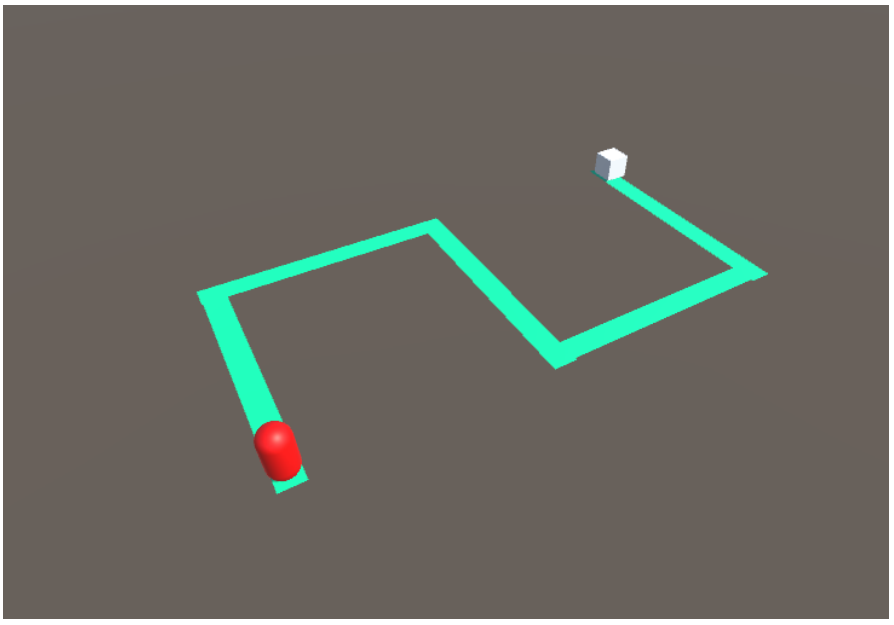
规定行径路径 让物体沿着路径移动



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

public class Move : MonoBehaviour {

    public GameObject[] targets;
    private int index;
    private Vector3 target;
    private float speed = 3.0f;
    private bool hasPath=true;
    void Start () {
        //给第一个次需要移动到的目标点赋值
        target.Set(targets[index].transform.position.x,
            transform.position.y,
            targets[index].transform.position.z);
    }

    // Update is called once per frame
    void Update () {

        if (!hasPath)
            return;
```

```
24
25          if (Vector3.Distance(transform.position, target) > 0.1f)
26          {
27              transform.LookAt(target);
28              transform.Translate(Vector3.forward * speed *
    Time.deltaTime);
29          }
30          else
31          {
32              index++;
33              if (index > targets.Length - 1)
34              {
35                  //删除最后一个目标点
36                  Destroy(targets[targets.Length - 1]);
37                  hasPath = false;
38                  return;
39              }
40
41              target.Set(targets[index].transform.position.x,
42                          transform.position.y,
43                          targets[index].transform.position.z);
44          }
45      }
46  }
47
```

Unity的物理碰撞

在Unity中碰撞的物体分为两种

1 发起碰撞的物体

2 接收碰撞的物体

1 发起碰撞的物体：

RigidBody (刚体) CharacterController（角色控制器）

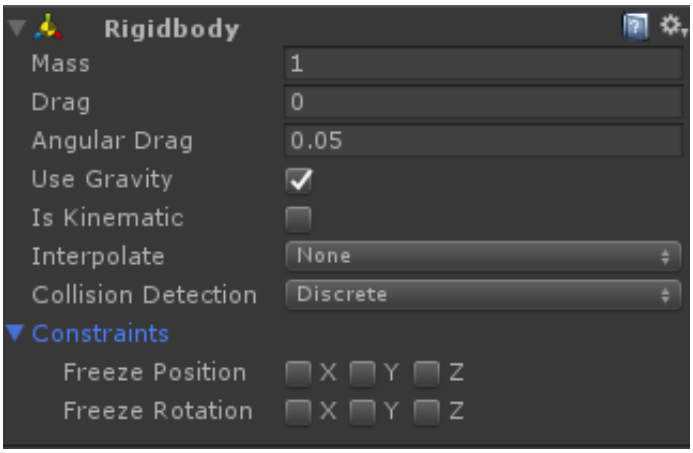
2 接收碰撞的物体：

所有的Collider

RigidBody 刚体组件：

当刚体组件添加到游戏物体身上

那么游戏物体的位移和旋转都可以使用刚体进行控制



Mass： 质量 正常情况下无需调整 默认为1即可

drag: 阻力 用来减缓刚体的速度 阻力越大减速效果越明显

Angular Drag: 角阻力 用来减缓刚体的旋转 角阻力越大旋转越慢

UseGravity 是否使用重力 如果为False 刚体不受重力的影响

IsKinematic 是否使用动力学 如果为False 力 碰撞或关节等不生效 刚体只能通过Transform或动画，脚本来进行修改

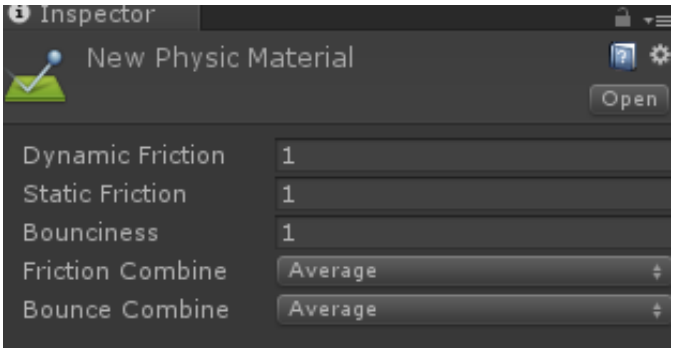
Interpolate 刚体插值 固定的帧率平滑物理效果 默认关闭 最多应用于玩家

Collision Detection 碰撞检测模式 效果从上往下依次递增 性能开销也依次递增

Freeze Position 锁定某个轴的位移变换

Freeze Rotation 锁定某个轴的旋转变换

物理材质



动态摩擦力 通常值0-1之间

静态摩擦力 通常值0-1之间

当值为0时 效果像冰面 设为1时 物体运动将会很快停止

弹力：反弹系数 0不反弹 1是没有任何能量损失

Unity物理引擎的几种碰撞器

StaticCollider 静态碰撞器

没有添加刚体 但是添加碰撞器组件的游戏对象

这类对象保存静态或微乎其微的运动

1 墙面 2 静止的房屋 3树木等

RigidBodyCollider 刚体碰撞器

添加了刚体同时也添加了碰撞器的游戏对象

Kinematic RigidBody Collider 运动学刚体碰撞器

不受力的作用 由Transform来控制游戏物体的运动或数据修改

碰撞的工作原理：

发生碰撞的物体必须要有"发起碰撞"的物体 否则碰撞不响应

一个带有RigidBody组件的Cube是可以发起碰撞的物体 能落到一个带有Collider属性的平面上

碰撞事件函数

```
    //碰撞事件函数 碰撞回调函数
    //当碰撞进入时调用一次
    void OnCollisionEnter(Collision collsion)
    {
        Debug.Log(collsion.gameObject);
        Debug.Log(collsion.transform);
        Debug.Log(collsion.collider);

        Debug.Log(collsion.rigidbody);//没有返回null
        Debug.Log(collsion.relativeVelocity);//相对的线性速度
        Debug.Log(collsion.contacts);//碰撞物体的接触点

```

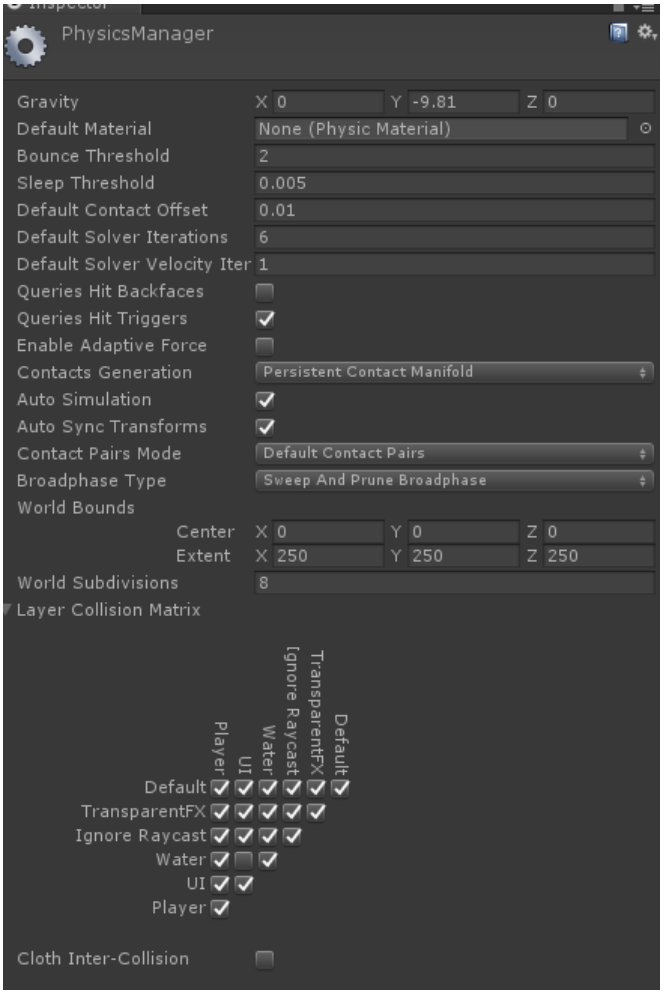
```
        Debug.Log("碰撞");
    }

    //当碰撞接触中调用 碰撞逗留
    void OnCollisionStay(Collision collision)
    {
       // Debug.Log("逗留");
    }
    //当碰撞结束调用一次  碰撞退出
    void OnCollisionExit(Collision collision)
    {
        Debug.Log("退出");
    }
```

数据过滤

```
    void OnCollisionEnter(Collision collsion)
    {
        //通过标签进行数据过滤
        if (collsion.gameObject.CompareTag("Point"))
        {
            //Debug.Log("碰撞");
        }

        //使用Layer来进行数据过滤
        if (LayerMask.LayerToName(collsion.gameObject.layer).Equals("UI"))
        {
            Debug.Log("碰撞");
        }

        //获取UI层的int数据 2的5次方
        int x=1<<LayerMask.NameToLayer("UI");
    }
```

直接过滤碰撞的发生

通过设置不同Layer或相同的Layer来决定是否有物理的碰撞效果




```
1     private Rigidbody body;
2
3    void Start()
4    {
5        body = GetComponent<Rigidbody>();
6    }
7
8    void Update()
9    {
10       //设置刚体的速度 来控制物体的位移
11       body.velocity = Vector3.up;
12   }
```

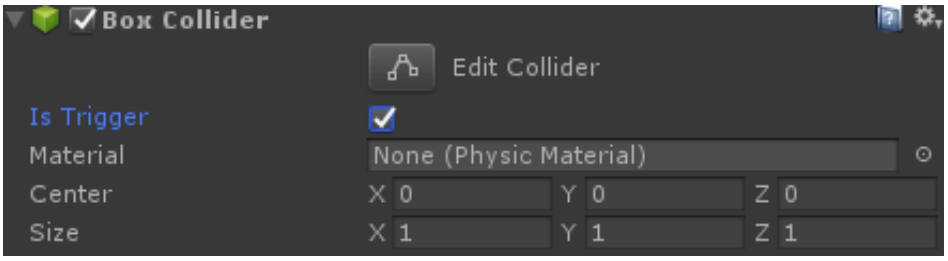
物体发生翻转

解决办法：

1 锁定旋转轴

2 取消重力影响

触发器 Trigger



勾选 IsTrigger

只保留碰撞的逻辑和数据 但是没有碰撞的物理效果

触发器的碰撞函数

```
    void OnTriggerEnter(Collider collider)
    {
        Debug.Log("触发");
    }
    void OnTriggerStay(Collider collider)
    {
        Debug.Log("逗留");
    }
    void OnTriggerExit(Collider collider)
    {
        Debug.Log("结束");
    }
```

将MeshRenderer组件设置为可用或不可用

```
    void OnTriggerEnter(Collider collider)
    {
        //设置组件为非激活状态
        //collider.gameObject.GetComponent<MeshRenderer>().enabled =
```

```
false;

        collider.gameObject.SetActive(false);
    }

    void OnTriggerExit(Collider collider)
    {
        //设置组件为激活状态
        //collider.gameObject.GetComponent<MeshRenderer>().enabled =
true;
        collider.gameObject.SetActive(true);
    }
```

Vector3.Forward 是一个常量 永远是0 0 1

transform.Forward 是一个变量 基于自身和世界旋转角度而变化

```
     void Update()
    {
         //当参数列表需要传递基于世界的向量时
         //不能使用Vector3.Forward 使用transform.forward
        transform.Translate(transform.forward,Space.World);
    }
```

使用刚体添加作用力

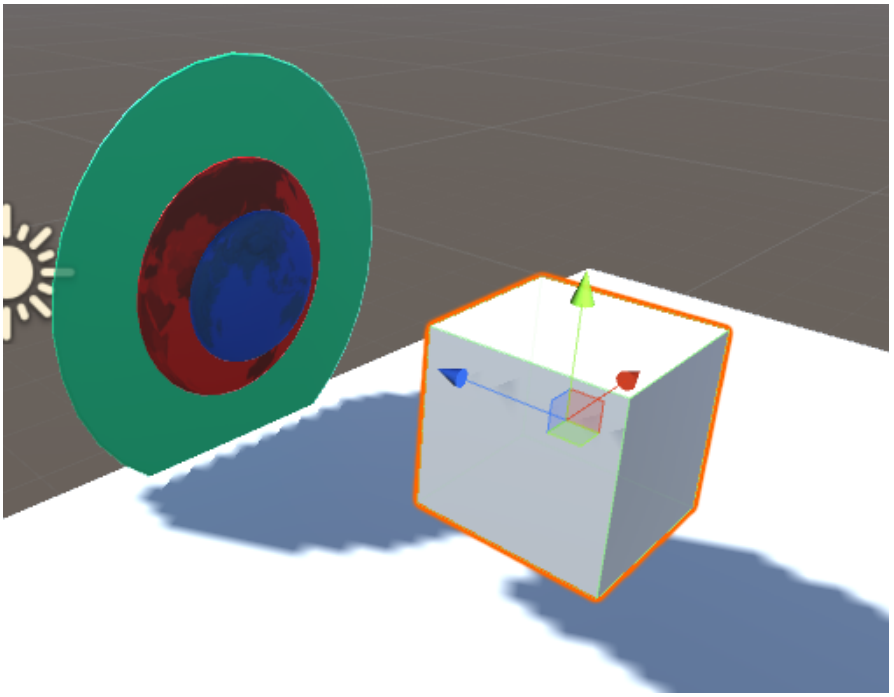
```
  private Rigidbody body;

    void Start()
    {
        //获取刚体组件 通过AddForce方法像物体的前方施加力
        body = GetComponent<Rigidbody>();
        body.AddForce(transform.forward*1000f);
    }
```

鼠标点击 让一个Cube的6个方向发射子弹

```
 public GameObject bulletObj;


    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            GameObject bullet = Instantiate(bulletObj,transform.position,transform.rotation);

            bullet.GetComponent<Rigidbody>().AddForce(transform.forward * 1000f);

            GameObject bullet1 = Instantiate(bulletObj, transform.position, transform.rotation);

            bullet1.GetComponent<Rigidbody>().AddForce(-transform.forward * 1000f);

            GameObject bullet2 = Instantiate(bulletObj, transform.position, transform.rotation);

            bullet2.GetComponent<Rigidbody>().AddForce(transform.up* 1000f);

            GameObject bullet3 = Instantiate(bulletObj, transform.position, transform.rotation);

            bullet3.GetComponent<Rigidbody>().AddForce(-transform.up * 1000f);

            GameObject bullet4 = Instantiate(bulletObj, transform.position, transform.rotation);

            bullet4.GetComponent<Rigidbody>().AddForce(transform.right * 1000f);

            GameObject bullet5 = Instantiate(bulletObj, transform.position, transform.rotation);
```

```
29
30              bullet5.GetComponent<Rigidbody>().AddForce(-
transform.right * 1000f);
31
32          }
33      }
```

制作枪靶 发射子弹 判断环数



```
1  public class CollisionText : MonoBehaviour {
2
3      public GameObject bulletObj;
4      void Update()
5      {
6          if (Input.GetMouseButtonDown(0))
7          {
8              GameObject obj= Instantiate(bulletObj, transform.position,
transform.rotation);
9              obj.GetComponent<Rigidbody>
().AddForce(transform.forward*1000f);
10         }
11     }
12 }
13
```

```
14 public class Bullet : MonoBehaviour {
15
16     void OnTriggerEnter(Collider collider)
17     {
18         if (collider.gameObject.CompareTag("Player"))
19             return;
20
21         Destroy(gameObject);
22         if (collider.gameObject.CompareTag("10"))
23         {
24             Debug.Log("10环");
25         }
26         if (collider.gameObject.CompareTag("9"))
27         {
28             Debug.Log("9环");
29         }
30         if (collider.gameObject.CompareTag("5"))
31         {
32             Debug.Log("5环");
33         }
34     }
35 }
```