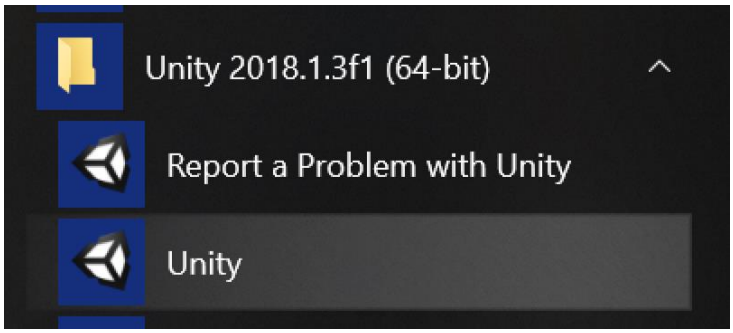
沉浸式环境教程

统一

目标：

- 碰撞
- 触发的事件
- 不断变化的场景
- 简单的 npc

加载统一

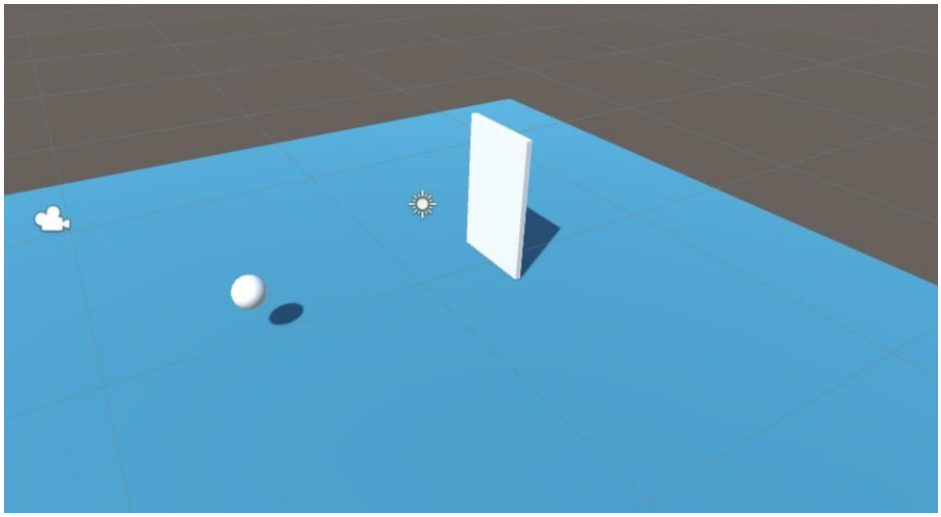


生成对象: 统一中的冲突

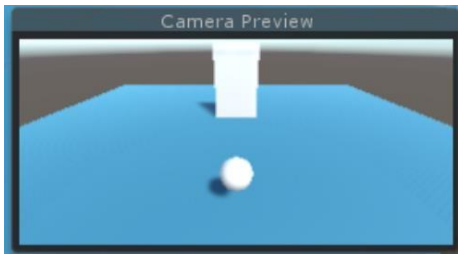
目的: 构建各种元素并测试不同的碰撞类型。

开门

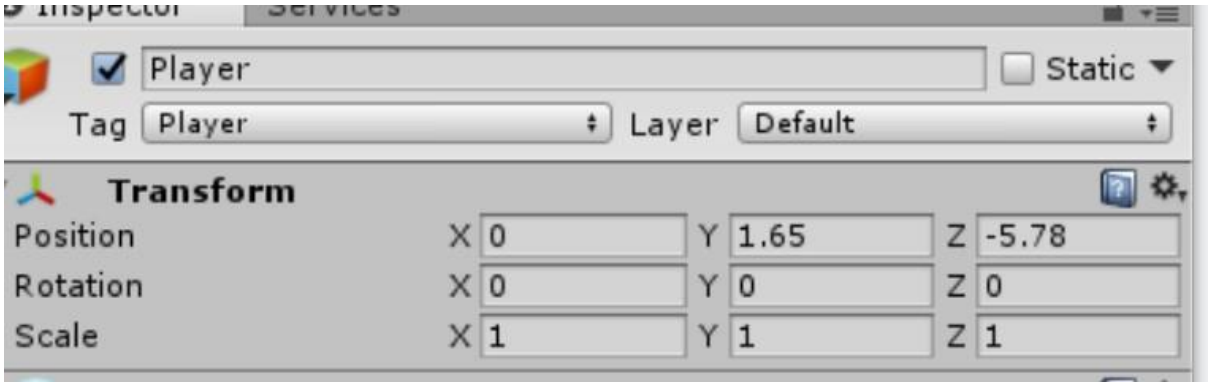
构建以下场景



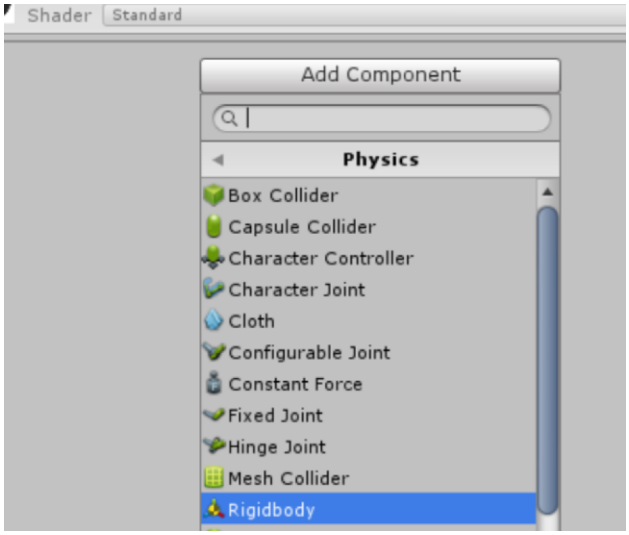
主摄像头是这样排列的:



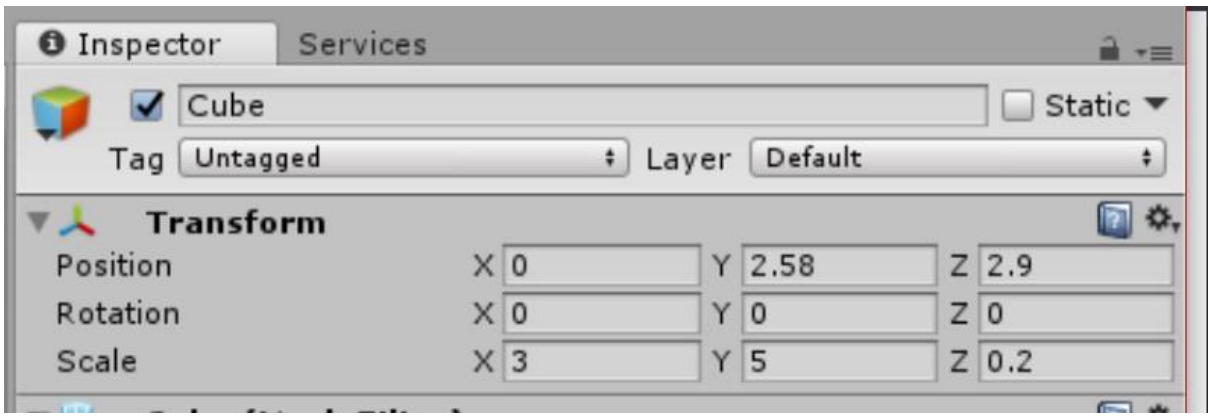
球体被称为玩家, 具有以下属性



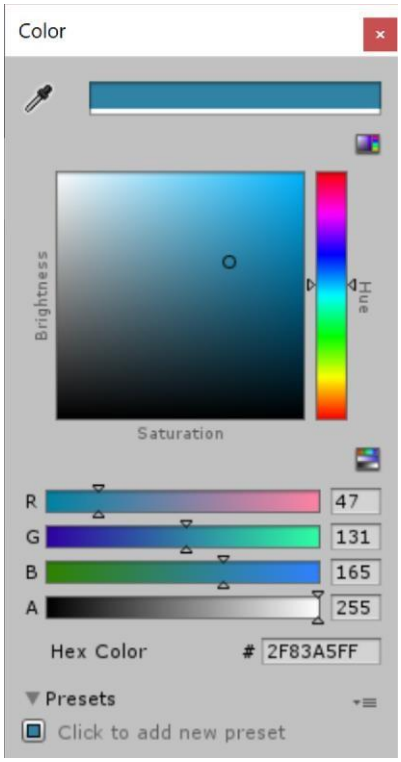
向播放机添加刚体。 组件→物理→刚性



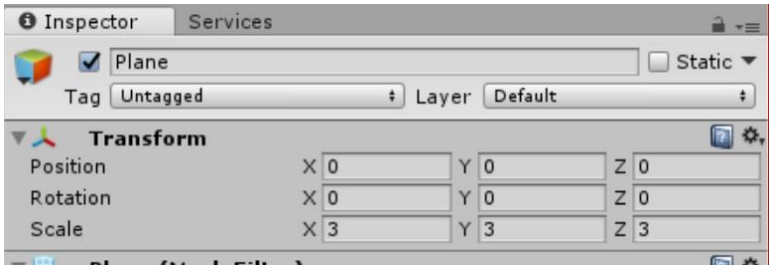
立方体的形状像一扇门, 具有以下属性



地板上的材料涂在上面

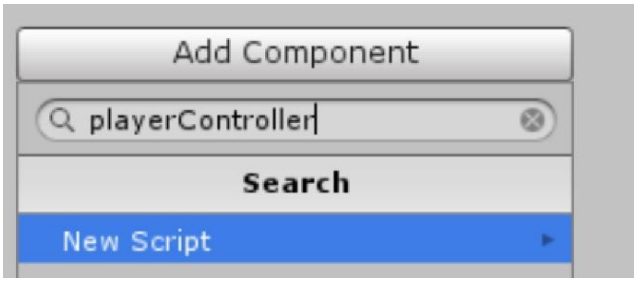


和以下信息



现在我们有了基本的布局, 我们将添加代码来移动玩家。

创建一个名为 "播放器控制器" 的新脚本, 并将其应用于播放器。



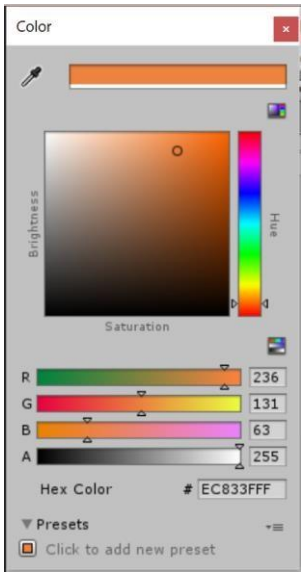
将其放在脚本文件夹中, 然后在可视工作室中打开它, 并应用以下代码


```
5   public class playerController : MonoBehaviour {
6
7       private Rigidbody rb;
8       public float speed = 7.0f;
9
10      // Use this for initialization
11      void Start () {
12          rb = GetComponent<Rigidbody>();
13      }
14
15      // Update is called once per frame
16      void FixedUpdate () {
17          float moveHorizontal = Input.GetAxis("Horizontal");
18          float moveVertical = Input.GetAxis("Vertical");
19
20          Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);
21          rb.AddForce(movement * speed);
22      }
23  }
```

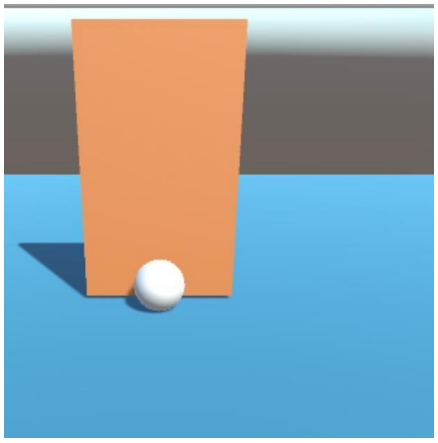
在 unity 中保存和测试。

为了确保玩家球体在朝向立方体时可见, 我们将更改立方体的颜色。

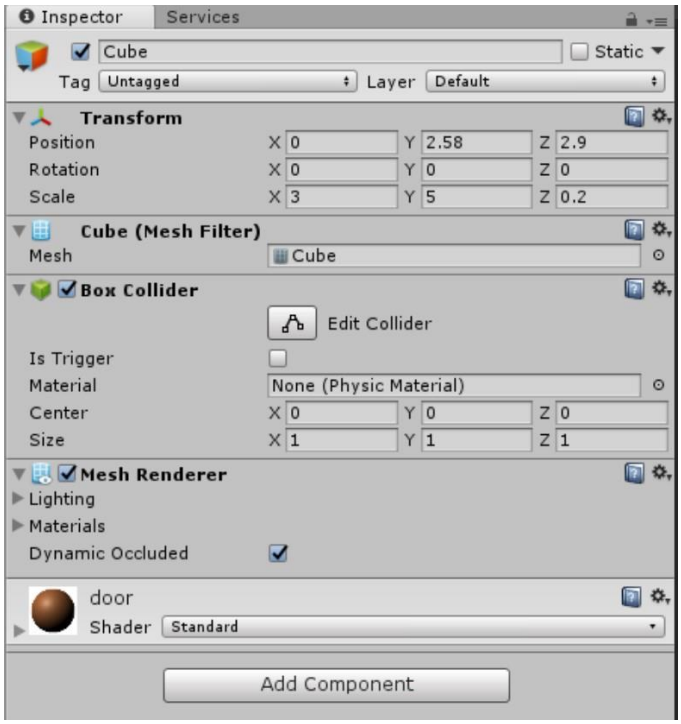
选择的颜色是:



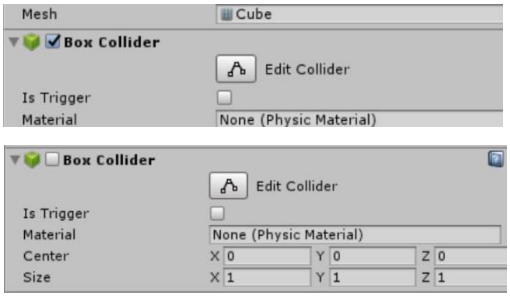
现在, 我们有能力看到球体接近门的时候, 我们需要实现一个碰撞区域和一个到了那里时发生的效果, 在这种情况下, 我们就会移动门。



现在, 让我们检查多维数据集上的检查器。



转换是预期的, 没有什么不寻常的是, 立方体网滤波器是很好, 但是请注意它是如何自动拥有箱 对撞机 它是积极的, 如果你使箱体对撞机处于活动状态, 即取下刻度, 然后播放。您将看到玩家能够直接通过 中门对象。



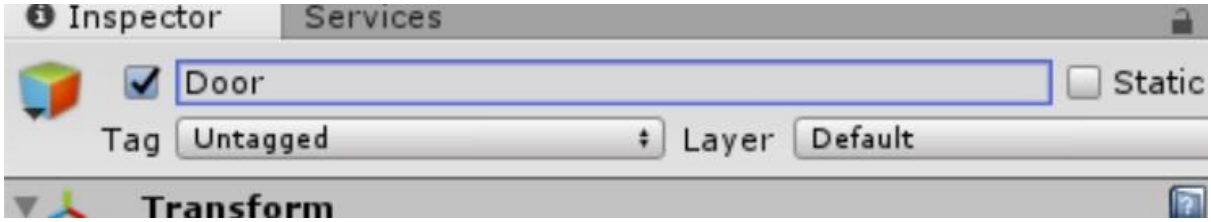
这样做并进行测试

重新打开箱式对撞机。

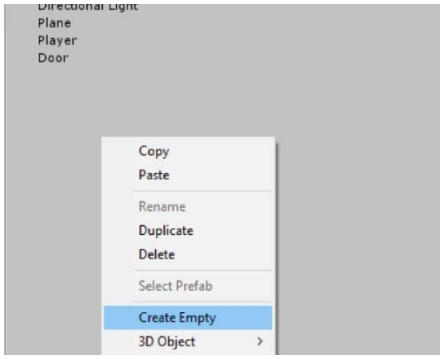
所以, 对撞机是让我们的物体相互互动的原因。

计划是在门对象周围添加一个对撞机 (现在是将立方体重命名到门的好时机), 并让它触发事件。

将多维数据集重命名为 "门"。



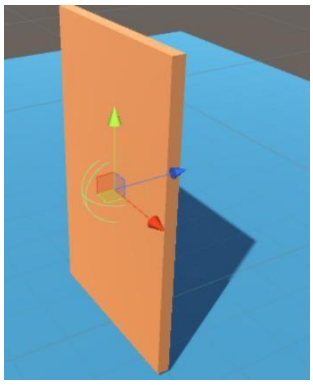
右键单击层次结构并添加一个空游戏对象。

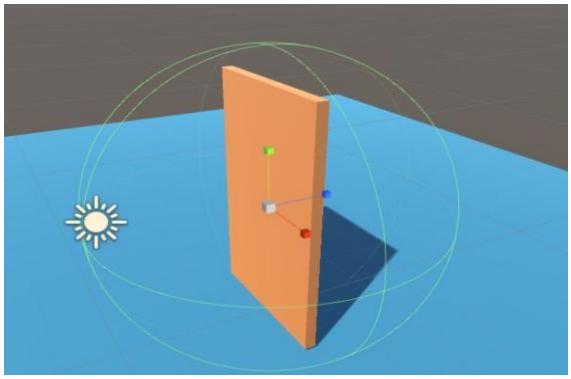


从这里, 我们添加一个 sphere 对撞机在这个对象上, 重命名从游戏对象到触发器门的对象。

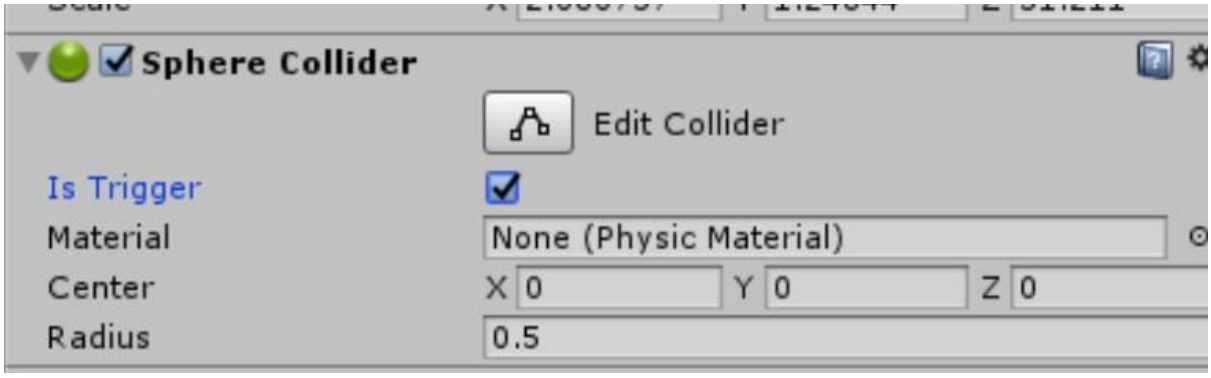
在层次结构中拖动门对象上的触发器门, 以这种方式创建一个父/子链接。

现在, 当看着现场, 球体对撞机太小, 无法使用, 因为我们希望它触发之前, 玩家打的门, 这样, 扩大它。

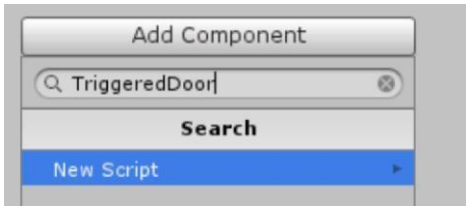


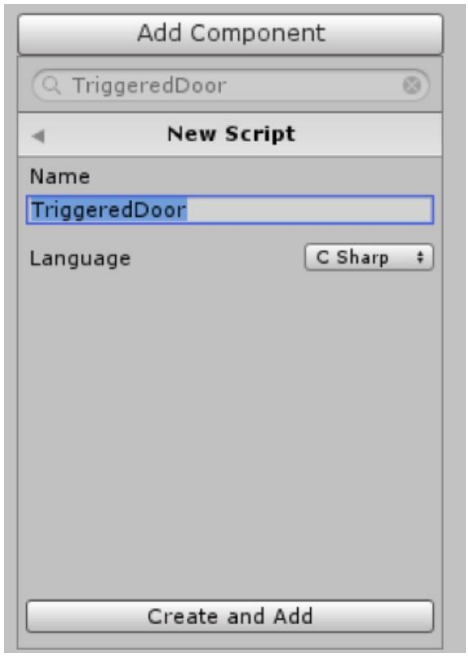


在检查器中, 我们需要改变球体对撞机的属性, 所以它是一个触发器, 而不是一个对撞机。否则我们的玩家将无法通过门区。所以, 在球体对撞机上, 检查是触发框。

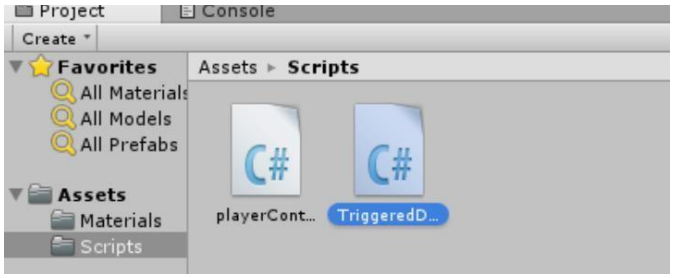


现在我们已经准备好了, 让我们向触发器门对象添加一个脚本。





将创建的脚本移动到脚本文件中, 然后将其打开视觉工作室并添加以下代码。

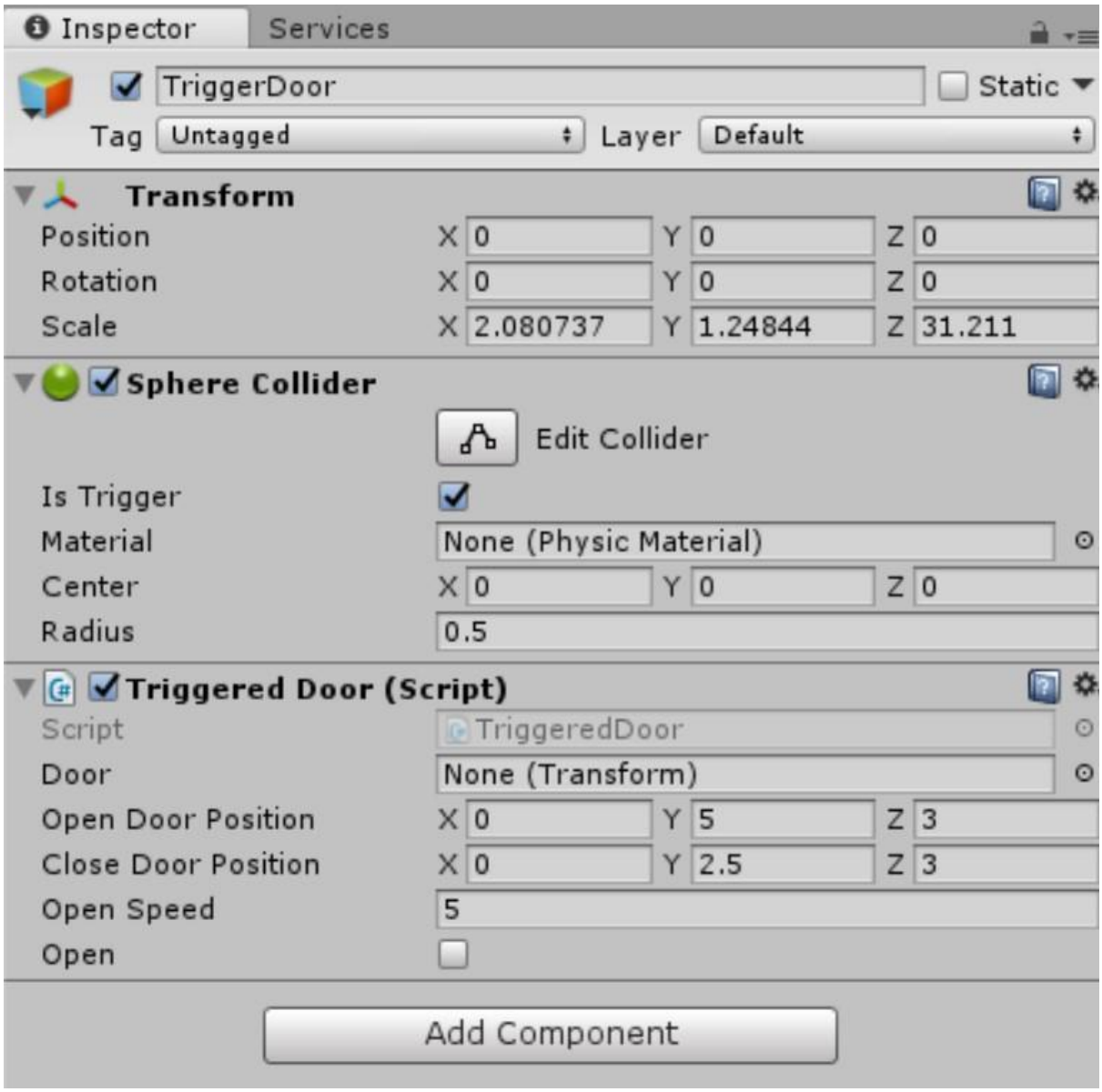


```
public class TriggeredDoor : MonoBehaviour {

    public Transform door;
    public Vector3 openDoorPosition = new Vector3(0.0f,5.0f,3.0f);
    public Vector3 closeDoorPosition = new Vector3(0.0f,2.5f,3.0f);

    public float openSpeed = 5.0f;
    public bool open;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        if (open)
        {
            door.position = Vector3.Lerp(door.position, openDoorPosition, Time.deltaTime * openSpeed);
        }
        else
        {
            door.position = Vector3.Lerp(door.position, closeDoorPosition, Time.deltaTime * openSpeed);
        }
    }
```

```
    private void OnTriggerEnter(Collider other)
    {
        if(other.tag == "Player")
        {
            TriggerOpen();
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if(other.tag == "Player")
        {
            TriggerClose();
        }
    }

    public void TriggerOpen()
    {
        open = true;
    }
    public void TriggerClose()
    {
        open = false;
    }
}
```

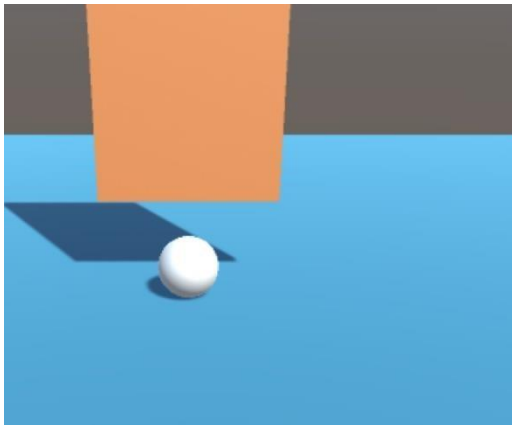
保存这一点, 然后回到统一。

确保脚本附加到触发器门对象



查看脚本时, 请注意转换门代码与 "无" 对齐的方式, 这是在等待指向实际对象的链接, 将门对象拖入其中或使用目标定位对象。

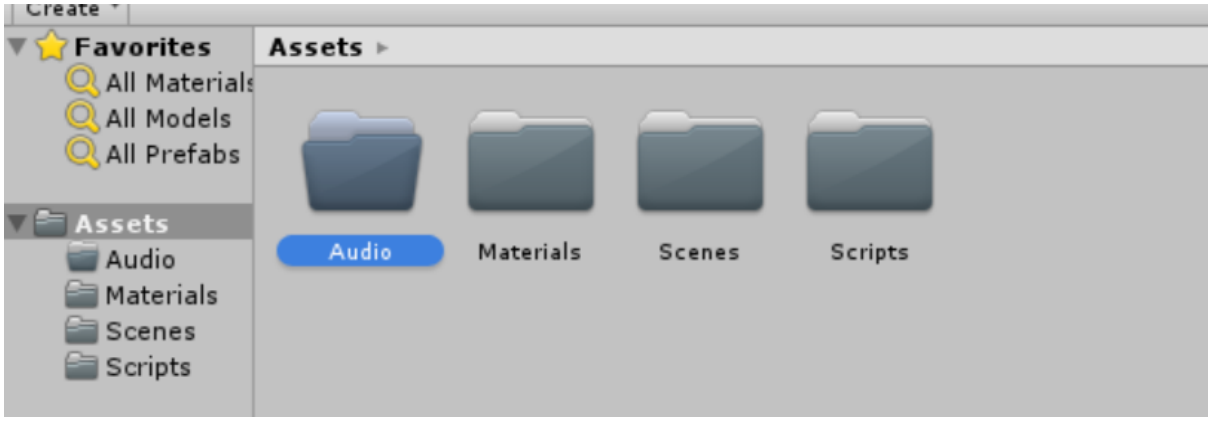
运行程序, 你应该看到的门抬起允许玩家球体在它下面移动。



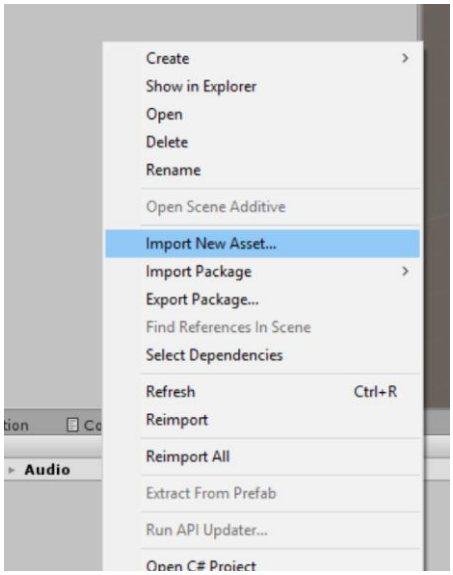
在测试速度5没有顺利打开门的时候, 当我把打开的速度降到1时, 门的打开就更顺畅了。

现在, 我们有了动画, 让我们添加一些音频到现场。从 I @ g 中获取音频文件, 或者如果您有一个免费的. org 帐户, 您可以从这里获得相同的声音 :https://freesound.org/people/mredig/sounds/120557/

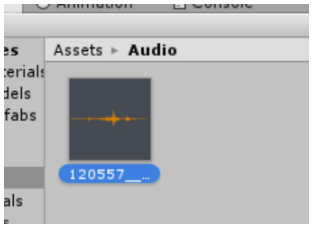
从添加音频文件夹开始, 然后将音频导入到系统中



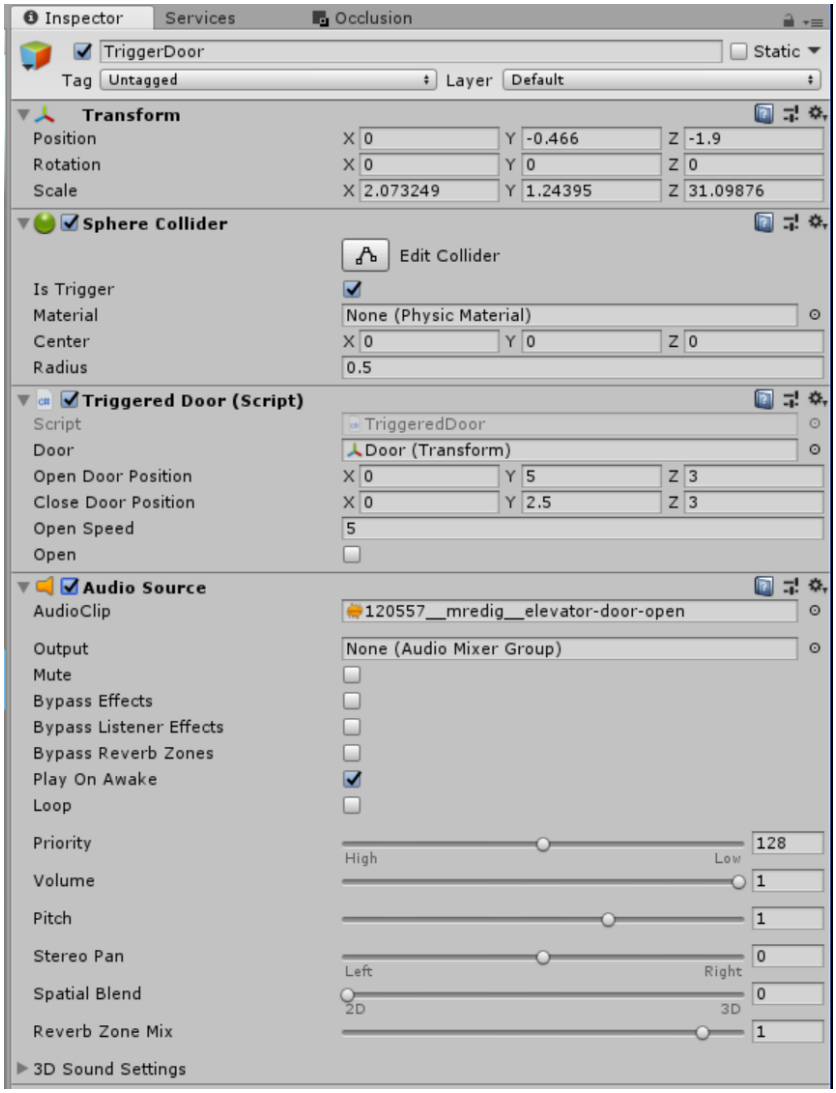
在音频文件夹中右键单击并导入新资产



音频文件夹现在将如下所示:

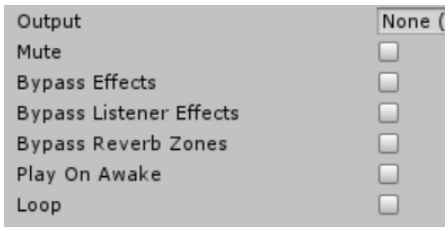


在触发器门对象上, 拖动音频文件, 它应该如下所示



现在, 如果你运行的游戏, 你会听到噪音开始, 这是比在醒着的时候使用这个游戏是有标记的。因此, 当游戏加载时, 它将运行音频文件。

为了阻止这种行为, 在醒着的时候不打游戏。



现在, 我们必须在触发器门上添加一些代码, 使音频能够激活一次我们打开门。

要做到这一点, 我们需要访问音频源, 并使其发生在正确的位置, 该位置是一旦我们开始运动的门。将代码修改为以下内容:

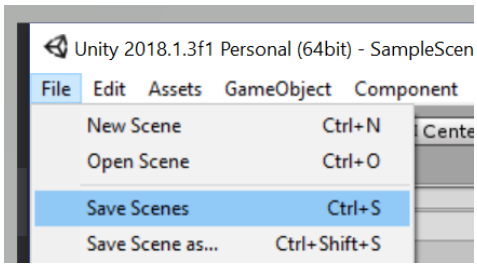
```
private void OnTriggerEnter(Collider other)
{
    AudioSource audio = GetComponent<AudioSource>();
    if (other.tag == "Player")
    {
        TriggerOpen();
        audio.Play();
    }
}

private void OnTriggerExit(Collider other)
{
    AudioSource audio = GetComponent<AudioSource>();
    if (other.tag == "Player")
    {
        TriggerClose();
        audio.Stop();
    }
}
```

我们两者兼而有之的原因播放 () 和 stop (), 以便我们可以对声音有更多的控制, 也取决于音频剪辑, 您可能需要修剪它, 使其发生在你想要的速度。

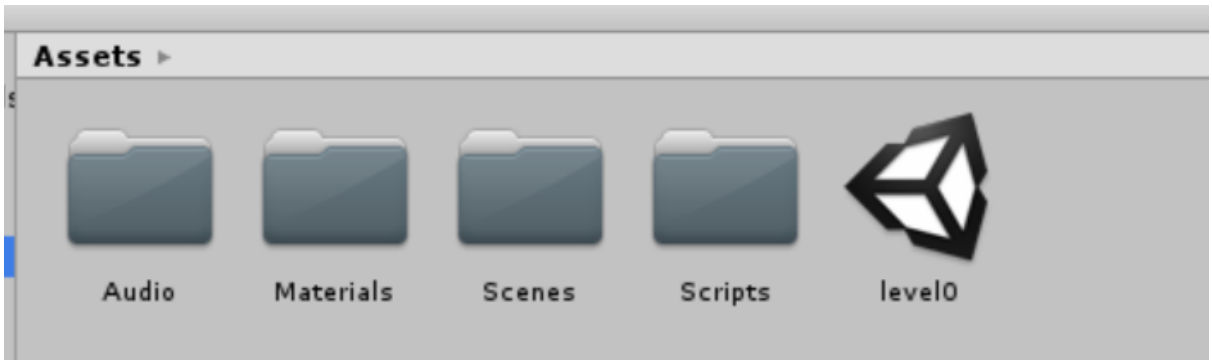
一旦编码, 保存和测试。

使用触发器更改场景

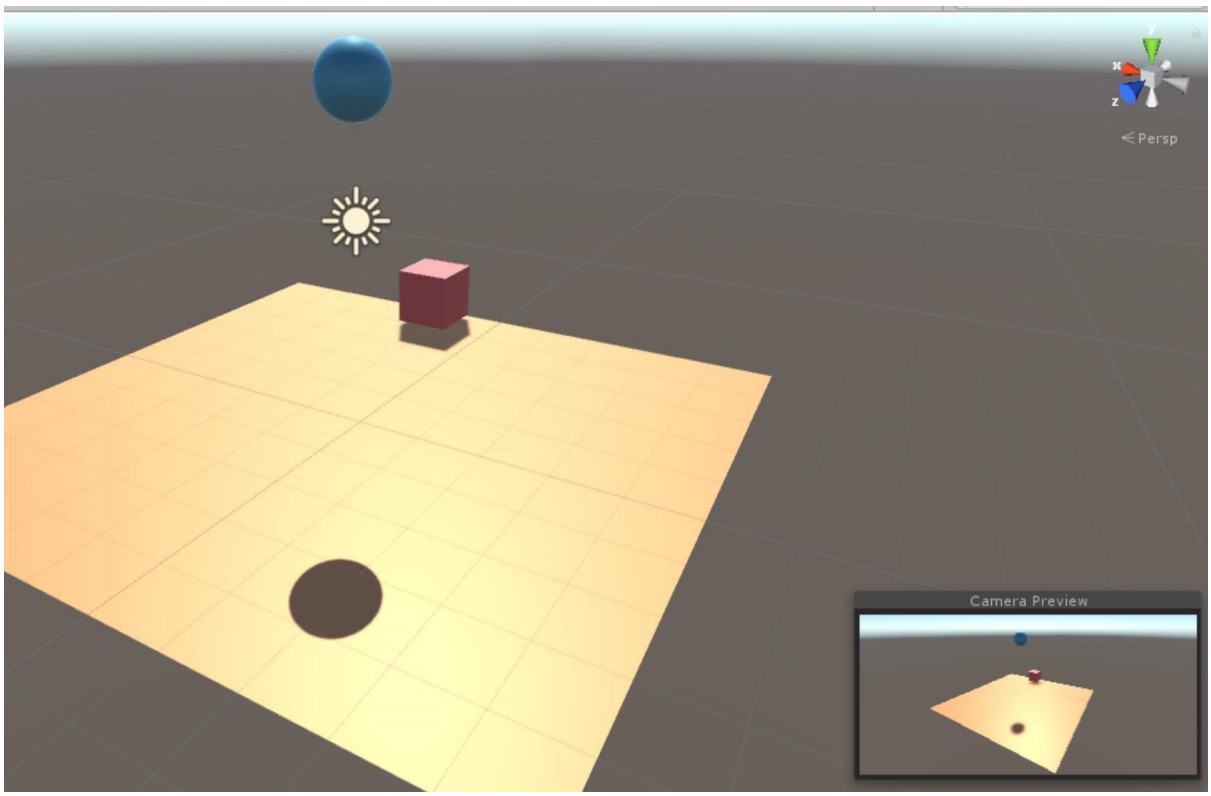
使用我们已经创建的相同场景, 将场景另存为 0级,



这将默认为资产文件夹;移动到 "场景" 文件夹。

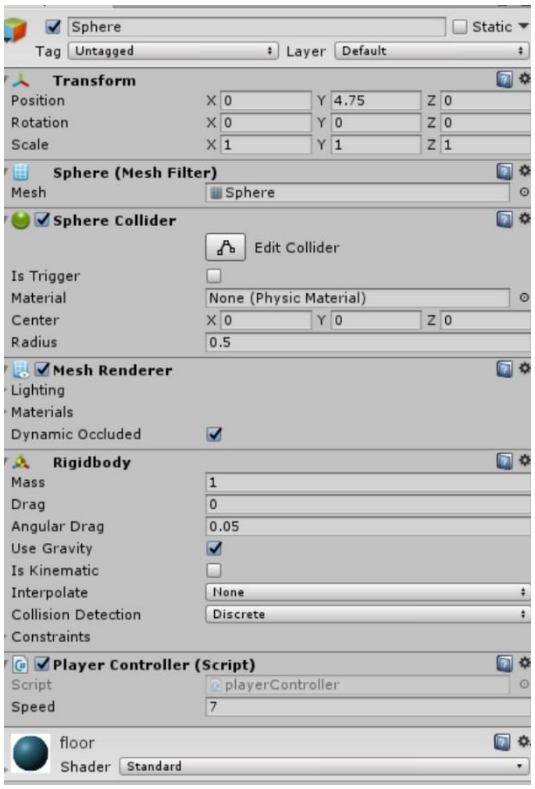


T母鸡创建一个新的场景如下, 并保存为1级。



地板材料被应用到玩家身上, 门材料被涂在飞机上, 一个简单的红色被涂在一个新的立方体上。

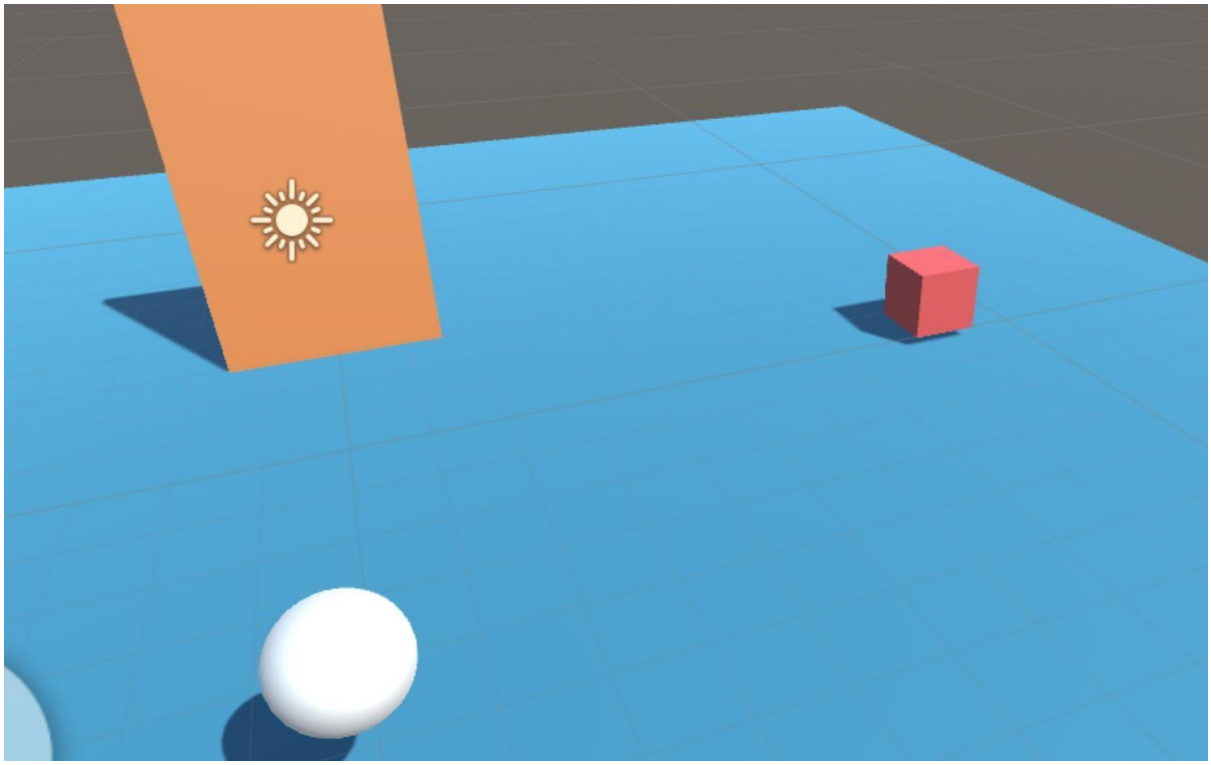
将脚本、播放器控制器连接到播放器, 以确保我们可以移动球体。



保存和测试, 如果相机的旋转关闭, 控件似乎倒置, 它的确定, 这只是为了测试的目的。

保存此场景并返回到0级。

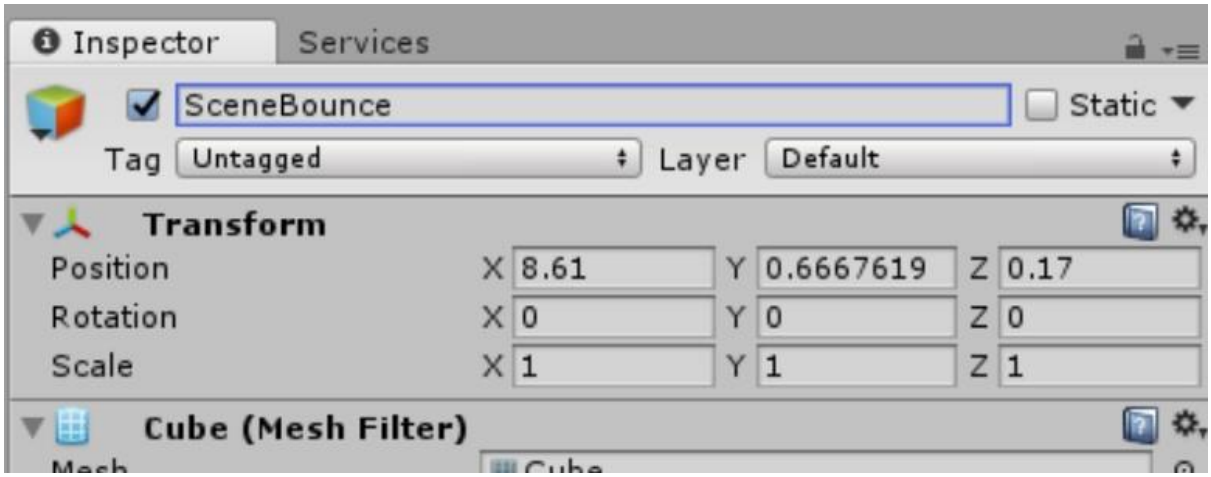
在0级上, 创建一个新多维数据集。并将其定位为这样。



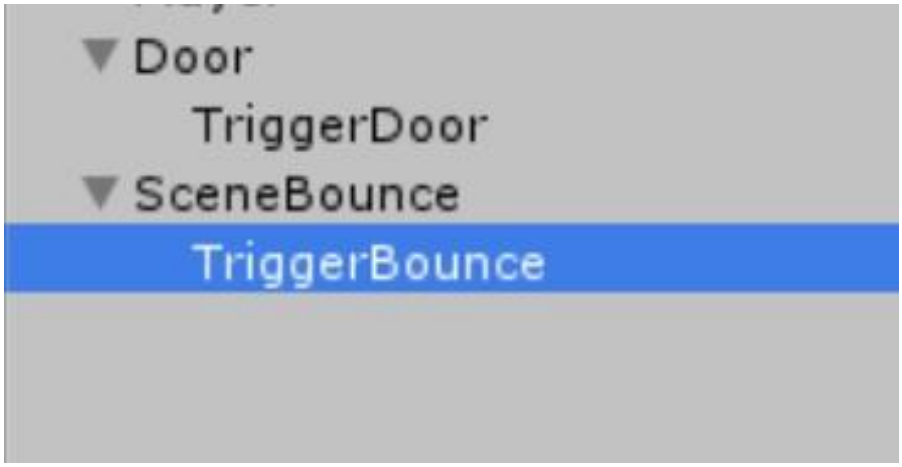
目标是分配一个对撞机, 就像门一样, 除了不是移动物体, 而是在场景之间跳来跳去。

从将这两个场景添加到生成设置开始。

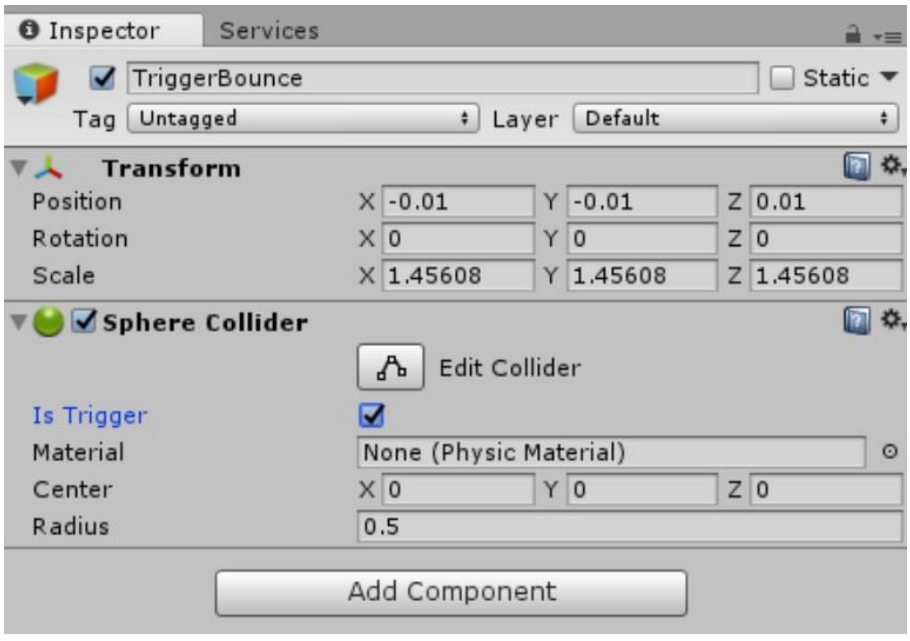
现在, 让我们创建一个新的空游戏对象, 就像我们以前一样, 将其命名为触发器弹跳并将其放置在多维数据集上, 然后向对象添加一个球体对撞机。将多维数据集重命名为 "场景弹跳"。



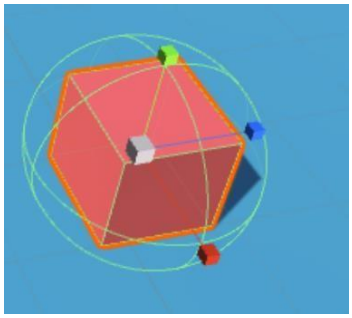
将 "触发器弹跳" 拖到 "场景弹跳" 上, 使其成为孩子。



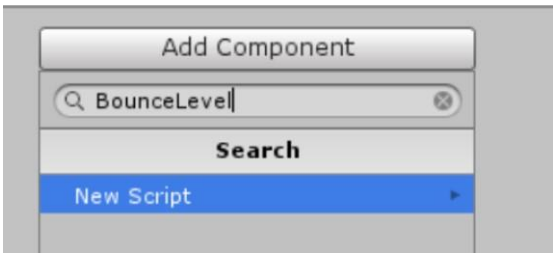
确保您已勾选触发触发触发触发触发

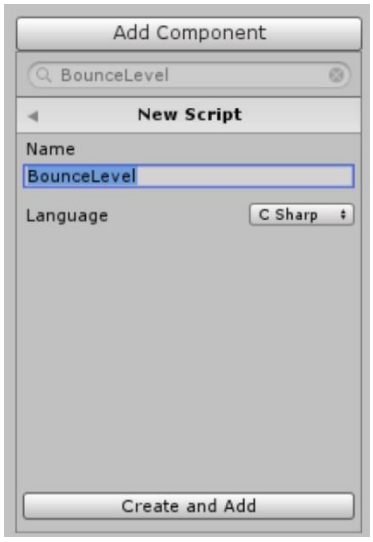


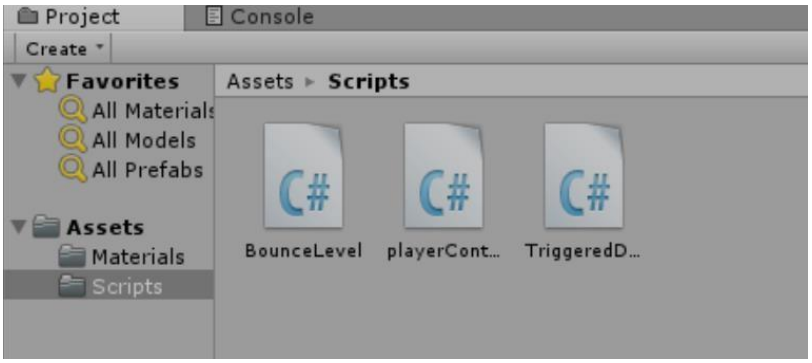
确保球面对撞机封装立方体。



在场景弹跳上添加一个名为 "bsecesalk" 的新脚本







一旦你做到了这一点, 移动 sc 「进入脚本文件夹, 并然后在 visual studio 中打开它。

添加以下代码。

```
6   public class BounceLevel : MonoBehaviour {
7
8       public int sceneIndex;
9
10      private void OnTriggerEnter(Collider other)
11      {
12          if (other.tag == "Player")
13          {
14              SceneManager.LoadScene(sceneIndex);
15          }
16      }
17  }
18
```

回到 unity, 对检查器进行更改, 以确保发送要更改的正确场景索引。

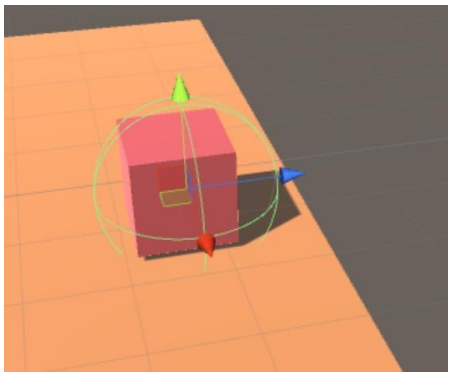
保存和测试, 你应该能够走到立方体中的球员, 并反弹到下一个场景。

在级别1上对另一个多维数据集重复此过程。

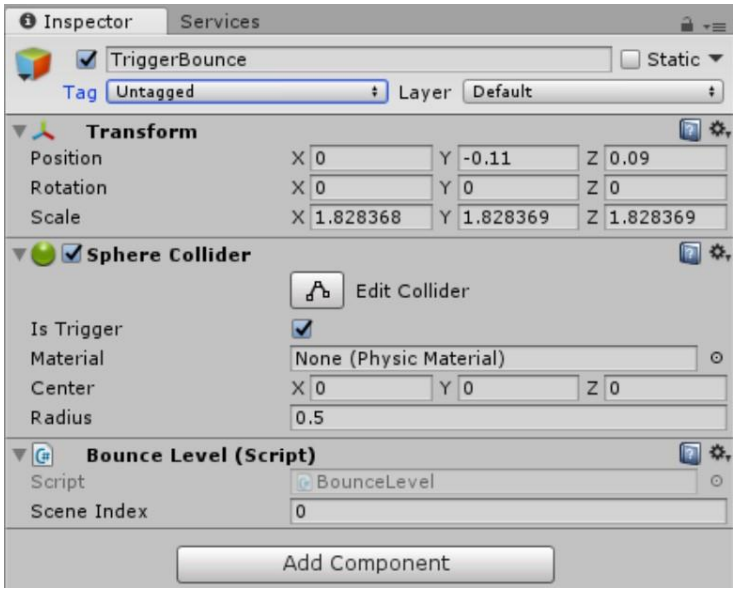
步骤如下:

- 空游戏对象
- 应用球面对撞机
- 确保已选中触发器
- 链接链接级脚本
- 确保应用了正确的场景索引。

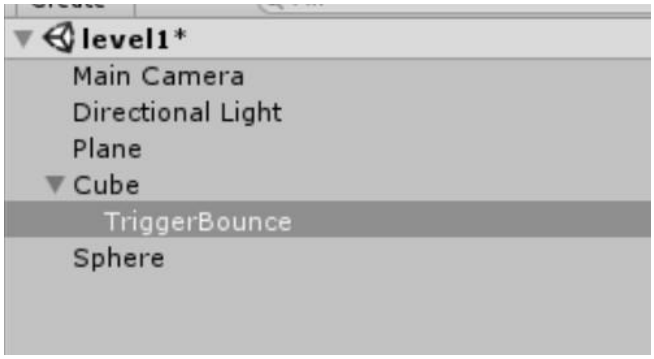
在现场



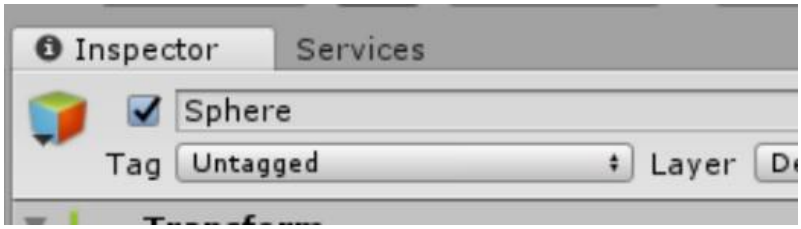
在检查器中

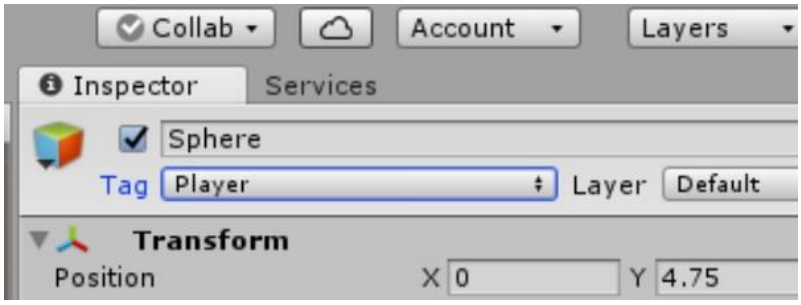


在层次结构中



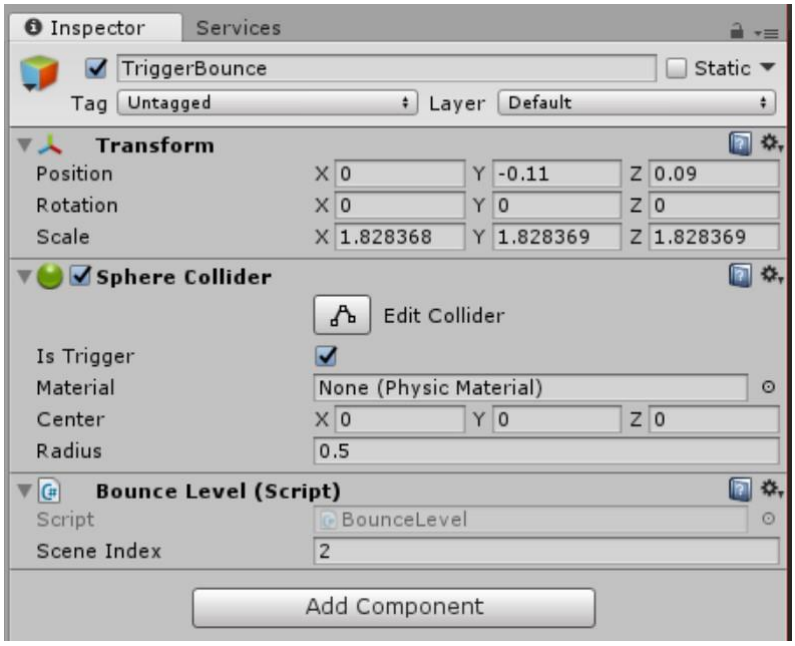
如果不起作用,请确保将球体标记为播放机。





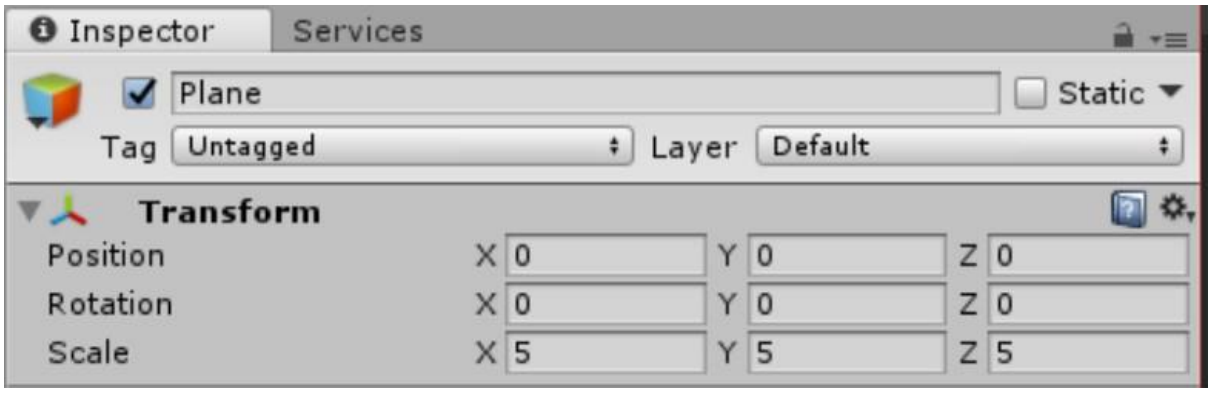
添加简单的 npc

使用相同的项目, 创建一个新场景并将其另存为级别 2, 然后将其添加到生成设置中, 并在1级场景中将弹跳级别从0更改为2。这样我们就可以从 0级-> 1 > 2

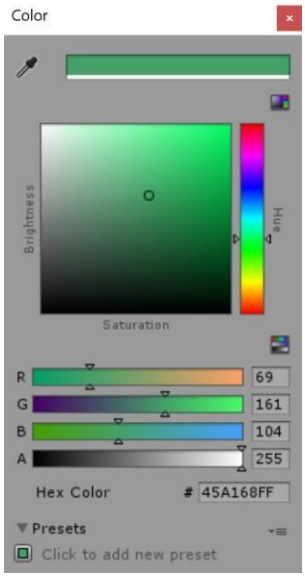


在2级上执行以下操作:

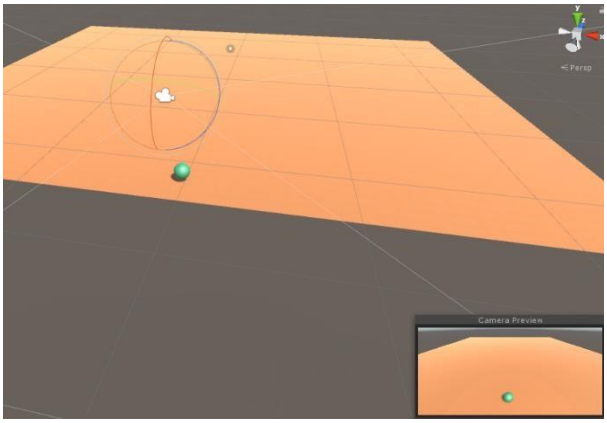
添加一个平面, 将其放大 5, 并在其上使用相同的门材料。



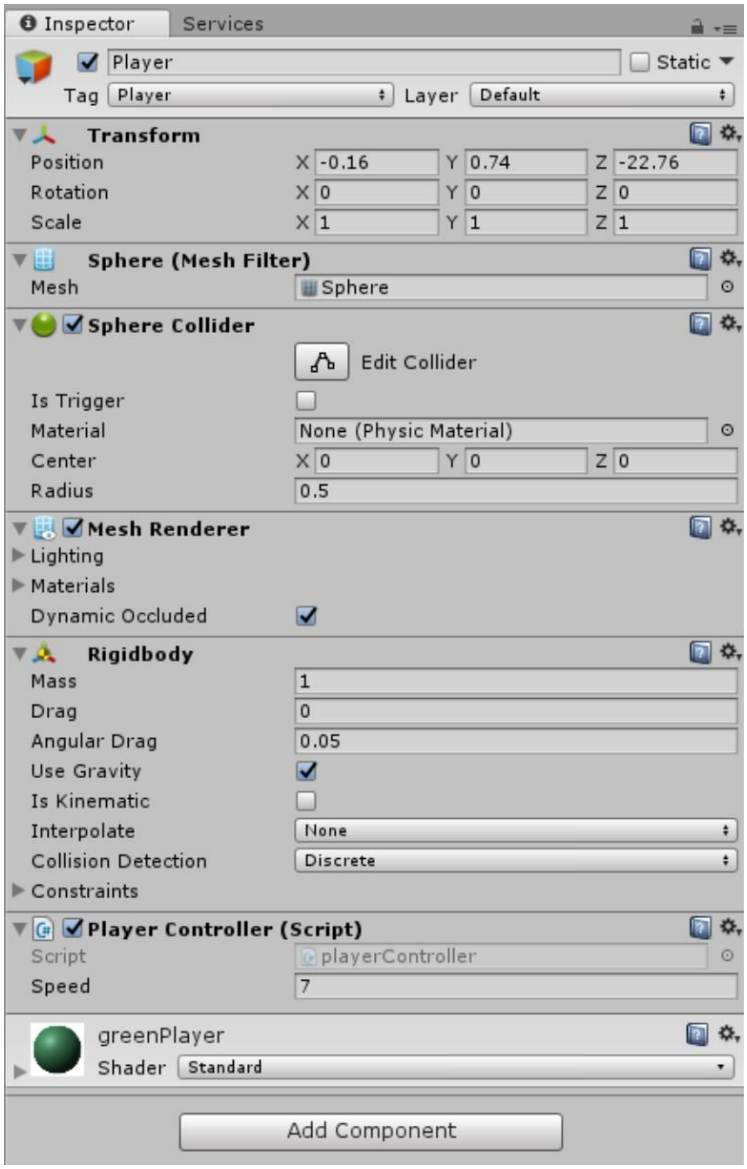
创建球体, 为其分配新的绿色材质。



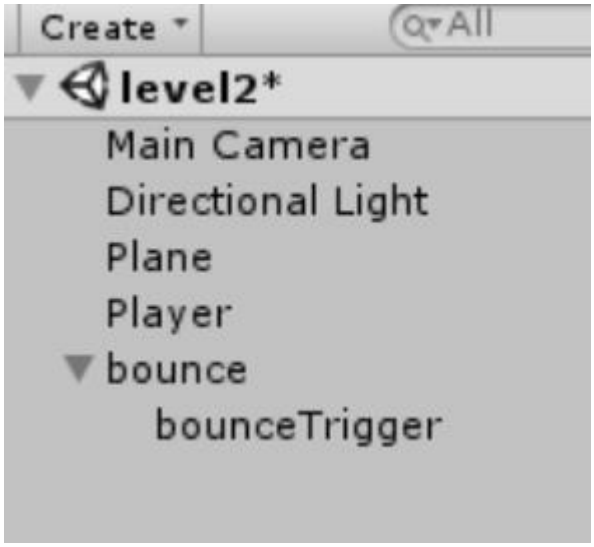
位置　　中　相机上方和后面的 p　　层, 所以我们可以看看是什么会发生的

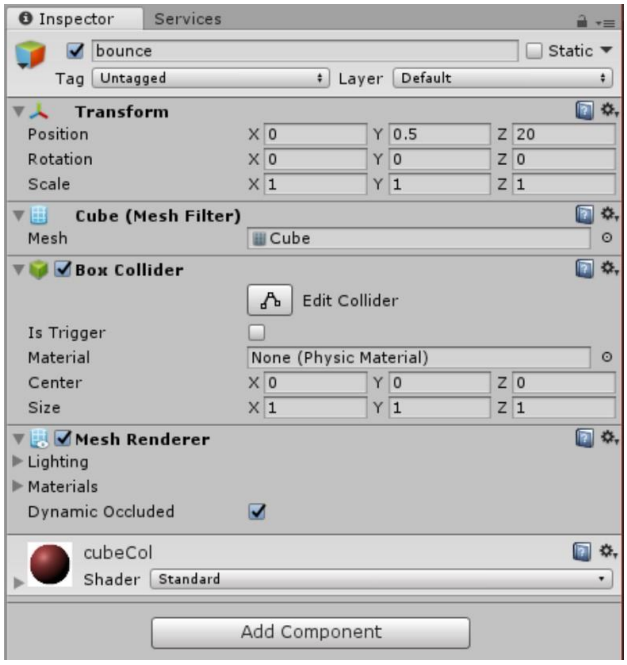


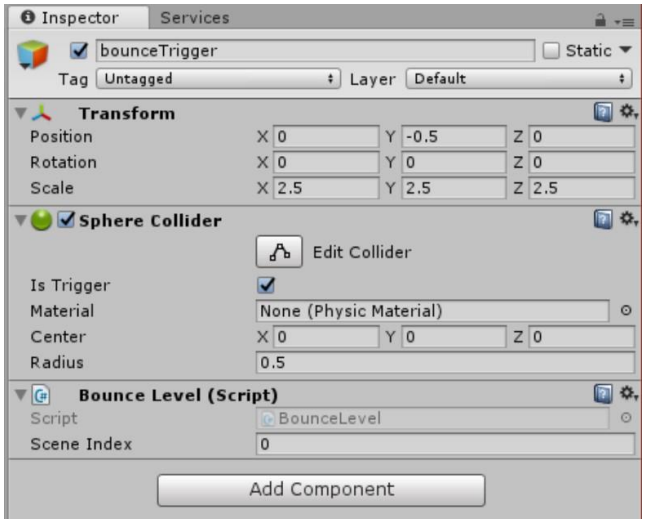
将刚体和运动脚本应用到玩家身上, 并确保球体被标记为玩家。



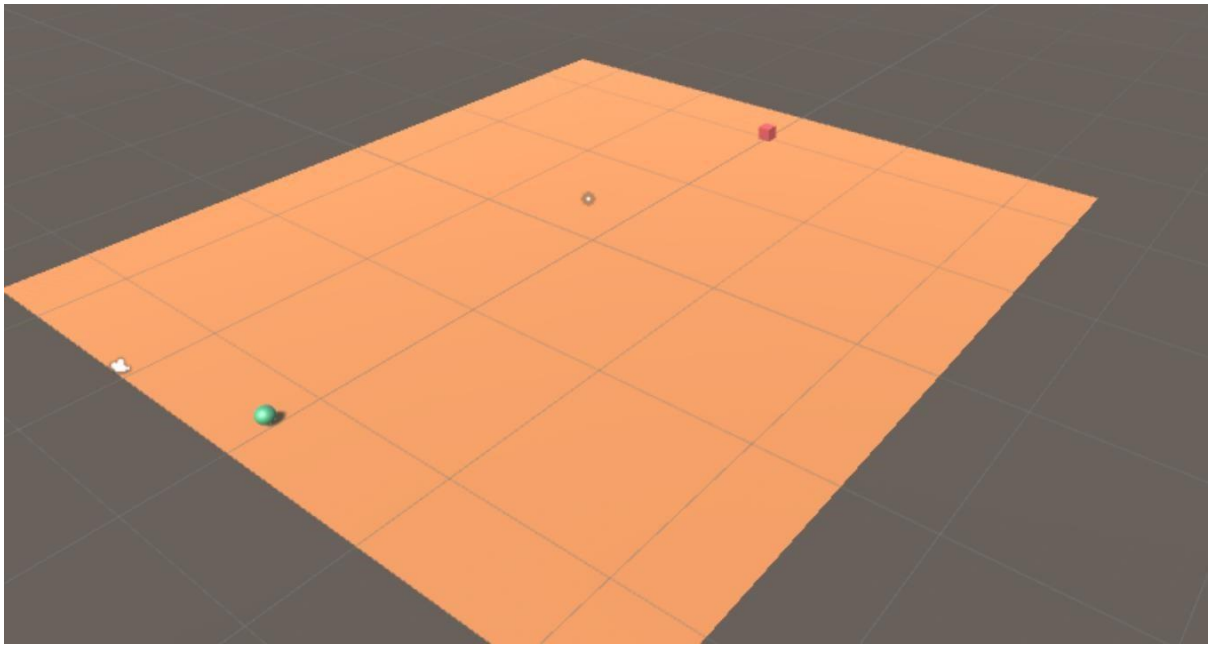
接下来创建一个多维数据集, 调用它反弹并应用一个空的游戏对象, 应用脚本使其, 这样, 如果玩家碰到它, 它会反弹到0级。这正是我们迄今已经做的。


Create
All
level2*
Main Camera
Directional Light
Plane
Player
bounce
bounceTrigger

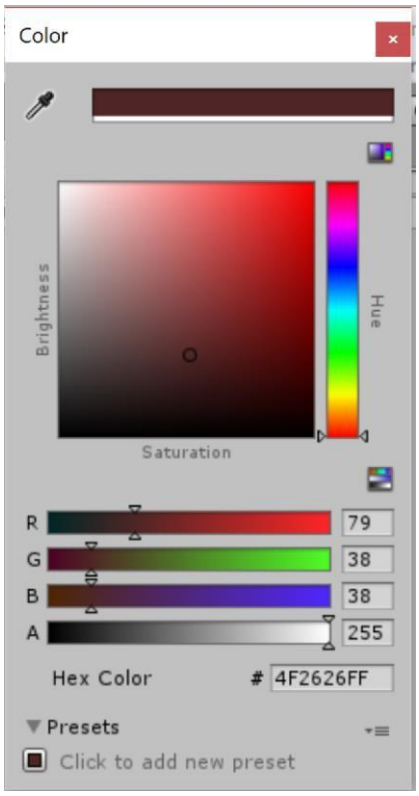

Inspector
Services
bounce
Static
Tag
Untagged
Layer
Default
Transform
Position
X 0
Y 0.5
Z 20
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1
Cube (Mesh Filter)
Mesh
Cube
Box Collider
Edit Collider
Is Trigger
Material
None (Physic Material)
Center
X 0
Y 0
Z 0
Size
X 1
Y 1
Z 1
Mesh Renderer
Lighting
Materials
Dynamic Occluded
cubeCol
Shader
Standard
Add Component

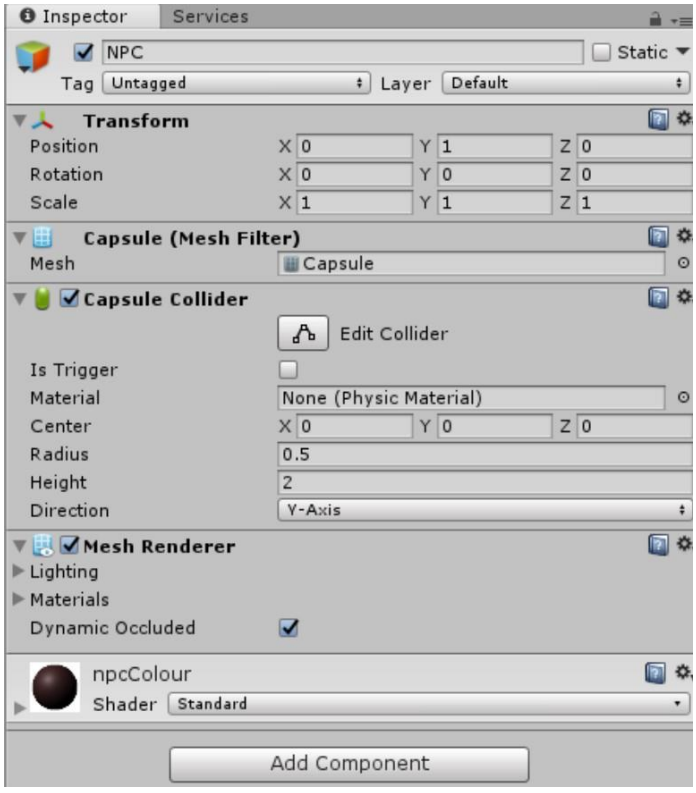

Inspector
Services
bounceTrigger
Static
Tag
Untagged
Layer
Default
Transform
Position
X 0
Y -0.5
Z 0
Rotation
X 0
Y 0
Z 0
Scale
X 2.5
Y 2.5
Z 2.5
Sphere Collider
Edit Collider
Is Trigger
Material
None (Physic Material)
Center
X 0
Y 0
Z 0
Radius
0.5
Bounce Level (Script)
Script
BounceLevel
Scene Index
0
Add Component

你有下面的场景



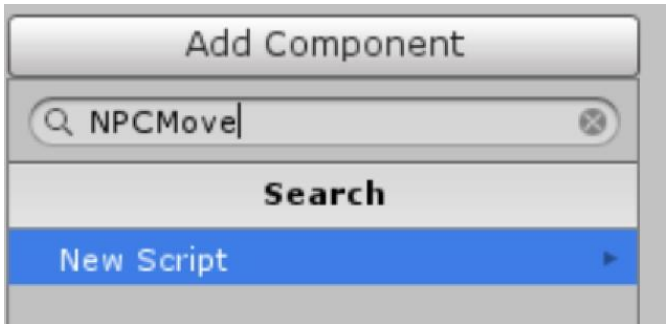
接下来, 我们添加一个 npc, 因为我们没有为此引入完全充实的字符, 创建一个胶囊, 命名它 npc, 并应用新的颜色。

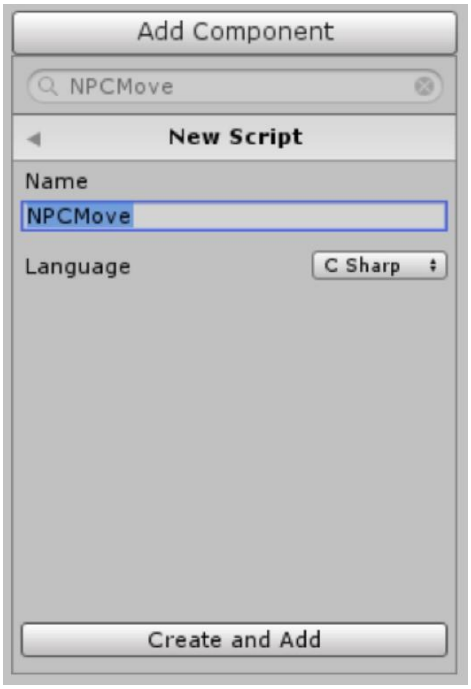




我们的 npc 将从一边到另一边, 如果他与球员相撞, 球员将被送回首发现场。

所以, 我们首先需要做的是创建一个 npc 可以移动的脚本。





将其移动到脚本文件夹中。

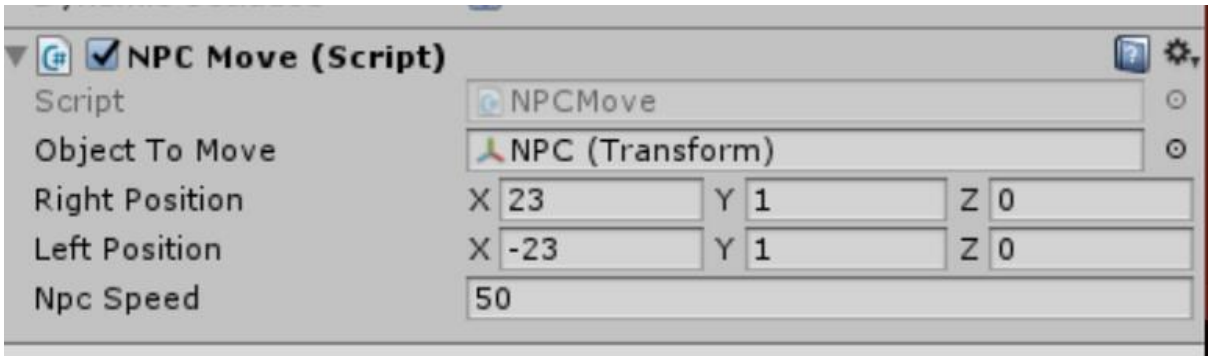
添加以下代码

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

public class NPCMove : MonoBehaviour {

    public Transform ObjectToMove;
    public Vector3 rightPosition = new Vector3(23.0f, 1.0f, 0.0f);
    public Vector3 leftPosition = new Vector3(-23.0f, 1.0f, 0.0f);

    public float npcSpeed = 50.0f;
    private bool moveLeft = true;
    private bool moveRight = false;

    // Update is called once per frame
    void Update () {
        if (moveLeft)
        {
            MoveLeftNPC();
        }
        else
        {
            MoveRightNPC();
        }
    }
```

```
26
27      void MoveLeftNPC()
28      {
29          ObjectToMove.position = Vector3.Lerp(leftPosition, ObjectToMove.position, Time.deltaTime * npcSpeed);
30          if(ObjectToMove.position == leftPosition)
31          {
32              moveLeft = false;
33              moveRight = true;
34          }
35      }
36      void MoveRightNPC()
37      {
38          ObjectToMove.position = Vector3.Lerp(rightPosition, ObjectToMove.position, Time.deltaTime * npcSpeed);
39          if (ObjectToMove.position == rightPosition)
40          {
41              moveRight = false;
42              moveLeft = true;
43          }
44      }
45  }
```

一旦完成, 保存, 然后回到团结。

在检查器中, 将 npc 对象分配给对象自移动脚本和测试的部分。您可能需要更改速度值。



你应该让胶囊从一边移动到另一边。

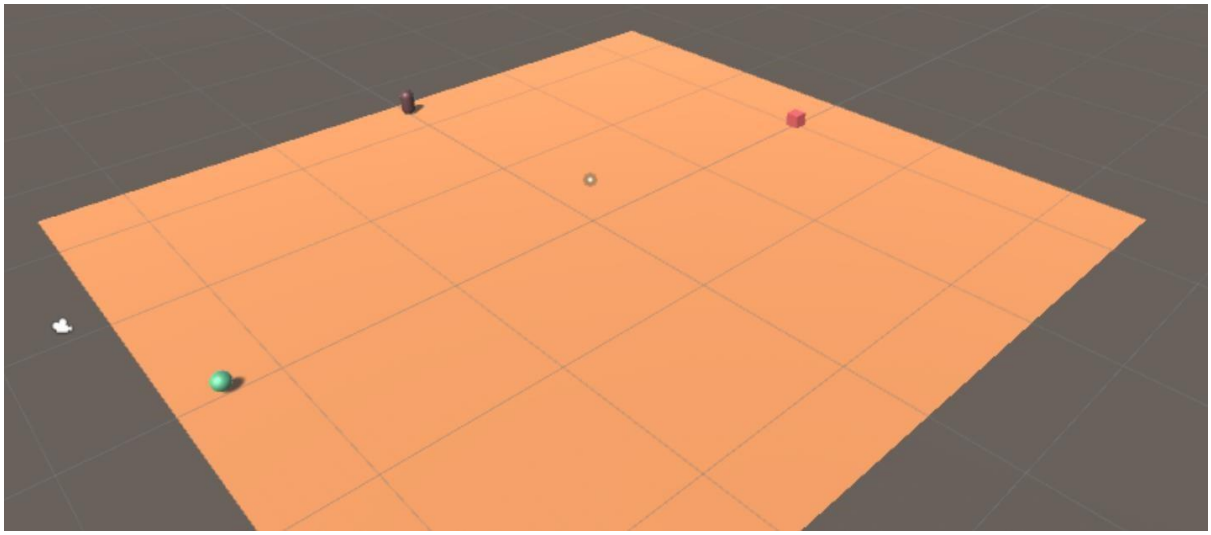
接下来, 将对撞机应用于 npc。

步骤如下:

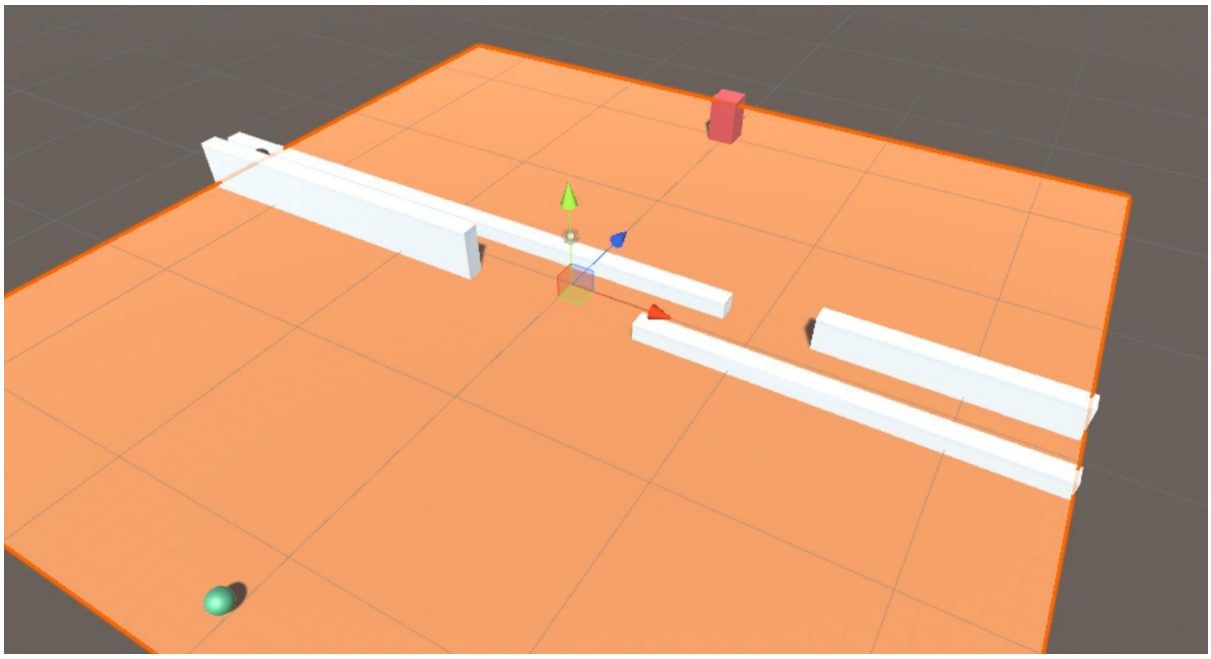
- 空游戏对象
- 应用球面对撞机
- 确保已选中触发器
- 链接链接级脚本
- 确保应用了正确的场景索引。

运行该程序, 你应该被迫回到主阶段, 如果你击中了 npc, 在董事会的两侧。您可以修改代码, 使 npc 根据需要在电路板上下移动, 但在这种情况下, 只是为了使其更具挑战性, 我们将添加一些多维数据集墙。

在墙壁之前



墙后



玩的速度, 对撞机的大小等, 这个简单的布局可以阻止用户达到最后的反弹。

当然, 你可以把球员带到一个新的场景, 祝贺他们通过关卡。