这套是官方的案例，老师讲解的很细，很适合复习基础知识以及拓宽思路。

**第一部分：几个基础知识**

1. Start()函数：

Unity只在Start中执行一次，且是在第一帧更新前执行。

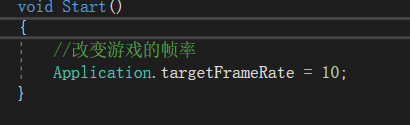
2. Update()函数：Unity逐帧调用。

3. 一般小写的是变量本身，大写的是变量类型。

4. 在这里可以查看输入设置，如何控制物体的行为。

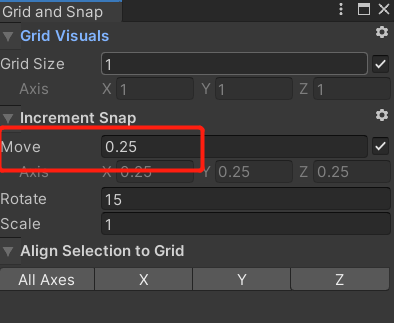


5. 改变帧率



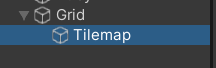
**第二部分：Tile Map**

1. 制作2D关卡场景的传统方法：拖拽法，在网格和移位的面板里可以调整按住ctrl时固定移动的数值（Move）。



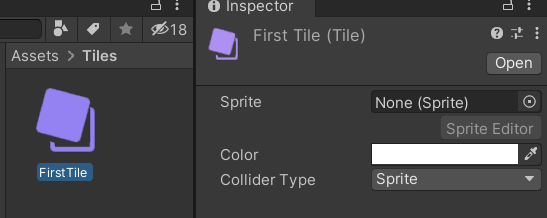
2. 高效的方法：Tilemap

创建方式：右键 -> 2D Object -> Tilemap

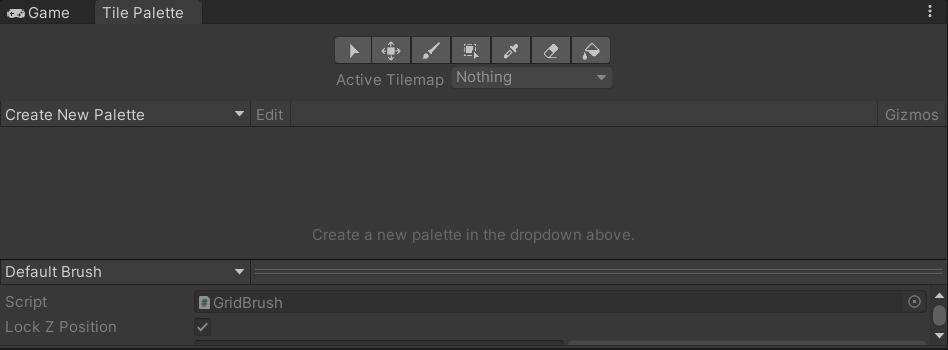


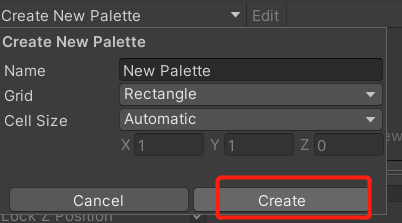
3. sprite可以理解为纹理的容器，而tile是一种特殊的sprite，如同颜料，而tilemap如同一个画布，可以在上面用tile进行绘画。

4. 创建tile及tile属性：



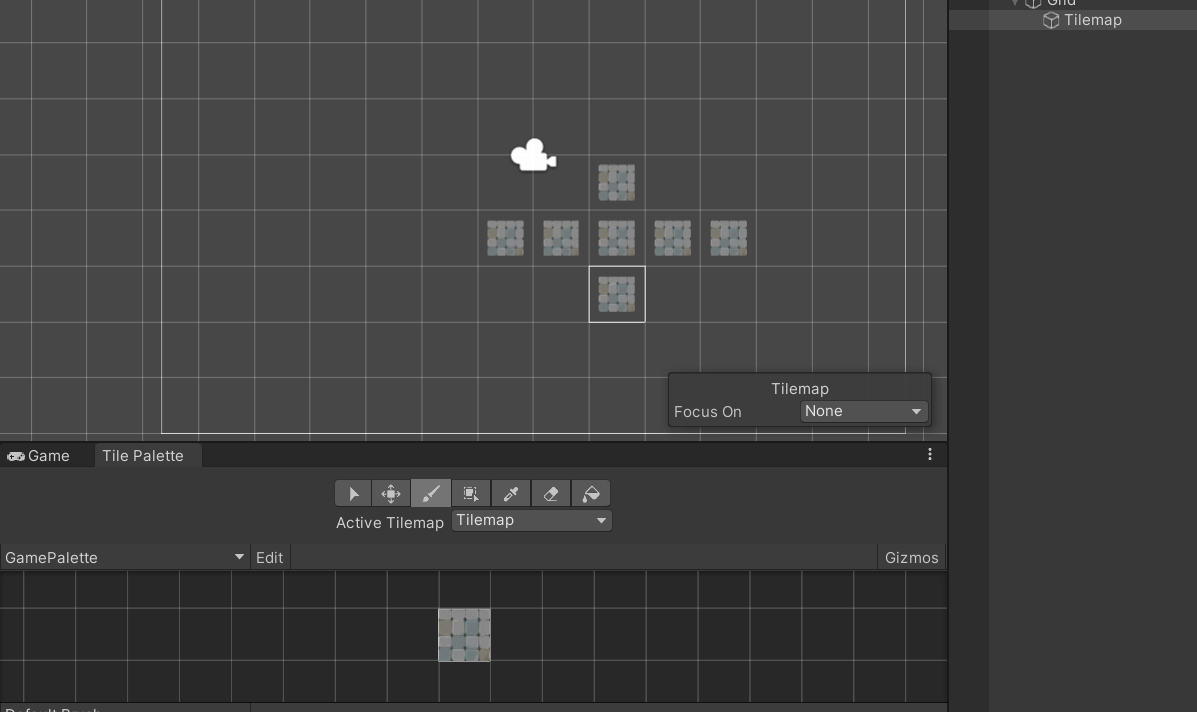
5. 从window->2D里创建tile palette





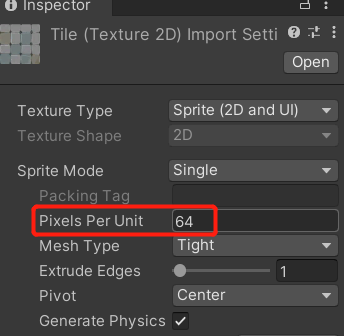
6. 绘制tile

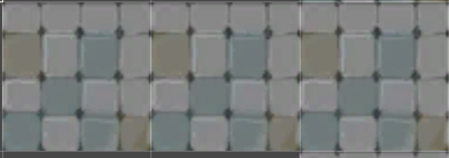
将tile素材放到画板上就能用它来绘制了。



7. 绘制连续不间断的块

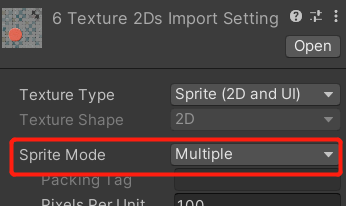
问题的根本是图片的大小（64x64）不满足一个Unity单位的大小（默认100x100）。所以需要将Pixels Per Unit改成图片的大小。



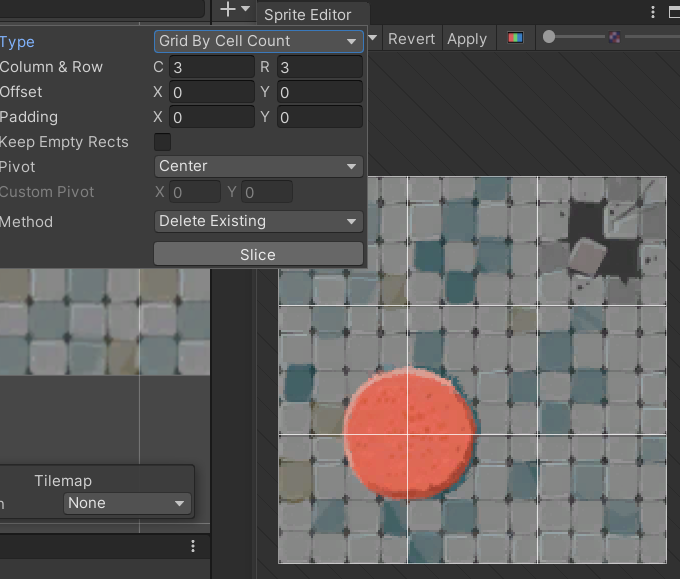
8. 图集的生成：

将Sprite Mode改为Multiple即可，用于图片切割。

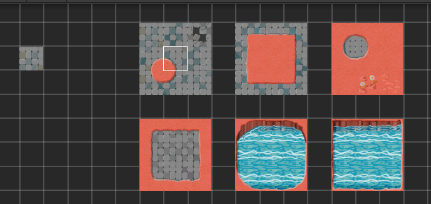


9. 图集切割

在sprite editor中切割：例如切成9等分。



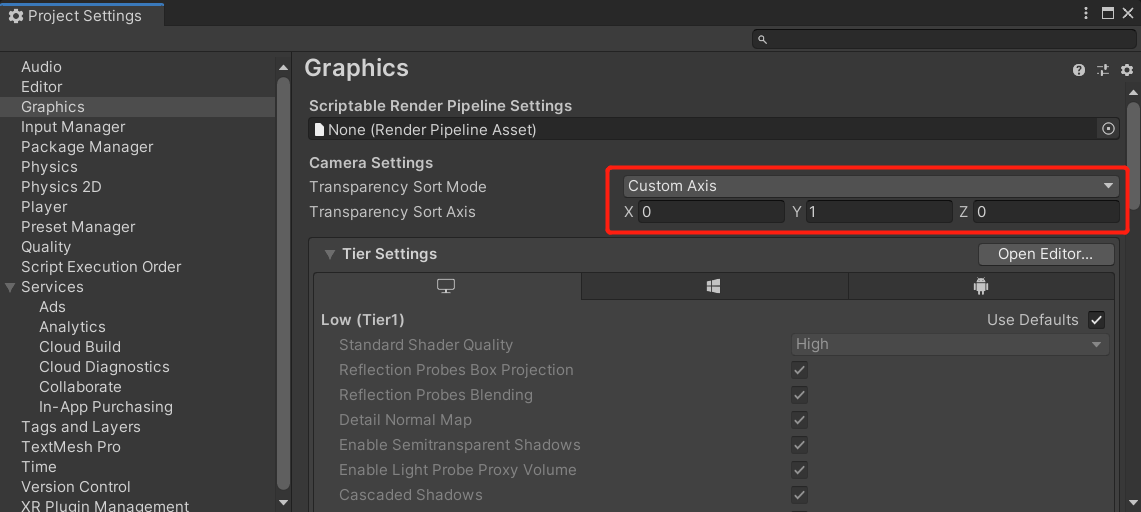
10. 将切好的图片放到画板中，就可以自由画地图了



**第三部分：2D游戏透视效果小技巧**

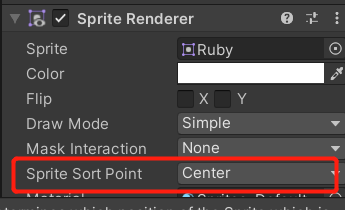
1. 2D游戏中常见的一个效果实现：角色在物体上方的部分不被遮住，而和物体重合的部分才会被遮挡。

首先是把透明度分类模式改为自定义的，然后将标准改为Y轴而不是Z轴。之后要用Y值大小来决定UI的遮挡关系。



接着关注渲染点的设置，unity是以物体的这个点来进行渲染顺序的比较。

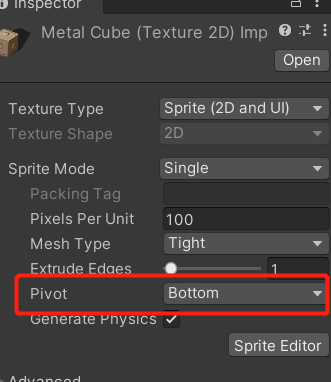


如果上述改成锚点模式，那么想要改变锚点的位置，需要修改Center并且角色身上要有Rect Transform组件。

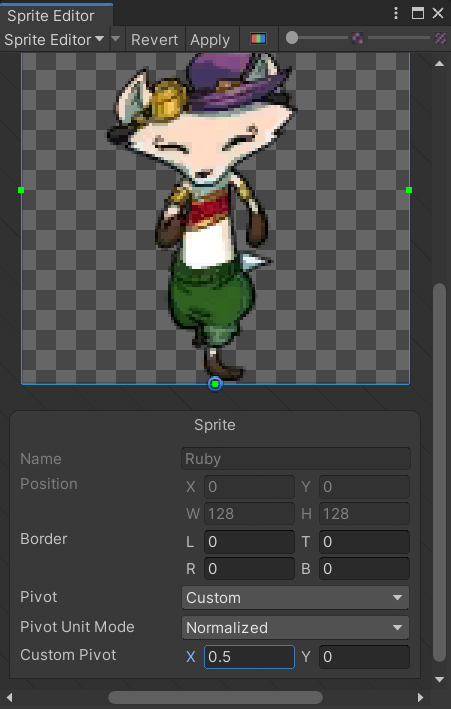


修改锚点的两种方式：

（1）对于单个物体来说可以在sprite的inspector面板里修改



（2）在sprite editor中修改



效果完美：

Ps. 预制体的编辑可以取消自动保存



**第四部分：游戏物体的碰撞处理**

1. Unity物理系统

RigidBody是用来进行受力作用的，而collider是用来进行碰撞检测的。

发生碰撞的条件：双方都有collider，而其中一方有rigidbody，最好是运动的那一方带有rigidbody，因为rigidbody长时间不发生碰撞检测会进入休眠，降低性能的损耗。

2. 编辑碰撞器

绿色的框就是碰撞检测边界，可以自行修改，通常是用矩形，如果想更加精致的话得单独为物体制作和轮廓匹配的碰撞器。





3. 碰撞时发生旋转的处理方式

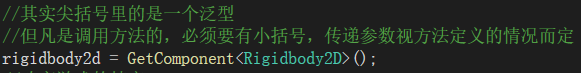
冻结某个坐标轴的旋转。



4. 不停碰撞时的抖动问题

原因是代码控制的位移和物理系统对刚体检测的矛盾。解决方法是获取rigidbody，调用rididbody的MovePosition方法来控制移动。







5. 静态方法和成员变量（普通）方法的调用

静态方法：类名.方法名（参数）。全局访问，无需实例化。



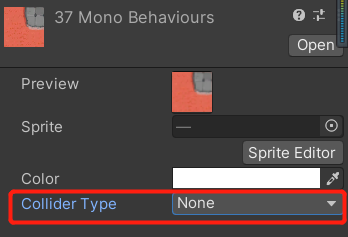
成员变量方法：对象名.方法名（参数）。需要实例化。



6. Tilemap碰撞器



局部添加：将不需要碰撞的瓦片的collider type属性值改为None就可以了。

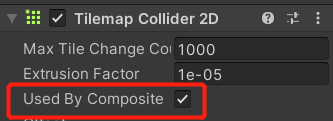


7. 优化tilemap碰撞器的方法

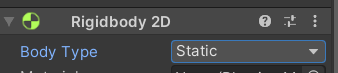
使用组合型碰撞体组件，可以降低碰撞性能的消耗，并且避免多个碰撞体之间存在缝隙的情况。



对需要组合的碰撞体要勾选上使用组合的选项。



将静止物体的刚体改为静态类型，也能降低性能的消耗。

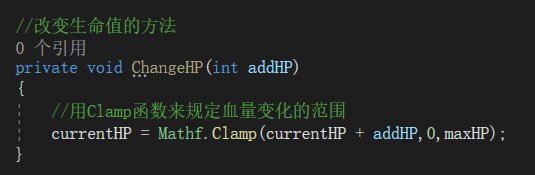


**第五部分：游戏物体的基础逻辑**

1. 主角的生命值添加与修改









2. 移动的调整，方便速度的更改



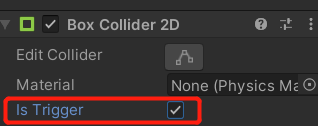


3. Unity赋值顺序：

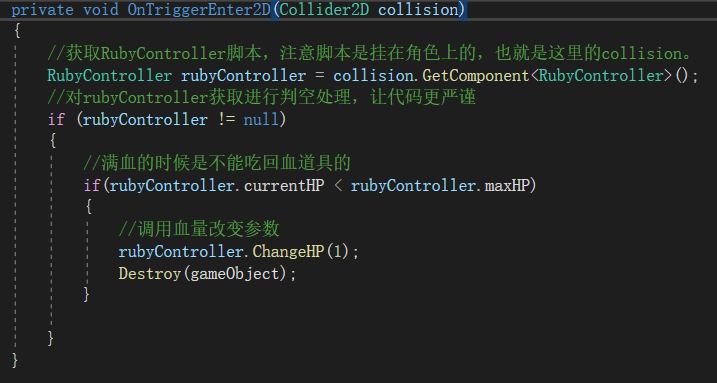
声明变量初始值赋值->Inspector面板->Start方法

4. 回血道具的触发事件

给collider勾选上trigger就可以把它设置为触发器，发生碰撞时会产生事件



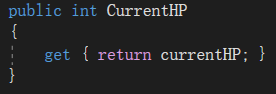
脚本编写



5. 属性

对于有读写限制需求的变量，需要将其声明为private，且写出它的属性，视情况添加get（允许读）和set（允许写）方法。





外部是直接通过属性进行变量的访问



6. 陷阱

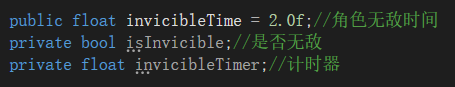
和加血的几乎一样，只是把传递给生命值增加方法的参数从1改成-1，取消对血量上限的判断，取消碰撞销毁。

7. 陷阱持续掉血的实现

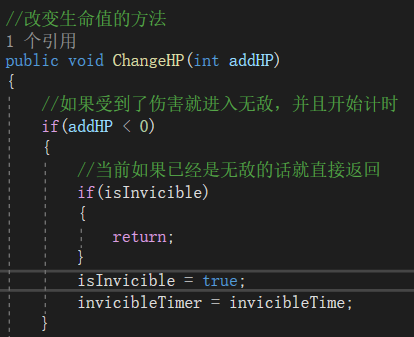
把之前的OnTriggerEnter2D改为OnTriggerStay2D



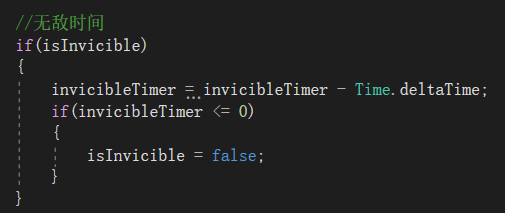
但这个方法是逐帧调用的，会导致角色瞬间死亡，所以需要定义一下角色受到伤害后的无敌时间。一般和时间有关的事件都会有这样的三个变量来进行处理。



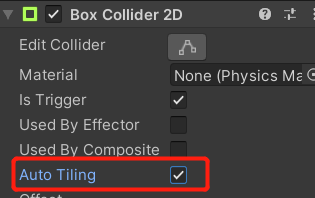
对无敌的触发条件进行处理



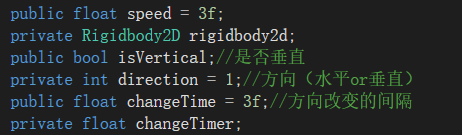
在update方法中用计时器累减来控制无敌时间，无敌时间到了就讲无敌状态变量置为false。

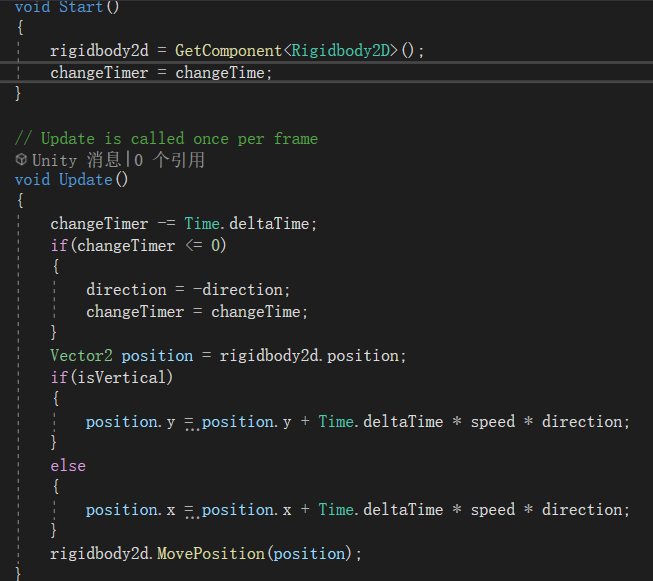


8. 刚体勾选自动平铺后会随着sprite的改变而进行自适应。



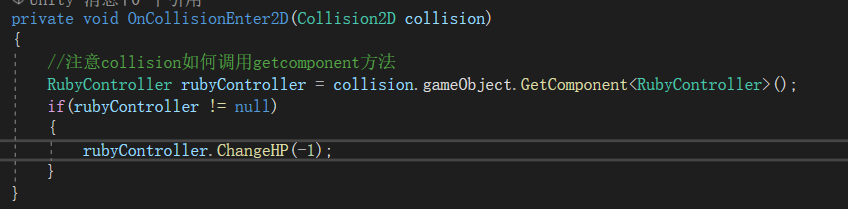
9. 敌人的控制脚本





10. 敌人的碰撞检测

注意collision和collider的区别，collision没有封装GetComponent方法，需要先获取挂着它的GO后才能调用GetComponent方法。



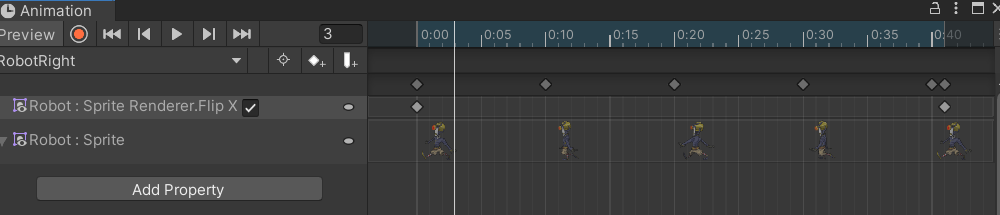
**第六部分：游戏物体的动画制作**

1. 动画的制作与添加

做sprite动画的时候注意观察顺序是否是对的，要让动画看起来是正确的。



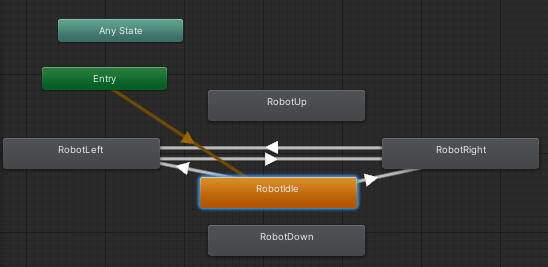
想要让图片翻转，可以用sprite renderer组件上的flip，水平翻转用x，垂直翻转用y。

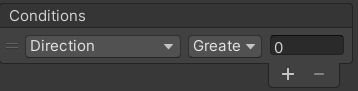


2. 动画控制器控制动画播放。

Parameters是状态切换的条件变量。

在transition的conditions里面可以将条件变量设置好，例如这里是通过判断小于还是大于Direction来决定物体播放朝左动画还是朝右动画。





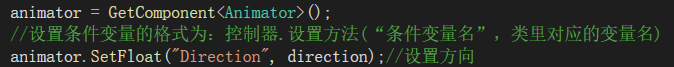
3. 代码控制动画播放器的播放。

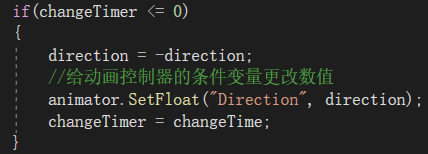
Ps.重载函数：函数名是一样的，但是传递的变量数不一样。

在Start方法中要先初始化条件变量，不然的话要先等第一次计时器倒计时为0的时候才会切换。

设置条件变量的格式为：控制器.设置方法(“条件变量名”，类里对应的变量名或者某个具体参数)。

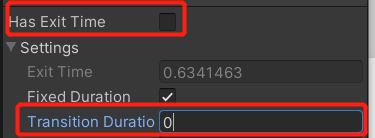






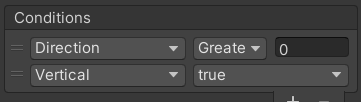
4. 完善移动的切换动画。

如果不想让动画有过渡，而是瞬间切换，就做如下处理：先取消Exit Time，然后把过渡的持续时间设为0。

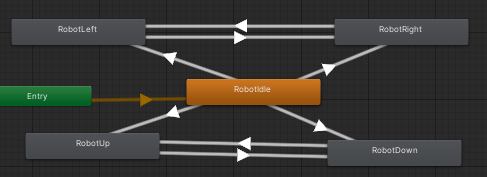


控制水平和垂直移动的变化条件









5. 混合树

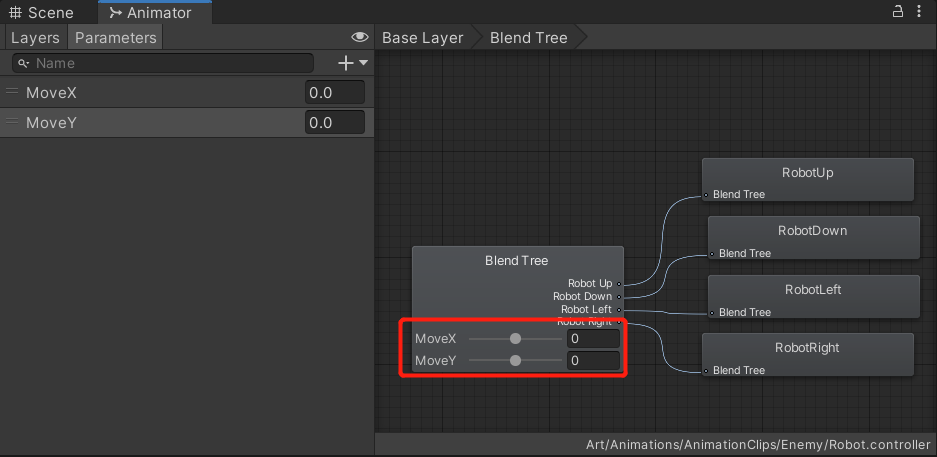
创建方式：动画控制器面板右键 -> Create State -> From New Blend Tree



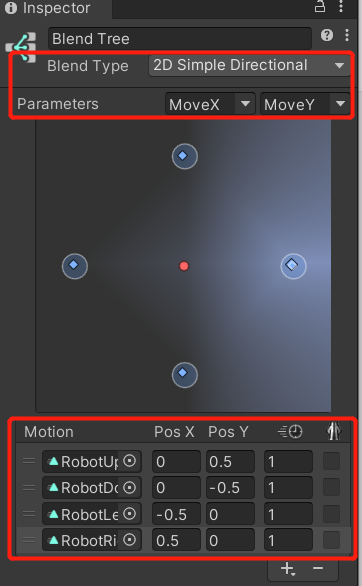
它的好处是能取消动画状态间的连线，同时用1个或多个条件同时控制不同动画状态间的切换。

注意对【相似性质】的理解。Eg. 根据不同的受伤值或者部位来播放不同的受伤动画。

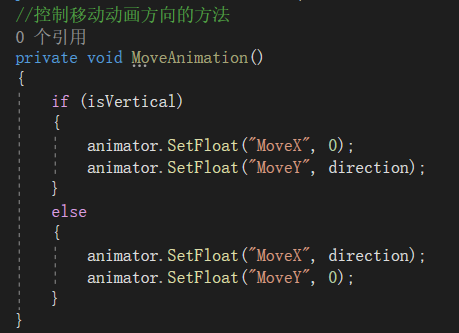
混合树对动画状态的控制其实就是将动画状态放置在一个二维坐标系下，而切换条件便是坐标值，通过坐标值看离哪个动画近就播放哪个动画。



Blend Type决定了条件变量的个数。下面的Motion可以指定动画状态，右侧的坐标则是在其上方二维数组中的位置。

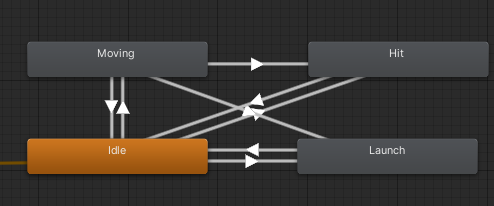


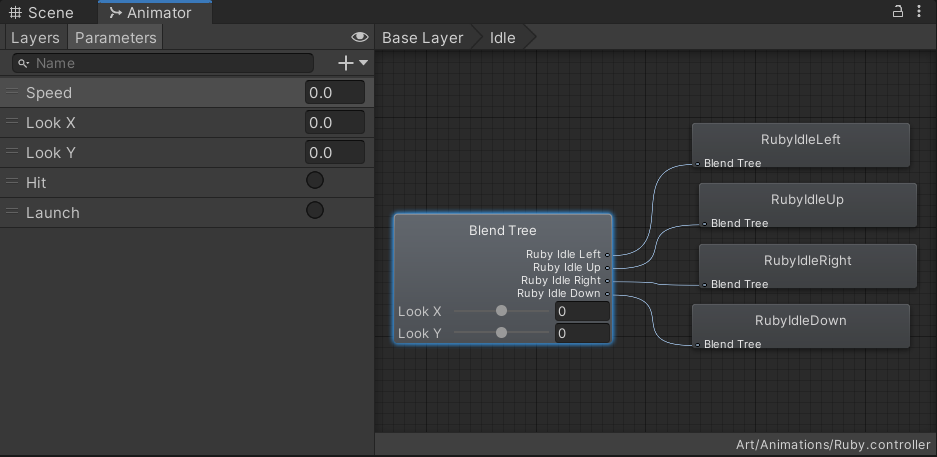
回到代码里来控制这两个变量。然后把这个方法分别在start和update方法里调用。

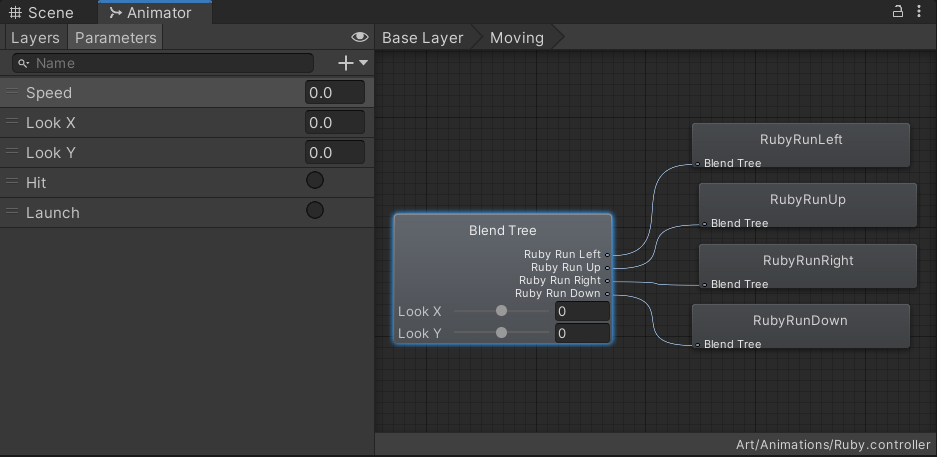


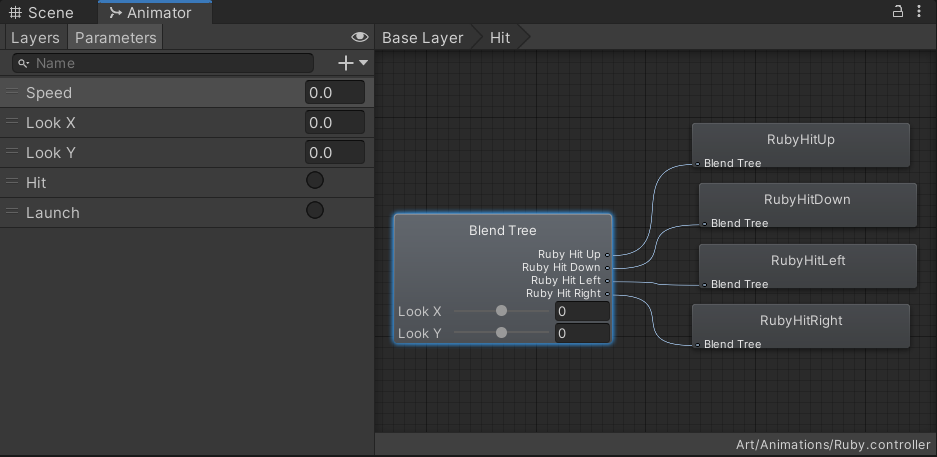
6. 官方做的主角混合树。

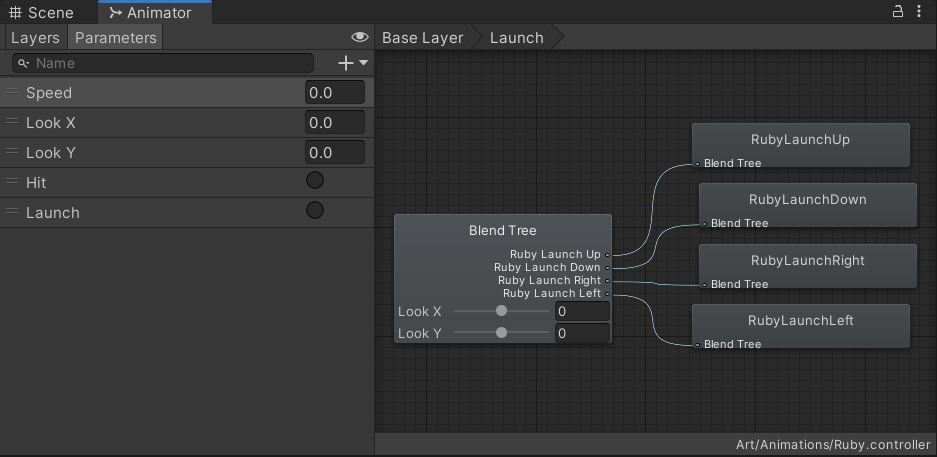
Look X和Look Y用来控制主角在静止的时候是看向哪里，这俩变量是同时控制Idle和Move动画的。每个状态其实都是一个混合树。Idle和move的状态切换条件是速度大小。受击和攻击的条件都是trigger瞬时触发。











7. 主角动画控制器的代码控制







效果完美~



**第七部分：子弹的逻辑**

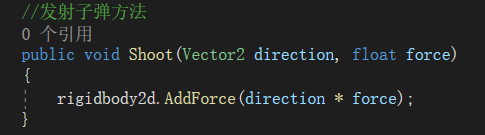
1. 子弹

位移的三种方式：

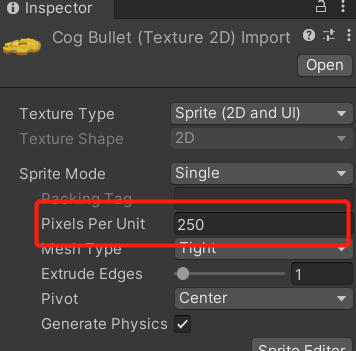
（1）改变transform的position属性。

（2）调用rigidbody的Move方法。

（3）给rigidbody施加外力。



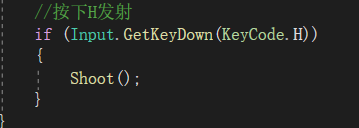
改变如下的值可以改变图片大小，值越大图片显示越小。



角色控制脚本里增加发射子弹的方法



Update里按钮按下调用

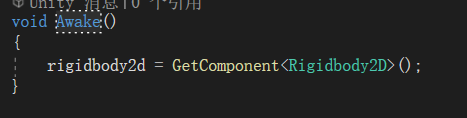


修BUG



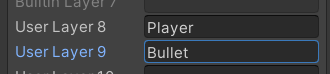
原因是在角色控制脚本中实例化子弹的时候并没有在子弹脚本的Start里面获取刚体，而直接调用了发射方法，因此会报空。

解决方法：将Start改为Awake，Awake是在所有Start之前执行的，无需关注GO是否enable。

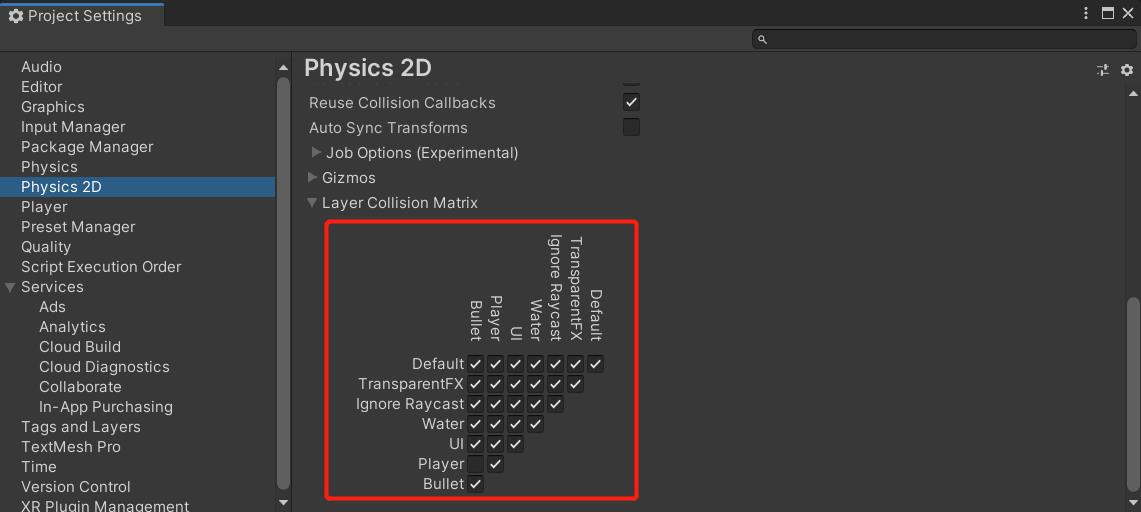


2. 通过层级（Layer）控制碰撞检测

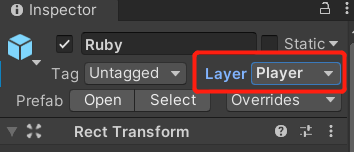
需解决的问题：子弹不能与角色进行碰撞。



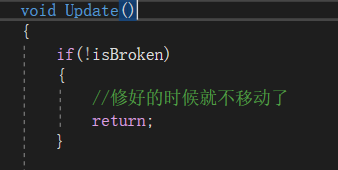
然后在Physics设定中将Player和Bullet取消勾选后就可以不产生碰撞检测。

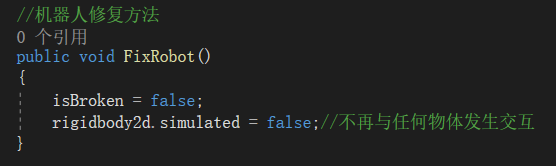


别忘了回去设置Layer

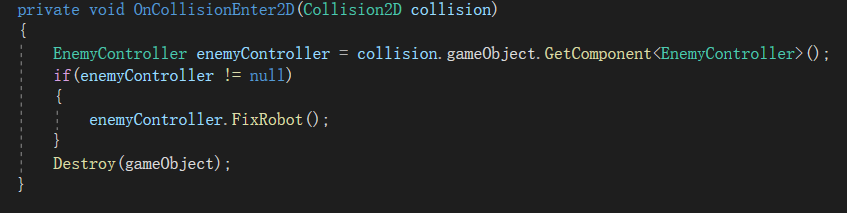


3. 机器人修复脚本





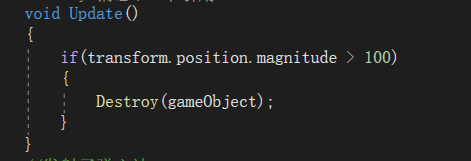
子弹的碰撞检测方法，获取机器人的脚本，然后调用修复方法。



4. 子弹的销毁

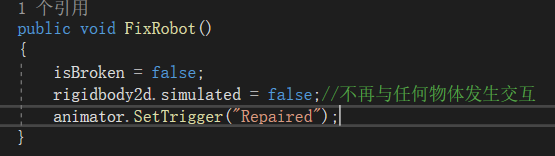
子弹飞出屏幕后要自动销毁，不然会消耗性能。

这里可以采用模长来判定，无需考虑方向。



5. 机器人修复动画

制作过程还是常见的帧动画，做完后记得在修复方法中设置动画状态的切换。

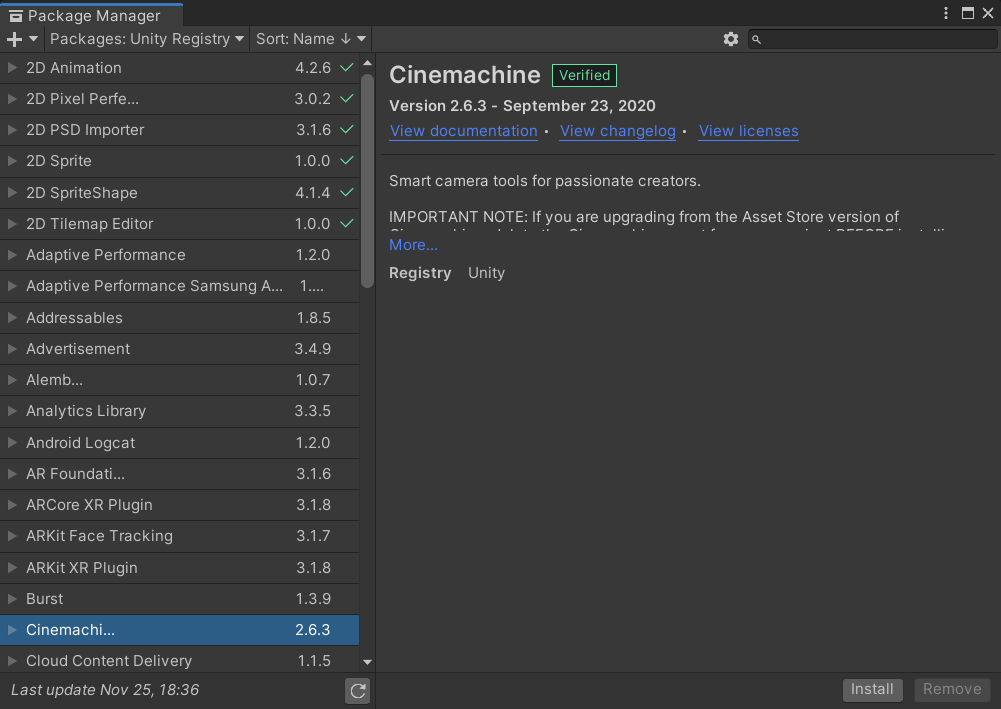


**第八部分：场景完善、摄像机、特效。**

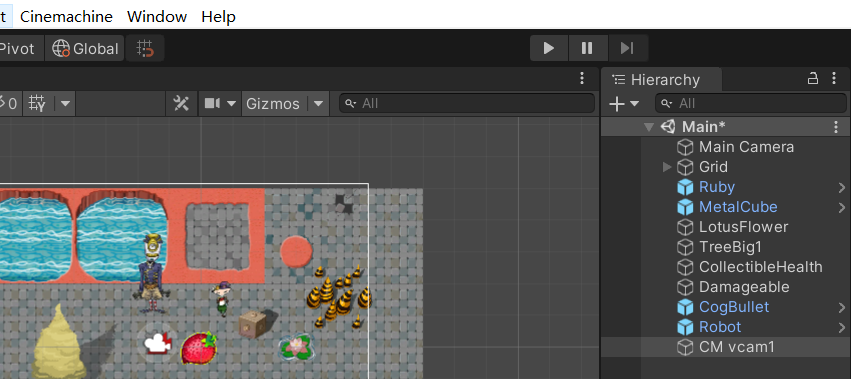
1. 摄像机跟随角色移动

如果自己去写摄像机的跟随脚本会比较麻烦，这里可以用unity自带的Cinemachine，里面封装了许多摄像机的功能和效果。

首先打开window -> Package Manager。记得改变筛选条件，然后找到Cinemachine，进行下载。

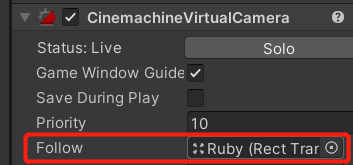


点击Cinemachine就可以创建虚拟摄像机，同时Main Camera上就有了一个CinemachineBrain组件。使得虚拟摄像机的渲染内容可以反馈给Main Camera。





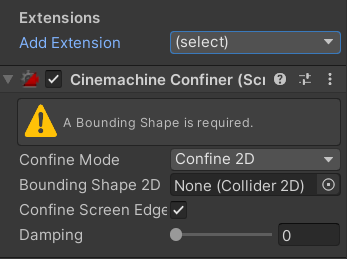
将需要跟随的对象指定到虚拟摄像机的Follow属性中。



2. 跟随边界的限制

角色运动到边界的时候相机就应该产生阻挡效果，不能越过边界。

虚拟摄像机有个Extensions属性，点击Add选取Confiner后添加边界的限制。需要为它添加一个2D碰撞器对象（空对象加一个碰撞器，然后修改碰撞器的尺寸）来当作相机的边界。

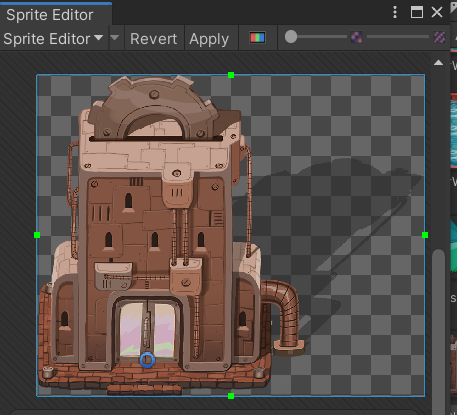


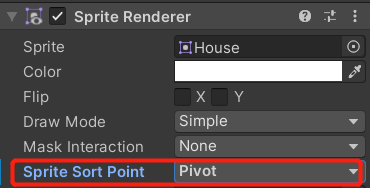
至于游戏的边界碰撞体，也就是空气墙，我们创建4个空物体，添加collider，让它们包围游戏UI的边界即可。

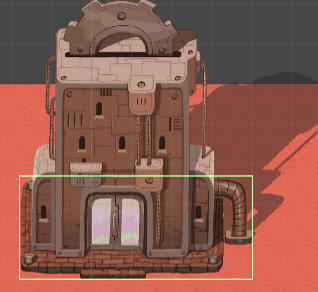
3. 完善游戏场景

注意其它2D游戏物体的制作都和之前的盒子一样，需要考虑主角和它们的遮挡关系以及碰撞关系。

房子的制作：改变轴心点来控制遮挡关系，改变碰撞体来让角色不会发生穿模现象。



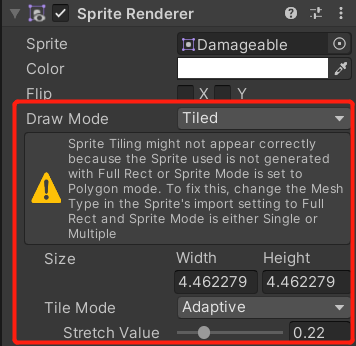




效果：

这里可以更改图片的绘制模式，如果是tiled平铺类型，就可以在放大图片时根据Tile Mode来进行平铺。

之后的环境物体制作都大同小异……

5. 粒子系统

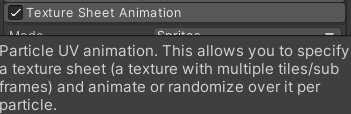
创建方式：层级面板 -> 右键 -> Effects -> particle system

先切粒子的sprite，改成Multiple模式。



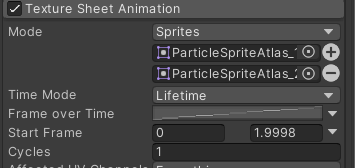


勾选纹理图片动画，看说明知道是制作粒子UV动画的。

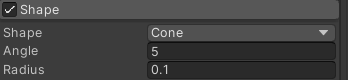


Mode选为sprites后粒子的素材就可以选sprite了。下面可以添加素材。

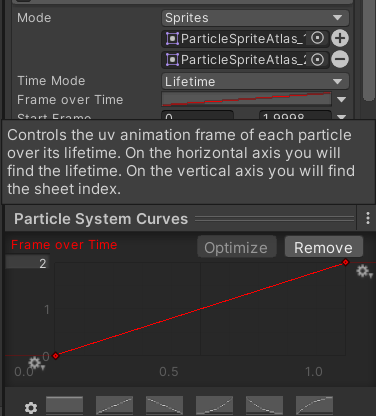
Start Frame可以换成随机粒子，此时会从粒子素材的数组中进行随机读取，注意数组下标从0开始，不包含右边界。



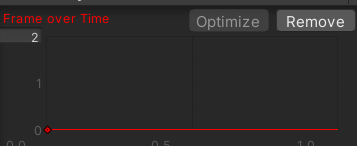
这里可以改变发射器的形状、角度和半径，和AE里的挺像的。



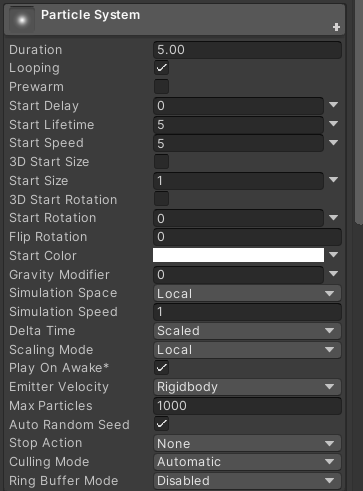
实机演示中出现了烟雾粒子中途改变样式的情况。这是由于Frame over Time这个属性影响的。看说明知道x轴标识的是粒子数组的下标，按照曲线的走势，到后面会改变粒子数组的下标，因此粒子的样式就改变了。



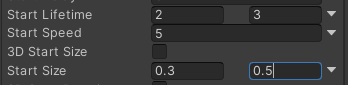
把曲线删除即可。



基础属性，很熟悉啊，AE里的P粒子系统属性既视感。

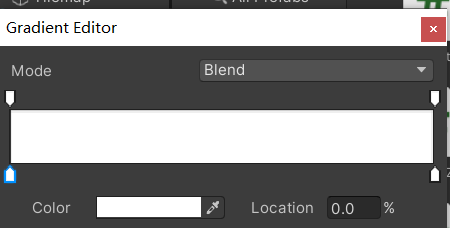


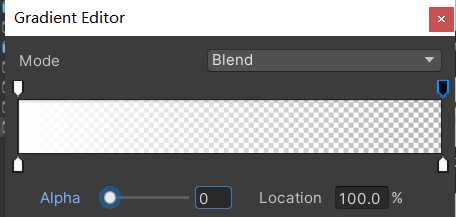
很多属性都可以设置为一个区间内的随机值。



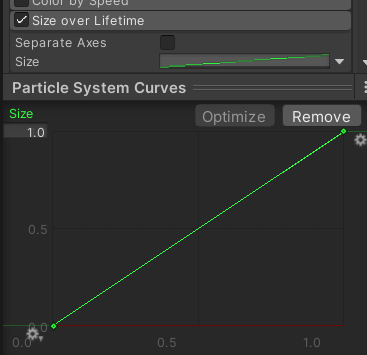
颜色和透明度随时间的设置：优化烟雾的消失过程。





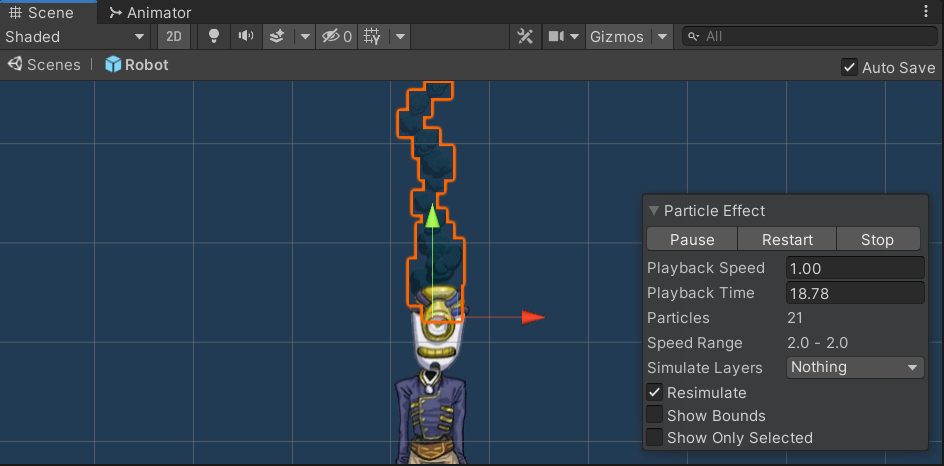


大小随时间的设置：优化烟雾先大后小的变化。

给机器人添加烟效果：只需让粒子系统成为机器人的子物体，记得在Prefab里面把粒子发射器移到机器人头上。





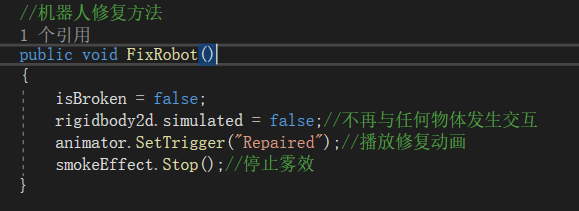
让烟雾粒子不会跟随物体移动：Particle System下把模拟空间从Local改为World即可。这样它们就会以世界坐标为基准，不考虑所处的模型。





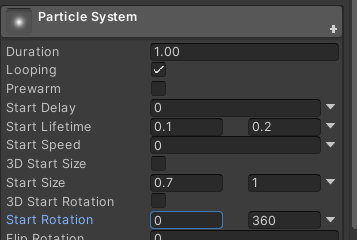
6. 脚本中对粒子特效进行控制



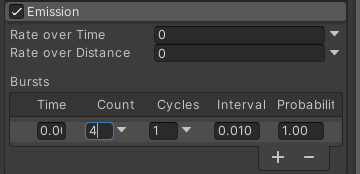


7. 爆炸特效

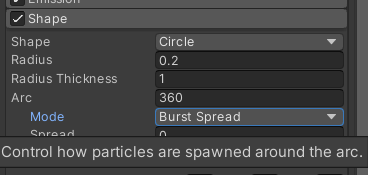
爆炸属于瞬时的特效，和持续循环的特效有所区别，前面的设置主要是按照瞬时性做的。



在发射器选项中可以修改瞬时爆发属性：将Rate over Time设为0，瞬时特效不需要随时间的属性。



再修改发射器的相关属性，其中Arc的Mode是指粒子如何再发射器周围产生的，这里换成爆发速度。



最后就是改尺寸和透明度……然后把Loop取消掉。

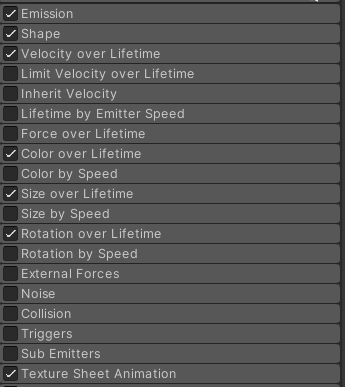
Ps. 按住shift拖动直线的点不会产生曲线。

7. 拾取道具特效

Particle System、Emission和Shape的设置和子弹爆炸的差不多。这里主要要改一个粒子在生命周期内的速度变化。把speed modifier改成曲线并调整。



要调的属性如图所示。



**第九部分：UI和NPC。**

1. UGUI

再次复习下Canvas的三种模式

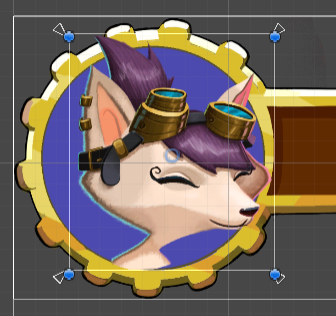
屏幕空间-覆盖/摄像机：固定在屏幕前的UI，比如一些信息面板。

世界空间：和游戏物体有直接关联的UI，比如血条、暴击伤害等。

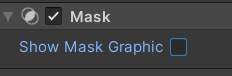
……

2. 制作主角血条

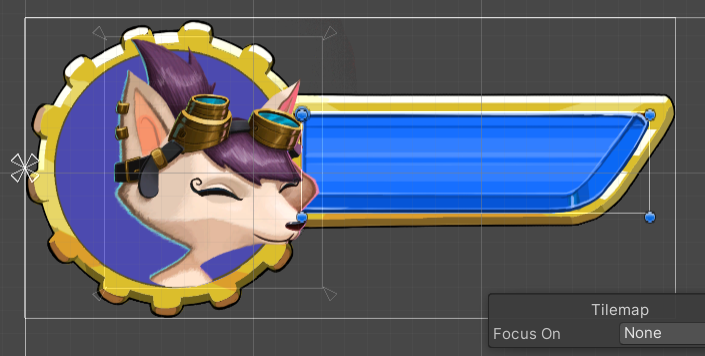
复习下锚点和轴心点对自适应的控制，即轴心点和锚点之间的距离（4合1）或者比例（分散开）是保持不变的。如图，如果要让头像随着头像框（父物体的一部分）一起等比缩放的话，可以把它的锚点设置在它的Rect Transform的四个角上。

使用遮罩来实现血条的扣除效果，为创建一个遮罩Image并为它挂上Mask组件。然后血条的Image要成为Mask的子物体。

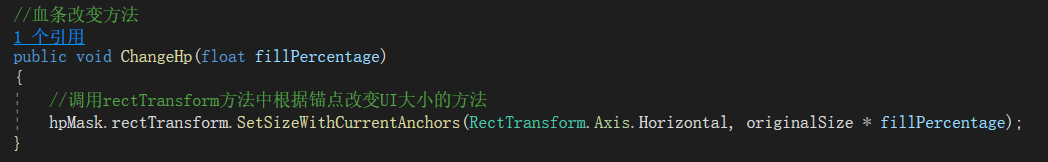
 

为了通过Rect Transform的属性来控制Mask基于左边的水平缩放，需要让它的轴心点移到最左边。这样才能保证Rect Transform中的Pos X不发生改变，因为此时锚点和轴心点之间的距离不会因为缩放而发生改变。此时就可以通过访问width属性来控制Mask的缩放了。



3. 控制血条UI的脚本





在主界控制器脚本中调用血条改变方法





4. 生命周期函数&单例

执行顺序：Awake > Start > Update

单例的好处：

平时如果像访问其它类里的成员变量或方法时，需要先获取这个类（例如Getcomponent）。而如果将一个类做成了单例，那么就可以在其它类里直接通过单例来调用类里的成员变量或方法。但是要注意，顾名思义，单例尽量是唯一的。单例多了会造成更多的内存占用，以及造成交叉引用的问题。

5. NPC和对话框的制作

做好NPC后，首先要单独为NPC创建一个Layer，用于后续的射线检测。

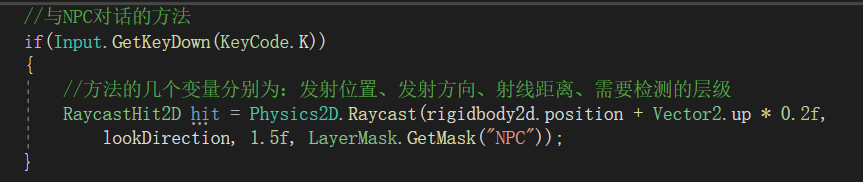


然后实现与NPC对话的功能：

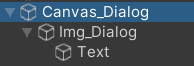
方法一：用一个空物体做个触发器。

方法二：射线检测。

使用方法二来实现，思路是让主角向某个方向发射射线，当射线检测到NPC时，就触发对话。



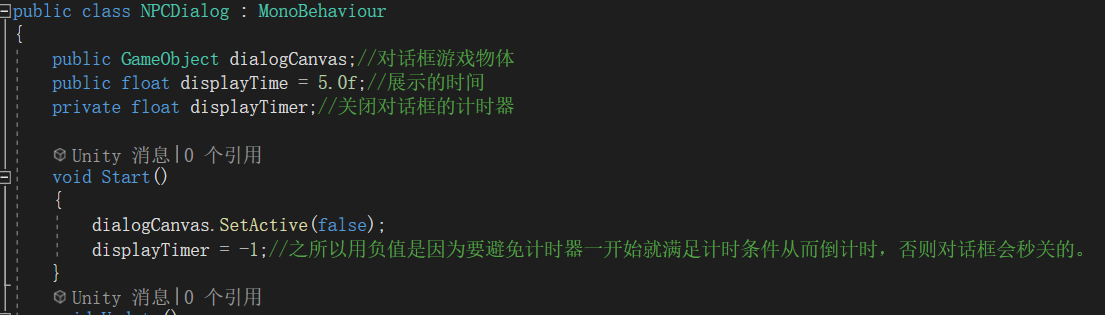
接着制作对话框，需要一个世界空间下的Canvas。

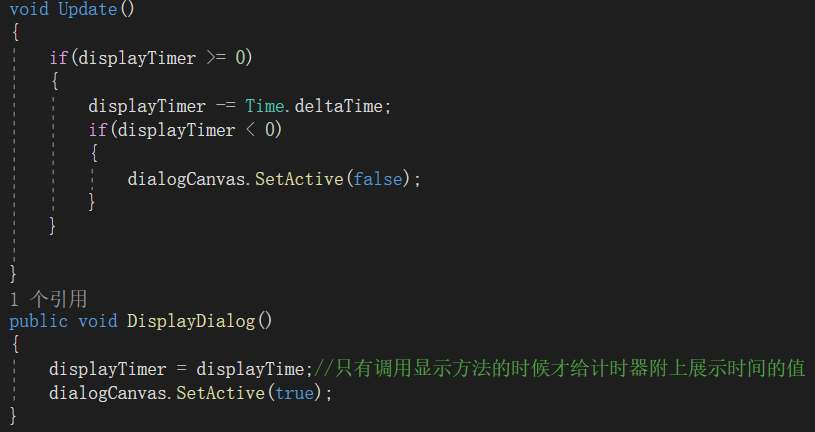


字体用到了outline组件来描边。



再用脚本控制对话框的显示。

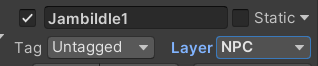




在主角控制脚本中射线检测时调用显示方法。



别忘了把NCP对话框脚本挂载到NPC身上，也别忘了指定NPC的Layer。

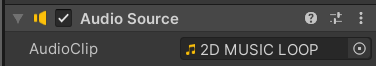


**第十部分：音效**

1. 添加音效

BGM：创建一个空物体，然后给它挂载Audio Source组件，并把音乐资源赋值给AudioClip。

另外要注意Camera上得有Audio Listener组件，这样才能正确接收音乐播放，其有且仅有一个Audio Listener。



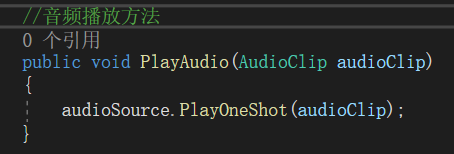


Audio Source简单的属性很容易理解，其它的去官方手册看。

为主角控制脚本添加音效播放的代码（别忘了给主角预制体挂上Audio Source组件）：







这样一来就能够在其它脚本中的一些事件中调用播放方法来播放对应的音效。

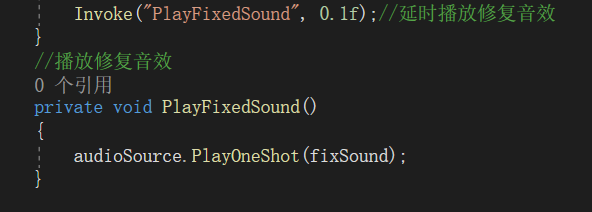


这里记录下如何播放随机音效，思路是创建一个audioClip数组，然后播放音效的时候随机从里面去一个出来。

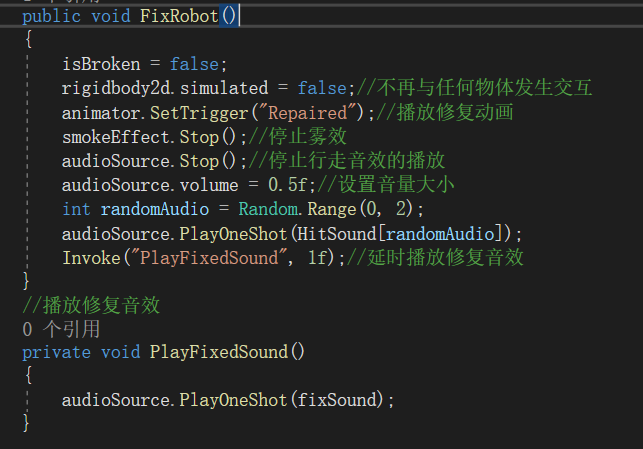




延时播放：当一个事件会触发多个音效播放时，要避免它们同时播放。

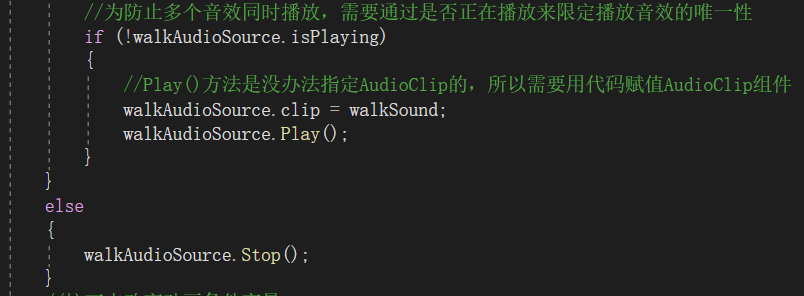


最后补上机器人修复好后停止走路音效的代码。



为主角的移动添加音效。由于主角身上有持续循环的音效（走路）和仅播放一次的音效（射击、受伤等）且会同时发生，所以需要有两个AudioSource去分别处理两种类型的音效。别忘了在外面主角身上再多挂载一个AudioSource组件。





另外代码中可以通过auidoSource类里的volume值来设置音量大小。



**第十一部分：任务和查缺补漏**

1. 补充添加一些特效

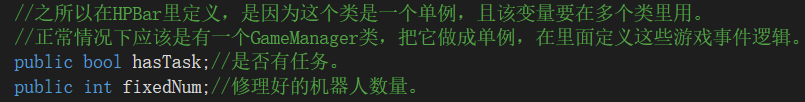
机器人被击中的特效：

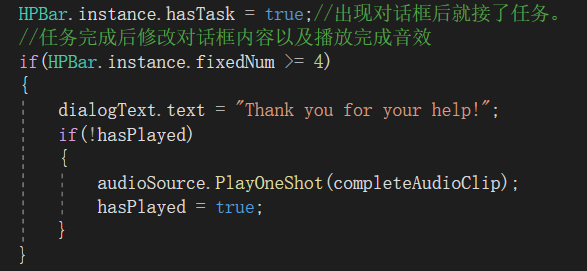


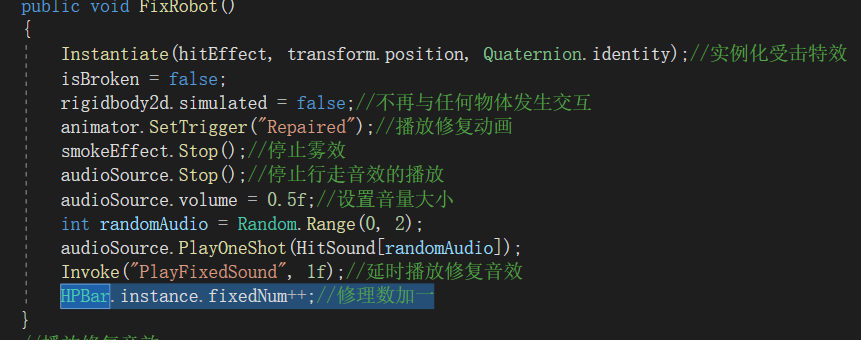


同理添加回血道具的拾取动画。

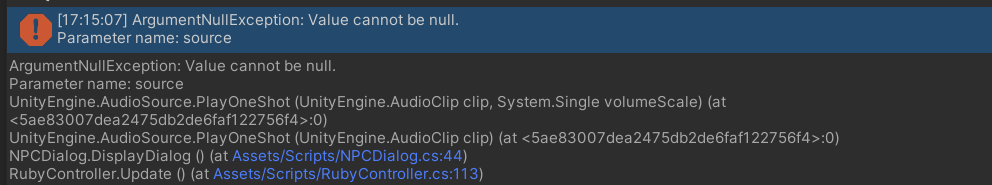
2. 任务系统



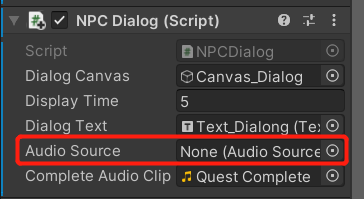




解决BUG：



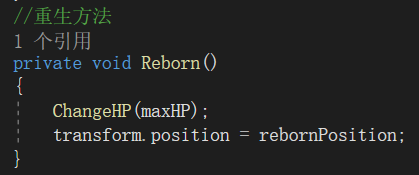
发现是Audio Source忘记赋值了。



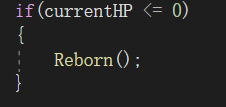
3. 主角复活

复活后满血+位置重置。





ChangHp()方法中调用重生方法。



4. 打包

……