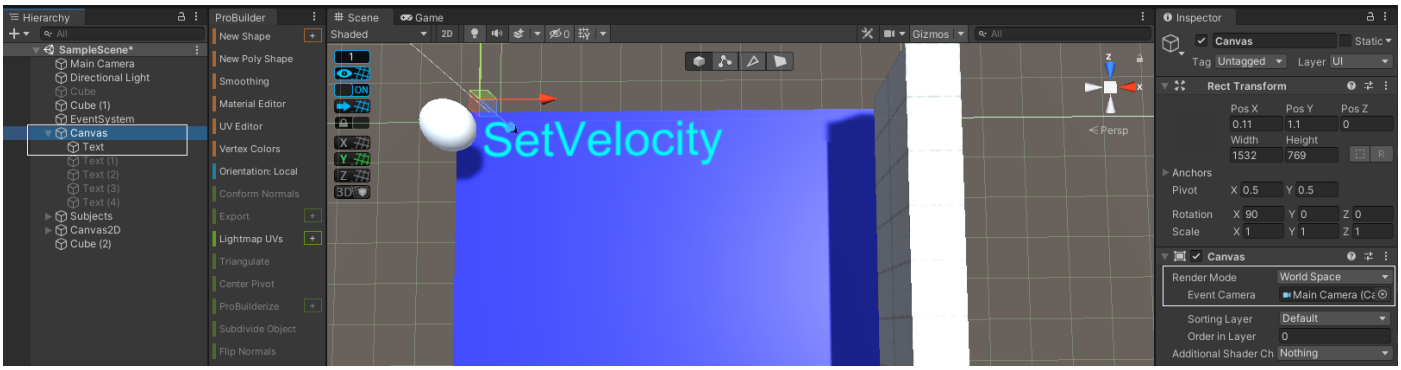
```
public GameObject[] spheres; //ссылка на несколько сфер

Ссылок: 0
public void reset()
{
    for (int i = 0; i < spheres.Length; i++)
        spheres[i].GetComponent<BallScript>().reset();
}
```

Таким образом, в качестве параметра функции можно будет указать более одного объекта:

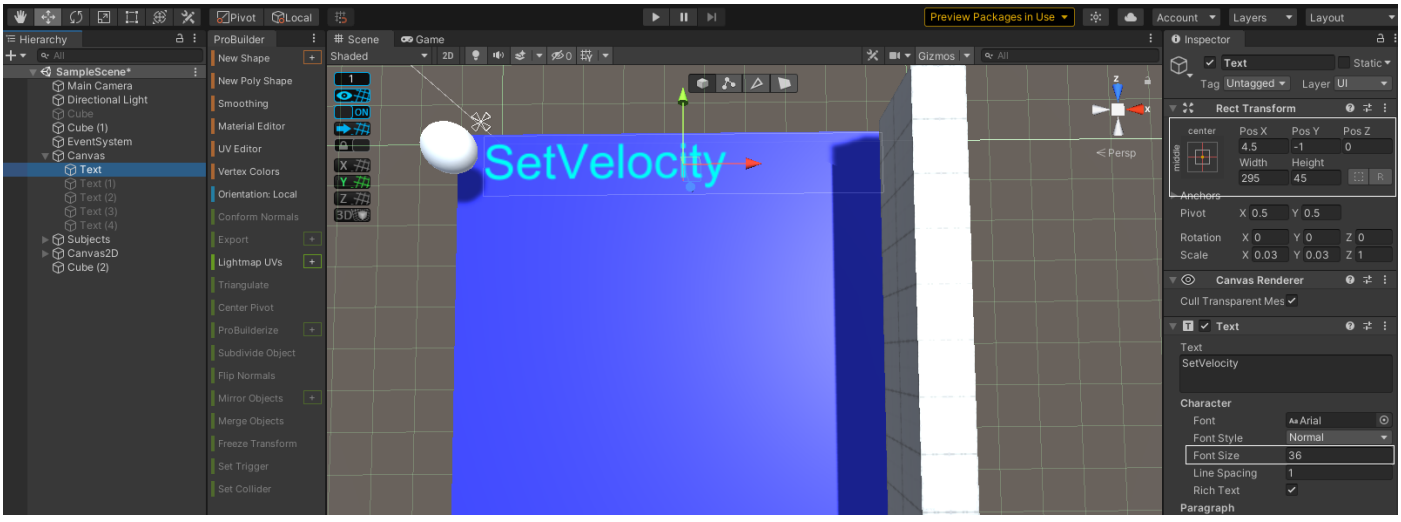



**Отображение текста в сцене**

Объекты типа Canvas могут быть использованы не только в экранной плоскости, но и в плоскости объектов трёхмерной сцены. Для этого необходимо задать параметр Render Mode = World Space:




Обратите внимание, что при добавлении текста в таком режиме, желательно использовать шрифт не слишком маленького размера и подгонять надписи под желаемый формат при помощи масштабирования элемента:

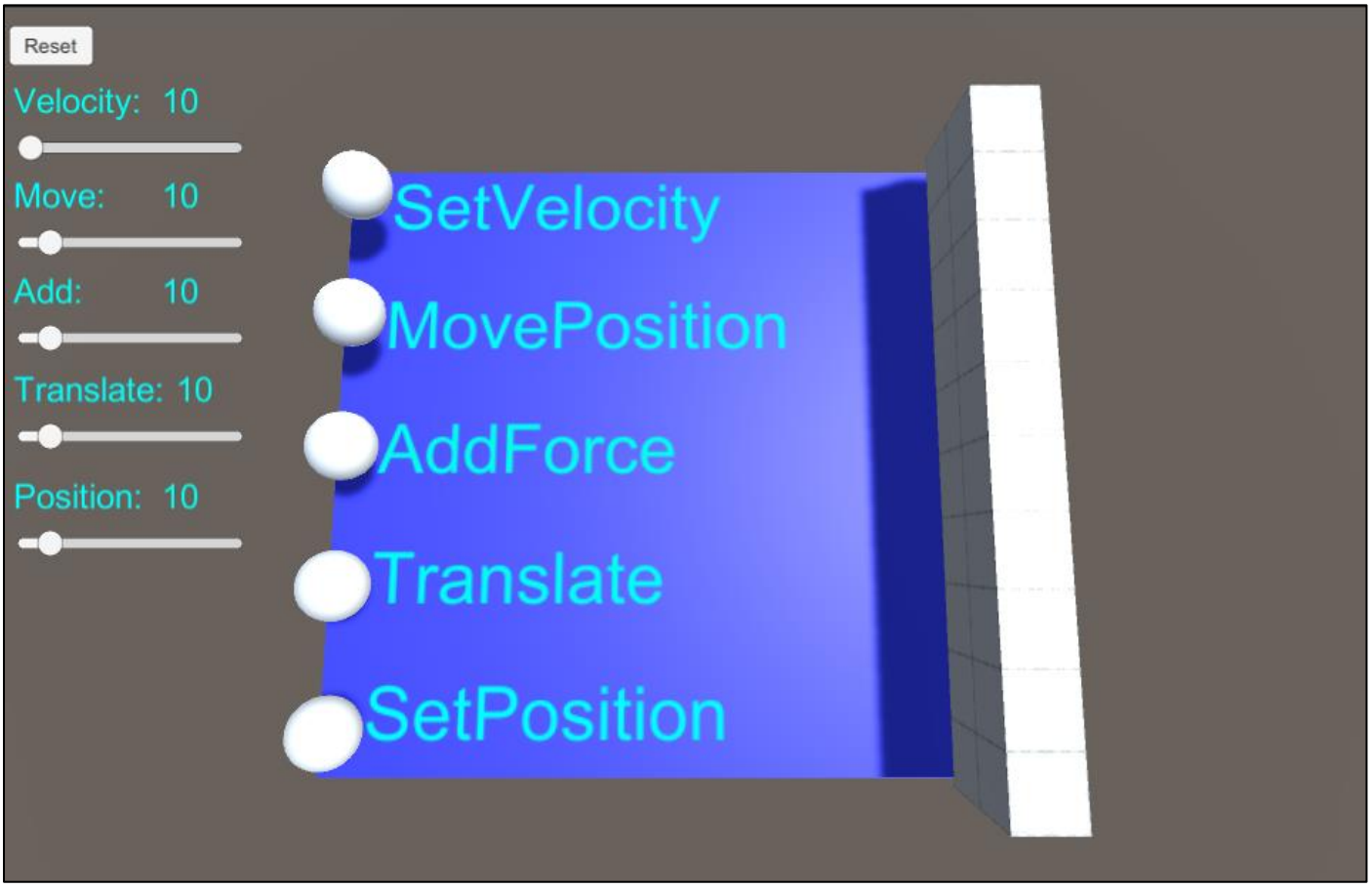


**Задание**:

Реализовать приложение, позволяющее опробовать разные способы перемещения.

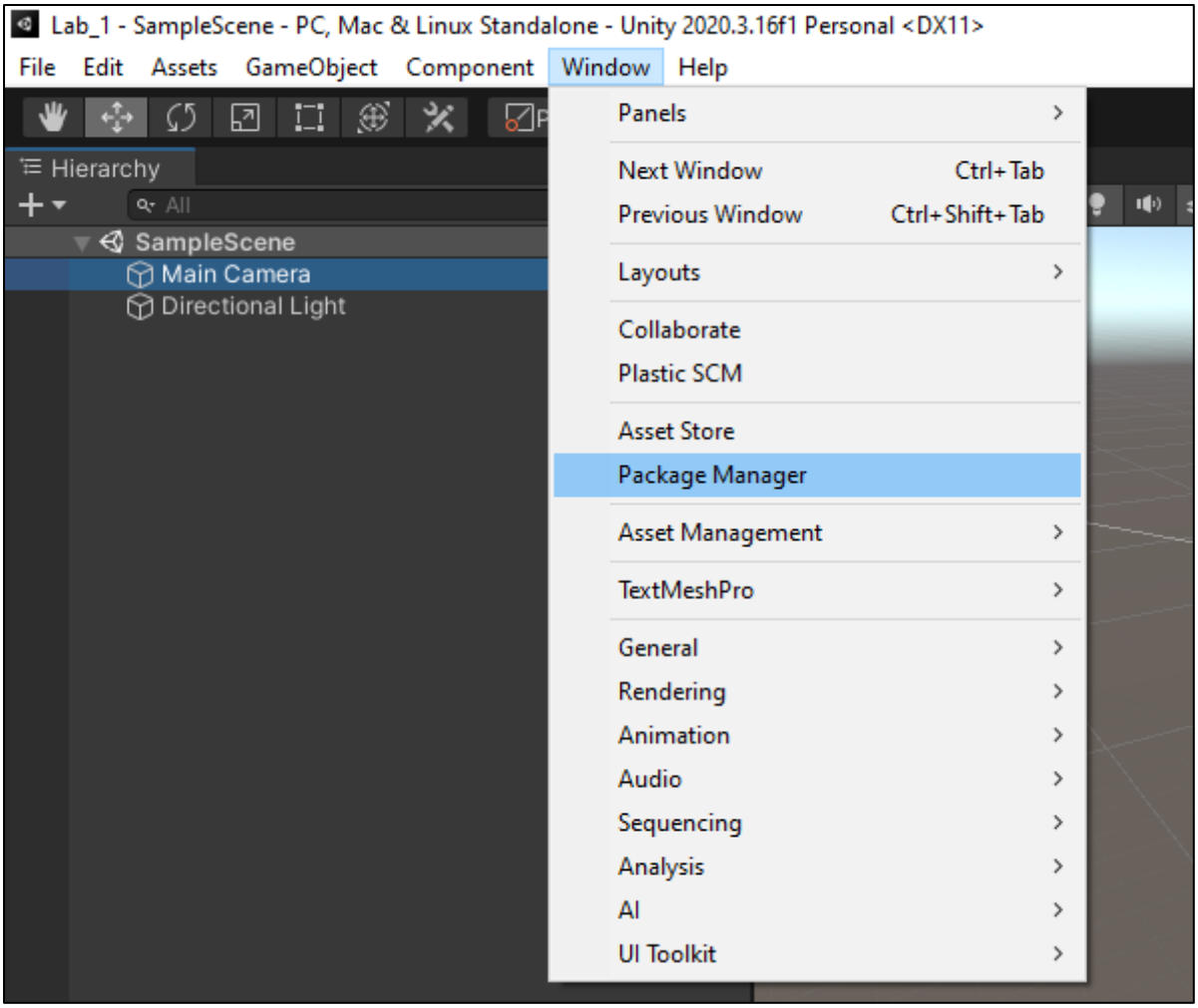
Приложение должно содержать по одной сфере, начинающее перемещение одним из описанных выше способов, при клике на неё кнопкой мыши. Скорости перемещения должны регулироваться при помощи соответствующих элементов интерфейса.

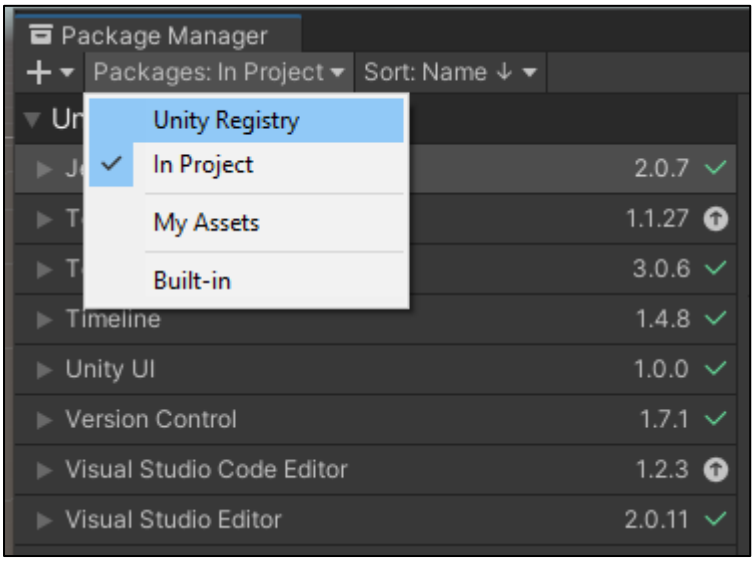
Внешний вид требуемого приложения:

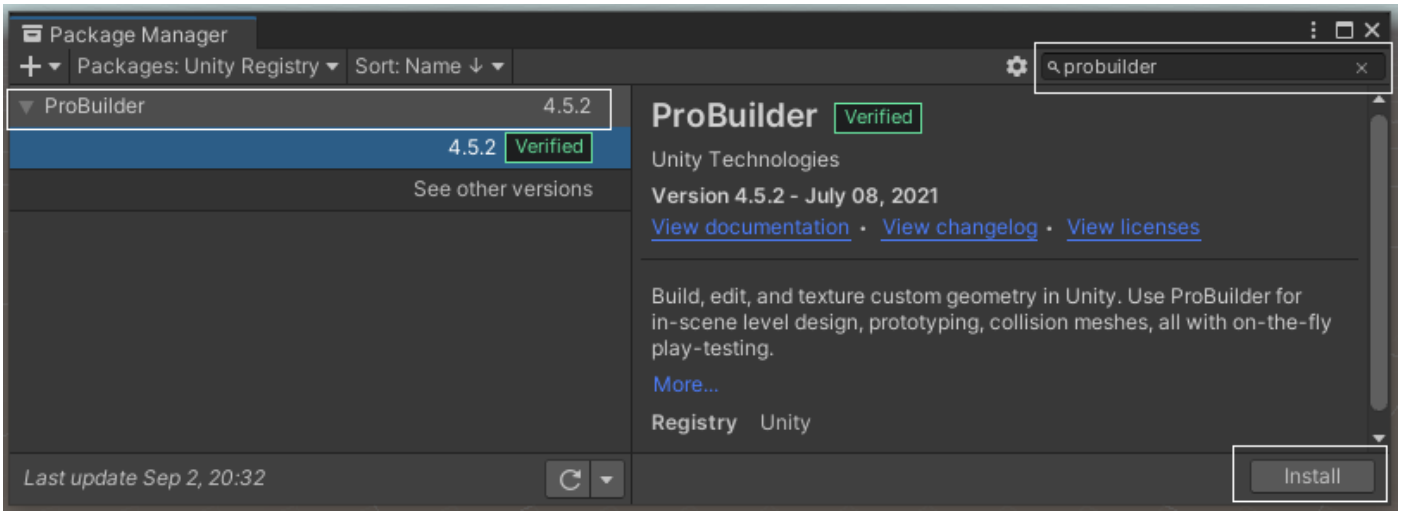


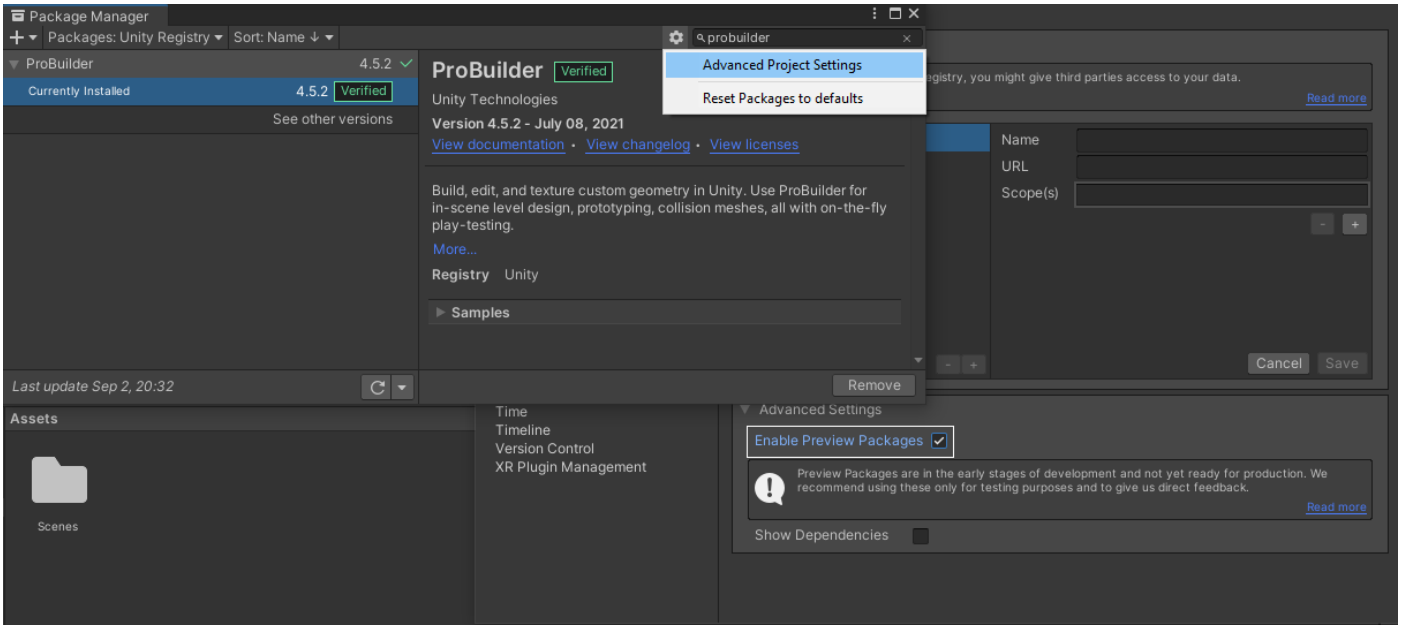

Онлайн демонстрация: https://h-anim.github.io/PTG1/

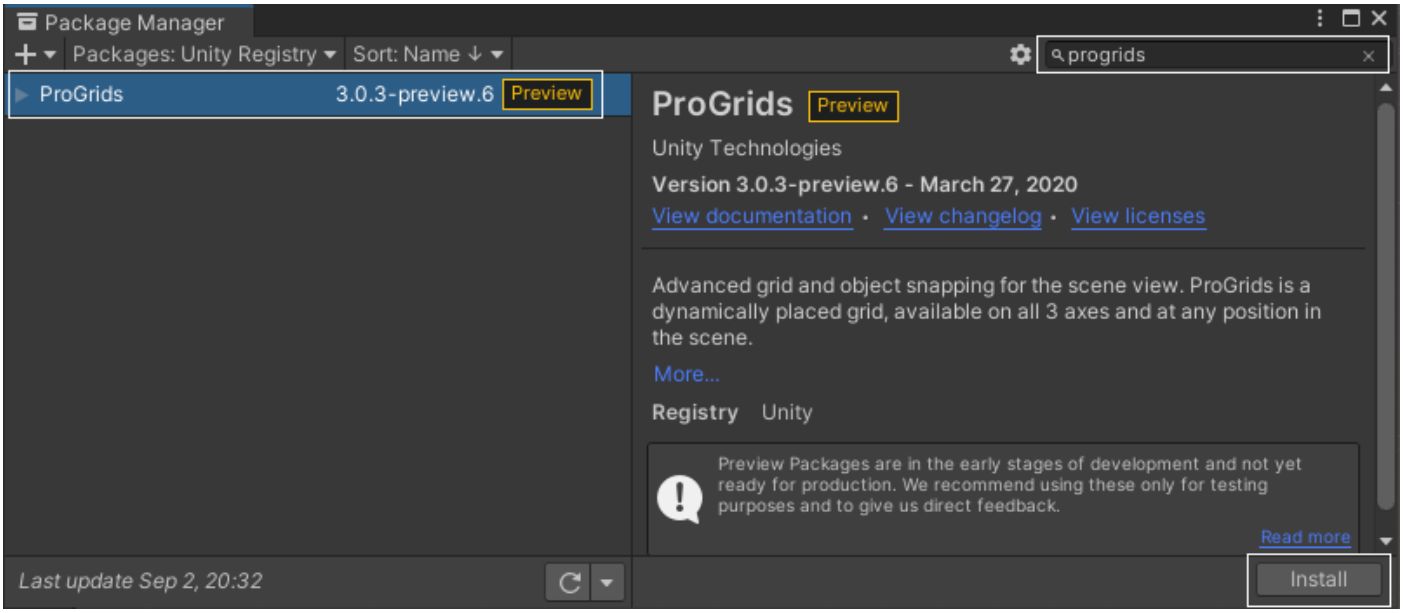
**Вспомогательные модули**

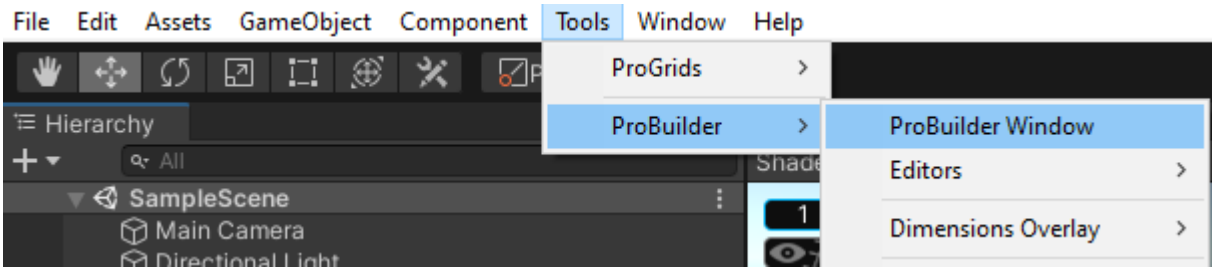

Lab_1 - SampleScene - PC, Mac & Linux Standalone - Unity 2020.3.16f1 Personal <DX11>
File
Edit
Assets
GameObject
Component
Window
Help
Hierarchy
All
SampleScene
Main Camera
Directional Light
Panels
Next Window Ctrl+Tab
Previous Window Ctrl+Shift+Tab
Layouts
Collaborate
Plastic SCM
Asset Store
Package Manager
Asset Management
TextMeshPro
General
Rendering
Animation
Audio
Sequencing
Analysis
AI
UI Toolkit


Package Manager
Packages: In Project
Sort: Name
Unity Registry
In Project
My Assets
Built-in
2.0.7
1.1.27
3.0.6
Timeline 1.4.8
Unity UI 1.0.0
Version Control 1.7.1
Visual Studio Code Editor 1.2.3
Visual Studio Editor 2.0.11


Package Manager
Packages: Unity Registry
Sort: Name
probuilder
ProBuilder 4.5.2
4.5.2 Verified
See other versions
ProBuilder Verified
Unity Technologies
Version 4.5.2 - July 08, 2021
View documentation · View changelog · View licenses
Build, edit, and texture custom geometry in Unity. Use ProBuilder for in-scene level design, prototyping, collision meshes, all with on-the-fly play-testing.
More...
Registry Unity
Last update Sep 2, 20:32
Install


Package Manager
Packages: Unity Registry
Sort: Name
probuilder
Advanced Project Settings
Reset Packages to defaults
ProBuilder 4.5.2
Currently Installed 4.5.2 Verified
See other versions
ProBuilder Verified
Unity Technologies
Version 4.5.2 - July 08, 2021
View documentation · View changelog · View licenses
Build, edit, and texture custom geometry in Unity. Use ProBuilder for in-scene level design, prototyping, collision meshes, all with on-the-fly play-testing.
More...
Registry Unity
Samples
Last update Sep 2, 20:32
Remove
Name
URL
Scope(s)
Cancel
Save
Assets
Scenes
Time
Timeline
Version Control
XR Plugin Management
Advanced Settings
Enable Preview Packages
Preview Packages are in the early stages of development and not yet ready for production. We recommend using these only for testing purposes and to give us direct feedback.
Read more
Show Dependencies


Package Manager
Packages: Unity Registry
Sort: Name
progrids
ProGrids 3.0.3-preview.6 Preview
ProGrids Preview
Unity Technologies
Version 3.0.3-preview.6 - March 27, 2020
View documentation · View changelog · View licenses
Advanced grid and object snapping for the scene view. ProGrids is a dynamically placed grid, available on all 3 axes and at any position in the scene.
More...
Registry Unity
Preview Packages are in the early stages of development and not yet ready for production. We recommend using these only for testing purposes and to give us direct feedback.
Read more
Last update Sep 2, 20:32
Install


File Edit Assets GameObject Component Tools Window Help
ProGrids
ProBuilder
ProBuilder Window
Editors
Dimensions Overlay
Hierarchy
SampleScene
Main Camera