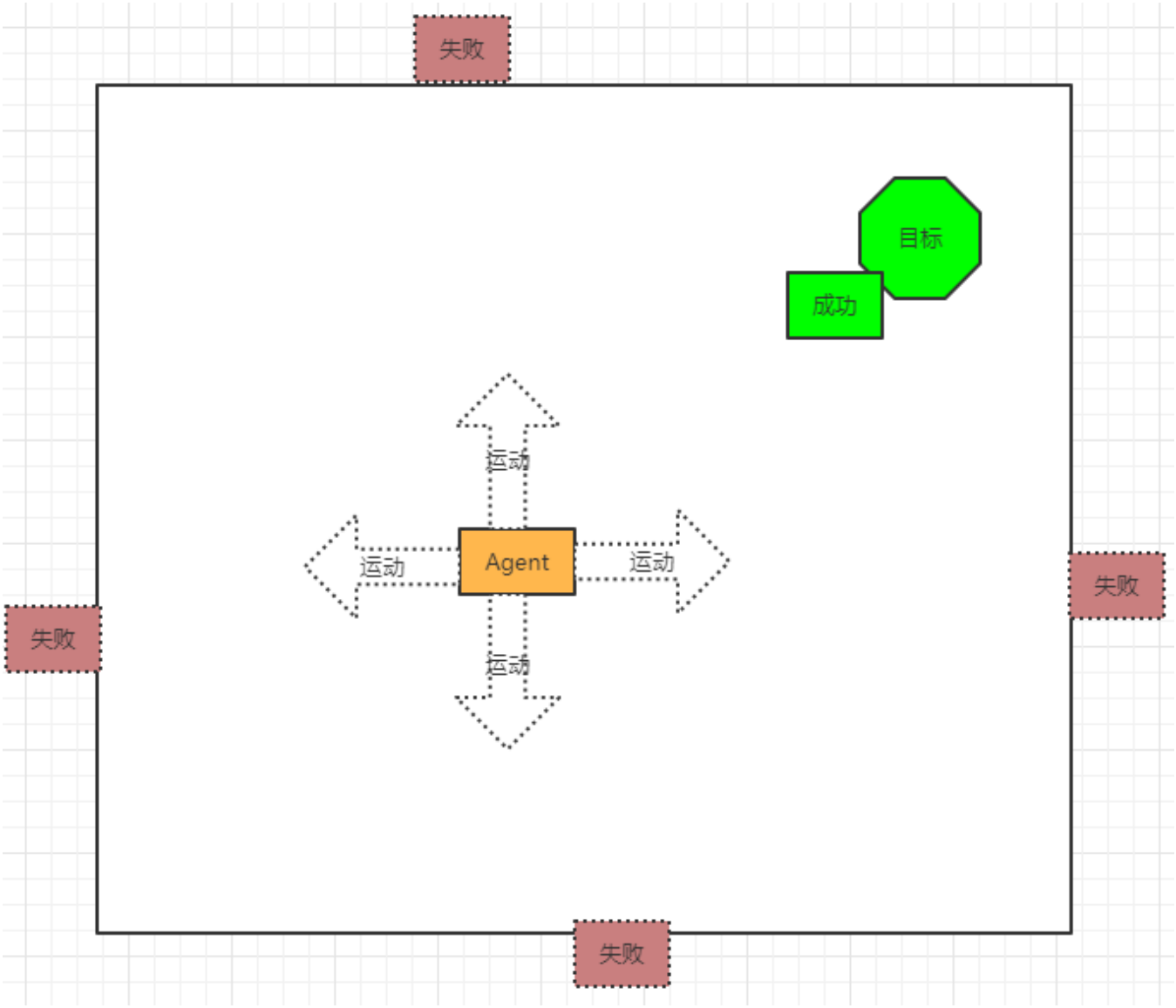
实战ml-angents：训练第一模型

一、简介

Unity Machine Learning Agents (ML-Agents) 是一款开源的 Unity 插件， 使得我们得以在游戏环境和模拟环境中训练智能 agent。您可以使用 reinforcement learning（强化学习）、imitation learning（模仿学习）、neuroevolution（神经进化）或其他机器学习方法， 通过简单易用的 Python API进行控制，对 Agent 进行训练。另外还提供最先进算法的实现方式（基于 TensorFlow），让游戏开发者和业余爱好者能够轻松地 训练用于 2D、3D 和 VR/AR 游戏的智能 agent。 这些经过训练的 agent 可用于多种目的， 包括控制 NPC 行为（采用各种设置， 例如多个 agent 和对抗）、对游戏内部版本进行自动化测试、以及评估不同游戏设计决策的预发布版本。ML-Agents 对于游戏开发者和 AI 研究人员双方 都有利，因为它提供了一个集中的平台， 使得我们得以在 Unity 的丰富环境中测试 AI 的最新进展， 并使结果为更多的研究者和游戏开发者所用。

之前我们已经介绍过如何搭建环境、如何运行官方给出的demo（文章地址），这次我们需要自己动手搭建一个训练环境，并且在这个环境中训练出能移动到正确位置的智能Angen。

二、设计拆解



我们需要设计模型，使得一个能沿着X,Z移动的Agent能正确地移动到目标点的位置。

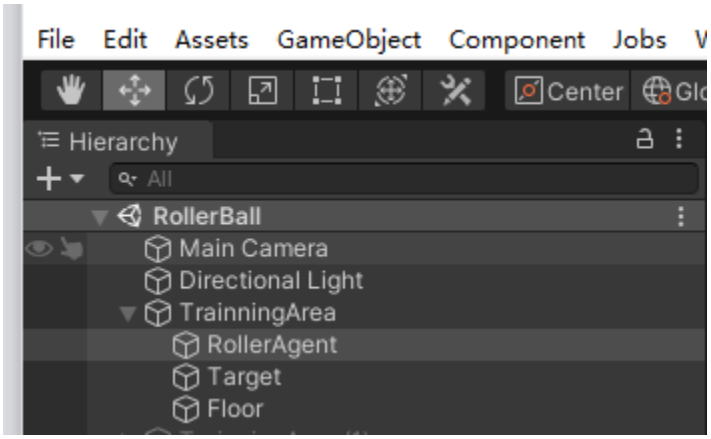
成功条件：离目标位置<1.42f

失败判定：超出范围

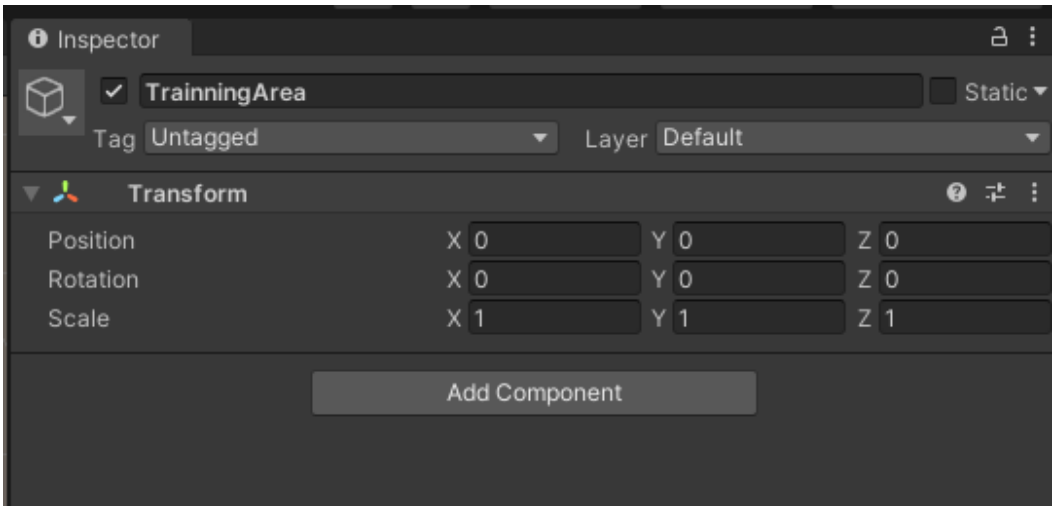
三、场景搭建

打开unity3d，新建场景RollerBall

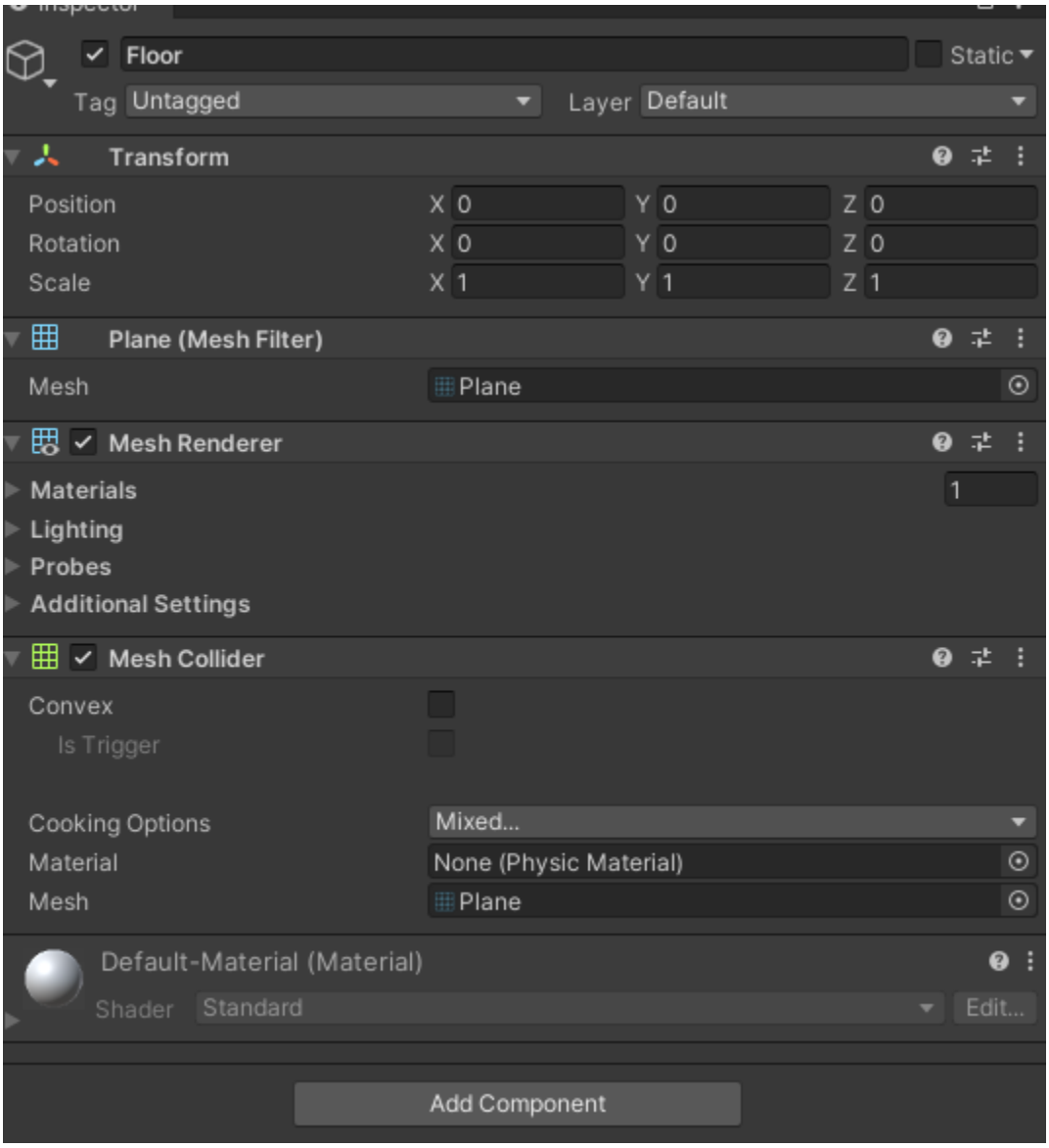
整体结构预览



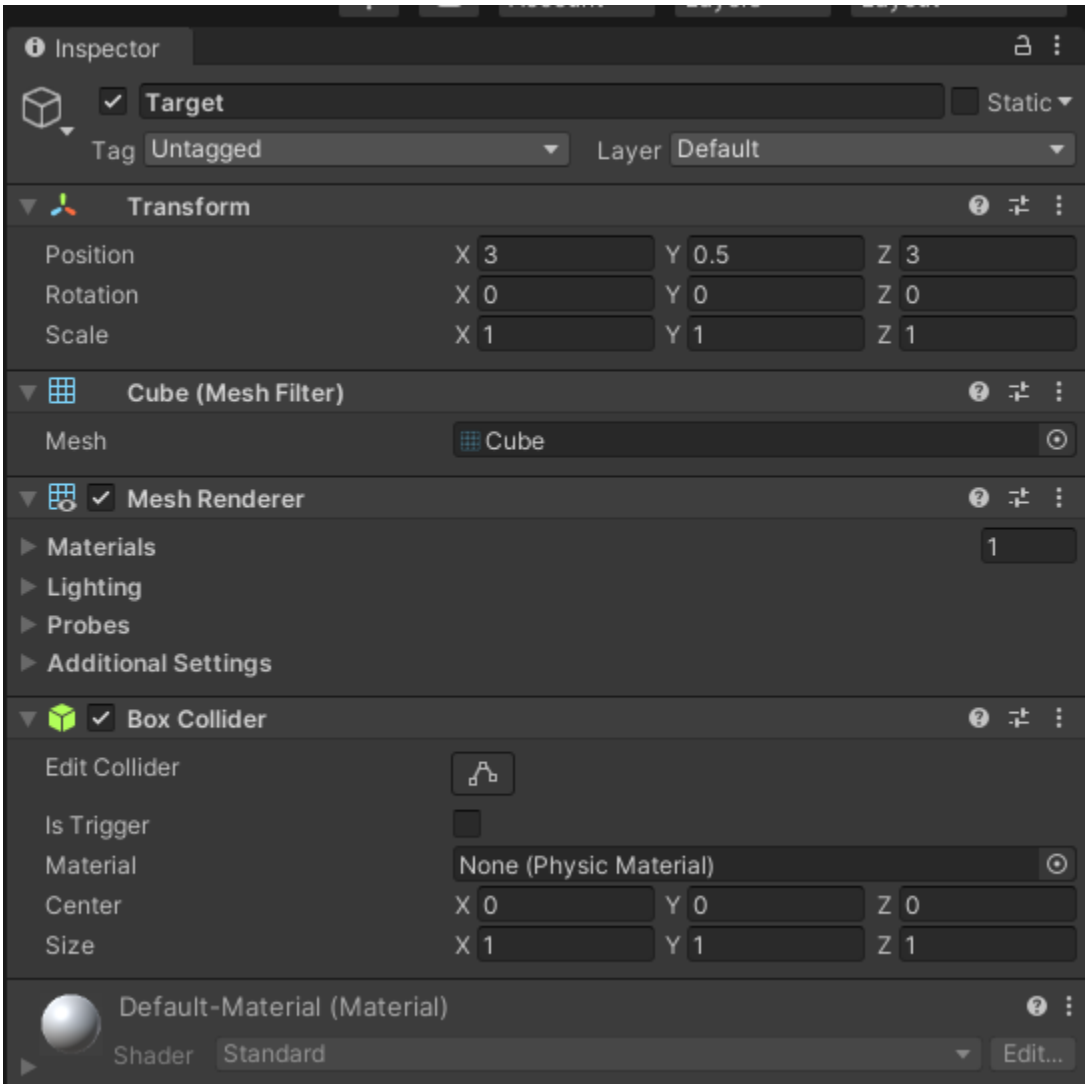
1，GameObject → Create Empty ，重命名 为 TraninningArea



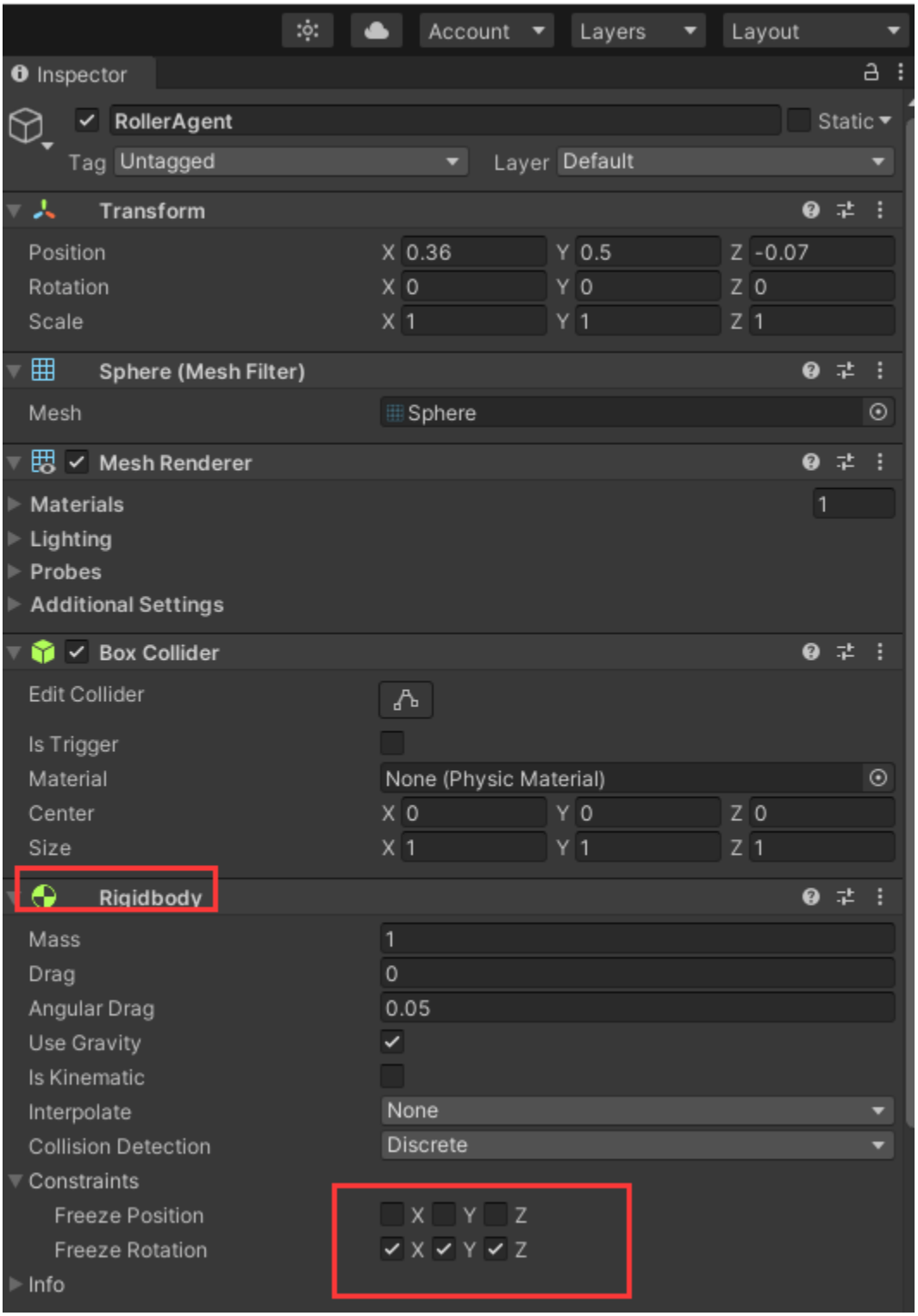
2， GameObject → 3D Object → Plane 重命名为 Floor



3，GameObject → 3D Object → Cube 重命名为 Target

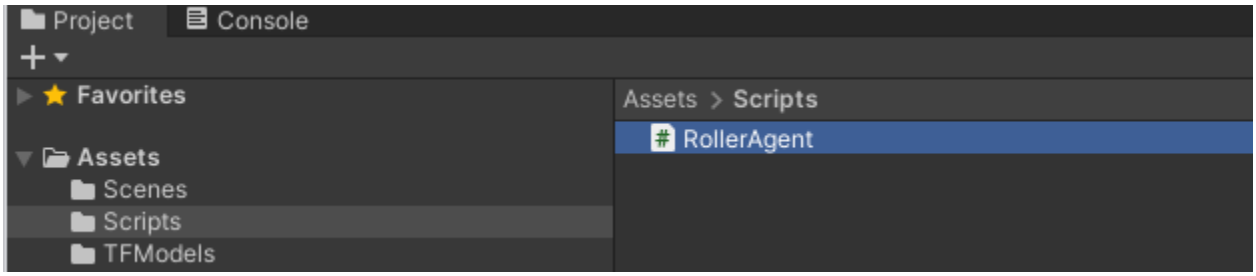


4，GameObject → 3D Object → Sphere 重命名为 RollerAgent


Account
Layers
Layout
Inspector
RollerAgent
Static
Tag Untagged
Layer Default
Transform
Position X 0.36 Y 0.5 Z -0.07
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Sphere (Mesh Filter)
Mesh Sphere
Mesh Renderer
Materials 1
Lighting
Probes
Additional Settings
Box Collider
Edit Collider
Is Trigger
Material None (Physic Material)
Center X 0 Y 0 Z 0
Size X 1 Y 1 Z 1
Rigidbody
Mass 1
Drag 0
Angular Drag 0.05
Use Gravity
Is Kinematic
Interpolate None
Collision Detection Discrete
Constraints
Freeze Position X Y Z
Freeze Rotation X Y Z
Info

四、编写程序

1，在Assets 目录下创建 Scripts 目录，在 Scripts 里创建脚本 RollerAgent 结构为：



2，打开 RollerAgent.cs 文件，修改引用、继承 Agent

```
using Unity.MLAgents;
using Unity.MLAgents.Actuators;
using Unity.MLAgents.Sensors;
using UnityEngine;

Unity 脚本(10 个资产引用)|0 个引用
public class RollerAgent: Agent
{
```

3，定义变量，引入Target

```
    public Rigidbody rbody;
    public GameObject target;
    public float forceMultiplier = 50;

    Unity 消息|0 个引用
    public void Start()
    {
        rbody = this.gameObject.GetComponent<Rigidbody>();
    }
    0 个引用
```

4，重写 Agent 方法：OnEpisodeBegin、CollectObservations、OnActionReceived、Heuristic

OnEpisodeBegin ：训练开始方法

CollectObservations：推送因子

OnActionReceived：接受动作

Heuristic：手动测试

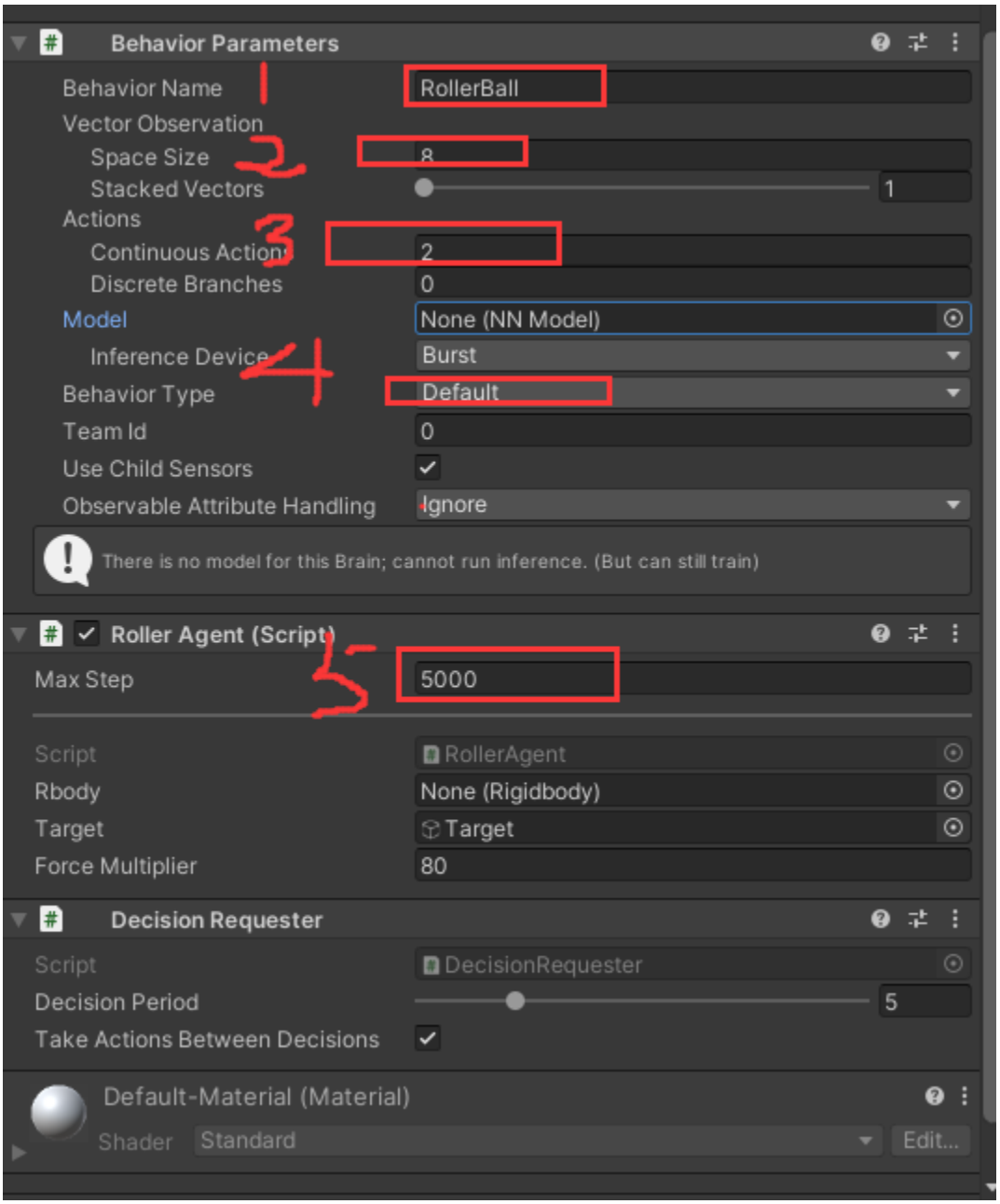
5，完整代码：

```
using Unity.MLAgents;
using Unity.MLAgents.Actuators;
using Unity.MLAgents.Sensors;
using UnityEngine;

public class RollerAgent: Agent
{
  public Rigidbody rbody;
  public GameObject target;
  public float forceMultiplier = 50;
```

```
public void Start()
{
    rbody = this.gameObject.GetComponent<Rigidbody>();
}
public override void OnEpisodeBegin()
{
    if (transform.localPosition.y < 0)
    {
        transform.localPosition = new Vector3(0, 0.5f, 0);
    }
    else
    {
        transform.localPosition = new Vector3(Random.value * 8 - 4, 0.5f, Random.value * 8 - 4);
    }
}

public override void CollectObservations(VectorSensor sensor)
{
    sensor.AddObservation(transform.position);
    sensor.AddObservation(target.transform.position);
    sensor.AddObservation(rbody.velocity.x);
    sensor.AddObservation(rbody.velocity.z);
}
```

```
    public override void OnActionReceived(ActionBuffers action)
    {
        rbody.AddForce(action.ContinuousActions[0] * forceMultiplier, 0, action.ContinuousActions[1] * forceMultiplier);
        Vector3 p1 = target.transform.position;
        Vector3 p2 = transform.position;
        if (Vector2.Distance(new Vector2(p1.x,p1.z), new Vector2(p2.x,p2.z)) < 1.42f)
        {
            AddReward(1.0f);
            EndEpisode();
        }

        if (transform.localPosition.x > 5.5f || transform.localPosition.x < -5.5f || transform.localPosition.z > 5.5f || transform.
localPosition.z < -5.5f)
        {
            EndEpisode();
        }
    }

    public override void Heuristic(in ActionBuffers actionsOut)
    {
        var discreteActionsOut = actionsOut.ContinuousActions;
        discreteActionsOut[0] = Input.GetAxis("Horizontal");
        discreteActionsOut[1] = Input.GetAxis("Vertical");
    }
}
```

五、设置环境



1，行为名，这个名字要和最后配置文件中的一致

2，space size 空间长度，要和程序中 CollectObservations 中输入的数据数量一致（vector3 占3个）

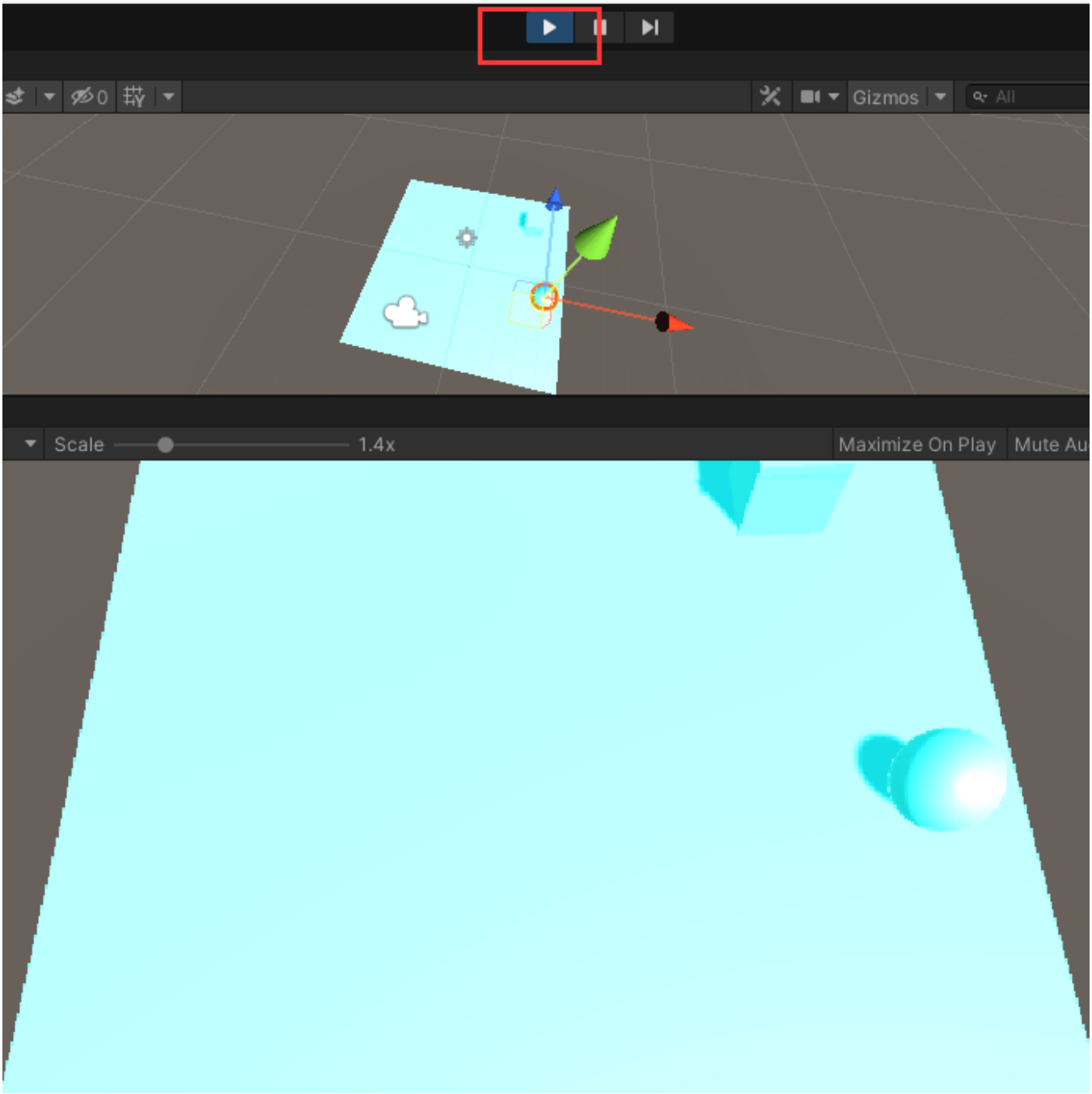
3，接受的动作长度，要和程序中OnActionReceived 中 ActionBuffers.ContinuousActions 长度一致

4，训练模式，选默认就行，程序在运行时会根据是否为训练环境自动判断

5，最大步长，不要太大，要不然训练时会过长

五、手动测试

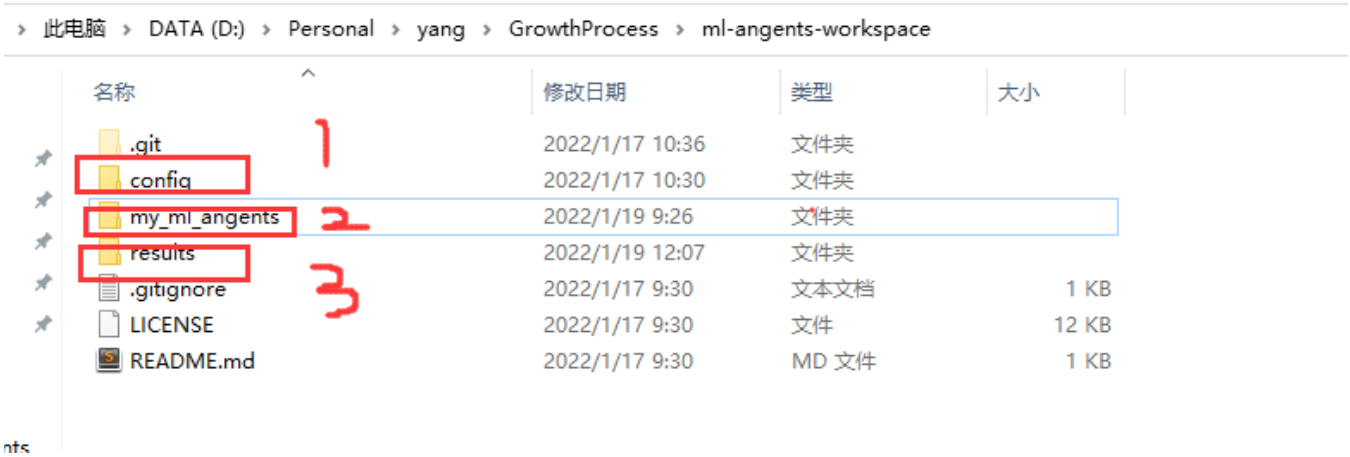
运行场景，



按电脑的上下左右键，观察球的运动是否正常

六、开始训练

修改训练配置，可以在官方测试样例中拷贝一份过来，在其基础上做少量修改，存储在项目的 config/rollerball_config.yaml



1，存放训练配置

2，unity项目

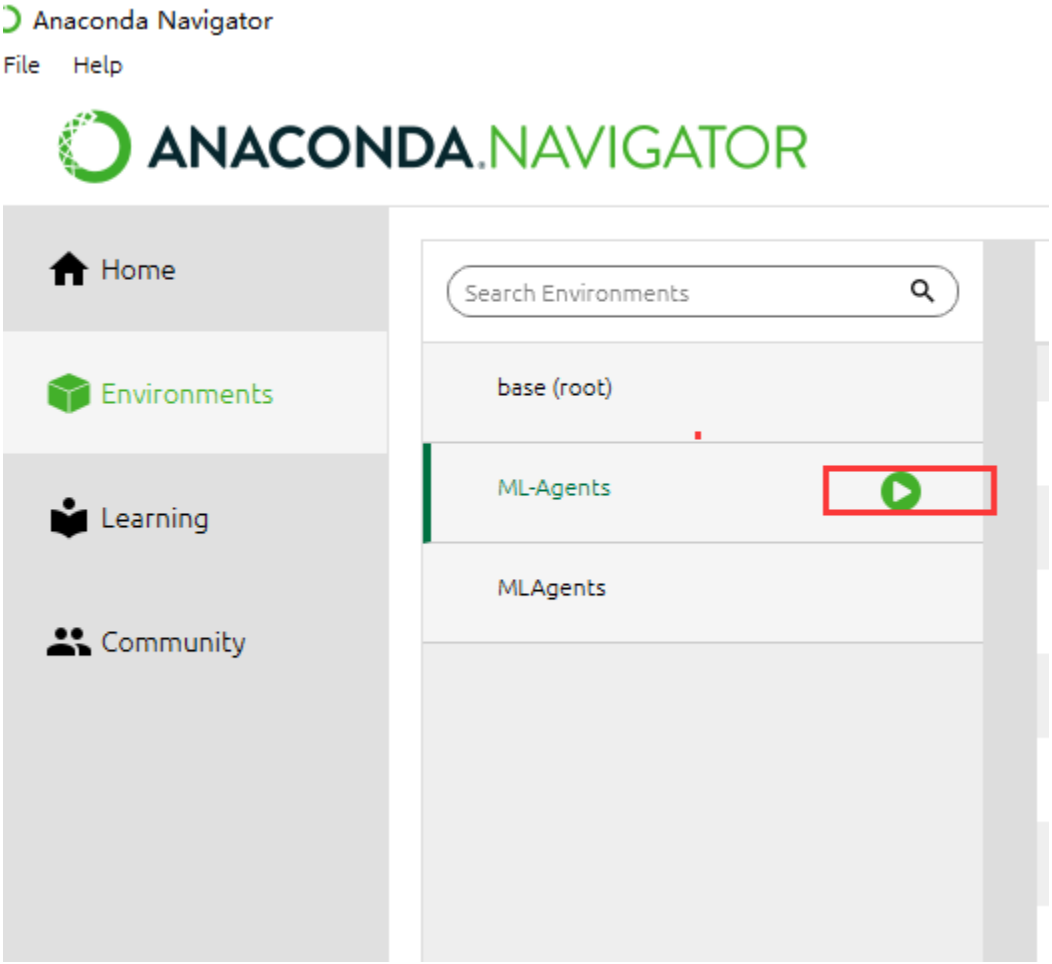
3，最终训练的结果（程序自动生成）

样式如下：

1KWordEveryday.txt | 文档.txt | rollerball_config.yaml | new 1

```yaml
behaviors:
  RollerBall:
    trainer_type: ppo
    hyperparameters:
      batch_size: 64
      buffer_size: 12000
      learning_rate: 0.0003
      beta: 0.001
      epsilon: 0.2
      lambd: 0.99
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: true
      hidden_units: 128
      num_layers: 2
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    keep_checkpoints: 5
    max_steps: 500000
    time_horizon: 1000
    summary_freq: 12000
```

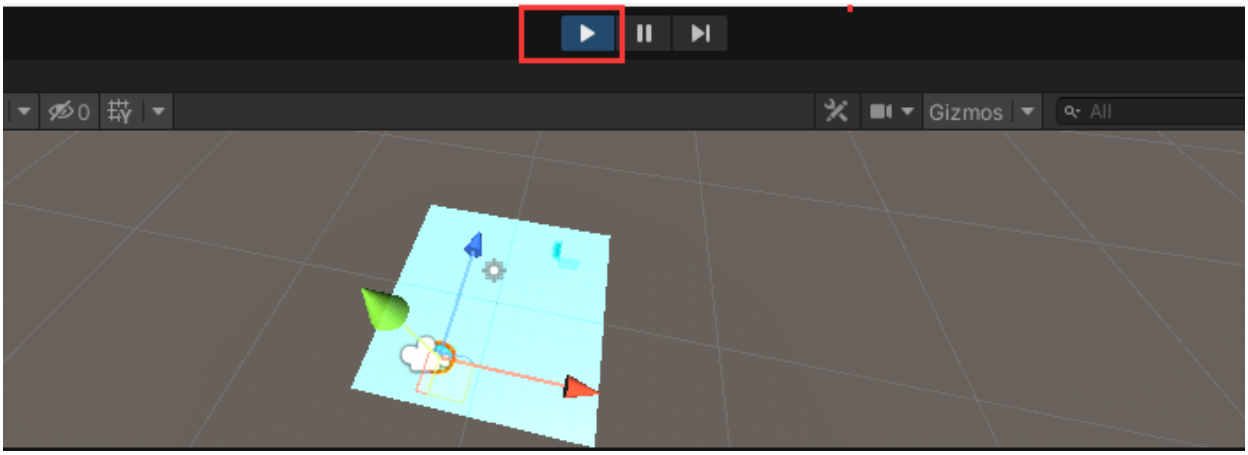
打开Anaconda，运行之前设置好的环境



输入指令：mlagents-learn config/rollerball_config.yaml --run-id=rollerball

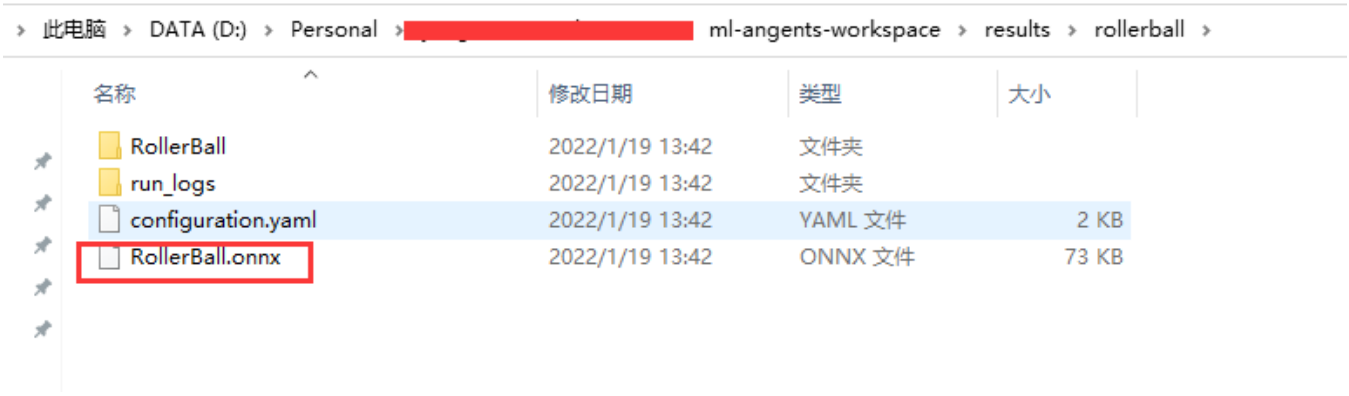
出现如下画面：

运行场景

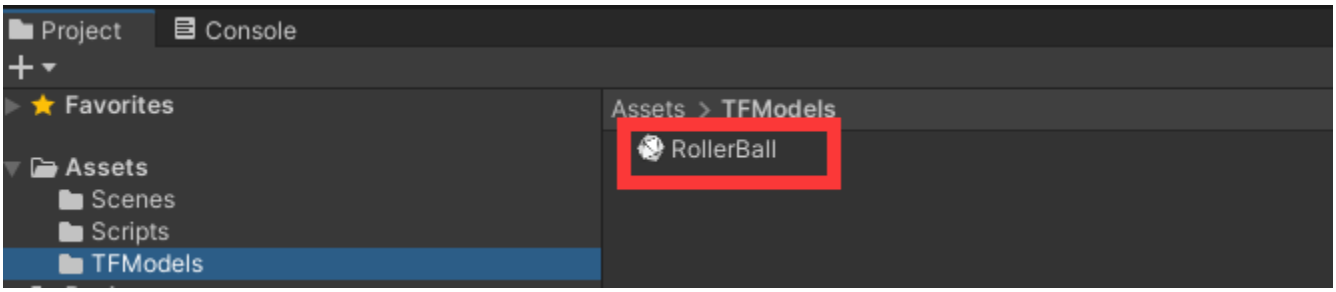


七、导入结果

1，等训练结束后，会在项目路径下多出一个 results 目录，将如下图所示：



2，将训练结果onnx文件拷贝入 unity 项目中：



3，将训练出来的模型挂接到Agent 之后，运行unity 场景便可以看到Angent 的运动效果了

Inspector

Size X 1 Y 1 Z 1

Rigidbody

Mass 1
Drag 0
Angular Drag 0.05
Use Gravity ✓
Is Kinematic ☐
Interpolate None
Collision Detection Discrete
Constraints
Freeze Position ☐ X ☐ Y ☐ Z
Freeze Rotation ✓ X ✓ Y ✓ Z
Info

Behavior Parameters

Behavior Name RollerBall
Vector Observation
Space Size 8
Stacked Vectors 1
Actions
Continuous Actions 2
Discrete Branches 0
Model RollerBall (NN Model)
Inference Device Burst
Behavior Type Default
Team Id 0
Use Child Sensors ✓
Observable Attribute Handling Ignore

Roller Agent (Script)

Max Step 5000

Script RollerAgent
Rbody None (Rigidbody)
Target Target
Force Multiplier 80

Decision Requester

Script DecisionRequester
Decision Period 5
Take Actions Between Decisions ✓

Default-Material (Material)
Shader Standard Edit...

Add Component