# 复刻坦克大战问题合集

```
public class player : MonoBehaviour
{
    public float moveSpeed=3;//这个变量用于控制物体的移动速度。
    // Start is called before the first frame update
    void Start()
    {
    }
    // Update is called once per frame
    void Update()
    {

       float h=Input.GetAxisRaw("Horizontal");//获取水平方向上的输入值

       transform.Translate(Vector3.right*h*moveSpeed*Time.deltaTime,Space.World);//将物体向右移动一个单位

       float v=Input.GetAxisRaw("vertical");//获取垂直方向上的输入值

       transform.Translate(Vector3.up*v*moveSpeed*Time.deltaTime,Space.World);

    }
```

```
float h=Input.GetAxisRaw("Horizontal");
transform.Translate(Vector3.right*h*moveSpeed*Time.deltaTime,Space.World);
```

这两行代码首先通过 `Input.GetAxisRaw("Horizontal")` 获取水平方向（通常是左右方向）上的输入值（通常是-1、0或1），然后使用 `transform.Translate` 方法将物体沿着世界坐标系的右方向移动。移动的距离是输入值 `h` 乘以速度 `moveSpeed` 再乘以 `Time.deltaTime`（上一帧到当前帧的时间间隔），这样可以确保移动速度与帧率无关，更加平滑。

## vctor3.up的意思

在Unity中，`Vector3.up` 是一个静态属性，它代表了一个指向世界空间上方（通常是Y轴正方向）的单位向量。具体来说，`Vector3.up` 的值是 `(0, 1, 0)`，这意味着它是一个没有缩放（长度为1）的向量，其方向垂直于地面（在Unity的默认设置下，Y轴是向上的）。

这个属性经常被用于移动、旋转或定位游戏对象，特别是在你想要相对于世界空间的上方进行操作时。例如，如果你想要让一个游戏对象向上移动一定的距离，你可以使用 `transform.Translate(Vector3.up * distance)`，其中 `distance` 是你想要移动的距离（以单位计）。

这里是一个简单的例子，展示了如何使用 `Vector3.up` 来移动一个游戏对象：

```
using UnityEngine;
public class MoveUp : MonoBehaviour { public float moveSpeed = 5f; // 移动速度 void Update() { // 如果按下空格键，则向上移动 if (Input.GetKeyDown(KeyCode.Space))
{ // 使用transform.Translate来移动游戏对象，向上移动的速度由moveSpeed控制
// Time.deltaTime用于确保移动速度与帧率无关
transform.Translate(Vector3.up * moveSpeed * Time.deltaTime); }
}
}
```

在这个例子中，每当玩家按下空格键时，游戏对象就会以 `moveSpeed` 指定的速度向上移动。`Vector3.up` 提供了移动的方向，而 `moveSpeed * Time.deltaTime` 则确定了移动的距离（考虑到了帧率的影响）。

问题

# The referenced script on this Behaviour is missing 如何把这个警告解决-很简单

预制体 脚本丢失
顺藤摸瓜 找到有问题的预制体， 然后重新添加正确的脚本/移除没用的脚本！
就是这么简单！！！
1、删除错误的预制体
2、重新制作预制体
3、重新挂脚本

## 碰撞检测条件

1、双方都要有碰撞器
2、运动的一方要有刚体

## Public

在C#（CSharp）编程语言中，`public`是一种访问修饰符（access modifier），用于定义类的成员（方法、属性、字段等）的访问级别。具体来说，`public`表示被修饰的成员可以从任何地方访问，包括同一程序集内的其他类、不同程序集中的类以及派生类。

这行代码看起来是在使用 Unity 引擎中的功能。让我解释一下这段代码的含义：

```
Instantiate(bulletPrefab, transform.position, transform.rotation);
```

## 解释：

1. **Instantiate**：
   - `Instantiate`是 Unity 中用来创建（实例化）新对象的方法。它允许你根据预制体（Prefab）创建新的游戏对象实例。
2. **参数**：
   - **bulletPrefab**：这是一个预制体（Prefab）对象，通常是在 Unity 编辑器中创建和配置的一个模板。在运行时，`Instantiate`会根据这个预制体创建一个新的实例。
   - **transform.position**：这是一个`Vector3`类型的参数，表示新实例将会被放置的位置。在这里，`transform.position`指的是当前代码所在对象（通常是脚本所附着的游戏对象）的位置。
   - **transform.rotation**：这是一个`Quaternion`类型的参数，表示新实例将会被放置的旋转。类似地，`transform.rotation`表示当前对象的旋转状态。

## 具体功能：

- 该行代码的作用是根据`bulletPrefab`预制体在当前对象的位置（`transform.position`）和旋转（`transform.rotation`）处创建一个新的实例。通常用于在游戏中生成子弹、特效或其他动态物体。

## 示例：

假设你有一个名为`Bullet`的预制体，它包含了子弹的模型、碰撞器等组件。当你调用`Instantiate(bulletPrefab, transform.position, transform.rotation);`时，Unity 会根据`bulletPrefab`创建一个新的`Bullet`实例，并将其放置在当前对象的位置和旋转下。

这种方式可以让你在游戏运行时动态地生成和管理游戏对象，非常适合处理需要频繁生成、销毁或移动的游戏元素。

## 四元数

四元数（Quaternion）是一种数学结构，用来表示三维空间中的旋转。在计算机图形学和游戏开发中，四元数被广泛用来描述物体的旋转，相比于欧拉角，四元数有着更好的性质和更少的奇点问题。

## 主要特点和定义：

1. **定义**：
   - 四元数是由一个实部（标量）和三个虚部（向量）组成的数学结构，通常写成 `q = w + xi + yj + zk`，其中 `w` 是实部，`x, y, z` 是虚部。
2. **用途**：
   - 主要用来表示和计算物体在三维空间中的旋转变换。相比于欧拉角，四元数可以避免万向锁问题（Gimbal Lock），并且在旋转计算上更加高效和精确。
3. **性质**：
   - **单位四元数**：长度为1的四元数被称为单位四元数，用于表示纯粹的旋转。
   - **旋转组合**：通过将两个四元数相乘，可以有效地组合两个旋转。
   - **插值**：四元数可以进行线性插值，用于平滑地在两个旋转之间过渡。
4. **数学性质**：
   - 四元数满足一些数学性质，如加法、乘法和共轭运算，这些运算使得旋转计算变得高效和准确。

## 在游戏开发中的应用：

- Unity 和其他游戏引擎广泛使用四元数来管理物体的旋转，因为它们可以更加高效地进行旋转计算，并且避免了欧拉角的一些限制和问题。
- 四元数也用于姿态控制、相机控制和动画系统中，因为它们提供了一种简洁而精确的方式来表示和操作三维空间中的旋转变换。

总结来说，四元数是一种强大且有效的数学工具，用于表示和计算三维空间中的旋转，特别适用于计算机图形学和游戏开发领域。
在游戏开发中，`Time.deltaTime` 是一个非常重要的概念，它表示了每一帧（Frame）之间的时间间隔。具体来说：

## `Time.deltaTime` 的含义和用途：

1. **时间间隔**：`Time.deltaTime` 是一个小数值，表示当前帧和上一帧之间的时间间隔，通常以秒为单位。它告诉你每一帧之间经过的时间，这对于实现平稳的动画和物理模拟非常重要。
2. **帧率无关性**：由于不同设备的帧率（FPS）可能不同，因此使用 `Time.deltaTime` 可以使得游戏的表现更加一致和可预测。即使在低帧率的情况下，游戏中的物体也可以以相同的速度运动，因为它们移动的距离会根据实际经过的时间间隔来计算，而不是依赖于每秒渲染的帧数。
3. **示例用法**：
   - **移动物体**：如果你希望一个物体每秒移动10个单位，你可以使用 `transform.Translate(Vector3.forward * speed * Time.deltaTime);`。这样无论帧率是60FPS还是30FPS，物体每秒移动的距离都是一致的。
   - **时间相关动画**：实现平滑的动画效果，比如淡入淡出、渐变效果等，都可以利用 `Time.deltaTime` 来控制每帧变化的量。
4. **物理模拟**：在处理物理引擎时，也经常使用 `Time.deltaTime` 来确保物理计算的稳定性和预测性。

## 使用注意事项：

- **固定时间步长**：有时候，特别是在物理引擎中，你可能需要固定时间步长（Fixed Time Step），这时可以使用 `Time.fixedDeltaTime`。
- **非更新循环使用**：在某些情况下，你可能需要非更新循环（非 Time.deltaTime 控制的）的时间，可以使用 `Time.unscaledDeltaTime`。

总之，`Time.deltaTime` 是 Unity 中用于获取每帧时间间隔的重要工具，它使得游戏开发者可以编写更加平滑和可靠的游戏逻辑和动画效果。

## 渲染层级

层级越高，越先渲染，视觉上先看到

## 玩家子弹发射不出来问题

原因：bullet脚本下，发射子弹，立马就销毁了
解决：把销毁代码放在if条件代码下，但是出现敌人子弹连续发射

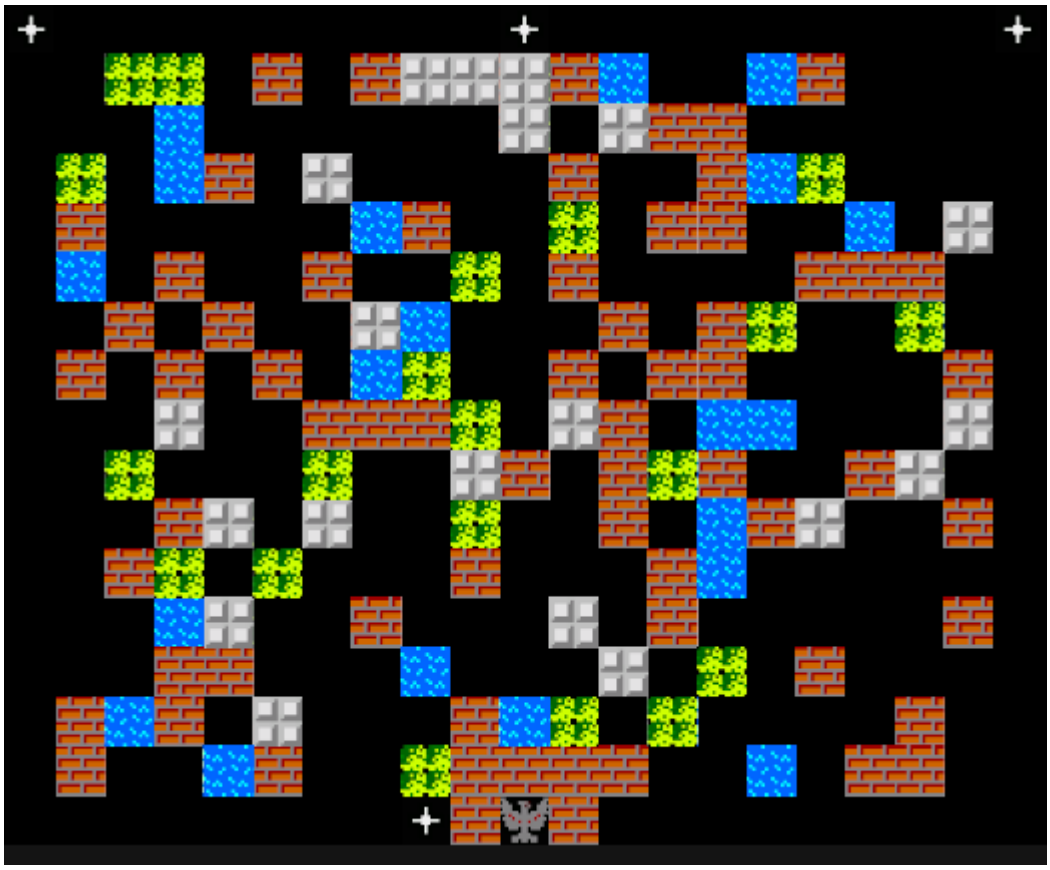
```
    switch (collision.tag)
   {
       case "Tank":
           if (!isPlayerBullet)
           {
               collision.SendMessage("Die");
               Destroy(gameObject);
           }

           break;
       case "Heart":
           collision.SendMessage("Die");
           Destroy(gameObject);
           break;
       case "Enemy":
           if (isPlayerBullet)
           {
               collision.SendMessage("Die");
               Destroy(gameObject);
           }

           break;
       case "Wall":
           Destroy(collision.gameObject);//销毁墙
           Destroy(gameObject);//销毁子弹
           break;
       case "Barrier":
           Destroy(gameObject);
           break;
       default:
           break;
```

怎么解决子弹连续发射问题？
解决：在enemy脚本下，把fixedupdate的attack方法删除

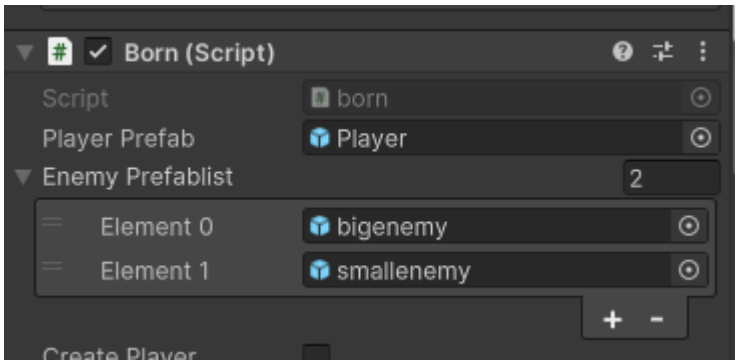
```
private void FixedUpdate()  //固定一帧的时间
{
    Move();

}
```

## 敌人一出现就是出现特效，没有具体的敌人出现

如图：

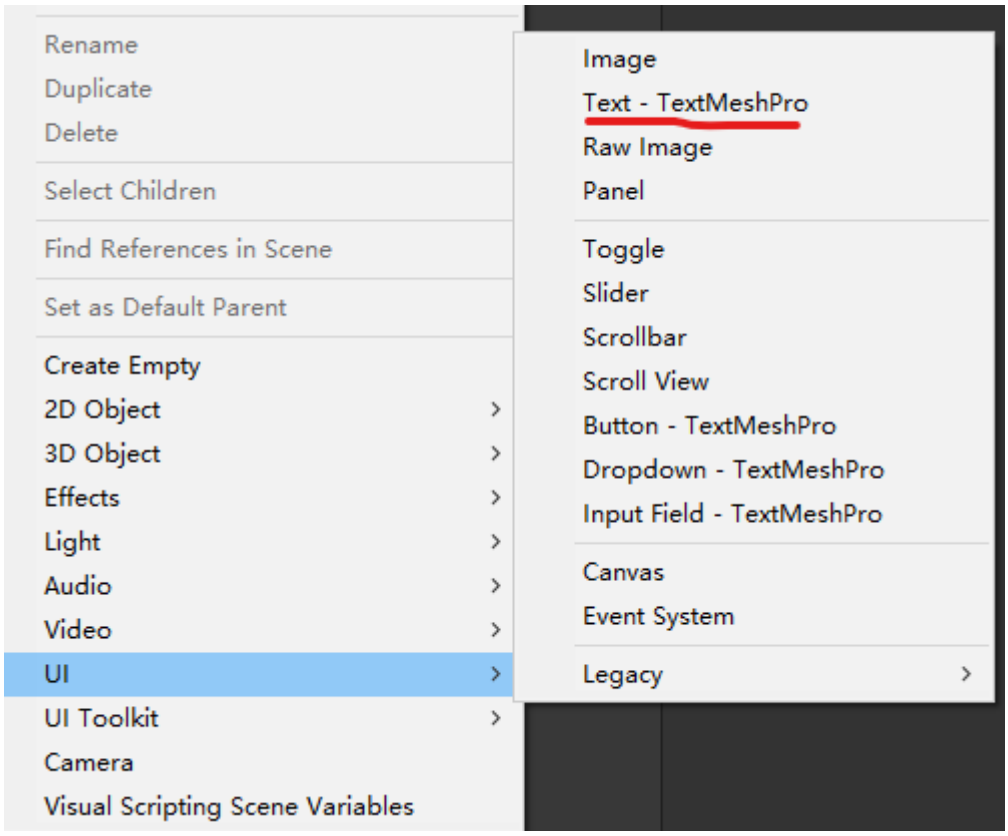


解决方案：
发现born的预制体内的脚本的敌人列表没有添加敌人模型
加入之后就可解决该问题
如图：

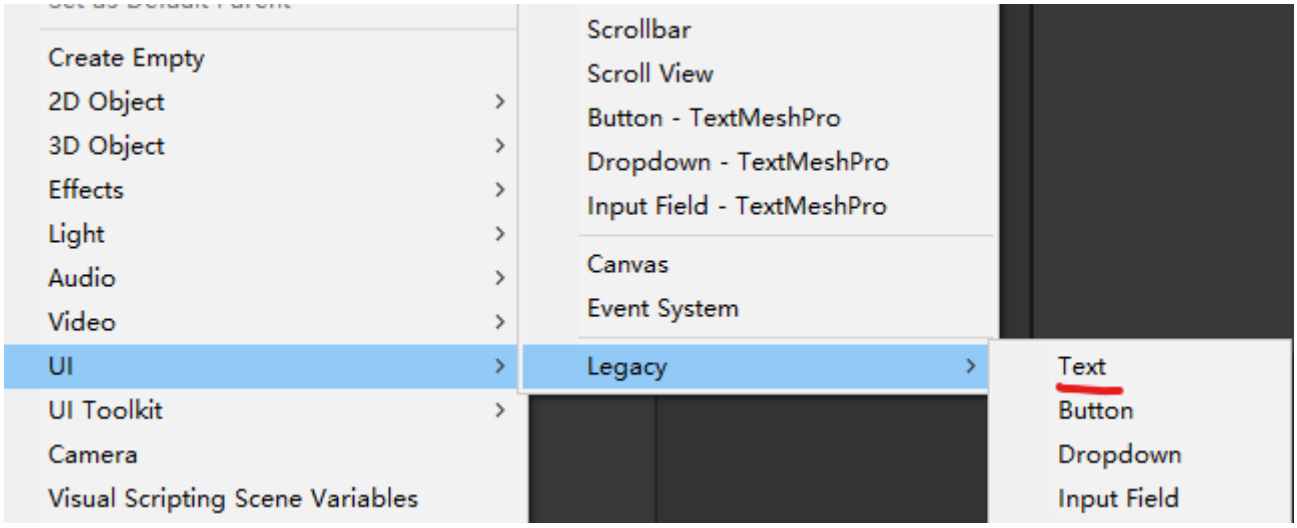



## unity ui的text拖不进脚本里

发现创建的组件不是text组件，是TextMeshPro组件




而text组件在UI⇨Legacy⇨Text




## 这里有个BUG 玩家有4条命

bug代码

```
void Update()
{
    if (isDefeat)
    {
        isDefeatUI.SetActive(true);
        Invoke("ReturnTheMainMenu", 3);
        return;
    }
    if (isDead)
    {
        Recover();
    }
    PlayerScoreText.text = playerScore.ToString();
    PlayerLifeValueText.text = lifeValue.ToString();
```

```
}
private void Recover()
{

    if (lifeValue <=0)
    {
        //游戏失败，返回主界面
        isDefeat = true;
        Invoke("ReturnTheMainMenu", 3);
    }
    else
    {
        lifeValue--;
        GameObject go = Instantiate(born,new Vector3(-2,-8,0),Quaternion.identity);
        go.GetComponent<born>().createPlayer =true;
        isDead = false;
    }
}
```

关键点在于recover中的减生命在判断失败的后面做的
比如：第一条命 lifevalue=3 运行else代码
第二条命 lifevalue=2 运行else代码
第三条命 lifevalue=1 运行else代码
第四条命 lifevalue=0时，运行if代码
解决方案：
把lifeValue--;提到if (lifeValue <=0)前

```
void Update()
{
    if (isDefeat)
    {
        isDefeatUI.SetActive(true);
        Invoke("ReturnTheMainMenu", 3);
        return;
    }
    if (isDead)
    {
        Recover();
    }
    PlayerScoreText.text = playerScore.ToString();
    PlayerLifeValueText.text = lifeValue.ToString();
}
private void Recover()
{
     lifeValue--;
    if (lifeValue <=0)
    {
        //游戏失败，返回主界面
        isDefeat = true;
        Invoke("ReturnTheMainMenu", 3);
    }
    else
    {
        GameObject go = Instantiate(born,new Vector3(-2,-8,0),Quaternion.identity);
        go.GetComponent<born>().createPlayer =true;
        isDead = false;
    }
}
```

## 怎么做开场向上滑的动画?

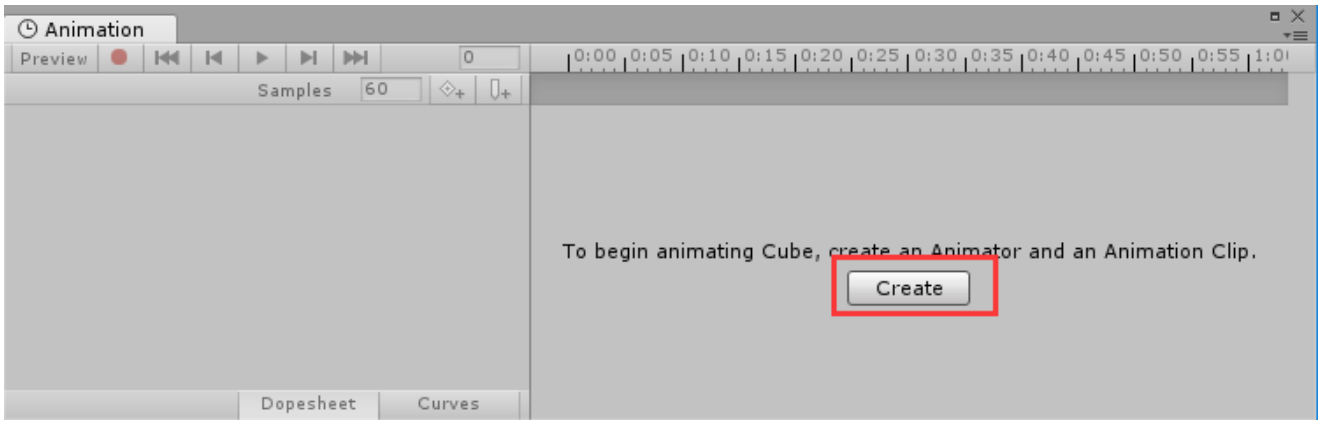
1、打开Animation窗口

Window > Animation打开Animation窗口

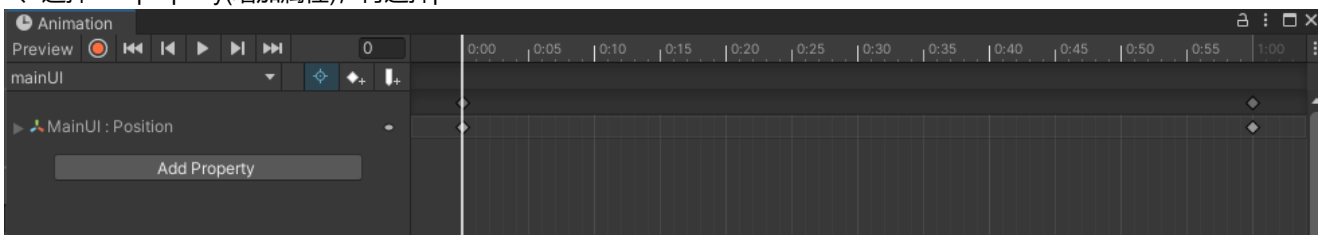
2、选中要制作动画的物体

3、创建新的动画Clip

把mainUI.anim保存在Animations文件夹中




4、选择add property(增加属性)，再选择position




5、加入两个关键帧

界面的画面下的关键帧

界面的画面居中的关键帧

6、不循环播放

取消mianUI动画的loop time下的勾