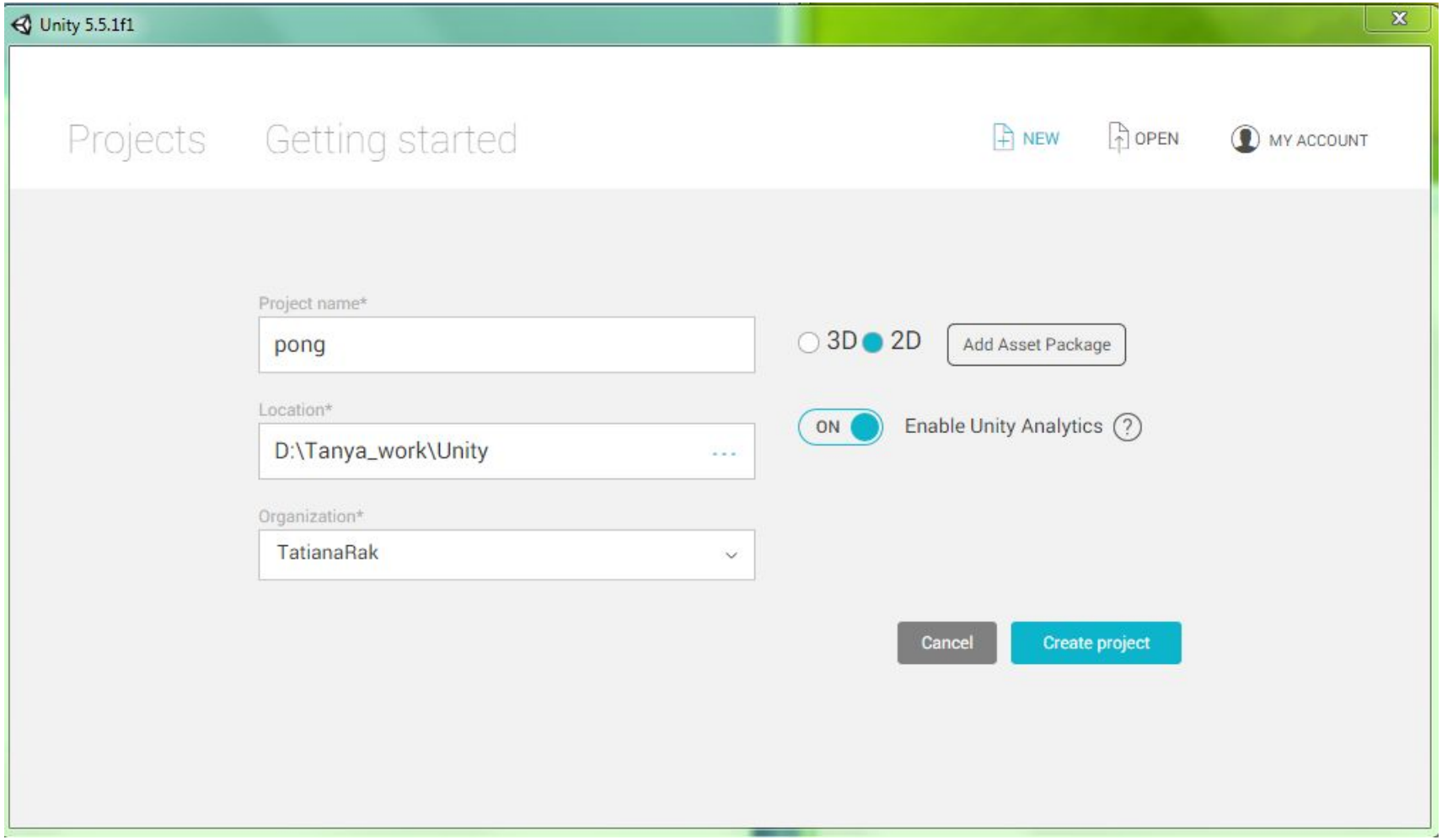
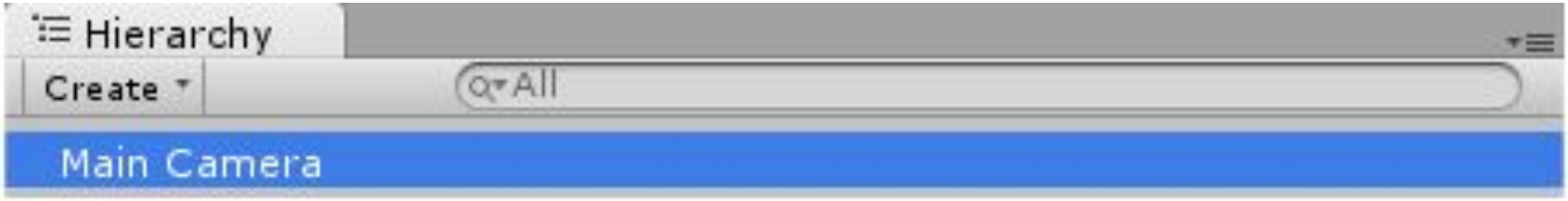
2D-игра ping-pong на Unity3D

Создаем пустой (без ассетов) 2D проект на Unity. Назовем его pong. Сохраним пустую сцену (Верхняя вкладка меню ***File->Save Scene***). Когда спросит имя, можно ввести что-нибудь вроде ***scene_main.***


Unity 5.5.1f1
Projects
Getting started
NEW
OPEN
MY ACCOUNT
Project name*
pong
3D
2D
Add Asset Package
Location*
D:\Tanya_work\Unity
ON
Enable Unity Analytics
Organization*
TatianaRak
Cancel
Create project

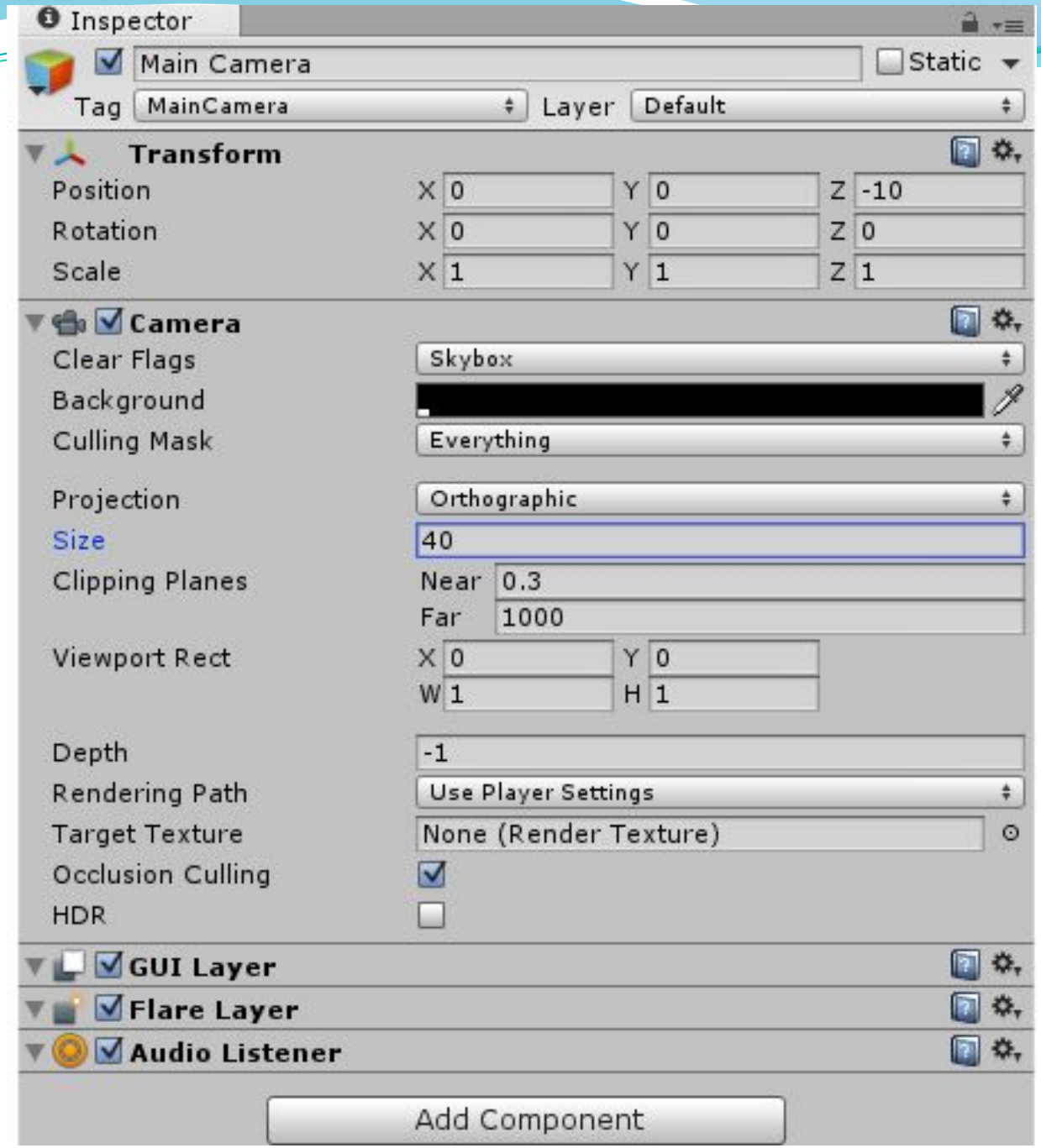
Настройка камеры

Начнем с настройки камеры, чтобы она отображала игру в правильном размере и цвете. Мы можем настроить камеру выбрав её в иерархии справа.



Настройка камеры

Далее можно увидеть настройки данного объекта в инспекторе. Мы поменяем цвет фона (**Background**) на черный и установим нужный нам размер (**Size**):


Inspector
Main Camera
Static
Tag
MainCamera
Layer
Default
Transform
Position
X 0
Y 0
Z -10
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1
Camera
Clear Flags
Skybox
Background
Culling Mask
Everything
Projection
Orthographic
Size
40
Clipping Planes
Near 0.3
Far 1000
Viewport Rect
X 0
Y 0
W 1
H 1
Depth
-1
Rendering Path
Use Player Settings
Target Texture
None (Render Texture)
Occlusion Culling
HDR
GUI Layer
Flare Layer
Audio Listener
Add Component

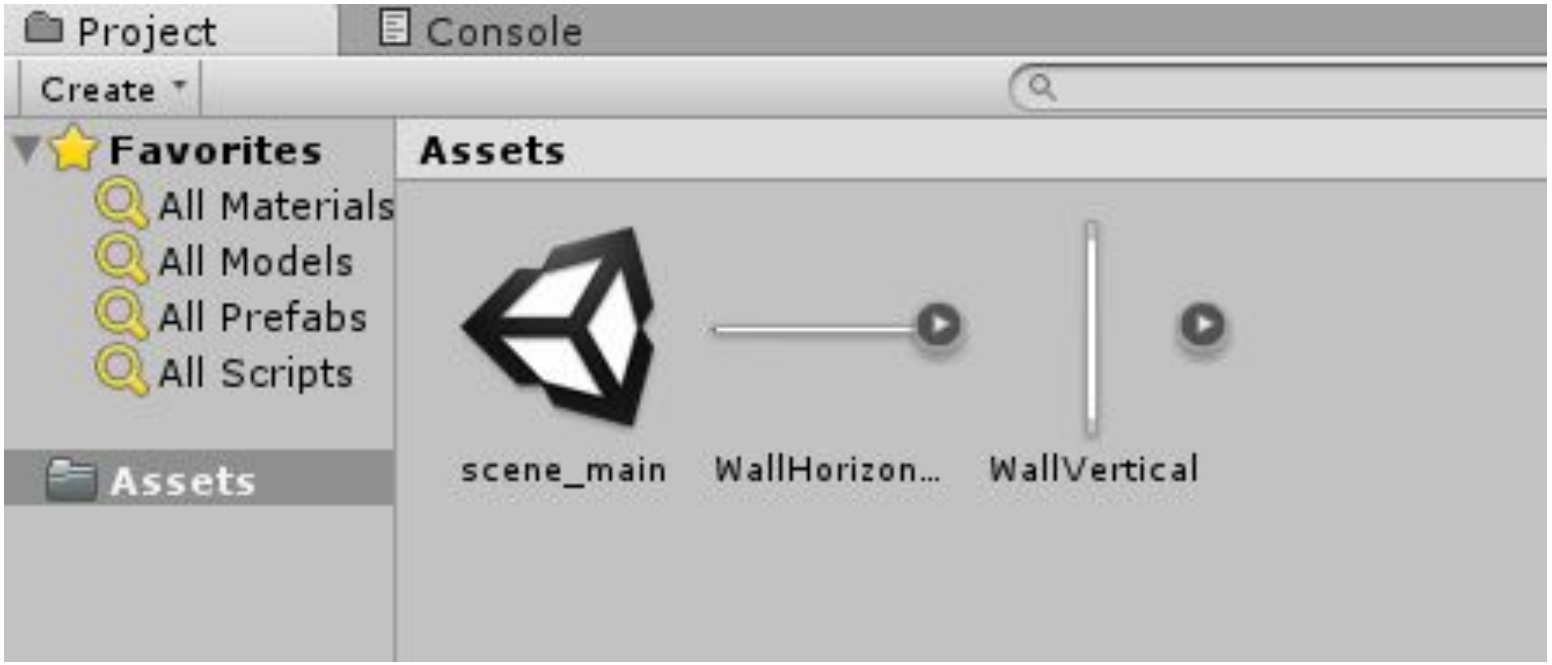
Создаем стены

Давайте добавим к нашему проекту 4 стены. Изображения стен называются **Sprite** или другими словами текстуры.

Используем 2 горизонтальных спрайта для верней и нижней стены и 2 вертикальных для левой и правой стенки. 2 файла WallHorizontal.png и WallVertical.png из приложения сохраняем в папку проекта Assets.

Создаем стены

Спрайты сразу появляются в области проекта:

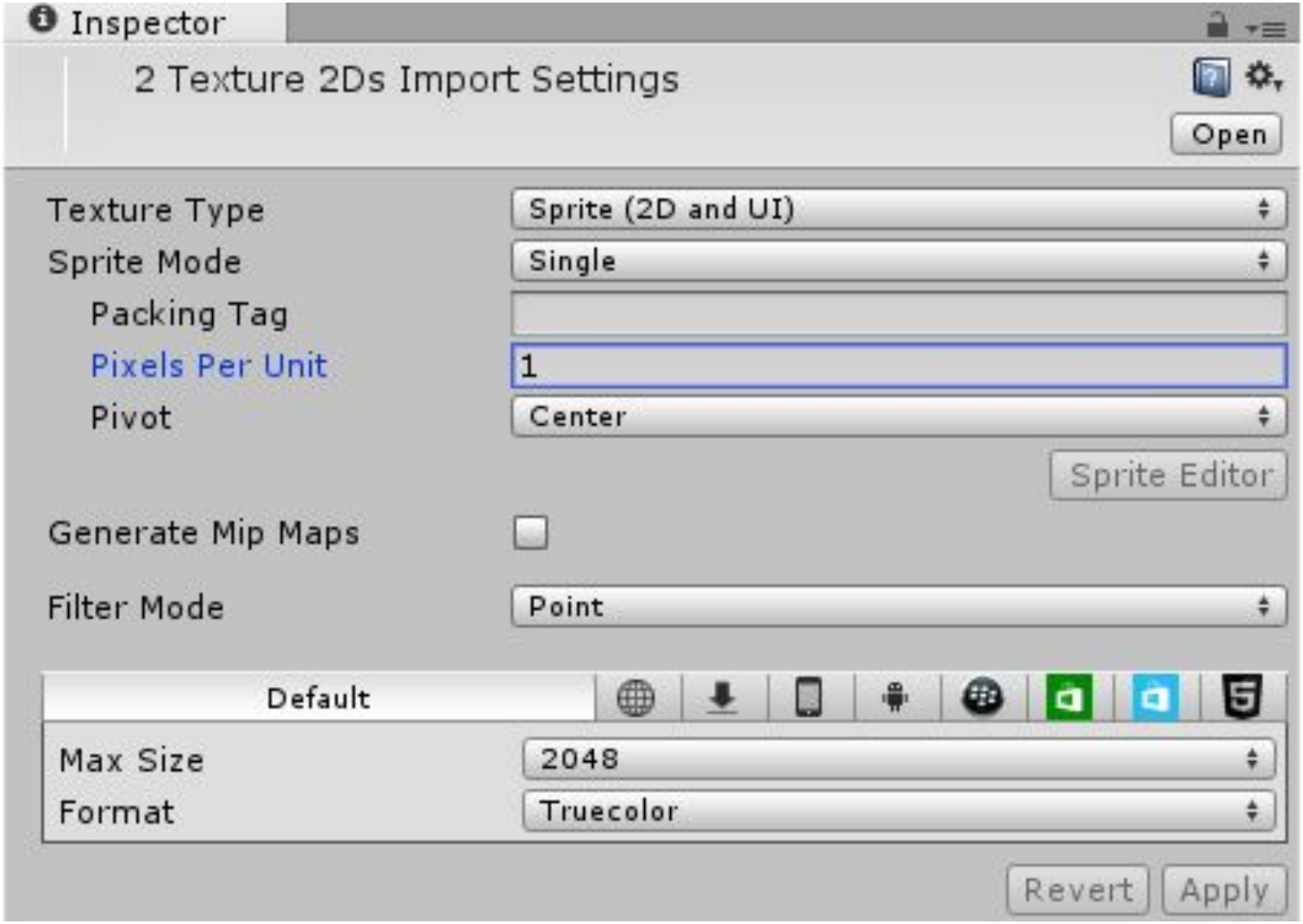


Создаем стены

Далее необходимо импортировать стены в проект, чтобы они хорошо выглядели.

Выделяем 2 спрайта, чтобы сразу применить свойства к двум изображениям и переходим в **Inspector.**

Создаем стены



Создаем стены

Pixels Per Unit равный 1 означает, что каждый пиксель ставится в соответствие 1 единице игрового мира.

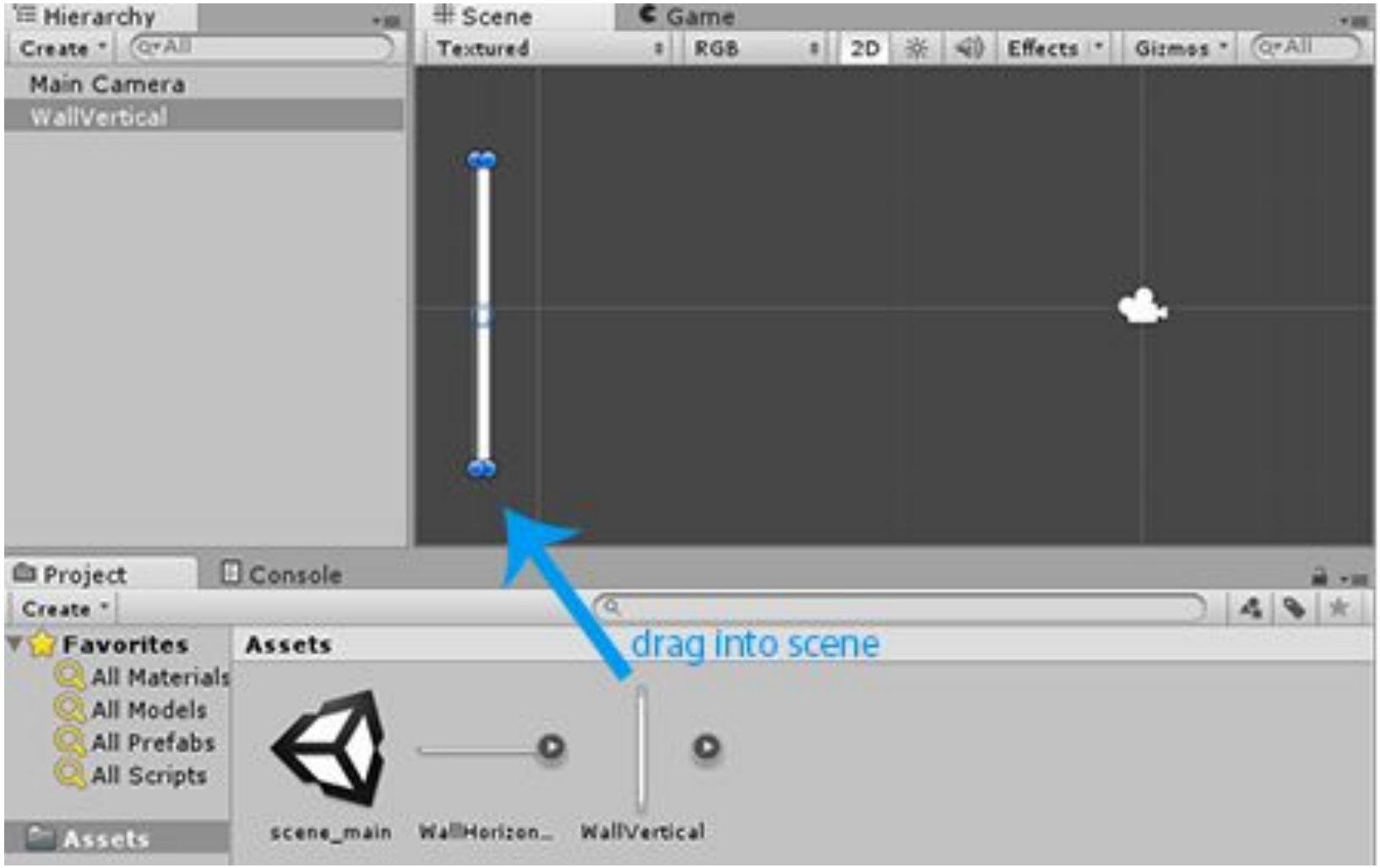
Настройка импорта кажется бесполезной, т.к. игра будет и так отлично работать. Но могут возникать проблемы с настройкой физики.

Создаем стены

Теперь надо добавить стены на сцену и расположить их симметрично вокруг камеры.

Для этого выделим изображение стены в **Project Area** и перетянем ее на **Scene**.

Создаем стены



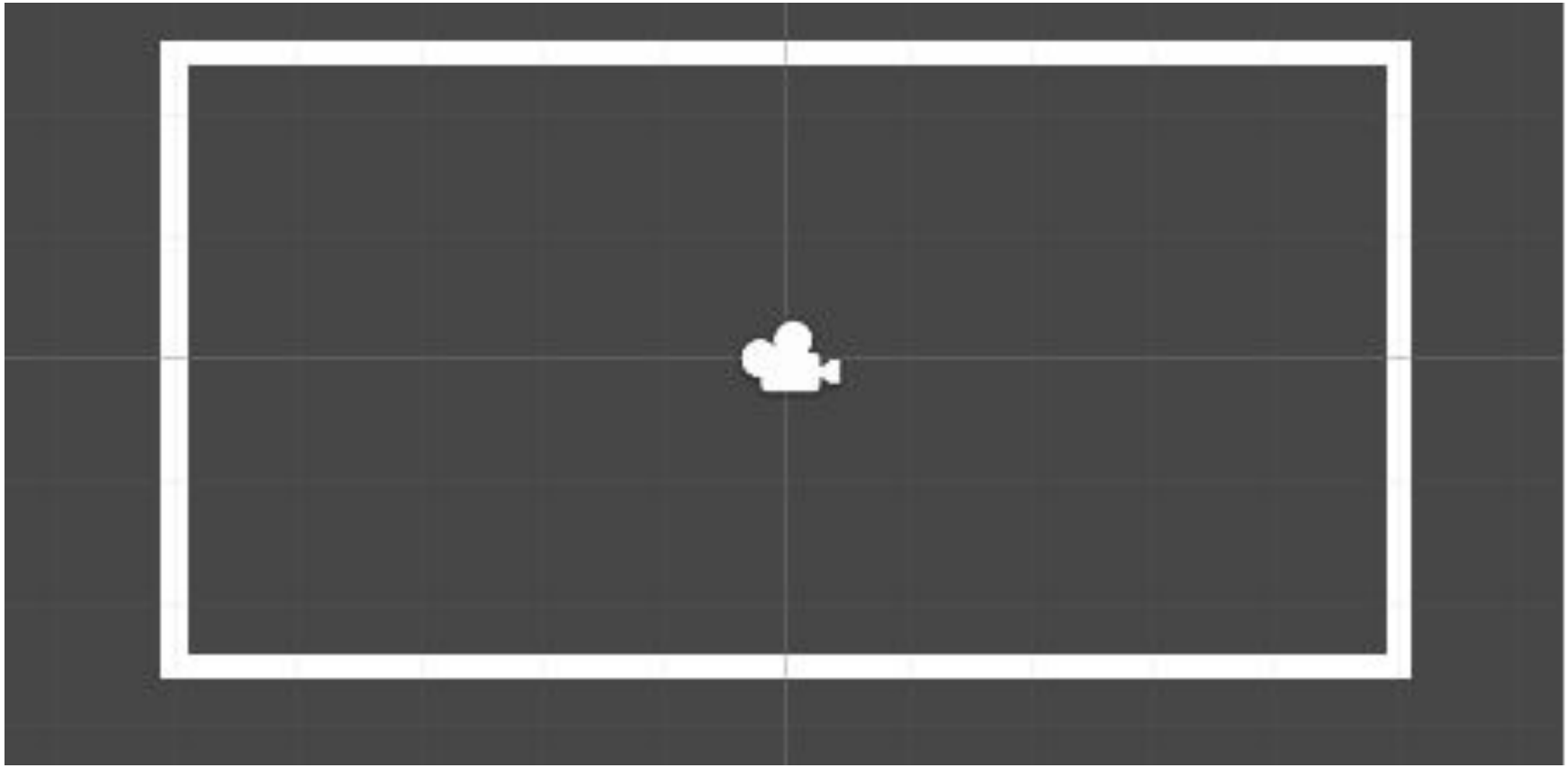
Создаем стены

У нас должно появиться 2 горизонтальные и 2 вертикальные стенки.

Чтобы составить из них прямоугольник с камерой в центре можно непосредственно перетягивать их на сцене, а можно настроить их положение через ***Inspector.***

После всех манипуляций должно получится так:

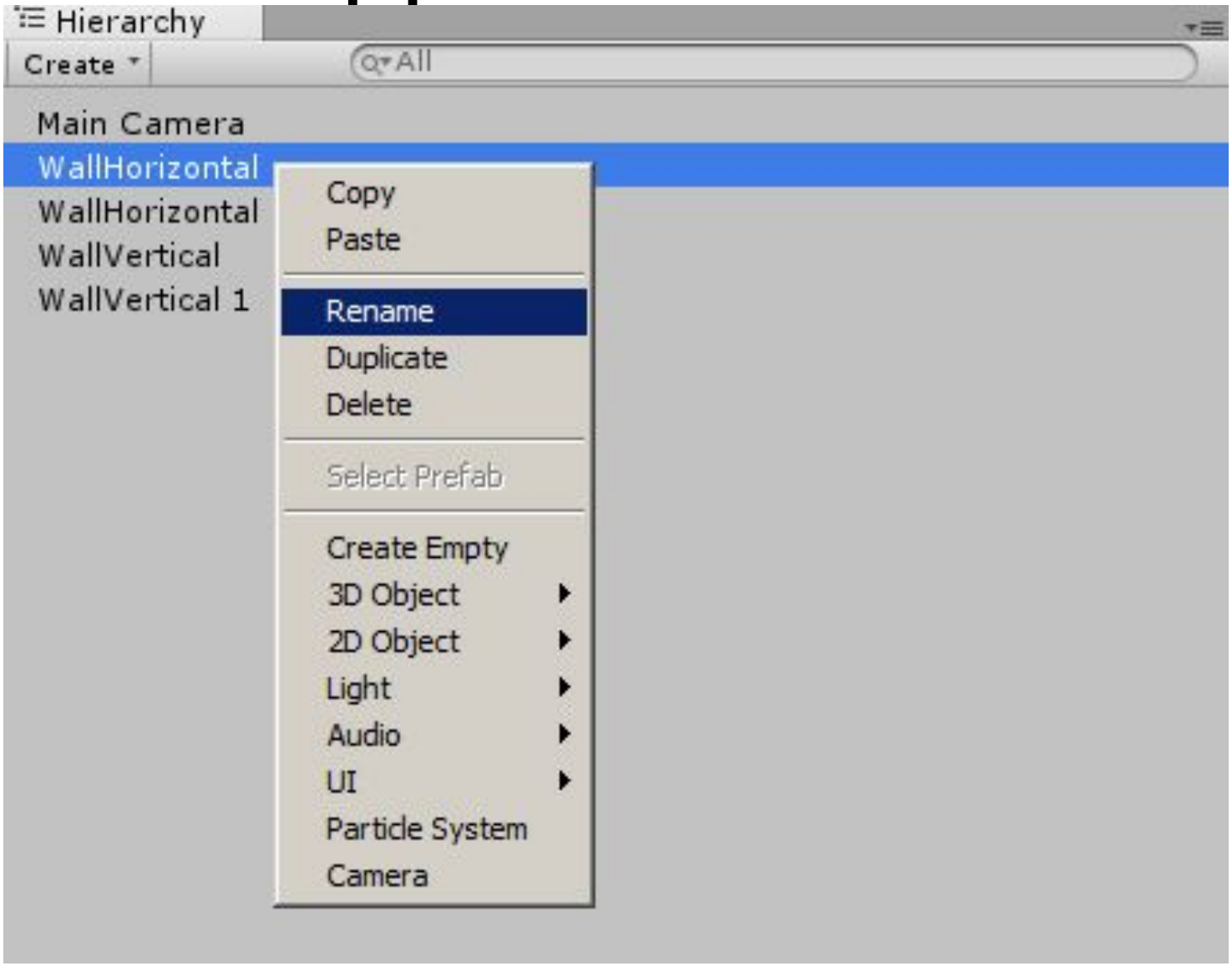
Создаем стены



Создаем стены

Дальше необходимо переименовать стены, чтобы было удобнее их различать. Это очень легко сделать. В окне иерархий выбираем объект и щелкаем правой кнопкой мыши, выбирая **Rename**.

Создаем стены



Создаем стены

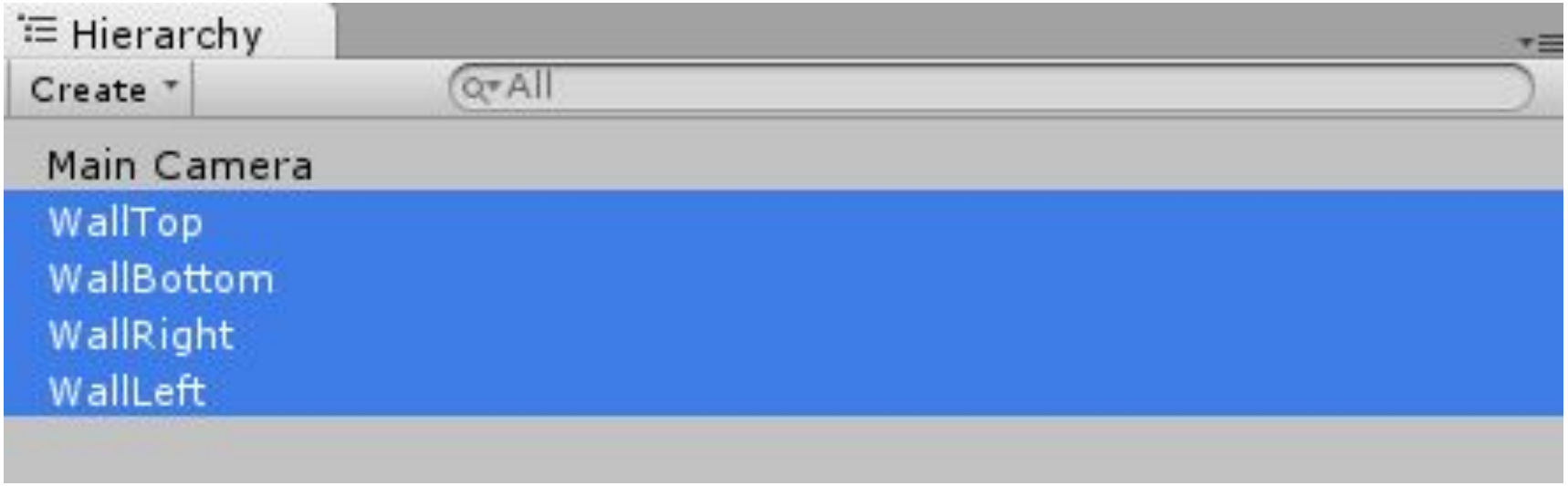
После переименования **Hierarchy** выглядит так:

Создаем физику стены

Сейчас у нас есть стены, но пока они просто картинки. Они не правильно взаимодействуют со средой и объектами. Надо сделать их настоящими стенами, чтобы ракетки и мячик не проходили сквозь них.

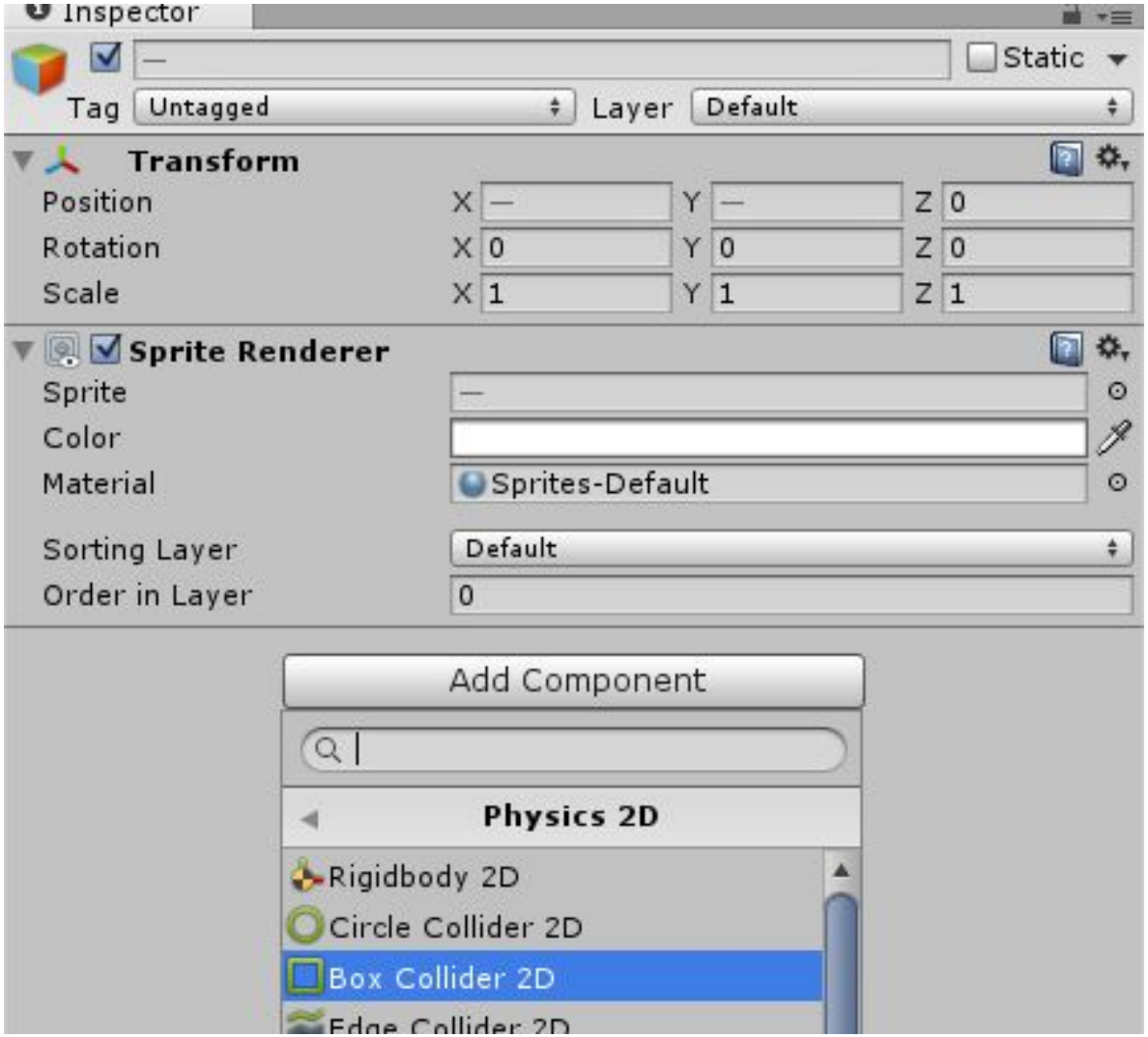
Мы должны сделать их **Colliders.** Выделяем все стены в иерархии:

Создаем физику стены



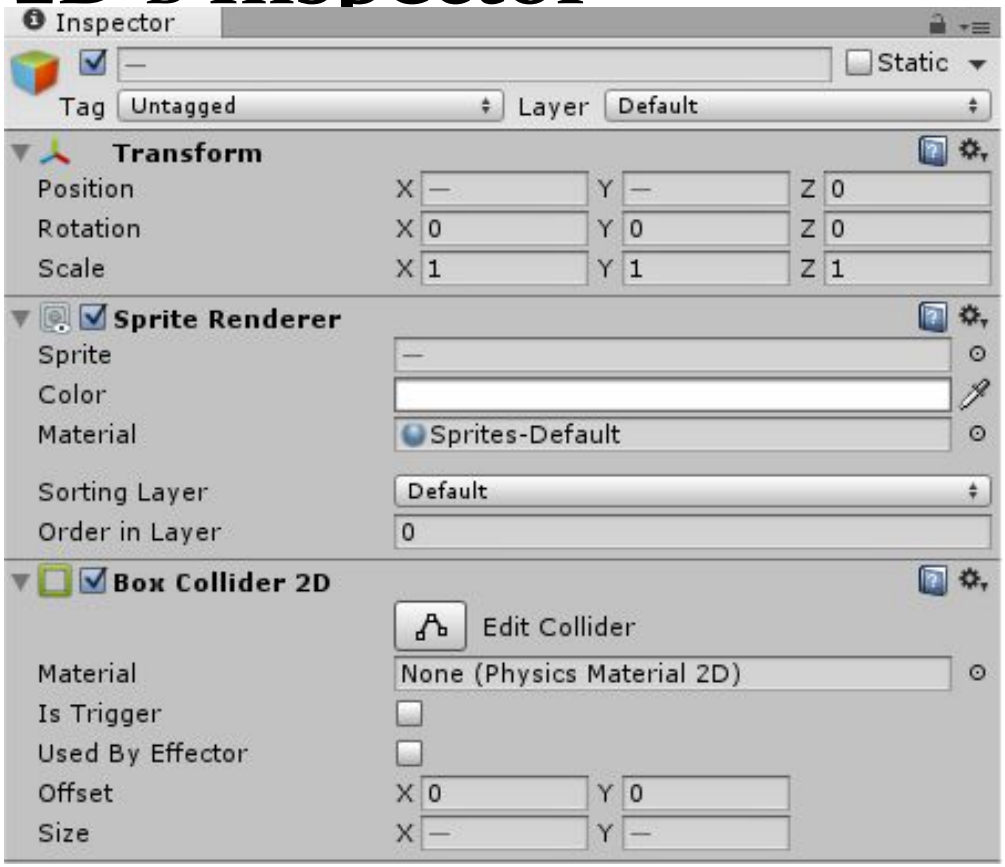
Дальше нажимаем кнопку **Add Component** в **Inspector** и выбираем **Physics2D->Box Collider 2D**

Создаем физику стены



Создаем физику стены

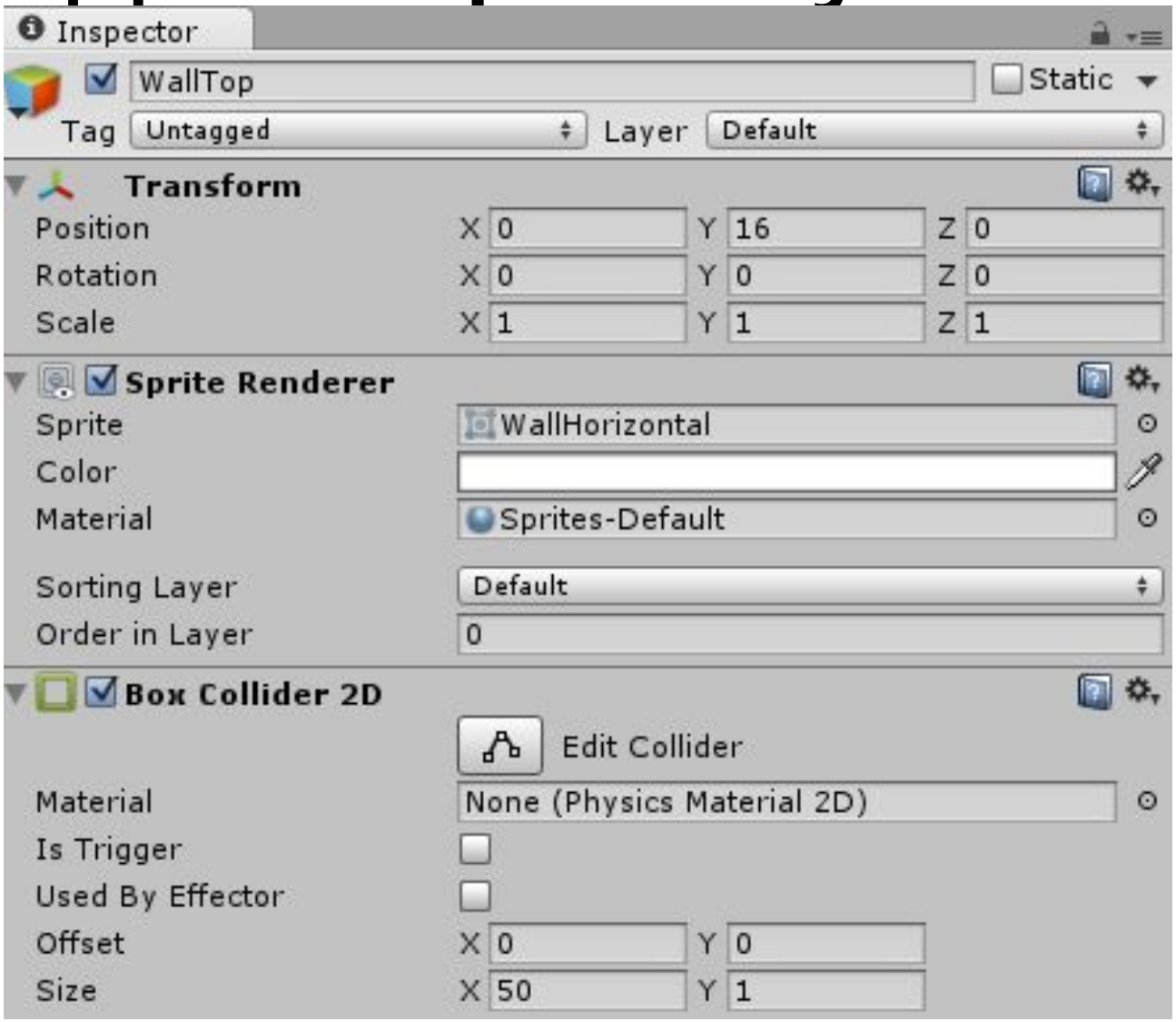
Теперь у всех 4 стен есть компонента **Box Collider 2D** в **Inspector**



Создаем физику стены

Теперь если мы посмотрим на **Scene**, мы увидим, что стены обведены зеленым прямоугольником который и есть ***colliders***. Они видны только на **Scene**, в самой игре их не будет видно.
Также можно выбрать одну любую стенку, чтобы посмотреть ее свойства.

Создаем физику стены



Добавляем штрихпунктирную линию

Сейчас добавим разделительную штрихпунктирную линию DottedLine.png из приложения в папку проекта **Assets**. Произведем с ней такие же манипуляции по импорту, как и со стенами ранее.
После перетянем ее на сцену проекта и установим посередине.
Должно получится так:

Добавляем штрихпунктирную линию

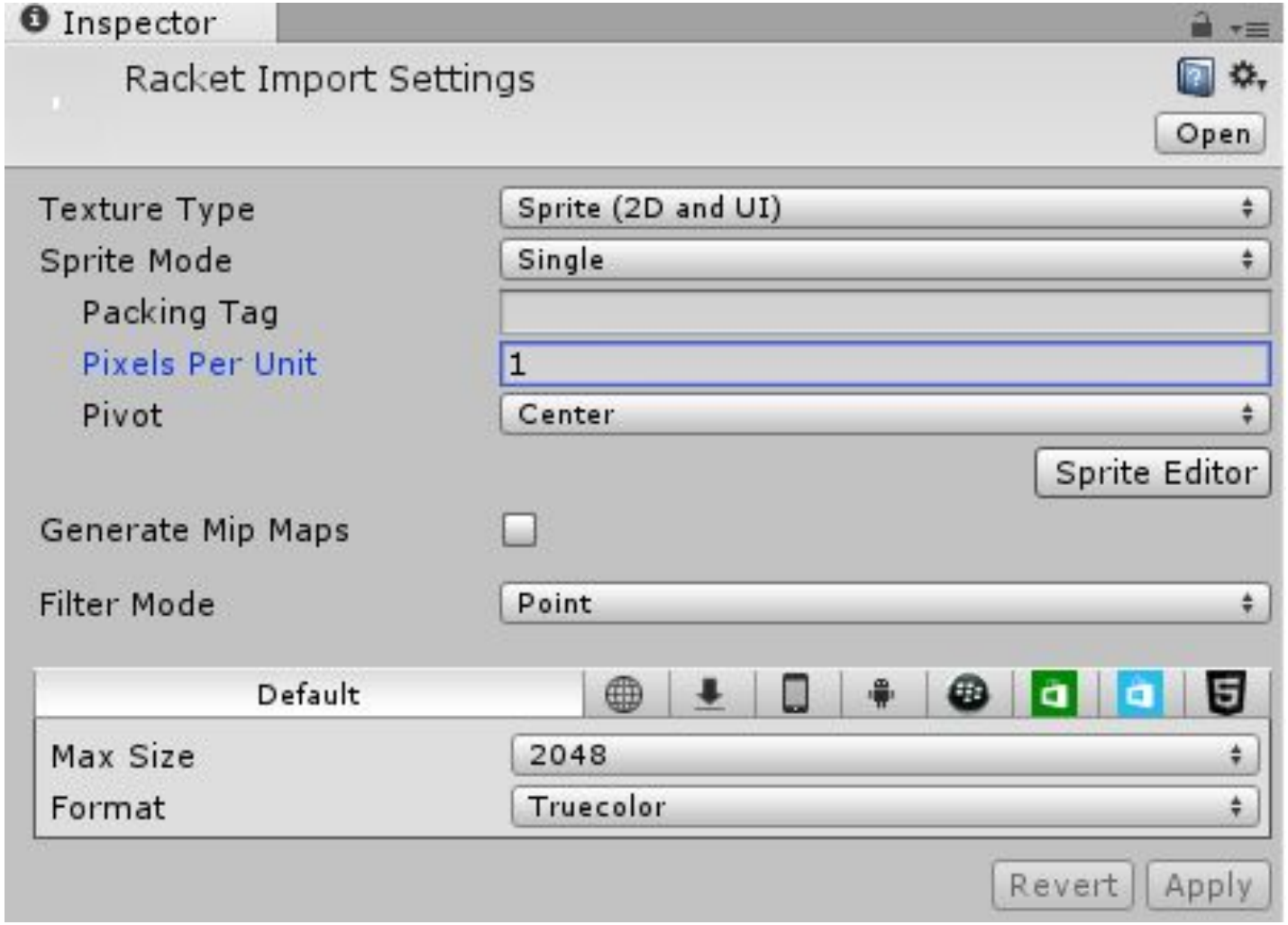
Добавляем штрихпунктирную линию

Т.к. на штрихпунктирную линию мы никаких физических свойств не повесили, шарик будет пролетать сквозь нее, не соударяясь с ней.

Создание ракеток

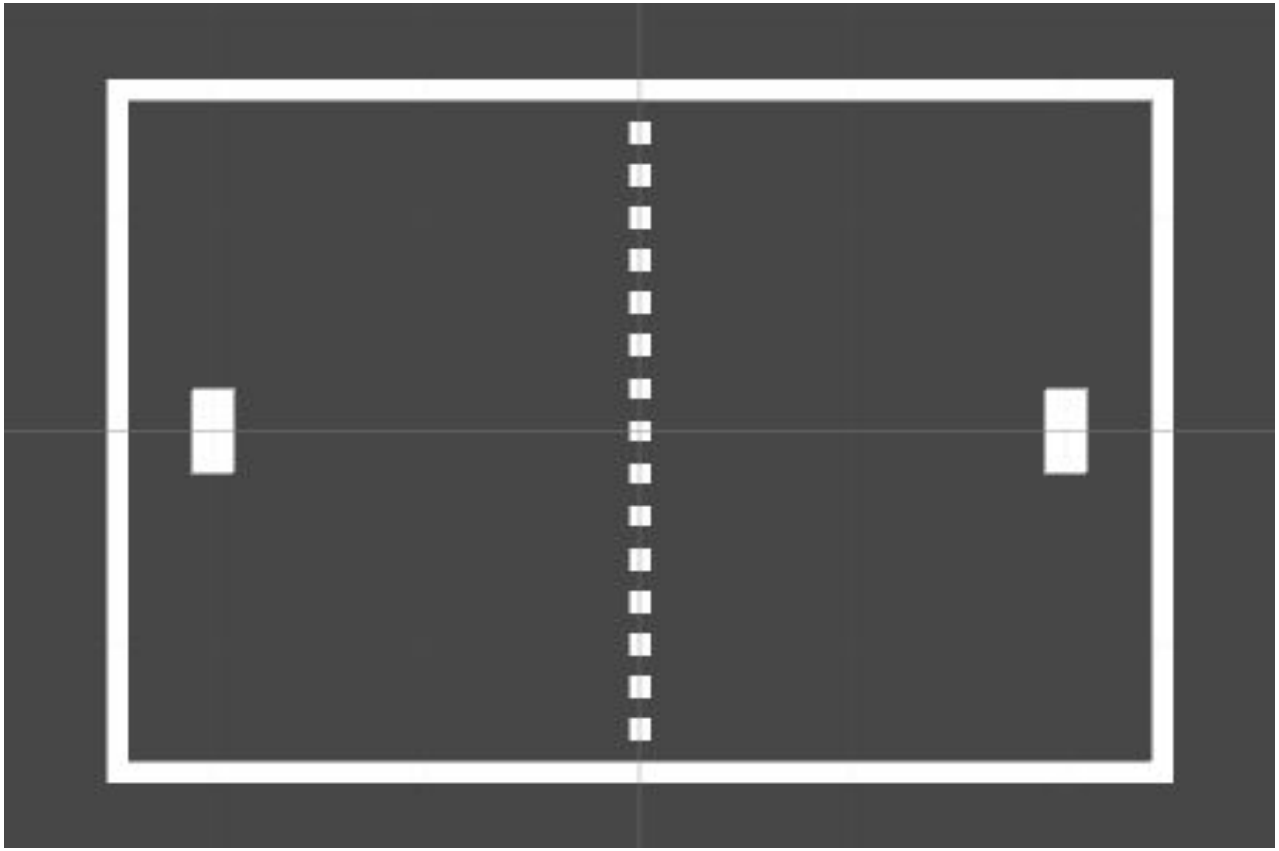
Используем следующую текстуру для ракеток: Racket.png.
Сохраним ее в папку проекта **Assets.**
Используем следующие настройки для импорта:

Создание ракеток



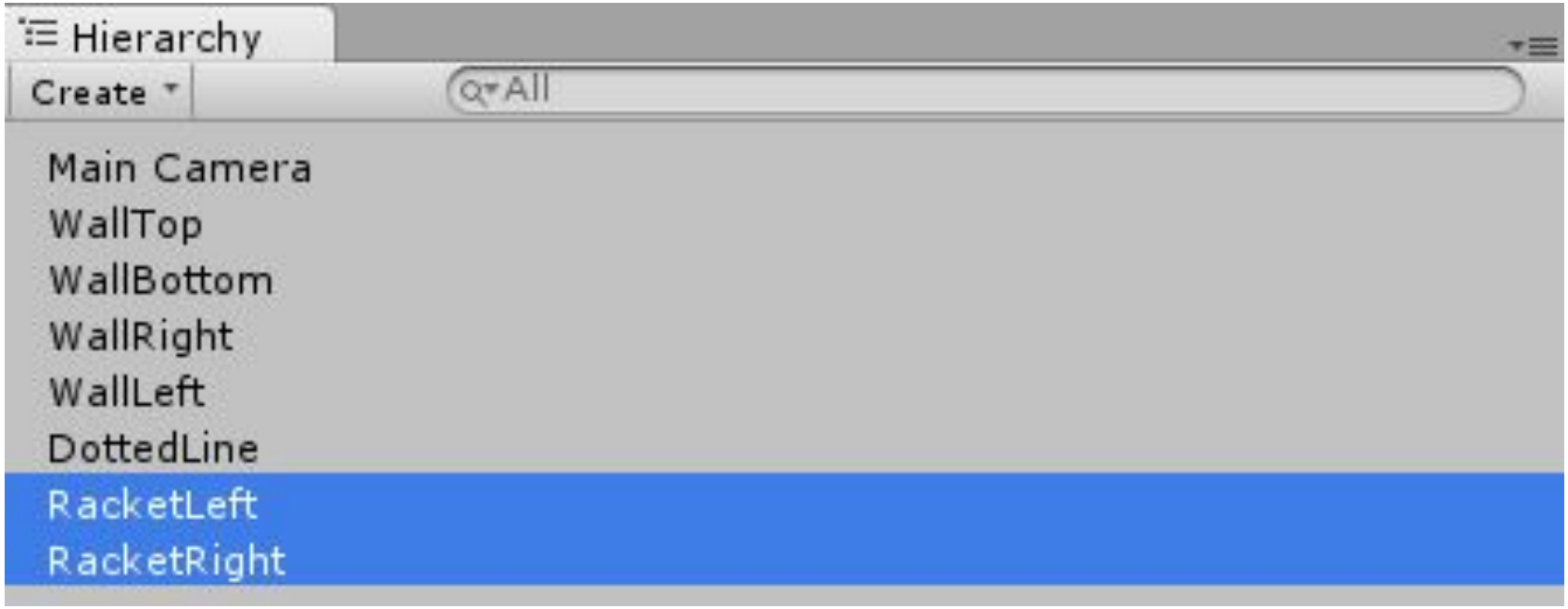
Создание ракеток

Т.к. у нас 2 игрока, ракетки будут располагаться слева и справа по центру.



Создание ракеток

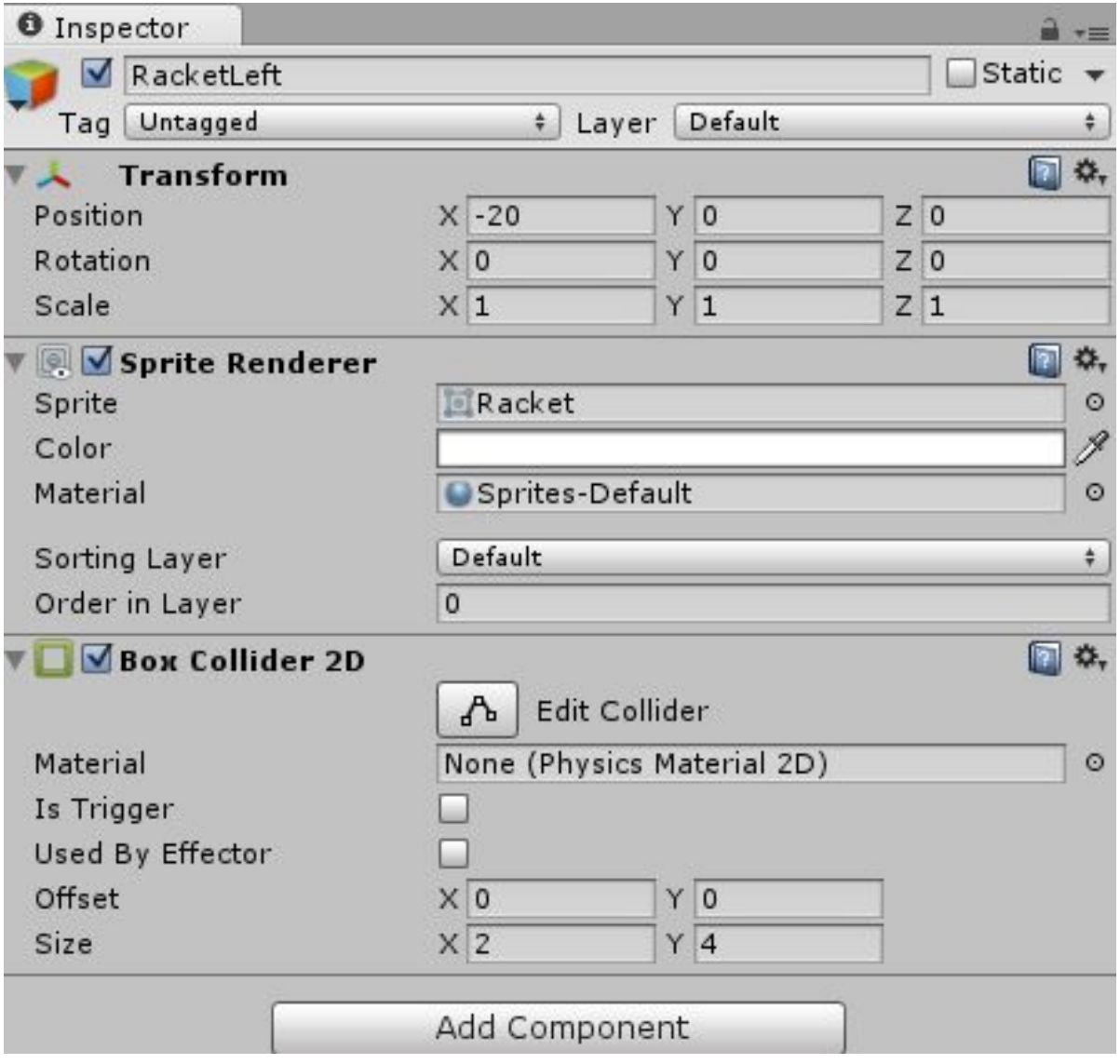
Переименуем ракетки в **Hierarchy**



Физика ракеток

Шарик должен отскакивать от ракеток, поэтому к ним тоже добавим взаимодействие. Нажмем кнопку **Add Component->Physics 2D->Box Collider 2D** в **Inspector.**

Физика ракеток



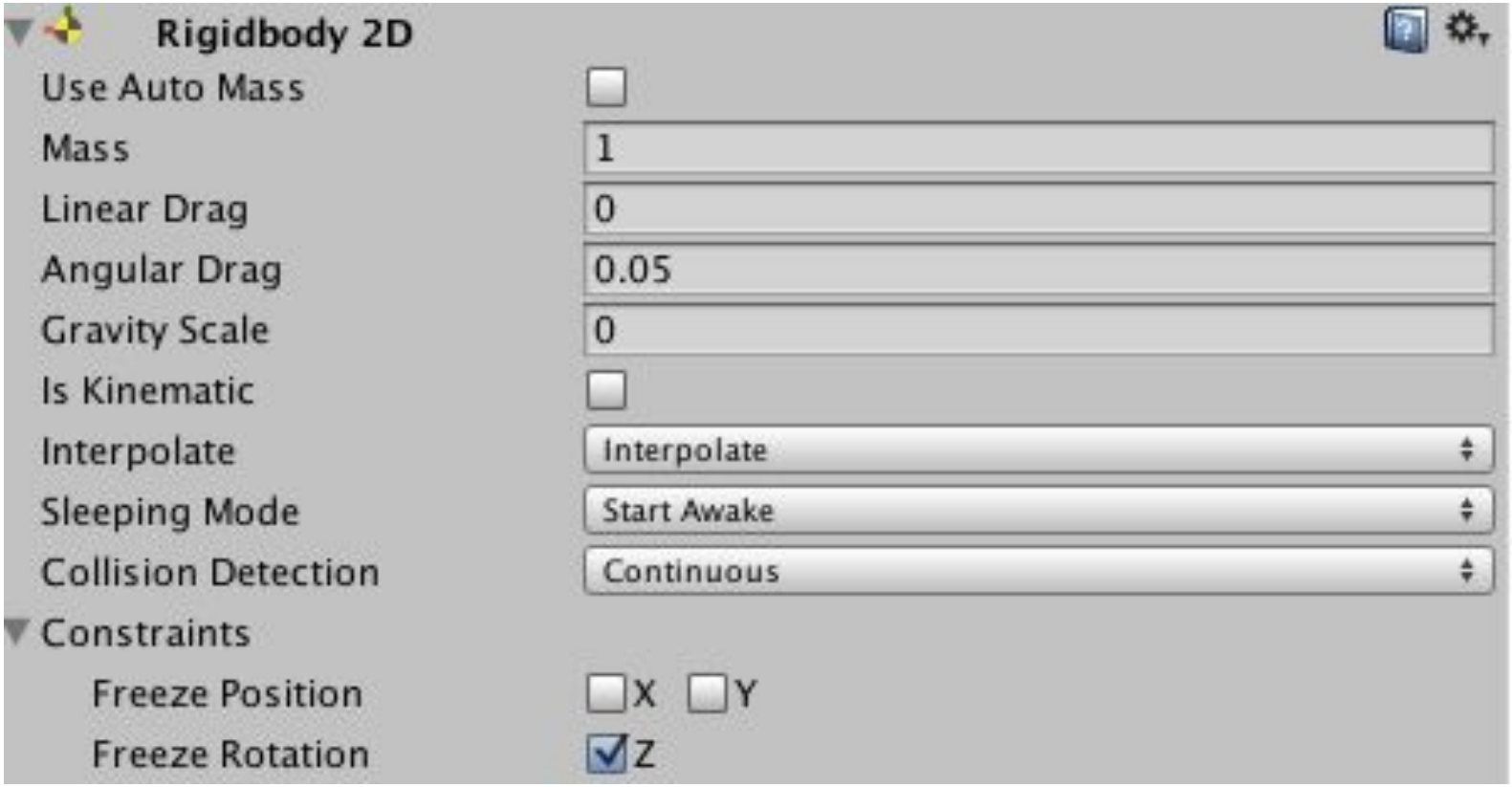
Физика ракеток

Ракетки у игрока должны двигаться вверх и вниз и не проходить сквозь стены. Для этого нам понадобится **Rigidbody.** Оно автоматически определяет положение объекта физически правильным образом. Поэтому, если в игре у нас есть движущийся объект, для корректного взаимодействия нужен **Rigidbody.**

Физика ракеток

Для того, чтобы добавить **Rigidbody** для наших ракеток, выберем их в иерархии и в **Inspector** нажмем **Add Component->Physics 2D->Rigidbody 2D**. Затем мы отключим в **Rigidbody** гравитацию, т.к. иначе ракетки будут падать, и зафиксируем ось **z** , чтоб ракетки не поворачивались.

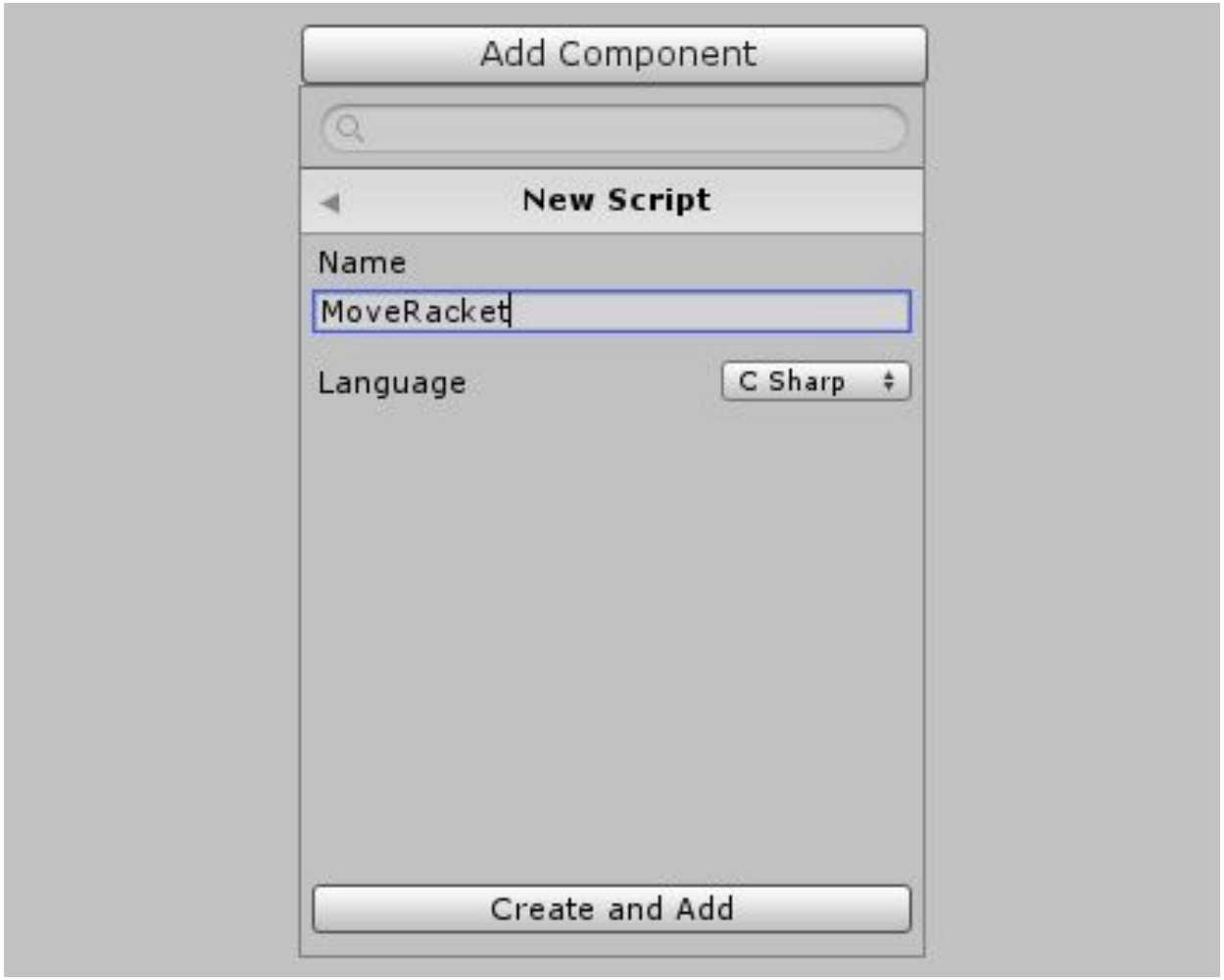
Физика ракеток



Движение ракеток

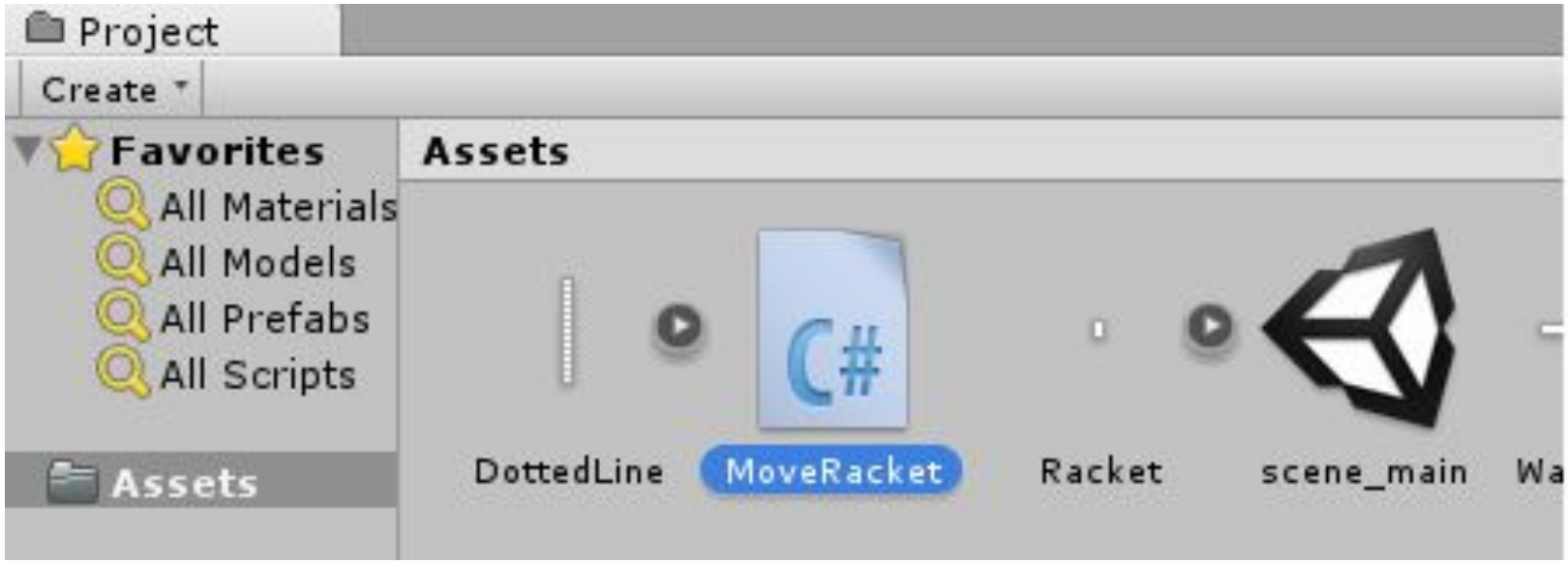
Теперь надо сделать так, чтобы игроки могли двигать свои ракетки. Для этого надо создать свой скрипт. Выделив 2 ракетки, нажимаем кнопку **Add Component->New Script,** назовем его **MoveRacket** и выберем язык **CSharp.**

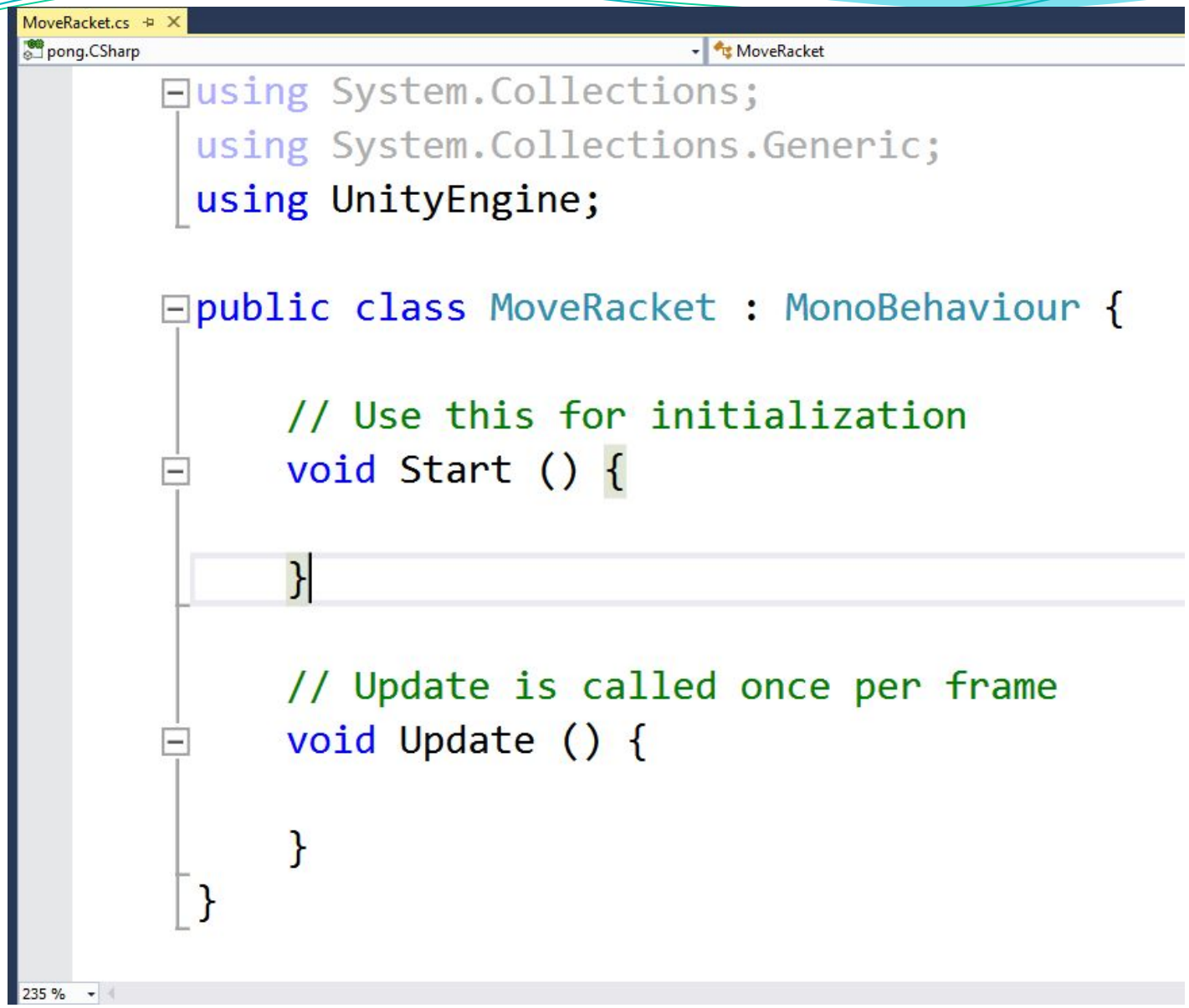
Движение ракеток



Движение ракеток

Теперь дважды щелкаем по нашему скрипту в области проекта, чтобы перейти в среду разработки.




MoveRacket.cs
pong.CSharp
MoveRacket
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class MoveRacket : MonoBehaviour {
// Use this for initialization
void Start () {
}
// Update is called once per frame
void Update () {
}
}
235 %

Движение ракеток

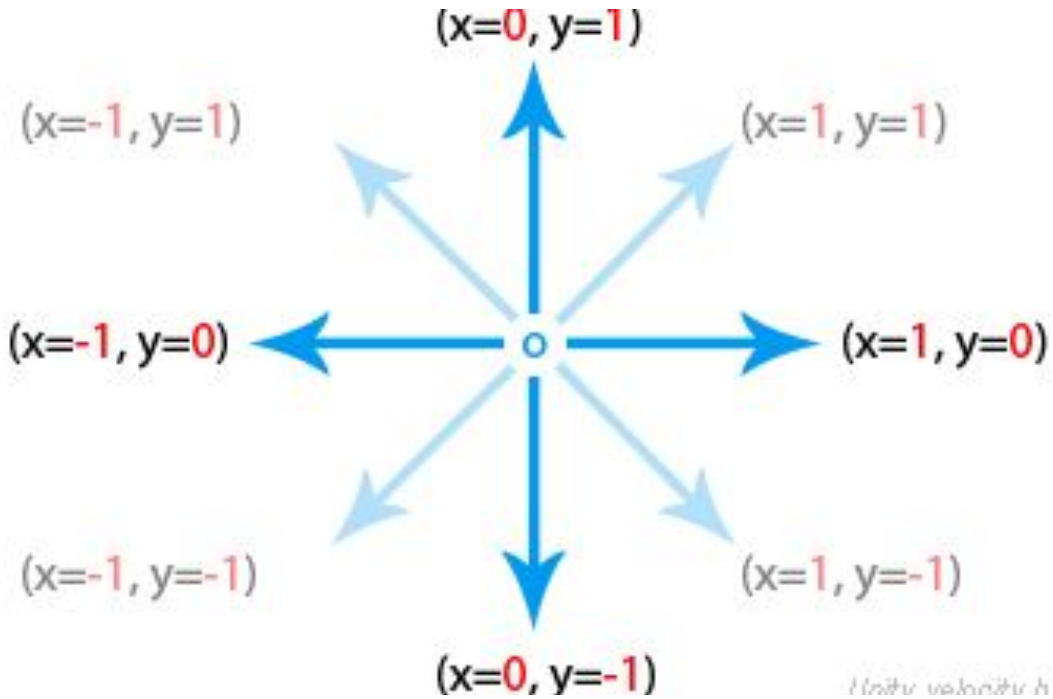
Функция **Start** вызывается **Unity** автоматически, когда запускается игра.
Функция **Update** автоматически вызывается снова и снова приблизительно 60 раз в секунду.
Есть другая функция **FixedUpdate.** Она вызывается фиксированное количество раз за интервал времени.

Движение ракеток

Физика в Unity пересчитывается через равные интервалы времени, поэтому удобнее использовать **FixedUpdate.**
Т.к. у ракеток есть **Rigidbody** мы будем использовать скорость (**velocity**) для их движения. Скорость это произведение направления движения на ускорение.
Направления движения задается **Vector2.**

Движение ракеток

Примеры движения **Vector2.**



Движение ракеток

Наши ракетки будут двигаться только вверх и вниз, поэтому будут изменяться только компонента y. -1 движение вниз, 1 движение вверх, 0 когда не двигается. Для отслеживания движения по осям используем функцию **GetAxisRaw.** Она возвращает 1, когда нажата w, -1, когда s, и 0, когда ничего не нажато.

Движение ракеток

```
using UnityEngine;
using System.Collections;

public class MoveRacket : MonoBehaviour {

    void FixedUpdate () {
        float v = Input.GetAxisRaw("Vertical");
    }
}
```

Движение ракеток

Теперь используем **GetComponent**, чтобы получить доступ к **Rigidbody** ракеток и установить скорость.
Далее добавим переменную ускорение в скрипт, чтобы контролировать скорость перемещения ракеток.
Т.к. переменная public, мы можем менять ее значение в **Inspector**, не изменяя скрипт.

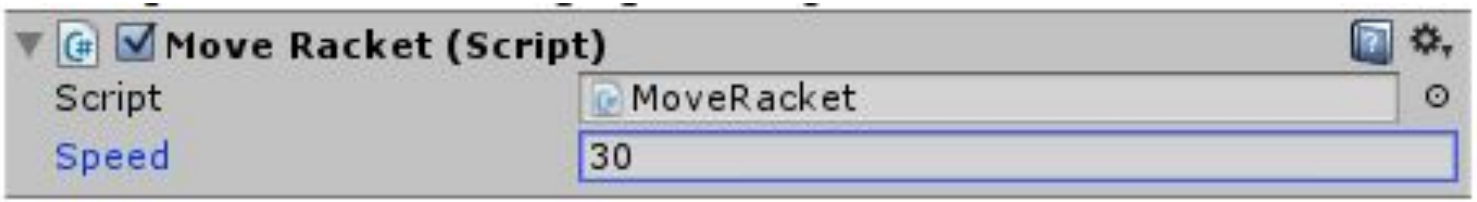
Движение ракеток

```
using UnityEngine;
using System.Collections;

public class MoveRacket : MonoBehaviour {
    public float speed = 30;

    void FixedUpdate () {
        float v = Input.GetAxisRaw("Vertical");
        GetComponent<Rigidbody2D>().velocity =
new Vector2(0, v);
    }
```

Движение ракеток

Теперь используем **GetComponent**, чтобы получить доступ к **Rigidbody** ракеток и установить скорость.
Далее добавим переменную ускорение в скрипт, чтобы контролировать скорость перемещения ракеток.
Т.к. переменная public, мы можем менять ее значение в **Inspector**, не изменяя скрипт.

Движение ракеток




```
using UnityEngine;
using System.Collections;

public class MoveRacket : MonoBehaviour {
    public float speed = 30;

    void FixedUpdate () {
        float v = Input.GetAxisRaw("Vertical");
        GetComponent<Rigidbody2D>().velocity =
new Vector2(0, v) * speed;
    }
```

Движение ракеток

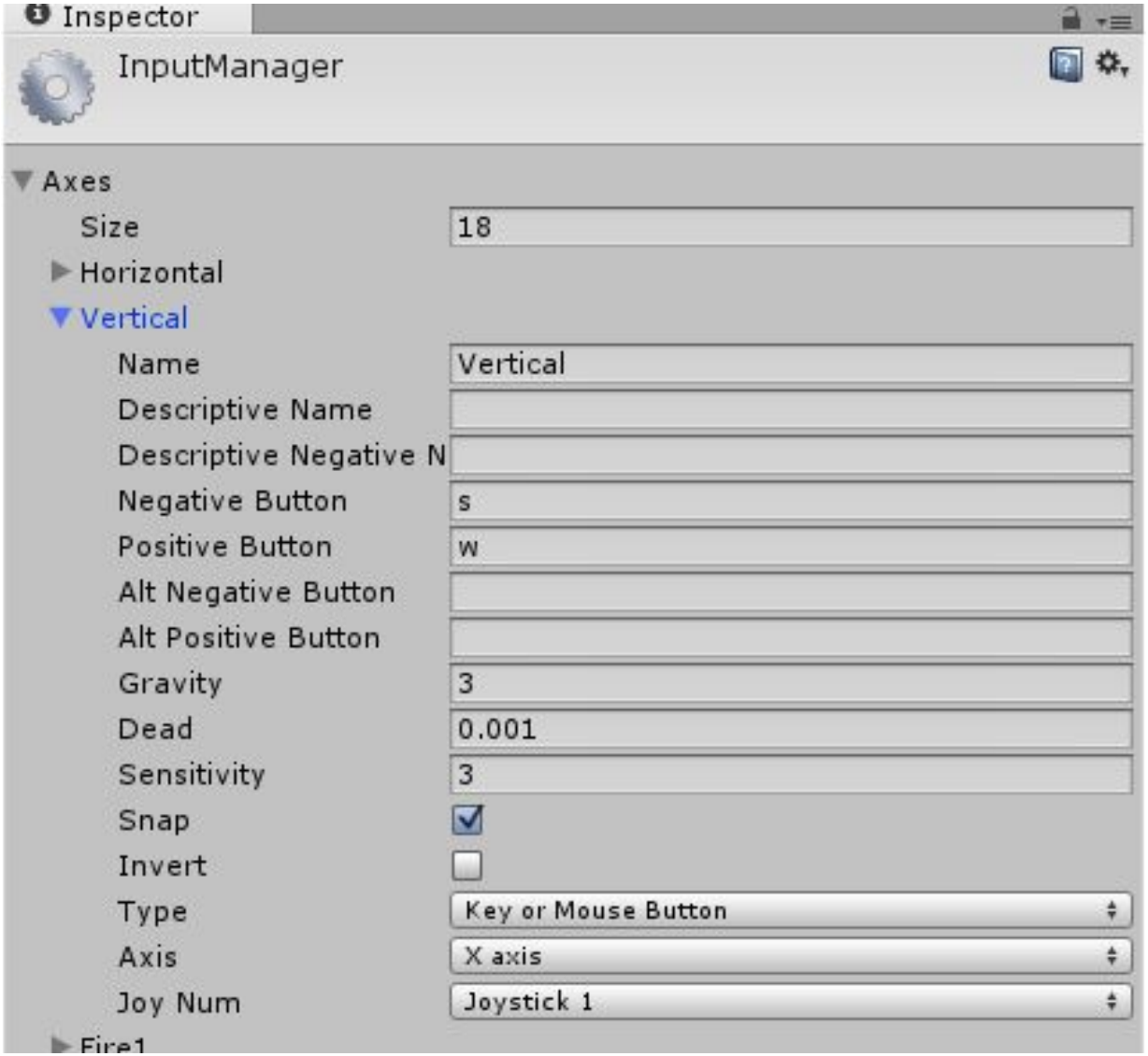
Если сохраним скрипт и запустим игру, то теперь мы можем перемещать ракетки. Но есть проблема, они двигаются вместе. Создадим для второй ракетки дополнительную переменную ось, чтобы менять ввод оси в инспекторе.

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

public class MoveRacket : MonoBehaviour {
    public float speed = 30;
    public string axis = "Vertical";

    // Update is called once per frame
    void FixedUpdate () {
        float v = Input.GetAxisRaw(axis);
        GetComponent<Rigidbody2D>().velocity = new Vector2(0, v) * speed;

    }
}
```

Движение ракеток

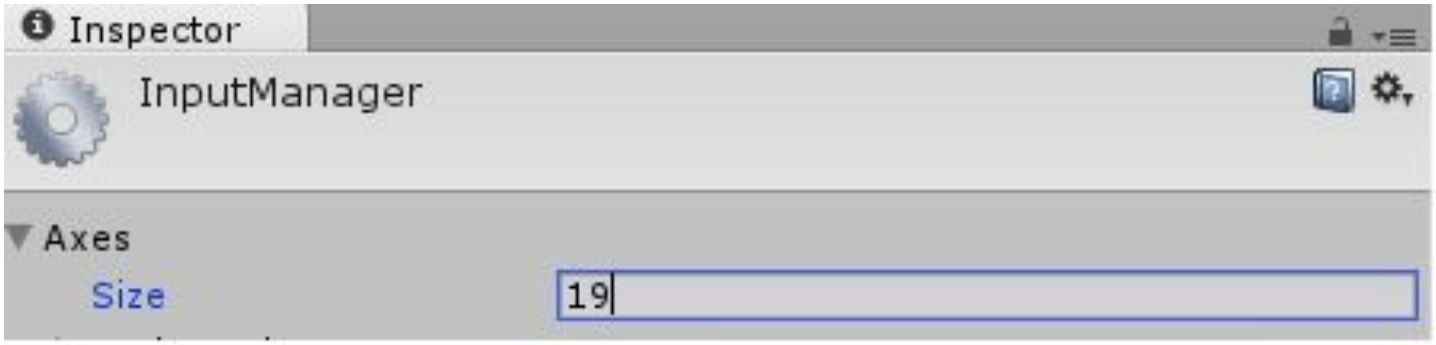
Менять оси будем через вкладку Input.
В меню сверху выбираем **Edit->Project Settings->Input .** Здесь изменим текущую вертикальную ось, таким образом, чтобы она принимала клавиши s и w.

Движение ракеток



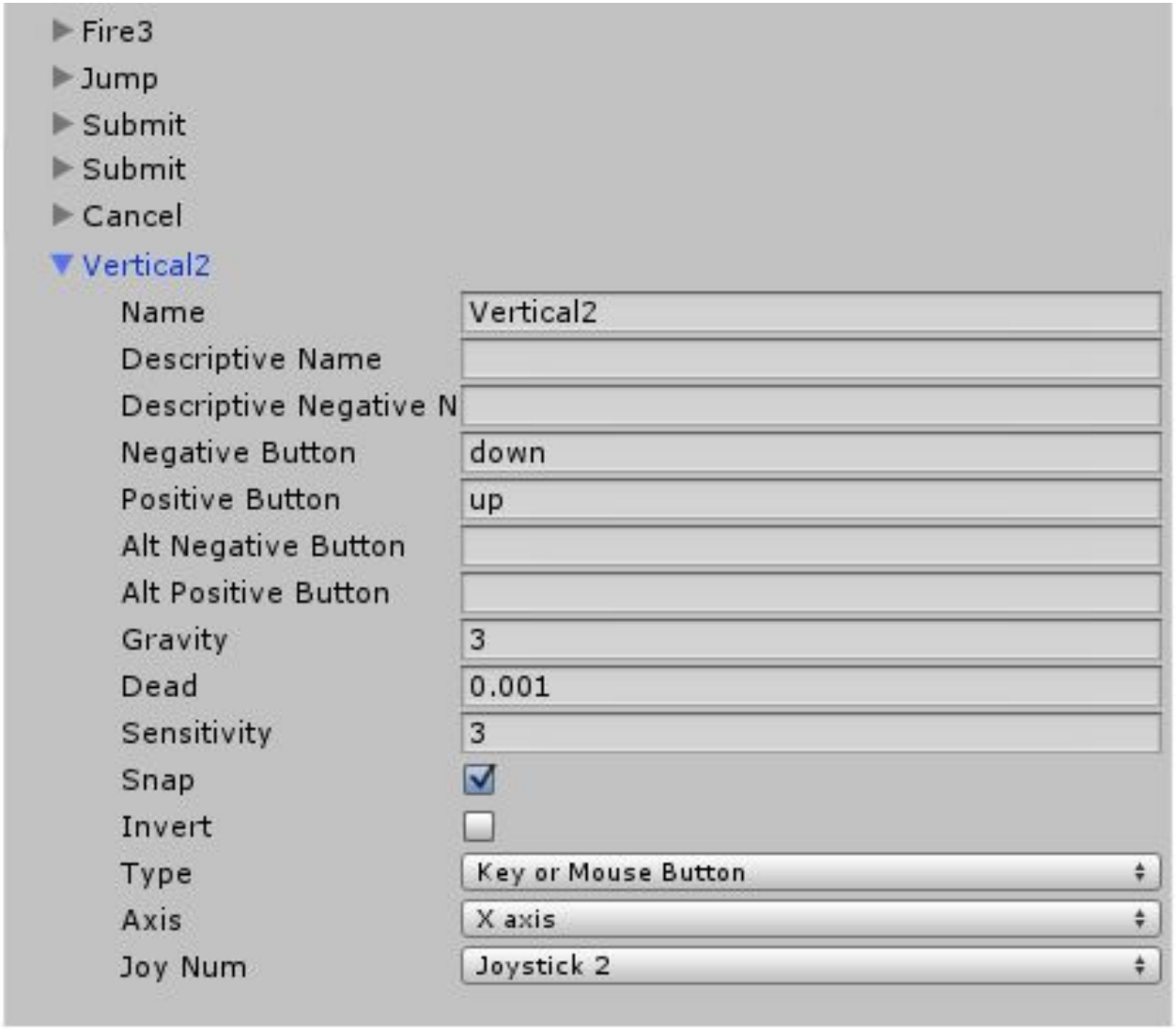
Движение ракеток

Добавим еще одну ось



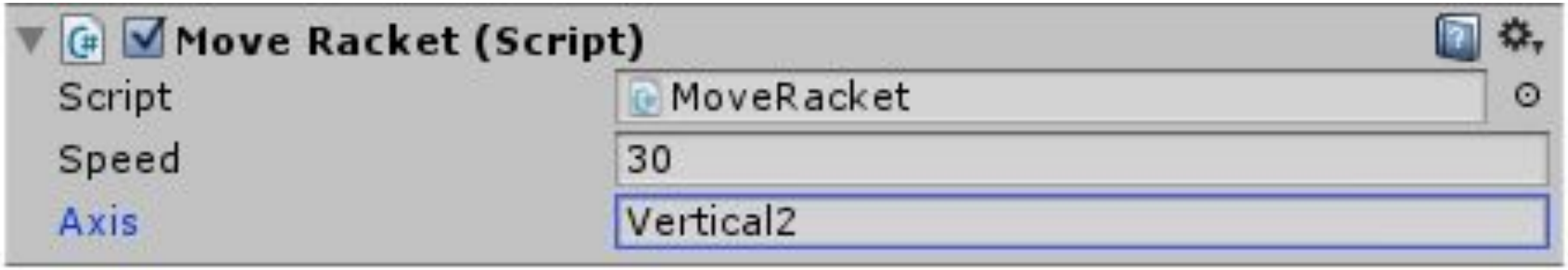
Назовем её **Vertical2** и изменим ее настройки.

Движение ракеток



Движение ракеток

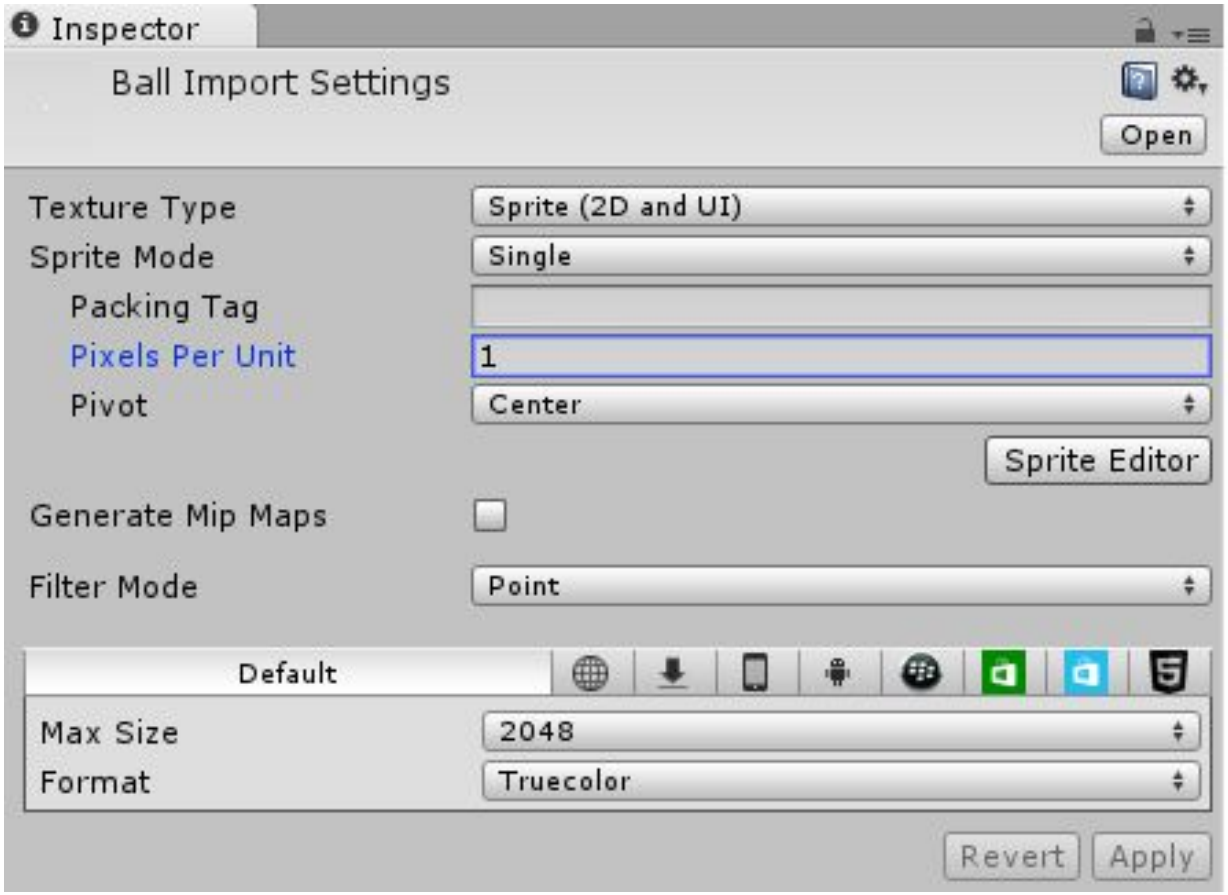
Теперь выберем **RacketRight** и изменим в ее скрипте **Axis:**



Теперь ракетки будут двигаться независимо.

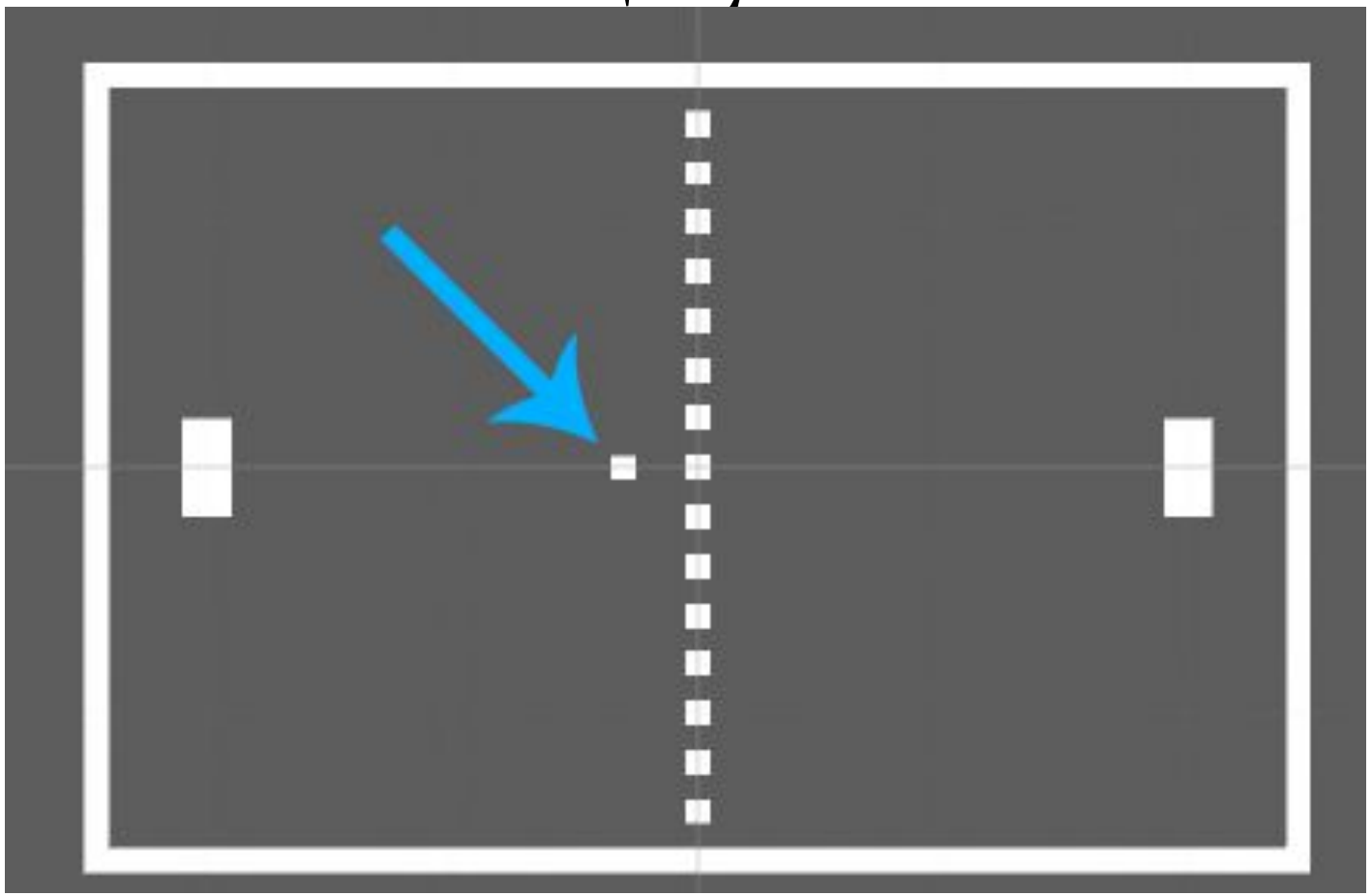
Мячик

Добавим мяч из приложения Ball.png в папку проекта **Assets**. Импортируем его:



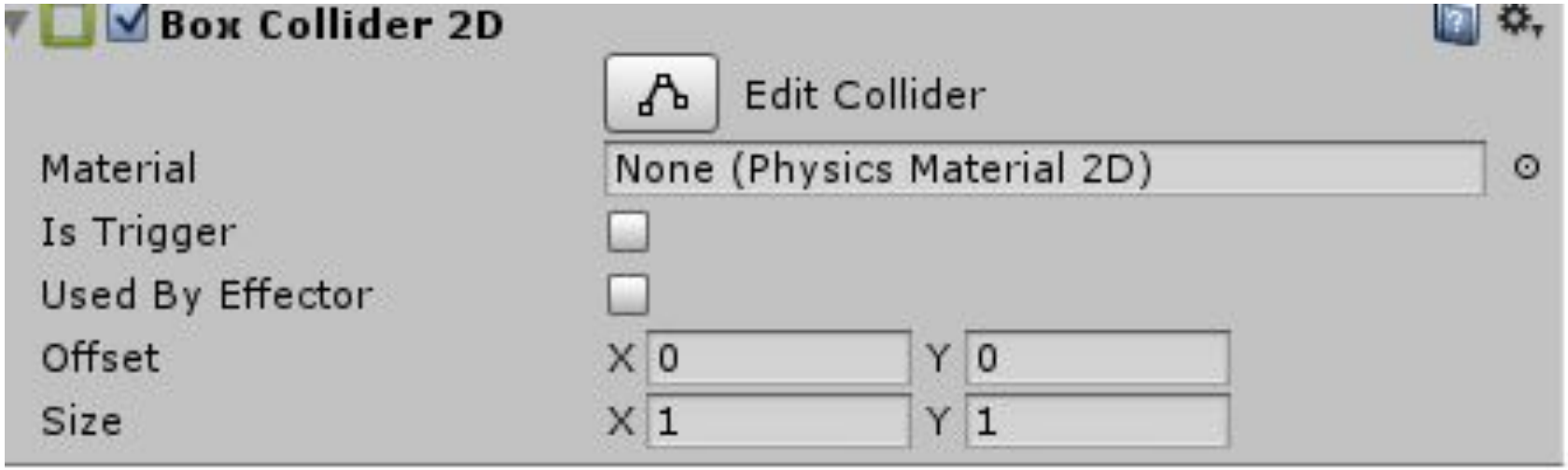
Мячик

Перетянем его на сцену



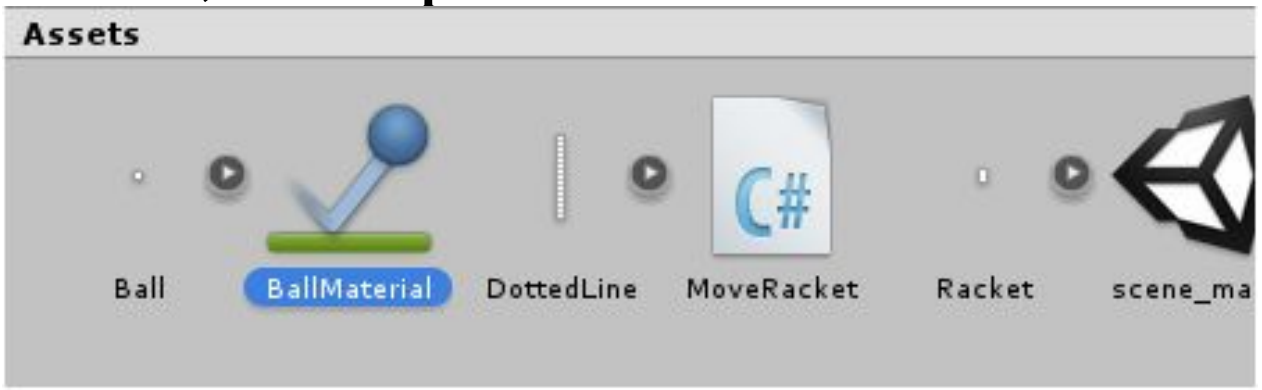
Мячик

Настроим физику для мяча:
Add Component->Physics 2D->Box Collider 2D



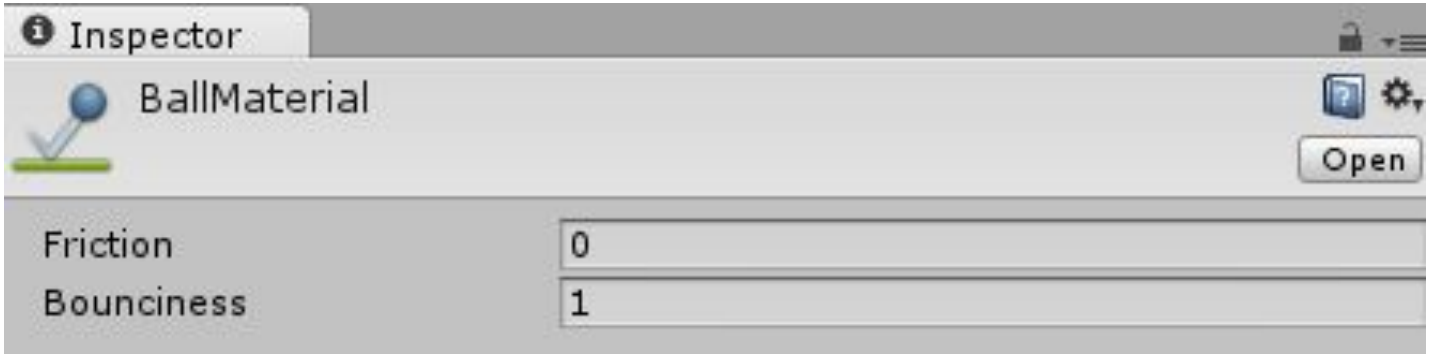
Мячик

Мячик должен отскакивать от стен и ракеток.
Добавим **Physics Material.**
Щелкаем правой кнопкой мыши в **Project Area** и выбираем **Create->Physics2D Material** , который назовем **BallMaterial.**



Мячик

В Inspector меняем настройки

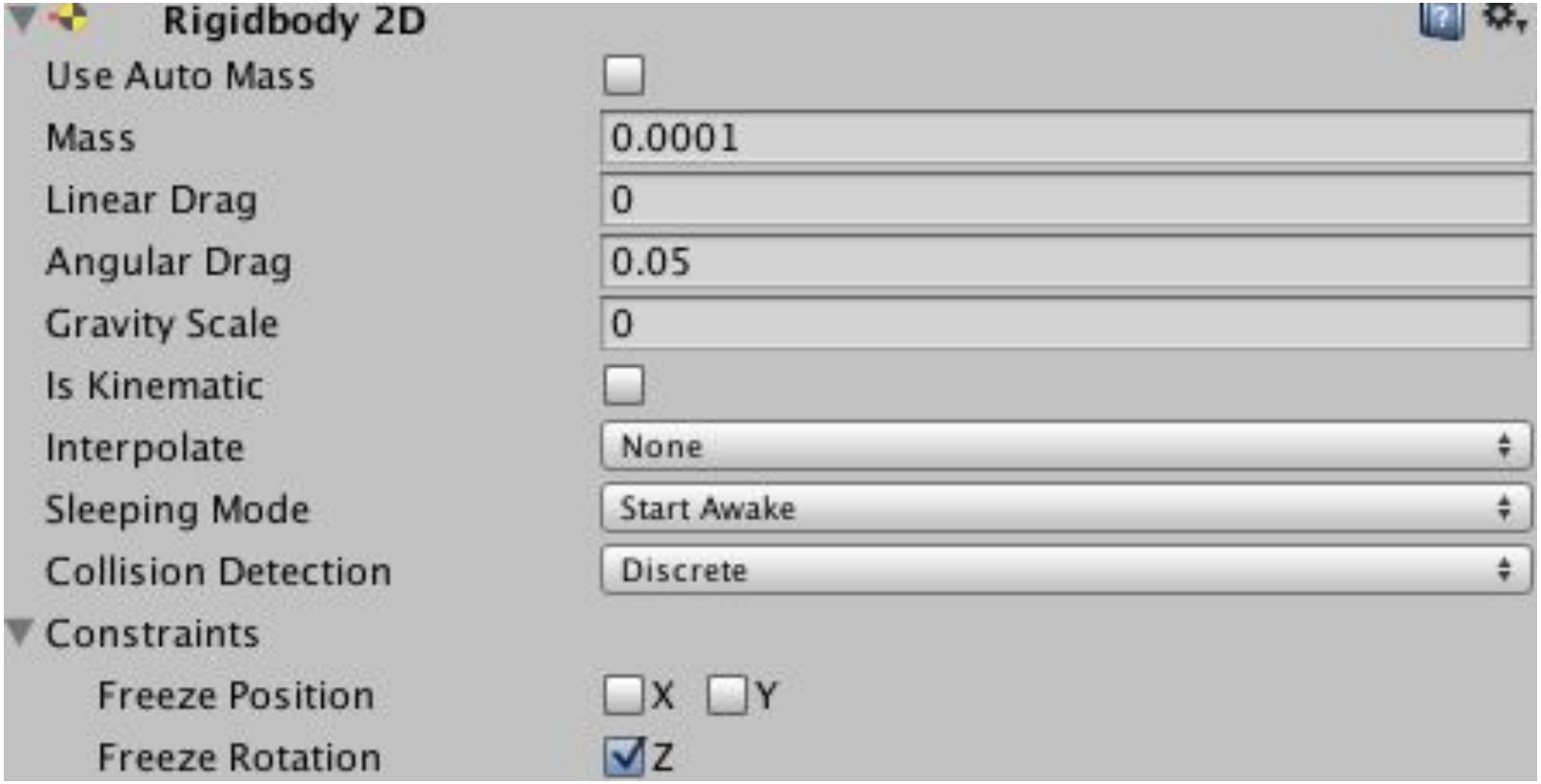


Затем перетягиваем материал из **Project Area** в слот материалов **Ball's Collider**

Мячик

Для того, чтобы мячик правильно взаимодействовал с игровым миром, добавим **Rigidbody2D.**
Add Component->Physics 2D->Rigidbody 2D
Настроим компоненту следующим образом

Мячик


Rigidbody 2D
Use Auto Mass
Mass
0.0001
Linear Drag
0
Angular Drag
0.05
Gravity Scale
0
Is Kinematic
Interpolate
None
Sleeping Mode
Start Awake
Collision Detection
Discrete
Constraints
Freeze Position
X
Y
Freeze Rotation
Z

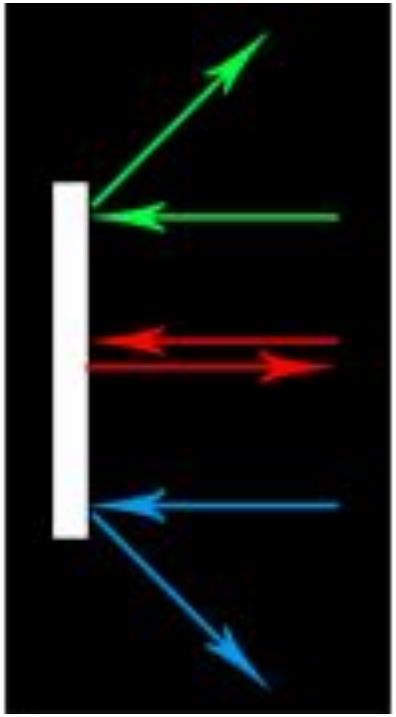
Мячик

Далее добавим скрипт **Add Component->New Script,** назовем его **Ball**.

```
using UnityEngine;
using System.Collections;

public class Ball : MonoBehaviour {
    public float speed = 30;

    void Start() {
        // Initial Velocity
        GetComponent<Rigidbody2D>().velocity =
Vector2.right * speed;
    }
}
```

Мячик

Если запустить игру, шарик будет не правильно отражаться от ракеток, он будет взаимодействовать линейно.



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

public class Ball : MonoBehaviour {
    public float speed = 30;
    // Use this for initialization
    void Start () {
        //initial velocity
        GetComponent<Rigidbody2D>().velocity = Vector2.right * speed;
    }

    void OnCollisionEnter2D(Collision2D col)
    {
        // Note: 'col' holds the collision information. If the
        // Ball collided with a racket, then:
        //   col.gameObject is the racket
```

```
//    col.transform.position is the racket's position
//    col.collider is the racket's collider
if (col.gameObject.name == "RaketLeft")
{
    // Calculate hit Factor
    float y = hitFactor(transform.position,
                        col.transform.position,
                        col.collider.bounds.size.y);

    // Calculate direction, make length=1 via .normalized
    Vector2 dir = new Vector2(1, y).normalized;

    // Set Velocity with dir * speed
    GetComponent<Rigidbody2D>().velocity = dir * speed;
}
if (col.gameObject.name == "RacketRight")
```

```
    {
        // Calculate hit Factor
        float y = hitFactor(transform.position,
                            col.transform.position,
                            col.collider.bounds.size.y);

        // Calculate direction, make length=1 via .normalized
        Vector2 dir = new Vector2(-1, y).normalized;

        // Set Velocity with dir * speed
        GetComponent<Rigidbody2D>().velocity = dir * speed;
    }
}
float hitFactor(Vector2 ballPos, Vector2 racketPos, float racketHeight)
{
    // ascii art:
    // ||  1 <- at the top of the racket
```

```
        // ||
        // ||  0 <- at the middle of the racket
        // ||
        // || -1 <- at the bottom of the racket
        return (ballPos.y - racketPos.y) / racketHeight;
    }

}
```