

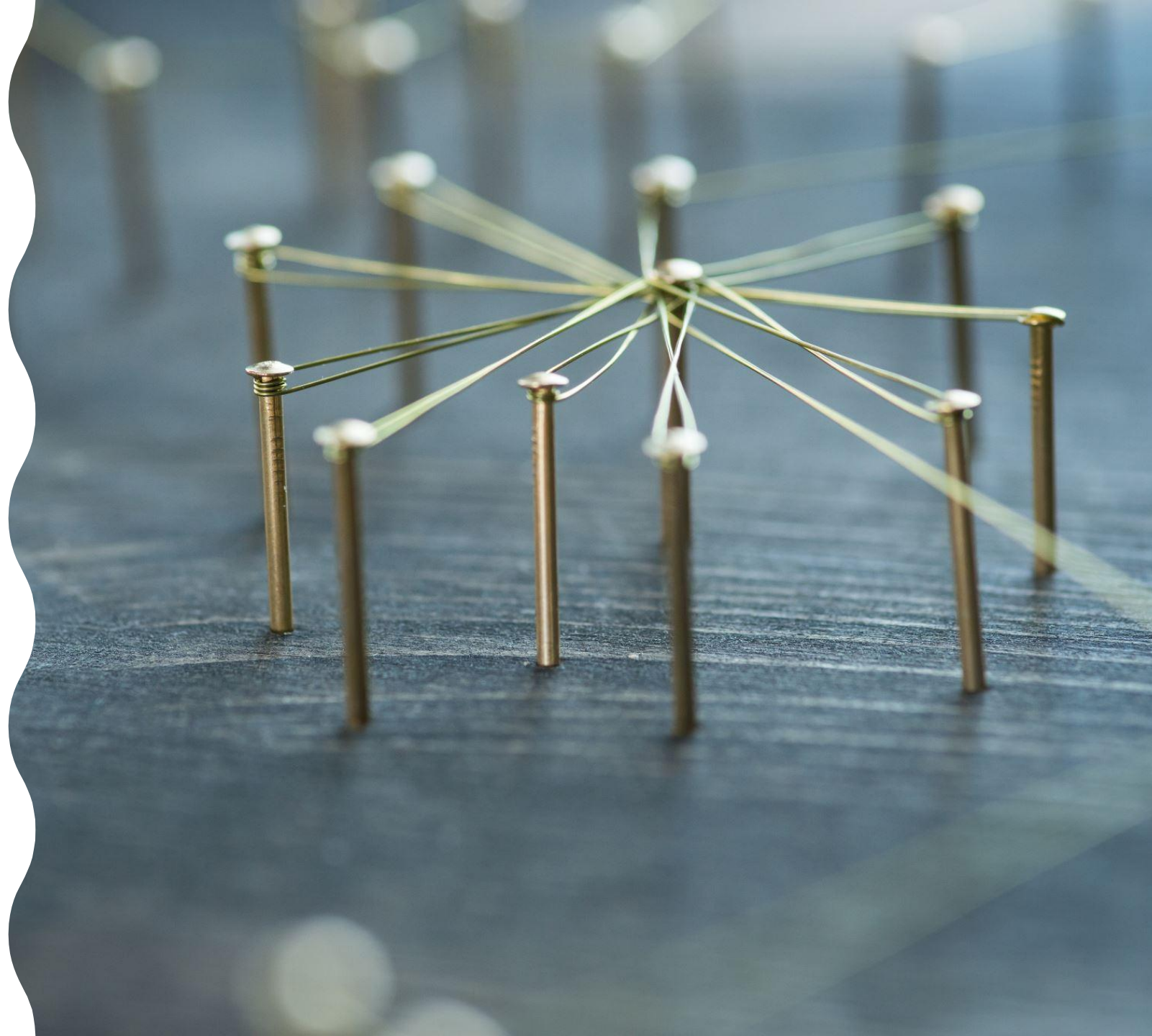
Знакомств о с Unity2D

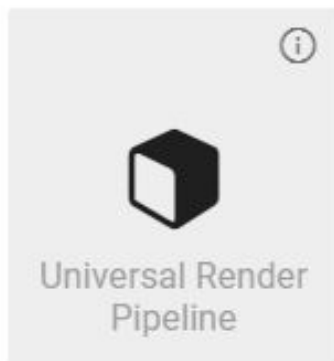
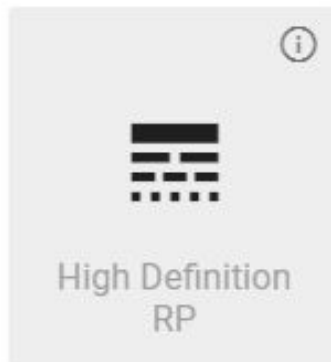
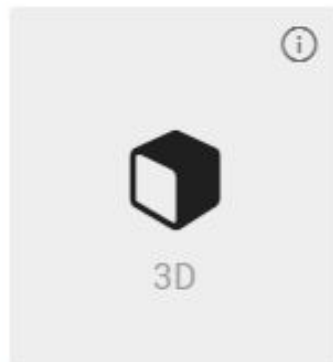
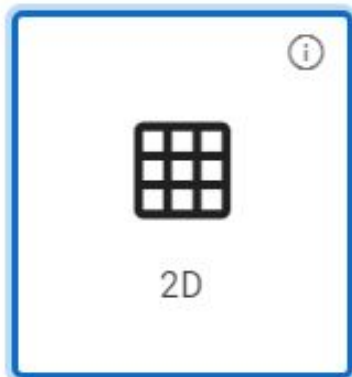
Занятие 1.

Хорошо бы ОЗНАКОМИТЬСЯ

Я:

- Руководство по: **Unity**
для 2D-игры
- https://unity3d.com/ru/2d/solution-guide?_ga=2.209545306.998543116.1612716530-2130069261.1588601792





Project Name *

Sample2D

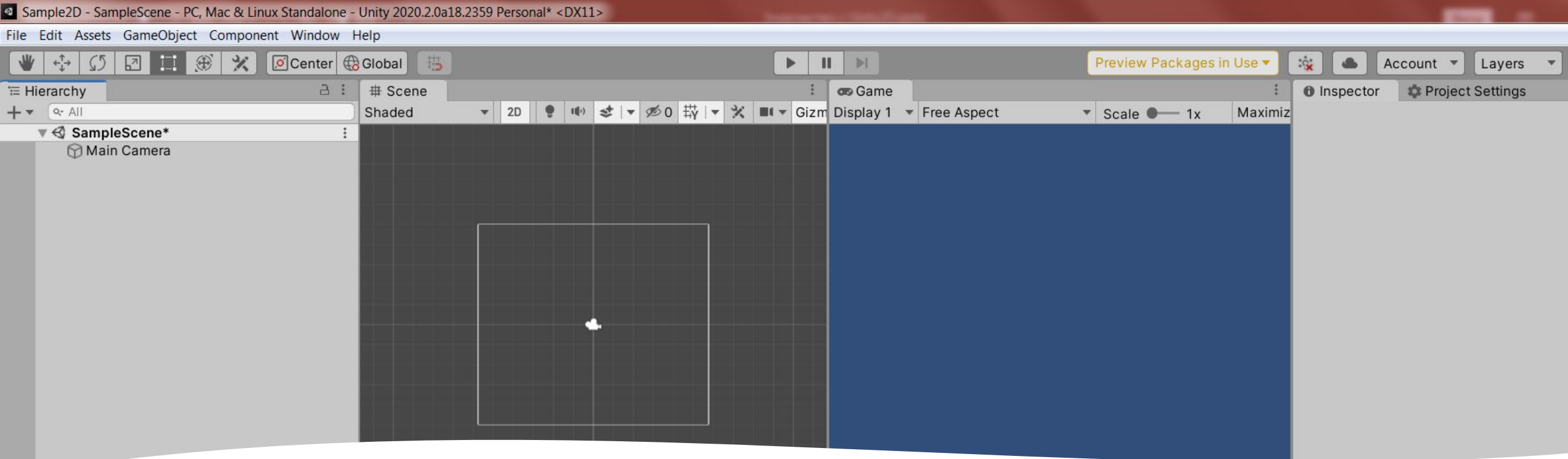
Location *

C:\Users\Admin\Documents\Unity



Начинаем работу из Unity Hub

Создаем новый 2D проект

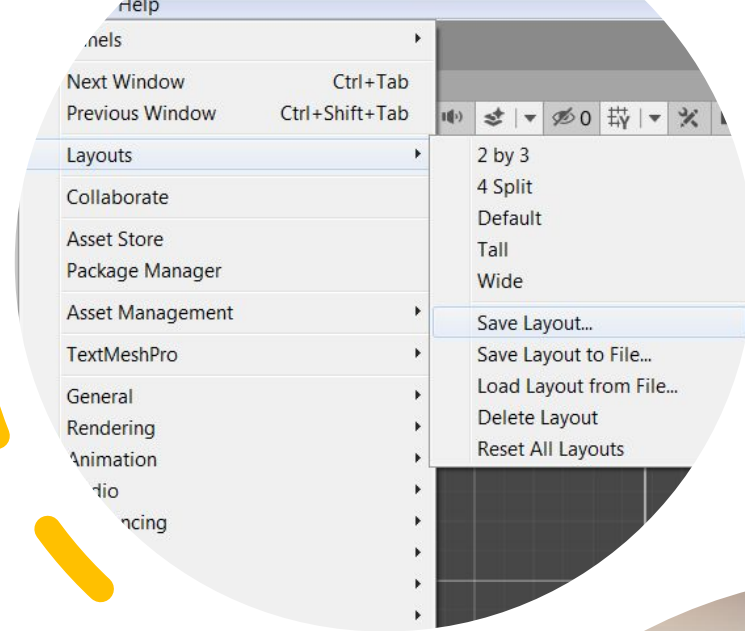


Создаем новый проект.

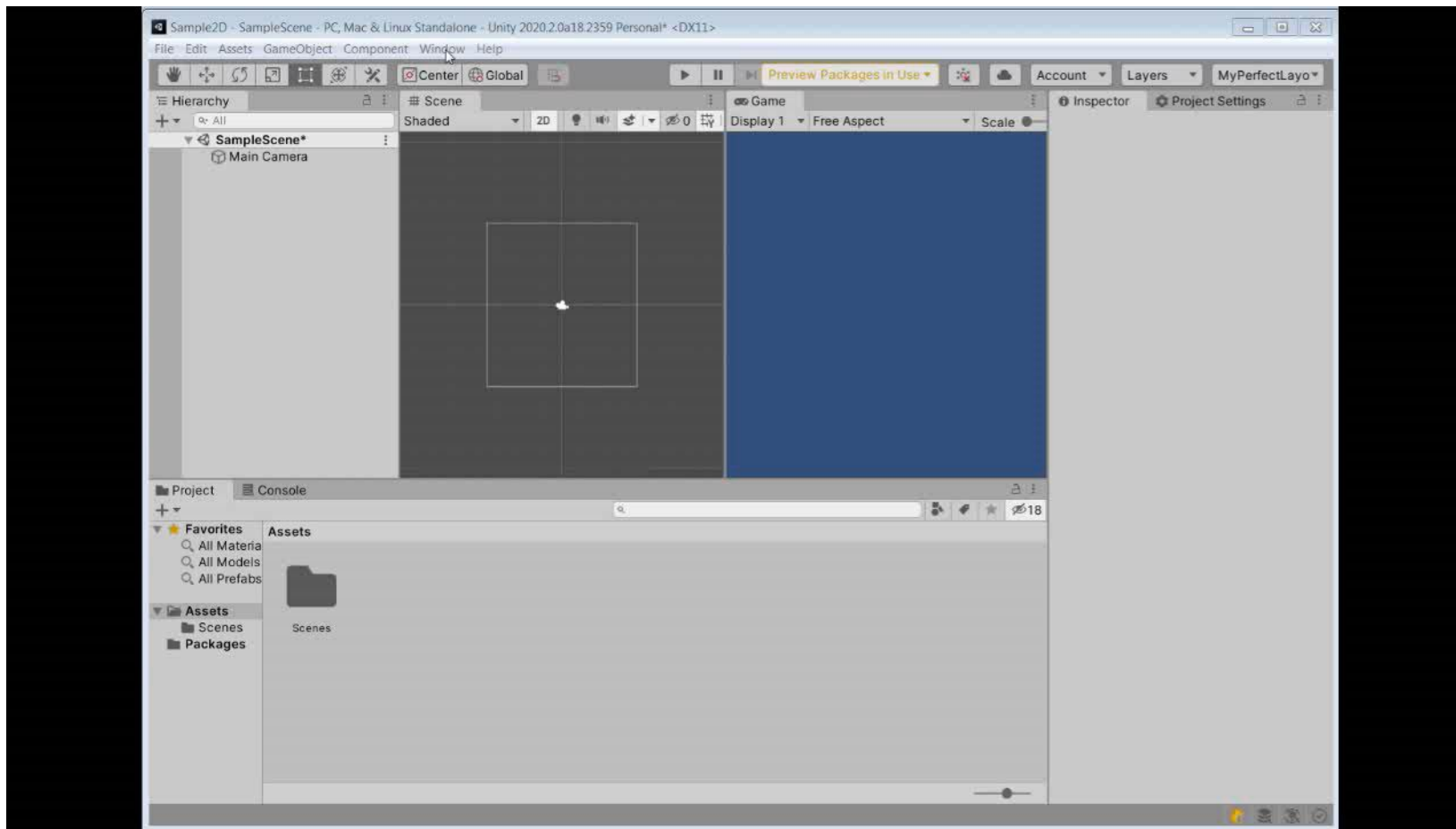
- При создании нового 2D проекта появляется пустая плоская сцена с камерой (Main Camera), название которой видно во вкладке Hierarchy (слева), а сама она видна на сцене (середина окна) в виде камеры (Микки Мауса).
- Справа – вкладка Inspector. Сейчас она пустая, но если выбрать на сцене или в Hierarchy объект, то Inspector заполнится свойствами этого объекта.

Расположение окон в редакторе (Layout)

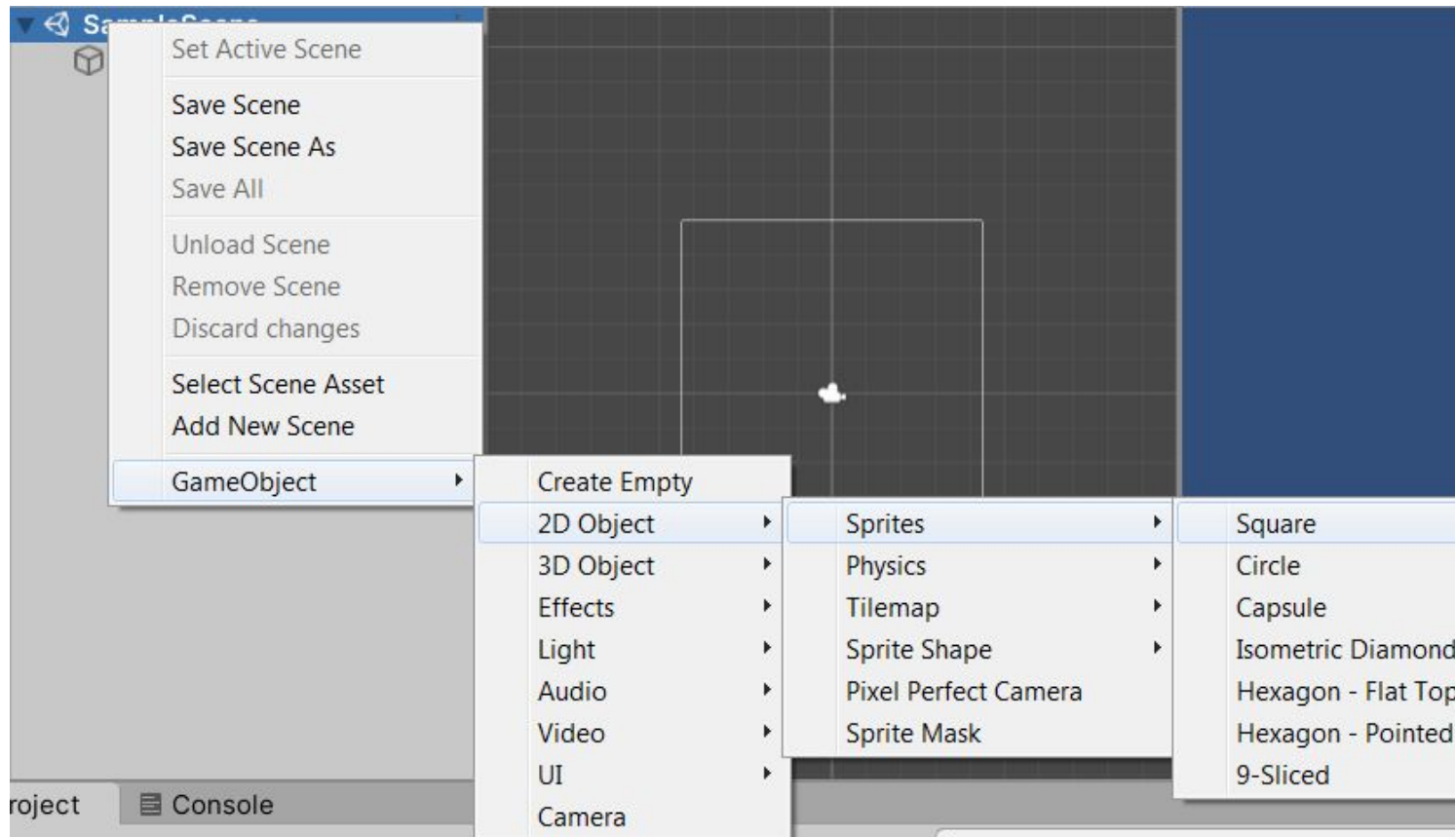
- В редакторе Unity Вы можете редактировать расположение окон по вашему вкусу. То расположение, которое Вы видели на предыдущем слайде – Layout, который удобен мне. Для формирования своего собственного, удобного Вам – зайдите в меню Windows -> Layouts -> (Далее на Ваш выбор). Затем, можно еще поперетаскивать панельки, то есть довести их взаимное расположение до совершенства и сохранить в том же меню.



Пример



Создадим первый объект – квадрат.



Во вкладке Hierarchy нажимаем правой кнопкой мыши на самом верхнем объекте (сцене), в выпадающих списках выбираем поочередно

GameObject->2D Object->Sprites->Square

- GameObject (игровой объект) – базовый класс для всех объектов сцены в Unity.
- <https://docs.unity3d.com/2020.2/Documentation/ScriptReference/GameObject.html>

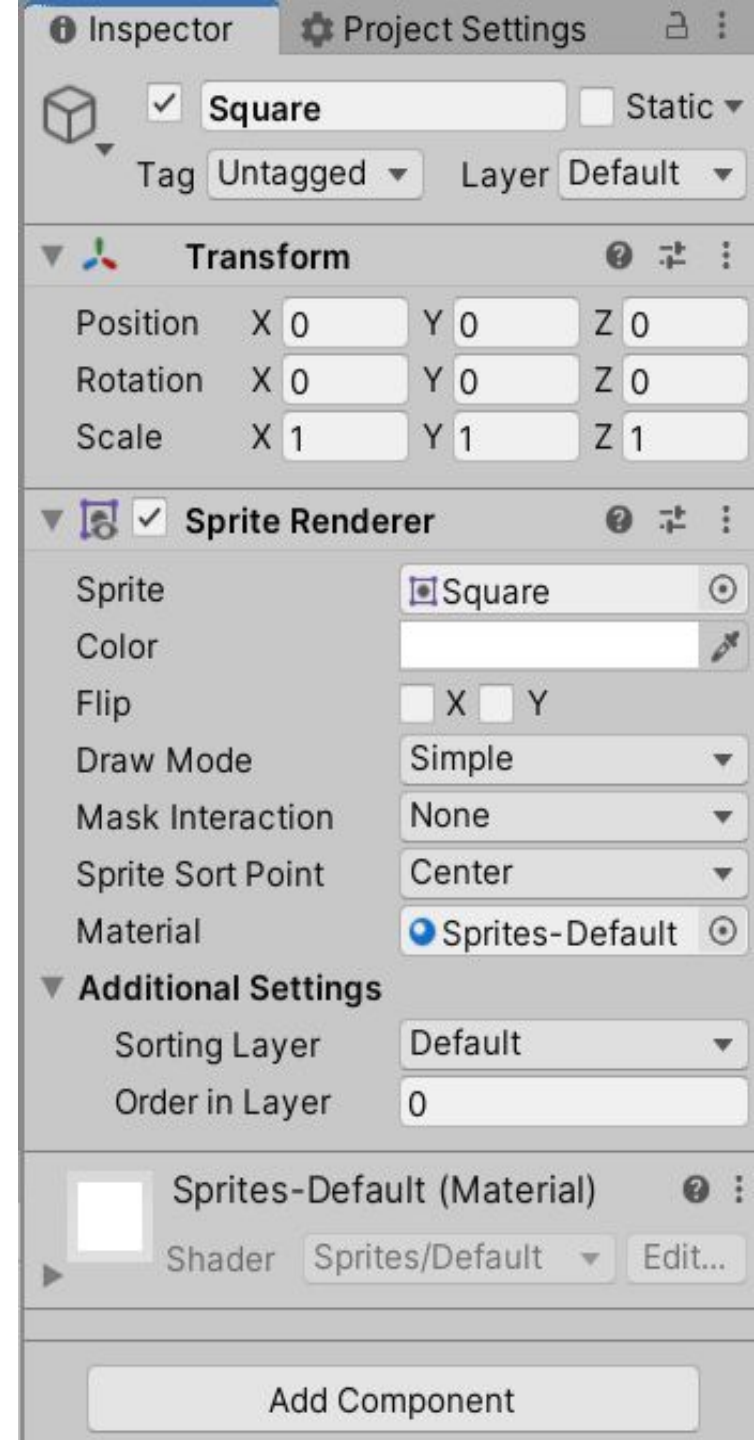
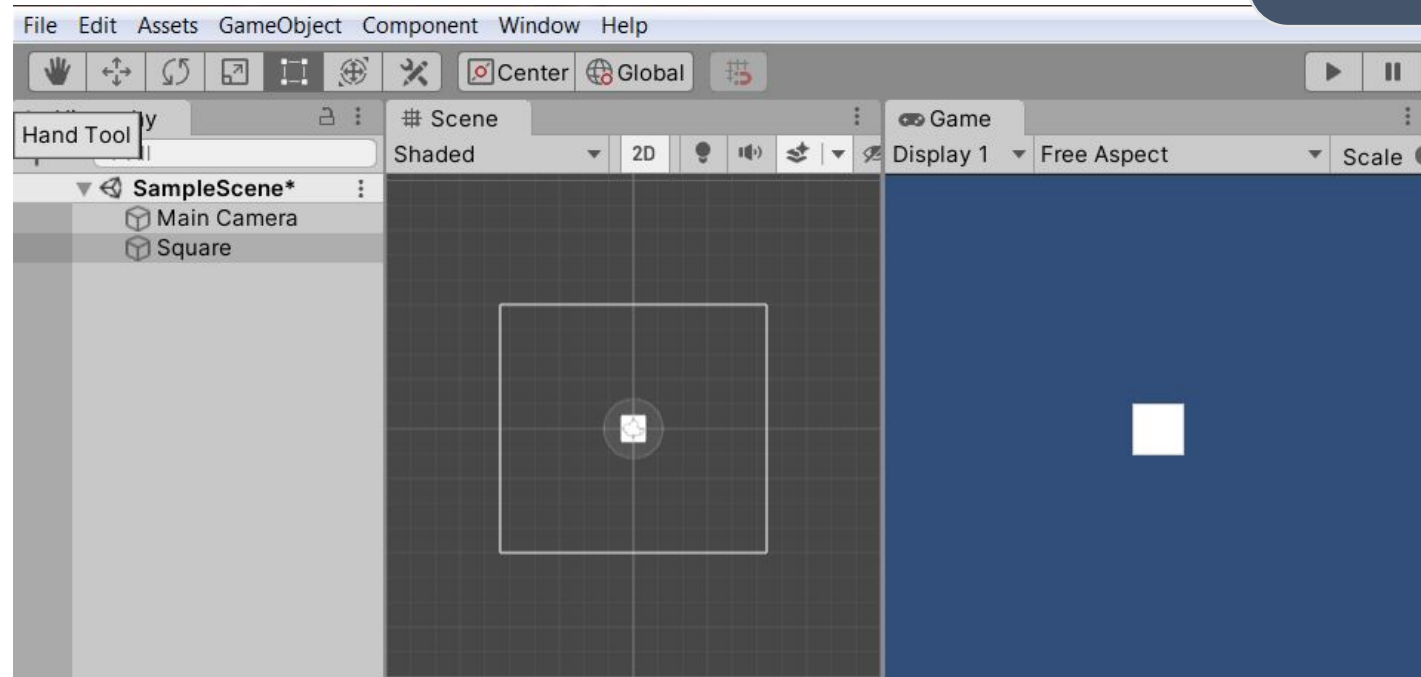
Результат
добавления
квадрата к
сцене.

Во вкладке Hierarchy
появился новый объект
Square (его можно
переименовать по
своему)

На сцене фокус на
квадрате

Вкладка Inspector
заполнилась свойствами
объекта, заданными по
умолчанию:

- Transform
- Sprite Renderer



Спрайты (Sprites)



- Спрайты - это объекты 2D-графики. Если вы привыкли работать в 3D, спрайты - это, по сути, просто стандартные текстуры, но есть специальные методы для объединения и управления текстурами спрайтов для повышения эффективности и удобства во время разработки.
- <https://docs.unity3d.com/ru/2019.4/Manual/Sprites.html>

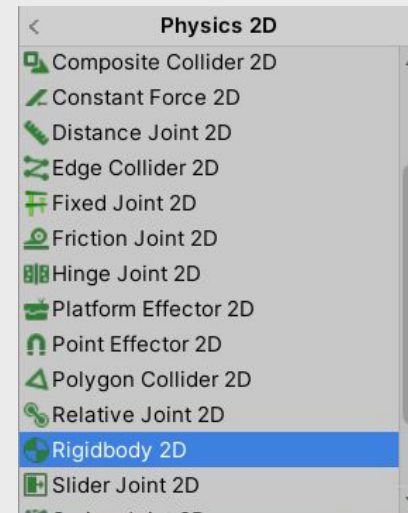
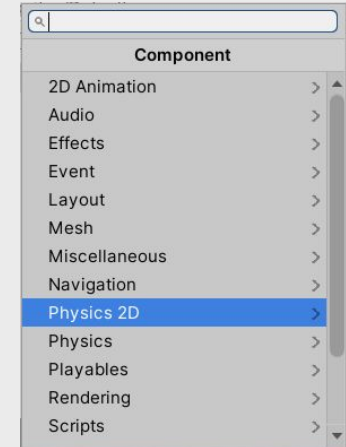
Свойства игрового объекта (GameObject) в 2D подробнее.

- Transform: компонент Transform задает местоположение, поворот и масштаб игрового объекта. Это единственный компонент, присутствующий во всех игровых объектах.
 - <https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Transform.html>
- SpriteRenderer: Компонент Sprite Renderer отображает спрайт и контролирует его визуальное отображение в сцене как для 2D, так и для 3D-проектов.
 - <https://docs.unity3d.com/ru/2020.2/Manual/class-SpriteRenderer.html>

Добавление нового компонента (Component) к игровому объекту.

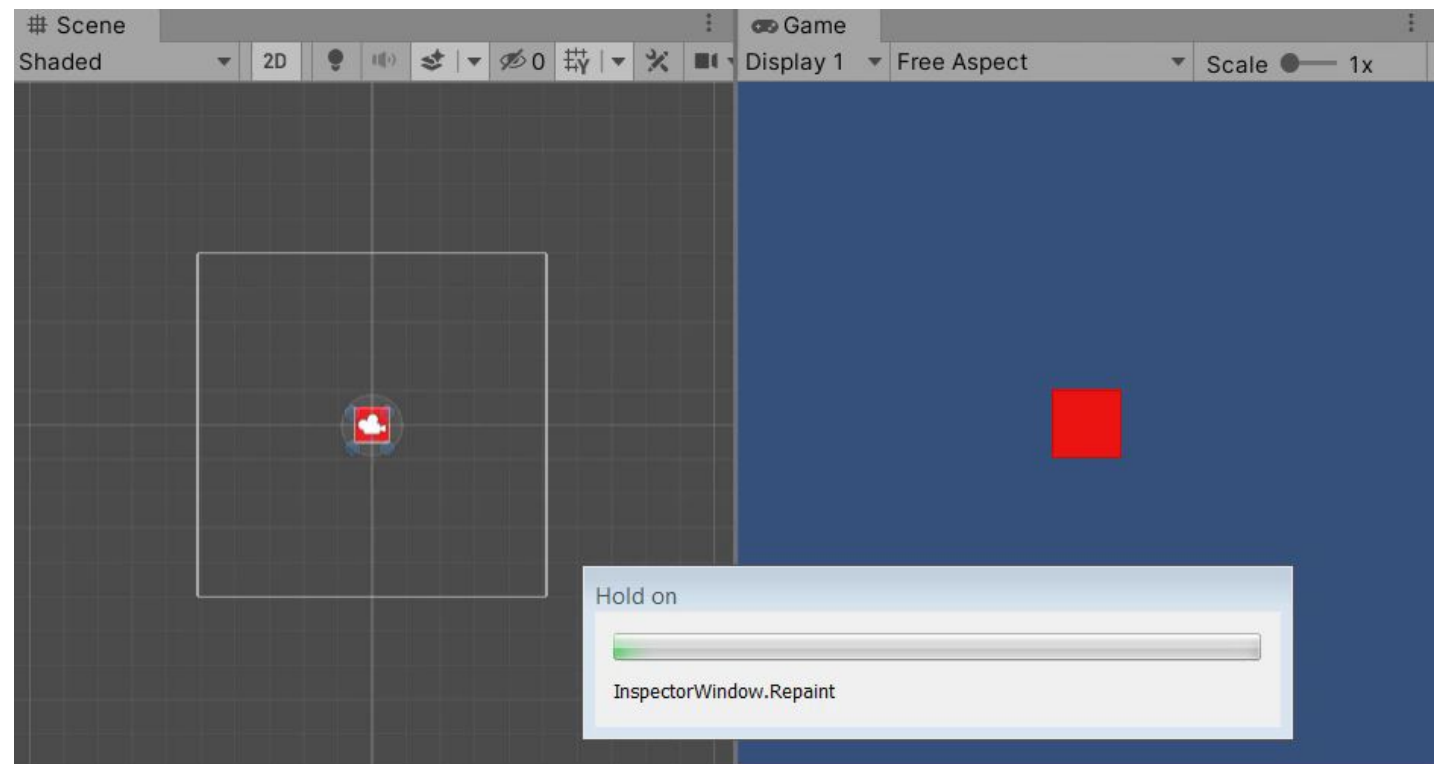
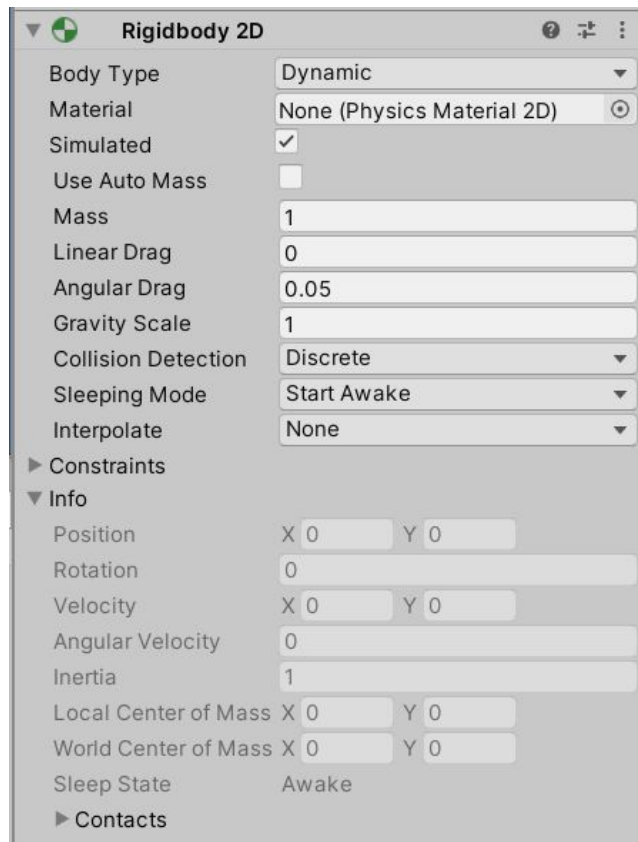
- Внизу во вкладке Inspector находим кнопку Add Component.
 - <https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Component.html>
- При нажатии в выпадающих списках поочередно выбираем Physics2D->Rigidbody2D
 - <https://docs.unity3d.com/ru/2020.2/Manual/class-Rigidbody2D.html>

Add Component



Компонент Rigidbody 2D

- Данный компонент сообщает движку Unity, что для данного игрового объекта должно имитироваться действие законов физики в 2D плоскости, например, действие сил тяжести, трения, сопротивления, удар. Также компонент Rigidbody 2D позволяет объекту с коллайдером перемещаться в пространстве. Без него при перемещении объекта путем изменения его координат коллайдер может действовать ненадежно.
- Сейчас включена гравитация (Gravity Scale равно 1), поэтому квадрат в игровом режиме падает.
 - <https://docs.unity3d.com/ru/2020.2/Manual/class-Rigidbody2D.html>



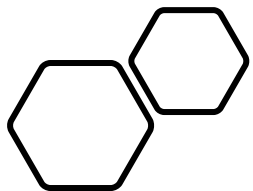
Физическое
моделирование
в Unity
осуществляется
в СИ, то есть

1 единица
расстояния
= 1м

1 единица
массы = 1кг

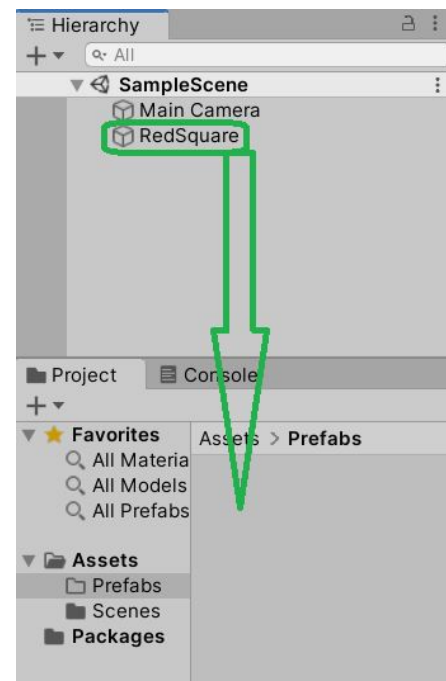
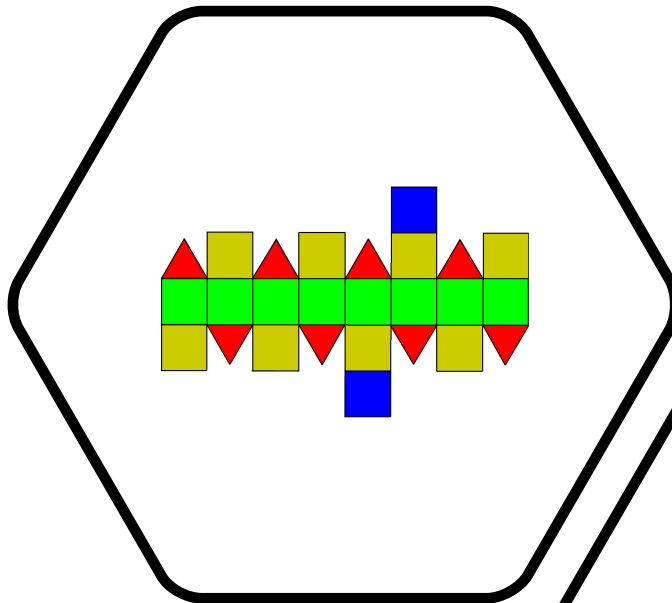
Ускорение
свободного
падения =
-9.8м/с

Средний
рост
персонажа-
человека
составляет
2 единицы.



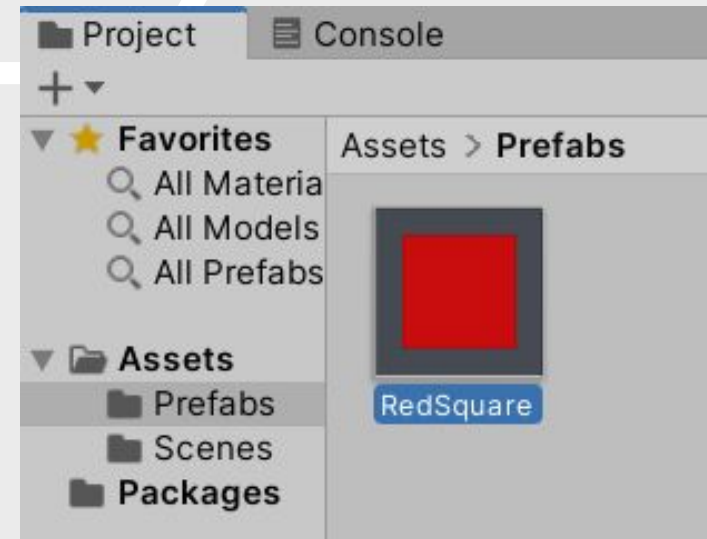
Создание шаблона

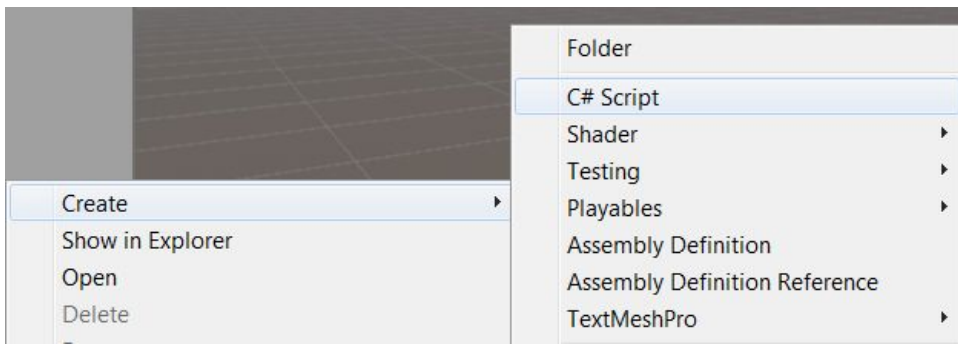
- Шаблон (prefab) – это многократно используемый элемент в проекте, который можно использовать для создания любого количества экземпляров.
- Для создания шаблона из кубика перетащите элемент Cube из Hierarchy в Project, папку Assets.



Результат перетаскивания:

- В открытой папке Assets/Prefab панели Project появился **новый элемент RedSquare**
- Во вкладке Hierarchy RedSquare стал синим.
- После этого будем работать только с шаблоном. Из Hierarchy RedSquare удалим



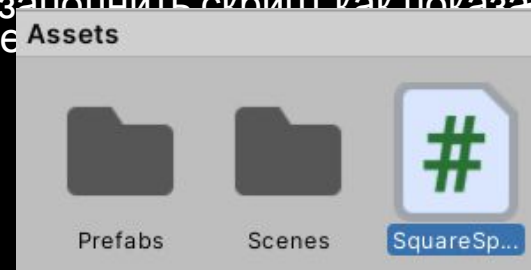


MonoBehaviour – базовый класс для любого unity-сценария (скрипта).
<https://docs.unity3d.com/2020.2/Documentation/ScriptReference/MonoBehaviour.html>

Теперь добавим немного кода в элемент Main Camera

Это можно сделать несколькими путями:

- С помощью контекстного меню в Assets создать новый C# скрипт, дважды кликнуть по нему, и, перейдя в Visual Studio, заполнить скрипт как показано слева на рисунке, а затем перетащить этот скрипт из Assets на объект Main Camera в Hierarchy.
- Выделить компонент Main Camera на сцене или в Hierarchy, в Inspector нажать кнопку Add Component и выбрать C# скрипт. Далее опять дважды кликнуть по нему, и, перейдя в Visual Studio, заполнить скрипт как показано слева на рисунке.



```
SquareSpawner.cs* [X]
Assembly-CSharp [v] SquareSpawner

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  [Unity Script | 0 references]
6  public class SquareSpawner : MonoBehaviour
7  {
8      public GameObject newSquare;
9      // Start is called before the first frame update
10     [Unity Message | 0 references]
11     void Start()
12     {
13         Instantiate(newSquare);
14     }
15     // Update is called once per frame
16     [Unity Message | 0 references]
17     void Update()
18     {
19     }
20 }
```

Instantiate() - Клонировать объект и возвращает клон (аналог Duplicate) в редакторе. Если вы клонируете GameObject, вы можете указать его transform.position и transform.rotation (по умолчанию они такие же, что и у оригинала). Если вы клонируете Component, GameObject, к которому он присоединен, также клонируется.

<https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Object.Instantiate.html>

Связь Unity и Visual Studio



Часто, особенно если Visual Studio уже стоит на Вашем компьютере, Unity автоматически привязывается к ней.

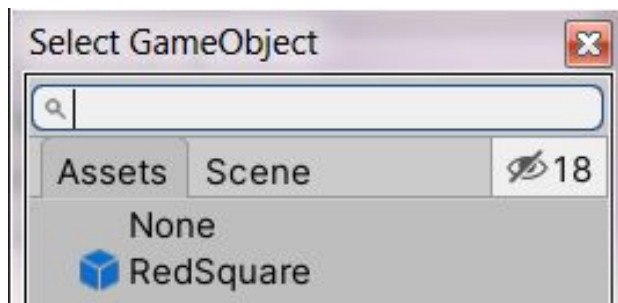
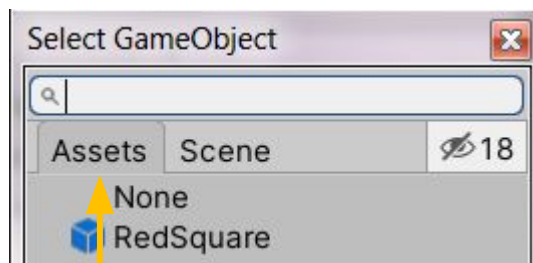
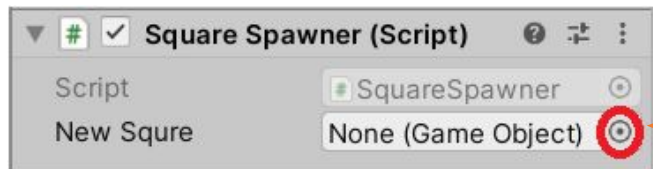
Однако, бывает, что эту привязку необходимо провести самостоятельно.

По следующей ссылке Вы найдете пошаговую инструкцию, как это сделать



<https://docs.microsoft.com/ru-ru/visualstudio/gamedev/unity/get-started/getting-started-with-visual-studio-tools-for-unity?view=vs-2019&pivots=windows>

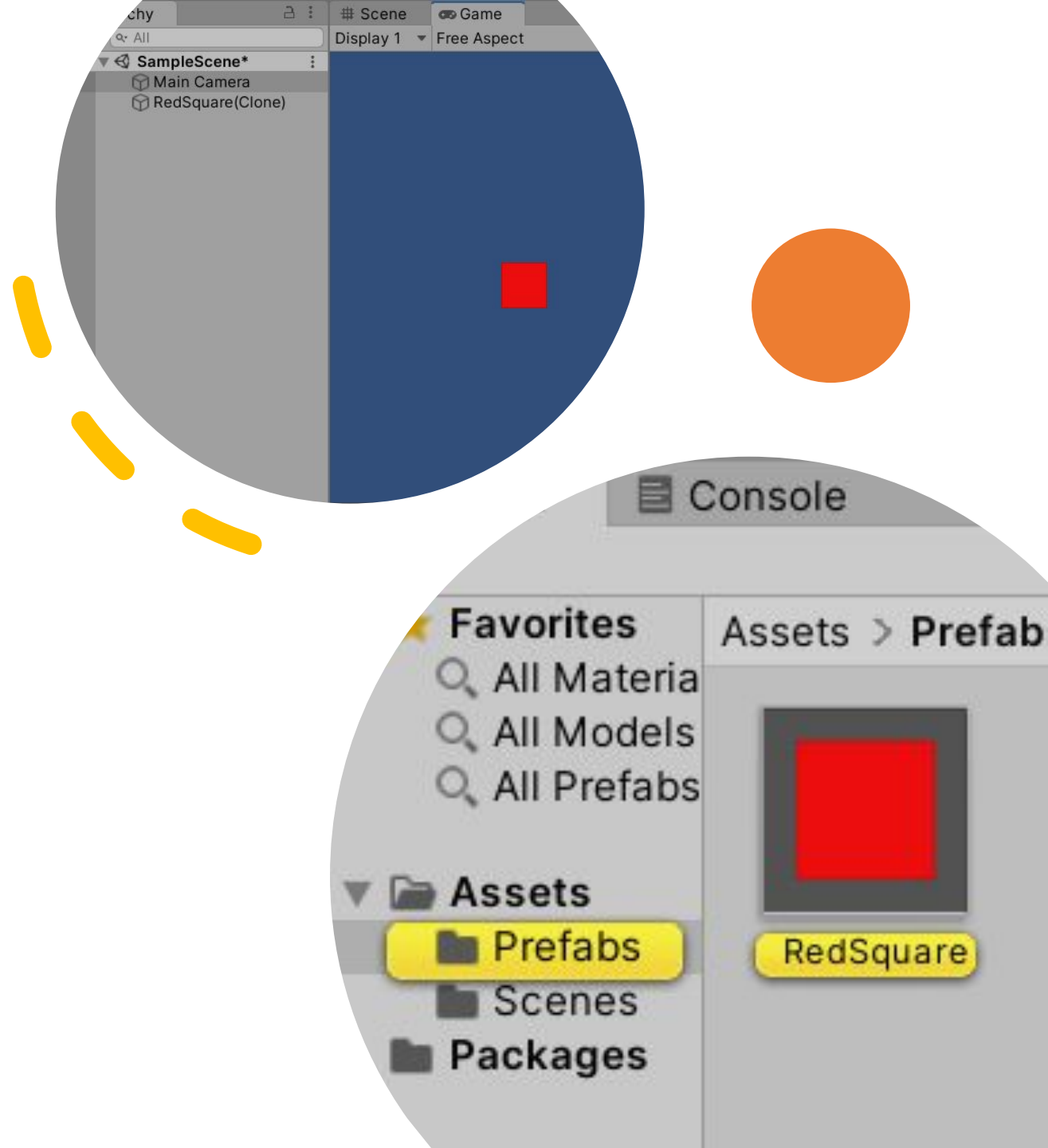
В свойствах камеры (Inspector)



- Появился компонент скрипта. Более того, в этом компоненте появилось поле по названию похожее на переменную в скрипте newSquare. Как ни странно, это и есть данная переменная. Привыкайте, что Inspector вставляет пробелы и начинает названия со строчной буквы.
- Данной переменной пока ничего не присвоено. На это намекает надпись None (Game Object).
 - Нажимаем на «мишень» справа
 - В появившемся диалоге переходим на вкладку Assets
 - Дважды кликаем на RedSquare

Результат манипуляций

- При выделении RedSquare в Inspector объекта Main Camera, он подсвечивается желтым в Project
- После запуска игрового режима
 - В окне визуализации появляется падающий квадрат
 - В Hierarchy возникает объект RedSquare(Clone)

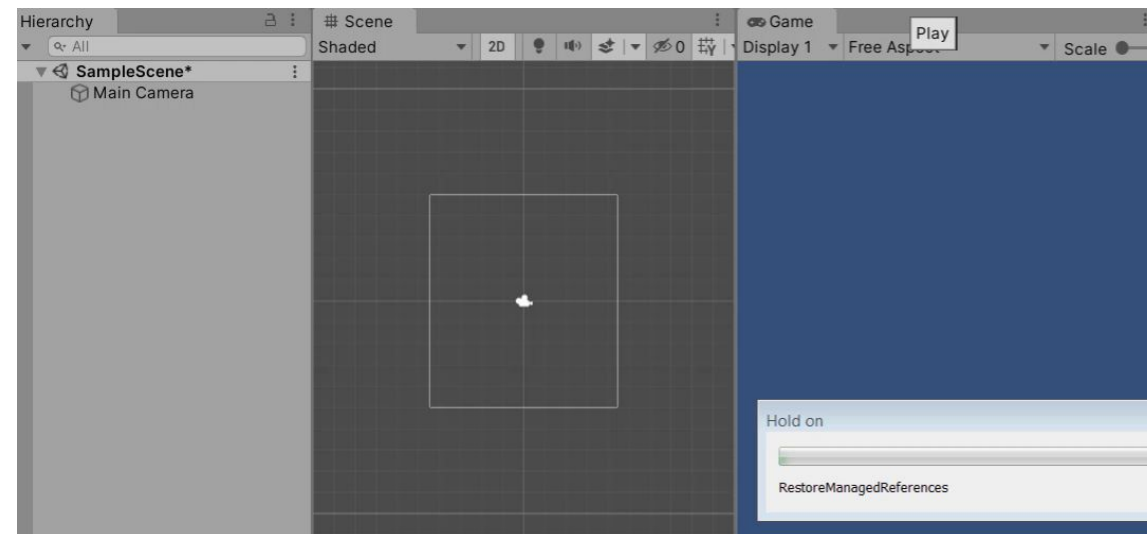


Теперь похулиганим))

В скрипте вызовем метод `Instantiate` не только при старте, но и при обновлении

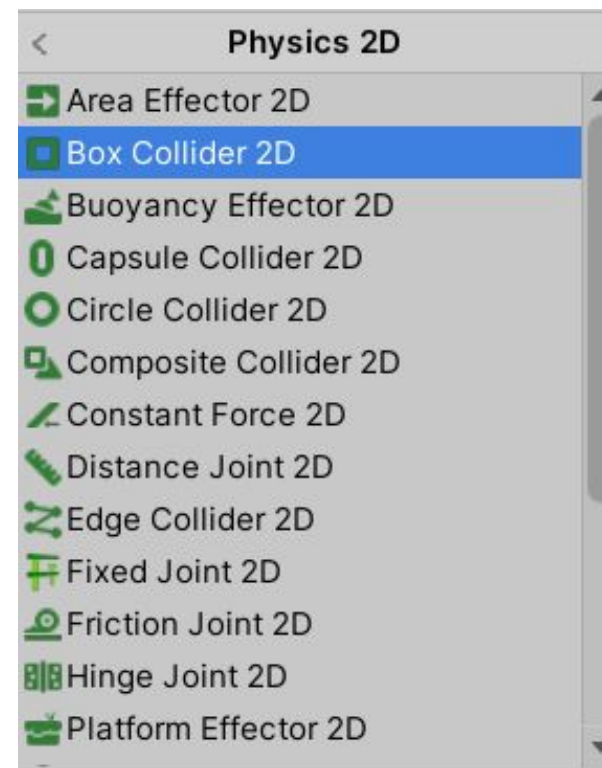
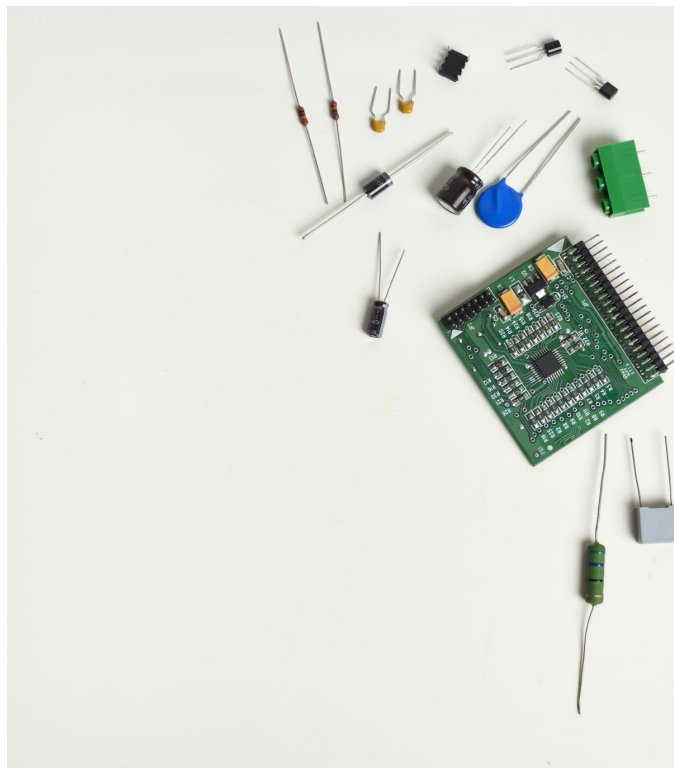
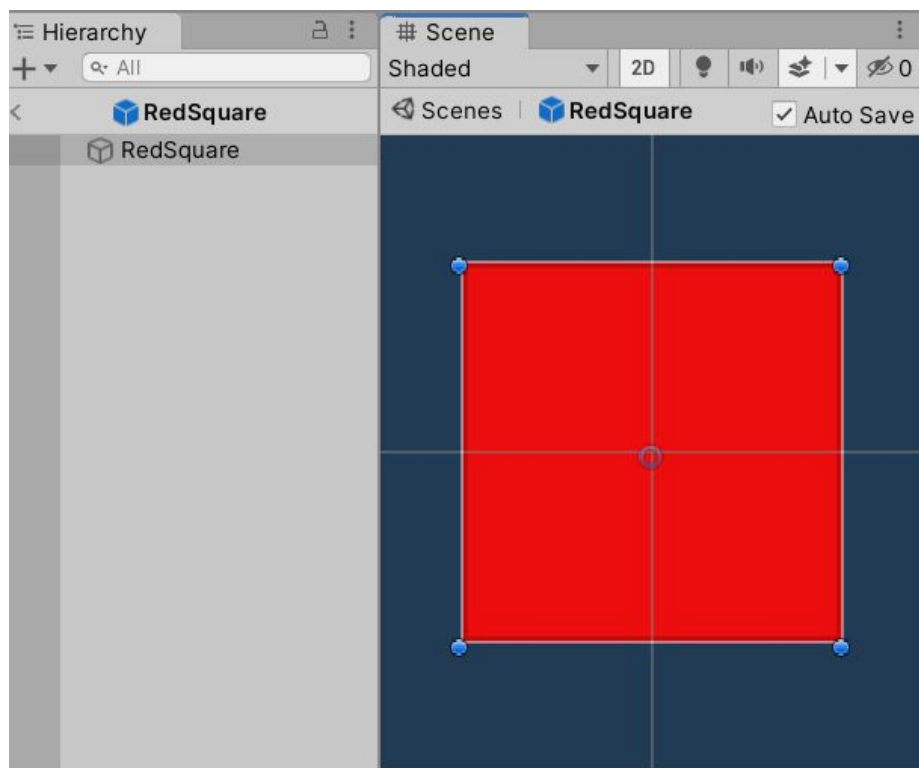
```
Unity Script | 0 references
public class SquareSpawner : MonoBehaviour
{
    public GameObject newSquire;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        Instantiate(newSquire);
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        Instantiate(newSquire);
    }
}
```



Добавим компонент Collider

- В Projects, в папке Assets, подкаталоге Prefabs дважды кликнем на RedSquare
- В Hierarchy вместо объектов сцены откроется наш prefab, в инспекторе покажутся его свойства
- Добавляем компонент Add Component -> Physics2D -> Box Collider 2D.

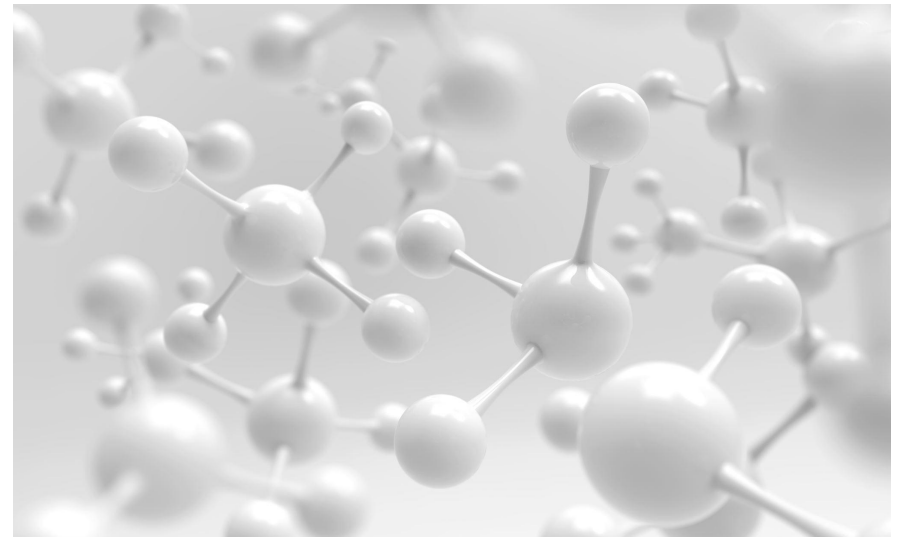
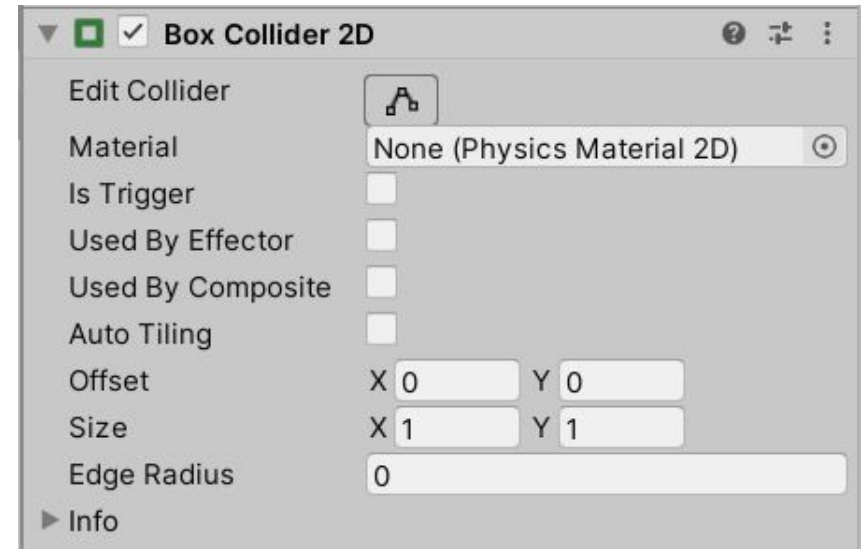


Что такое Collider

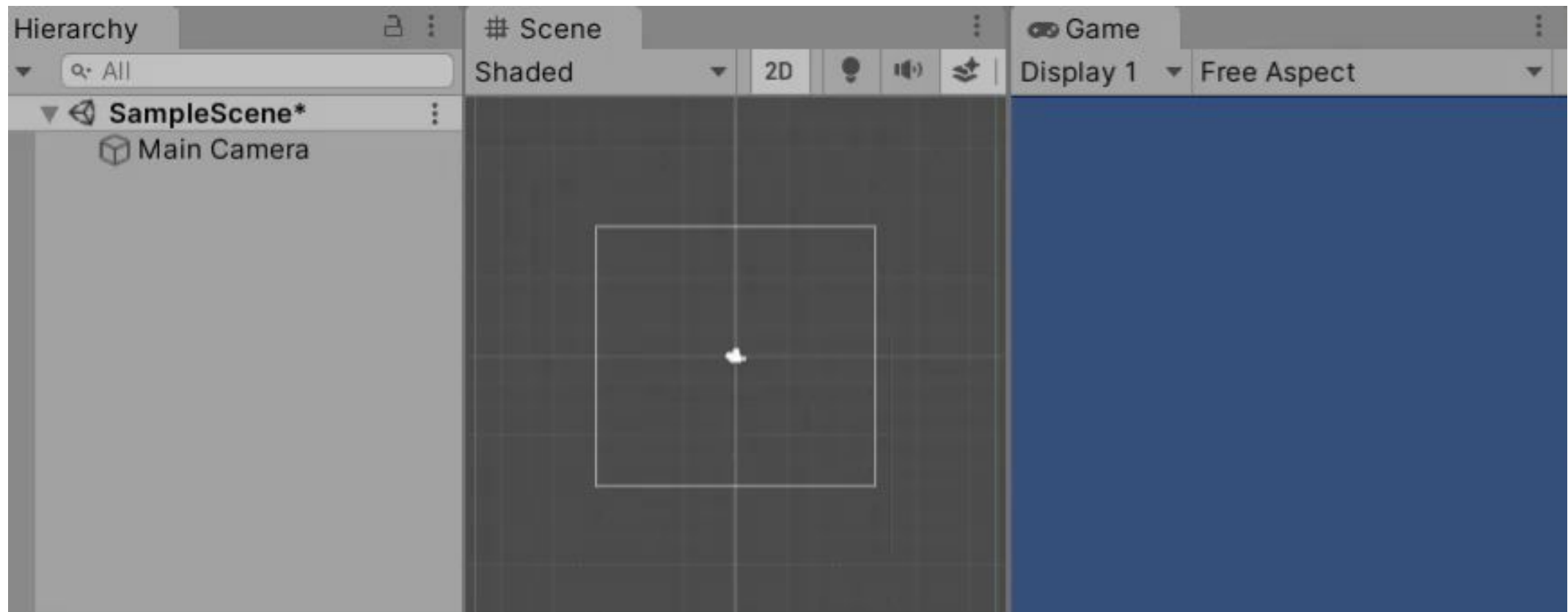
Компоненты Collider 2D определяют форму 2D-объекта GameObject **для учета физических столкновений**. Невидимый коллайдер не обязательно должен иметь ту же форму, что и визуальный GameObject; на самом деле, грубое приближение часто более эффективно и неотличимо в игровом процессе.

Все коллайдеры для 2D-объектов GameObject имеют имена, оканчивающиеся на «2D». Коллайдер, в названии которого нет «2D», предназначен для использования с 3D GameObject. Обратите внимание, что нельзя смешивать 3D-объекты GameObject и 2D-коллайдеры или 2D-объекты GameObject и 3D-коллайдеры.

<https://docs.unity3d.com/ru/current/Manual/Collider2D.html>



Результат игры с Collider 2D



Добавим цвета в скрипт.

Создадим новый скрипт. Назовем его DifferentColors.

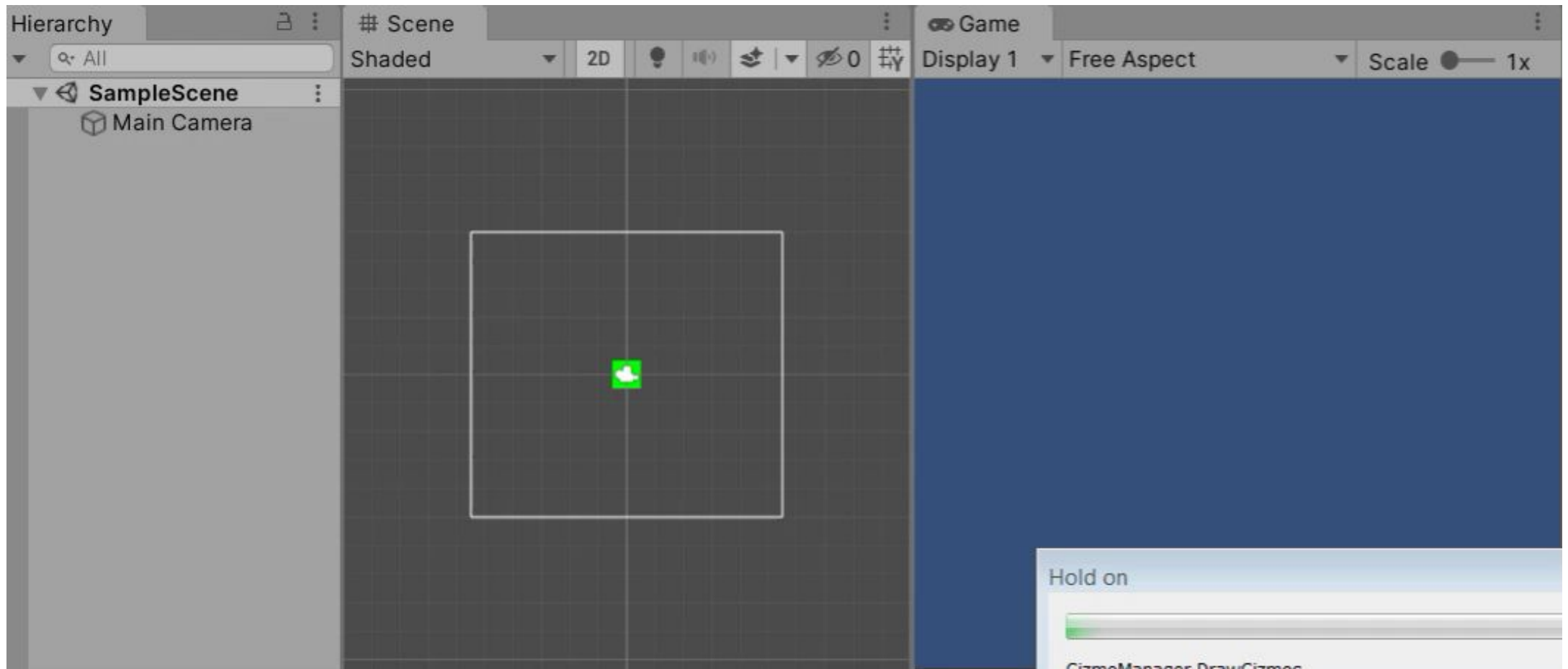
Здесь использован обобщенный вариант очень хорошего метода GetComponent<T>. Данный метод возвращает нужный компонент (определяется T) вызвавшего его игрового объекта.

<https://docs.unity3d.com/2020.2/Documentation/ScriptReference/GameObject.GetComponent.html>

```
Unity Script | 0 references
public class DifferentColors : MonoBehaviour
{
    public GameObject newSquare;
    Color[] colors = new Color[] {Color.red,Color.green,Color.blue,
    Color.cyan,Color.magenta,Color.yellow};
    int count;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        count = 0;
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        GameObject squareClone = Instantiate<GameObject>(newSquare);
        SpriteRenderer sp = squareClone.GetComponent<SpriteRenderer>();
        sp.color = colors[count];
        count = (++count) % (colors.Length);
    }
}
```

Цветные квадратики!



Создание примитивной игры.

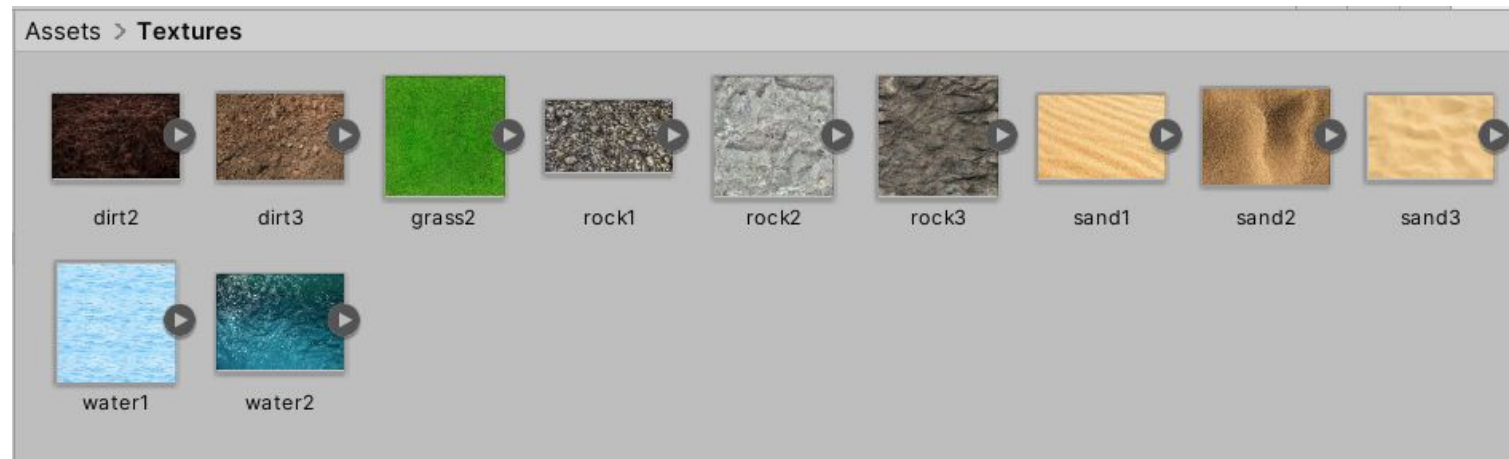
- Игра будет содержать
 - Поле (земля, трава, песок – что-то по чему можно ездить)
 - Танк, который будет передвигаться по полю, реагируя на ввод игрока
 - Заправки с топливом, пройдя через которые танк может заправляться (ведется учет топлива, танк без топлива не едет)
 - Монетки (сундуки, грибы, птички), которые можно собирать и зарабатывать очки
- Цель игры
 - Заработать наибольшее количество очков



Создание поля. Шаг 1.

dirt2.jpg
dirt3.jpg
grass2.jpg
rock1.jpg
rock2.jpg
rock3.jpg
sand1.jpg
sand2.jpg
sand3.jpg
water1.jpg
water2.jpg

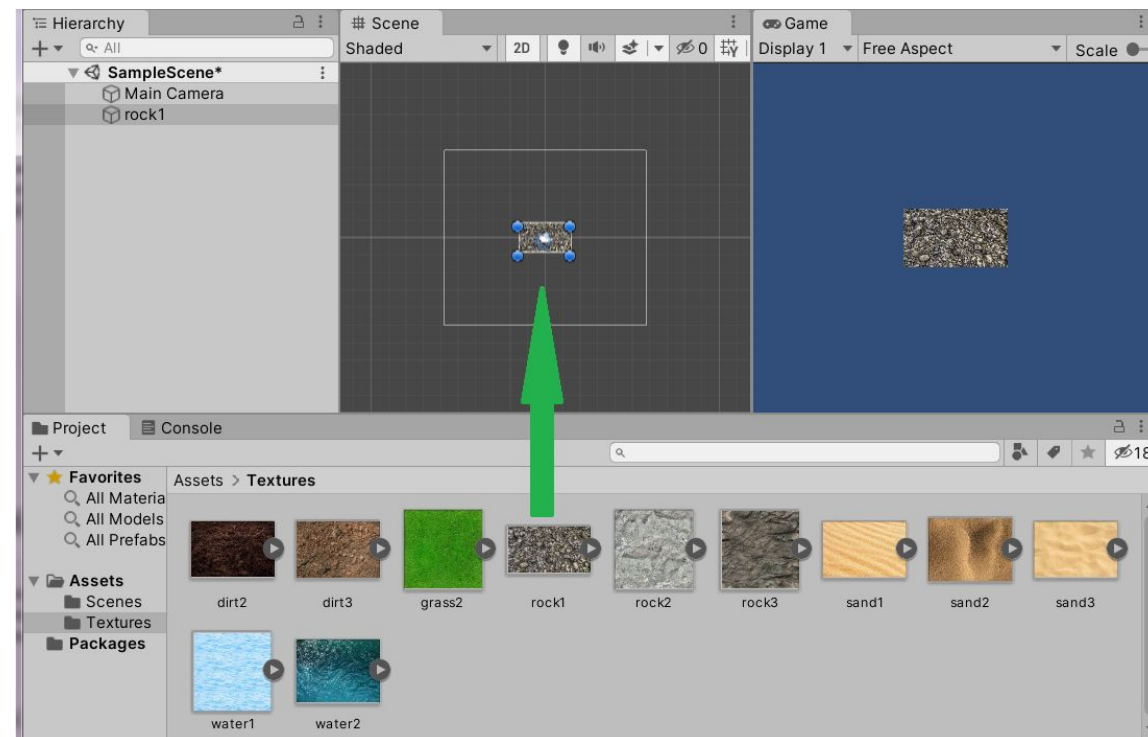
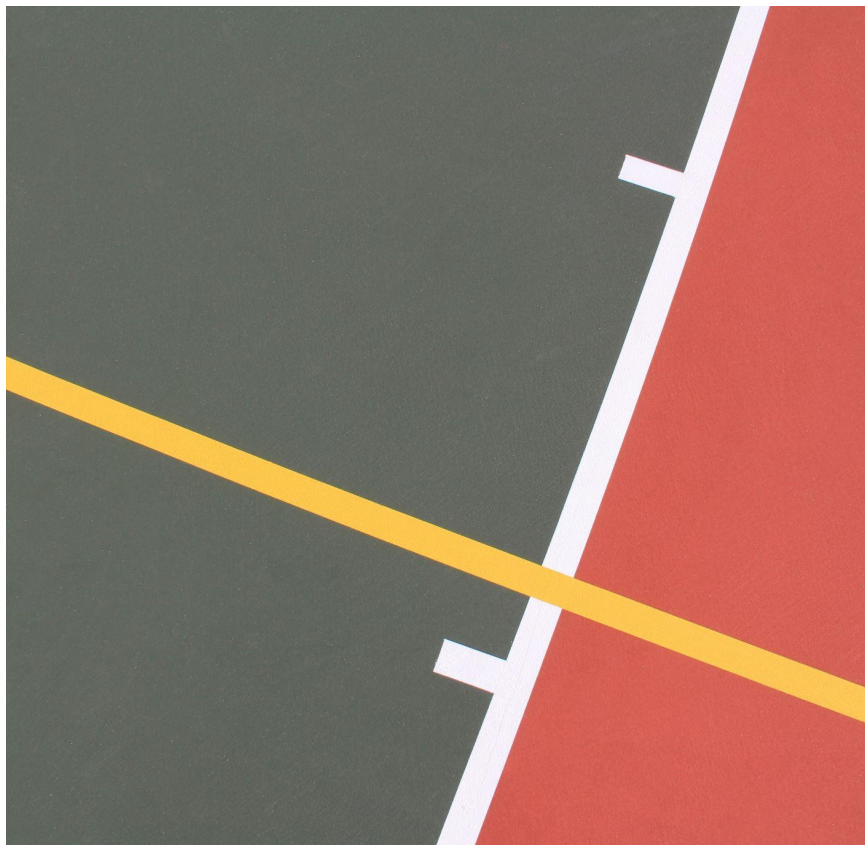
- Скачиваем на бесплатных сайтах бесшовные текстуры земли, песка, травы, того, что больше нравится. Вставляем в папку Assets проекта (я предпочитаю создавать под все разные подкаталоги, в данном случае – Textures). Вставить можно как в unity, так и в проводнике.

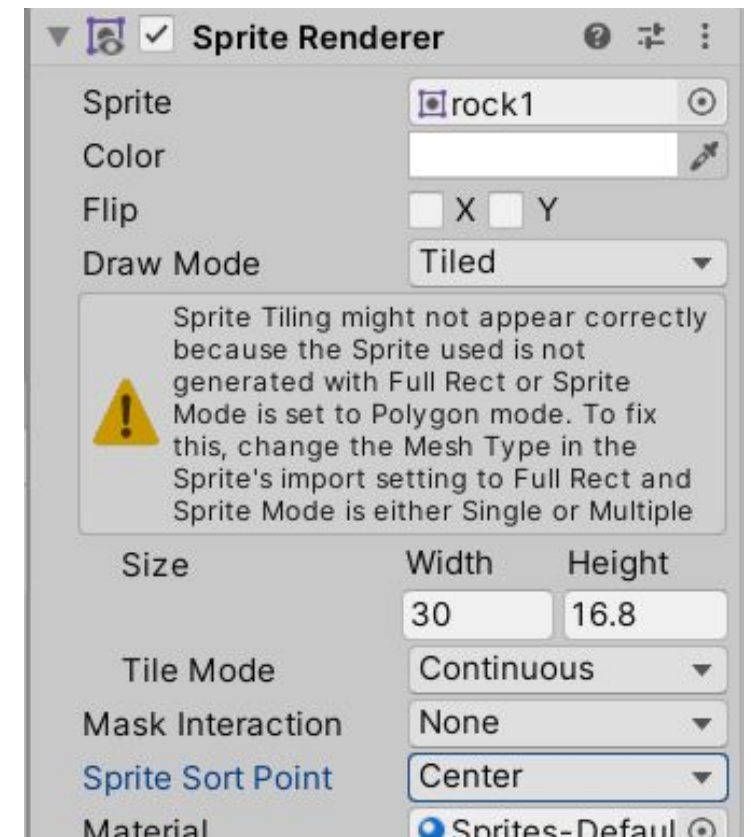
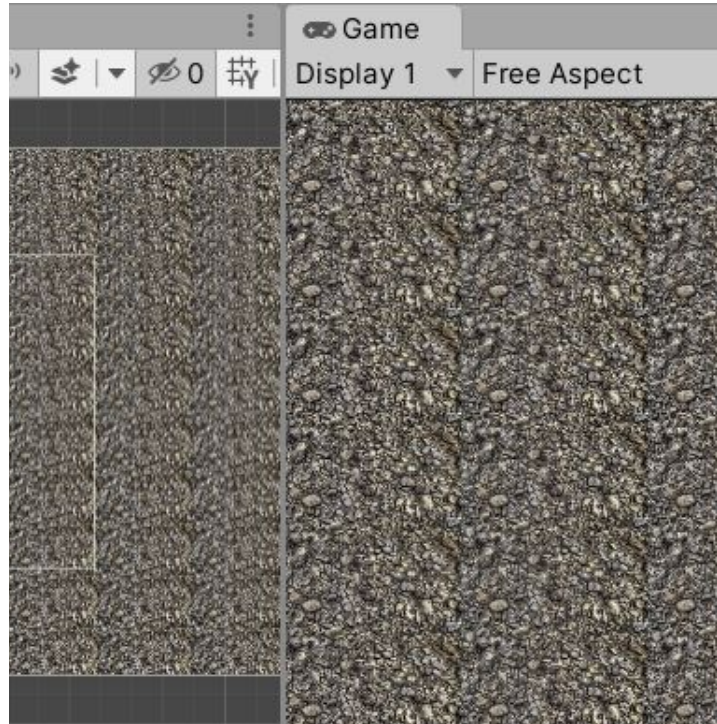
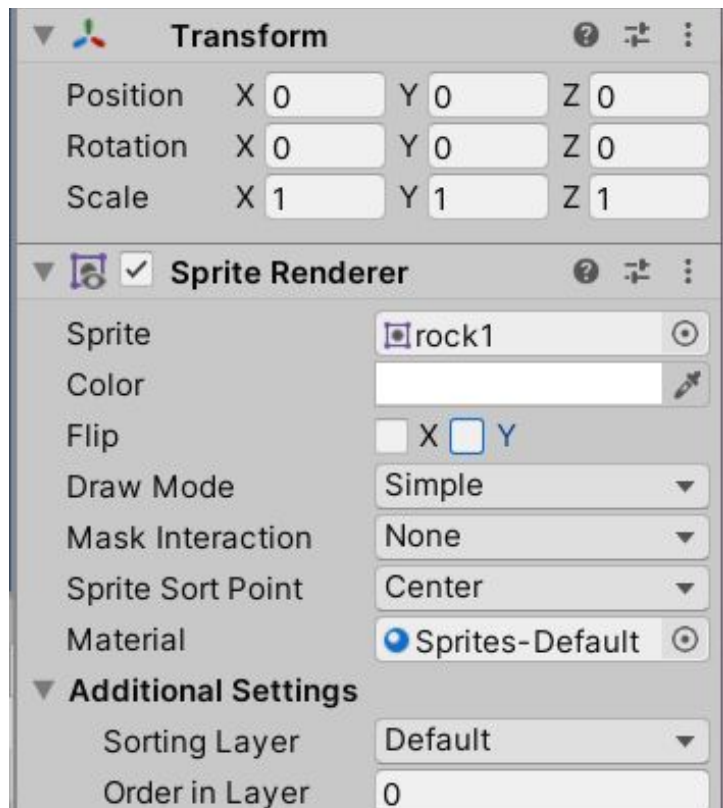


dirt2.jpg
dirt2.jpg.meta
dirt3.jpg
dirt3.jpg.meta
grass2.jpg
grass2.jpg.meta
rock1.jpg
rock1.jpg.meta
rock2.jpg
rock2.jpg.meta
rock3.jpg
rock3.jpg.meta
sand1.jpg
sand1.jpg.meta
sand2.jpg
sand2.jpg.meta
sand3.jpg
sand3.jpg.meta
water1.jpg
water1.jpg.meta
water2.jpg
water2.jpg.meta

Создание поля. Шаг 2.

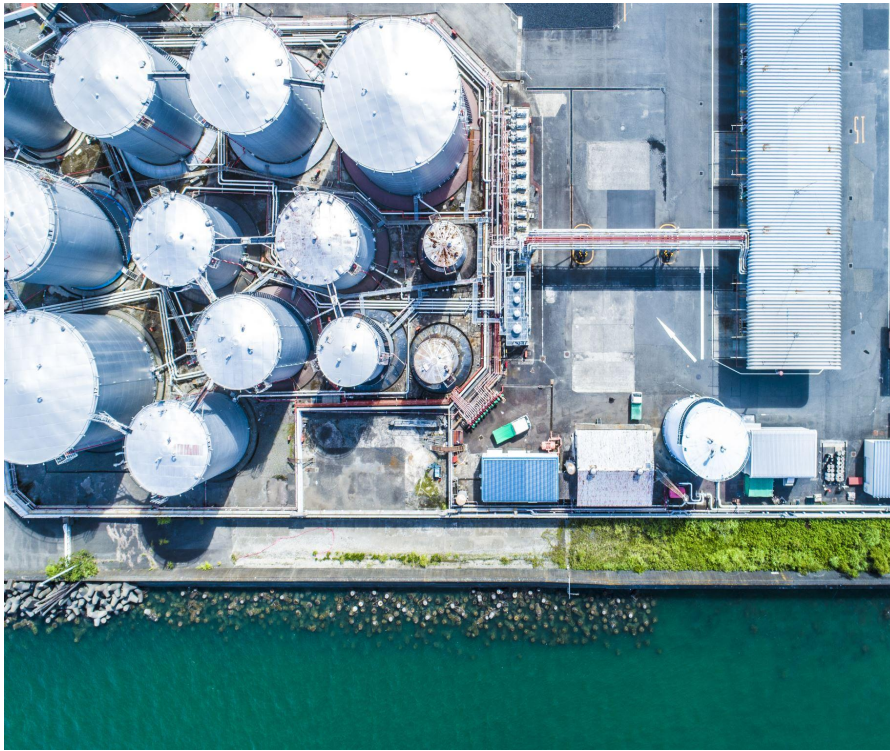
Простым перетаскиванием размещаем выбранную текстуру на сцену. В Hierarchy сразу появляется ее имя, а в Инспекторе – свойства.





Создание поля. Шаг 3.

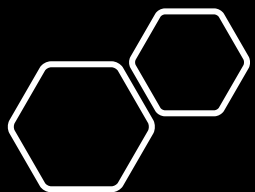
Кусочек на сцене слишком мал, надо бы его «замостить» на большую площадь. Для этого в Draw Mode выбираем Tiled



Создание танка.

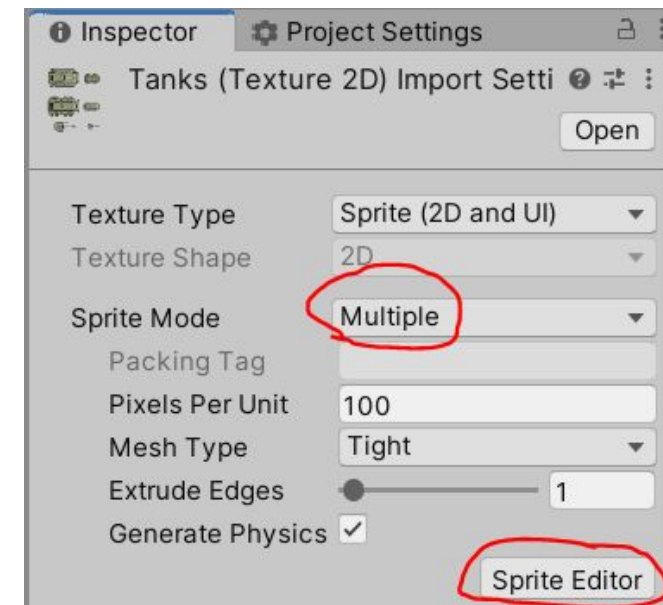
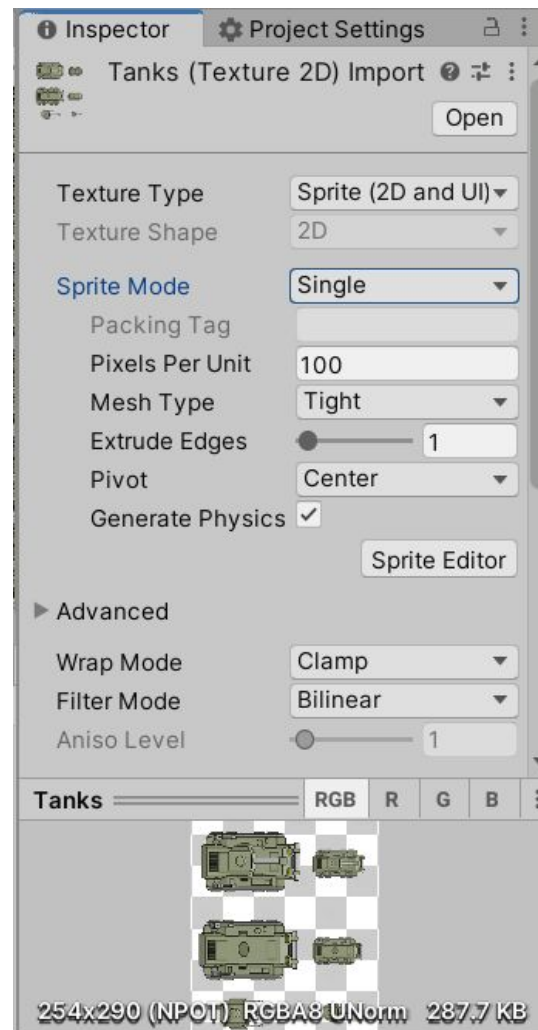
И опять, качаем картинки танков, вид сверху. (Заодно уж и картинку топлива)

Обратите внимание, что на данной картинке несколько танков, плюс есть «разобранные» варианты. То есть до использования картинки надо с ней поработать.



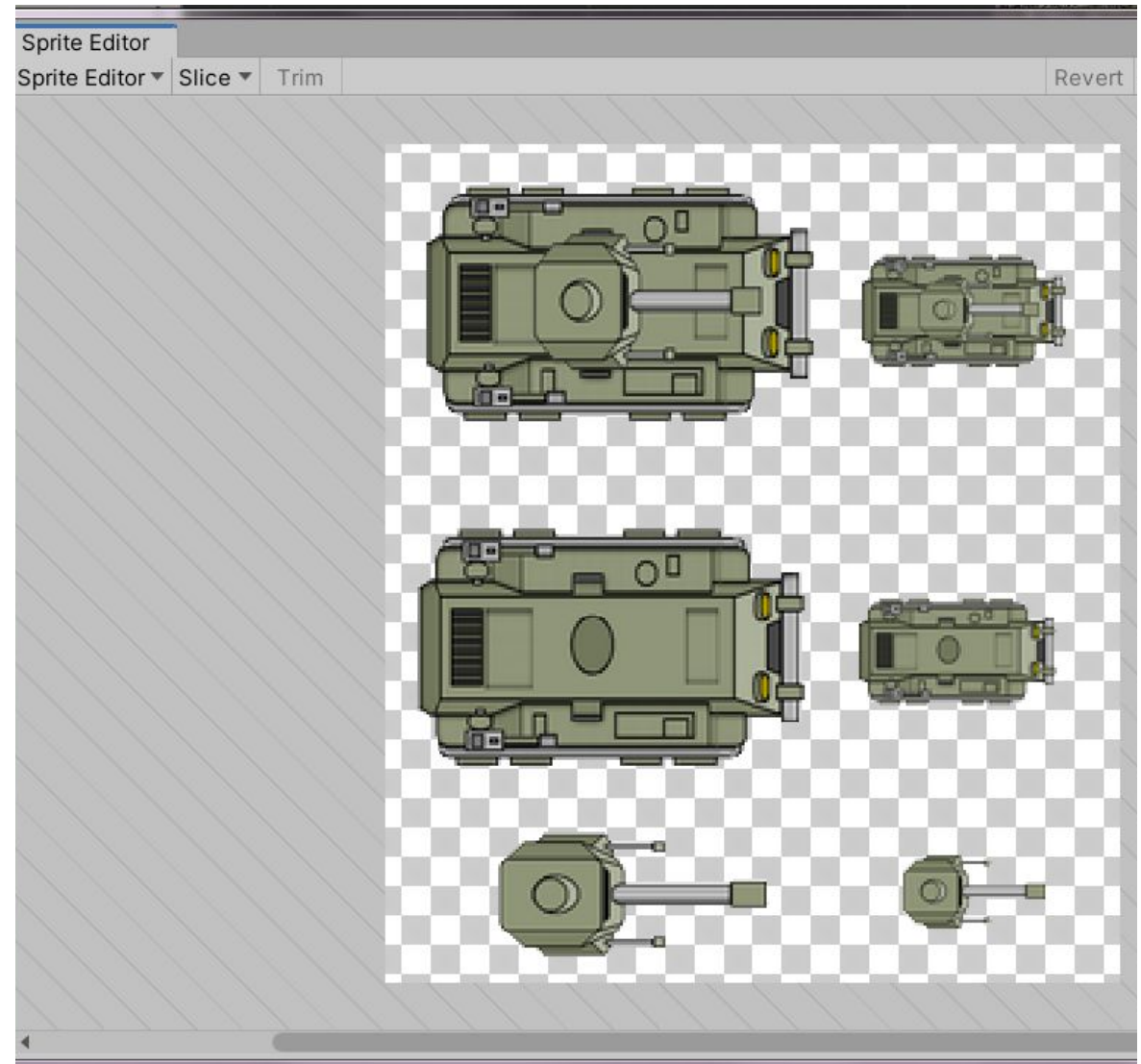
Разбор картинки на составляющие

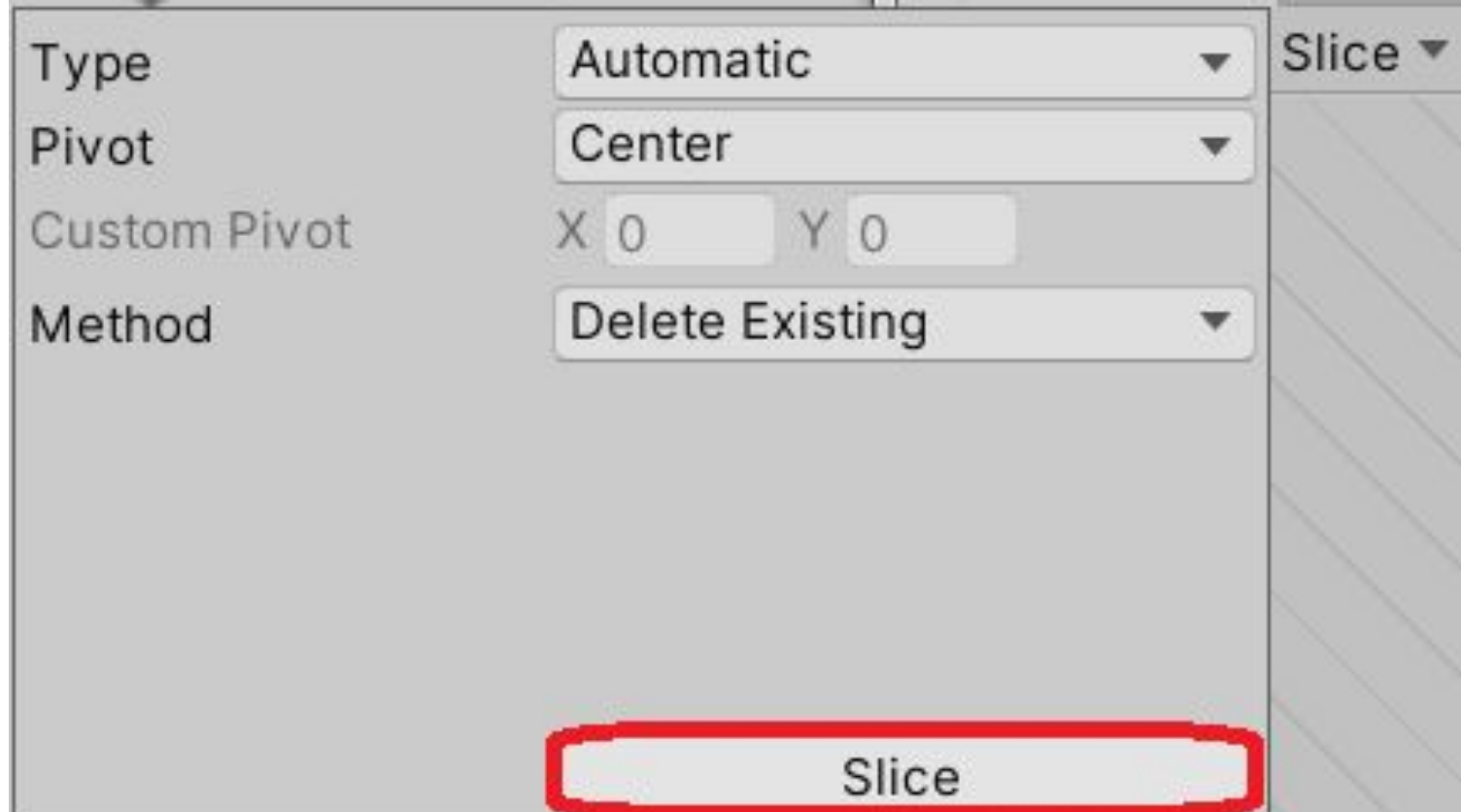
- Для этого нужно
- Выделить ее в папке Assets. В инспекторе появятся свойства.
- Установить Sprite Mode в позицию Multiple
- Перейти в Sprite Editor



Sprite Editor

- Иногда текстура спрайта содержит только один элемент графики, но часто гораздо удобней объединить несколько изображений связанных друг с другом в одно изображение. Например, изображение может содержать составные части персонажа, как пушка танка, которая может двигаться независимо от корпуса. Для этих целей Unity предоставляет **Sprite Editor** позволяя с легкостью извлекать элементы составного изображения.
- <https://docs.unity3d.com/ru/2021.1/Manual/SpriteEditor.html>
- <https://ios-apps.ru/blog/sprite-editor-i-rabota-s-nim/>

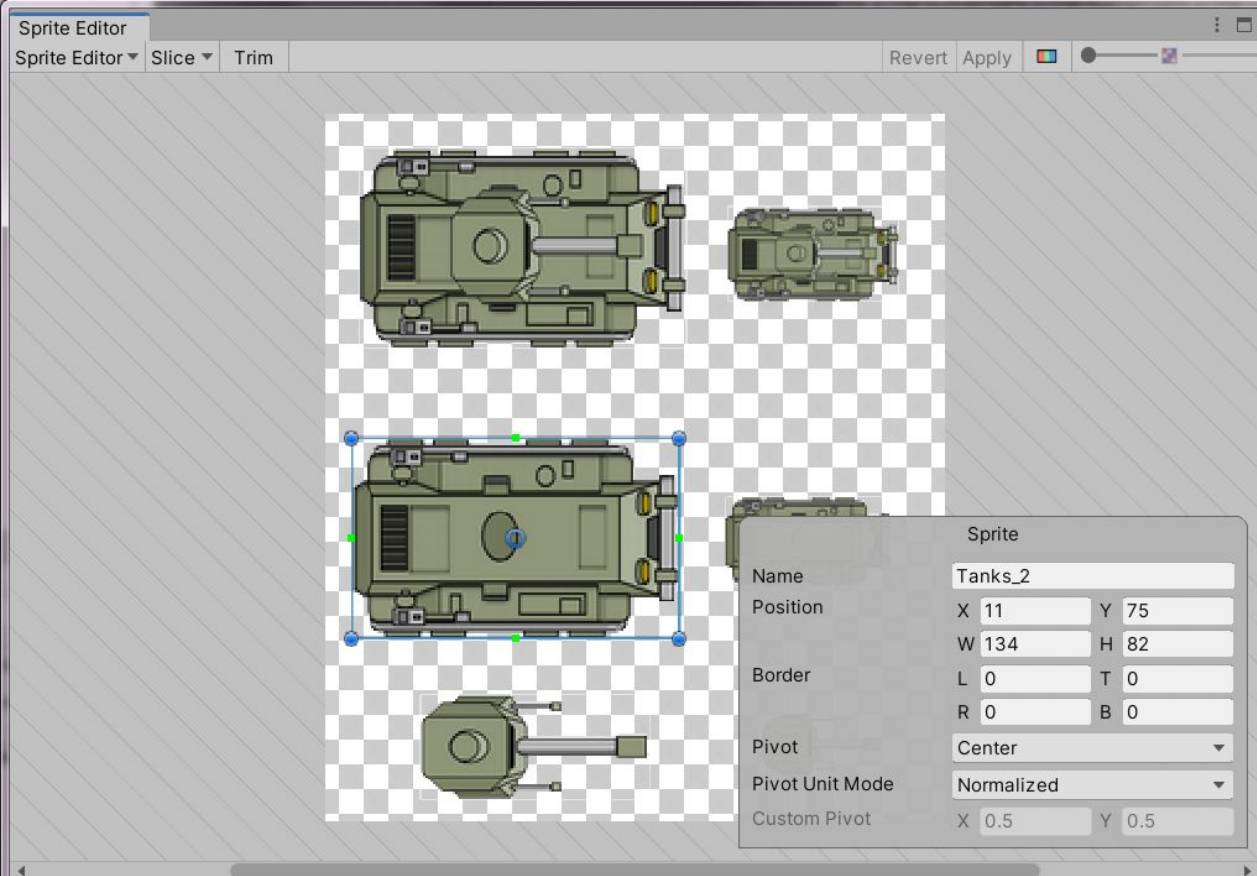




Slice automatically

Разрежем наш танк на части. Для этого в выпадающем меню Slice оставим все установки по умолчанию и нажмем собственно кнопку Slice. В правом углу Sprite редактора станут доступными кнопки Revert и Apply. Жмем Apply.





Подгонка результата вручную

После нажатия Apply можно будет вручную подогнать размеры каждого объекта

Работа с отдельным танком.

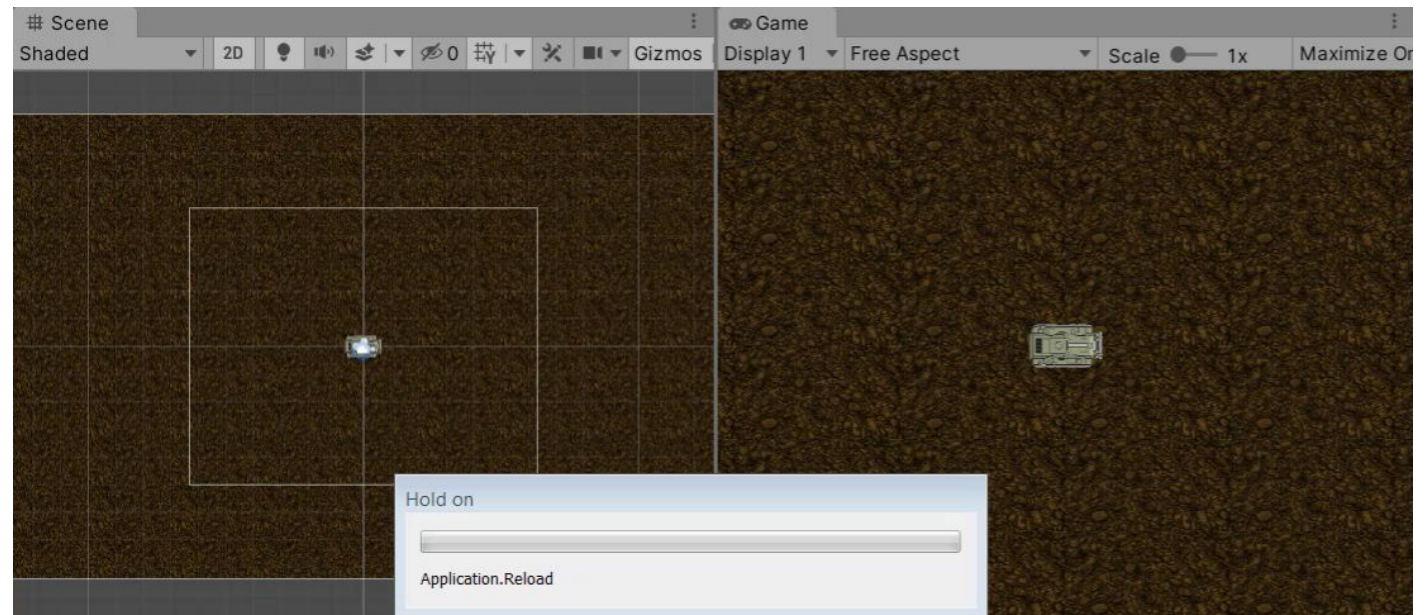
- Теперь можно перетащить выбранный так на сцену. В Assets/Sprite танки и их пушки теперь расположены отдельно. Для данной программы я выберу целиковый большой танк.
- Когда поместила серый танк на серые камни, поняла, что сделала что-то не то и изменила цвет камней в Sprite Renderer, сделала их больше похожими на землю.
- На танк навесила Box Collider.



Движение танка

- Добавляем скрипт TankBehaviourScript и навешиваем его на танк.
- В скрипте создаем метод TanksMoving и вызываем его в Update()
- Класс Input позволяет отследить ввод пользователя с клавиатуры, мыши, других устройств ввода.
- <https://docs.unity3d.com/Manual/Input.html>

```
// Update is called once per frame
Unity Message | 0 references
void Update()
{
    TankMoving();
}
1 reference
private void TankMoving()
{
    float step = 0.05f;
    if (Input.GetKey(KeyCode.UpArrow))
        transform.position = transform.position + new Vector3(0, step, 0);
    else if (Input.GetKey(KeyCode.RightArrow))
        transform.position = transform.position + new Vector3(step, 0, 0);
    else if (Input.GetKey(KeyCode.DownArrow))
        transform.position = transform.position + new Vector3(0, -step, 0);
    else if (Input.GetKey(KeyCode.LeftArrow))
        transform.position = transform.position + new Vector3(-step, 0, 0);
}
```



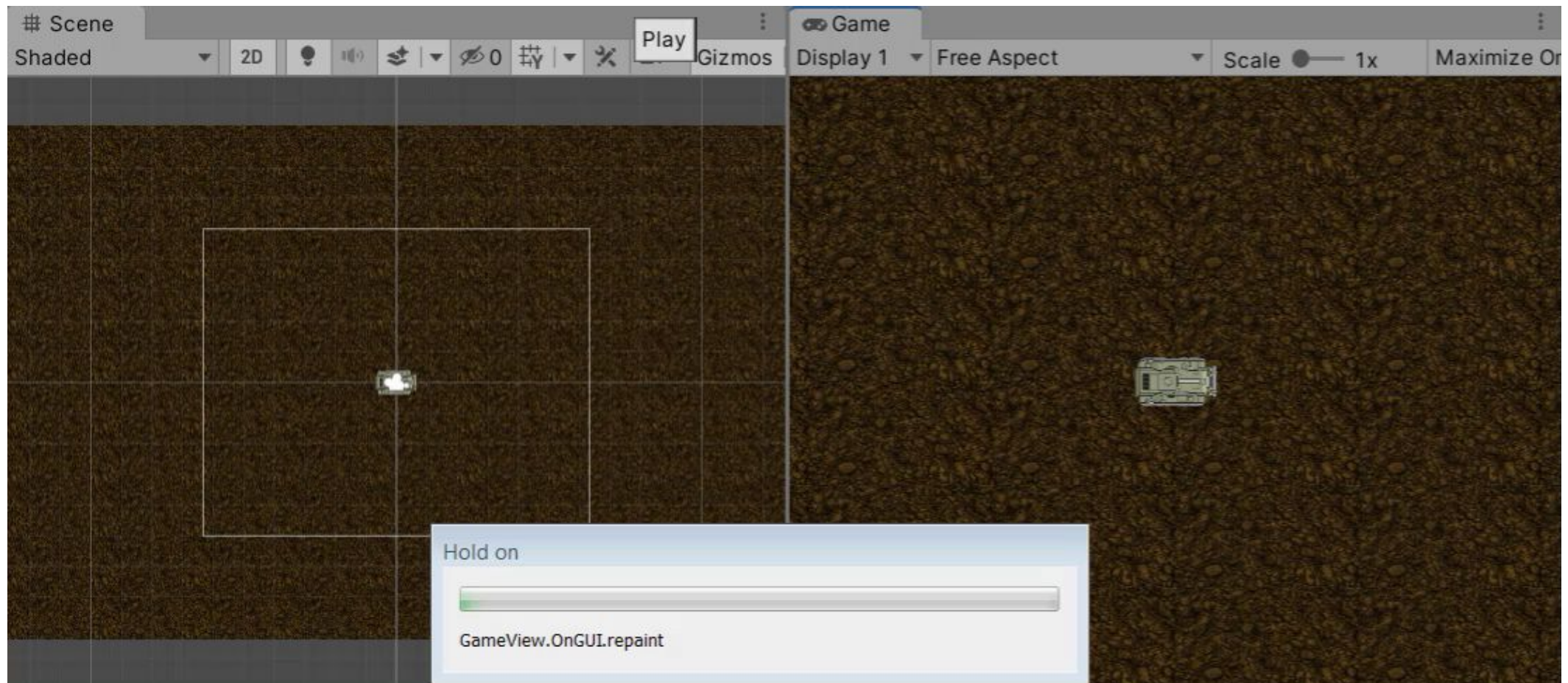
Поворот танка

```
private void TankMoving()
{
    float step = 0.05f;
    if (Input.GetKey(KeyCode.UpArrow))
        if (rotationStatusOfTank == RotationStatus.Up)
            { transform.position = transform.position + new Vector3(0, step, 0); }
        else { transform.rotation = Quaternion.Euler(0, 0, 90); rotationStatusOfTank = RotationStatus.Up; }
    else if (Input.GetKey(KeyCode.RightArrow))
        if (rotationStatusOfTank == RotationStatus.Right)
            { transform.position = transform.position + new Vector3(step, 0, 0); }
        else { transform.rotation = Quaternion.Euler(0, 0, 0); rotationStatusOfTank = RotationStatus.Right; }
    else if (Input.GetKey(KeyCode.DownArrow))
        if (rotationStatusOfTank == RotationStatus.Down)
            { transform.position = transform.position + new Vector3(0, -step, 0); }
        else { transform.rotation = Quaternion.Euler(0, 0, -90); rotationStatusOfTank = RotationStatus.Down; }
    else if (Input.GetKey(KeyCode.LeftArrow))
        if (rotationStatusOfTank == RotationStatus.Left)
            { transform.position = transform.position + new Vector3(-step, 0, 0); }
        else { transform.rotation = Quaternion.Euler(0, 0, 180); rotationStatusOfTank = RotationStatus.Left; }
}
```

```
public enum RotationStatus
{
    Right = 0, Up = 90, Left = 180, Down = 270
}
// Unity Script | 0 references
public class TankBehaviourScript : MonoBehaviour
{
    // Переменная, следящая за поворотом танка
    RotationStatus rotationStatusOfTank = RotationStatus.Right;
```

Поворот будем осуществлять при нажатии тех же клавиш. Если в данный момент танк развернут не направо, а игрок нажимает стрелку вправо, то танк сначала разворачивается, если он уже развернут в нужную сторону, то едет.

Игра с поворотом танка.



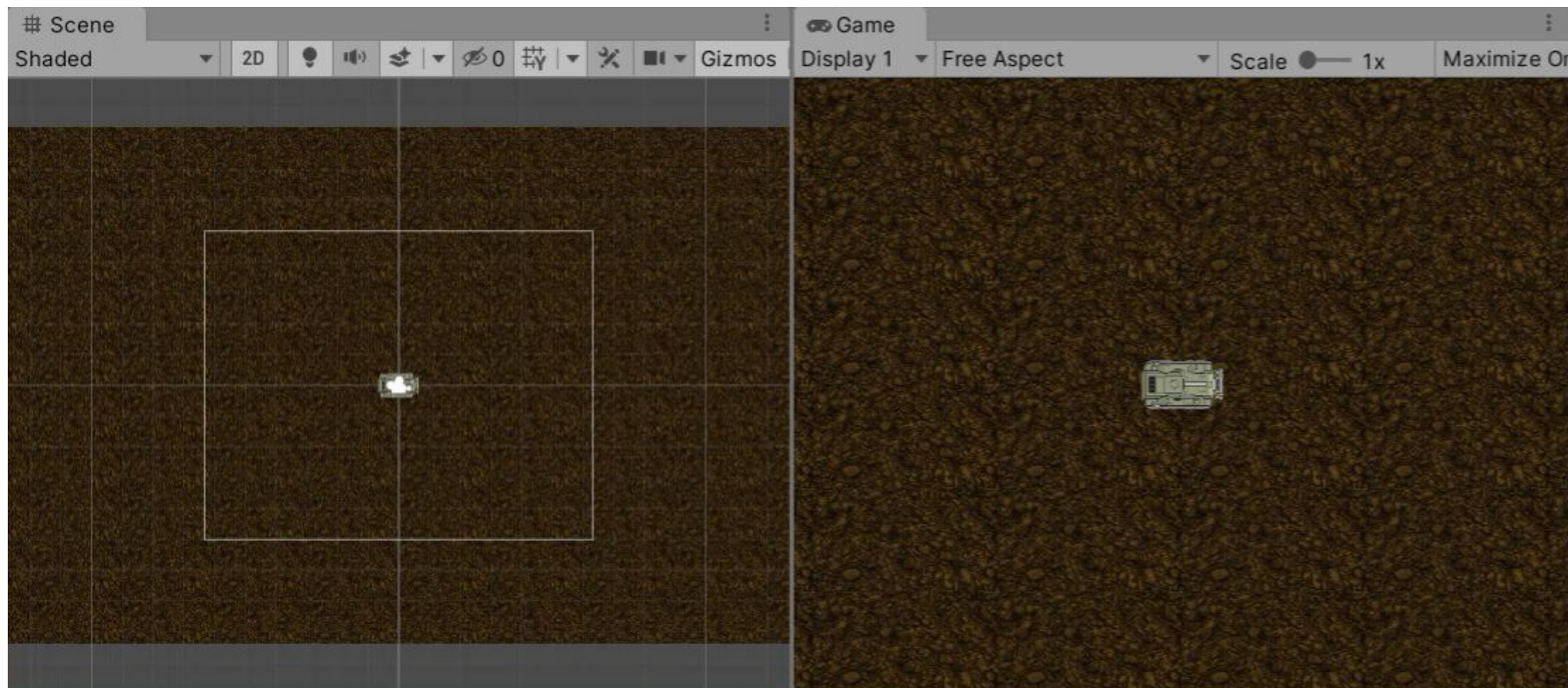
Плавный поворот

- Если я хочу, чтобы танк поворачивался более плавно, то стоит использовать сопрограмму.
- Сопрограмма (Coroutine) - это функция, которая может приостановить свое выполнение (yield) до завершения данной инструкции YieldInstruction.
<https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Coroutine.html>

```
StartCoroutine(Rotation(rotationStatusOfTank, RotationStatus.Up));
```

```
IEnumerator Rotation(RotationStatus start, RotationStatus end)
{
    int angleStart = (int)start;
    int angleEnd = (int)end;
    int anglePlus;
    if (Mathf.Abs(angleEnd - angleStart) <= 180)
        if (angleEnd < angleStart) anglePlus = -5;
        else anglePlus = 5;
    else
    {
        if (angleEnd < angleStart)
        { anglePlus = 5; angleEnd += 360; }
        else
        { anglePlus = -5; angleEnd -= 360; }
    }
    int angle = angleStart;
    while (angle != angleEnd)
    {
        transform.RotateAround(transform.position,
                                new Vector3(0, 0, 1), anglePlus);
        angle += anglePlus;
        yield return null;
    }
}
```


Игра с плавным поворотом танка.



Движение камеры за объектом

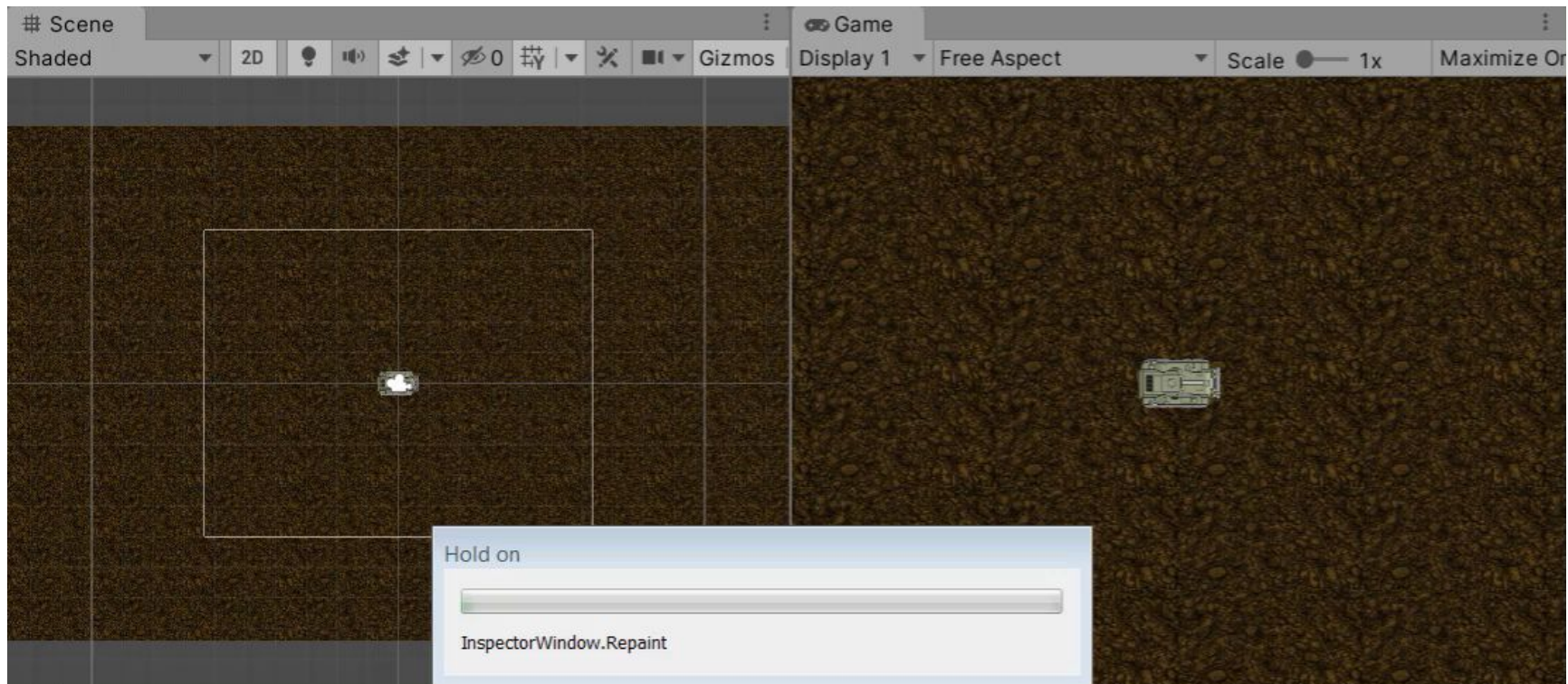
- Чтобы не терять танк из виду напишем скрипт, перемещающий камеру за ним. И не забудем в редакторе установить значение открытой переменной Tank Transform на наш танк.

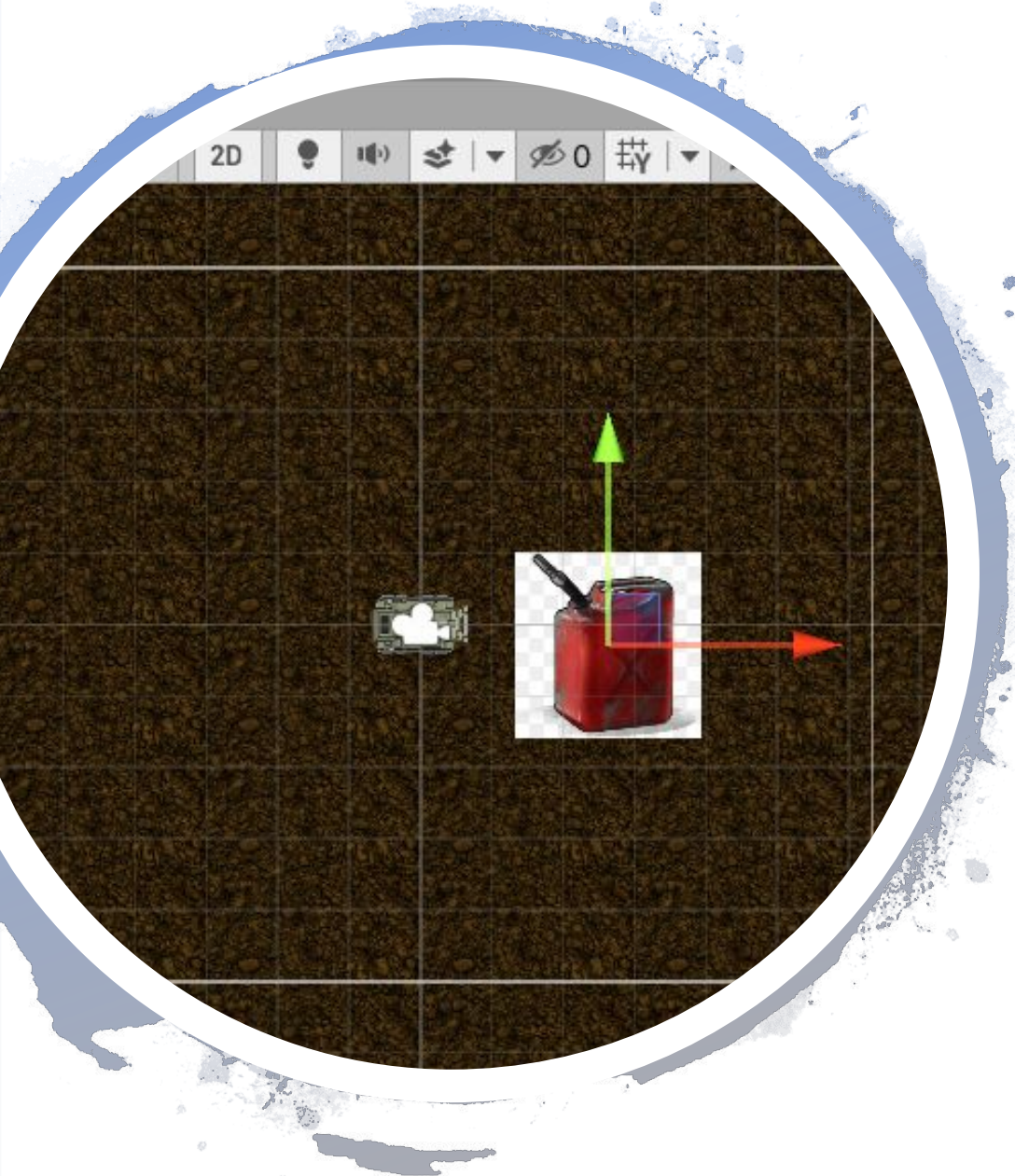
```
public class CameraFollow : MonoBehaviour
{
    public Transform tankTransform;

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        transform.position = new Vector3(tankTransform.position.x,
            tankTransform.position.y, transform.position.z);
    }
}
```



Результат





Теперь, когда танк ездит, перейдем к горючему.

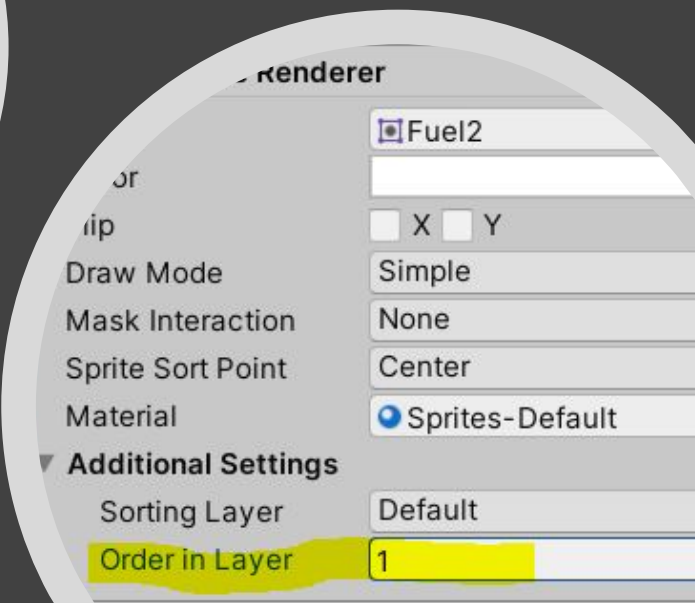
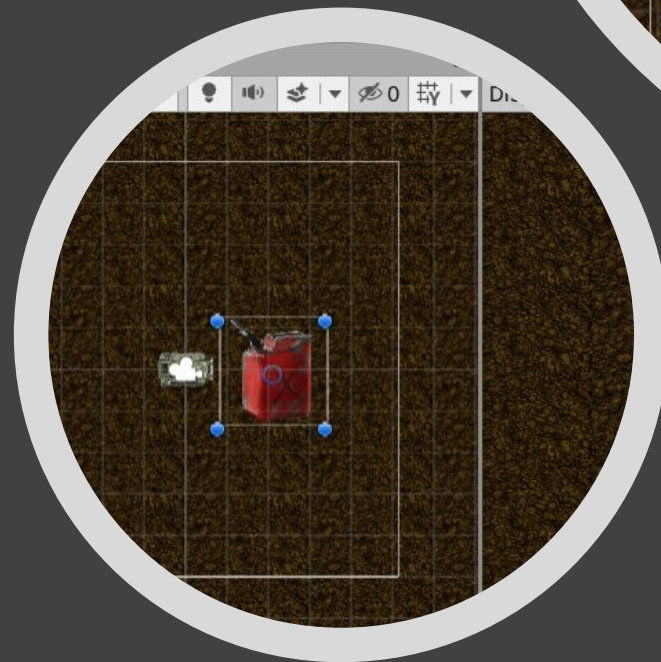
- Цистерны с горючим будут у нас случайно раскидываться по полю. Следовательно, начнем с создания prefab цистерны. Для этого перетащим картинку с цистерной из папки Assets на сцену (как проделывали это с танком). И тут меня (ну и Вас соответственно) поджидает первый сюрприз: картинка оказалась с фоном! Следовательно, опять идем в редактор спрайтов.

И опять
редактор
спрайтов



Еще одна попытка создать prefab

- Теперь второй сюрприз – за землей (раскрашенными камнями) не видно нашей цистерны! Дело в том, что по умолчанию, все объекты находятся на одном нулевом уровне, и как их unity разместит, неизвестно. Для контроля этого процесса надо задать объектам разные уровни. Делается это в инспекторе, в компоненте Sprite Renderer. Зададим Fuel уровень 1, танку – 2, а земле оставим 0.



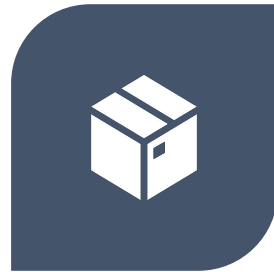
Доводим prefab до ума.



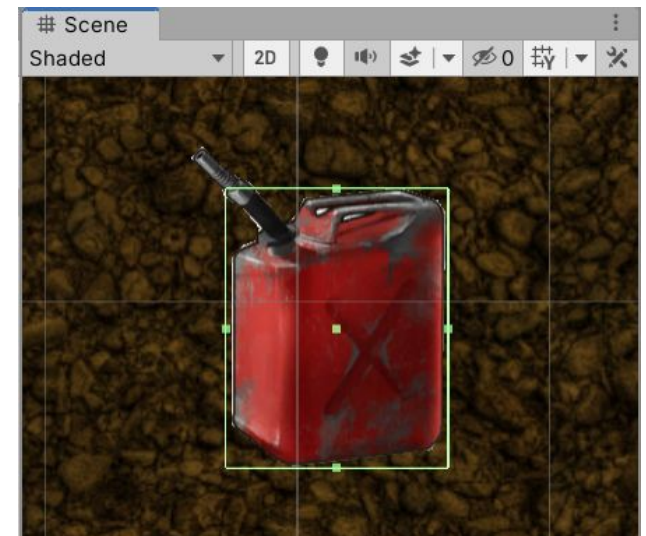
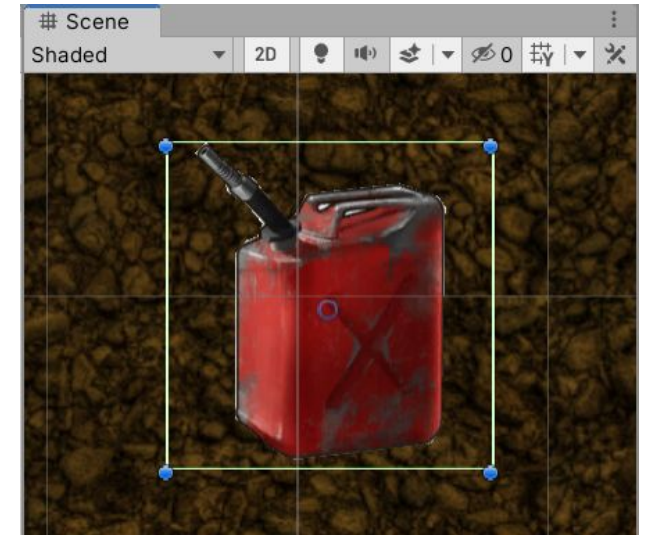
УМЕНЬШАЕМ
РАЗМЕРЫ (TRANSFORM
SCALE X=0.5, Y=0.5)



ДОБАВЛЯЕМ
КОМПОНЕНТ
Rigidbody, УБИРАЕМ
ГРАВИТАЦИЮ



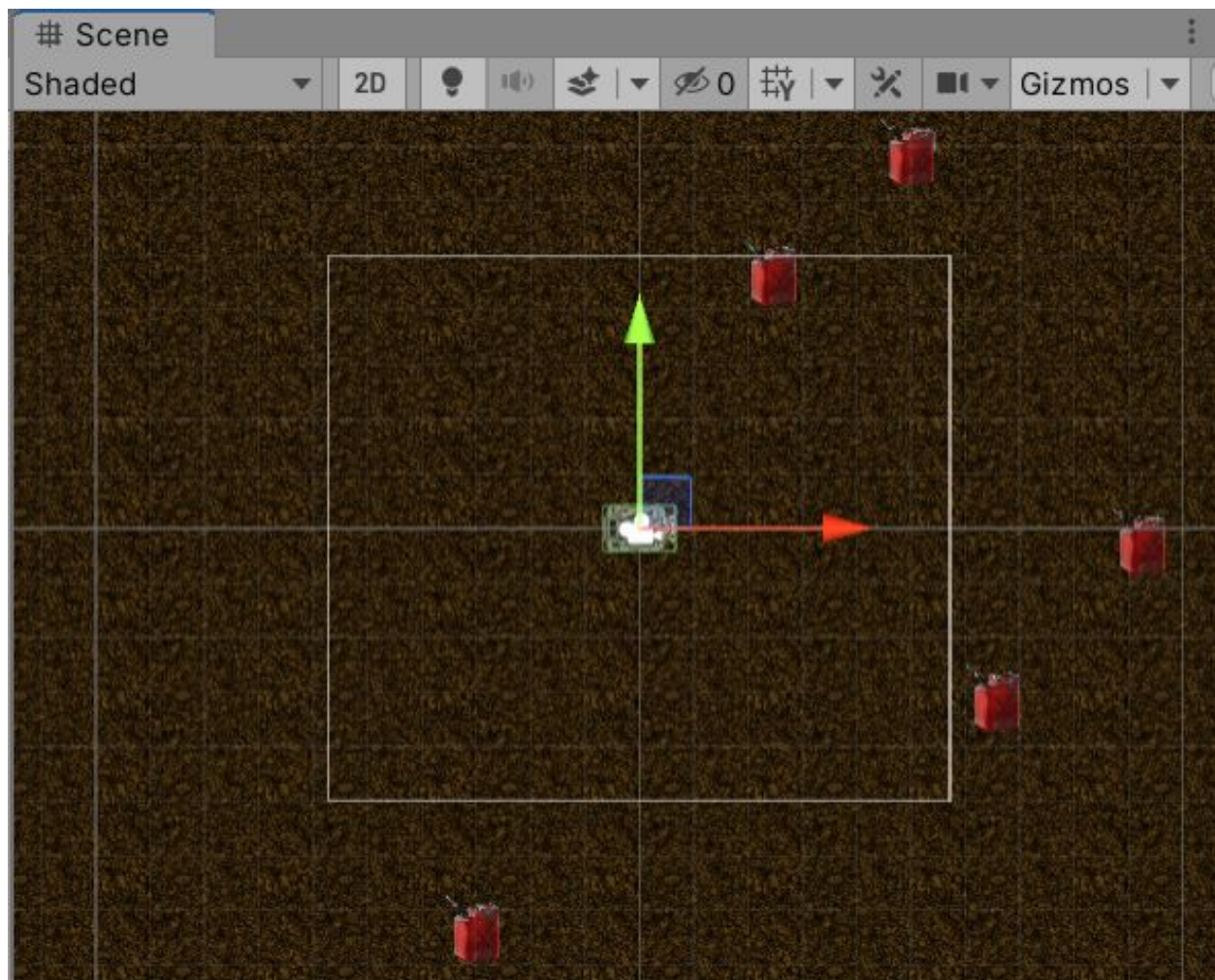
ДОБАВЛЯЕМ BOX
COLLIDER. ПОДГОНЯЕМ
РАЗМЕРЫ ВРУЧНУЮ.



Объект FuelManager.

- Prefab готов. Перемещаем его в папку Assets/Prefabs, а со сцены удаляем.
- Создадим пустой объект FuelManager, набросим на него скрипт FuelManagerScript.
- Не забываем в редакторе назначать public переменные.

```
public GameObject fuelPrefab;
public int numberOfFuels=5;
public Vector3 fuelPos;
GameObject[] listOfFuels;
readonly float maxX = 14;    //Оставляем запас, чтобы горючее
readonly float maxY = 8.3f;  //не появилось совсем на краю
// Start is called before the first frame update
Unity Message | 0 references
void Start()
{ CreateFuelsOnField(); }
1 reference
private void CreateFuelsOnField()
{
    listOfFuels = new GameObject[numberOfFuels];
    for (int i = 0; i < numberOfFuels; i++)
    {
        listOfFuels[i] = Instantiate(fuelPrefab,
            new Vector3(Random.Range(-maxX, maxX),
                Random.Range(-maxY, maxY), 0), Quaternion.identity);
        Debug.Log(listOfFuels[i].transform.position.x + ", "
            + listOfFuels[i].transform.position.y);
    }
    fuelPos = listOfFuels[0].transform.position;
}
```

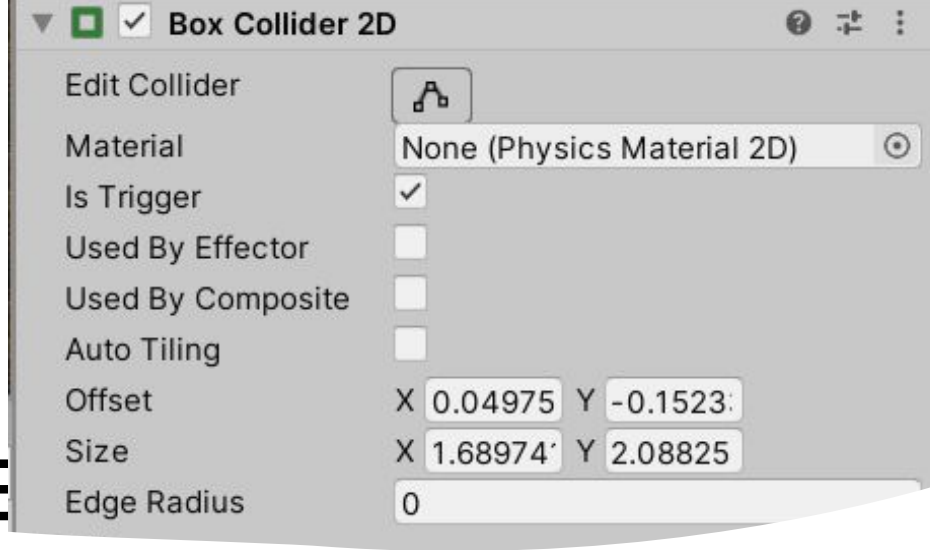


Теперь при
старте
игры на
поле
появляется
горючее

Использование горючего.

- Теперь надо заставить танк расходовать горючее при движении и заправляться горючим при прохождении через цистерны. Сейчас, кстати, и танк и горючее являются твердыми телами с коллайдерами, поэтому танк будет просто расталкивать цистерны. Первое, что нужно сделать – это поставить галочку Is Trigger в Box Collider танка. Теперь горючее будет оставаться на месте, но будут фиксироваться столкновения. Чтобы реагировать на них, напомним в скрипт танка функцию OnTriggerEnter2D.

```
private void OnTriggerEnter2D(Collider2D other)
{
    Destroy(other.gameObject);
}
```



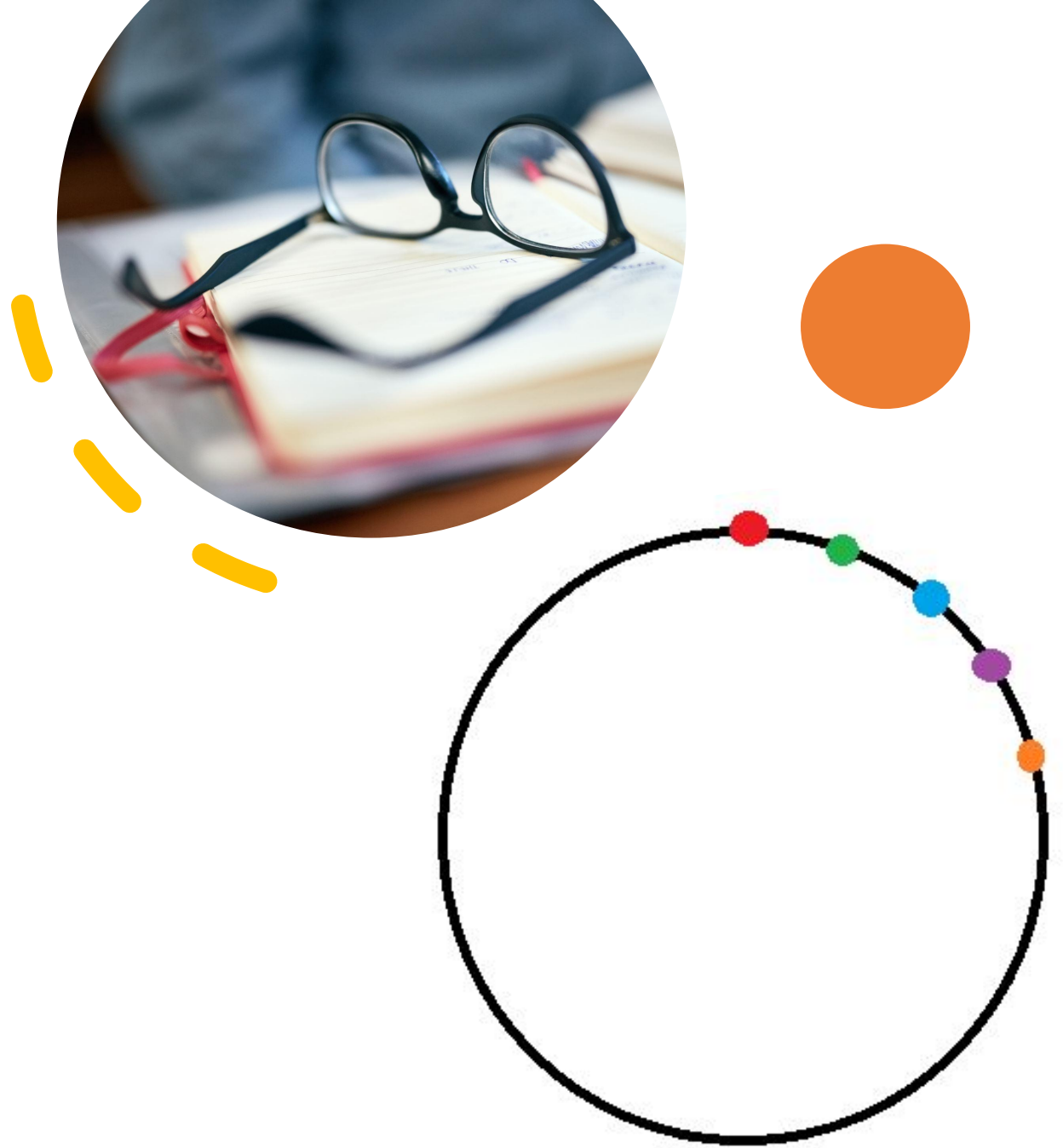


Продолжение следует....

- На следующем занятии мы продолжим делать игру «Удивительные танки» и познакомимся с анимацией и UI (пользовательским интерфейсом в Unity)

Домашнее задание. Часть 1. Легчайшая.

Написать приложение, аналогичное рассыпающимся цветным кубикам, только с цветными кружочками (ого!), которые будут возникать не в одном месте, а рисовать некоторую фигуру (ну, например, квадрат, окружность, фигуру Лиссажу)



Домашнее задание. Часть 2. Творческая.

- Создать свою примитивную игру, используя полученные сегодня навыки. Можно повторить существующую, можно придумать самому. Отлично могут подойти головоломки, собиралки (как сегодня в примере), ловилки (с неба что-то падает, надо подставить корзину). Если Вы придумали игру, но не знаете, как что-то делается в Unity – пишите в тимс, в личные сообщения, постараюсь ответить. Данное задание дается на два занятия.

