逼真的眼球运动

2.1.3版

对一个已有预设的角色进行快速启动。

你也可以观看快速入门视频。

REM为用Mixamo Fuse、Autodesk Character Generator、MakeHuman、Character Creator 3、DAZ Studio和UMA生成的角色配备了预置。如果你有这样的角色。

将Scripts文件夹中的LookTargetController和EyeAndHeadAnimator脚本添加到你的角色中。点击 "导入 "按钮，为你的角色导入预设。完成了!

关于逼真的眼球运动

谢谢你购买逼真的眼球运动!我希望这个资产能帮助你把你的角色带入生活。如果你有任何问题或建议，请给我留言，tore.knabe@gmail.com!

关于此资产的讨论，请访问Unity论坛的主题。

这个资产的网页是Unity资产：真实的眼球运动。

真实眼球运动（REM）可以控制你的角色的眼睛、头部和眼睑运动，使他们以逼真的方式看向周围，看向玩家，或看向物体。你可以把它用在眼睛与Mecanim人形骨架相连的角色上，或者用在有两个独立的眼睛游戏对象的角色上。你可以在角色的环境中指定兴趣点让角色看，或者让他们闲着看，或者让他们在玩家进入视野或一直盯着他们看时自动看玩家。

这些动画使用了已发表的研究论文中的数据。更多信息请见我的博客文章。

这个资产只提供控制动画的脚本，没有网格或着色器。对于逼真的眼睛网格和着色器，我推荐Scruvystorm Studios的RealEyes资产或Tanuki Digital的Eye Advanced资产，也有Ricardo的Realistic Eye资产或RRFreelance的Eye Shader资产。你可以在网络播放器的演示中比较这三种资产。

该资产只有在场景开始时头部相对于角色变换直视前方时才能正确工作。如果你的角色的头部在编辑器中不是直视的（例如，看向角色的左边），请旋转它以直视。

如何使用

在RealisticEyeMovements/Scripts文件夹下有两个主要脚本。

LookTargetController.cs

选择看什么，何时转换看的目标。

眼部和头部动画师.cs

控制给定造型目标的眼睛、头部和眼睑的动画。

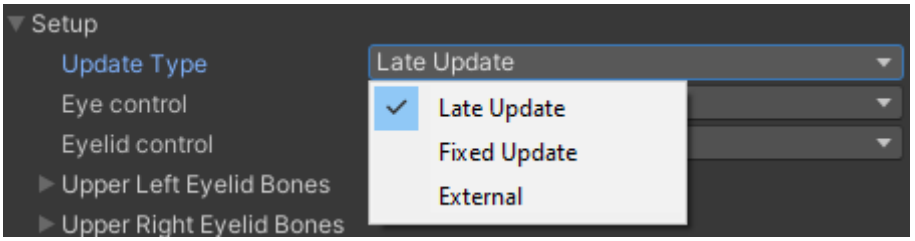
如果你想对看的位置和时间有更多的控制，你可以修改LookTargetController.cs或者用你自己的脚本来替换它，调用EyeAndHeadAnimator.cs的函数。你可能还需要修改EyeAndHeadAnimator.cs。

把这两个脚本拖到场景层级结构中的角色游戏对象上。然后，要么导入一个预设，要么继续进行下面的设置。

设置

*如果你的角色是用Mixamo Fuse、Autodesk's Character Generator、MakeHuman、Character Creator 3、DAZ Studio或UMA生成的，你可以通过导入相应的预设为其加载正确的配置，而不是像下面所说的那样手动配置眼睛和眼皮：见下面的**预设**部分。*

如果你有一个你还没有预设的角色，第一步是在
设置EyeAndHeadAnimator组件的折叠。



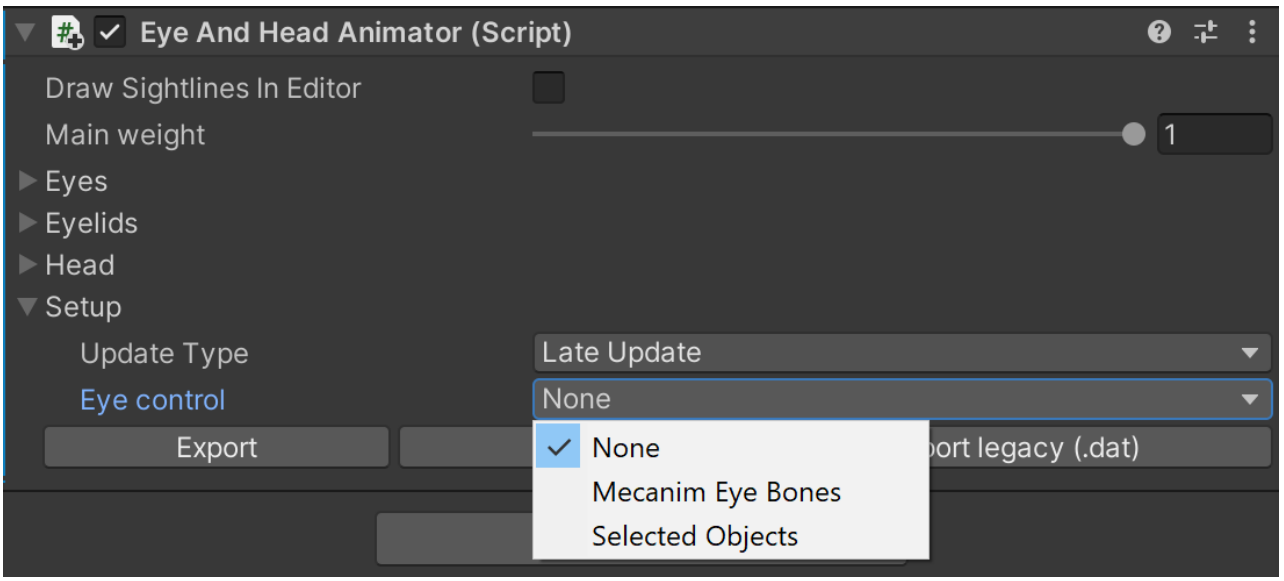
更新类型

对于大多数情况，你可以把**更新类型**留到**晚期更新**。如果你的动画师被设置为*Animate Physics*，你可以将其改为**Fixed Update**，这样当REM旋转头部时，物理学就会与角色的其他部分保持一致。这个设置只有在动画师确实被设置为*Animate Physis*时才会起作用。

外部设置可以让你更精细地控制头部和眼睛运动的具体更新时间。通常你不需要这样做，但在某些特殊情况下，比如你使用了不同的FinalIK组件，如LookAtIK、BoxingIK，而且更新的顺序很重要，选中这个框，然后每一帧都在EyeAndHeadAnimator组件上调用两个函数Update1和Update2（用正确的deltaTime作为参数）。确保在FinalIK调整头部方向之前调用Update1（因为Update1设置了头部目标），并在FinalIK调整完头部方向后调用Update2，因为Update2会移动眼睛。比如说。

```
eyeAndHeadAnimator.Update1（Time.deltaTime
）；lookAtIK.solver.Update();
eyeAndHeadAnimator.Update2（Time.deltaTime）。
```

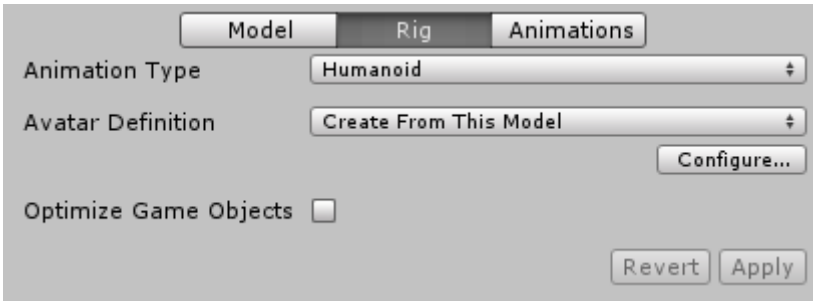
眼睛设置



将**眼球控制**设置为*Mecanim眼球骨骼*或*选定对象*。

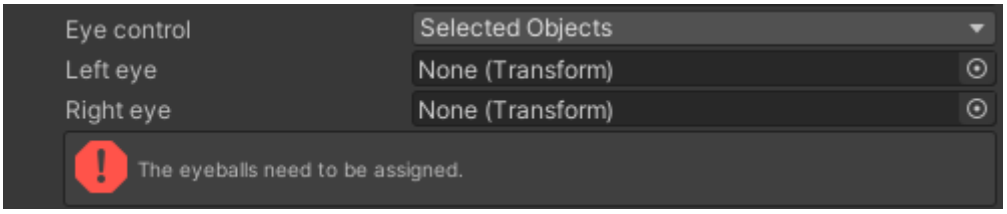
如果你选择*Mecanim眼骨*，你的角色必须有一个Mecanim类人装备和一个Animator组件。如果你的Mecanim装备有正确分配的眼骨，脚本会找到并使用它们来为眼睛制作动画。如果你的角色的眼睛网格是由Mecanim眼骨控制的，就选择这个选项。

如果你选择了Mecanim眼骨，而你得到的错误信息是
"找不到眼骨"，那么也许你已经在导入设置中勾选了 "优化游戏对象"。



请确保优化游戏对象没有被选中。即使你暴露了眼睛和眼皮的骨骼，这些骨骼的变换也是只读的，所以REM无法控制它们。

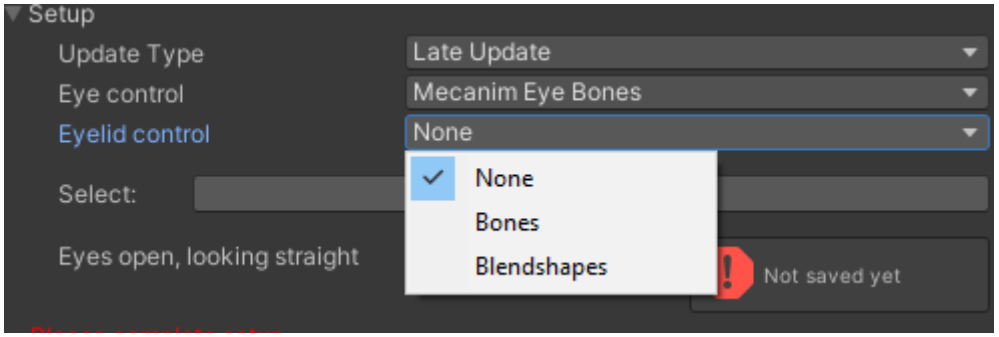
如果你选择了
"选定的对象"，你就可以在角色的对象层次中选择作为你的角色的眼睛的游戏对象，由脚本来控制。把每个对象拖到组件中相应的槽里。



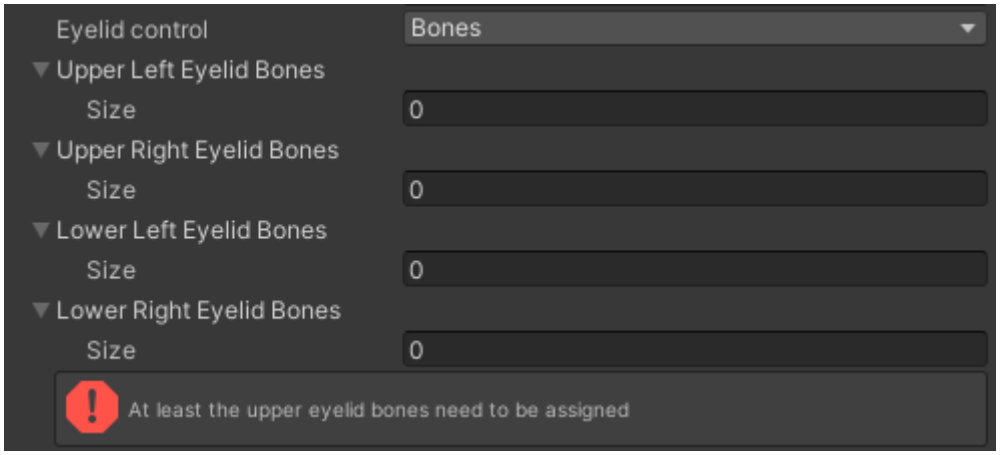
不支持通过眼睛网格混合形状移动眼睛。

眼睑设置

一旦你完成了眼睛控制，就会出现**眼睑控制的**方框。如果你的角色有可以由骨骼或混合形状控制的眼睑，REM可以控制它们以增加真实感。如果没有，请将**眼睑控制**框设置为*无*。如果你的角色确实有眼皮，把它设置为*骨骼*或混合*形状*。



如果你选择*骨头*，在你的角色的层次结构中找到眼皮的骨头，然后把它们拖到相应的槽里。每个眼皮槽可以接受一个骨骼阵列。大多数角色每个眼皮只有一根骨头，但如果你的角色有的话，你可以给每个眼皮分配几根骨头。



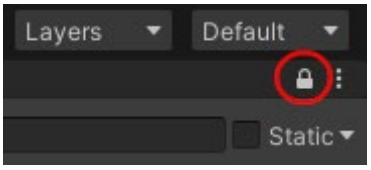
如果你的角色没有下眼皮可供制作动画，下眼皮的插槽可以留空。

如果你选择*Blendshapes*来控制眼皮，你不需要在这里分配任何东西。

保存职位

下一步是保存直视、仰视、俯视的位置，如果你启用了眼睑控制，还要保存闭眼的位置。这将告诉脚本眼睛可以向上和向下看多远以及如何移动眼睑的限制。

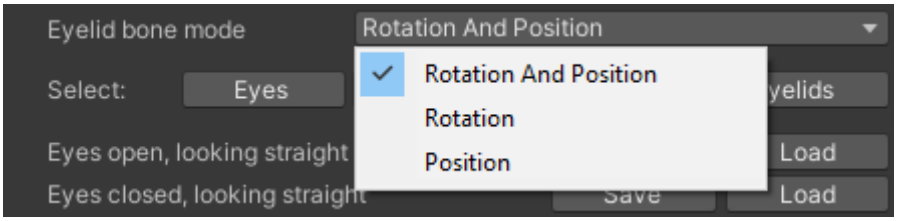
对于下面的步骤，我建议用检查器窗格右上方的小挂锁符号锁定拥有EyeAndHeadAnimator组件的GameObject的检查器标签。



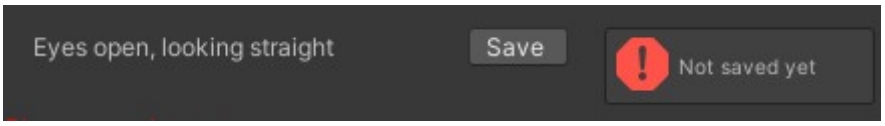
首先，如果有必要的话，通过旋转眼睛来确保角色看起来是直的。如果你将眼睛控制设置为*Mecanim骨骼*，那么就旋转眼睛的骨骼。如果你将眼睛控制设置为*游戏对象*，则旋转你指定的游戏对象。

如果你启用了眼睑控制，把眼睑设置成你希望角色睁眼直视时的样子。

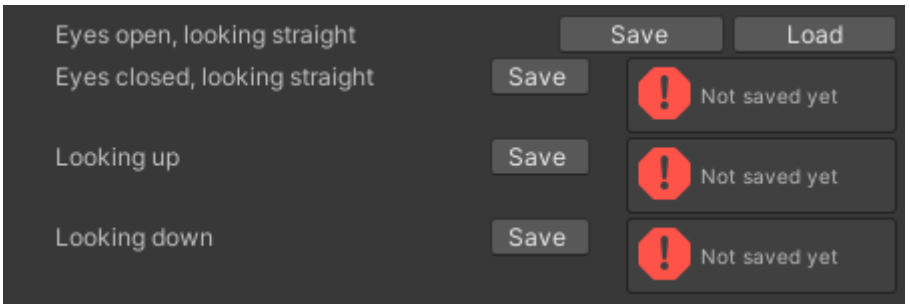
如果你通过骨骼设置眼睑控制，旋转或定位眼睑骨骼到所需的位置。你可以设置**眼睑骨骼模式**，只保存不同姿势的眼睑骨骼的位置、旋转或两者。



如果你将眼皮控制设置为混合形状，请设置各自的 SkinnedMeshRenderers 的混合形状值，使眼皮正确。当眼睛和眼皮在睁开眼睛直视时是正确的，按下**眼睛睁开，直视旁边的保存**按钮。

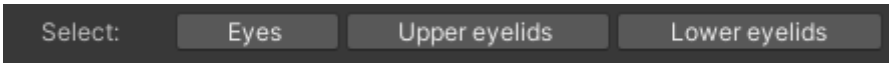


之后，会出现更多的按钮。



只有当你启用了眼睑控制，才会出现**闭眼看直线。**

对于每一条线，设置相应的眼睛和眼睑位置，然后按保存。最简单的方法是使用选择眼睛或下/上眼睑骨的便利按钮，然后为各自的姿势调整眼睛/眼睑。



对于**闭眼，直视**，确保眼睛再次直视前方，通过移动或旋转上眼皮向下，下眼皮向上，关闭眼皮，直到它们看起来适合闭眼的位置。这通常是最简单的，首先将两个上眼皮向下旋转，直到两边只看到虹膜的一部分。

然后将下眼皮旋转起来，闭上眼睛，在行中按下**保存 眼睛闭上，看直**。

现在按**Load**在行 **眼睛打开，直视**，以打开眼睛，使之更容易调整下一个姿势。

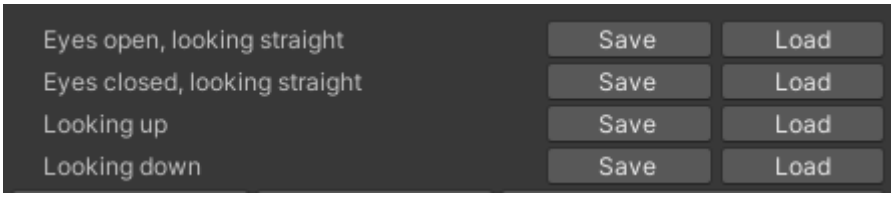
对于**向上看**，将眼睛向上旋转到看起来还不错的程度。这个位置将被保存为眼睛的最大向上角度。如果你启用了眼睑控制，请将眼睑调整到这个位置。在人类中，当我们抬头时，上眼皮会向上移动一点，下眼皮也会向上移动一点，所以眼皮会 "跟随
"眼睛。这为眼睛的运动增加了很多真实感。
然后在**向上看的**一行中按**保存**。

*提示：*如果你的眼睛骨骼或眼睛游戏对象的方向是这样的，很难让他们向上看，因为没有一个旋转轴是垂直于 "向上
"的，只要旋转整个角色，让他/她的头沿着全局Z轴看，并将旋转支点控制设置为*全局*。

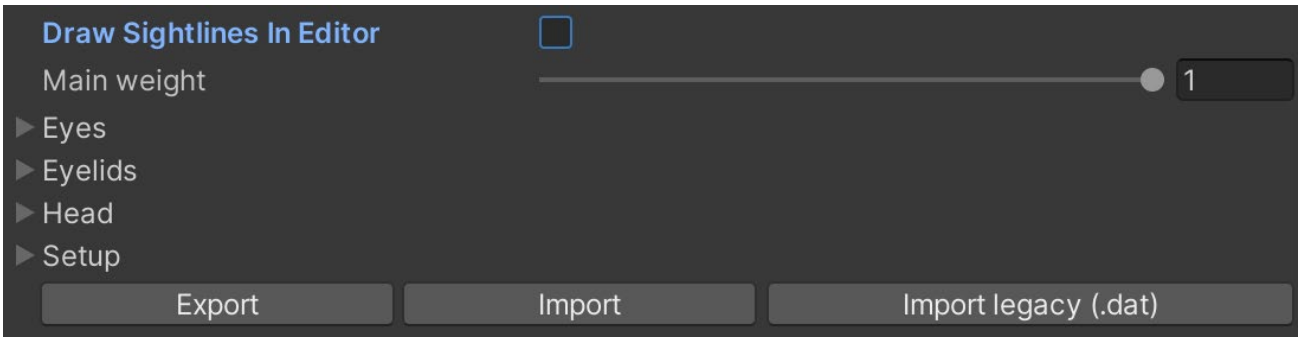
现在你可以轻松地围绕全局水平轴旋转眼睛了。你可以在保存完所有位置后再把你的角色旋转回来。

为 **"向下看**
"做相应的保存：将眼睛旋转到看起来很真实的程度，如果你启用了眼睑控制，则调整眼睑（通过旋转和/或移动它们），然后按**保存**。

在你保存了所有需要的位置后，该组件就知道了它所需要的关于如何对眼睑进行动画的所有信息。点击**眼睛**旁边的**加载，直视**，让你的角色直视前方。



完成设置后，你可以调整组件中的其他参数。



勾选 "**在编辑器中绘制视线**
"可以让你在编辑器窗口的播放模式中看到眼睛所看的确切位置。

主权重是控制REM移动眼睛、眼皮和头部多少的主要因素：如果设置为0，REM根本不移动任何东西。如果设置为1，更具体的权重决定了移动眼睛、眼皮和头部的程度。具体来说，REM对眼睛的总影响是

主体重量 * 眼睛重量

眼睛的重量滑块在眼睛折页中。

REM对眼皮的总影响是

主体重量*眼睛重量*眼睑重量

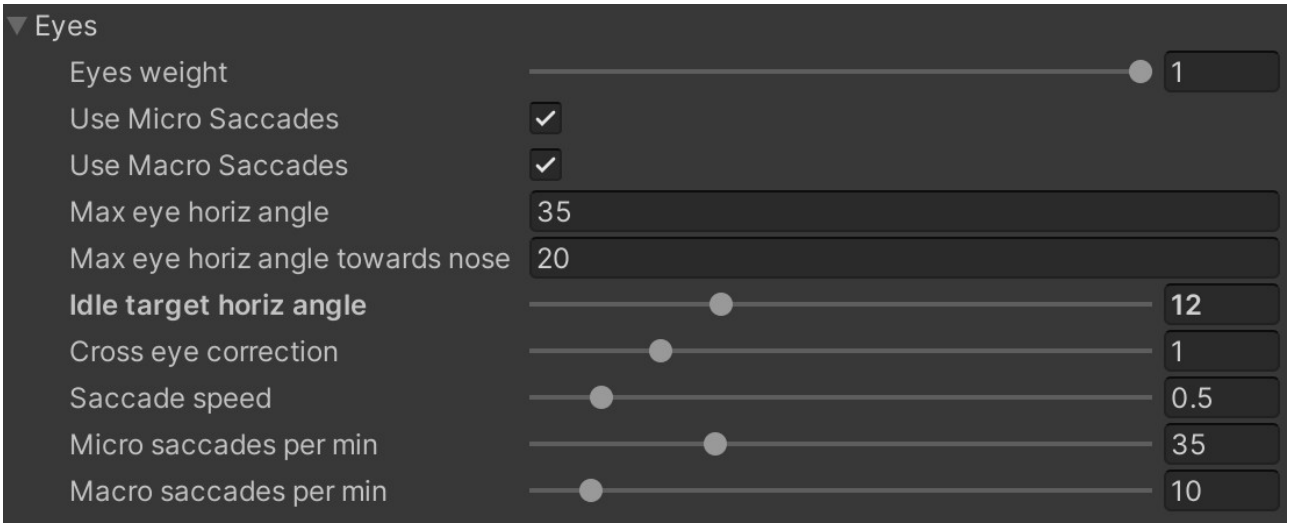
REM对头部的总影响是

主体重量*头部重量

例如，当你想让一个临时动画控制角色的眼睛时，你可以使用主重来淡化REM。

其他参数被归入三个折页。眼睛、眼睑和头部。

眼部折页



眼睛权重决定了REM对移动眼睛的控制程度，从0到1。0表示REM不移动眼睛，让它们保持原样（默认姿势或由动画控制等）。1意味着REM完全控制眼睛的方向。

使用微动复选框决定了眼睛是否不时地做那些人类眼睛通常会做的小动作，这使眼睛看起来更加逼真。

使用Macro Saccades复选框决定了眼睛是否不时地做较大的飞镖动作（类似于微观的saccades，但频率较低，角度较大）。当角色看着玩家的脸或调用**LookAtPoiDirectly**函数时，不使用Macro saccades。

默认情况下，对于最逼真的动画来说，同时使用微观和宏观的囊状动作。

眼睛的最大角度值决定了眼睛可以水平旋转多少，远离鼻子（所以对于左眼来说，它可以相对于头部向左看多远的角度）。

Max Eye Horiz Angle Towards Nose（眼睛朝向鼻子的最大角度）值决定了眼睛可以朝向鼻子水平旋转的程度（所以对于左眼来说，角度向右）。对于某些模型，如果*眼睛朝向鼻子*的*最大水平角度*小于*眼睛的最大水平角度*，看起来会更好。

闲置目标水平角度。在闲置模式下，下一个观察目标是在相对于向前看的水平角度的数量内选择。

交叉眼矫正值决定了在固定在靠近头部的物体上时，要在多大程度上防止眼睛出现交叉眼。默认值对于以下情况来说应该是很好的
大多数设置，但如果你发现你的角色对于近距离的物体仍然是横眉竖眼的，你可以增加它
。

眨眼速度决定了眼睛在眨眼过程中移动的速度。1是最现实的，但
0.5在大多数角色上看起来更好。

每分钟微缩决定了每分钟微缩的平均数量。

每分钟宏观扫视决定了每分钟宏观扫视的平均数量。

眼睑折叠

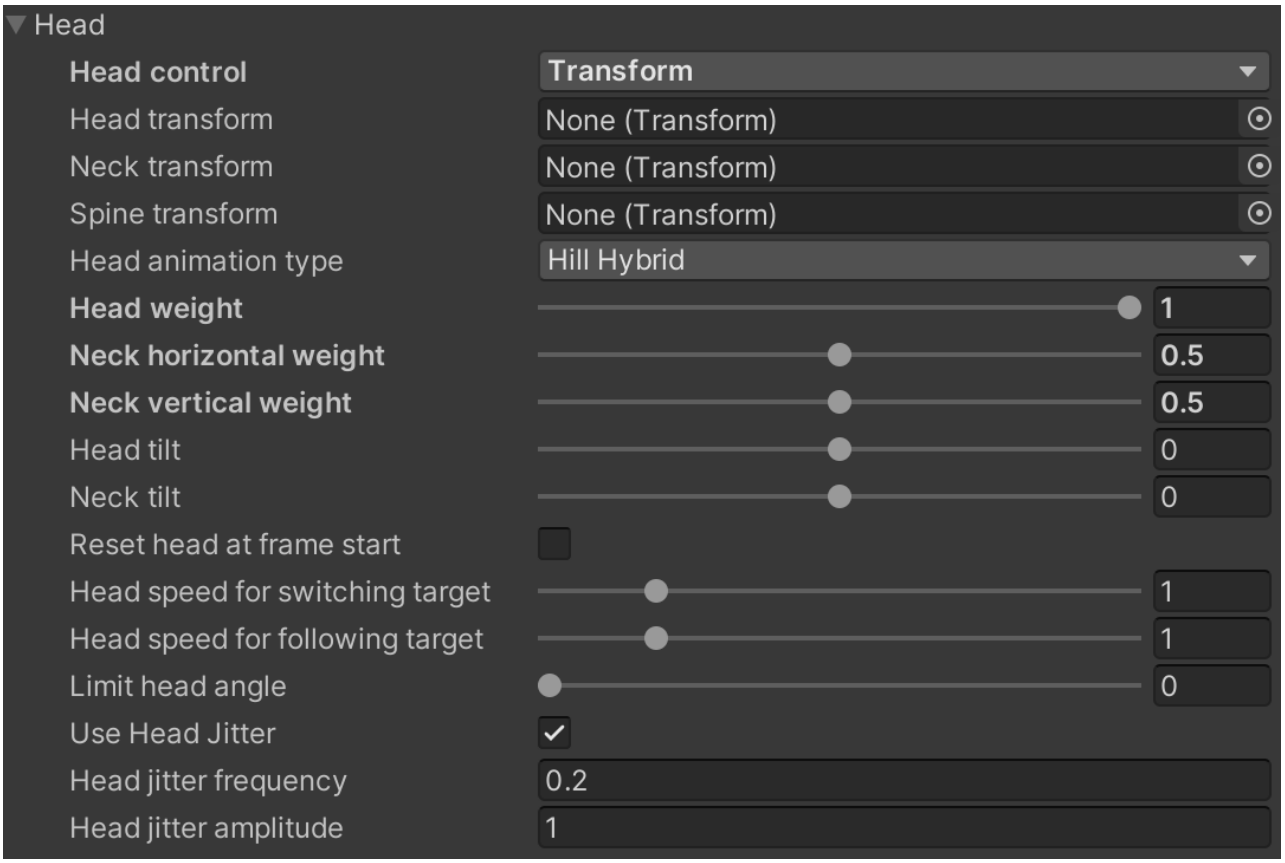
眼皮权重决定了REM对移动眼皮的控制程度，从0到1。0表示REM不移动眼皮，让它们保持原样（在默认姿势或由动画控制等）。1意味着REM完全控制眼皮的移动。

你可以通过设置**最小下一次眨眼时间**和**最大下一次眨眼时间**来控制眨眼的频率。在一次眨眼之后，到下一次眨眼的时间是最小眨眼时间和最大眨眼时间之间的一个随机秒数。闪烁**速度**允许您修改闪烁速度。把它保持在1的默认闪烁速度。

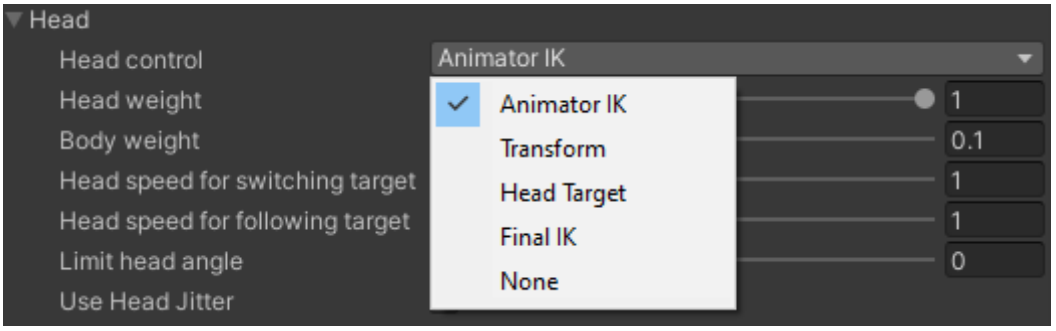
眼睑垂直跟随眼睛的复选框决定了眼睑是否跟随眼睛的垂直运动，当眼睛向上移动时，眼睑会向上移动，当眼睛向下移动时，眼睑会向下移动。这增加了真实感，所以默认情况下它是启用的。

眼睛变宽或变斜的数值决定了眼睛变宽（惊奇）或变斜的程度。值为0意味着没有扩大或眯眼。大于0的值意味着睁大，小于0的值意味着眯眼。睁大眼睛只有在眼皮被骨头控制时才有效。
眯眼对两种控制眼皮的方式都有效：骨骼或混合形状。

头部折叠式



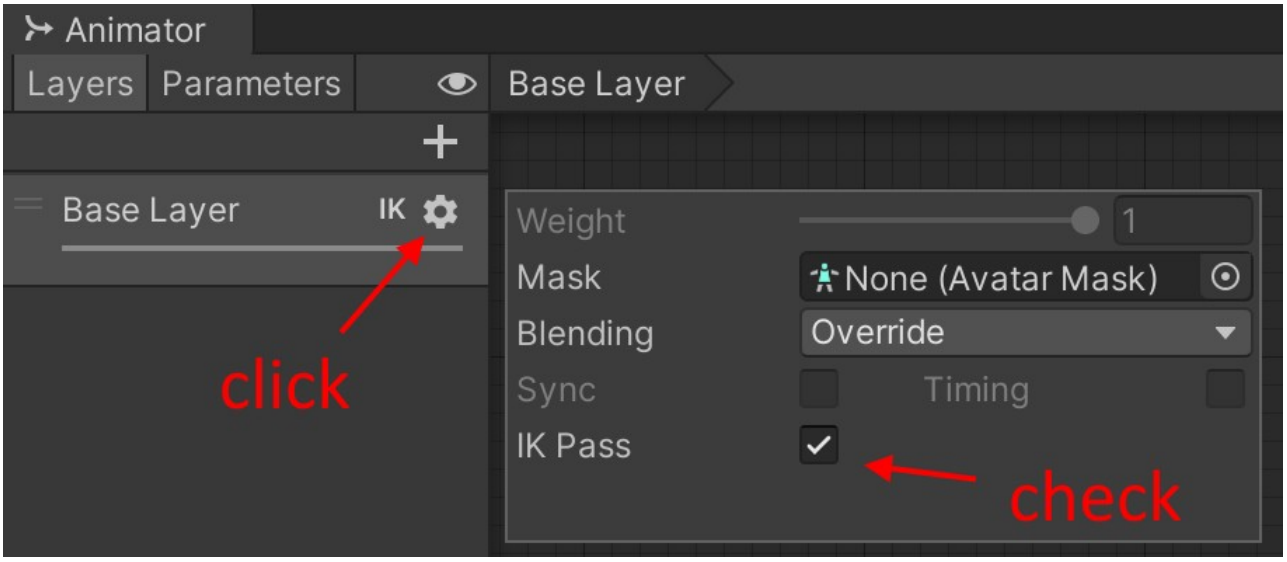
将**头部控制设置**为你希望REM控制头部运动的方式。



头部控制。AnimatorIK

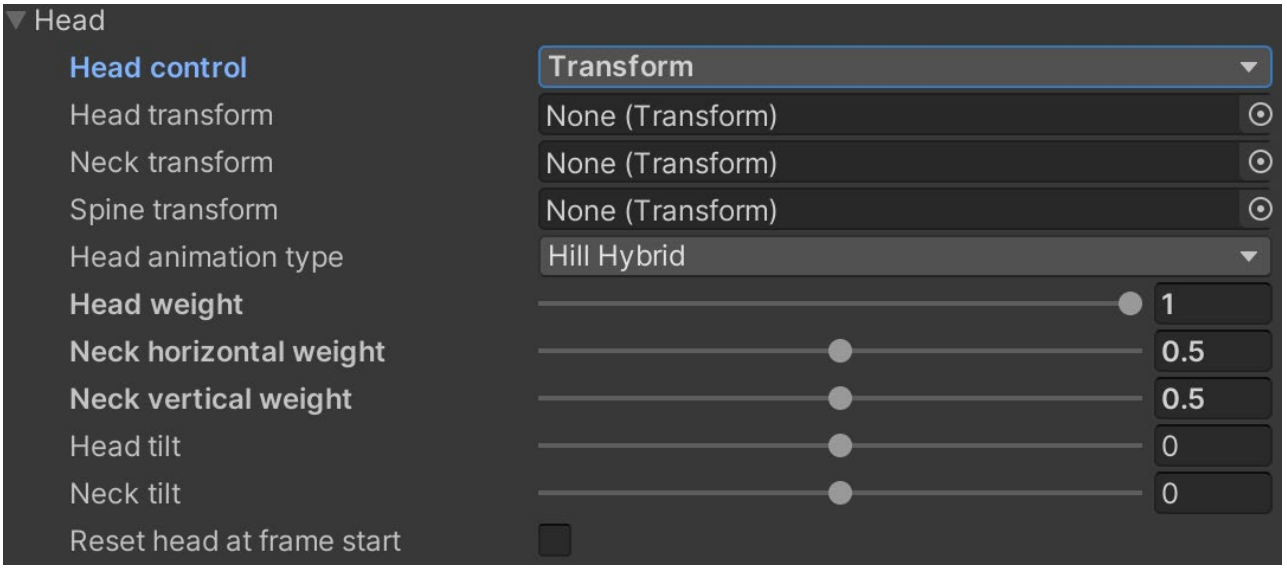
AnimatorIK使用Mecanim的OnAnimatorIK函数来确定头部的方向。为你的角色调整**头部重量**和**身体重量**。Mecanim的OnAnimatorIK似乎对某些角色效果最好，如果**头部重量**设置为低于1的值，因为否则头部在看向侧面时就会 "过冲"。

如果你使用这个选项，请确保你的角色的动画师已经启用了IK通道。



头部控制。转变

你也可以选择**Transform**来控制云台，它使用一个自定义的算法。



你可以分配一个头、颈和脊柱对象（或骨骼）。REM将确定头部的方向（如果有的话，根据颈部重量确定），并使用脊柱作为计算头部角度的参考
"前进方向"。如果你的角色是Mecanim人形，你可以不填这些槽；REM会自己找到合适的骨头。如果你不喜欢OnAnimatorIK对你的人形角色的头部定位，可以选择**Transform**（你可以把头部等的插槽留空）；它不会有过头的问题。

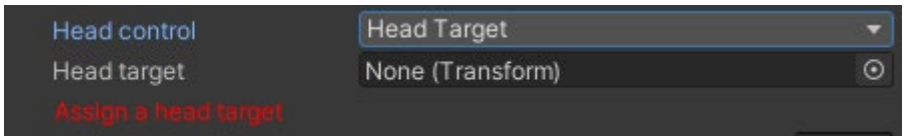
如果你的角色不是人形，你必须在插槽中为头部和脊柱分配正确的对象/骨骼。然后这个头部游戏对象将被旋转，以使角色的头部看起来方向正确。如果你设置了这个，请确保拥有EyeAndHeadAnimator组件的游戏对象的前进方向是沿着头部和眼睛的前进方向，所以它的前进方向是角色沿着 "看 "的地方。还要确保该对象有一个父游戏对象。

头部倾斜和**颈部倾斜**决定了头部围绕头部前轴 "滚动
"的角度。如果你想让你的角色倾斜他们的头来表达好奇心，你可以通过代码来制作这个/设置它。

如果你的角色的头部没有被动画师组件动画化，那么在使用头部控制变换时，头部方向可能会随着时间的推移而漂移；头部仍然看向正确的方向，但其向上的方向会徘徊。在这种情况下，**在帧开始时**检查**重置头部**来解决这个问题。

头部控制：头部目标

为**头部控制**选择**头部目标**，可以让REM持续更新场景中指定对象的位置。



在场景中创建一个空对象，并将其分配到*头部目标*槽中。然后你可以把这个对象分配给另一个可以调整头部方向的系统，例如你的角色上的动画装配设置。REM将移动这个目标，另一个系统将旋转头部来观察它。

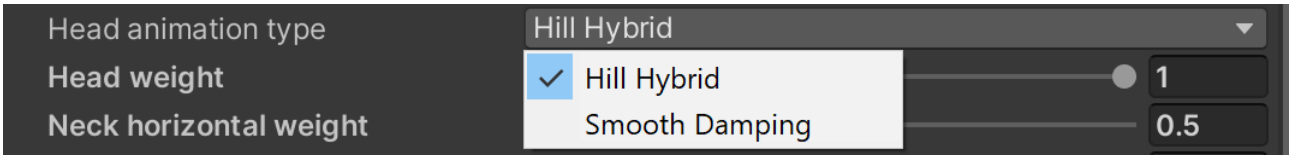
头部控制。最后的IK

如果你的项目中有**Final IK**，并想用它来旋转头部，请将FinalIK的LookAtIK组件添加到角色中（如果需要，还可以添加FullBodyBipedIK组件），并将头部控制设置为**Final IK**。此外，为了使REM能够使用FinalIK，需要在项目设置中定义USE_FINAL_IK这个符号。

> 打开*项目设置*，在*其他设置*下的*播放器*选项卡中，将 USE_FINAL_IK 加入到*脚本定义符号*。

如果你使用LookAtIK，你可能想把LookAtIK的头部重量滑块设置为1，因为REM已经考虑到了头部限制，滑块值为1时，角色会直接看向他们的目标。

头部动画类型



默认的头部动画类型是HillHybrid，这是一种实验性的方法，使头部旋转看起来更真实。如果你愿意，你可以改用旧的SmoothDamping方法。

切换目标的头部速度值允许你在从一个目标切换到另一个目标时增加或减少头部转动速度。

跟随目标的磁头速度值允许你增加或减少磁头跟踪其当前观察目标的速度。

头部重量滑块的值决定了脚本在观察目标时控制角色头部运动的程度。值为0意味着头部

只由动画控制。

头部角度限制值决定了限制头部的左/右旋转的程度。在零时。

角色将他或她的头转向感兴趣的点，在较大的数值下，头部保持正前方，只有眼睛看向目标点。

如果勾选了
"使用头部抖动"，那么在看一个点的时候，头部会保持轻微的移动，以使它看起来更自然，不像是一个僵硬的机器人。你可以用**头部抖动频率**和**头部抖动幅度**来控制头部抖动的参数。

预设

导出和**导入**按钮让你保存和加载预置。一旦你设置了组件，你就可以把设置导出到一个文件。然后，只要把LookTargetController和EyeAndHeadAnimator组件拖到他们身上并导入保存的文件，就可以快速设置类似的角色。为了使预设适用于一个角色，该角色需要在眼睛骨骼/混合形状等方面有一个兼容的结构。例如，如果你的设置使用了眼睑骨，那么眼睑骨的名称必须在你保存预设的角色和你想应用它的角色中是相同的。

在Presets文件夹中已经有这些类型的角色的预设。MakeHuman, Autodesk Character Generator, Mixamo, Character Creator 3, DAZ Studio和UMA。如果你的角色是这些类型的，只要在他身上拖一个LookTargetController和EyeAndHeadAnimator组件，从RealisticEyeAnimations/Presets中导入相应的文件就可以了。如果你不使用默认的眼睛形状，而是使用随机或自定义的，你可能需要调整设置。

> 如果你正在为一个Mixamo角色导入Mixamo预设，并得到一个错误信息，请确保角色中的**眼睑**网格对象被称为 "眼睑"--
> 有些Mixamo角色对它的命名不同，然后预设将无法工作。

在运行时添加脚本

你可以在运行时添加脚本（例如，如果你在运行时生成你的角色），并加载一个预设，让他们的眼睛正确地产生动画。下面是一个关于如何在运行时添加脚本的例子。将预设复制到你的项目中一个名为StreamingAssets的文件夹中（有这个名字的文件夹在最终构建时将会包含其所有文件而不作修改）。然后，在你的代码中生成一个角色后（比方说你把它保存在GameObject变量newCharacterGameObject中），使用这段代码（如果你使用Android或WebGL，见下文除外）。

```
EyeAndHeadAnimator eyeAndHeadAnimator =
newCharacterGameObject.AddComponent<EyeAndHeadAnimator>()
。

eyeAndHeadAnimator.ImportFromFile(Application.streamingAssetsPath +
"/mypreset.json")。
LookTargetController lookTargetController
=
```

newGO.AddComponent<LookTargetController>()
。
lookTargetController.Initialize()。

如果你使用的是**Android**或**WebGL**，访问StreamingAssets文件夹的内容的方式是不同的。请使用以下代码来代替上述代码。

```
#if UNITY_EDITOR || UNITY_STANDALONE
        EyeAndHeadAnimator eyeAndHeadAnimator =
        newCharacterGameObject.AddComponent<EyeAndHeadAnimator>()。
        eyeAndHeadAnimator.ImportFromFile(Application.streamingAssetsPath +
        "/mypreset.json")。
        LookTargetController lookTargetController =
        newCharacterGameObject.AddComponent<LookTargetController>()。
        lookTargetController.Initialize()。
#else
        StartCoroutine(LoadPreset("mypreset.json"))。
#endif
```

并在同一个类中，添加这段代码。

对于**安卓系统。**

```
using System.Collections;
using System.IO。
IEnumerator LoadPreset(string presetFilename)
{
string path = "jar:file://" + Application.dataPath + "！/assets/" +
presetFilename;
WWW loadPreset = new WWW(path);


产量返回loadPreset。


string newPath = Application.persistentDataPath + "/" + presetFilename;
File.WriteAllBytes(newPath, loadPreset.bytes);
EyeAndHeadAnimator eyeAndHeadAnimator =
newCharacterGameObject.AddComponent<EyeAndHeadAnimator>()。
eyeAndHeadAnimator.ImportFromFile（newPath）。
LookTargetController lookTargetController =
newCharacterGameObject.AddComponent<LookTargetController>();
lookTargetController.Initialize();
}
```

对于**WebGL。**

```
IEnumerator LoadPreset(GameObject character, string presetFilename)
{
string uri = Application.streamingAssetsPath + "/" + presetFilename;
```

```
UnityWebRequest www = UnityWebRequest.Get(uri);
       yield return www.SendWebRequest();
EyeAndHeadAnimator eyeAndHeadAnimator =
character.AddComponent<EyeAndHeadAnimator>()。
eyeAndHeadAnimator.ImportFromJson(www.downloadHandler.text)。
LookTargetController lookTargetController =
character.AddComponent<LookTargetController>()。
lookTargetController.Initialize()。
}
```

使用UMA（Unity Multipurpose Avatar）。

REM
2.1.0版有一个UMA预设，用于旧版本的UMA。如果你使用较新的UMA，请使用这个预设。

如果你使用UMA，你需要在运行时添加组件，因为这些字符是在运行时生成的。上一节描述了如何做到这一点。只要使用预设文件夹中现有的预设UMA.json即可。在UMA的情况下，你可能需要在角色生成后等待一帧，以便在应用脚本前添加所有的骨骼。你可以这样做：在生成角色后，使用这个代码。

```
StartCoroutine(AddREM(newCharacterGameObject))。
```

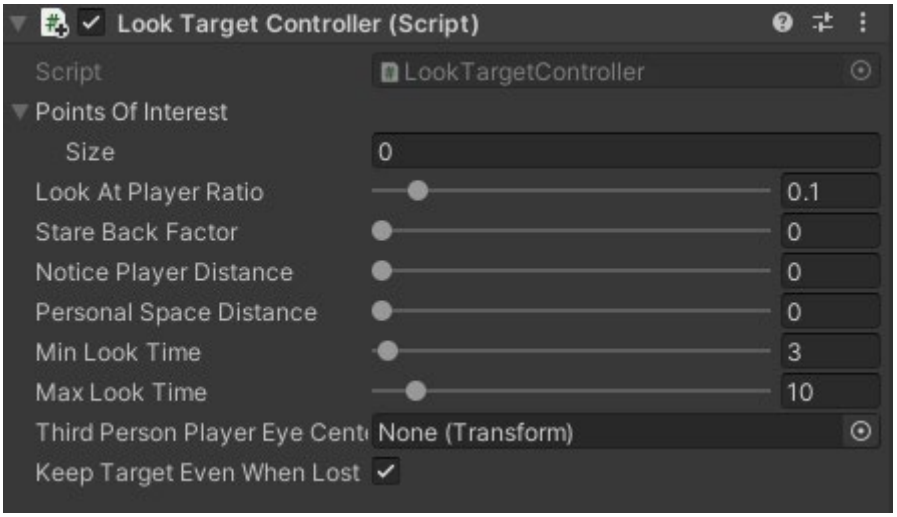
这里是函数AddREM（它在一帧之后添加组件，以确保所有的骨骼都被UMA添加）。

```
IEnumerator AddREM(GameObject newGO)
{
      产量返回null；产量返
      回null。

      EyeAndHeadAnimator eyeAndHeadAnimator
= newGO.AddComponent<EyeAndHeadAnimator>()。
      eyeAndHeadAnimator.ImportFromFile(Application.streamingAssetsPath +
"/UMA.json")。

      LookTargetController lookTargetController
= newGO.AddComponent<LookTargetController>()。
      lookTargetController.Initialize()。
}
```

视觉控制单元（LookTargetController



查看目标

LookTargetController组件让你控制你的角色在哪里看。默认情况下，角色会在随机的方向上闲看（接近于直视），如果玩家在视野中，有时也会看玩家。如果你在角色的环境中有特定的物体，你想让角色选择这些物体作为观察目标，可以把它们拖到**兴趣点**阵列中。角色会从这个列表中随机选择一个物体看上一段时间，然后从列表中选择另一个物体看。

如果玩家在角色的视野中，**玩家处的战利品比率**的滑块值决定了角色选择玩家作为下一个目标的频率：例如，滑块值为0.1意味着角色90%的时间选择随机方向或兴趣点列表中的一个物体，10%的时间选择玩家作为观察目标。要想让角色不看玩家，就把它设置为0。

滑块 "**回视因素**
"决定了如果角色看到玩家一直盯着他或她，他回视玩家的速度有多快（所以角色对被盯着的敏感程度）。

滑块**Notice Player**
Distance（玩家距离）决定了玩家要靠近到什么距离，角色才会开始看他。值为0意味着玩家走近没有影响。值为2意味着如果角色之前没有注意到玩家，而玩家出现在视野中并且距离超过2个单位，角色就会开始看玩家。

如果你将滑块 "**个人空间距离**
"设置为大于0的值，并且玩家靠近的距离超过这个距离，角色就会看向远方（以一种 "羞涩 "的方式避开玩家）。

角色在选择另一个目标之前看一个目标的时间是**最小看时间**和**最大看时间**之间的一个随机秒数。

玩家眼球中心。逼真的眼球运动默认使用主摄像头作为玩家（对于像看玩家比例这样

的设置），所以如果你正在开发一个3rd 人的游戏，其中主摄像头的位置在哪里？

摄像机不显示玩家的视角，你需要告诉EyeAndHeadAnimator玩家的眼睛在哪里。为此，你要给非玩家角色的EyeAndHeadAnimator组件（应该看着玩家的那个）的**玩家眼睛中心**槽分配一个始终位于玩家角色眼睛之间的变换。如果有必要，你可以创建一个空的游戏对象，将其置于玩家角色的眼睛之间，并将其赋值于玩家的头部骨骼。
还要确保变换的前进方向与玩家头部的前进方向相同。*注意：玩家眼睛中心是玩家角色眼睛的中心，而不是你正在编辑的拥有EyeAndHeadAnimator组件的角色的眼睛中心*

复选框**Keep Target Even When Lost**决定了角色是否继续尝试观察目标（通过尽可能地转动头部和眼睛），即使目标在角色的后面，所以离开了视线。如果该复选框是关闭的，角色将停止追踪目标（并返回到闲置的或另一个感兴趣的点上查看）。

LookTargetController中有四个事件你可以订阅。**OnStartLookingAtPlayer, OnStopLookingAtPlayer, OnPlayerEntersPersonalSpace,** and **OnLookAwayFromShyness**。例如，你可以让你的角色在每次看向玩家时都微笑。

VR: 虚拟现实头盔

默认情况下，脚本使用标记为MainCamera的摄像机来确定玩家的位置。如果你使用Unity的VR支持，脚本会使用它来找出玩家的左眼和右眼的位置。
当角色看向玩家时，他们会在左眼、右眼和嘴巴之间随机循环，作为看的目标（所谓的"社交三角"，人们在看别人的脸时使用）。你不需要改变任何东西来使脚本在VR场景中工作。

脚本API

你可以在LookTargetController组件上调用这些函数以获得更多的控制权（如果可能的话，从你的LateUpdate函数而不是Update函数中调用它们）。

LookTargetController.cs

空白 **眨眼( )**

让角色眨眼。 void

ClearLookTarget()

清除当前的观察目标，使角色直视前方，直到发出新的命令（因此，即使根据诸如观察玩家比例的设置，角色应该注意到玩家，但他也不会注意到玩家）。

bool **IsPlayerInView**()

返回角色是否能看到玩家（只检查视角，不检查范围或角色和玩家之间的视觉障碍）。

空白 **LookAtPlayer**(float duration=-1, float headLatency=0.075f)

观察玩家的时间为秒。如果要继续看，直到有新的命令，请将持续时间设置为-1。
HeadLatency决定了头部开始移动的时间比眼睛晚多少。

空白的**LookAroundIdly**()

开始在随机方向（接近直行）或在兴趣点（如果兴趣点列表有对象）寻找周围。这使用了组件的设置，比如看玩家的比例等，所以在某些条件下，玩家会被注意到或避开。

void **LookAtPoiDirectly**(Transform targetTransform, float duration=-1, float headLatency=0.075f)
)

观察一个特定的变形，持续数秒。如果变体移动，则用眼睛继续跟踪变体。在持续时间过后，角色会根据组件的设置（比如看着玩家的比例等）继续看。要继续看，直到有新的命令发出，请将持续时间设置为-1。
HeadLatency决定头部开始移动的时间比眼睛晚多少。

void **LookAtPoiDirectly**(Vector3 targetPoint, float duration=-1, float headLatency=0.075f)

观察一个特定的点，持续时间为秒。如果要继续看，直到有新的命令，请将持续时间设置为-1。 HeadLatency决定了头部开始移动的时间比眼睛晚多少。

如果你需要对动画进行更多的控制，而不是组件在当前状态下所能提供的，请让我知道，tore.knabe@gmail.com。

祝你的项目取得好成绩!

Tore Knabe