# 第6章 动画

## 基础概念

### 什么是帧

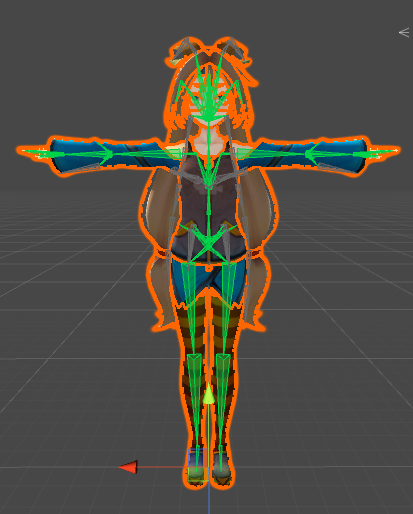
帧是一个量词，在古代一幅字画叫一帧，而在计算机中每次渲染完毕一次所有图像并显示出来，这一副静止的图像就是一帧。

连续的帧形成了动画，动画系统中我们通过制定一系列的帧来记录某个物体（GameObject）会发生的和位置、形变、渲染、事件等相关的改变，便形成了这个物体在游戏中的动画效果。

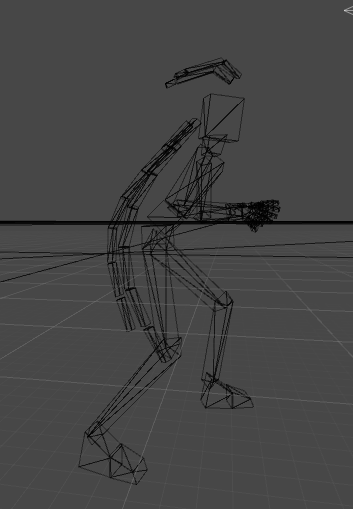
### 模型动画与非模型动画

当前有两种模型动画的方式：顶点动画和骨骼动画。骨骼动画顾名思义，就像人体移动是通过肌肉带动骨骼，骨骼传动肢体形成了动作。

骨骼动画比顶点动画要求更高的处理器性能，但同时它也具有更多的优点，骨骼动画可以更容易、更快捷地创建（有时候针对游戏优化，会将骨骼动画转换为顶点动画）。



顶点动画理解前，需要先了解游戏中任何看得到的模型都是基于**Mesh（网格）**实现的。而顶点动画就是对Mesh的顶点做位移旋转，因为是直接对mesh修改形成的动画，不需要通过骨骼做二次计算，因此顶点动画更加省性能。但由于一个模型顶点数量经常几百上千，它用来做复杂的动画操作难度大。



非模型动画经常是对游戏内GameObject物体的动画，比如UI界面上的物体实现大小、偏移和制作摄像机摄像轨道等效果。非模型动画可以通过Unity的曲线编辑工具配置。

### 动画混合核心——插值计算与权重

在游戏中我们知道角色向前移动和向左移动是不同的动画，而如果操作摇杆向左上移动呢？或者我们摇杆向左偏一点点向上呢？这种输入是无穷的，在FC游戏时代我们可以规定游戏只有4个方向，那么美术可以只用制作4个方向上的动画就好了。而现在3D游戏时代，移动的方向已经偏向自由。因此我们不可能让美术人员再去制作各个方向上的动画，而是让计算机“智能”地为我们计算出动画。

比如我们现在已经有了4个方向上的动画，现在计算机要为我们计算出角色播放“↖”这个动画。这个计算便依托于**插值。**

插值是离散函数逼近的重要方法，利用它可通过函数在有限个点处的取值状况，估算出函数在其他点处的近似值**。**例如“↑”和“←”这两个动画分别会修改A骨骼的位置为（0，1，0）和（1，0，0）坐标，当做“↖”融合动画的时候，计算机会修改A骨骼的位置在（0~1，0~1，0）坐标中选取点。而选取值更加偏向“↑”还是更加偏向“←”，则取决于**权重**了。

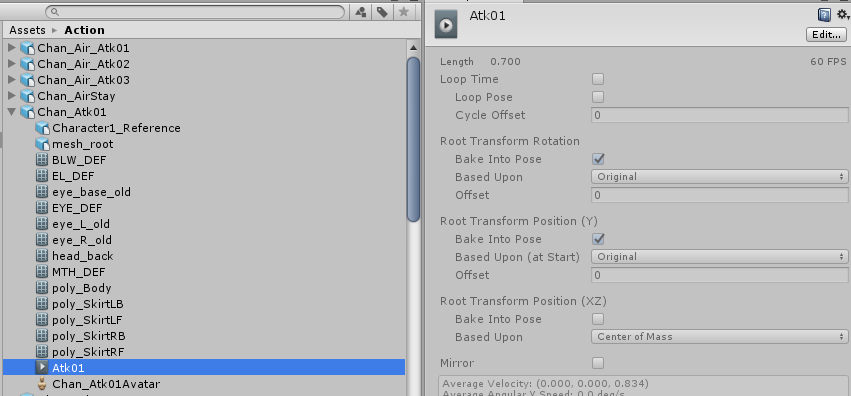
权重是一个相对的概念，针对某一指标而言。某一指标的权重是指该指标在整体评价中的相对重要程度。权重是要从若干评价指标中分出轻重来，一组评价指标体系相对应的权重组成了权重体系。

## Mecanim动画系统

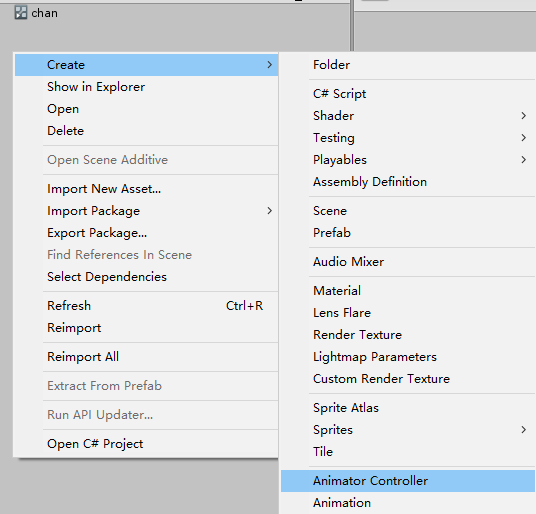
### 动画系统工作流

Unity引擎中，任何功能系统最后都可以简单拆分为三个模块，资源模块、控制&编辑模块、实体模块。

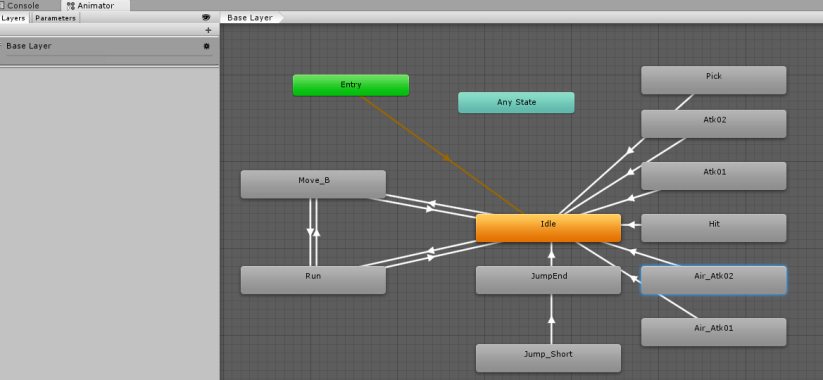
动画剪辑（Animation Clips）是资源模块，它包含了对象如何随着时间改变它们的位置、旋转或其他属性的信息。每一个动画剪辑都是单一线性记录，这些动画剪辑的来源可以是外部软件制作也可以Unity内部创建利用曲线编辑器编辑。



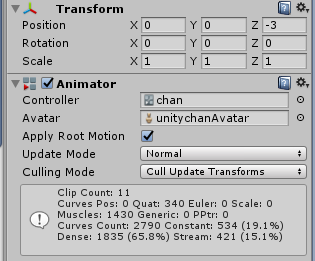
动画控制器（Animator Controller）是控制&编辑模块，我们需要对动画剪辑进行管理，因此需要记录下动画之间的关系，这些信息便存放于动画控制器中。



通过打开动画控制窗口（Window->Animator），控制对象何时播放Clips，何时改变，何时混合在一起。在Unity中，动画控制窗口能把动画剪辑梳理成结构化的流程图。可以很方便的查看当前的状态。



动画组件（Animator）是游戏实际运行时动画功能的实体模块，可以对当前对象设置读取动画控制器，动画刷新时间和动画剔除等功能。



总而言之，一个物体的所有动画剪辑都存放于动画控制器中，一个游戏对象通过动画组件读取到动画控制器得到当前应该播放的动画。

### 动画Clips

动画剪辑是Unity动画系统的核心元素之一。Unity支持从外部源导入动画，并提供使用动画窗口在编辑器中从头创建动画剪辑的能力。

#### 来自外部来源的动画

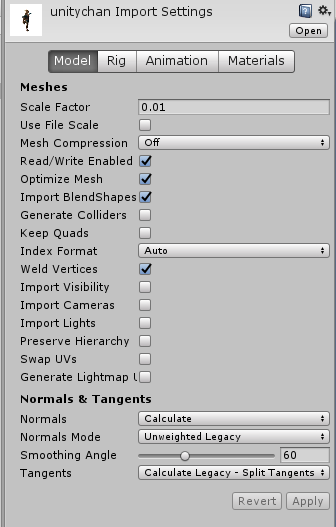
从外部源导入的动画剪辑可以包括：

* 动作捕捉捕获的人形动画
* 由外部3D应用程序中创建的动画（如3DS Max 或 Maya）
* 来自第三方库的动画（例如，来自Unity的商店）
* 对长的动画剪辑的切割

来自外部源的动画是以FBX文件类型导入Unity中的。这些文件，无论他们是通过何种软件导出，都可以在Unity中得到对象线性记录形式的动画数据。

##### 动画的导入文件属性

###### Model tab



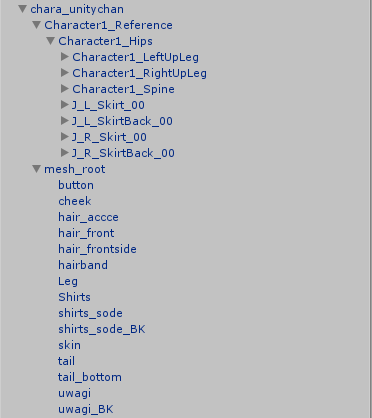
|  |  |  |
| --- | --- | --- |
| **参数** | | **功能** |
| **Meshes** | | |
| **Scale Factor（比例因子）** | | 因为文件在不同的编辑软件中，“显示单位”和“系统单位”是不一样的。为了让文件在Unity中的大小和编辑软件中大小是一样的，就需要设置比例因子。比如通常情况下，3DMax导出的FBX文件Unity得设置0.01大小的比例因子。如果修改3DMax中的系统单位为1cm，那么设置为1的比例因子就可以了。以下为文件格式通常格式的比例： .fbx .max .jas .c4d = **0.01** .mb .ma .lxo .dxf .blend .dae = **1**  .3ds = **0.1** |
| **Use File Scale（使用文件默认缩放）** | | 勾选复选框来使用默认的模型缩放，可以和**Scale Factor结合使用，如果勾选后，调整Scale Factor为1，则完全使用编辑软件中的文件缩放。** |
| **Mesh Compression（网格压缩）** | | 选择不同的值，Low、medium和High分别为压缩的效果。压缩效果越高，则网格显示效果越差，但网格的计算会越快。优化游戏性能时可以采用网格压缩的方式，兼顾效果和性能选择参数。 |
| **Read/Write Enabled(网格读写开关)** | | 如果启动，当网格在内存中时，脚本可以通关访问读取/修改该网格，如果禁用会节省内存。但如果禁用该选项而游戏中又出现了访问该网格数据的操作时，会导致系统崩溃情况，因此需要明确了解是否有读写该模型网格的地方，做优化时一定要慎重。 |
| **Optimize Mesh（网格优化）** | | 勾选该选项会使几何体更快的绘制，因为Unity将会在计算模型所有三角形并形成有顺序批次的三角形列表。如果不勾选将会节省部分内存，但会影响性能。 |
| **Import Blendshapes（导入BlendShapes）** | | BlendShapes是一种Maya软件混合动画的一种方式。比如制作表情动画，如果一根一根的骨骼调整去制作一个表情动作相对来说很麻烦，而如果制作一个通常的表情，和一个笑容的表情，利用BlendShapes实现中间动画的过渡，相对来说简单许多。如果导入了**Blendshapes可以让动画变换地更加平滑。** |
| **Generate Colliders（生成碰撞）** | | 如果启用，创建模型的时候会自己添加碰撞组件，该碰撞组件将会直接创建在骨骼节点上，因为是网格碰撞将会非常的吃系统性能。 |
| **Keep Quads（继续周围细分）** | | Unity对网格的绘制都是通过三角形绘制的，如果模型有四边形存在，Unity将会分割为两个三角形计算。而如果要继续进行曲面细分（三角形并不是最小的分割单元），就需要用到Tessellation (曲面细分DirectX 11的特性) 。曲面细分是一种将多边形分解成更加细小的碎片以提升几何逼真度的方法。如果勾选该选项，便可以支持（Tessellation）曲面细分的渲染。 |
| **Index Format（索引格式）** | | 顶点索引，就是将我们的所有顶点进行标号索引，之后我们若再使用它的时候，调用它的索引就可以了，无需重新创建一个新的顶点。  例如：一个立方体需要应该6个面，每个正方形面两个三角形组成，一个正方体最少是需要12个三角形组成，每个三角形需要3个顶点来确定，那么就意味着我们需要给系统36个顶点信息来绘制这个正方体。而实际上去掉重复的顶点我们只需要8个顶点就可以确定一个立方体了。  该选项为定义网格索引缓冲区的大小。  注意：对于带宽和内存存储大小的原因，通常希望保留16位索引作为默认值，必要时只使用32位。 |
| **Weld Vertices（熔接顶点）** | | 勾选此复选框来组合在空间中共享相同位置的顶点。通过减少网格的总数来优化网格上的顶点计数。默认情况下勾选此复选框。 |
| **Import Visibility（导入可见）** | | 勾选后可以从FBX文件中读取可见性属性，将会在动画控制器中可以看到MeshRender的激活和关闭（猜测制作角色突然消失，之后出现这一类技能动作会用到）。 |
| **Import Cameras（导入摄像机）** | | 导入FBX文件中的摄像机。 |
| **Import Lights（导入灯光）** | | 导入FBX文件中的灯光。 |
| **Swap UVs（切换UV）** | | 场景烘焙有关，当有光照贴图的物体的UV通道不正确时启用此选项。这将交换你的主UV通道与次UV通道。 |
| **Generate Lightmap UVs（生成光照UVs）** | | 启用此选项会为光照贴图创建第二个UV通道。 |
| **Normals & Tangents** | | |
| **Normals（法线）** | | 决定是否使用法线和如何计算法线。此选项对优化游戏的大小是有用的。 |
|  | **Import（文件导入）** | 默认选项，从文件导入法线。 |
|  | **Calculate（法线计算）** | 根据平滑角度（Smoothing angle）计算法线。如果选中，启用平滑角度（Smoothing angle），下面会介绍到平滑角度。 |
|  | **None** | 禁用法线。如果网格既没有法线贴图，也不受实时光照影响，则可使用此选项。 |
| **Normals Mode（法线模式）** | | 决定法线计算的模式。 |
|  | **Unweighted Legacy（传统 无加权）** | 在2017.1版本之前的Unweighted计算方式。和最新的版本计算的有差异。 |
|  | **Unweighted（无加权）** | 无加权计算。 |
|  | **Area Weighted（区域加权）** | 法线按面的面积加权。 |
|  | **Angle Weighted（顶点角加权）** | 法线由每个面上的顶点角加权。 |
|  | **Area and Angle Weighted（区域和顶点角加权）** | 默认选项。法线由每个面上的面面积和顶点角加权。 |
| **Tangents（切线）** | | 决定如何定义和计算切线和副法线。 |
|  | **Import** | 根据文件数据导入。 |
|  | **Calculate** | 默认选项，通关计算得到切线。 |
|  | **None** | 关闭切线和副法线。 |
| **Smoothing Angle（平滑角度）** | | 平滑角度：平滑角度的大小设置，会影响边作为硬边处理的锋利程度。它还被用来切分法线贴图切线。例如：一个矩形有四个角，如果我们将最上面两个角平滑，那么矩形会变为倒U形，上面两个角的锋利程度被影响了。 |
| **Split Tangents（分离切线）** | | 如果法线贴图灯光被网格上的接缝破坏，则启用该复选框。 |

###### Rig tab

|  |  |  |
| --- | --- | --- |
| **参数** | | **功能** |
| **Animation Type（动画类型）** | | 设置当前动画的类型 |
|  | **None** | 无动画设置 |
|  | **Legacy** | 传统动画设置 |
|  | **Generic** | 通用的Mecanim动画系统，选用该选项，模型将不会创建骨骼映射，相关人形动画的设置将会关闭 |
|  | **Humanoid** | 类人形Mecanim动画系统，将会创建骨骼映射，同时对于角色相关的动画设置都会开启。 |
| **Avatar Definition（骨骼化身定义）** | | 骨骼化身选用何种方式生成。 |
|  | **Create from this model** | 基于自身模型创建。 |
|  | **Copy from other Avatar** | 基于其他的模型创建的骨骼化身创建。 |
| **Configure（化身确认）** | | 将会跳转到查看角色骨骼映射界面。 |
| **Optimize Game Object（优化游戏物体）** | | 当该选项开启，将会删除角色自身骨骼创建的游戏对象，会将该对象存储在骨骼映射和动画组件中。角色的skinnedmeshrenderers组件将直接使用Mecanim系统内部骨架。  该选项将会提高动画角色的性能，在该模式下蒙皮网格矩阵是多线程计算的。 |

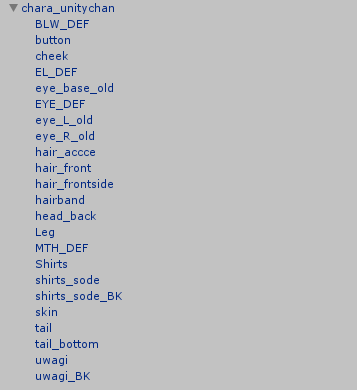
**Optimize Game Objecet选项的效果的实例说明：**

**若我们不勾选该优化选项，可以看到角色模型将会把子物体各个骨骼都创建成为一个游戏物体，如下图：**

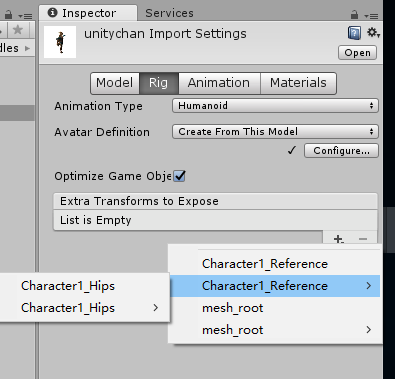


**这样做我们可以看到模型各个骨骼的层级关系，但是对于我们游戏本身来说是没有必要的，因为我们只关系我们需要用到的骨骼物体，因此每个骨骼都创建了一个游戏物体，实际上是牺牲了游戏性能。**

**如果勾选该选项后，Unity将会只创建蒙皮相关的游戏物体，如下图：**



**因为我们有时需要在角色骨骼上挂载物体，因此我们需要部分特殊的骨骼创建成挂点物体，比如武器挂点。我们可以在Extra Transforms to Expose选项框下去添加我们要用到的额外骨骼。**



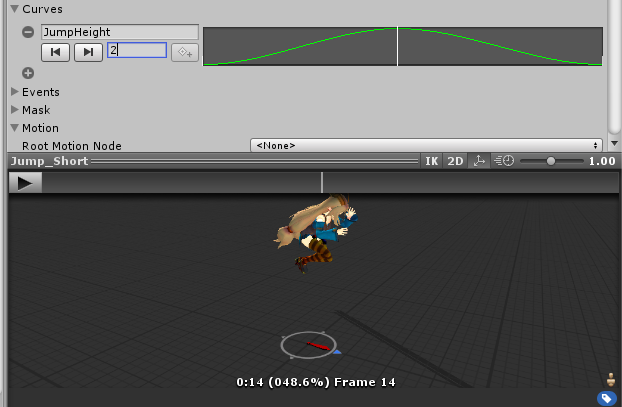
**总的来说该选项节约了无用的骨骼节点计算，优化游戏性能，建议勾选。同时比如当我们需要遍历模型用来修改模型与其子物体层级实现关闭模型和模型特效的功能，如果子节点众多遍历将会造成游戏卡顿。**

###### Animations tab

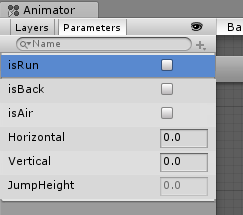
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **参数** | | | **功能** | |
| **Animations** | | |  | |
| **Bake Animations（烘焙动画）** | | | Unity将转换为导入运动学。此选项仅适用于Maya、3DSMAX和Cinema4D文件。当动画选项为Humanoid且使用到了IK动画特性时才会用到 。 | |
| **Anim. Compression（动画压缩）** | | | 设置网格动画的压缩类型 | |
| **Off（关闭）** | | | 关闭动画压缩。意味着Unity将不会对关键帧的数量进行减少，动画精度将会提高。但是会造成文件大小的增加和运行内存的更多消耗。 | |
| **Keyframe Reduction（关键帧缩减）** | | | 减少导入的关键帧，选中后，动画压缩错误的选项将会显示。 | |
| **Optimal（最优设置）** | | | 最优化压缩，压缩效率最高，动画效果失真度也相应提高。 | |
|  | **Rotation Error** | | 这个三个参数默认都是 0.5。该值值越小动画的旋转、位移、缩放相关的精度就越高，失真度越小，而压缩效果越差。 | |
|  | **Position Error** | |
|  | **Scale Error** | |
| **Clip特定属性** | | | | |
| **Name（名字）** | | | 当前动画剪辑的名字 | |
| **Source Take（资源获取）** | | | 源动画文件资源轨道，比如美术人员制作了一套动画放在了一个Clip里面，我们可以通过设置截取其中一段动画的开始帧和结束帧的位置，来得到想要的那个动画。 | |
| **Start（开始帧）** | | | 当前动画剪辑的开始帧。 | |
| **End（结束帧）** | | | 当前动画剪辑的结束帧。 | |
| **Loop Time（开启动作循环）** | | | 设置动画剪辑是否循环播放。 | |
|  | **Loop Pose（重复姿势）** | | 使动画播放无缝衔接。 | |
|  | **Cycle Offset（周期偏移）** | | 影响动画开始时的位置，例如设置为0.5，将会在动画长度最中间的关键帧开始播放动作。 | |
| **Root Transform Rotation**  **（根变换旋转设置）** | | **Root Transform Position (Y)**  **（根变换Y轴位置设置）** | | **Root Transform Position (XZ)**  **（根变换XZ平面位置设置）** |
|  | **Bake into Pose（合并姿势）** | | 勾选时，利用外部控制器带动角色位移和旋转，不勾选时，使用动画本身的旋转和位移带动角色移动。  比如一个角色播放冲刺攻击的动作，如果我们勾选，则需要外部去控制冲刺的距离。如果不勾选，则是使用动画本身的冲刺距离，当动画播放完毕后回复到原来位置。  当勾选Animator组件中的“Apply Root Motion”选项后，模型动画播放结束后，对坐标的影响将会保留在模型上。 | |
|  | **Based Upon** | | 选用根变换的方式。 | |
|  | - **Original（初始值）** | | 保持在源文件中写入的旋转/Y/XZ。 | |
|  | - **Body Orientation（身体方向）** | | 播放动作时让角色旋转始终面向保持身体前方。 | |
|  | **- Center of Mass（重心）** | | 角色的位置与质量中心的位置一致。 | |
|  | **- Feet（脚）** | | 角色的位置与脚步位置保持一致。 | |
|  | **Offset（偏移）** | | 对角色（根节点）旋转的偏移量。 | |
| **Mirror（镜像）** | | | 是角色动画左右反转播放（只能在类人形骨骼上使用）。 | |
| **Additive Reference Pose（附加参考姿势）** | | | 该选项勾选后可以创建一个参考指针在**Source Take上，当我们拖动他或者直接设置下方的pose Frame可以看到该帧的参考姿势。**  **这个的作用是用来去提取单帧动作使用的，在没有这个选项前，我们都是通过拖动start开始帧来观看单帧效果（对已经剪辑好的动画是不利的）。而单帧的动作可以用来很好的制作混合层的动作融合，例如绝地求生中角色站立不动的左右瞄准（左瞄和右瞄通过单帧的动画实现的，这样枪才会稳）。** | |
| **Mask（动画遮罩）** | | | 在Humanoid选项下可以设置该动画关闭对某些肢体的影响，只适用于类人形动画。同时可以设置IK逆向运动学功能的开启和关闭（左键点击肢体部分变为红色则为关闭）。  如果要直接屏蔽对部分骨骼影响，可以在Transform选项下选择开关骨骼节点。这个选项可以适用于非人形的骨骼动画。 | |
| **Curves（参数曲线）** | | | 可以给该动画添加一个控制参数的曲线，如果该参数在Animator动画控制器组件中有定义，那么这个值将会被动画控制修改。  例如：我们一个跳跃动作的FBX文件的该选项下添加设置一个名为“JumpHeight”的曲线，然后设置一段关于当前角色跳跃高度值的关键帧。那么当我们同样在Animator中创建一个参数同样名为“JumpHeight”，那么当播放到跳跃动画的时候，Animator中的这个参数将会被动画剪辑中的曲线值控制（值得一说的是，该参数将只能被动画剪辑中的曲线影响）。 | |
| **Events（动画事件）** | | | 可以在该动画剪辑的某个关键帧下创建一个触发事件，该事件可以传入值类型变量以及资源对象。  例如：我们将攻击动作绑定音效，想在挥舞武器某个时间段播放破空声音就可以通过动画事件添加。 | |
| **Motion（根运动节点）** | | | 允许用户自己定义该动画影响的**根运动节点**。根运动节点是很重要的，比如我们选取角色模型的**根节点**为根运动节点，那么该动画的位移将会带动根运动节点移动,也就是角色移动。但有时我们会遇到设置角色模型的其他节点为根运动节点的情况，该选项便支持该设置。 | |
| **Import Messages（动画导入的报告信息）** | | | 生成该动画文件是如何被导入的信息，其中有一个“Retargeting Quality Report” 选项，点击后可以查看动画重定向质量的报告信息。 | |

**Curves和Events的实例说明：**

**我们将一个跳跃动画的Curves选项下添加一个名为“JumpHeight”的曲线，然后在角色跳跃中点击输入框右边的“”图标，便可添加一个可以设置的关键帧。**



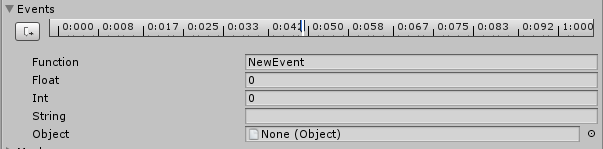
之后当我们将该动画拖入动画控制器中，然后在Parameters栏添加**“JumpHeight”的float值类型的参数，当我们运行时，可以看到该参数的值部分被置灰，即是该参数已被动画文件中的Curves控制。**



该值可以在脚本中读取，通过调用Animator组件的API接口GetFloat来访问浮点类型的参数。

例如：Animator.GetFloat(“JumpHeight”)获得当前的参数值。

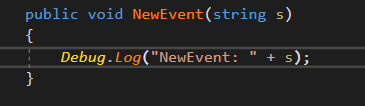
事件的添加通过选取该动画剪辑片段的关键帧，点击“”按钮可以添加一个触发事件。定义Function的名字为控制脚本中（绑定在角色身上的，和Animator组件同层级的脚本）的函数名字，当动画播放到该事件的关键帧时，动画将会调用函数并实现参数传递。



脚本中创建的函数必须为Public公有函数才能被动画控制器访问到，其中形参可以不设置。

如果要获得设置事件的参数比如Float、Int、String、Object，则函数需要创建相应的形参。需要注意的是，函数形参最多只能有一个。

例如：下图中NewEvent()函数形参为string则可以接受string类型的参数。改为Object则可以接受Object类型的参数。

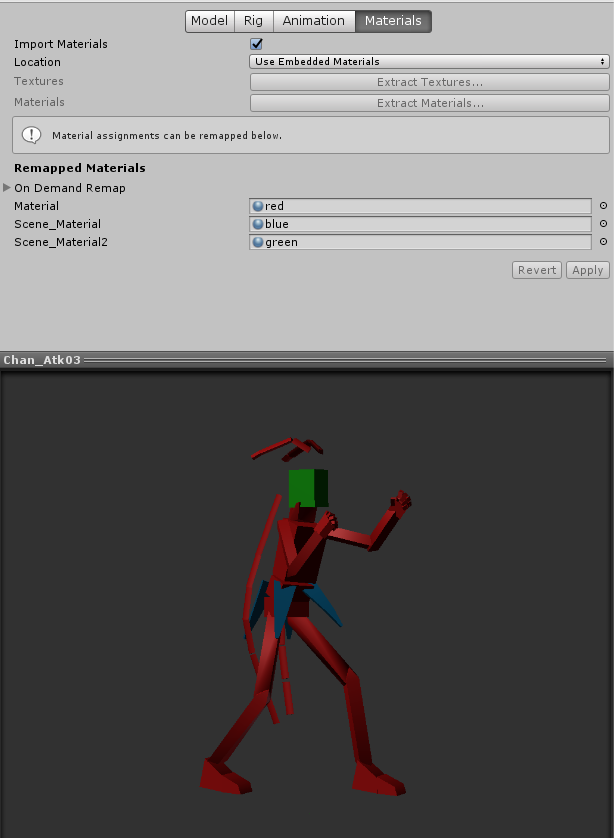


###### Materials tab

勾选Import Materials后可以使用文件自带的材质属性，以下为勾选后出现的功能选项。

|  |  |
| --- | --- |
| **参数** | **功能** |
| **Use Embedded Materials（使用嵌入材质）** | 该参数为2017.2后默认选项，选择该选项后，将外部创建的材质设置到模型文件中。 |
| **Use External Materials (Legacy)**  **（使用外部材质）** | 该参数为2017.1之前版本的默认选项，选择此选项后，模型将会每次导入自动创建材质球，该材质球将会存放到该模型同级的Materials文件夹下。 |

以下为选用嵌入材质的效果图，red、blue、green为我们自定义的材质球：

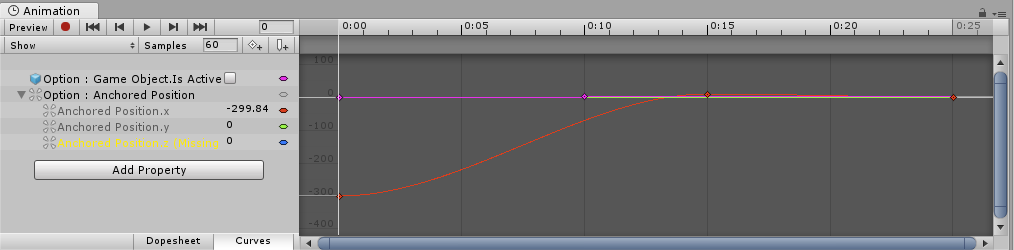


|  |  |  |
| --- | --- | --- |
| **参数** | | **功能** |
| Textures | | 如果模型自带纹理，可以提取纹理得到材质中的纹理贴图。如果是灰色的代表没有纹理可以提取。 |
| Materials | | 如果模型自带材质，可以提取导入模型的材质资源。如果是灰色的代表没有材质可以提取。 |
| Remapped Materials（重映射材质） | | |
| On Demand Remap（需求映射） | | 如果设置了使用“Use External Materials（使用外部材质）”， 那么将会出现以下设置。 |
|  | Naming | 决定材质球如何命名。 |
|  | By Base Texture Name | 通过材质球的漫反射纹理的贴图名字命名。 |
|  | From Model’s Material | 根据导入模型的材质名字命名。 |
|  | Model Name + Model’s Material | 模型名字和材质名字相结合命名。 |
|  | Search（搜寻材质） | 通过“Nameing”设置的选项规则去寻找在满足条件的材质资源。 |
|  | Local（本地） | 在和模型同目录的文件夹下（包括子文件夹）搜索命名规则的材质球。 |
|  | Recursive-Up（递归） | 递归所有材质资源的父文件夹和其子文件夹去寻找命名规则匹配的材质球。 |
|  | Everywhere（任何位置） | 寻找工程中所有位置的材质资源匹配。 |
|  | Search and Remap | 点击其按钮，Unity将会根据设置搜寻重新映射材质球。 |
| List of imported materials | | 该列表显示发现的所有导入材质，可以在这里手动重新映射存在的材质资料。 |

#### 在Unity内创建和编辑的动画

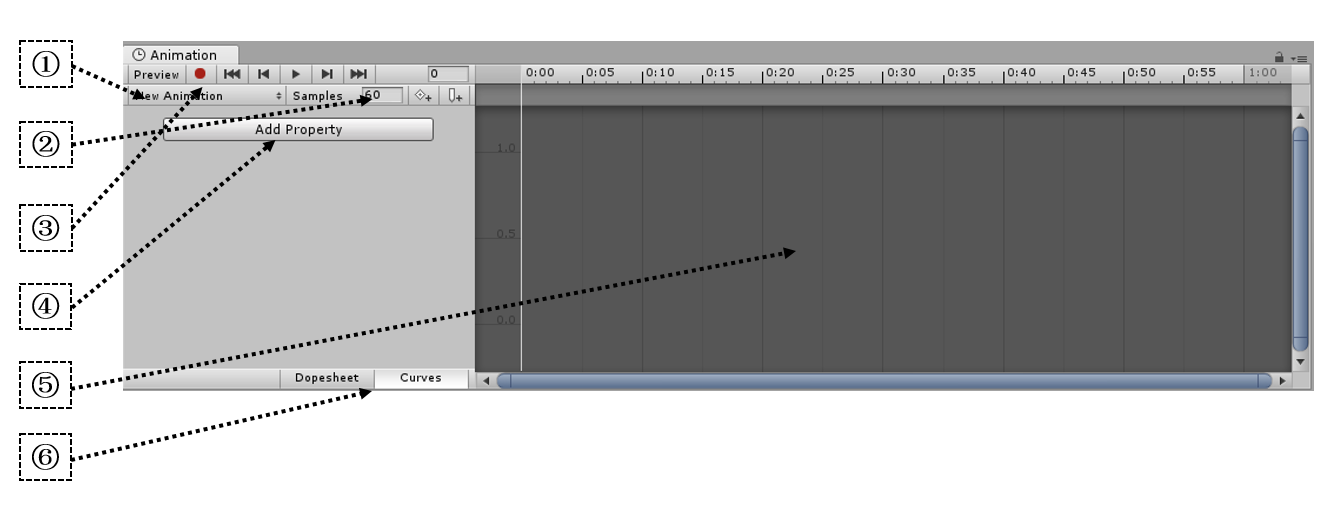
Unity的动画窗口可以创建和编辑动画剪辑：

* 对象的位置、旋转和缩放
* 部件属性，如材料的颜色、光的强度、声音的音量
* 控制对象身上脚本参数，包括浮点、整数、枚举、向量和布尔变量
* 设置动画事件调用脚本中的函数

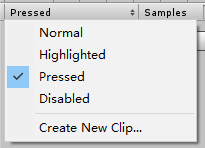


##### 动画窗口

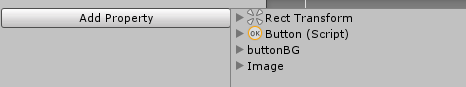
动画窗口在Unity4.0后添加到Unity中。它提供了一个简单的方法来创建动画短片和动画游戏对象。



* 1号位置：点开选择你要编辑的动画Clip，它是读取Animtor动画控制组件里面信息查找的。



* 2号位置：设置动画的帧率，一般游戏30帧动画已经非常流畅了。
* 3号位置：录制模式，点击后，Unity将会把你对物体的修改操作直接保存在时间帧窗口中。
* 4号位置：添加属性按钮，点击后会查找这个物体上的所有组件，如果我们挂在了一个脚本在这个物体上，这里也会把脚本控制参数暴露出来。

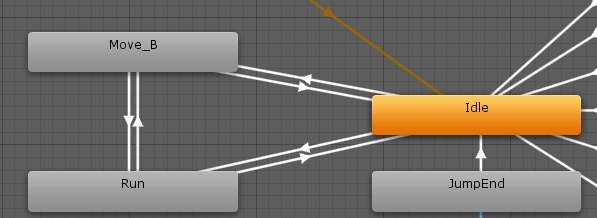


* 5号位置：控制每一帧状态的窗口，这是可视化编辑的，我们主要操作都在这里进行。
* 6号位置：控制当前动画控制窗的编辑模式为曲线编辑还是关键帧编辑。

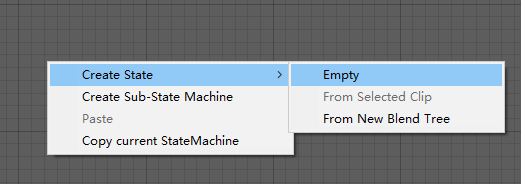
## 动画控制器（Animator Controllers）

### 动画状态机（State Machine）

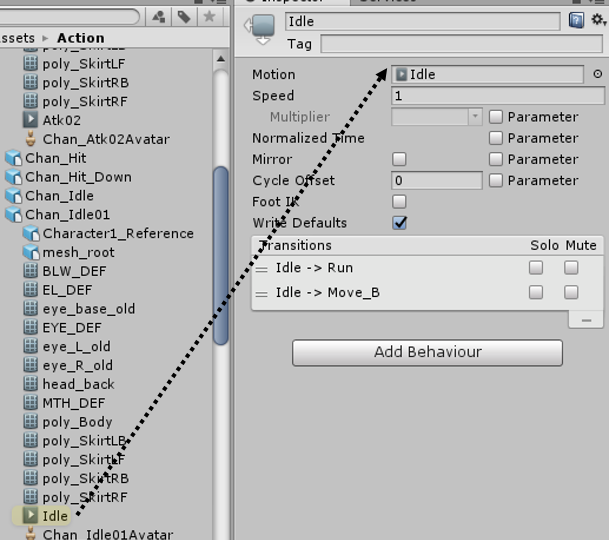
对于一个角色来说，有几个不同的动画对应于它在游戏中可以执行的不同动作。而这个动画如何触发（是否有触发的限制条件），触发后退出到哪个状态，是否需要提高动画的播放速度等问题都是由动画状态机做处理的。



我们可以右键动画控制器空白处创建状态机。



之后将动画剪辑拖动到状态机的Motion参数处，完成状态机与动画剪辑的绑定。



#### 参数定义

|  |  |
| --- | --- |
| **属性** | **功能描述** |
| Speed | 动画默认速度 |
| Motion | 此状态对应的动画片段 |
| Foot IK | 是否对足部使用反向动力学（IK），仅对人形动画有效。 |
| Write Defaults | 动画状态是否写回默认值。解释：在动画初次播放之前会有一个起始状态，如果勾选此项，那么在动画中止时当前状态会覆盖起始状态。 |
| Mirror | 镜像开关。仅对人形动画有效。 |
| Transitions | 所有状态转移的列表。 |

#### Solo 和 Mute 功能

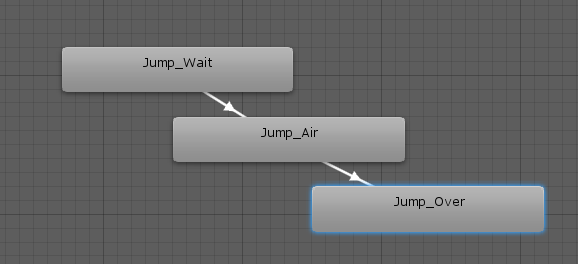
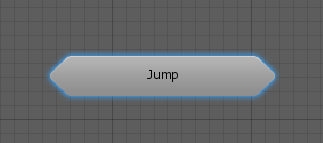
一般在复杂的状态机中做调试作用的，以下是勾选Solo和Mute的特性：

* 如果选择了mute，那么被选择的状态转移一定会被禁用；
* 如果不选择solo，在没有变量控制（结束条件为“exit time”）的情况下，该状态优先选择动作列表中最前（或者说最上的）的状态转移；
* 如果选择了某个solo，那么在没有变量控制（结束条件为“exit time”）的情况下，优先选择标记solo的状态转移；
* 如果有多个状态转移选中了solo，那么优先选择这些已选中solo的状态转移中，在动作列表中靠前的状态转移；

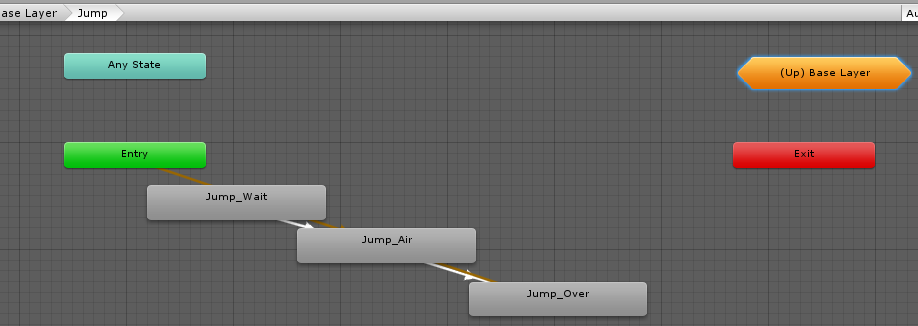
#### 子状态机

一个角色有复杂的动作，这是由多个阶段组成的。与其用单个状态处理整个操作，还不如确定单独的阶段，并为每个单独使用一个状态。 比如一个角色跳跃到落下有弯腰、起跳、落地三个动作，我们可以将三个动作合并在一个跳跃阶段中处理。

这样处理的好处是可以将一系列相同状态的动作都变为一个状态机处理，简化流程图了，避免了状态机变得庞大而笨拙。

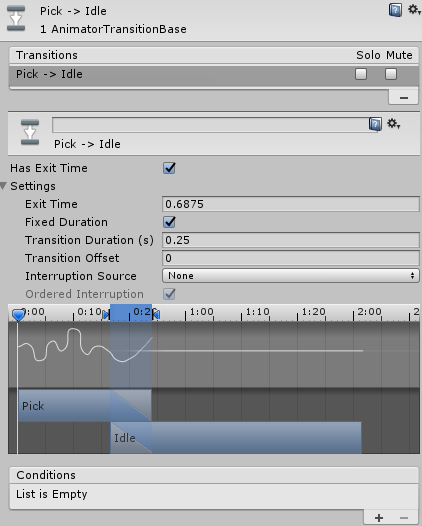
双击子状态机后，可以允许继续编辑子状态机，就像它是一个完全独立的状态机一样。



#### 动画状态转换

Unity的动画转换允许动画在不同状态下过渡。动画转换不仅定义了状态之间的混合需要多长时间，而且定义了它们应该在什么条件下激活。只有当某些条件为真时，才能设置转换。

如果转换时间不为零，则动画切换时将会对过渡前后的动画进行混合，混合的结果取决于两个动画位移、旋转等属性的插值。



|  |  |
| --- | --- |
| **属性** | **功能描述** |
| Has Exit Time | 是否启用退出时间，如果开启本选项，那么在本状态切换到其它状态时，要先花时间退出本状态，用于必须有动画过渡的情形。 |
| Settings | 详细设置，可以展开以下选项。 |
| - Exit Time | 当勾选了Has Exit Time时本选项生效，本选项指定了具体的退出时间。这个值是一个归一化的值，和动画长度有关。例如如果取值0.75，代表当动画播放了75%时开始过渡到下一个动画。  For looped animations, transitions with exit times smaller than 1 are evaluated every loop, so you can use this to time your transition with the proper timing in the animation every loop.  对循环的动画来说比较复杂。如果这个值小于1，那么每一帧都可以在指定的时机过渡到下一个状态；如果这个值大于1，那么就会在一定时间后再过渡到下一个状态。例如3.5代表在播放3次半动画以后，再进行切换。 |
| - Fixed Duration | 选中此项，则切换时间以绝对时间（秒）表示，否则以归一化的时间比例表示。 |
| - Transition Duration | 过渡时间，下面有详情图示可以参考。 |
| - Transition Offset | 过渡时间的偏移量，同样可以参考详情图示。 |
| - Interruption Source | 选择过渡是否可以被打断。 |
| -- Ordered Interruption | 与动画打断相关。 |
| Conditions | 当动画变量符合相应条件时触发动画状态转移，这是动画与脚本交互的关键要素。 |

**转换中断**

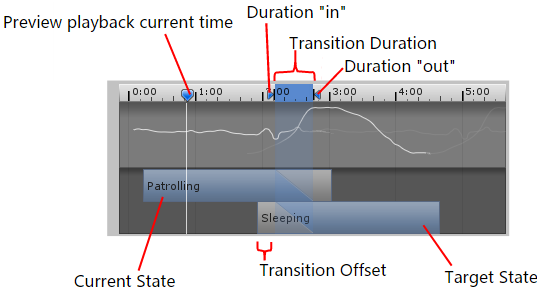
以下详细解释Interruption Source选项，它用来控制状态如何被打断。

例如我们的动画系统下有一个默认的Idle动画，Idle动画有Jump动画分支和Walk动画分支，而Walk动画分支下又有一个Jump2动画分支，当我们同时满足两个Jump动画的切换条件时，Interruption Source就可以判断优先从哪个动画源切换到Jump或Jump2。Interruption Source中四个选项如下:

| 取值 | 描述 |
| --- | --- |
| None | 关闭Interruption Source选项 |
| Current State | 从当前源打断 |
| Next State | 从下一个源打断 |
| Current State then Next State | 优先从当前源打断，如果当前源不能切换到下一个状态则从下一个源打断。 |
| Next State then Current State | 与上一个选项相反。 |

**转换图形编辑界面**

可以通过直接操作图形编辑界面修改一些参数的值，以代替直接输入数字。



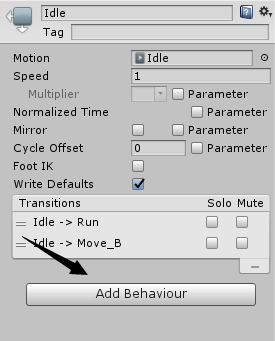
如图：

* 拖动“out”标记修改动画过渡的时间长度。
* 拖动“in”标记改变过渡长度与退出时间。
* 拖动下一个动画标签，可以调整过渡的偏移。
* 拖动当前预览的帧“Preview playback current time”，可以观察过渡效果。

#### 状态机脚本（State Machine Behaviours）

动画状态机上可添加脚本，该脚本类是特殊的脚本类。该脚本允许编写当状态机进入、退出或保留在特定状态下时执行的代码。

通过状态机下的“Add Behaviour”可以创建添加该类脚本。



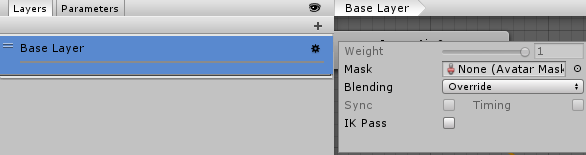
状态机脚本有一些预定好的消息：OnStateEnter, OnStateExit, OnStateIK, OnStateMove, OnStateUpdate。

|  |
| --- |
| using *System*.*Collections*;  using *System*.*Collections*.*Generic*;  using UnityEngine;  public class Idle : StateMachineBehaviour {  // 本函数在开始进入某个状态时调用  override public void OnStateEnter(*Animator* animator, *AnimatorStateInfo* stateInfo, int layerIndex)  {  }  // 在进入和退出之间，每帧调用  override public void OnStateUpdate(*Animator* animator, *AnimatorStateInfo* stateInfo, int layerIndex)  {  }  // 动画状态退出时调用  override public void OnStateExit(*Animator* animator, *AnimatorStateInfo* stateInfo, int layerIndex)  {  }  // 本函数在Animator.OnAnimatorMove()之后调用，适合做移动物体的操作。  override public void OnStateMove(*Animator* animator, *AnimatorStateInfo* stateInfo, int layerIndex)  {  }  // 动画IK更新后调用，适合做设置IK的操作  override public void OnStateIK(*Animator* animator, *AnimatorStateInfo* stateInfo, int layerIndex)  {  }  } |

### 动画层级（Animation Layers）

如果一个角色要播放移动射击动作，上下身的动作是需要分离的。因此我们需要有不同的动画层的来控制身体不同部位复杂动作。

通过点击在动画窗口左上角的“Layers”标签可以管理当前动画层的信息。点击“➕”号按钮可以添加一个新的层级。在已有的层级右侧的齿轮图标点击后可以设置当前动画层的“Weight（权重）”、“Mask（遮罩）”、“Blending（混合类型）”、“IK（逆向运动学）”等属性。



|  |  |
| --- | --- |
| Weight（权重） | 当前动画层级播放的动画对角色动作的影响程度。 |
| Mask（遮罩） | 通过对Avatar Mask（模型化身遮罩）的设置来屏蔽肢体部分动画效果。比如当前动画层负责角色下半身移动动作，该层不能对玩家的上半身的动作有影响，因此需要屏蔽移动动画对上半身肢体的控制。 |
| Blending（混合类型） | 设置动画混合为“Override（覆盖）”或“Additive（叠加）” |
| Sync（同步） | 点击该设置，可以去选择“Source Layer”去同步其他层的动画，选择后会复制该层的所有状态机，不同的是复制的状态机是没有动画片段的（Clips），玩家重新复制新的动画片段实现混合同步。该设置的作用是，比如你在需要角色在不同“受伤”和“健康”状态下有不同的攻击动作，比如受伤状态脚本移动是跛脚的，那么就可以同步移动动画层级下的状态机。 |
| Timing（时间配置） | 和Sync选项关联，如果选中那么融合两个的动画层级下的状态长度将会取决于Weight的大小，Weight值越大融合时间越偏向自身。如果不选中，那么同步的动画将会调整到原始层动画的长度。 |
| IK Pass（逆向运动学传递） | 开启后，动画状态机将会传递IK信息到状态机脚本。调用函数“OnStateIK”。 |

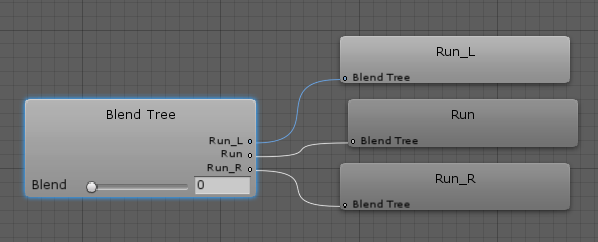
### 动画混合树（Blend Trees）

在游戏开发的需求中，通常会遇到需要对两个或多个动画进行混合。比如从行走到跑步动作，或者在跑步中往左和往右偏移。如果采用动画过渡的方式来实现，很难去处理一个动画可对多个动画混合的效果，因为动画过渡是用于一段时间内完成由一个动画到另一个动画状态的平滑过渡。因此这里如果要实现一个动画状态会对多个动画片段的混合，就需要用到动画混合树。动画混合树可以作为状态机中一个特殊的动画状态存在。

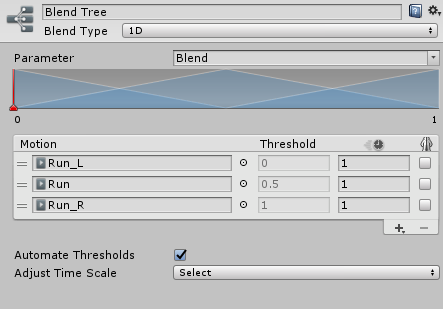
动画混合是利用插值技术对多个动画片段的混合，每个动作对最终结果的影响取决于权重（混合参数）。

制作一个动画混合树需要以下步骤：

1. 在Animator Controller视图中点击空白区域
2. 弹出菜单中选择Create State->From New Blend Tree
3. 双击混合树可以进入混合树视图



如果选中BlendTree可以看到当前选中节点和相邻子节点设置。

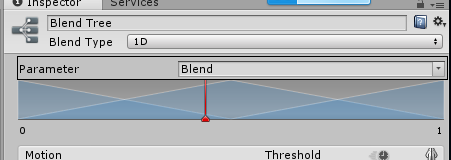


#### 1D混合

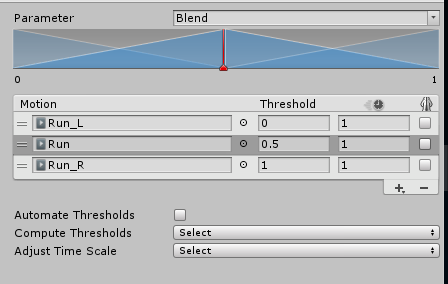
在Inspector视图中的Blend Node属性面板中，第一个选项就是混合类型。

其中1D混合即是通过唯一一个参数来控制子动画的混合。需要选择Parameter来控制混合树，拿人物往前跑举例， 0.5为向前跑动，0则为向左偏移，1则为向右偏移。

下图中，中间的图像表示动画混合权重，比如在0位置将会播放向左跑动画，此时图形没有和其他部分重合。在0~0.5的区域则看到一个重合的深色三角形，这里根据权重混合向左和向前跑的动画。

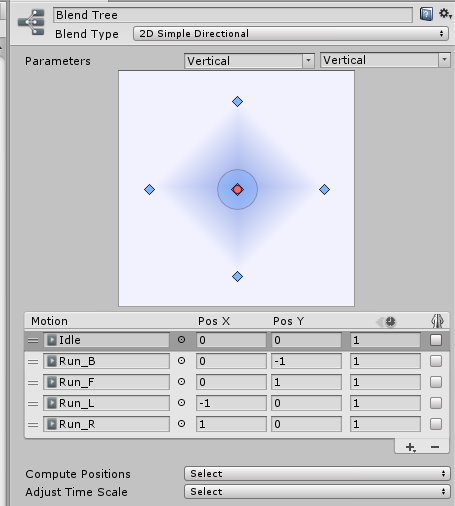


通过点击Add Motion Field在混合树添加动画片段，添加完成后Motion的Thresh选项可以调节混合参数的临界值（如果该值设为了2，则当混合参数变为了2才会完全播放该动作）。闹钟图标为设置该动画剪辑播放速度，最右图标选项为设置左右镜像的动画（只支持类人形动画）。



#### 2D混合

2D混合是指通过两个参数来控制子动画的混合。2D混合有三个模式，区别为每个片段影响的方式。



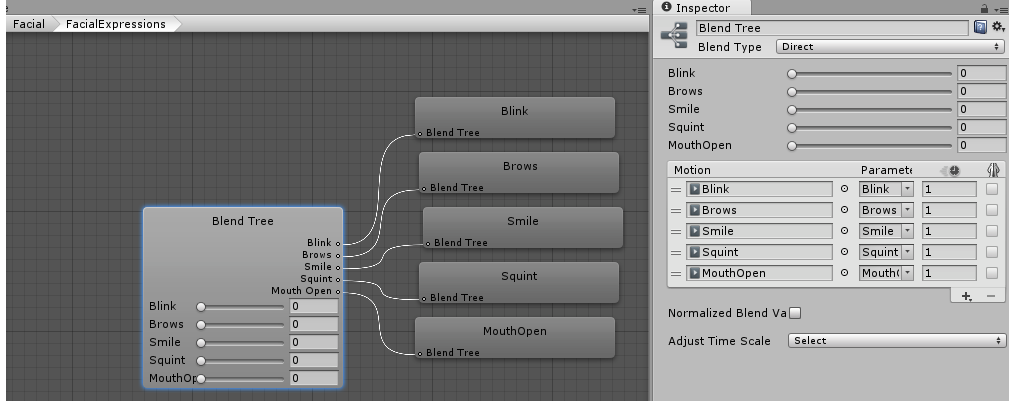
* 2D Simple Directional：2D简单方向，适用于在单个方向上只有一个动画的情况，例如只有向前、后、左、右移动，以及原地不动这5个动作的融合。当同一个方向有多个动作，例如向前走和向前跑，这时不适用这个选项。
* 2D Freeform Directional：2D自由方向，也用于方向性动作的融合。在前者的基础上，可以融合同一个方向有多个动作的情况。
* 2D Freeform Cartesian：用于非简单方向性动作的融合，例如X轴代表左转、右转，以及可以融合“向前跑动并右转”等等非简单方向性的动作。

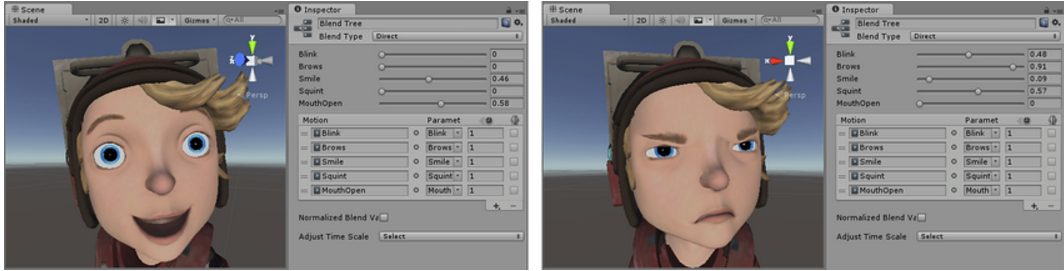
简单的说，2D混合创建了一个2个参数的坐标系，当Parameter的参数值（X，Y）偏向逼近哪个值的动画剪辑时，将会收到该动画的影响。最终会根据所有点在该区域的影响权重，来计算动画混合的效果。

#### 直接混合（Direct Blending）

游戏开发中会遇到部分动画混合与一两个参数没有关系的情况，比如想实现某个参数对某个动作的精确影响，而不会影响到其他动作就需要使用直接混合功能。

比如说我们想设置多个参数分别影响对不同的动画造成影响。例如表情动画，如果要实现表情的自然过渡，可以对混合树进行如下设置：





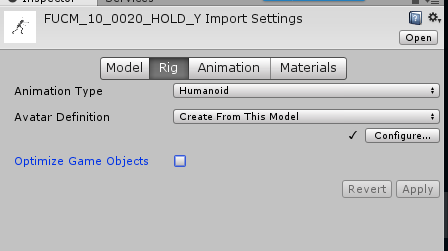
## 使用人形角色动画

Mecanim动画系统特别适合人形动画角色的制作。人形骨架是在游戏中最普遍的一种骨架结构。由于人形骨骼结构的相似性，用户可以实现将动画效果从一个人形骨架映射到另外一个人形骨架上去，从而实现动画重定向的功能。创建人性动画的基本步骤是建立一个骨架结构到用户实际的骨架结构映射，这种映射关系为Avatar。

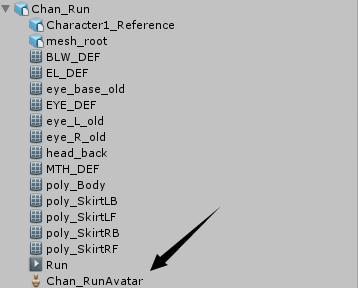
### 人形骨架映射（Avatar）

#### 创建Avatar

人形角色动画创建是在对导入的模型资源Rig标签页中，修改Animation Type参数为Huamnoid。此时可以看到如果该模型可以创建骨架结构映射，那么Configure按钮处应该会有勾的提示（如果该模型无法创建，则会显示为叉）。点击Apply保存后，引擎会在资源中生成Avatar。

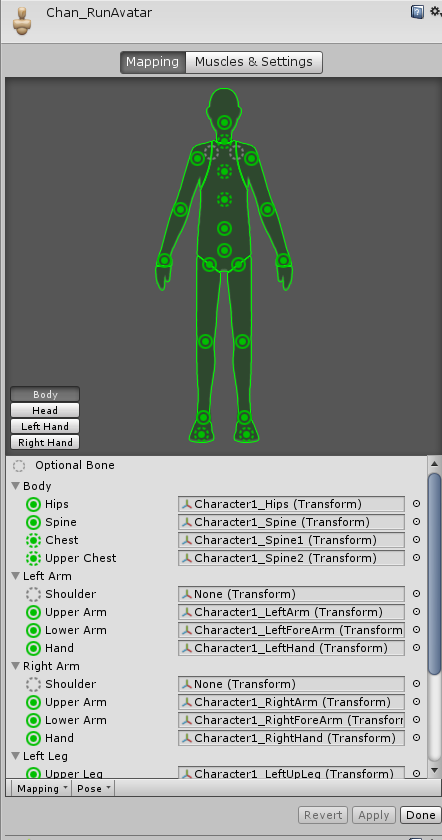


生成的Avatar资源如下图，选择该资源可以配置Avatar:



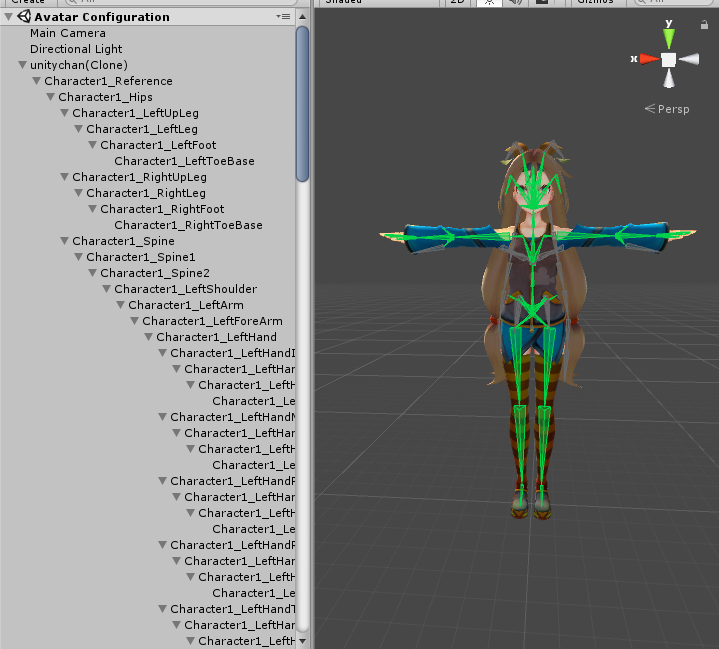
#### 编辑Avatar

当我们点击Rig标签处的Configure按钮将会进入映射编辑界面，因为之前的匹配，仅仅是匹配成功了必要的骨骼关节。如果要达到更好的效果，需要将非关键骨骼也进行匹配，并使模型处于T形姿态（比如下图肩膀附近的骨骼并没有做匹配）。



Avatar是Mecanim系统中极为重要的模块，因此要确认模型资源是否正确的设置。必须匹配的骨骼是实线圆圈，可选匹配的骨骼是虚线圆圈。

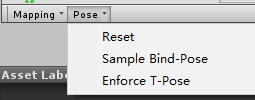
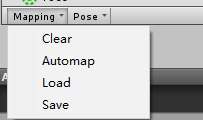
在Avatar创建成功后，最好进入界面去确认Avatar的有效性，确认模型资源骨骼是否正确匹配，以及模型是否显示为T形。



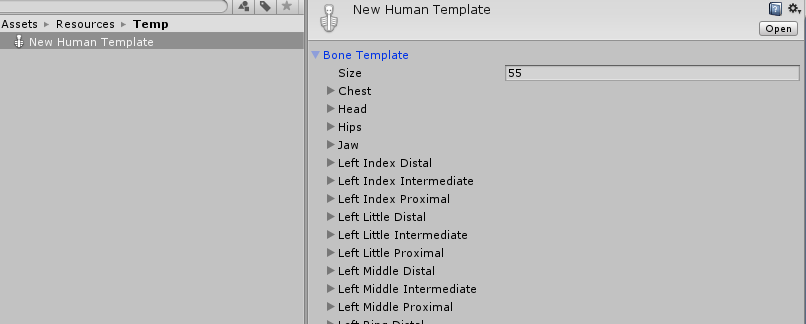
Scene视图将会用来显示当前选中模型的骨骼、肌肉和动画信息，而不再显示场景。因此点击前注意场景的保存。

如果系统无法自动为模型找到合适的匹配，我们可以进行手动配置。

1. 点击pose->Sample Bind-pose得到模型的原始姿态
2. 点击Mapping->AutoMap可以让系统基于原始姿态创建映射。Mapping->Load可以读取外部设置好的映射文件，同理Save为保存当前设置好的骨骼映射。
3. 匹配失败的部分骨骼可以通过Hierarchy视图中拖动骨骼并指定在映射骨骼位置。
4. 点击Pose->Enforce T-Pose让模式恢复到T形姿态。

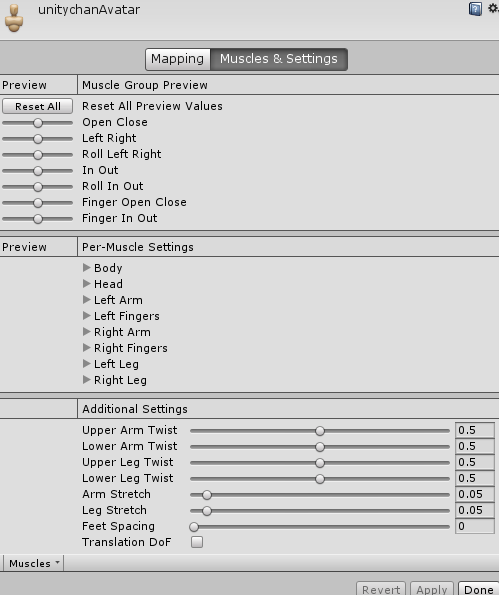
上诉的骨骼映射信息储存信息保存为一个人形模板文件（Human Template File），其文件扩展名为.ht，这个文件可以在所有使用这个映射关系的角色间复用。



#### 设置肌肉（Muscle）参数

Mecanim使用肌肉（Muscle）来限制不同骨骼的运动范围。一旦Avatar配置完成，Mecanim就能解析骨骼结构，为了确保骨骼运动看起来真实，用户就可以在Avatar面板的Muscles选项中调节相关参数。

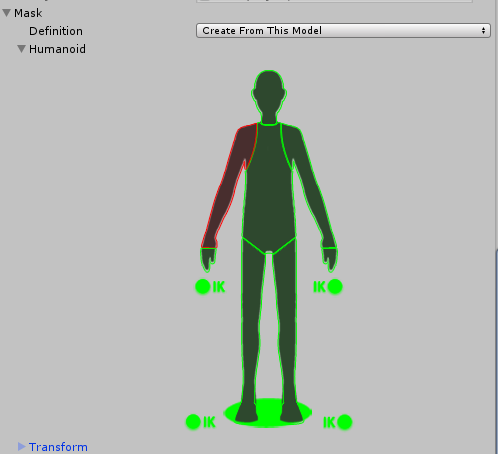
在Preview参数下的Muscle Group Preview可以对一个模块的骨骼进行批量调整，用户也可以在Per-Muscle Settings对每一个骨骼进行细调。



### 人形动画身体遮罩（Avatar Mask）

之前的动画层级控制器中，已经提到过身体遮罩，在游戏开发中我们常常需要控制身体的一部分受动画影响，而其他部分是被屏蔽的。比如一个角色抱着物体移动，这时就不需要移动的动画继续控制双手摆动，这样就需要在Mask中屏蔽手臂运动。

在模型资源的Animation选项卡的面板中找到Mask控制选项。



点击身体区域，变红即表示该动画屏蔽了该肢体的运动。同时可以看到手脚可以点击IK的选项，表示可以通过点击开关该部位的IK功能。

另外我们可以通过Asset->Create->Avatar Mask命令创建Mask资源，并保存为.mask文件，这意味着我们可以创建mask模板在动画控制器（Animator Controllers）中来实现资源的复用。

### 人形动画的重定向

在游戏开发中经常会通用一套动作模型，比如一款RPG游戏，角色可能千奇百怪但是模型动作却不会那么多。重新利用现有动画以供多个模型使用的过程，从而无需创建全新动画，这种方法便是动画重定向。

而在Unity中，重定向只能支持人形模型。因为目前只有人形模型才能创建骨架映射。

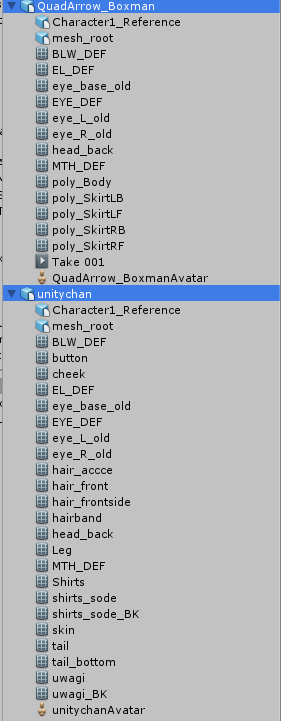
动画重定向有两种形式：

一种是要与其共享动画的角色模型将和最初创建该动画模型使用相同骨架，这种方式相当于保证模型骨骼的所有层级和命名都相同，这样一个动画将会在相同骨架模型上生效，因为动画是根据名字和路径去查找操作的骨骼的。

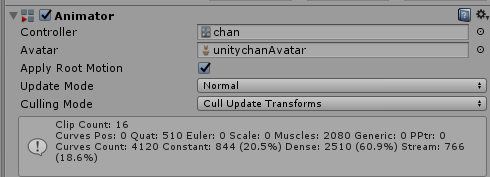
另一种形式为通过Avatar，让动画通过映射实现在多个不同骨架下的相同模型动作。

我们先实现Avatar方式的动画重定向：

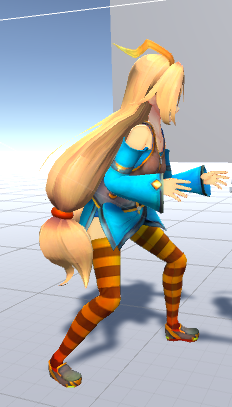
首先我们需要准备两个不同的模型，分别生成他们的人形骨骼映射Avatar。



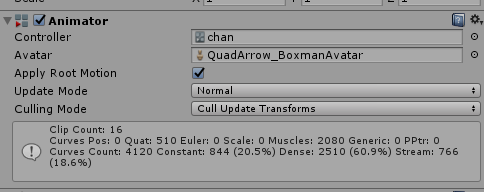
将我们需要加入动画的模型拖到场景中去，然后在模型根节点上创建一个动画控制器，该控制中需要加入一组动画片段。之后在角色挂载的控制器上Avatar选项选择我们在该模型资源下生成的人形骨架映射Avatar。



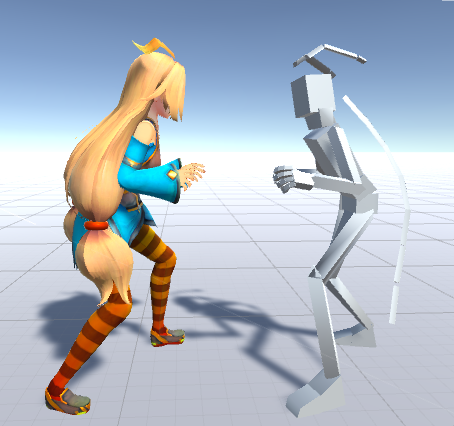
点击运行后，模型将会播放动画控制器中的默认动作。



之后我们将另外一个模型拖入场景中，同样在其模型根节点上创建一个动画控制组件，之后Avatar选项拖入自身的人形骨架映射Avatar。而Controller选择之前模型的动画控制器。



最后运行Unity，我们可以看到两个模型都执行相同的动作。

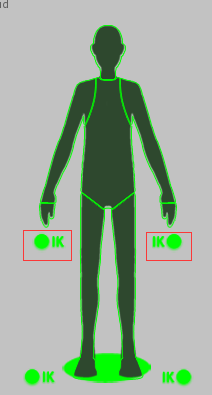


### IK逆向运动学

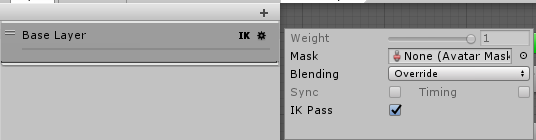
相对于由手臂肌肉带动手腕然后举起物体这种FK正向运动，那么你抓住一个物体，当这个物体位置和旋转变动时会带动你的手臂，从而带动你整个身体运动（比如角色挥舞武器攻击，砍到墙壁后武器停下，动作也随之停下）这种就叫做IK逆向运动。

Unity中的逆向运动学需要人形角色才能使用，使用前我们需要有以下准备：

1. 人形动画资源，模型和动作。模型需要创建Avatar骨架映射。该角色需要做逆向运动学动画的资源设置Animation选项卡中，确认Mask参数的肢体IK是打开的。



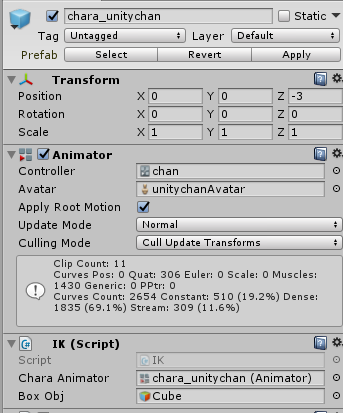
1. 动画控制器（Animator）的Layer层设置下，打开了IK Pass。



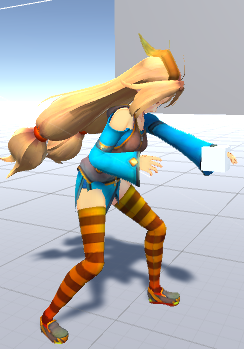
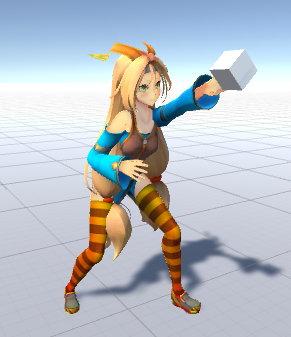
1. 需要准备一个接受IK信息的脚本。新创建一个Monobehaviour脚本，其中加入OnAnimatorIK()函数接受动画控制器传递的IK信息（当动画控制器播放打开了IK属性的动画状态时，将会调用该函数）。然后将该脚本添加到角色带有动画控制器的物体上。

|  |
| --- |
| using UnityEngine;  public class IK : MonoBehaviour  {  public Animator CharaAnimator;  /// <summary>  /// 影响角色动画的物体  /// </summary>  public GameObject BoxObj;  void OnAnimatorIK()  {  //让角色看向该物体 头部盯着boxObj  CharaAnimator.SetLookAtWeight(1);  CharaAnimator.SetLookAtPosition(BoxObj.transform.position);  //让角色的左手位置收到BoxObj的位置影响  CharaAnimator.SetIKPositionWeight(AvatarIKGoal.LeftHand, 1);  CharaAnimator.SetIKPosition(AvatarIKGoal.LeftHand, BoxObj.transform.position);  //让角色的左手角度收到BoxObj的角度影响  CharaAnimator.SetIKRotationWeight(AvatarIKGoal.LeftHand, 1);  CharaAnimator.SetIKRotation(AvatarIKGoal.LeftHand, BoxObj.transform.rotation);  }  } |

脚本创建后给暴露的参数赋值，Animator为自身的动画控制器，Box Obj是Unity预制的Cube物体。



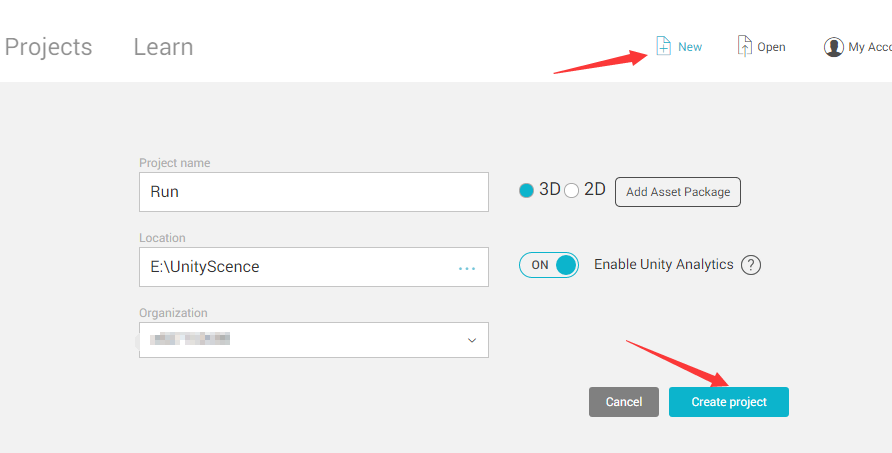
运行并拖动Cube物体，可以看到角色左右会被Cube物体拉扯移动，并且角色会看向该物体。

## 实践：实现一个带有动画且操作流畅的角色控制器

### 工程创建

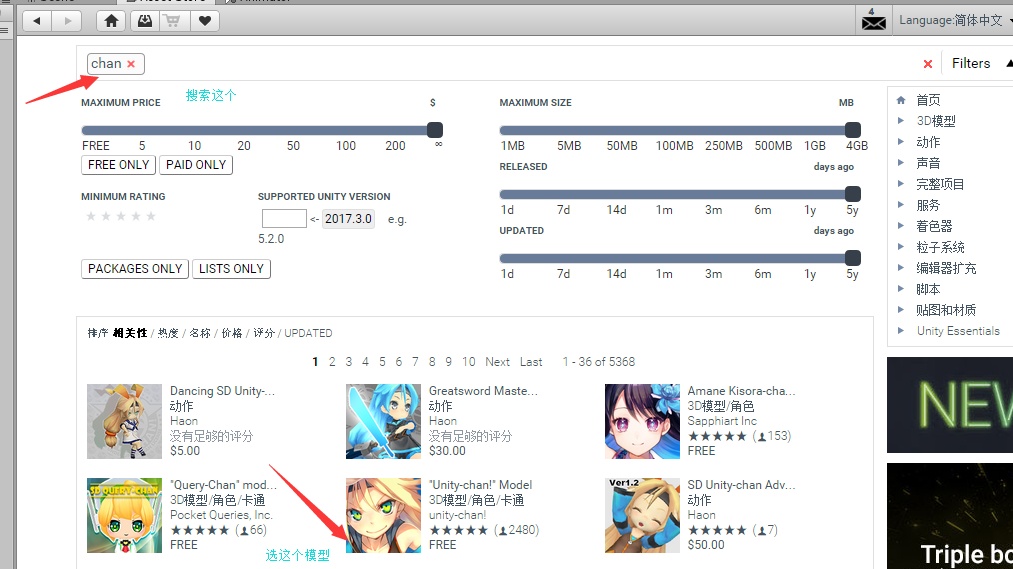
首先运行UNITY3D，点击右上角的NEW选项进行工程创建。在选择弹出后的页面输入你的工程名称，点击Create Project按钮，这样完成了我们的工程创建。



创建Unity工程

### 模型下载

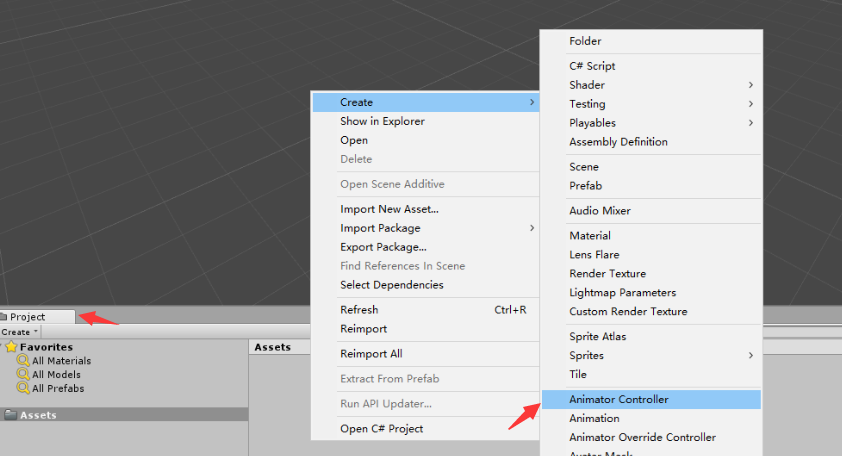
本例中的人物模型我们可以使用官方免费提供的模型，在unity中Window菜单中点击Asste Store ，打开资源商店进行下载，完成后导入即可。



在Unity商店下载资源

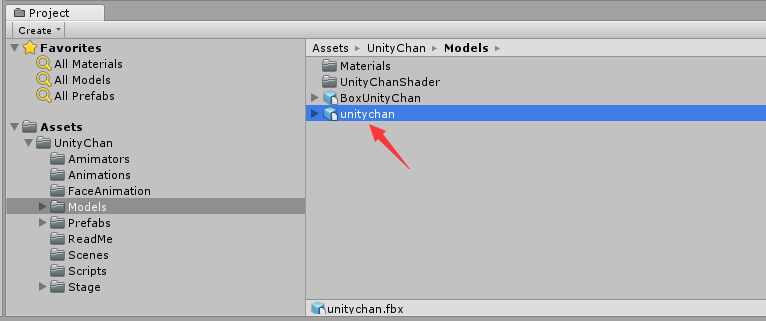
### 动画状态机创建

在上面的章节中，我们已经介绍了如何创建动画状态机，以及相应的功能。在本例中就来进行实际的创建与应用实践。首先在Unity的Project界面新建一个动画控制文件，并命名为PlayerAnimatorController.controller。如图：



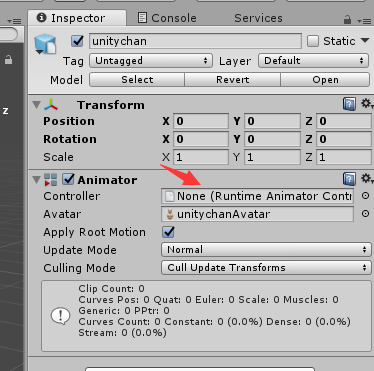
创建动画状态控制机

完成后，找到我们之前下载导入后的人物模型，路径如下：



人物模型路径

将其拖入场景中，并在旁边的Inspector视图Animtor组件中的Controller选项拖入我们创建好的动画控制器文件进行应用。如图：



动画控制器应用

完成以上步骤后，双击PlayerAnimatorController.controller文件，进入Animtor视图中，就可以进行动画状态机的动作导入以及动作控制。

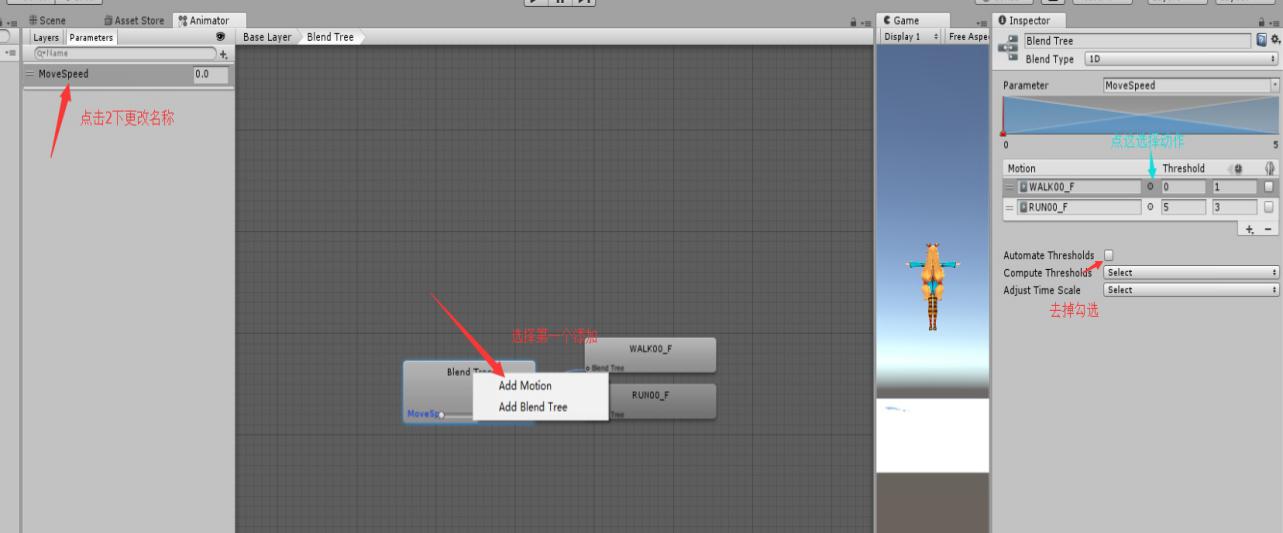
### 动画状态机配置

创建完成后，我们现在需要让人物能够动起来，这个时候就需要在动画状态机中设置动画的触发条件，以及播放什么动作。这里，我们设定我们人物暂时只播放3种动画，如下：

1. 跑步动作。
2. 跳跃动作。
3. 下滑动作。

#### 跑步动作

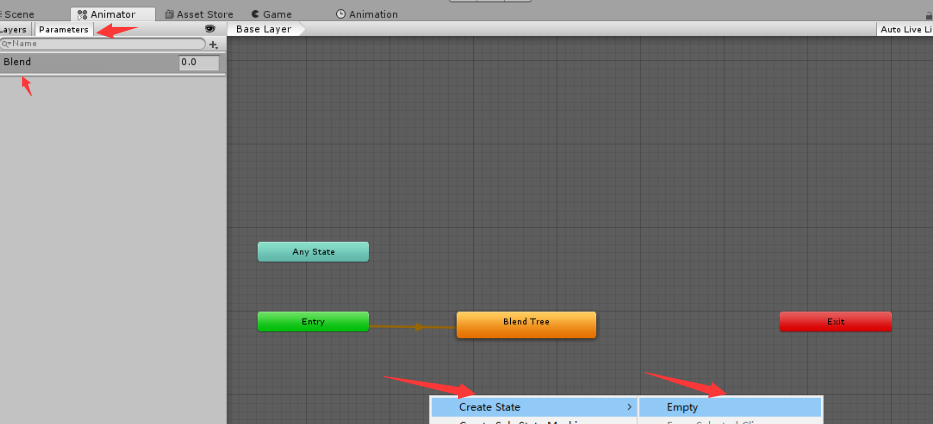
首先我们先进行常规游戏中最主要的动作设置—跑步动作。跑步动作是由2部分组成。一是静止时的动作，一是跑起来的动作。所以为了完成我们的跑步动作，我们需要新建一个BlendTree（动画混合树）,进行动画的混合。完成后双击进入编辑。点击混合树，添加2个Motion,并在右边的Inspector视图选择好动作。这边动作分别选择我们导入好的WALK00\_F与RUN00\_F就行。并在左上角的Parameters（参数）界面，将Blend参数重新命名为MoveSpeed。方便我们后续代码获取这个值，进行动画速度的修改。然后去掉Automate Thresholds勾选，进行动画速度的设置。如下：



动画混合树设置

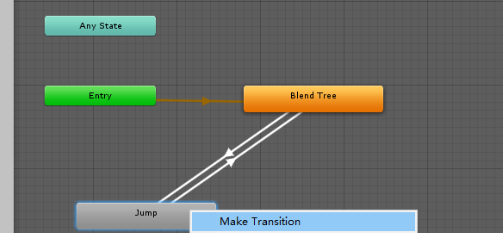
#### 跳跃动作

完成了我们跑步动作的混合后，再来设置我们游戏的跳跃动作。这个时候先回到刚开始创建动画混合树的界面，重新创建一个新的空的状态。如图：



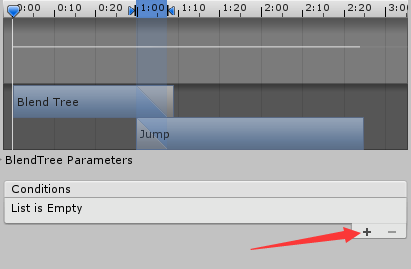
新建动画状态

在右边的Inspector视图的Motion选项选择好人物的跳跃动作—JUMP00。然后我们选择创建好的状态，新建一个Make Transition连线，与创建好的动画混合树连接起来，同时，动画混合树也创建一个连线，与跳跃状态联系起来，形成一个双向连接，用于2个动画之间的过度。如图：



新建Make Transition连线

接着，在左上角的Parameters界面的“+”号按钮，新建一个Bool开关，并命名为IsJump，后续用于控制动画是否进行过渡。然后点击连线，将创建好的开关进行应用，如图：



动画参数应用

在Blend Tree——Jump连线方向，设置为true。Jump——Blend Tree连线方向设置为false，并去掉勾选Has Exit Time选项。此勾选的作用是在动画播放完成后，自动退出当前动画，默认勾选。

#### 下滑动作

下滑动作与跳跃动作设置大致相同，不同的是选择的动画是—SLIDE00。这里就不再过多阐述。

### 代码控制

经过上面的步骤，我们已经配置好了玩家的动画状态控制机，接下来就是在代码中控制并播放玩家的动作了。

#### 播放跳跃动画

首先新建一个PlayerController.cs文件，然后将其挂载到玩家身上，用于控制玩家的的行为。那么第一步我们得拿到玩家当前的动画控制器，那么我们在代码中写入以下代码：

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class PlayerController : MonoBehaviour {  public Animator playAnimtor; //动画控制组件  public float jumpPower; //玩家的跳跃高度  public float speed;  void Start ()  {  playAnimtor = GetComponent<Animator>();  } |

获取动画组件

这样我们就拿到了玩家当前动画的组件，以及当前的动画控制器。紧接着第二步就是进行玩家动画的播放以及关闭了。如下：

|  |
| --- |
| void Update ()  {  if (Input.GetKeyDown(KeyCode.Space)) //按下空格键  {  playAnimtor.SetBool("IsJump", true); //播放跳跃动画  }  } |

简单播放动画

这个演示中我们在Update函数中进行检测，在游戏运行时，如果玩家按下了空格键，跳跃动画就会播放。细心的读者会发现，我们只进行了打开跳跃动画的开关的操作，没有进行关闭，就会造成动画在跳跃动画进行循环。怎么解决呢？通常的方法是在动画播放完毕后，执行以下代码，进行动画的关闭。可以参考之前**Curves和Events的实例说明小节中所提到的动画事件**。

|  |
| --- |
| playAnimtor.SetBool("IsJump", false); //关闭跳跃动画 |

关闭动画播放

#### 播放下滑动画

播放下滑动画与跳跃相似，且由于之前我们已经获取到了当前的动画组件。所以，现在我们只需要在原先的代码的基础上添加一段代码就可进行播放。如下：

|  |
| --- |
| if (Input.GetKeyDown(KeyCode.K)) //按下K键  {  playAnimtor.SetBool("IsSlide", true); //播放下滑动画  } |

#### 播放跑步动画

由于我们的跑步动画是由静止动画与前跑动画组合而成，且在我们的动画状态机的设置当中，我们是根据玩家的速度来进行两个动画的播放与切换，所以在游戏一开始玩家速度为0时就在进行静止动画的播放。现在我们并没有进行玩家的移动操作，但是想要看到跑步的效果，我们只需要更改一下，之前在动画状态机中设置好的参考值—MoveSpeed就行。如下：

|  |
| --- |
| void Update ()  {  if (Input.GetKeyDown(KeyCode.L)) //如果按下L键  {  playAnimtor.SetFloat("MoveSpeed",speed); //根据玩家速度进行更改  speed++；//速度更新  }  } |

这样一来，我们只需要在游戏运行时按下L键就可以看到玩家根据速度播放不同的动画了。在实际游戏中，通常是在Upadte函数中实时更新玩家的速度，用来控制动画的播放，而不是使用按键来进行更新。而且跑步动画不需要停止，只需要更改当前玩家的移动速度就可以满足我们的需求了。我们会在后续的游戏示例中详细运用，这里就不再过多阐述了。