РЫБИНСКИЙ ФИЛИАЛ ГОСУДАРСТВЕННОГО ОБРАЗОВАТЕЛЬНОГО
АВТОНОМНОГО УЧРЕЖДЕНИЯ ДОПОЛНИТЕЛЬНОГО ОБРАЗОВАНИЯ
ЯРОСЛАВСКОЙ ОБЛАСТИ
ЦЕНТРА ДЕТСКО-ЮНОШЕСКОГО ТЕХНИЧЕСКОГО ТВОРЧЕСТВА

Детский технопарк «Кванториум»

ДИДАКТИЧЕСКИЕ МАТЕРИАЛЫ

IT-КВАНТУМ

«Упражнения по основам работы в среде Unity»

Автор: Титова Ирина Игорьевна,
педагог дополнительного образования

г. Рыбинск
2020 год

Пояснительная записка

Unity – самый популярный движок для создания игр в мире. Его используют мировые гиганты типа Blizzard, Disney, NASA наравне с indie-разработчиками:

- Unity используют более 47% разработчиков игр во всем мире;
- с помощью Unity можно развернуть приложение на 20+ платформ всего одним кликом;
- визуальный редактор Unity легок в использовании и позволяет избежать трудоемкой работы с программированием.

Огромное сообщество в сети готово помочь, а официальные представители Unity Technonogies окажут профессиональную поддержку.

Цель разработки: дать педагогам удобный инструмент для знакомства обучающихся со средой Unity.

Упражнения готовы для распечатки и самостоятельной работы детей на занятии, а так же могут применяться педагогами как часть занятия с фронтальной работой. Помимо готовых скриптов они содержат вопросы, которые помогают раскрыть смысл скрипта и использовать его в дальнейшем.

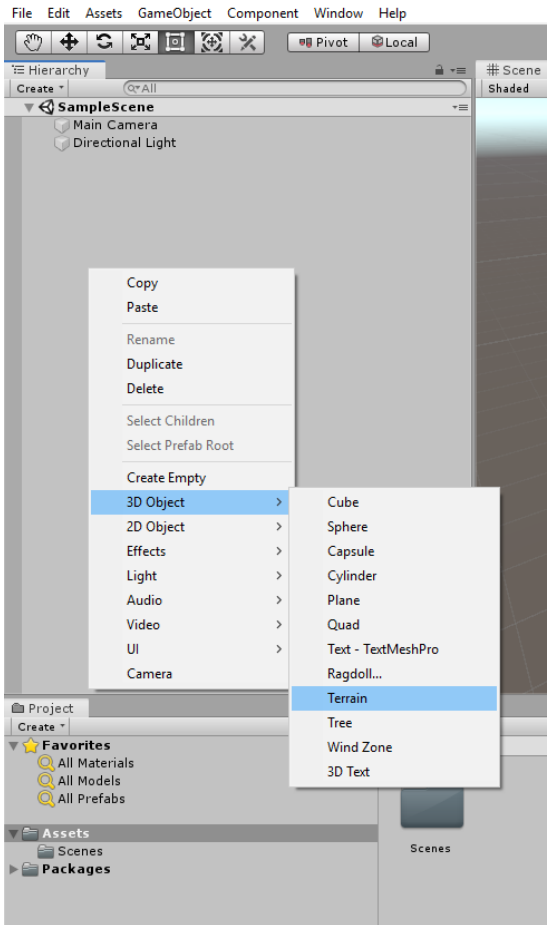
Оглавление

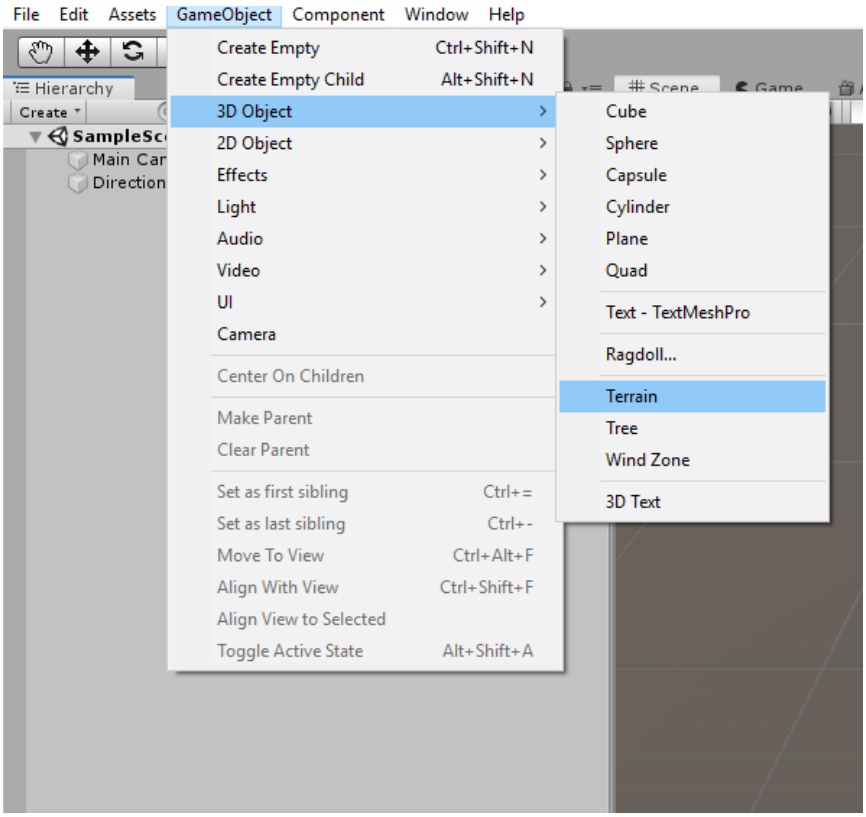
Создание ландшафта

Стандартный процесс моделирования ландшафта в Unity выглядит следующим образом:

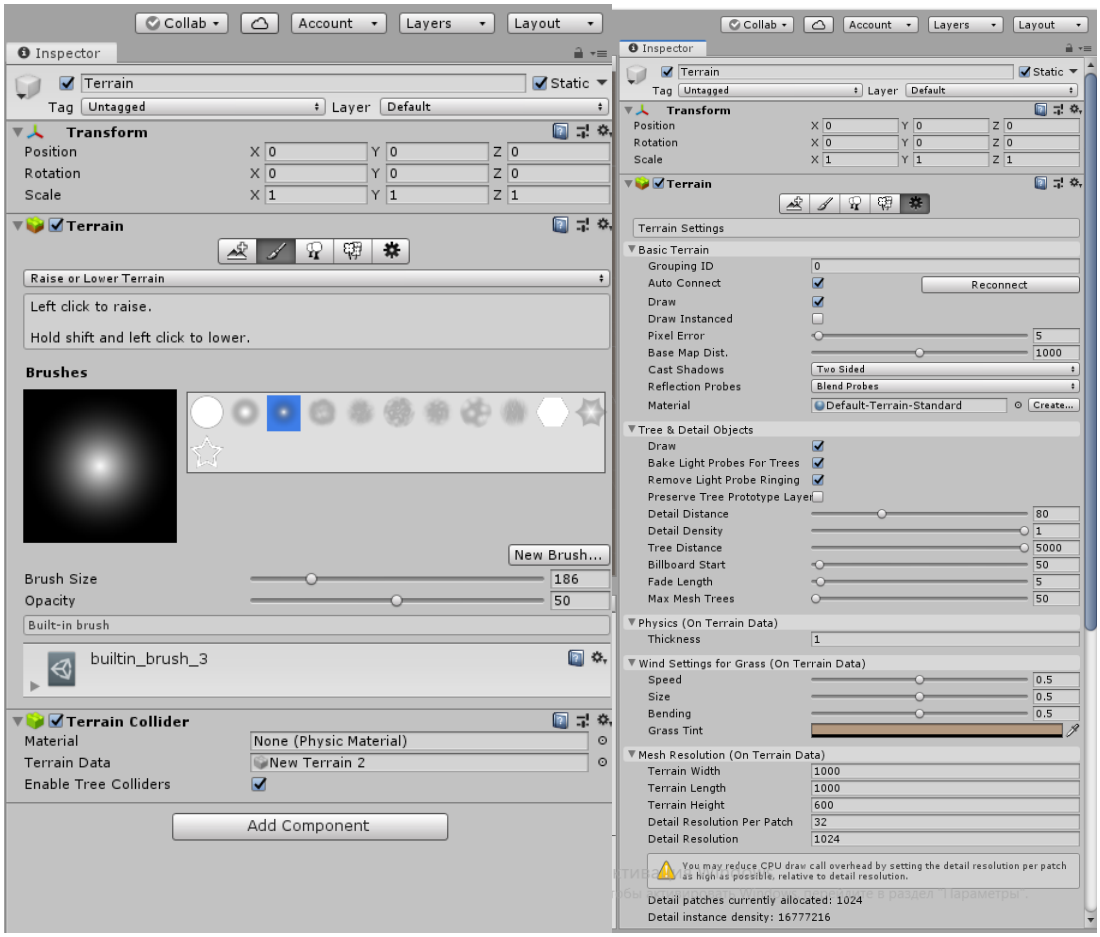
- создание и задание основных параметров (высота, длинна, ширина) для пустого объекта Terrain (выглядит в виде прямоугольной плоскости);
- создание карты высот, рельефа и его обработка при помощи инструментов скульптурного моделирования (обрабатывается исключительно из видения вашей конечной цели);
- нанесение текстур (могут быть использованы как базовые текстуры, так и созданные в различных графических редакторах; наносить текстуры рекомендуется после создания конечного рельефа, т.к. в противном случае можно получить деформированные текстуры);
- добавление различных источников света (глобальных и локальных) и создание неба (в контексте Unity именуется, как SkyBox);
- проработка деталей, особенностей местности и объектов (деревья, трава, вода, камни и т.д.);
- расстановка камер (позволяет создавать визуальные эффекты или принуждает пользователя смотреть так, как хочет создатель приложения).

1. Создание Terrain (2 способа)

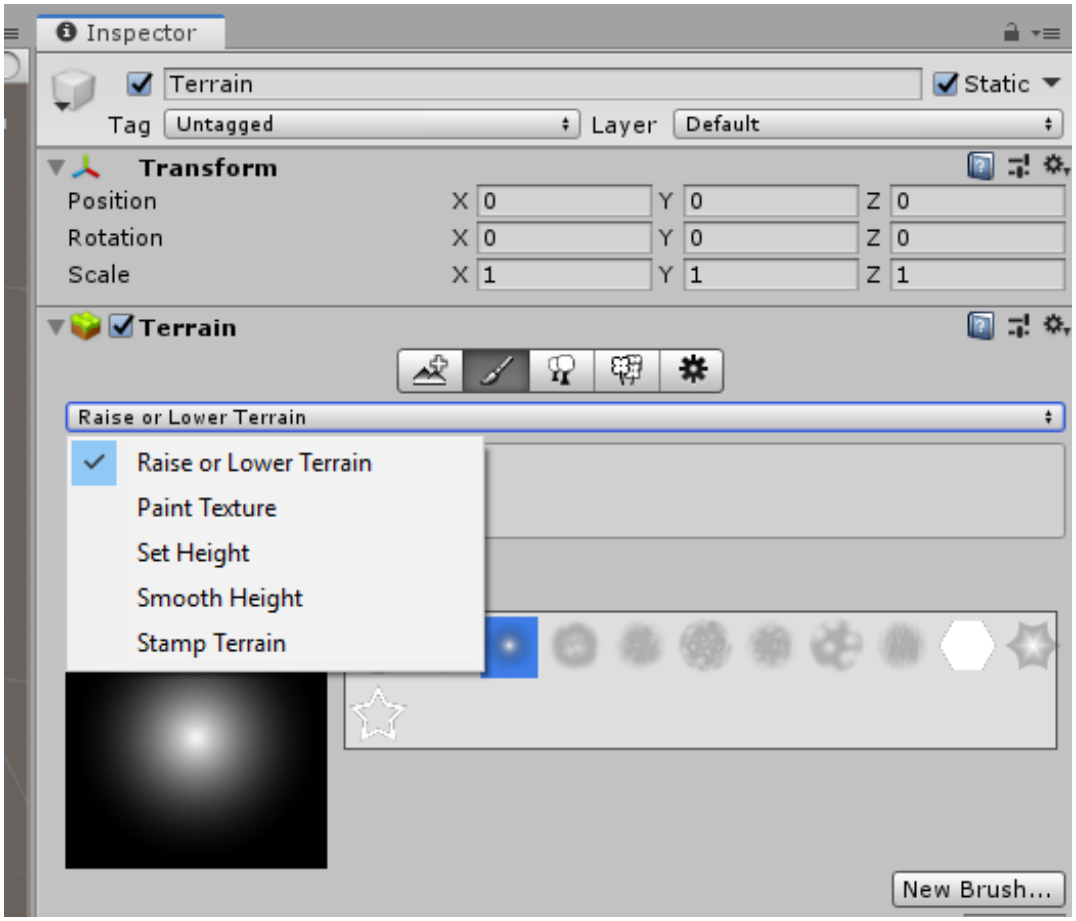




2. Свойства созданного объекта



3. Рисование ландшафта



- **Создание окрестностей** Terrain: этот инструмент создает дополнительные плитки ландшафта рядом с плиткой ландшафта, которую вы создали. Это помогает создавать большие среды и позволяет инструментам ландшафта непрерывно влиять на местность беспрепятственно.
- **Поднять или опустить местность** : позволяет поднять или опустить местность с помощью курсора мыши.
- **Paint Texture** : Это позволяет вам рисовать текстуры на местности, создавая Terrain Layers, набор текстур, которые будут смешиваться вместе, когда вы будете рисовать, чтобы быстро добавить детали к вашему ландшафту.
- **Установить высоту** : это позволяет вам установить высоту вашего ландшафта. Это полезно для создания плато, выравнивания местности или создания долин и русел рек. Вы можете установить высоту всего ландшафта, выбрав значение с помощью ползунка, а затем нажав кнопку «Свести».
- **Smooth Height** : сглаживает ландшафт до заданной высоты, создавая постепенный наклон / спад.
- **Штамп на местности** : позволяет вам штамповать ландшафт на указанной высоте.

Инструмент Tree распознает SpeedTree, сторонний продукт IDV Inc., который предоставляет готовые древовидные активы и программное обеспечение для моделирования. Эти кисти позволяют рисовать деревья, листья, траву и камни прямо на местности, сокращая тем самым необходимость размещать каждый объект вручную. Этот инструмент работает как Paintbrush, но с одной оговоркой: вы должны определить дерево для размещения кисти, так как сами деревья являются сетками. (**Рисунок 06**)

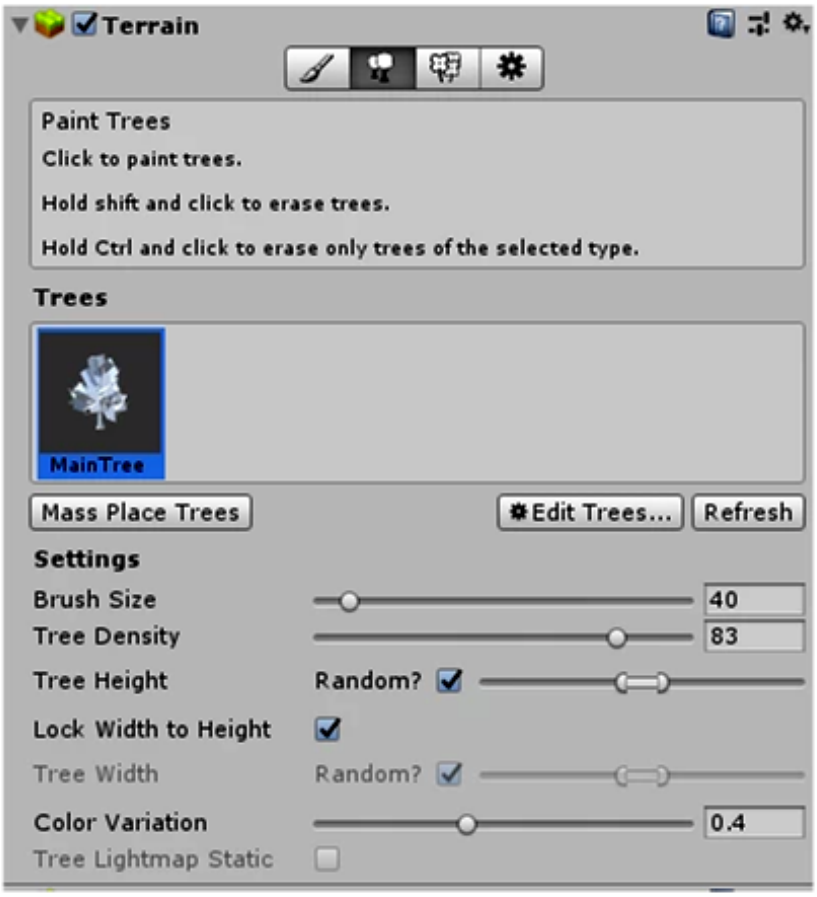
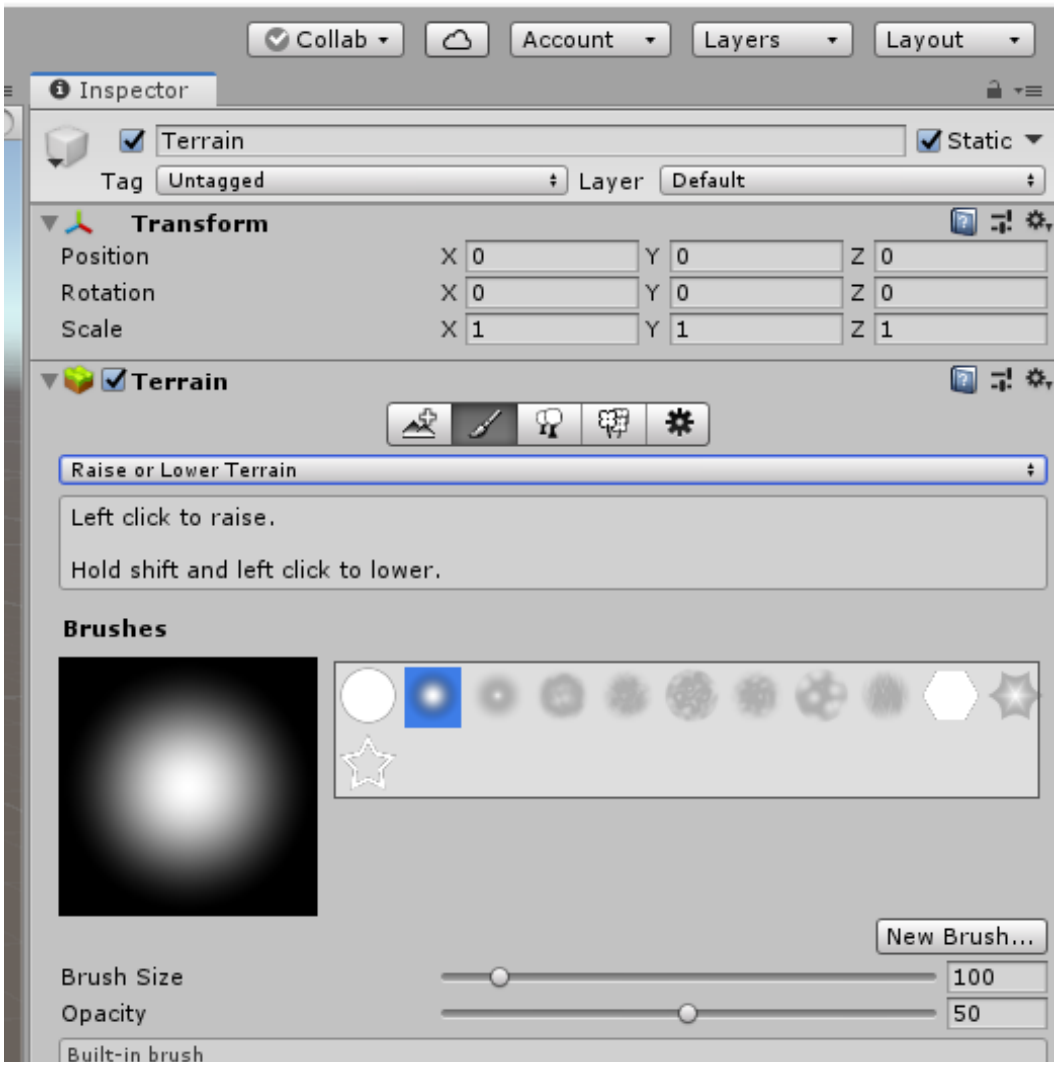


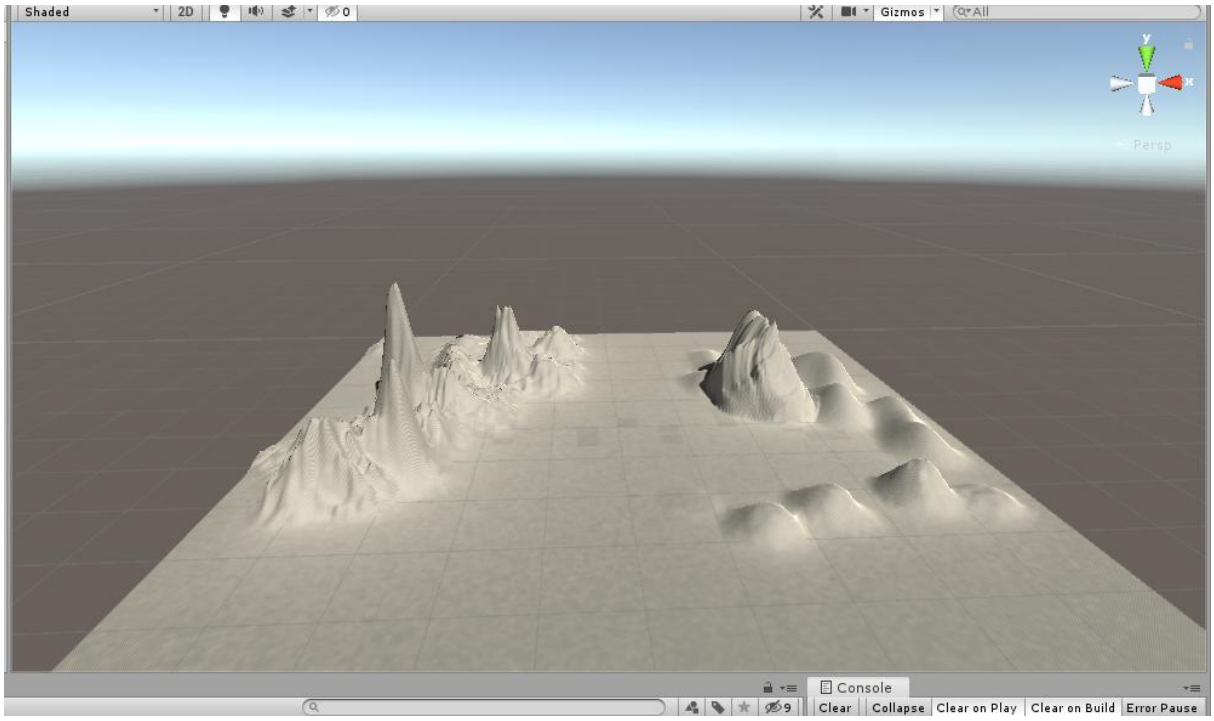
Рисунок 06: Кисть для дерева

Задание:

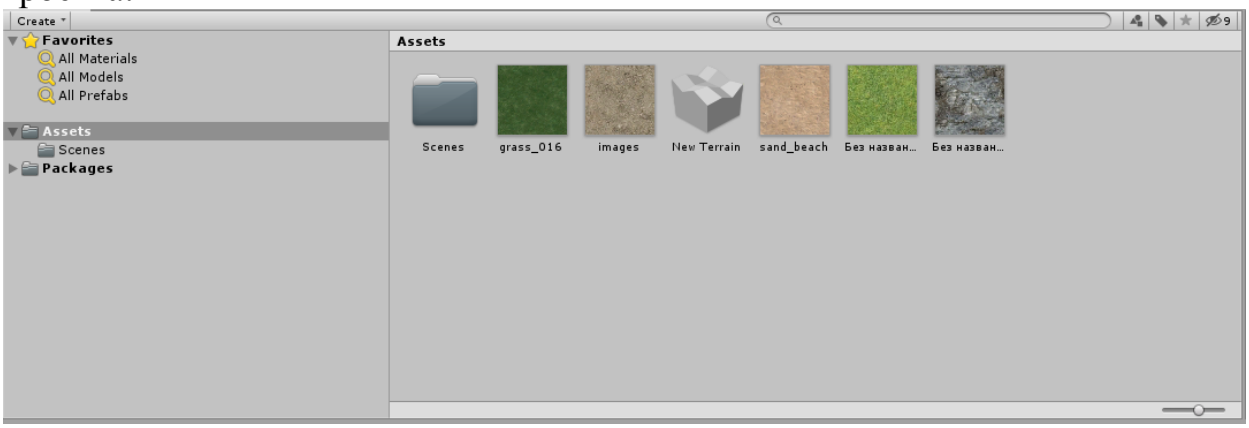
Используя режим Raise or Lower Terrain и различные кисти нарисуйте горный ландшафт. Если использовать кисть с зажатым Shift , то ландшафт становится вогнутым, таким образом рисуют различные лощины, овраги и т.д.



Например, что-то вроде этого:



Введите в браузере: бесшовные текстуры …(камень, песок, трава и т.д.). Создайте в своей папке на рабочем столе подпапку «Текстуры», и скачайте несколько вариантов. Перетащите их из своей папки в папку проекта:



Чтобы начать использовать текстуры, мы должны создать текстуры, которые будут применяться к ландшафту в редакторе ландшафта. Чтобы начать этот процесс:

- Нажмите кнопку кисти на панели инструментов выбранного ландшафта в окне инспектора.
- Нажмите на выпадающий список и выберите опцию Paint Texture (**рисунок 01**).

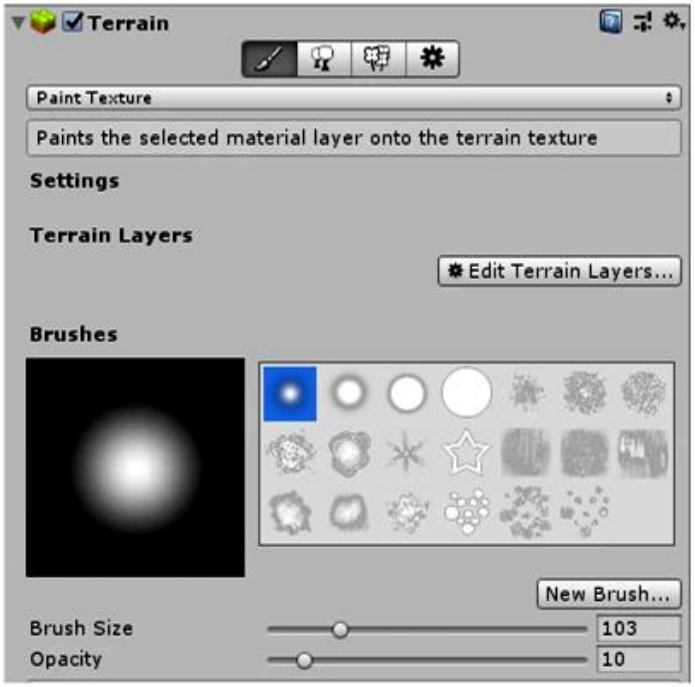
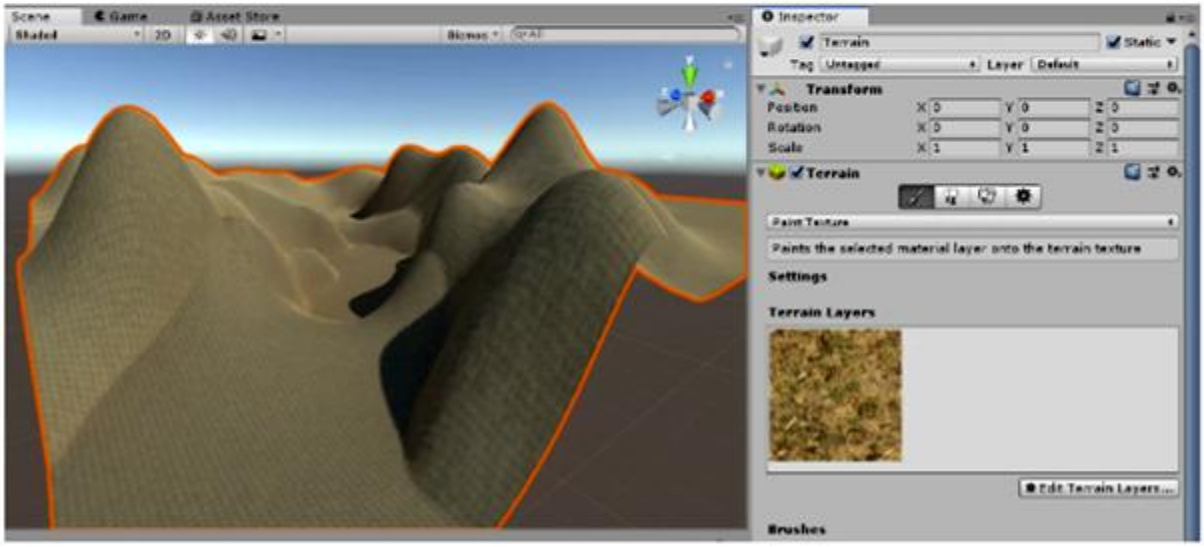


Рисунок 01: Пустые слои Terrain

Материалы рельефа определяются как слои рельефа. Первоначально ландшафт не будет иметь слоев Terrain, созданных для рисования. Нам нужно добавить несколько слоев, чтобы начать рисовать. Самый первый слой, который вы создадите, будет применен ко всей местности как фоновый материал. Чтобы начать добавлять слои Terrain:

Нажмите Редактировать слои Terrain.

- Нажмите Create Layer ... Это откроет окно Unity Asset Picker.
- Найдите текстуру фона, которую вы хотите использовать. После того, как вы выберете текстуру, Unity будет применять ее ко всему ландшафту в качестве фонового материала (**рисунок 02**).



Повторите описанные выше шаги, чтобы создать новый слой ландшафта, который мы можем использовать для рисования ландшафта (**рисунок 03**).

Рисунок 03: Слой Terrain с добавленной текстурой травы

Теперь, когда вы добавили дополнительную текстуру, вы заметите, что, выбрав любой из материалов, вы увидите дополнительные настройки для этих текстур чуть ниже секции Terrain Layers (**рисунок 04**).

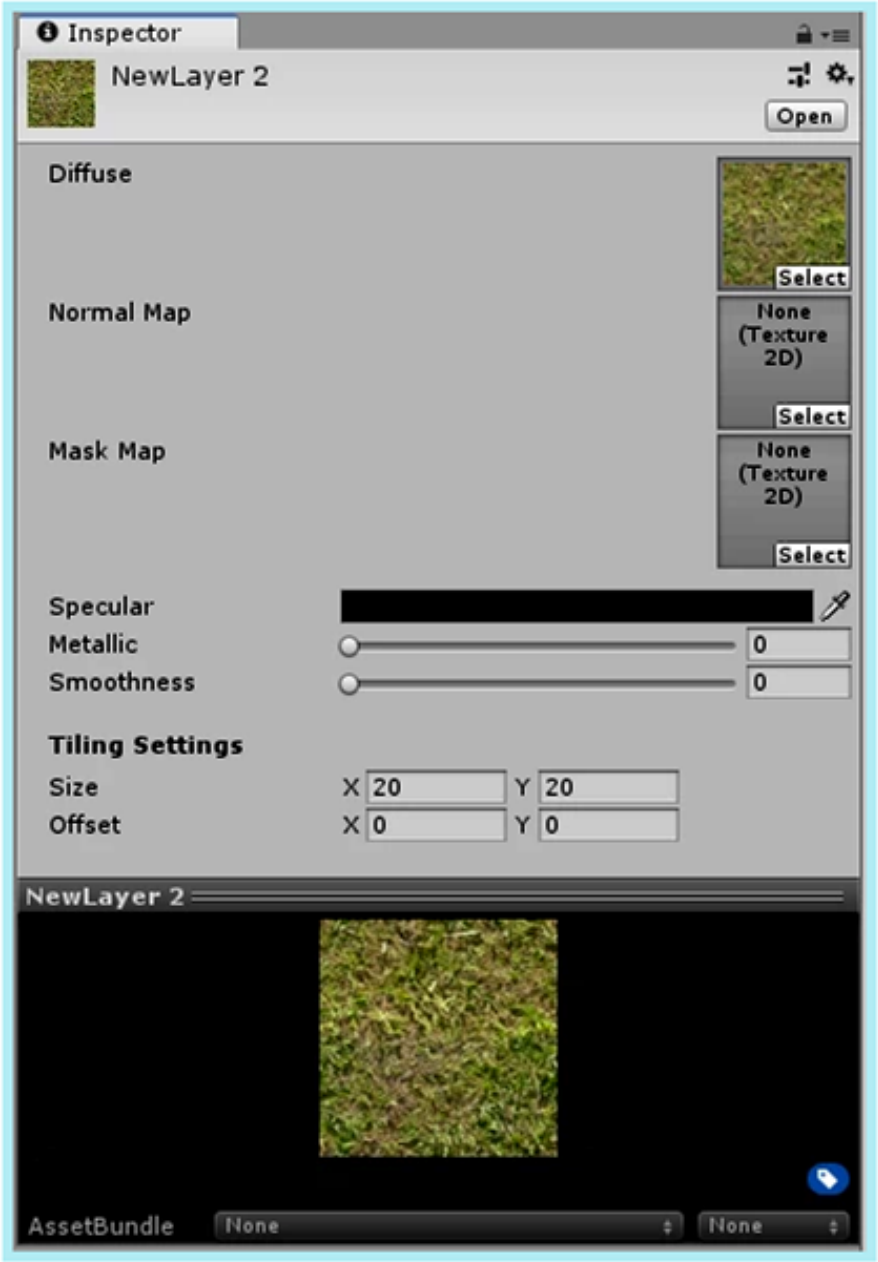
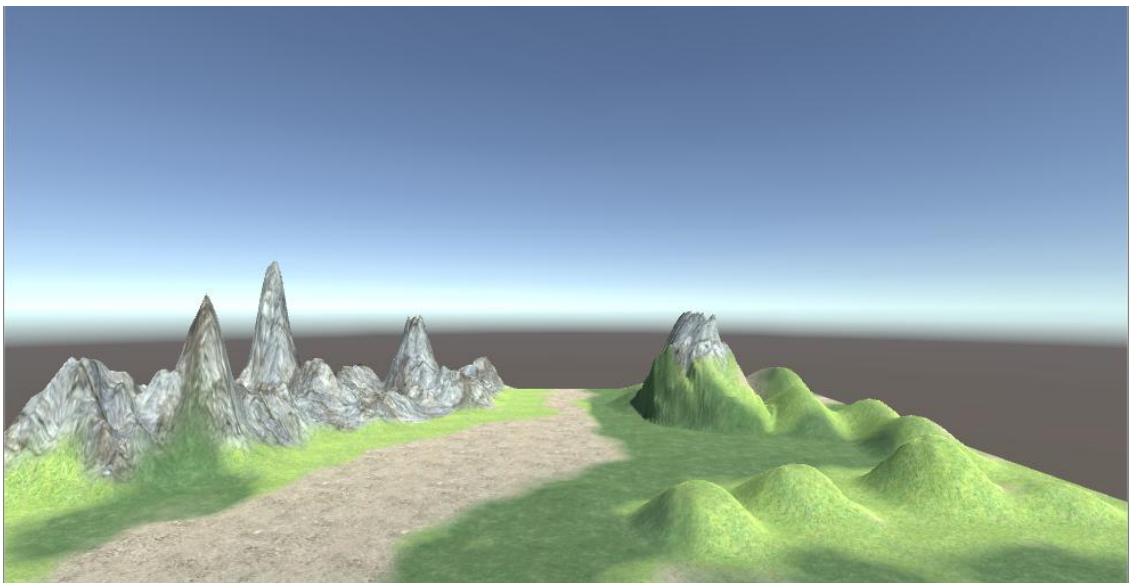


Рисунок 04: Свойства Terrain Layer

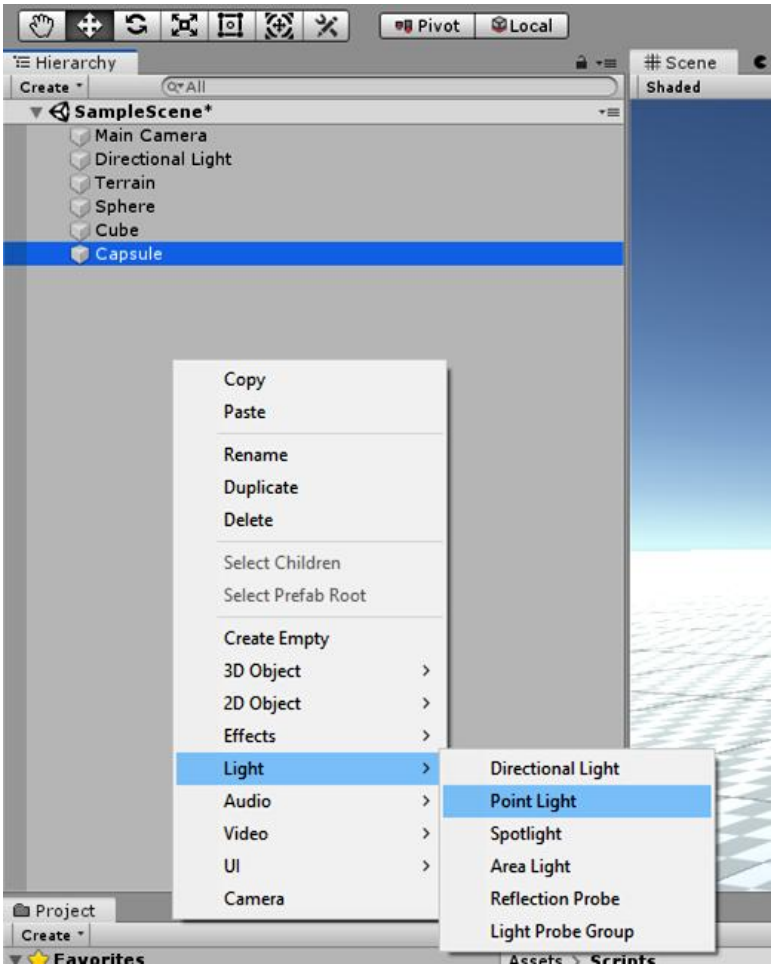
Теперь, когда у вас есть несколько Материалов, настроенных в слоях Terrain, пришло время рисовать ландшафт. Выберите кисть, которую вы хотите использовать, затем нажмите и перетащите по местности (**Рисунки 05 и 06**).



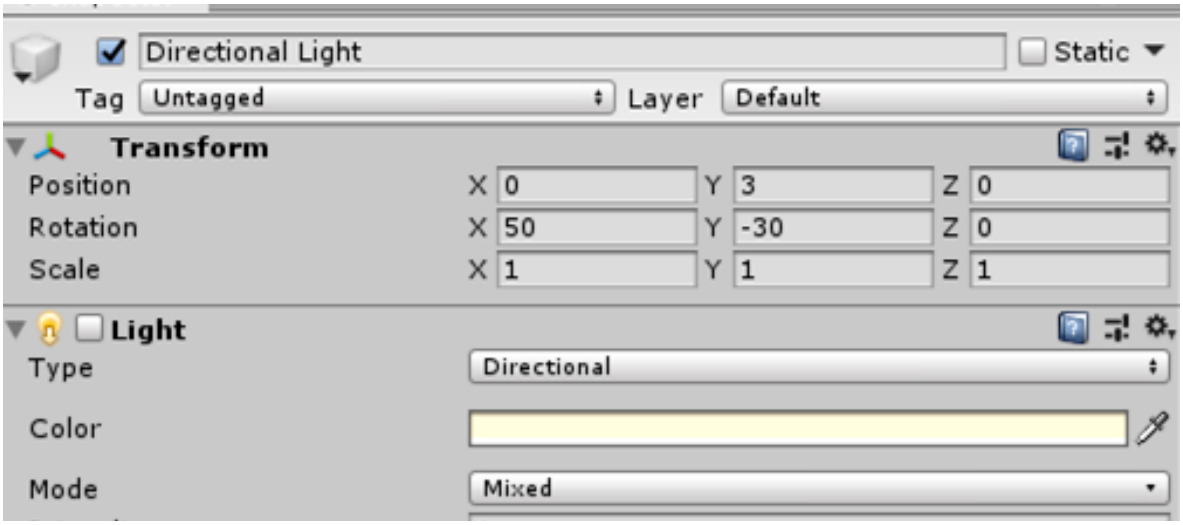
Освещение

Освещение важный компонент любой сцены, он позволяет создать нужное настроение.

Создадим один из вариантов источников света – точечный.

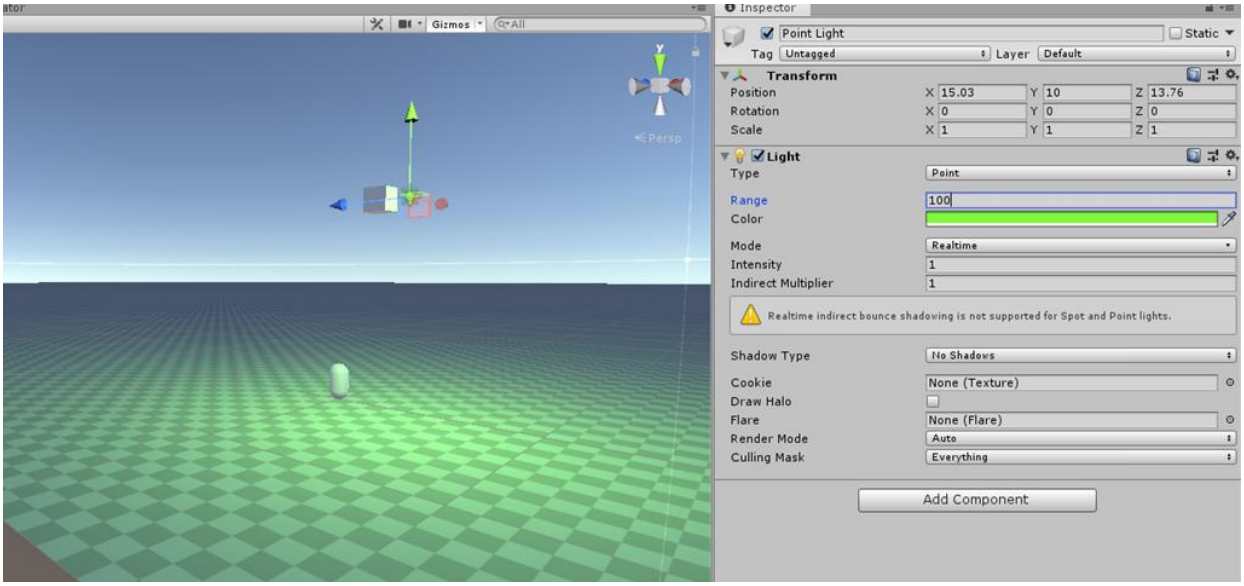


Отключите Directional Light, для этого нажмите на него в окне иерархии и уберите галочку в окне инспектор.



И поменяйте координаты у Point Light (Position), меняя координаты, включайте режим игры и смотрите изменения.

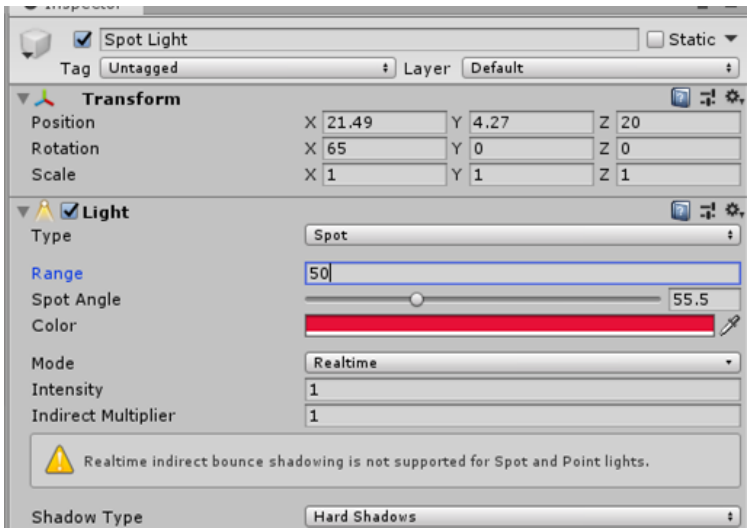
У света можно поменять некоторые из свойств:



Color – цвет, Range – распространение света, Shadow Type – тип тени . Поменяйте каждый параемтр и посмотрите, что выйдет.

Еще один вид освещения Spot Light.

В отличие от точечного светильника, этот свет работает только в одном направлении, как фонарь или лампа. Создайте такой светильник и настройте следующим образом:



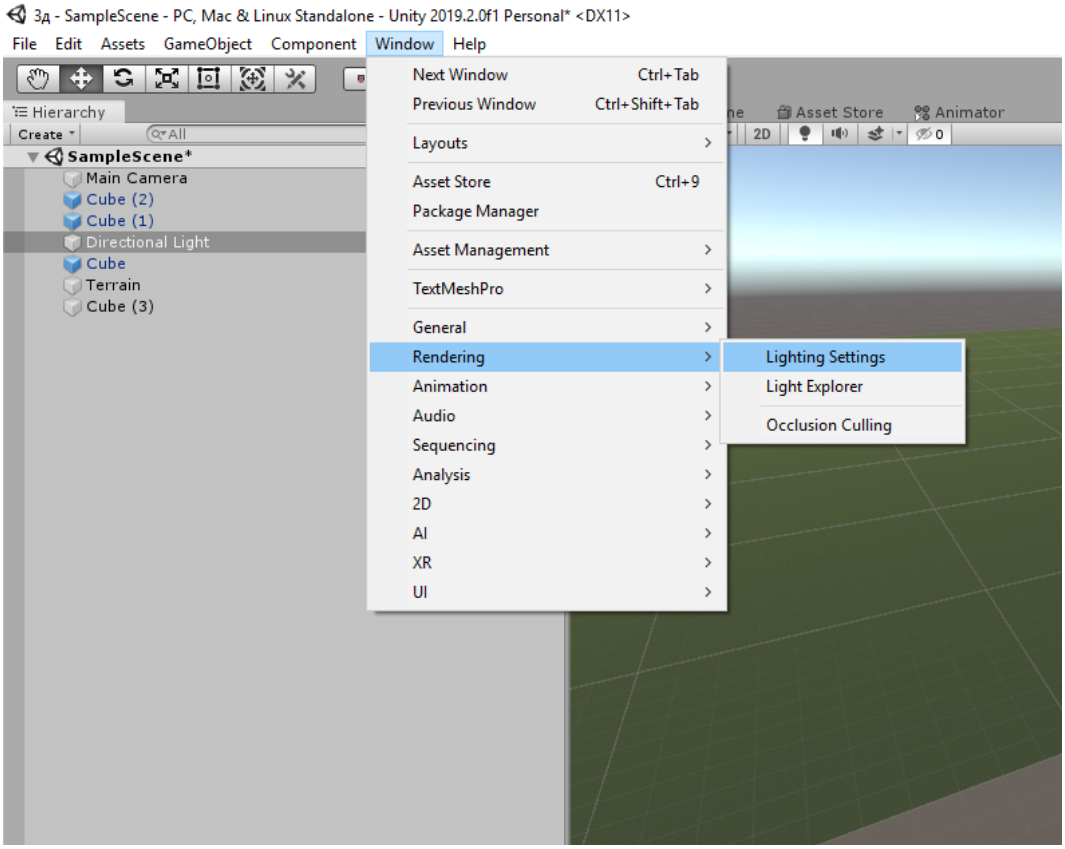
Основной свет (Directional Light) – представляет собой свет большого объекта (например солнца), он равномерно освещает всю поверхность.

Выберите Directional Light на сцене, зажмите E и повращайте, вы увидете как меняется освещение , таким образом можно задать время суток в вашем проекте.

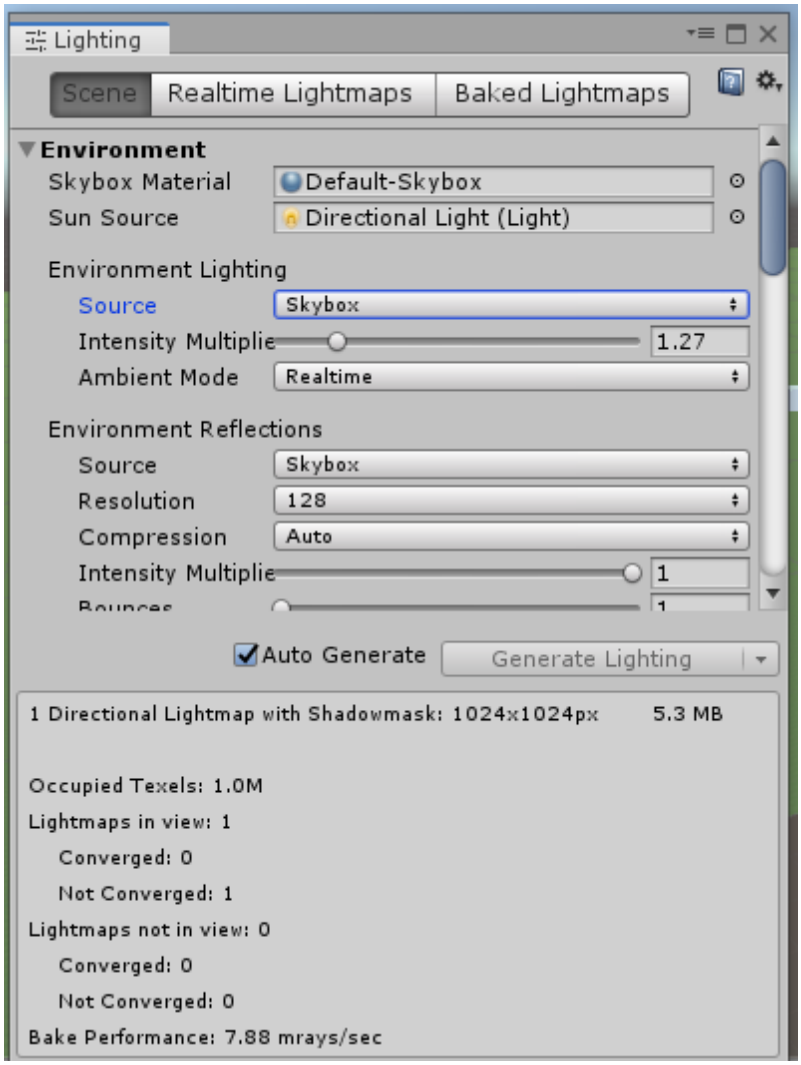
Настройка неба и глобального освещения

Откройте проект «упражнение» или проект с игрой.

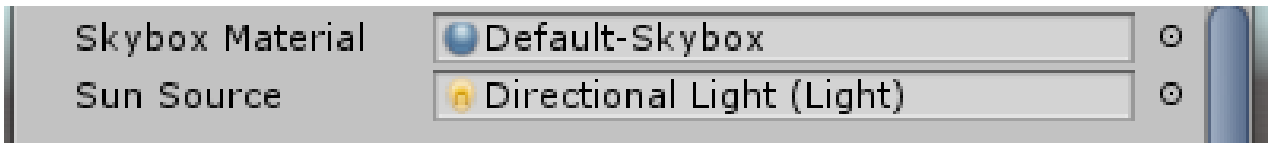
Кроме различных источников света в юнити за освещение отвечает так называемое «Глобальное освещение» (GL) , которое настраивается сразу для всех сцен. На изображении ниже указано как вызвать настройки.



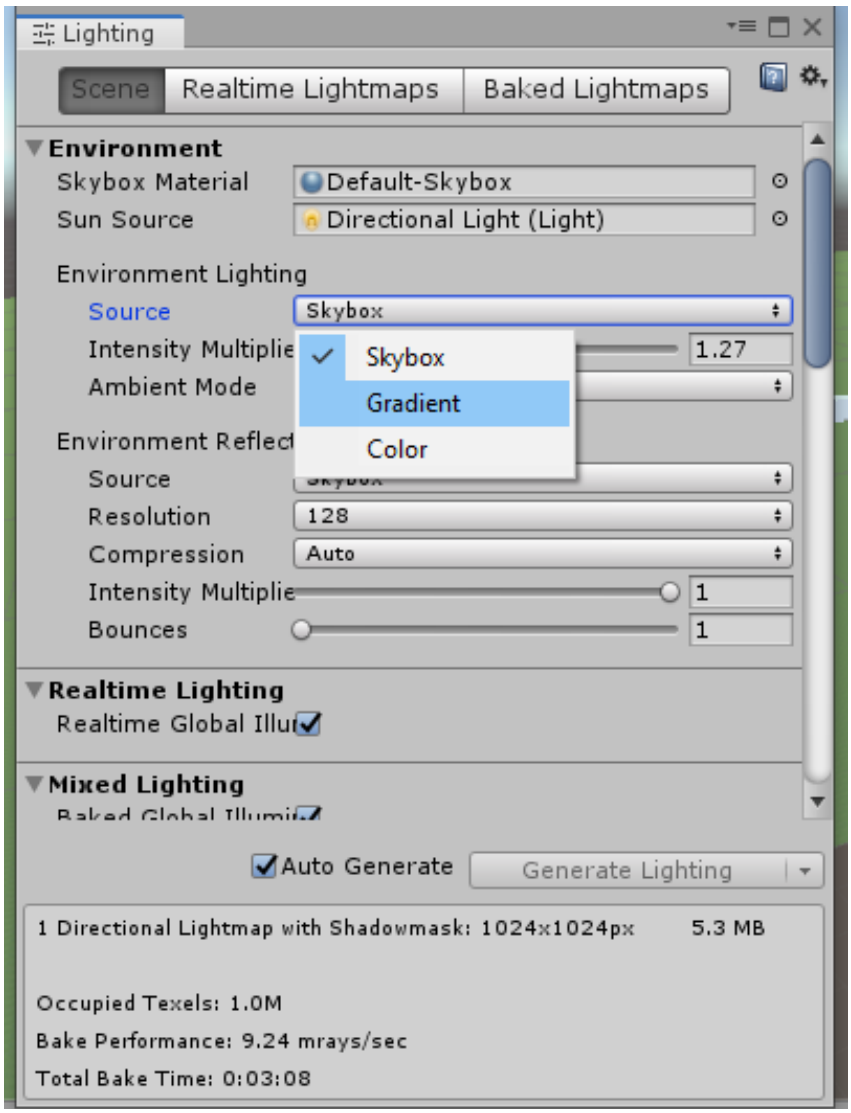
По умолчанию настройки имеют примерно следующий вид:

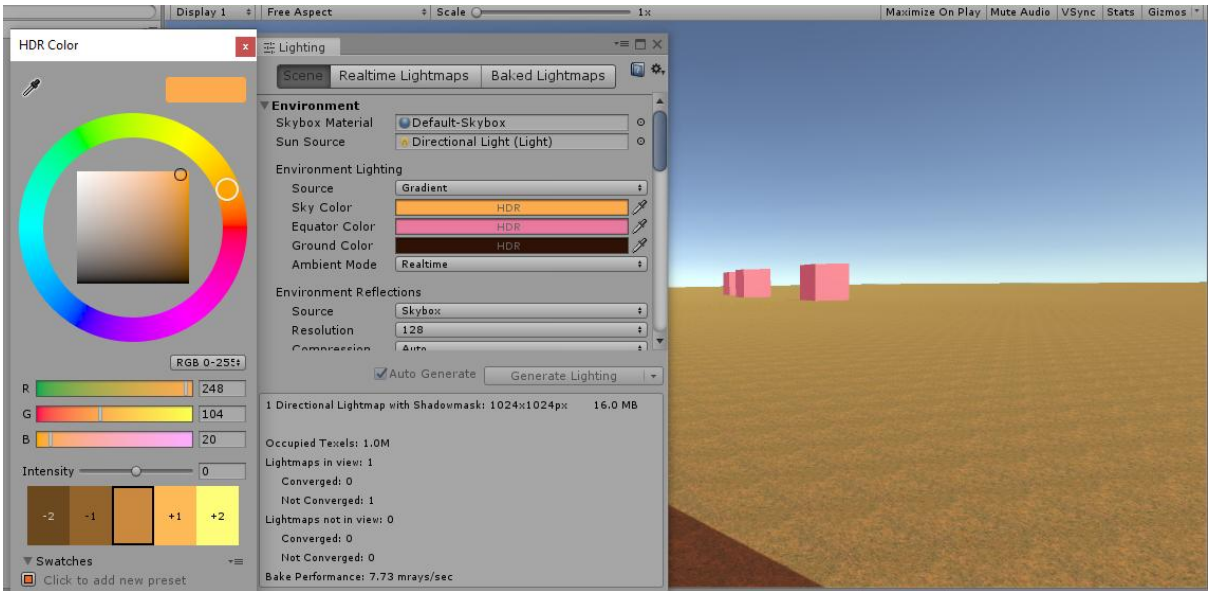


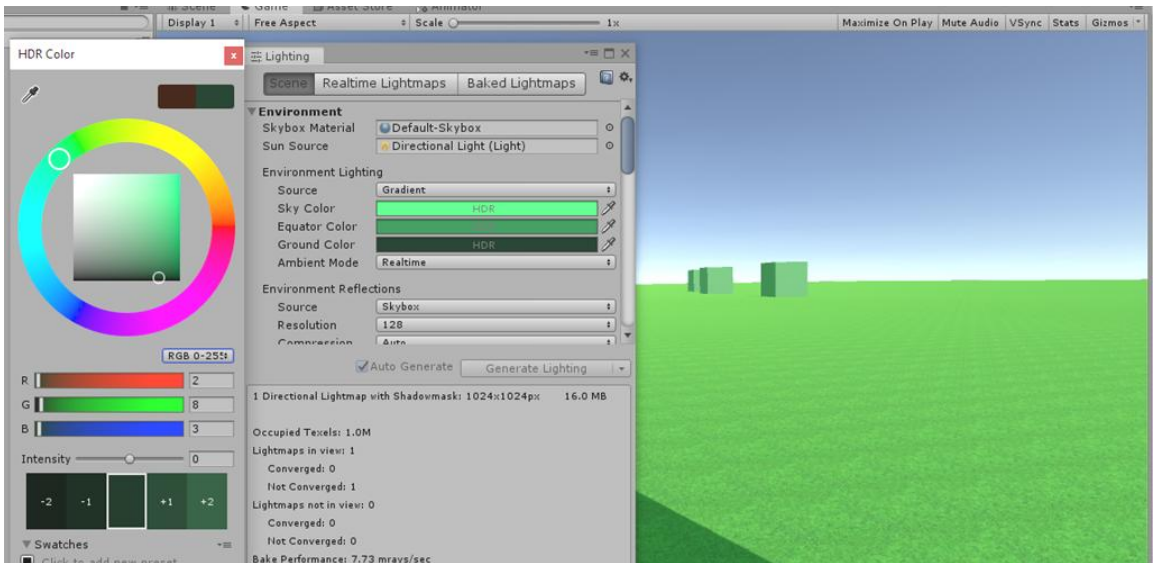
Directional Light – служит солнцем Skybox (неба). В каждом проекте существует предустановленный Skybox , он становится основным по умолчанию.



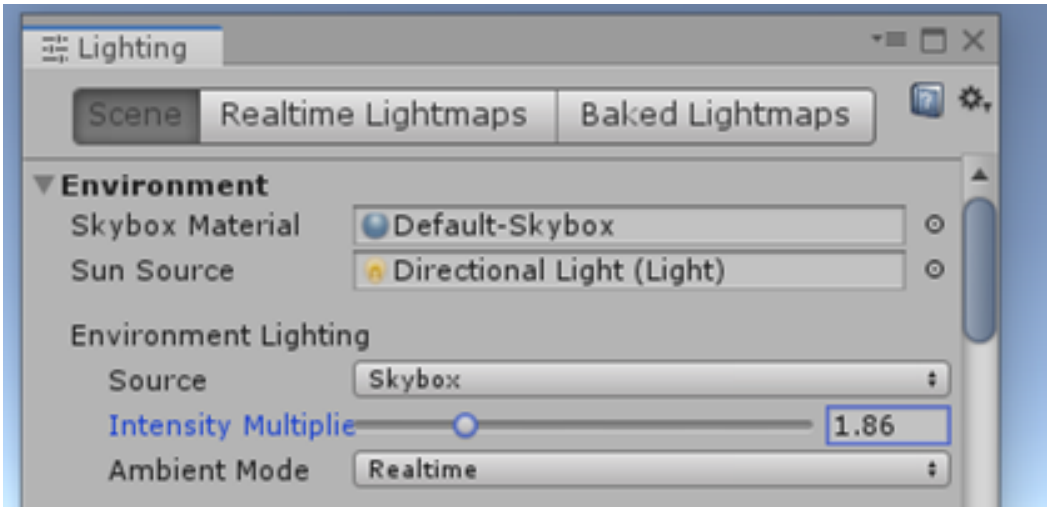
Однако можно настроить небо и освещение, используя настройки. Войдите в режим игры, и поменяйте настройки:


Lighting
Scene
Realtime Lightmaps
Baked Lightmaps
Environment
Skybox Material
Default-Skybox
Sun Source
Directional Light (Light)
Environment Lighting
Source
Skybox
Intensity Multiplie
1.27
Ambient Mode
Skybox
Gradient
Color
Environment Reflect
Source
Resolution
128
Compression
Auto
Intensity Multiplie
1
Bounces
1
Realtime Lighting
Realtime Global Illu
Mixed Lighting
Auto Generate
Generate Lighting
1 Directional Lightmap with Shadowmask: 1024x1024px 5.3 MB
Occupied Texels: 1.0M
Bake Performance: 9.24 mrays/sec
Total Bake Time: 0:03:08

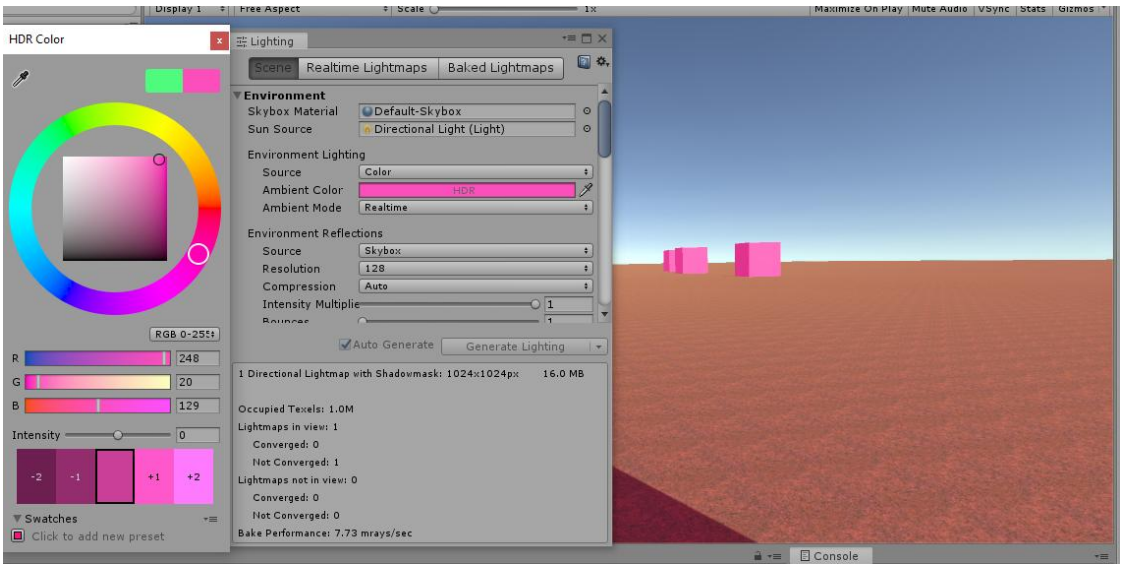

HDR Color
Lighting
Scene
Realtime Lightmaps
Baked Lightmaps
Environment
Skybox Material
Default-Skybox
Sun Source
Directional Light (Light)
Environment Lighting
Source
Gradient
Sky Color
Equator Color
Ground Color
Ambient Mode
Realtime
Environment Reflections
Source
Skybox
Resolution
128
Auto Generate
Generate Lighting
1 Directional Lightmap with Shadowmask: 1024x1024px 16.0 MB
Occupied Texels: 1.0M
Lightmaps in view: 1
Converged: 0
Not Converged: 1
Lightmaps not in view: 0
Converged: 0
Not Converged: 0
Bake Performance: 7.73 mrays/sec
RGB 0-255
R 248
G 104
B 20
Intensity 0
Swatches
Click to add new preset



В зависимости от выбранных цветов меняется ваш ландшафт. Вернем настройку Skybox и попробуем передвинуть ползунок:

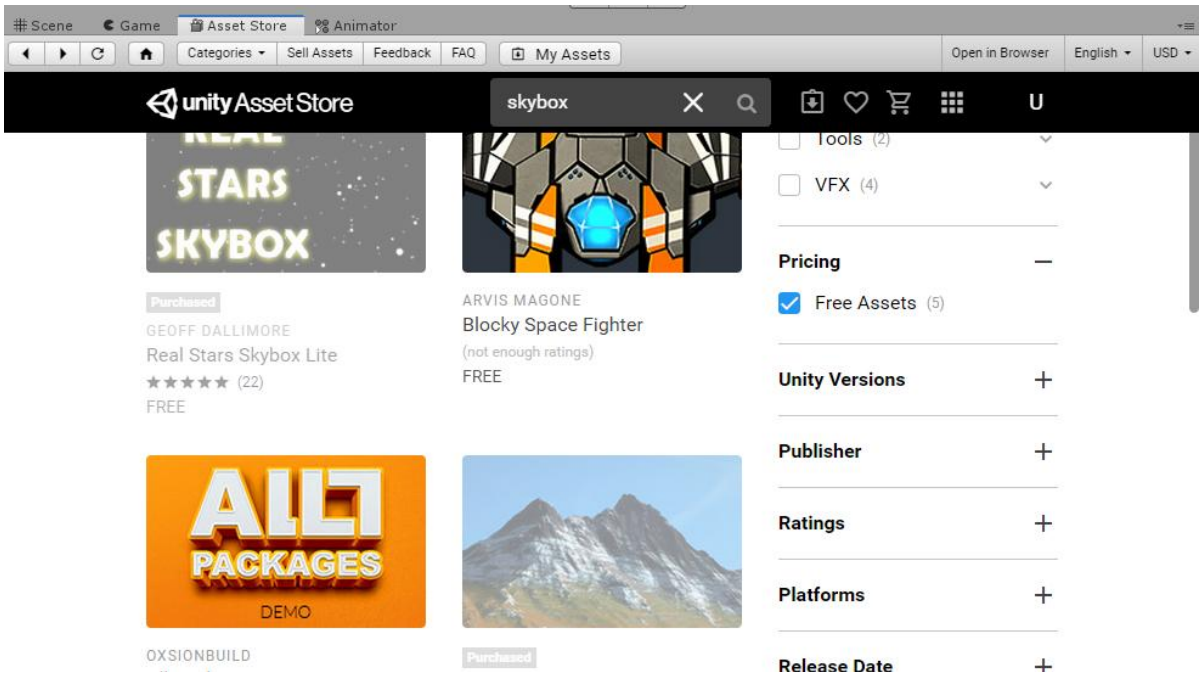


Измените Source на Color и посмотрите, что изменится.



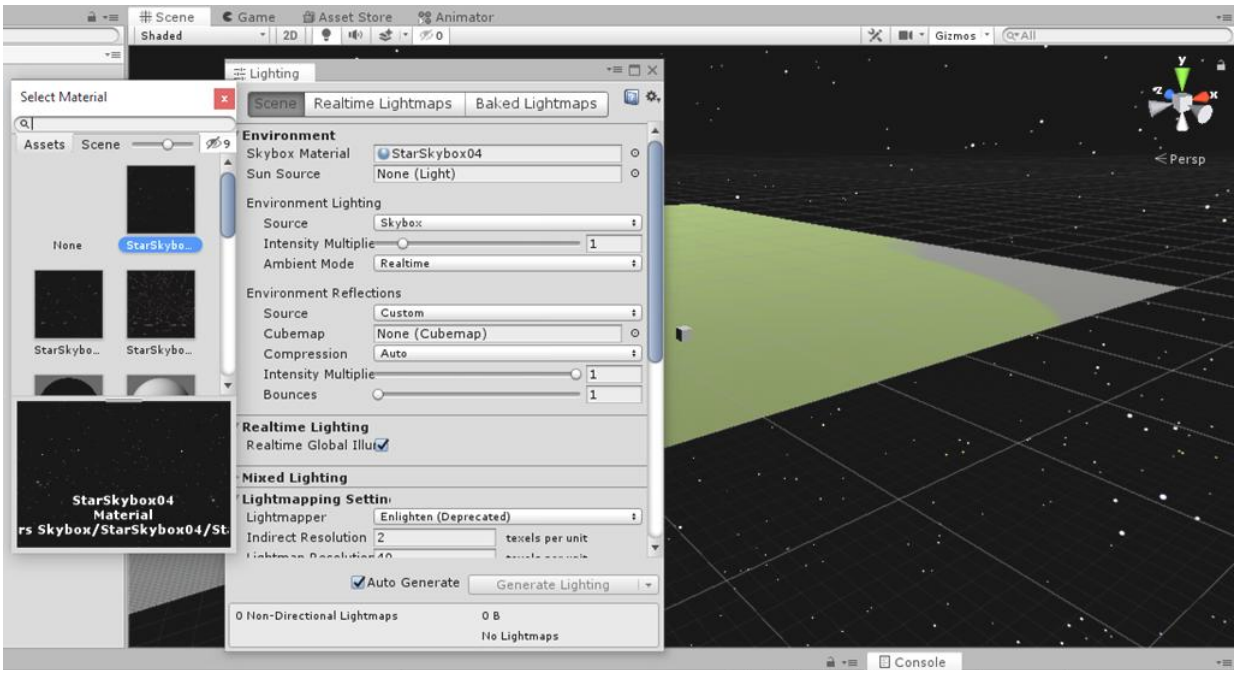
Теперь давайте настроим само небо. Так как большинство космических объектов не имеет атмосферы, то с них видно космос.

Заходим в asset sore и вводим в строку поиска skybox , измените настройку Pricing на Free Assets и скачайте выбранный asset.



После того как ассет скачается, импортируйте его.

После этого снова откройте настройки освещения, там, где Skybox Material, нажмите на кружочек справа, откроется дополнительное окошко с текстурами. Выберите подходящую.

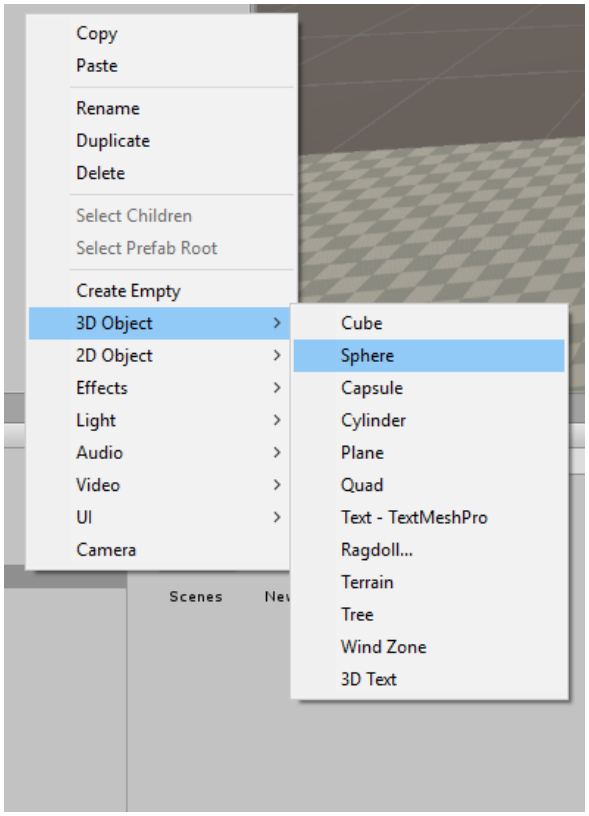


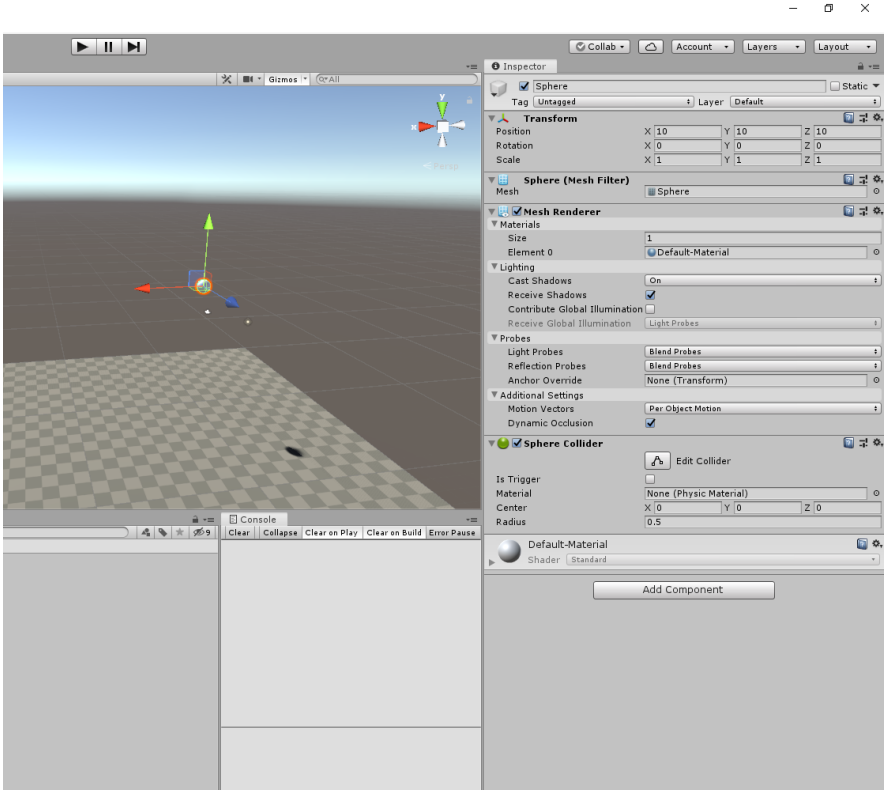
Затем в Sun Source так же нажмите на кружок, в открывшемся окошке выберете вкладку Scene и там Directional Light.

Запустите игру, посмотрите, что получилось.

Физика 3D объектов

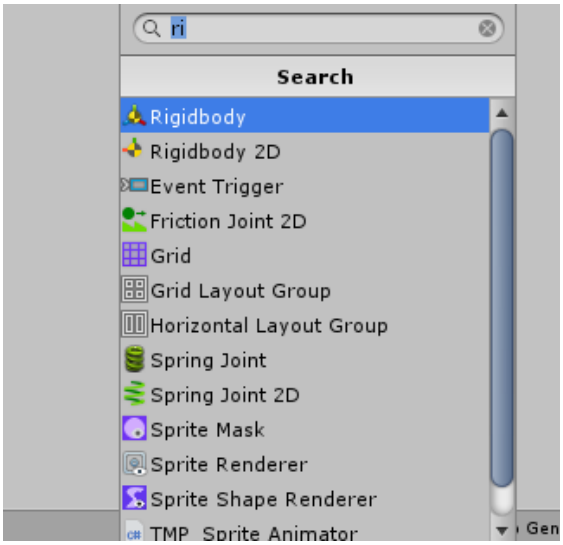
Создадим сферу с координатами (Position) 10,10,10.





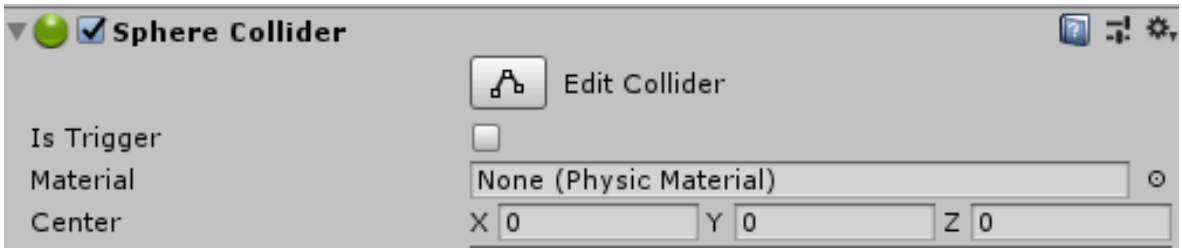
И куб с координатами 15,10,15.

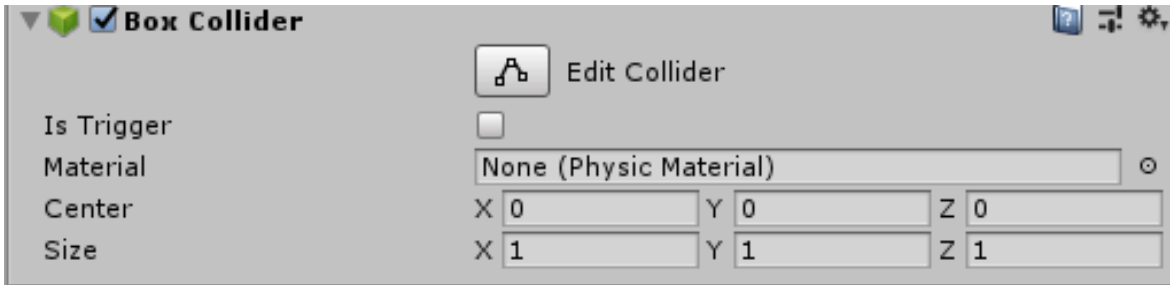
Для сферы добавим компонент Rigidbody (твердое тело), для этого в окне инспектор нажмите кнопку Add Component (внизу окна).



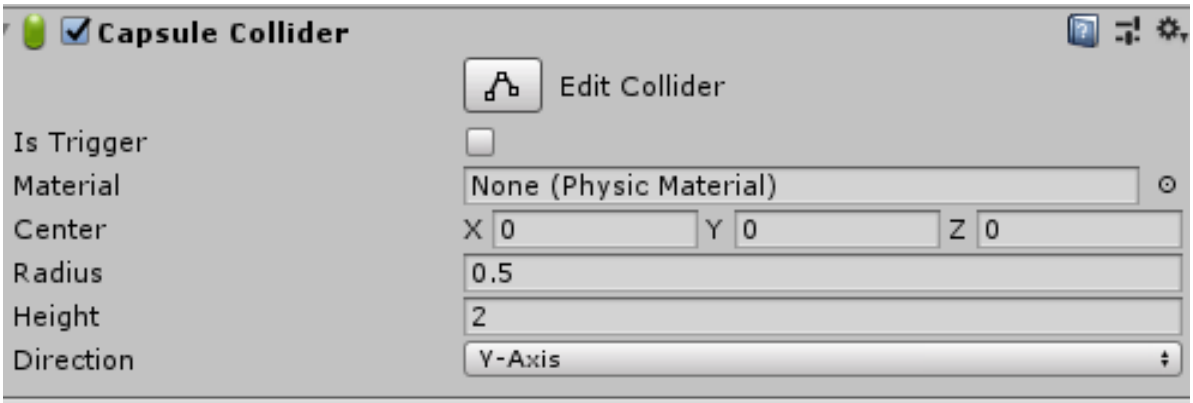
Если сейчас запустить игру, то сфера упадет, а куб нет. Все потому что сфера, после добавления компонента приобрела массу, и на нее стала действовать сила гравитации.

Теперь давайте посмотрим на еще один компонент у сферы и куба:





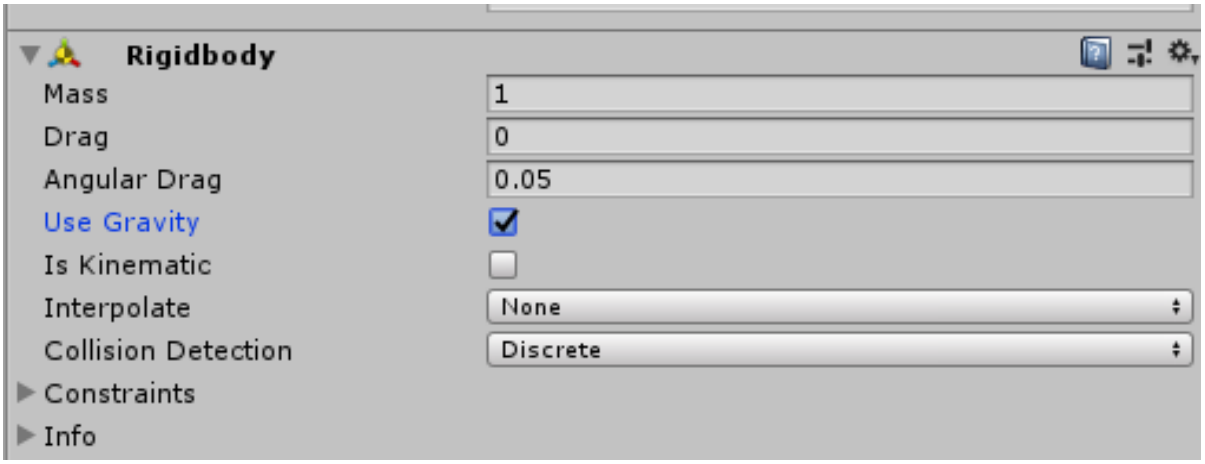
Если создать еще один объект Capsul с координатами 20,10,20, то в окне инспектор увидим еще один вариант коллайдера:



Коллайдер делает наш объект «непроницаемым», то есть при столкновении объектов они не проходят сквозь друг друга.

Измените Position у сферы на 15,20,15 и запустите игру, что произошло?

Для того чтобы ваш объект был включен в физический мир у него должны быть подключены оба компонента: коллайдер и твердое тело.



Mass – масса объекта. При столкновении объекты с большей массой выталкивают объекты с меньшей.

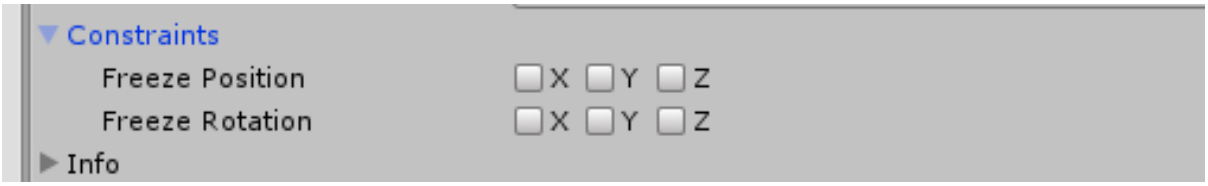
Drag – добавляет сопротивление объекту, то как быстро он перемещается.

Давайте изменим массу сфере с 1 до 10 и запустим игру. А если изменить Drag на 5, что изменилось?

Angular drag – это угловое сопротивление, то есть сопротивление при вращении.

Если снять флажок с Use Gravity ваша сфера зависнет в воздухе, потому что теперь на нее не действует сила тяжести.

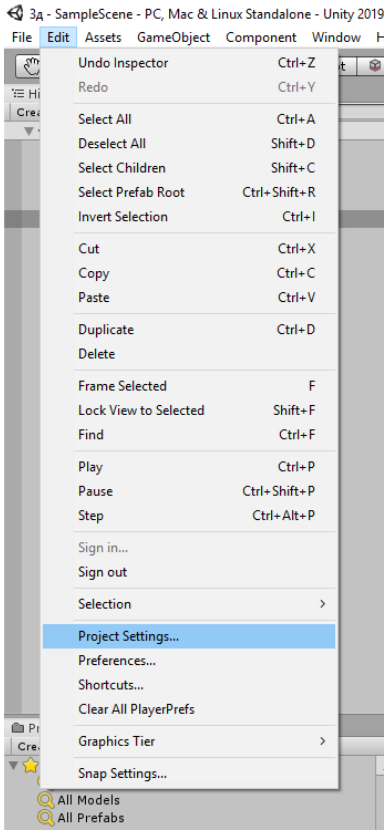
Is Kinematic отключает воздействие физики на объект (это позволяет оставаться ему твердым телом, но при этом экономит ресурсы).

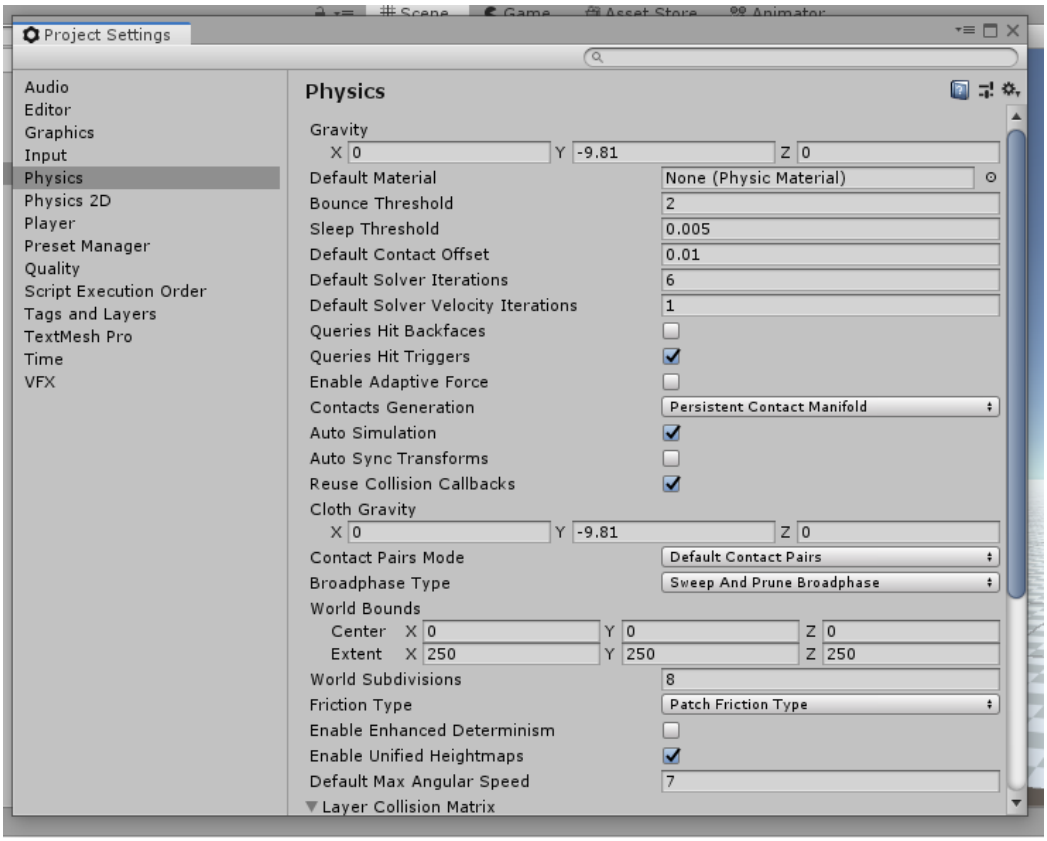


Флажок в одном из окошек Constraints не дает перемещаться или вращаться в выбранном направлении.

Сместите сферу на координаты 15.1, 20,15, добавьте кубу компонент Rigidbody и отключите гравитацию, запустите игру, что получится?

Настроить гравитацию можно в разделе edit.

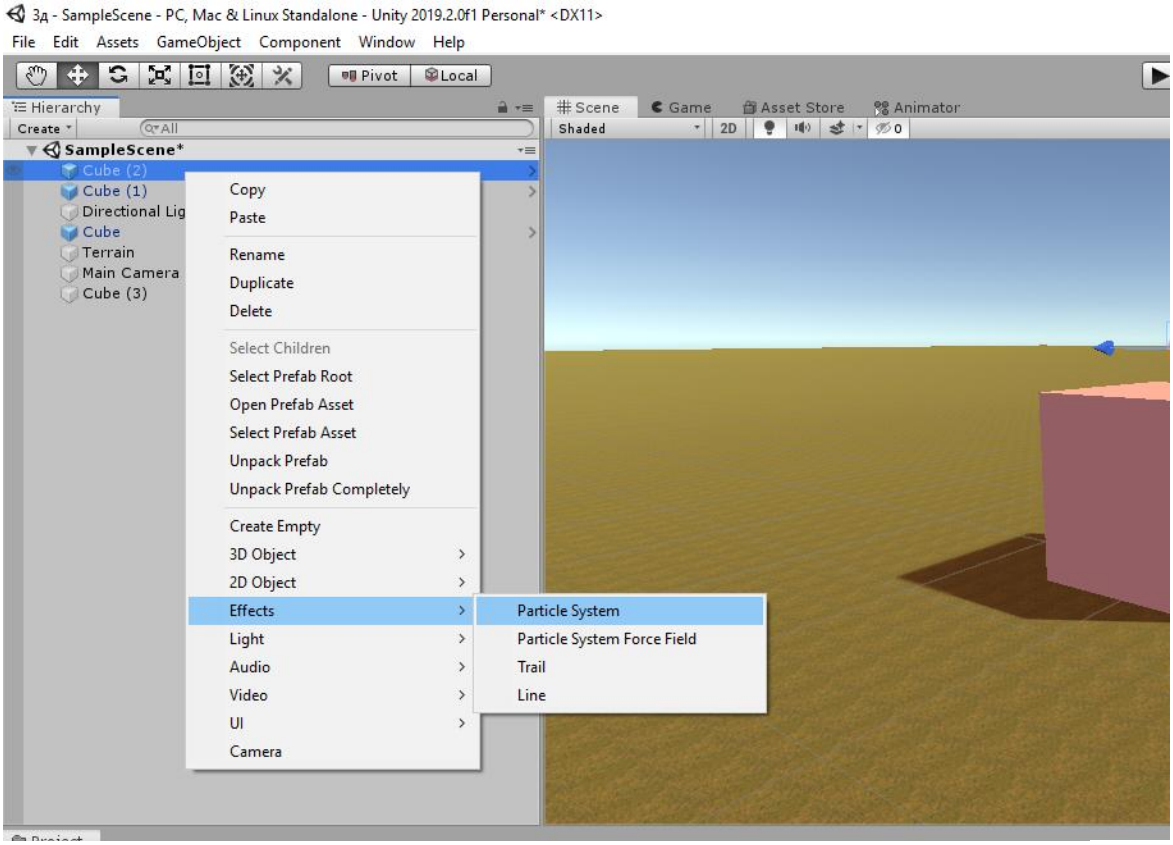




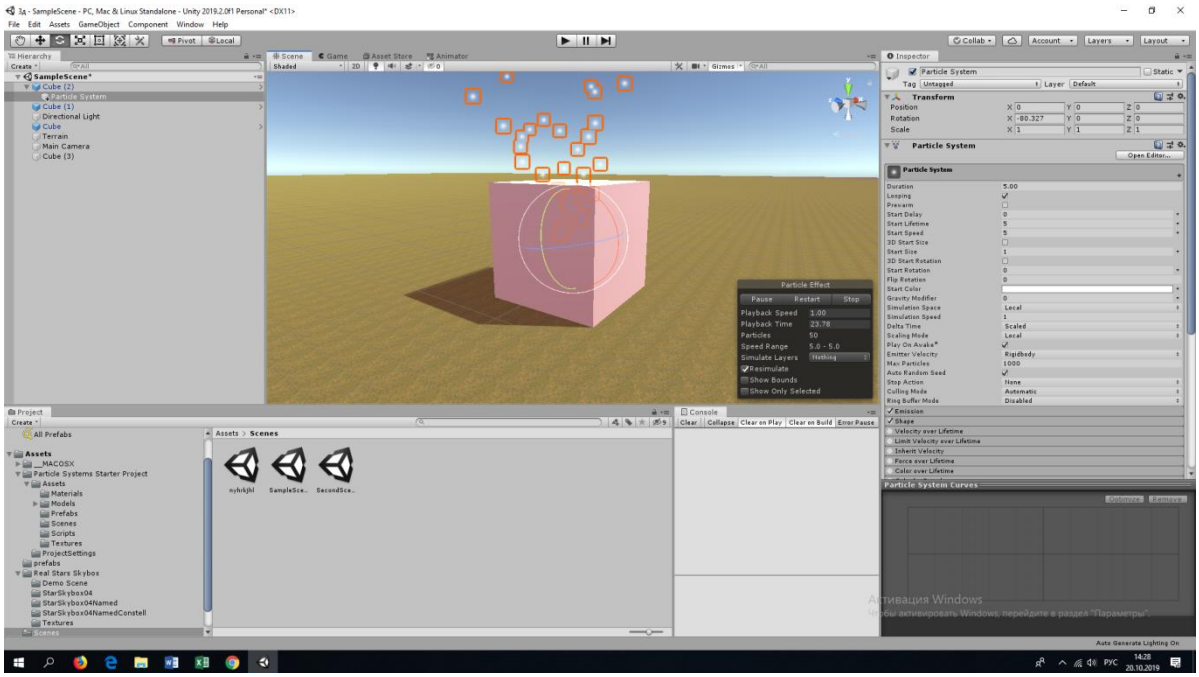
Настройте гравитацию таким образом, чтоб она совпадала с гравитацией на выбранной вами планете.

Система частиц (*Particle System*)

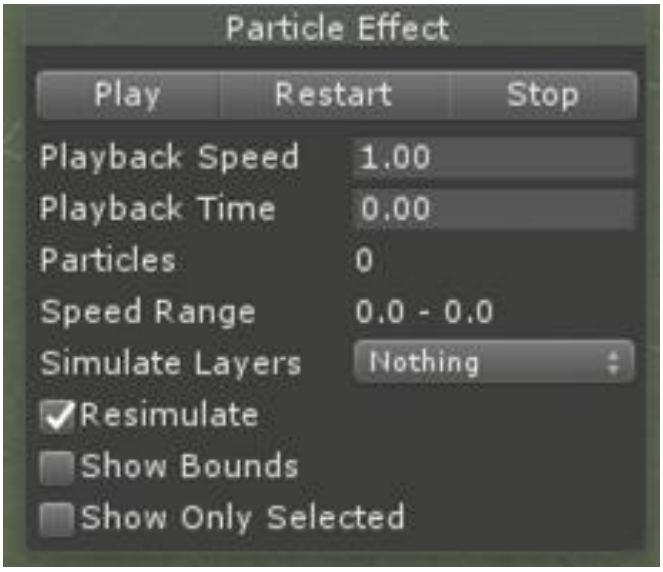
✓ Откройте проект упражнение, выберите один из не используемых объектов, или создайте новый.



✓ Выделите систему частиц, и с помощью инструмента  разверните в нужную сторону.

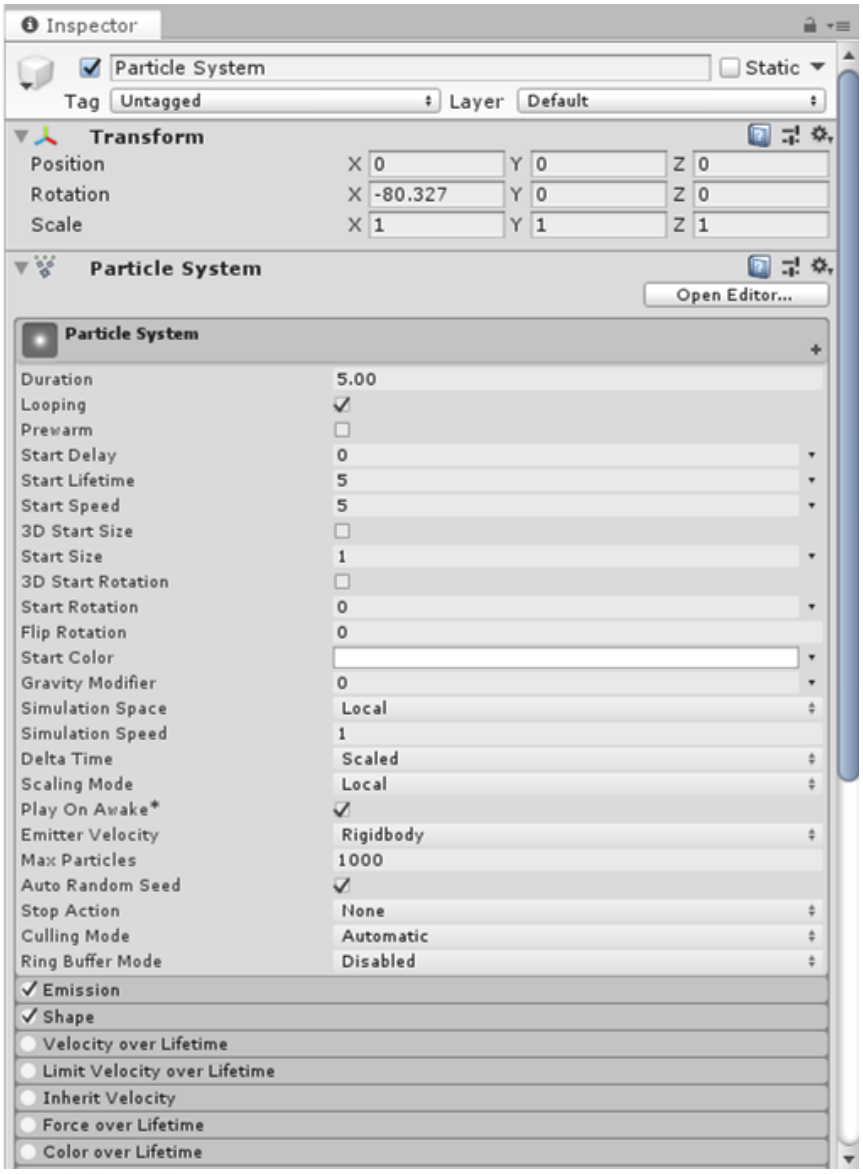


При выборе GameObject с присоединённой к нему системой частиц вы заметите в правом нижнем углу окна сцены чёрное диалоговое окно. Это диалоговое окно позволяет симулировать или останавливать систему частиц. Нажатие на кнопку *Simulate* активирует систему частиц и заменяет её на кнопку “Pause”. Чтобы остановить симуляцию, нужно нажать на кнопку “Stop”.



Это диалоговое окно полезно при создании систем частиц, выполняемых в течение ограниченного периода времени, например, для взрывов.

Взгляните на Inspector. Вы заметите, что добавленный нами компонент системы частиц имеет несколько подразделов:



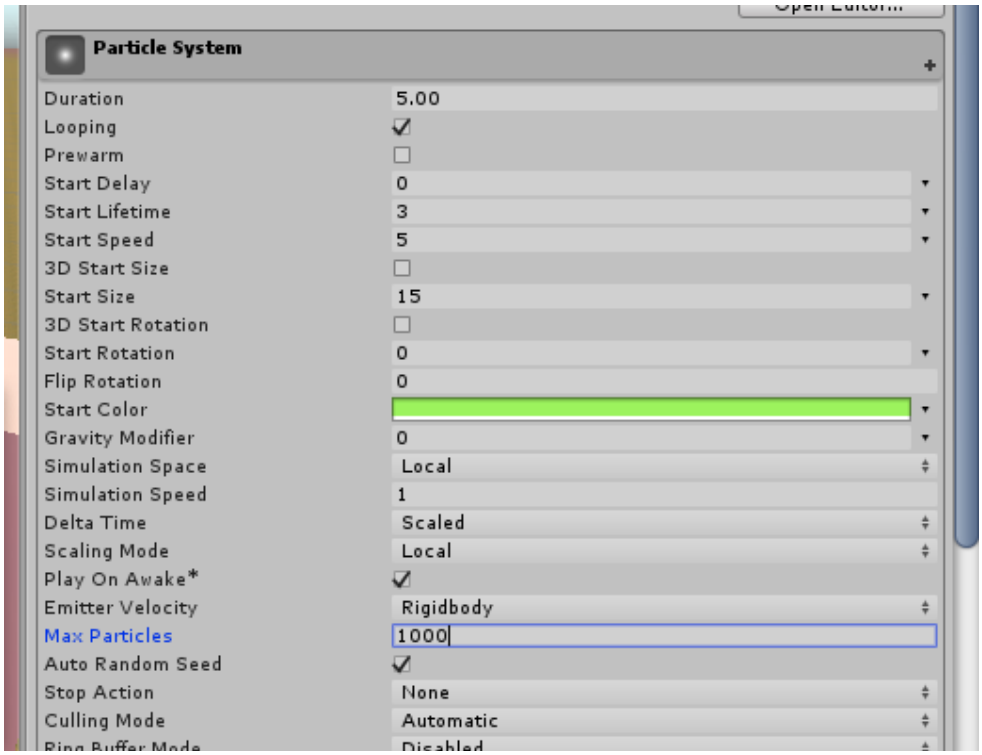
Каждый из этих подразделов называется модулем (*Module*). В этих модулях содержатся параметры системы частиц. Развёрнутый по умолчанию модуль называется модулем *Main.*

Модуль Main — фундамент любой системы частиц в Unity. Здесь находятся самые основные параметры частиц:

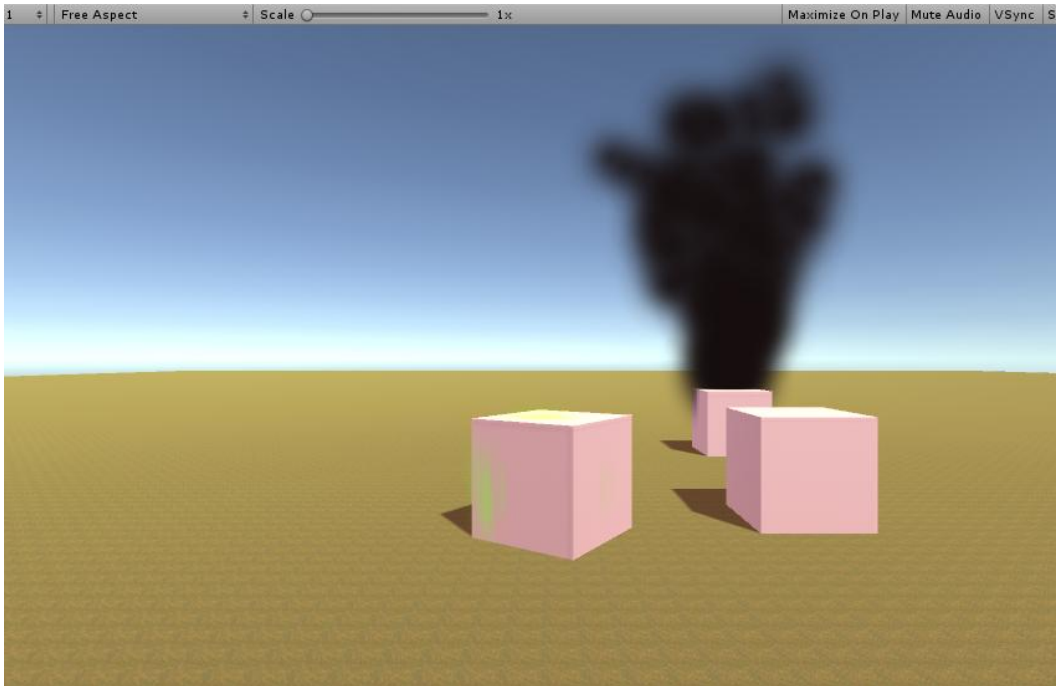
- *Duration*: время выполнения системы частиц в секундах. Оставим здесь значение по умолчанию*5.00.*
- *Looping*: повторное испускание частиц до остановки системы частиц. Цикл перезапускается после достижения времени *Duration*. Огонь должен гореть постоянно, так что оставим этот параметр включённым.
- *Prewarm*: используется, только когда включен *Looping*. Система частиц будет вести себя так, как будто выполнила полный цикл при запуске.
- *Start Delay*: задержка в секундах перед тем, как система частиц начинает испускать частицы. Оставим здесь значение по умолчанию *0.*
- *Start Lifetime*: исходное время жизни частиц в секундах. После завершения времени частица уничтожается.

- *Start Speed*: исходная скорость частиц. Чем больше скорость частиц, тем сильнее они будут распространяться.
- *Start Size*: исходный размер частиц.
- *Start Rotation*: исходный угол поворота частиц.
- *Start Color*: исходный цвет частиц.
- *Gravity Modifier*: изменяет масштаб значения *гравитации*, заданного в окне Unity *Physics Manager*. Если он равен 0, то гравитация будет отключена.
- *Simulation Space*: перемещает частицы в *Local Space* вместе с системой частиц. При выборе *World Space* частицы после испускания перемещаются свободно.
- *Play On Awake*: начинает испускать частицы сразу же после включения. Если этот параметр отключен, то вам придётся вручную запускать систему частиц через скрипт или систему анимаций. Оставьте его включенным, потому что мы хотим, чтобы пламя начинало гореть при запуске сцены.
- *Max Particles*: максимальное количество частиц, которые могут быть живы в системе частиц одновременно. Если попробовать испускать больше частиц, чем заданное здесь значение, то они не будут испускаться вообще. Этот параметр в основном нужен из соображений производительности, и значения по умолчанию в *1000* частиц в нашем случае более чем достаточно.

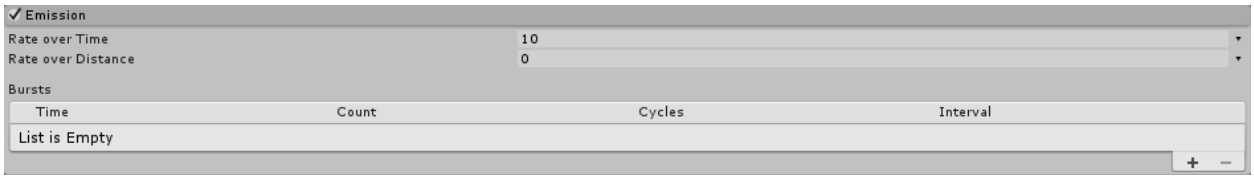
✓ Измените настройки следующим образом:



✓ Измените настройки, чтобы стало похоже на дым:



Модуль *Emission* управляет количеством и временем испускаемых частиц в системе и позволяет создавать любые эффекты, от постоянного потока до резкого взрыва частиц.



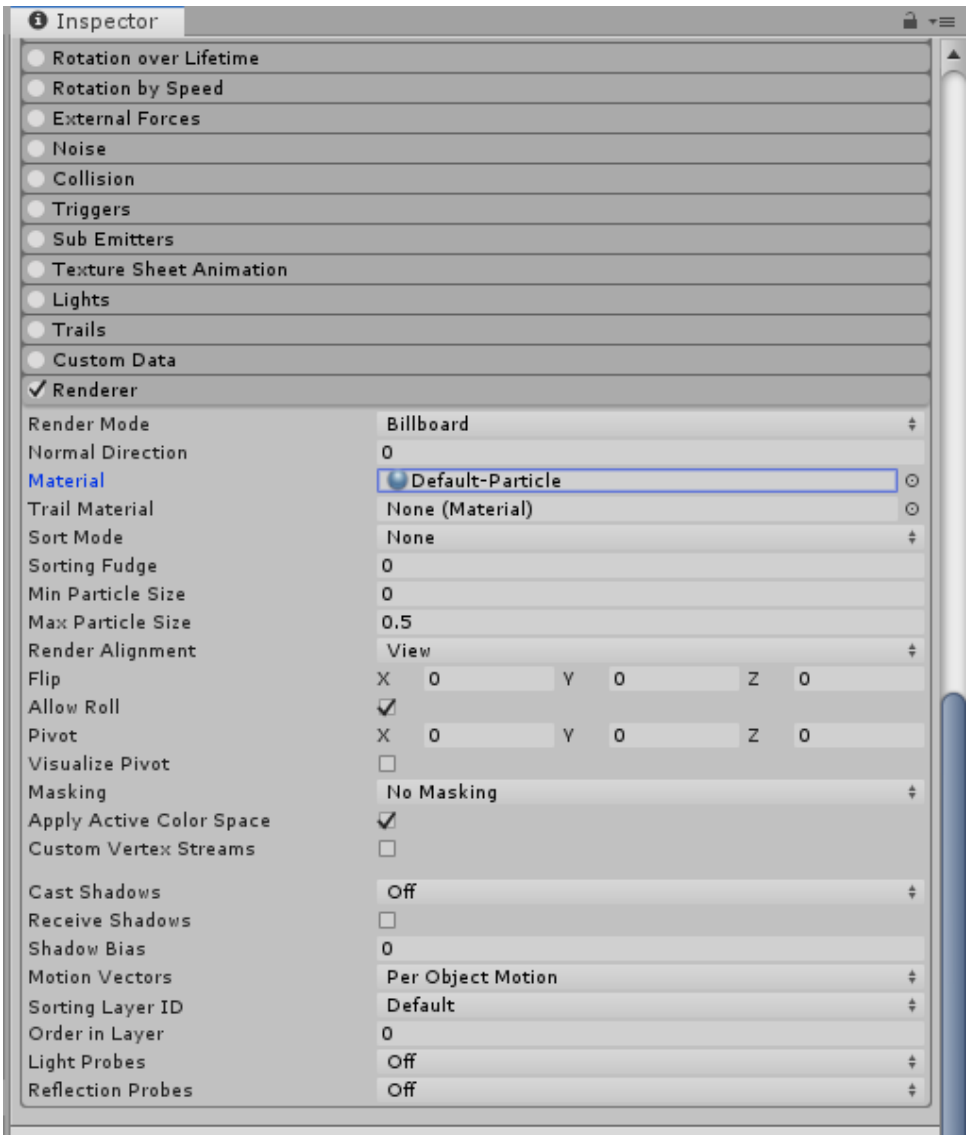
✓ *Rate over Time* представляет количество частиц, испускаемых за секунду. Задайте для *Rate over Time* значение *15*.

У всех частиц есть материал частиц и текстура, задающая их внешний вид. С текстурой по умолчанию мы можем сделать только это. Заменив текстуру, мы можем создавать эффекты наподобие волшебных звёздочек, дыма и, разумеется, огня.

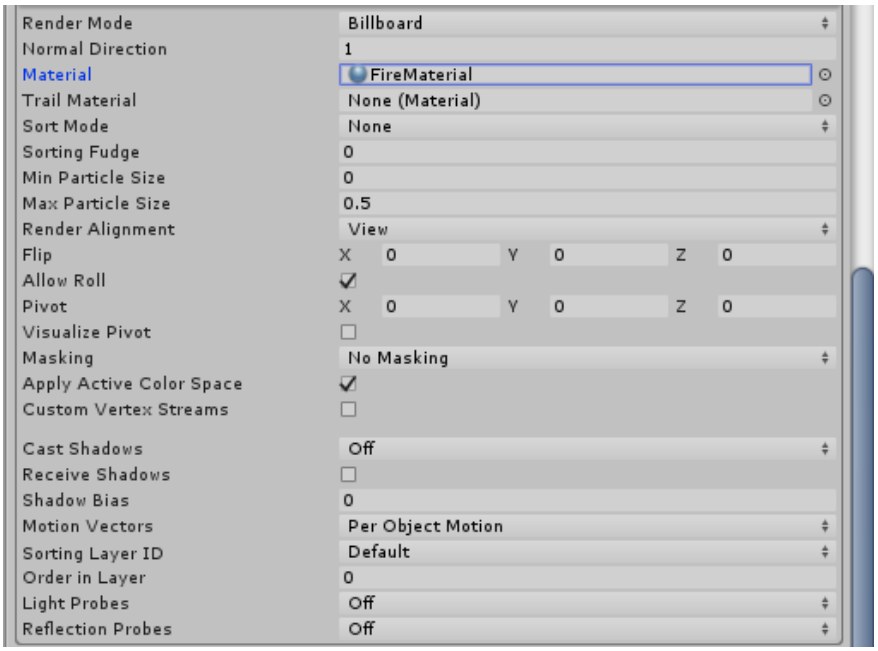
Заменить текстуру частиц достаточно просто. До этого момента частицы отрисовывались на экране с помощью материала *Default-Particle*, который является материалом частиц с круговой градиентной текстурой.

✓ Загрузите текстуру FireMaterial

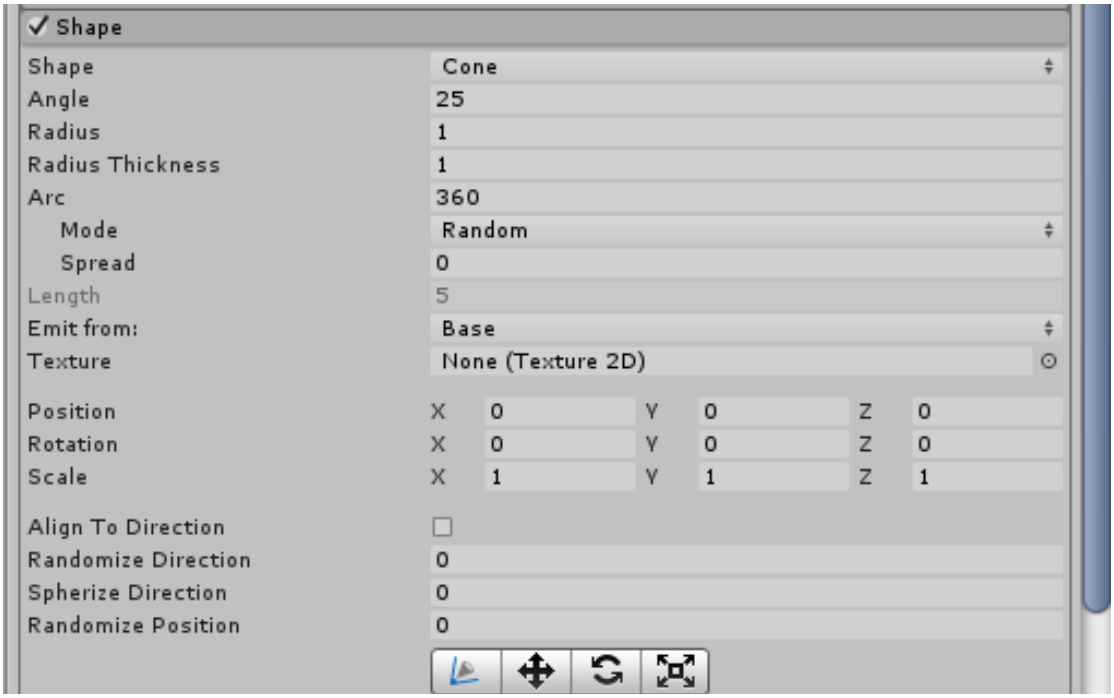
✓ Найдите в инспекторе компонент системы частиц и откройте модуль *Renderer* системы частиц.



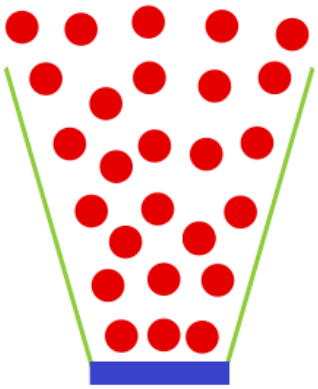
✓ Перетащите FireMaterial из папки asset в окошко Material, как на картинке:



Модуль *Shape,* как можно понять из названия, управляет формой и поведением частиц в этой форме. Можно выбрать одну из нескольких различных форм; каждая имеет свои собственные параметры. Это позволяет создавать частицы в параллелепипеде, в сфере или даже в собственном меше.



В качестве формы (*shape*) системы частиц выбран конус (*cone*), то есть частицы испускаются из основания *base* и движутся наружу под углом (*angle*):



В показанном выше примере основание окрашено *синим*, угол *зелёным*, а частицы — *красным*.

- ✓ Меняя *Angle*, можно изменять размер конуса, делая его шире или уже. Измените угол на 7.
- ✓ Меняя *Radius*, мы можем менять размер основания. Чем больше значение, тем больше будут рассеиваться частицы при испускании.

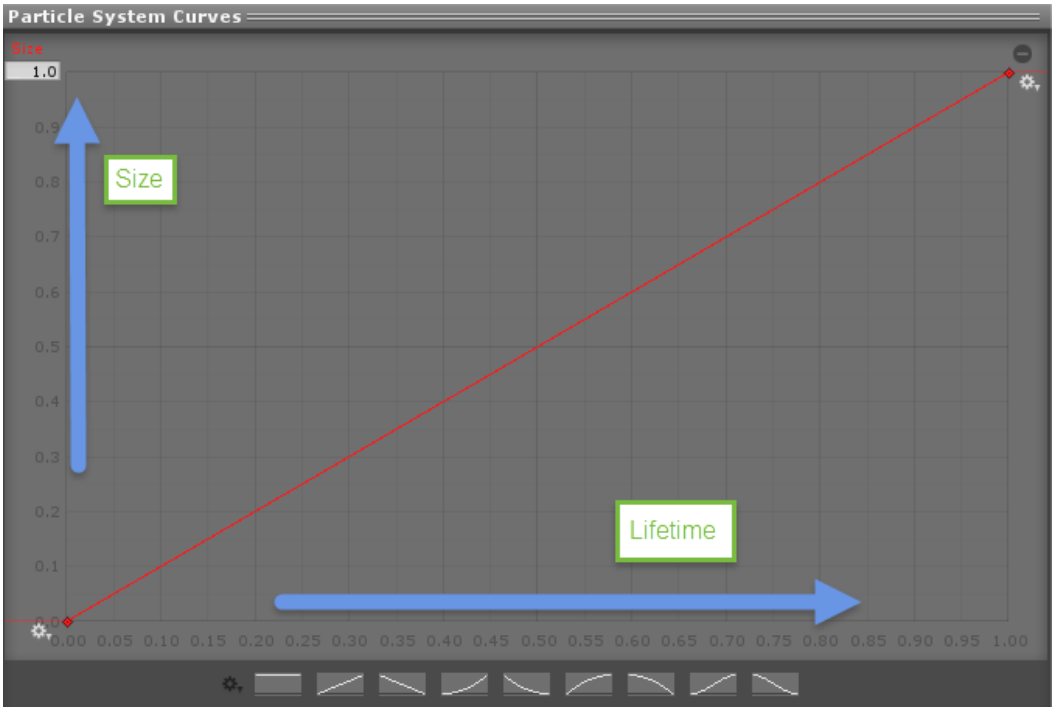
Size over Lifetime можно создавать частицы, увеличивающиеся или уменьшающиеся с течением их жизни, или даже пульсирующие, как светлячки в лесу.

Найдите в списке модулей системы частиц раздел *Size over Lifetime*. По умолчанию от выключен, поэтому поставьте флажок рядом с названием модуля:

Разверните модуль Size over Lifetime, нажав на его название. Так вы откроете тёмно-серый фон с плоской кривой (*Curve*) в верхней части:

✓

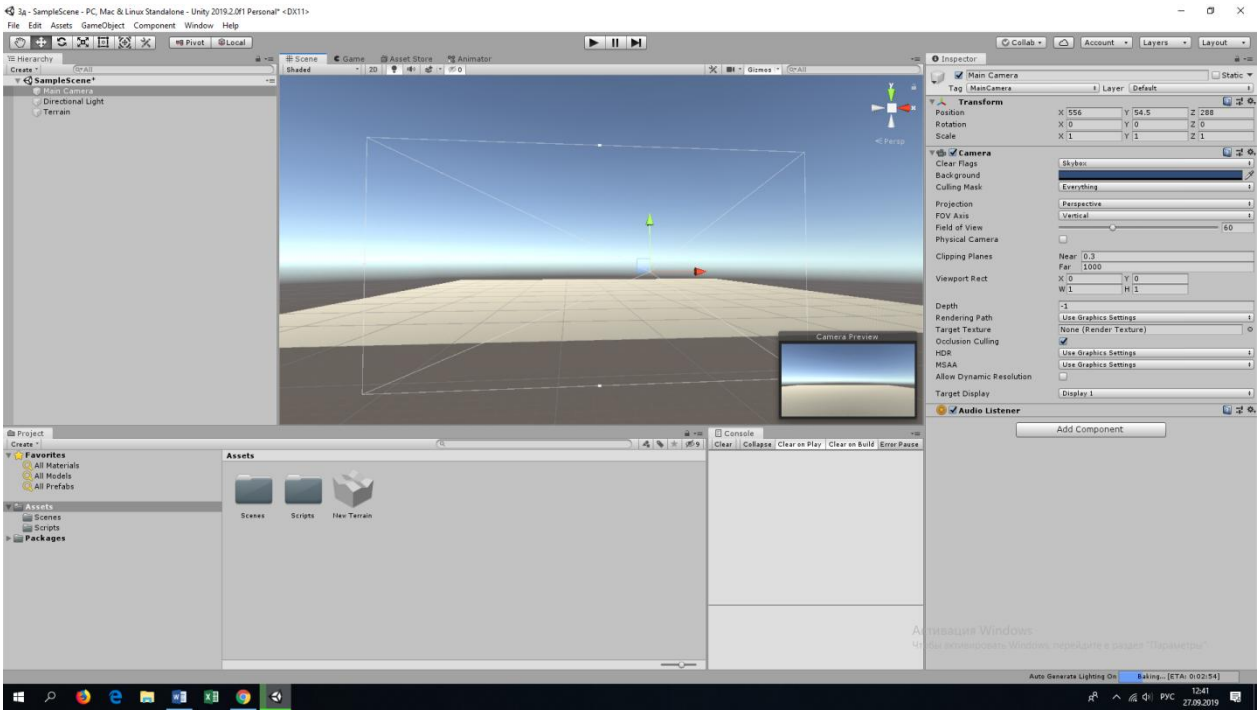
Нажмите на *тёмно-серый фон*, чтобы открыть редактор кривых в нижней части инспектора. Горизонтальная ось отображает время жизни частицы, а вертикальная ось — её размер:



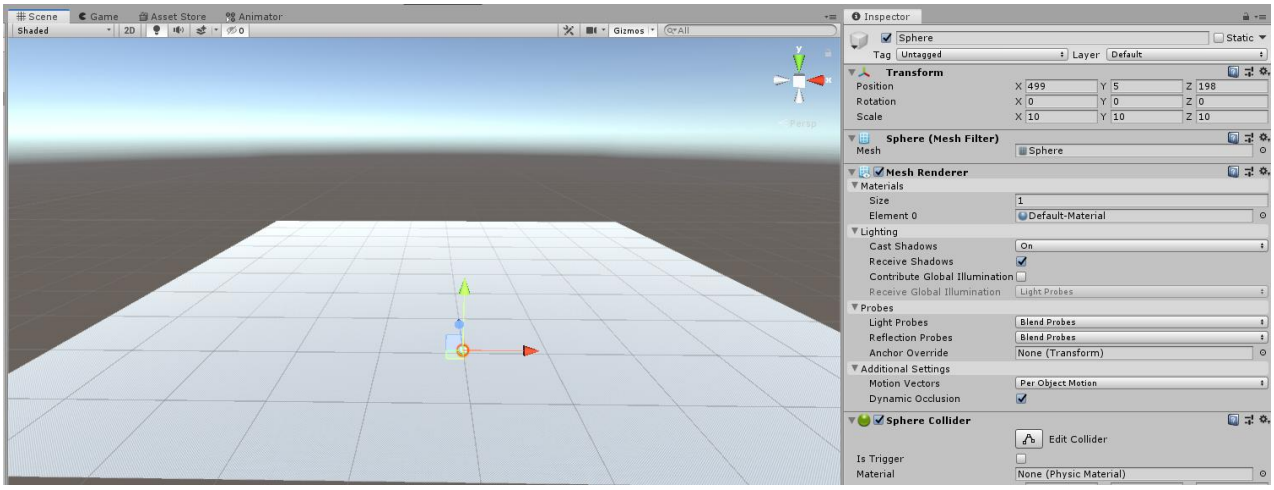
Для редактирования кривой можно перемещать ключевые точки с обоих сторон красной линии; добавлять новые ключи можно двойным нажатием на любое место кривой. Для удаления ключевых точек нажмите правой клавишей мыши на точку и выберите *Delete Key*.

Управление

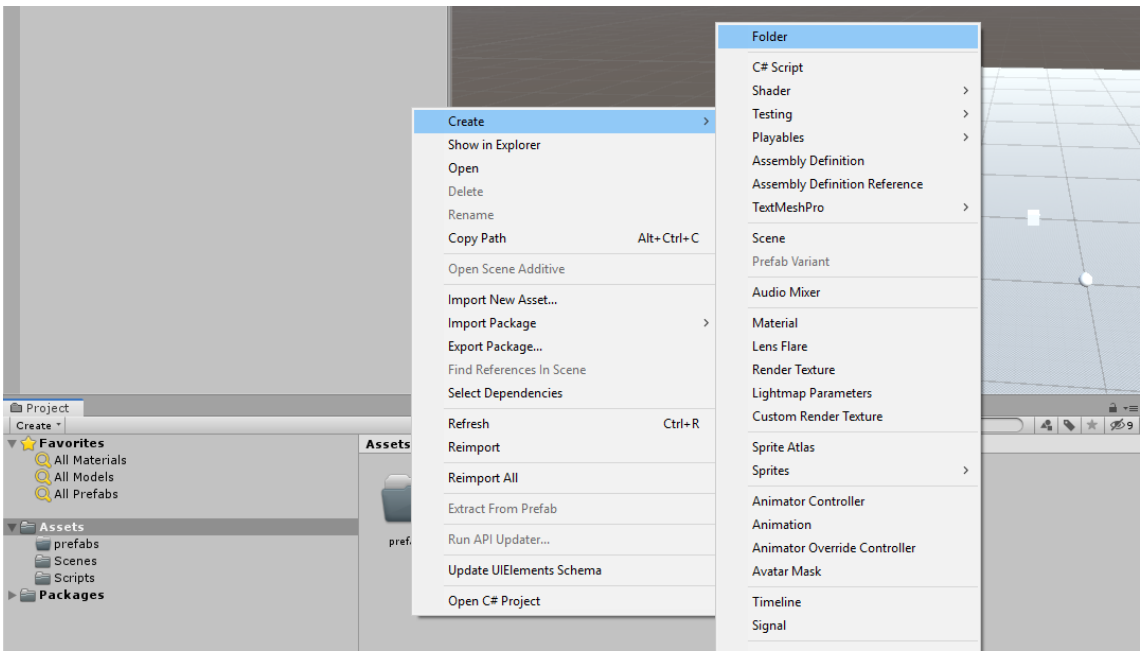
Откроем проект «упражнение» (если вы не создали его на прошлом занятии, то создайте). Поместим террайн, и настроим камеру, переместив так, чтобы видно было большую часть (это можно сделать выбрав main camera в окне иерархии, и нажав значок с 4мя разнонаправленными стрелочками.



Создадим сферу, изменим ее размер по всем направлениям на 10, разместим сферу так, чтобы ее было видно с камеры. При необходимости настройте камеру, освещение и раскрасьте террайн.



В пустом пространстве поля asset нажмите правой клавишей мыши и создайте новую папку, назовите ее prefabs:



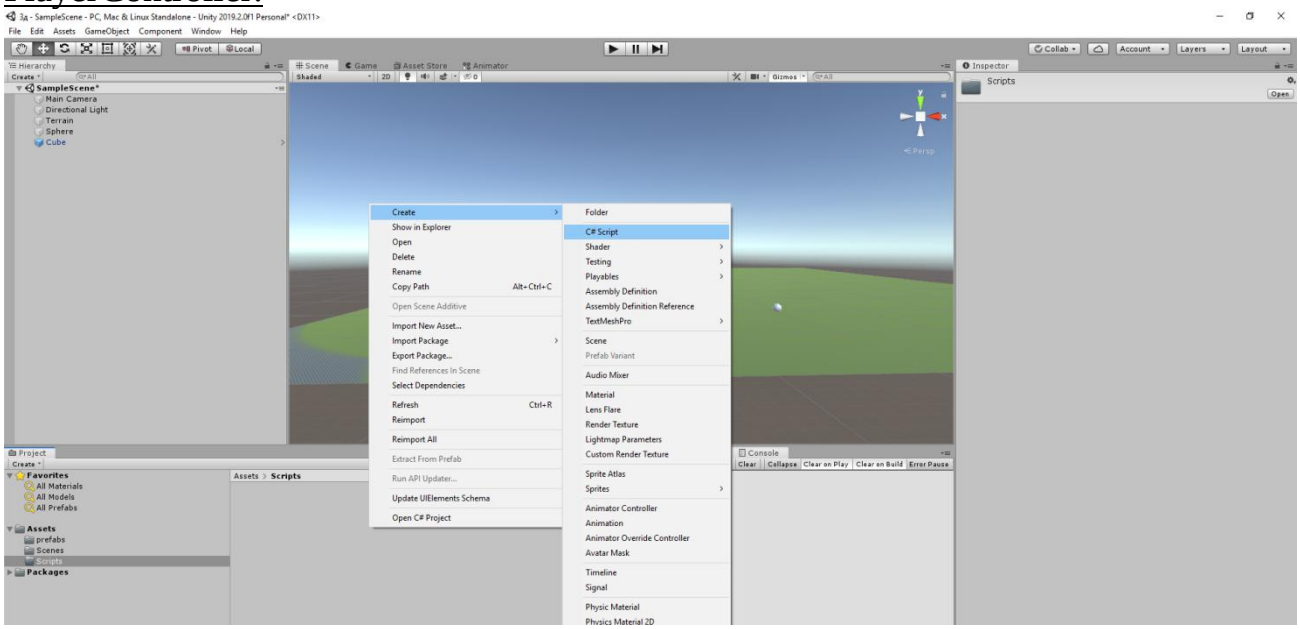
Перетащим наш куб из окна иерархии в нашу папку. Теперь у нас есть первый префаб.

Из официальной документации:

Префаб — это один из типов ресурсов, предназначений для многократного использования и хранящийся в Project View. Префаб может быть вставлен в любое количество сцен и многократно в одну сцену. Когда префаб добавляется в сцену, создаётся его экземпляр. Все экземпляры являются ссылками на оригинальный префаб и фактически его клонами. Независимо от того, как много экземпляров в проекте, при изменении префаба изменяются соответственно и все его экземпляры.

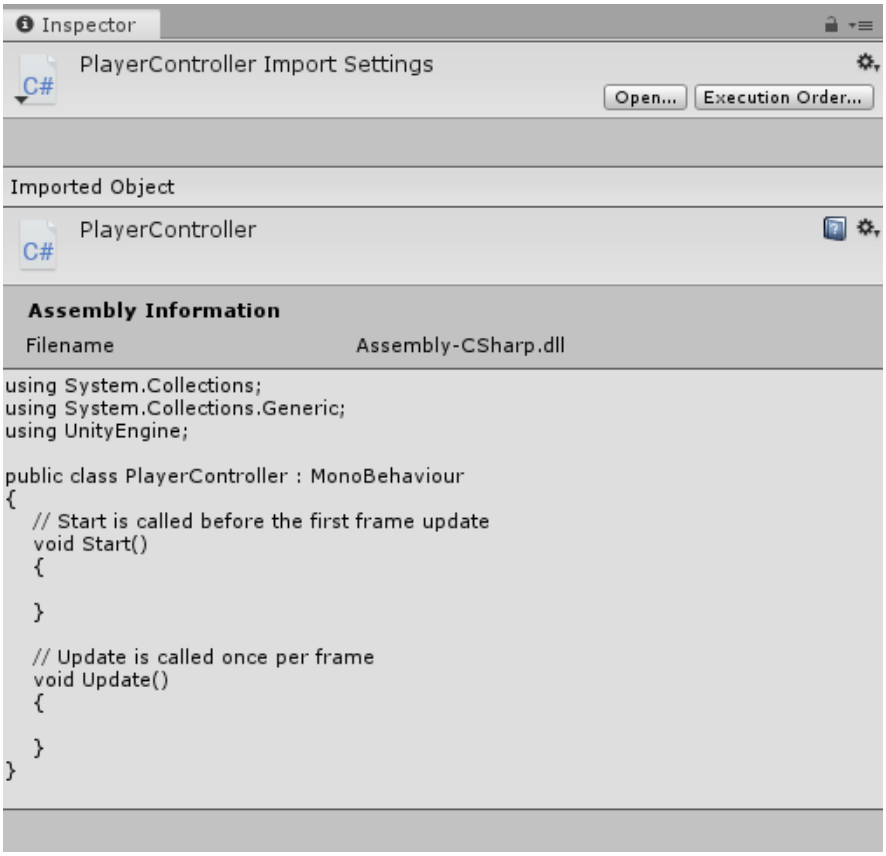
Создайте новую папку, назовите ее scripts

Откройте папку и создайте новый скрипт с названием PlayerController.



Перетащите наш скрипт на сферу, для этого выделите сферу, в окне ассет мышкой захватите скрипт и перетащите его в пустое место окна инспектор

Если нажать на созданный скрипт, в окне инспектор вы увидите следующее



Первые три строчки дают доступ к различным элементам C# в Юнити.

public class – обозначает, что мы создаем новый класс объектов, доступный из любого места игры

void Start(){} – метод, действия внутри которой выполняются один раз при старте

void Update(){} - метод, действия внутри которой выполняются при каждой смене кадра.

Обратите внимание, в C# важен регистр:
Start и start – это разные слова

Откройте скрипт (для этого 2 раза щелкните на нем)
Допишите одну строчку в void Update:

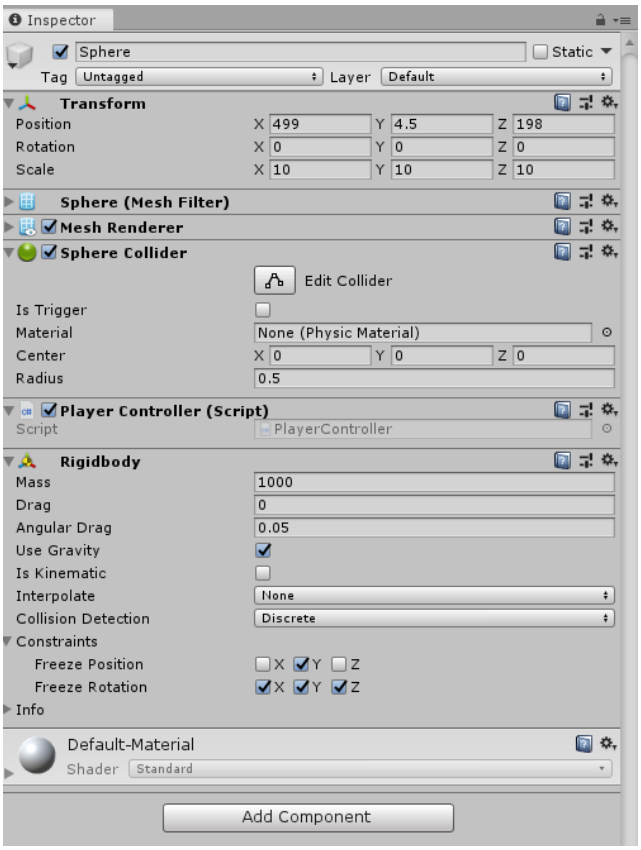
```
void Update()
  {
    transform.Translate(Vector3.forward);
  }
```

Запустите игру, что происходит?
Vector3 – переменная, которая хранит в себе значение x,y,z,
transform.Translate – отвечает за перемещение объекта
Изменим на

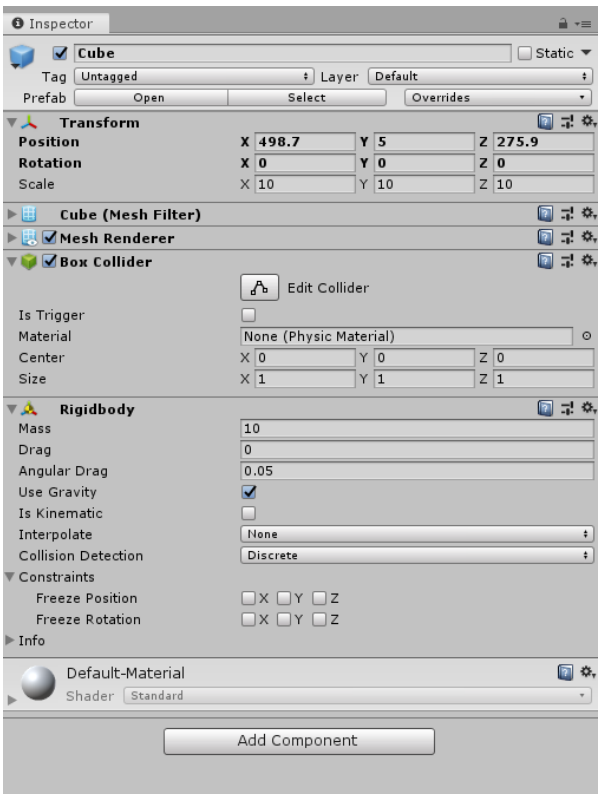
```
transform.Translate(Vector3.forward * Time.deltaTime*20);
```

Расположите куб на пути движения шара, запустите игру. Что происходит?

Настроим **RigidBody** и у сферы:



и у куба (в папке префабс):



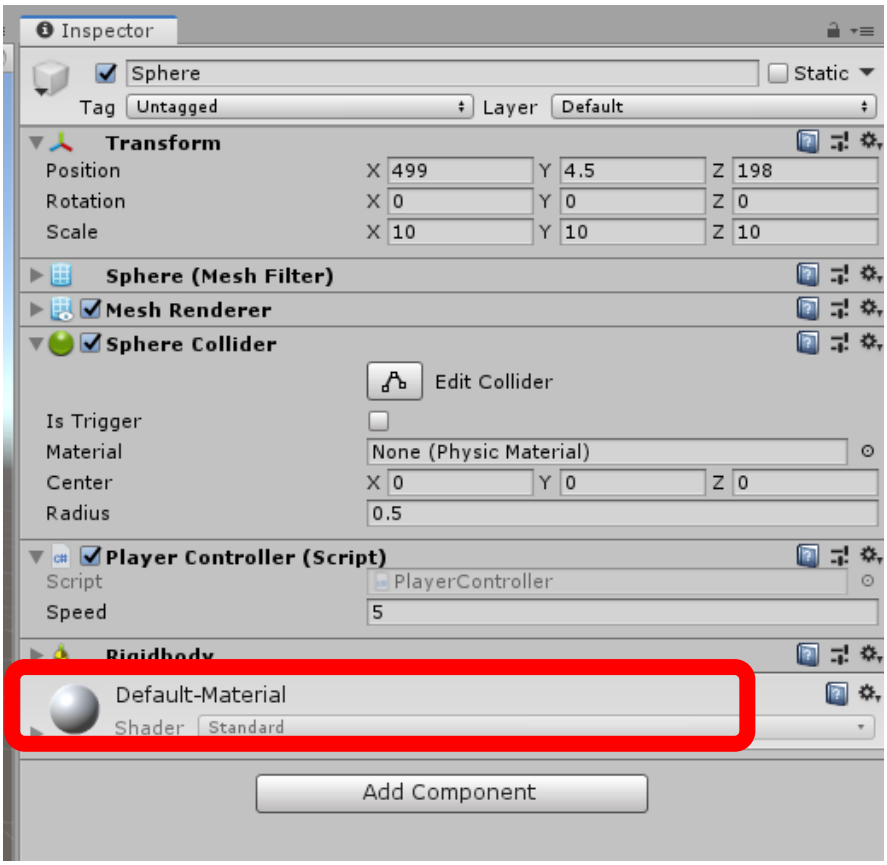
Скоростью объекта можно управлять, для этого создадим переменную ***public float speed = 5.0f;***
теперь ваш скрипт выглядит следующим образом:

PlayerController.cs

C: > Users > user > 3д > Assets > Scripts > PlayerController.cs

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

public class PlayerController : MonoBehaviour
{
    public float speed = 5.0f;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        transform.Translate(Vector3.forward* Time.deltaTime*speed);
    }
}

```

Сохраните и запустите игру. Теперь мы можем управлять скоростью, не меняя ее в коде:

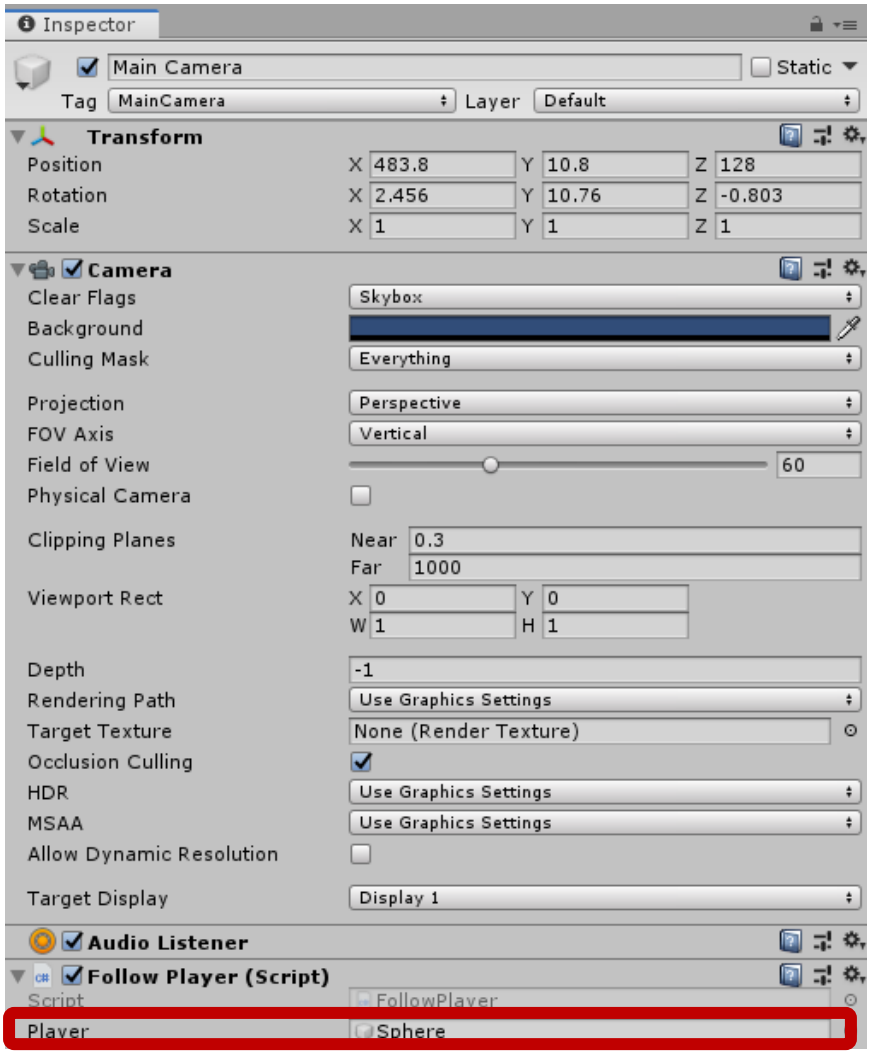


Создадим новый скрипт с названием FollowPlayer и присоеденим его к камере.

Откройте файл, и запишите:

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

public class FollowPlayer : MonoBehaviour
{
    public GameObject player;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        transform.position=player.transform.position;
    }
}
```

В настройках камеры перетащите сферу в окошко Player



Запустите игру.
Измените строку в Update на следующую:

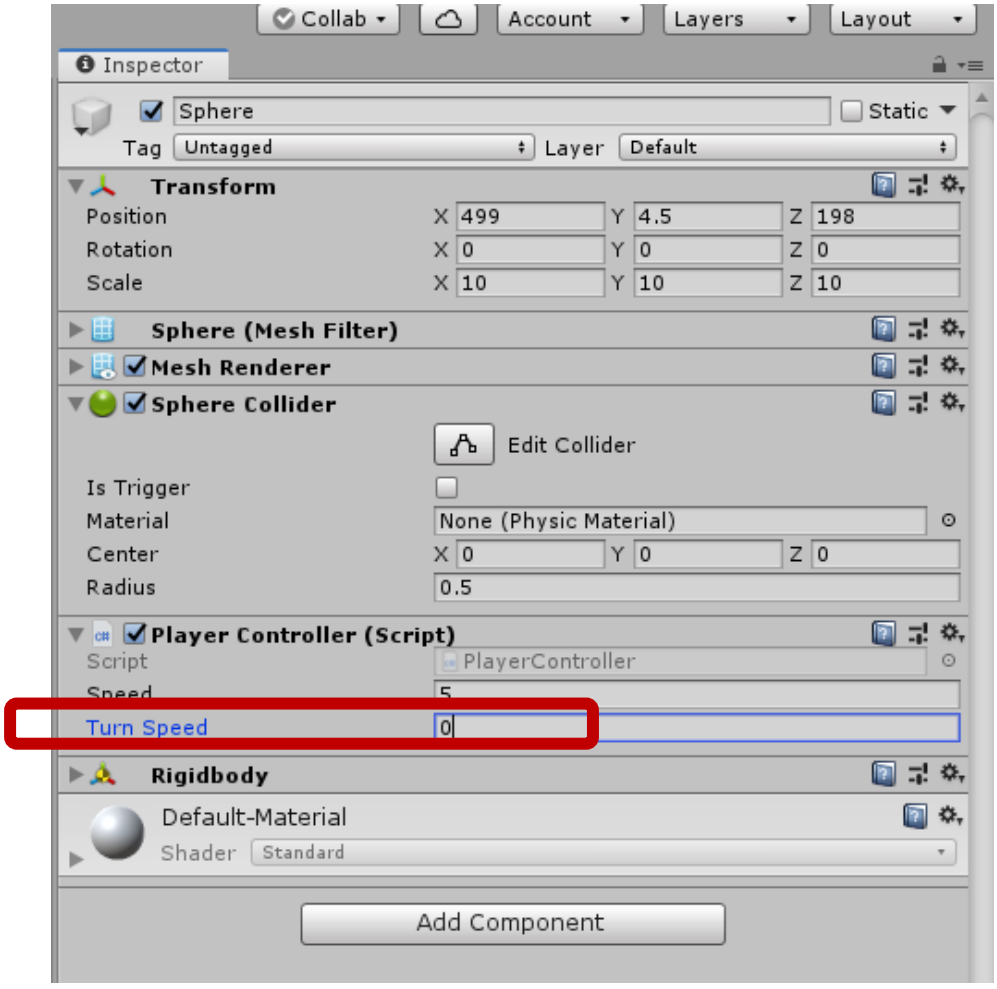
```
transform.position=player.transform.position+new Vector3 (0, 15, -20);
```

меняя координаты, добейтесь хорошего вида.
Откроем скрипт управления сферой, изменим код на следующий:

C: > Users > user > 3д > Assets > Scripts > PlayerController.cs

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class PlayerController : MonoBehaviour
6   {
7       public float speed = 5.0f;
8       public float turnSpeed;
9       // Start is called before the first frame update
10      void Start()
11      {
12
13      }
14
15      // Update is called once per frame
16      void Update()
17      {
18          transform.Translate(Vector3.forward* Time.deltaTime*speed);
19          transform.Translate(Vector3.right* Time.deltaTime*turnSpeed);
20      }
21  }
22
```

Теперь в режиме игры, меняя переменную Turn Speed, мы можем поворачивать:



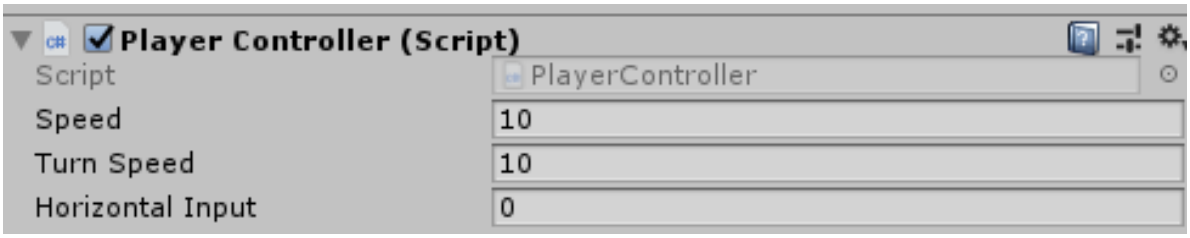
Однако, гораздо удобнее управлять с помощью кнопок
Для этого внесем изменения в скрипт

FollowPlayer.cs | PlayerController.cs

C: > Users > user > 3д > Assets > Scripts > PlayerController.cs

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

public class PlayerController : MonoBehaviour
{
    public float speed = 5.0f;
    public float turnSpeed;
    public float horizontalInput;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        horizontalInput = Input.GetAxis("Horizontal");
        transform.Translate(Vector3.forward* Time.deltaTime*speed);
        transform.Translate(Vector3.right* Time.deltaTime*turnSpeed*horizontalInput);
    }
}
```

В окне инспекторе настроим скорости следующим образом:



Запустите игру, стрелками вы можете управлять сферой.
Создадим управление движением вперед-назад, для этого создадим новую переменную

```
public float forwardInput;
```

и добавим строку в Update

```
forwardInput = Input.GetAxis("Vertical");
```

и изменим на

```
transform.Translate(Vector3.forward* Time.deltaTime*speed*forwardInput
);
```

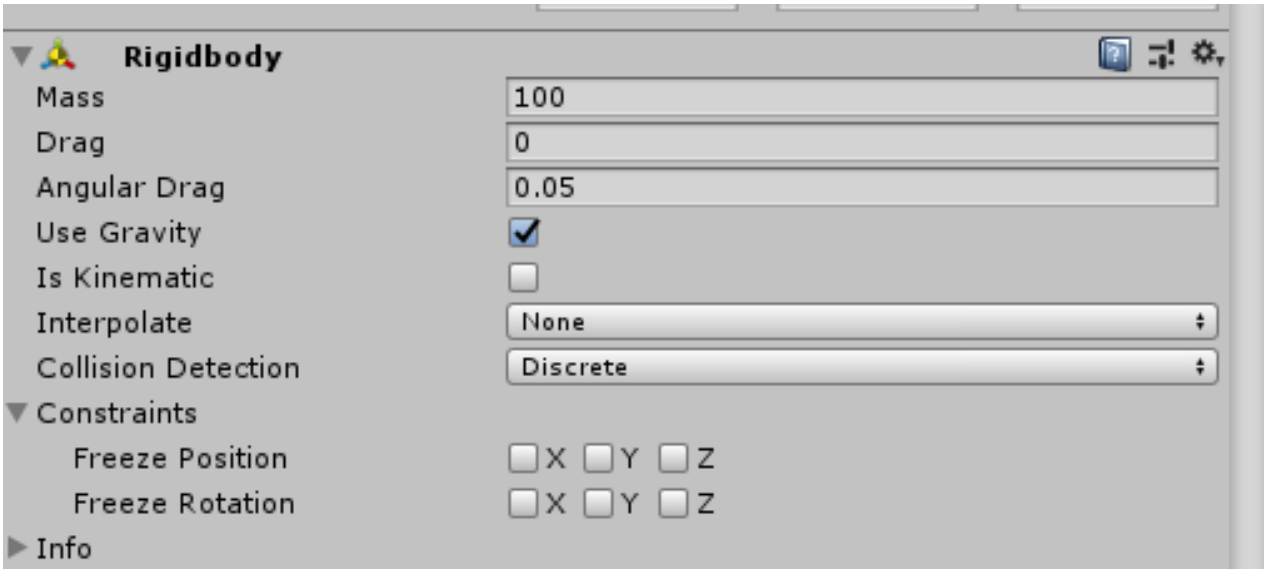
Сохраняем и запускаем игру.
* ваш код должен выглядеть следующим образом:

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

```
public class PlayerController : MonoBehaviour
{
    public float speed = 5.0f;
    public float turnSpeed;
    public float horizontalInput;
    public float forwardInput;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        horizontalInput = Input.GetAxis("Horizontal");
        forwardInput = Input.GetAxis("Vertical");
        transform.Translate(Vector3.forward* Time.deltaTime*speed*forwardI
nput);
        transform.Translate(Vector3.right* Time.deltaTime*turnSpeed*horizon
talInput);
    }
}
```

Удалим сферу со сцены.

Создадим еще один куб (не из префаб), настроим **RigidBody следующим образом**



Добавим скрипт PlayerController, просто перетащив его в пустое поле инспектора.

В камере добавим наш куб

Запустите игру. Что происходит?

Изменим код:

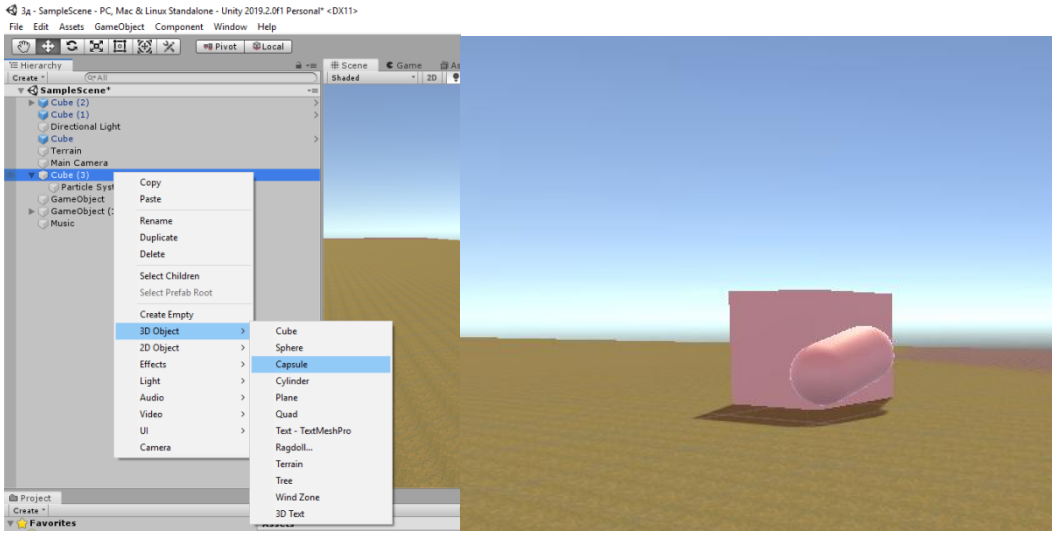
```
        void Update()
          {
             horizontalInput = Input.GetAxis("Horizontal");
             forwardInput = Input.GetAxis("Vertical");
             transform.Translate(Vector3.forward* Time.deltaTime*speed*forwardI
nput);
            transform.Rotate(Vector3.up, horizontalInput* Time.deltaTime*turnSpe
ed);
          }
```

Запустите игру, что изменилось? Добавьте результаты в свой проект.

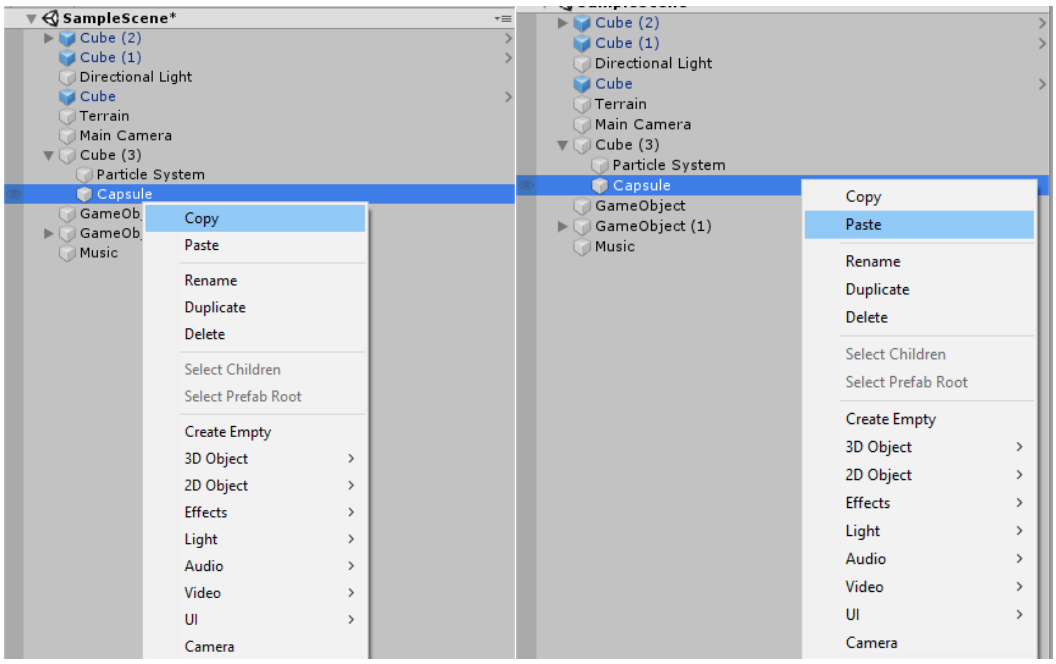
Поворот за мышкой

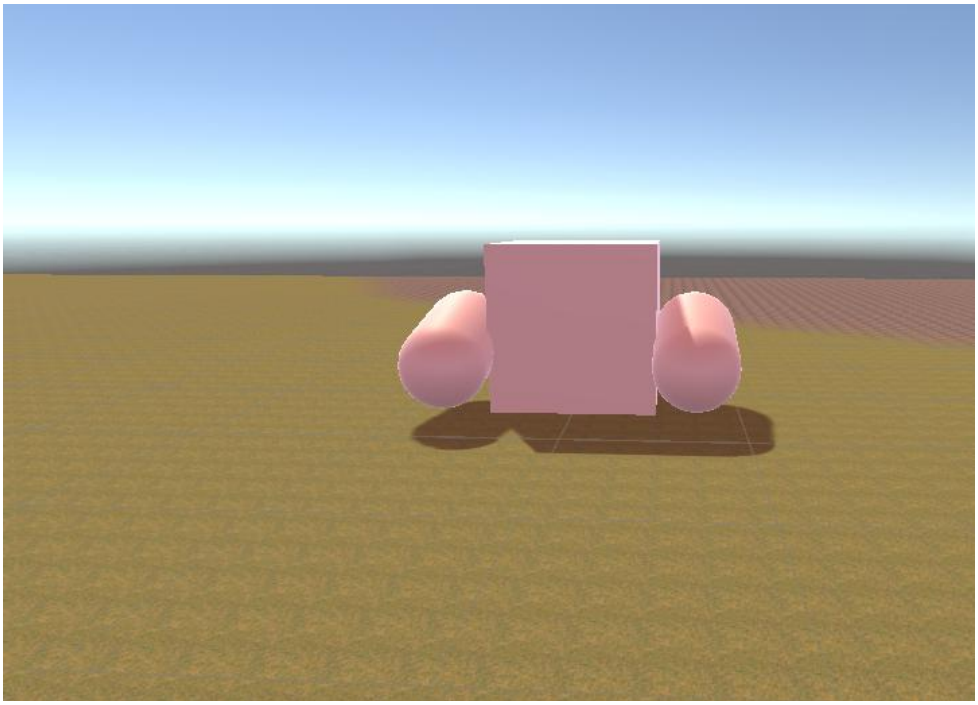
Откроем проект упражнение, и выберем наш подвижный кубик (если его нет, то создадим и присоединим к нему скрипт движения).

Сделаем наш кубик более сложным, для этого добавим ему «ручки» и «голову», для этого выделим наш объект, и нажмем правую клавишу мыши

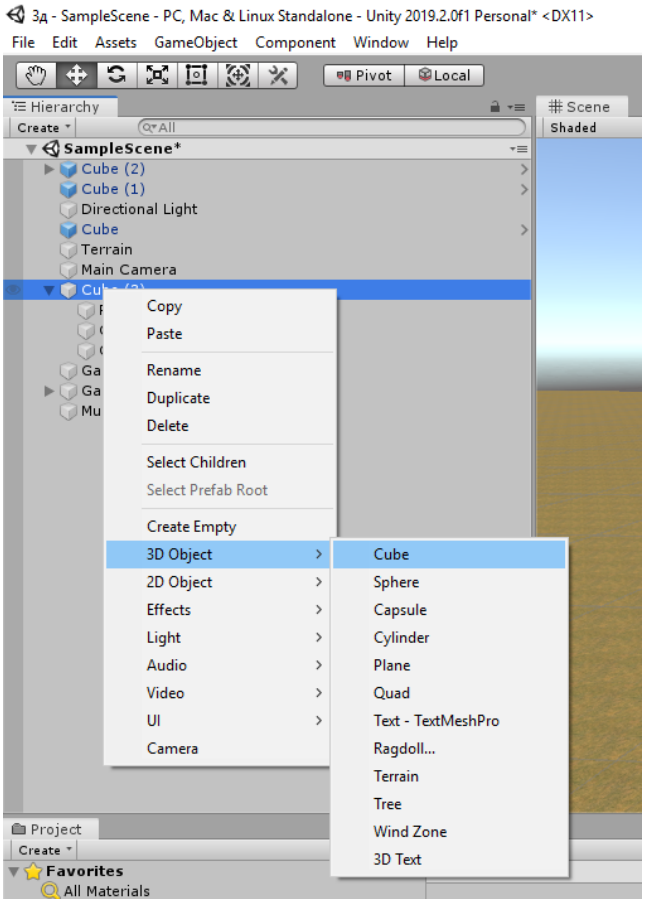


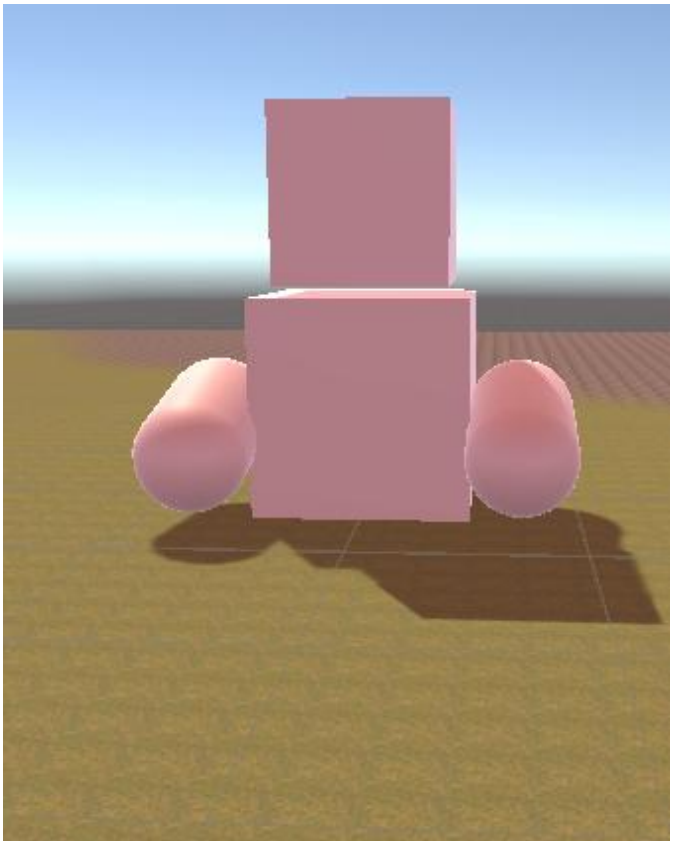
Скопируем капсулу и вставим ее



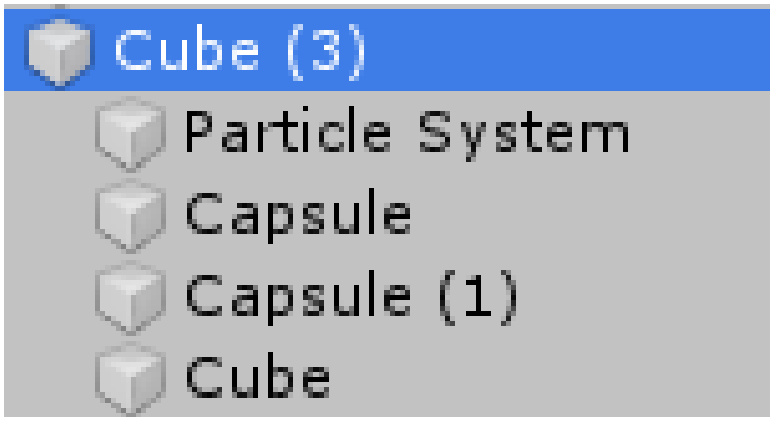


Добавим «голову», для этого создадим куб





Обратите внимание как должны располагаться объекты: капсулы-руки и куб-голова:



Это обозначает, что они являются дочерними объектами основного куба, а значит двигаться будут вместе с ним. Запустите игру и проверьте. Если вы все сделали правильно, то ваш герой по дороге не потеряет ручки и голову.

Теперь попробуем сделать так, чтобы голова героя поворачивалась за мышкой. Для этого создадим в папке Scripts новый скрипт с названием MouseFollow3D, и присоединим его к «голове»

В скрипте напишем (//это комментарии, их можно не набирать)

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

public class MouseFollow3D : MonoBehaviour {

 void Update()
    {
        var mousePosition = Input.mousePosition;
        mousePosition.z = transform.position.z - Camera.main.transform.position.z;
        // это только для перспективной камеры необходимо
        mousePosition = Camera.main.ScreenToWorldPoint(mousePosition);
        //положение мыши из экранных в мировые координаты
        var angle = Vector2.Angle(Vector2.right, mousePosition - transform.position);
        //угол между вектором от объекта к мыше и осью x
        transform.eulerAngles = new Vector3(0f,transform.position.y < mousePosition.y ? angle : -angle , 0f);
    }
}
```

Запустите игру, что происходит?

Попробуйте в последней строчке поменять на

```
transform.eulerAngles = new Vector3(0f , 0f,transform.position.z < mouseP
osition.z ? angle : -angle);
```

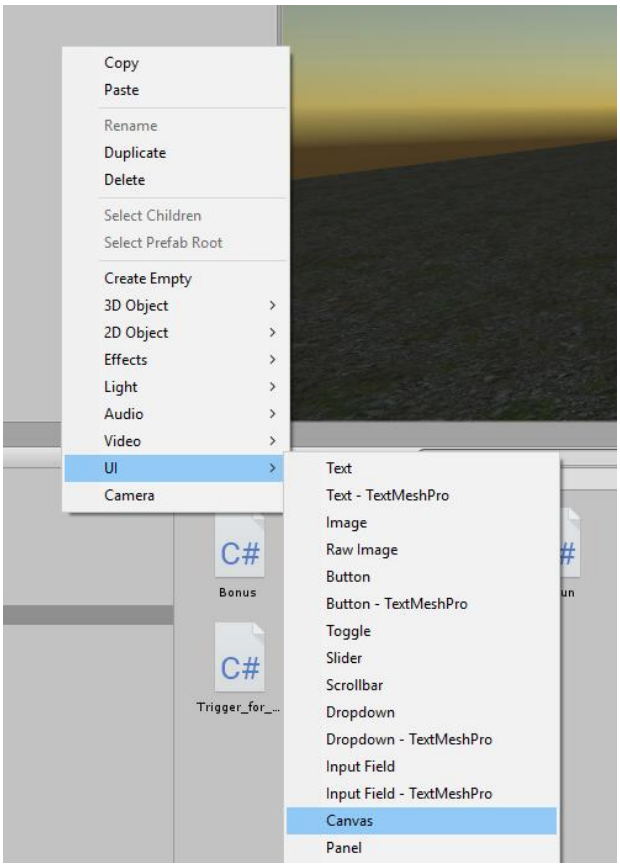
что изменилось?

А если так?

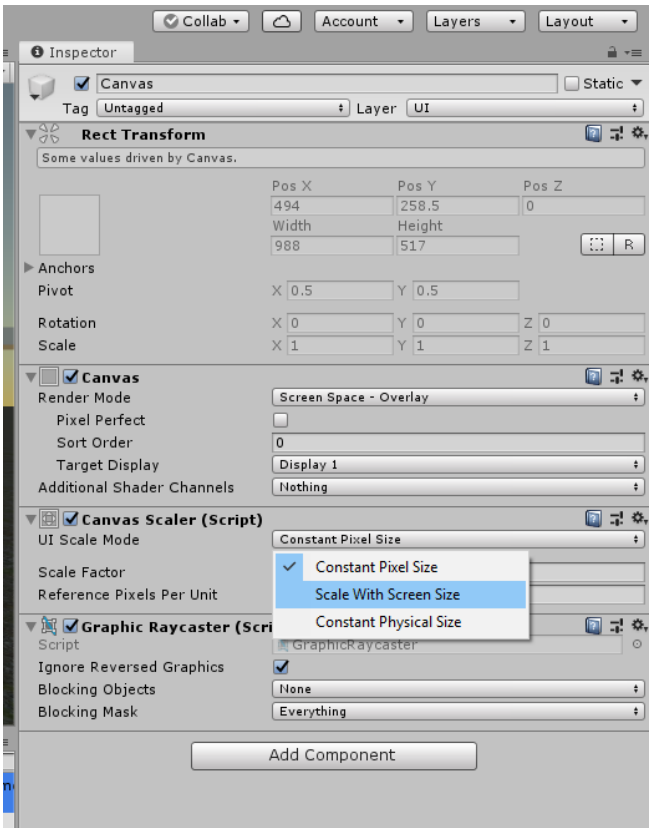
```
transform.eulerAngles = new Vector3(transform.position.x < mousePosition.
x ? angle : -angle ,0f, 0f);
```

Отображение переменных: Счет в игре

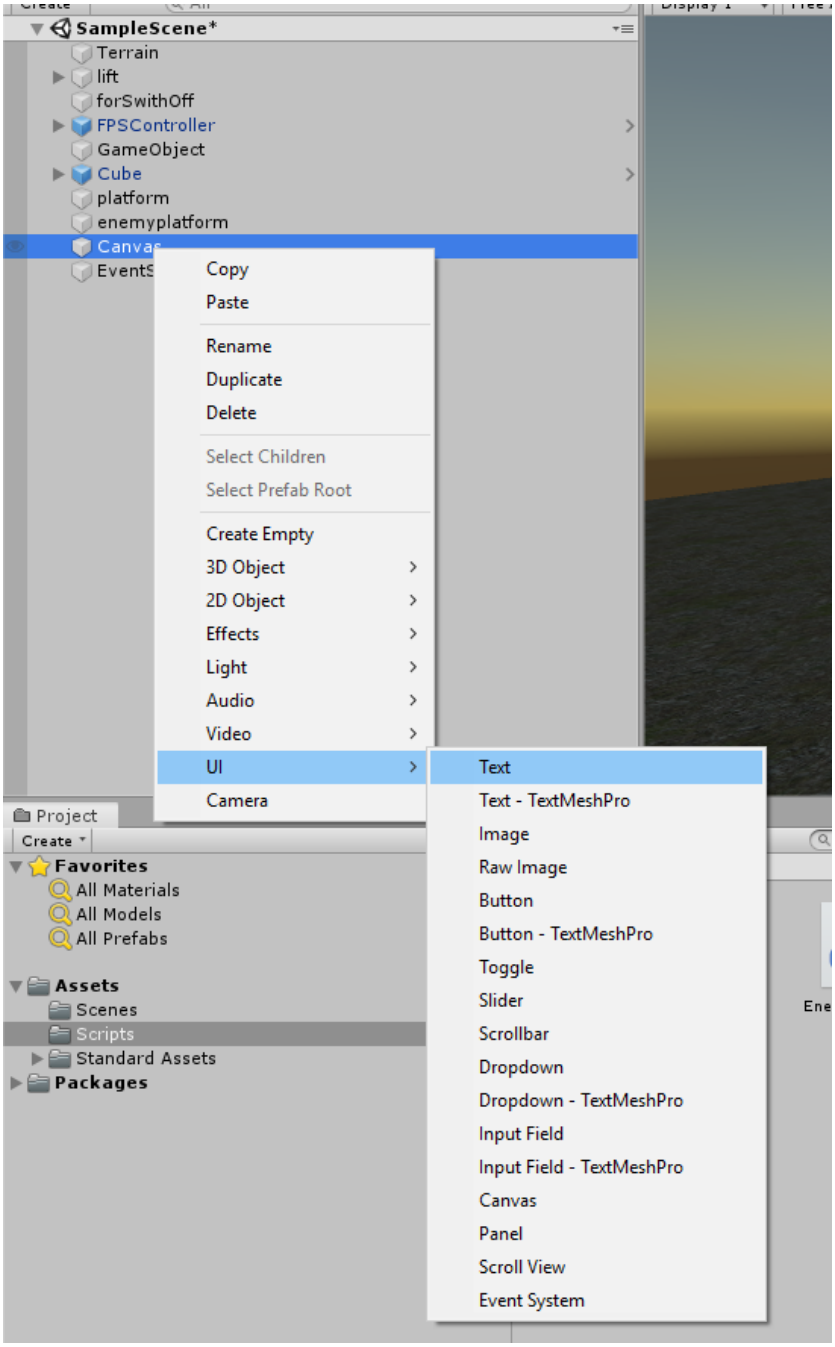
1. Создайте объект Canvas



2. Измените размеры

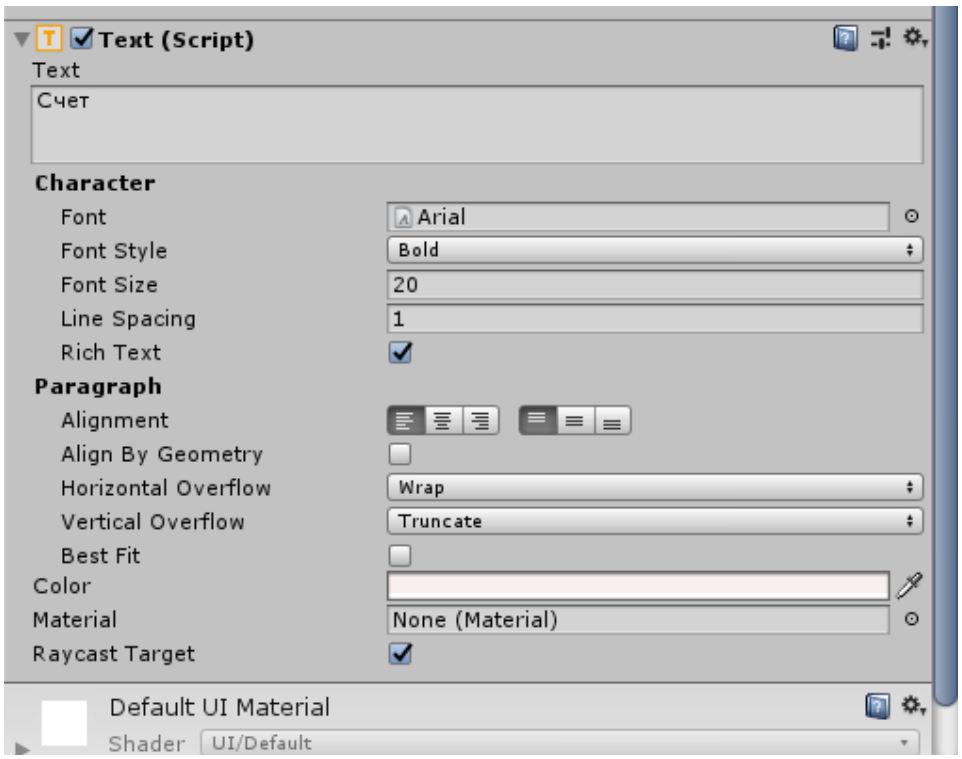


3. Создайте элемент UI- Text.

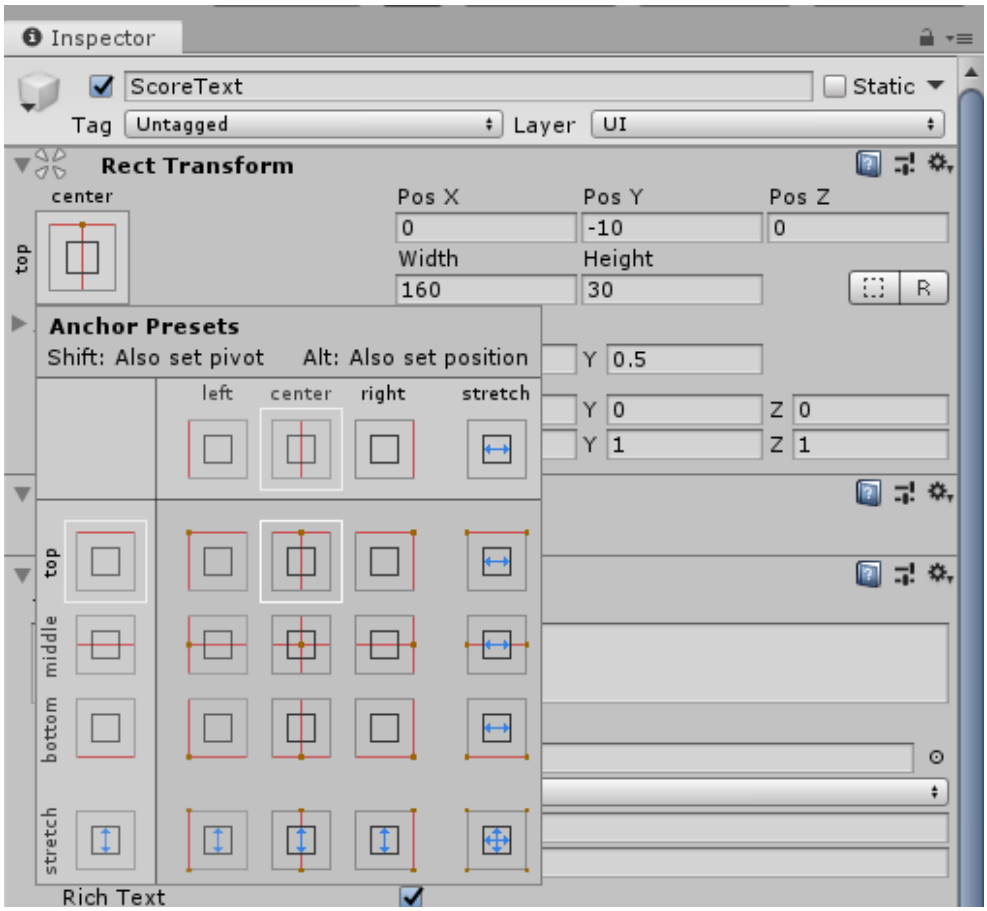


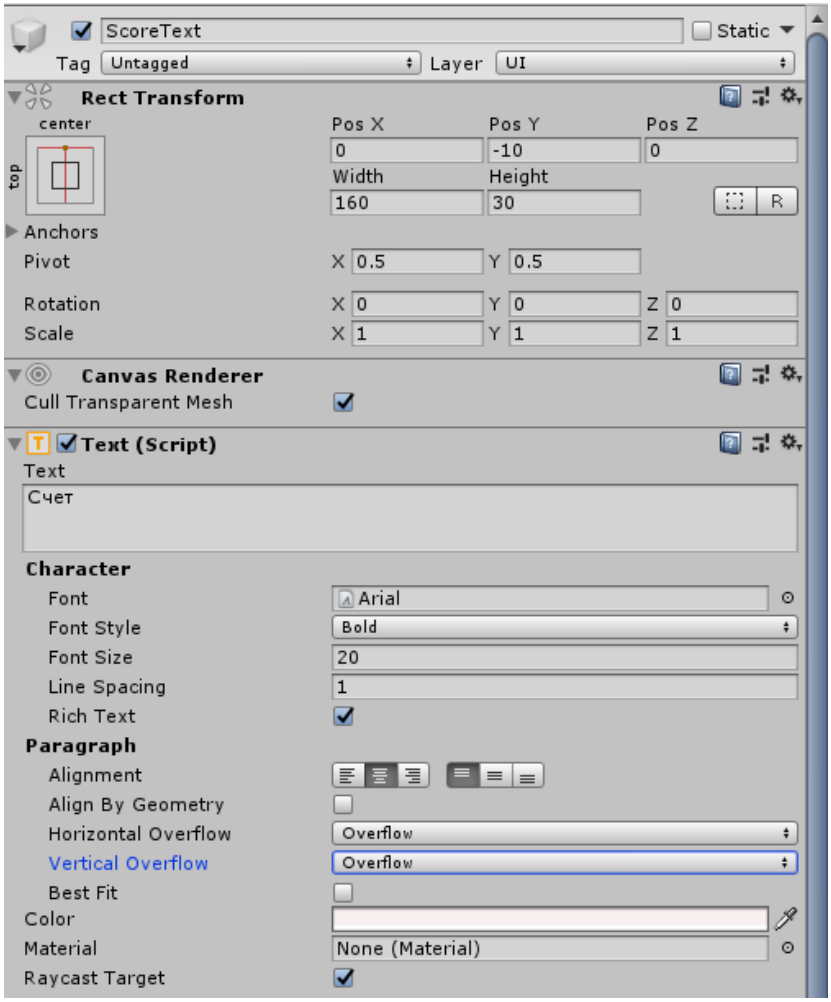
4. Переименуйте в ScoreText.

5. Настройте цвет, шрифт, размер текста в окне инспектор.



6.Расположите счет вверху в центре, для этого измените в окне инспектор как показано на картинке:





7. Создайте скрипт Score.
8. Внутри напишите:

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

public class Score : MonoBehaviour
{
  public static int score;
  Text text;
  void Awake()
  {
     text=GetComponent<Text>();
     score=0;
  }
 void Update()
  {
    text.text="Счет: "+ score;
  }
}
```

Обратите внимание на выделенный желтым текст:

```
using UnityEngine.UI;
```

без этой строки у вас скрипт не будет работать с элементами UI

static – позволяет обращаться к этой переменной из любого другого скрипта с указанием Score.score (Класс – Score, переменная score)

```
void Awake()
```

похоже на void Start(), позволяет производить настройки UI один раз, в прошлых версиях можно было использовать void Start()

1. Добавьте скрипт Score к нашему текстовому элементу (для этого просто перетащите)
2. Чтобы менять счет используйте конструкцию:

```
Score.score+=10;
```

(возможно использовать не только +=10, но и любые другие алгебраические действия, не желательно деление).