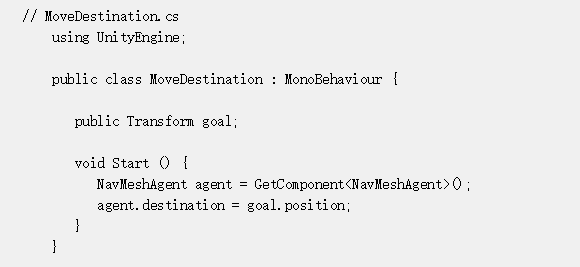
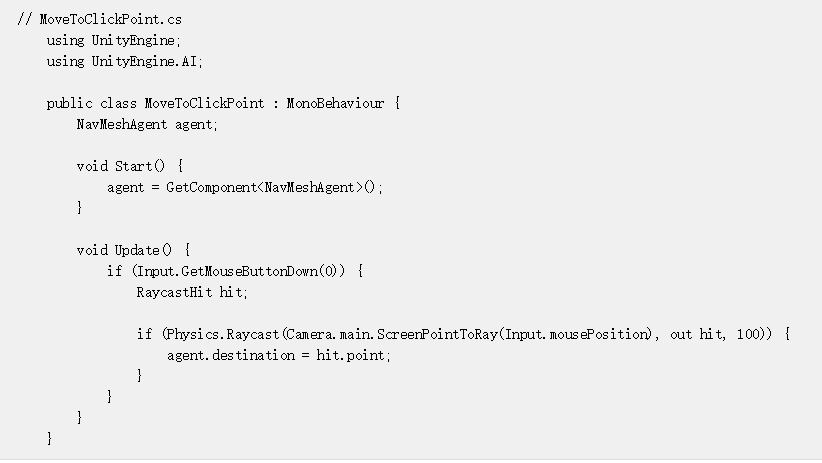
**告诉导航网格代理移动到目标位置**

只需将 [NavMeshAgent.destination](file:///E:\UnityDocumentation\ScriptReference\AI.NavMeshAgent-destination.html) 属性设置为您希望代理移动到的点，即可告诉代理开始计算路径。计算完成后，代理将自动沿路径移动，直至到达目标位置。下面的代码实现了一个简单的类，该类使用一个游戏对象来标记在 *Start* 函数中分配给 *destination* 属性的目标点。请注意，该脚本假定您已从 Editor 中添加并配置了导航网格代理 (NavMeshAgent) 组件。



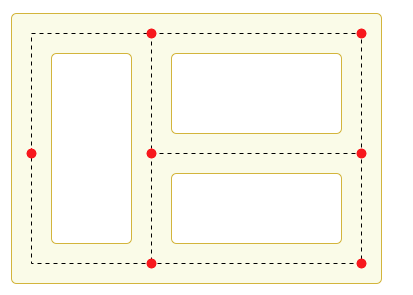
# 将代理移动到鼠标单击的位置

此脚本允许通过在对象表面上单击鼠标来选择导航网格上的目标点。单击位置由\_射线投射\_确定，而非像将激光束指向对象来查看其所在位置（有关此技术的完整描述，请参阅[摄像机射线](file:///E:\UnityDocumentation\Manual\CameraRays.html)页面）。由于 [GetComponent](file:///E:\UnityDocumentation\ScriptReference\GameObject.GetComponent.html) 函数的执行速度相当慢，因此该脚本在 Start 函数期间将其结果存储在变量中，而不是在 Update 中重复调用它。

// MoveToClickPoint.cs

# 在一组点之间进行代理巡逻

许多游戏都有 NPC 负责在游戏区域周围自动巡逻。使用导航系统可实现此行为，但它比标准寻路方法稍微复杂一些；标准寻路方法仅使用两点之间的最短路径就可以实现有限且可预测的巡逻路线。为获得更有说服力的巡逻模式，可保留一组“有用”的关键点让 NPC 以某种顺序通过并访问它们。例如，在迷宫中，可将关键巡逻点放置在交叉点和拐角处，从而确保代理检查每个走廊。对于办公楼，关键点可能是各个办公室和其他房间。

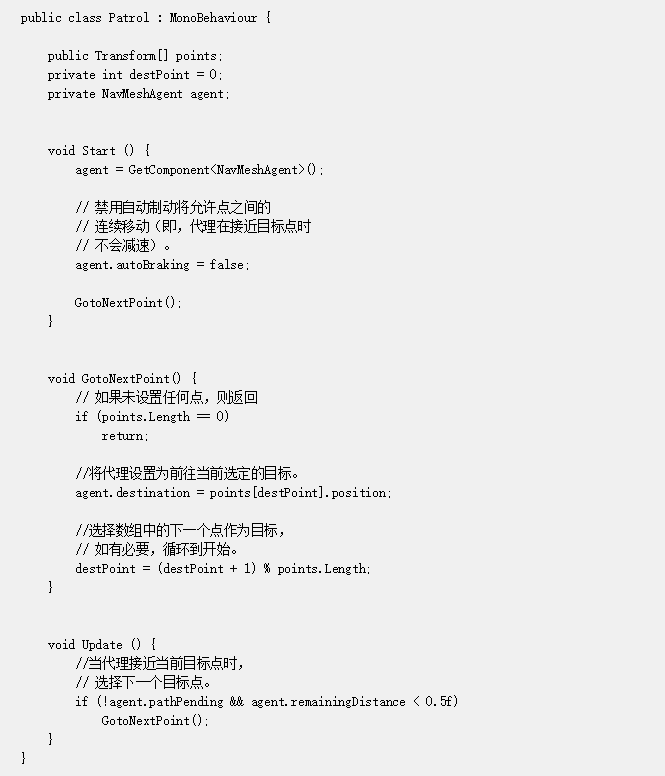


标有关键巡逻点的迷宫

理想的巡逻点序列将取决于所需的 NPC 行为方式。例如，机器人可能只是按照有条不紊的顺序访问这些点，而人类守卫可能会尝试通过使用更随机的模式来发现玩家。可使用下面显示的代码实现机器人的简单行为。

应使用公共变换数组将巡逻点提供给脚本。可从检视面板分配此数组，并使用游戏对象来标记巡逻点的位置。GotoNextPoint 函数可设置代理的目标点（也开始移动代理），然后选择将在下次调用时使用的新目标。就目前而言，该代码将按照巡逻点在数组中出现的顺序循环遍历巡逻点，但您可以轻松修改这种设置，例如使用 [Random.Range](file:///E:\UnityDocumentation\ScriptReference\Random.Range.html) 来随机选择数组索引。

在 Update 函数中，该脚本使用 [remainingDistance](file:///E:\UnityDocumentation\ScriptReference\AI.NavMeshAgent-remainingDistance.html) 属性检查代理与目标的接近程度。当此距离非常小时，将调用 GotoNextPoint 来启动巡逻的下一段。



# 耦合动画和导航

本文档的目标是指导您设置人形角色的导航以使用导航系统进行移动。

我们将使用 Unity 的内置动画和导航系统以及自定义脚本来实现这一目标。

本文假设您已熟悉 Unity 和 Mecanim 动画系统的基础知识。

此处提供了一个示例项目，因此无需从头开始添加脚本或设置动画和动画控制器：

* [NavigationAnimation\_53.zip](file:///E:\UnityDocumentation\uploads\Examples\NavigationAnimation_53.zip) 适用于 Unity 5.3+

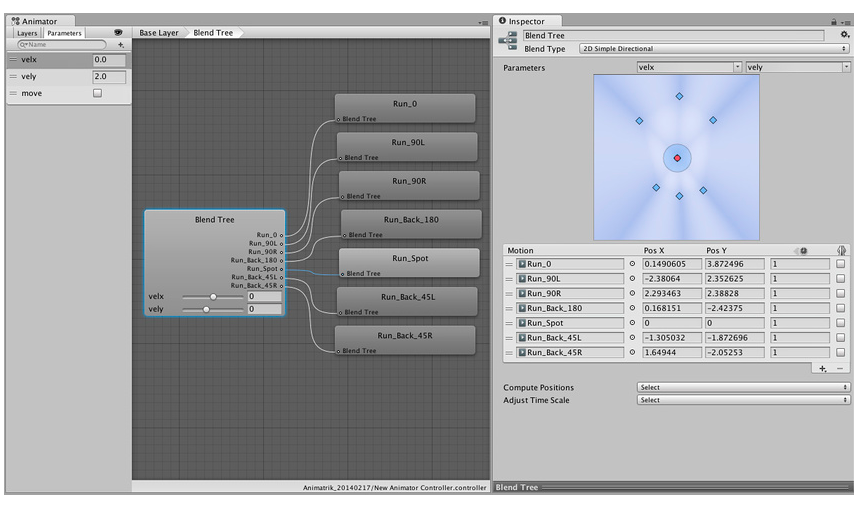
## 创建动画控制器

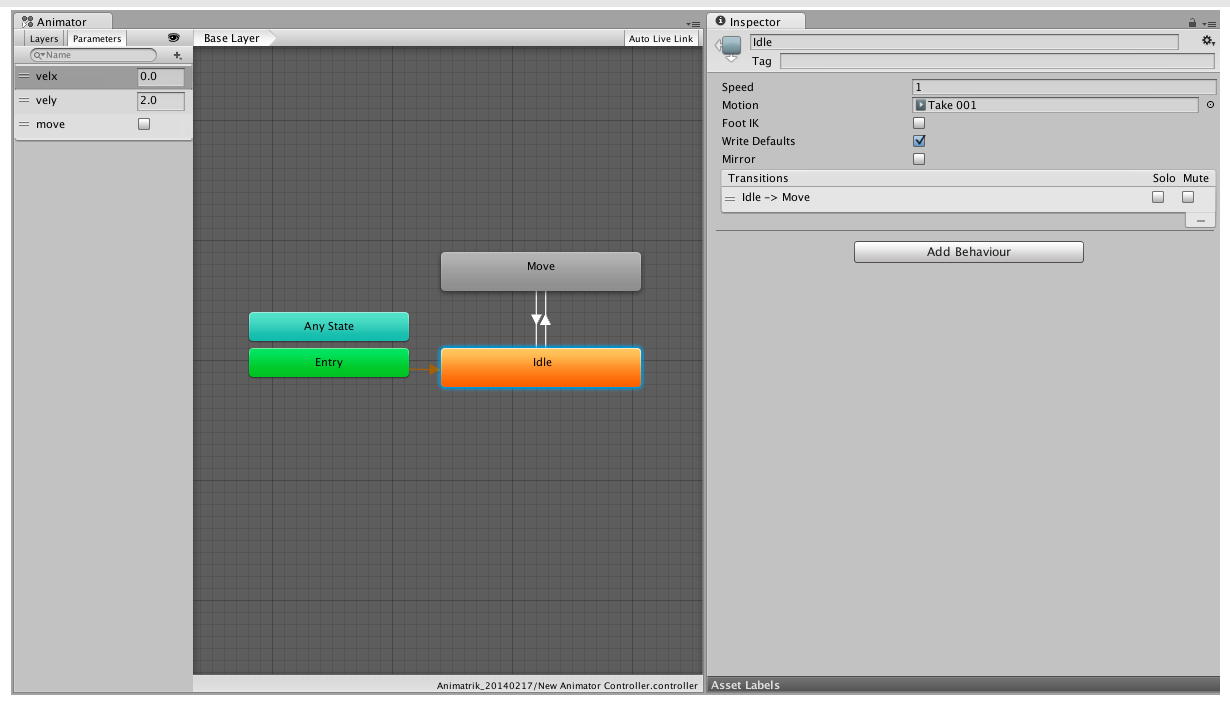
为了获得响应迅速且多功能的动画控制器（涵盖各种动作），我们需要一组向不同方向移动的动画。有时将其称为扫射集 (strafe-set)。

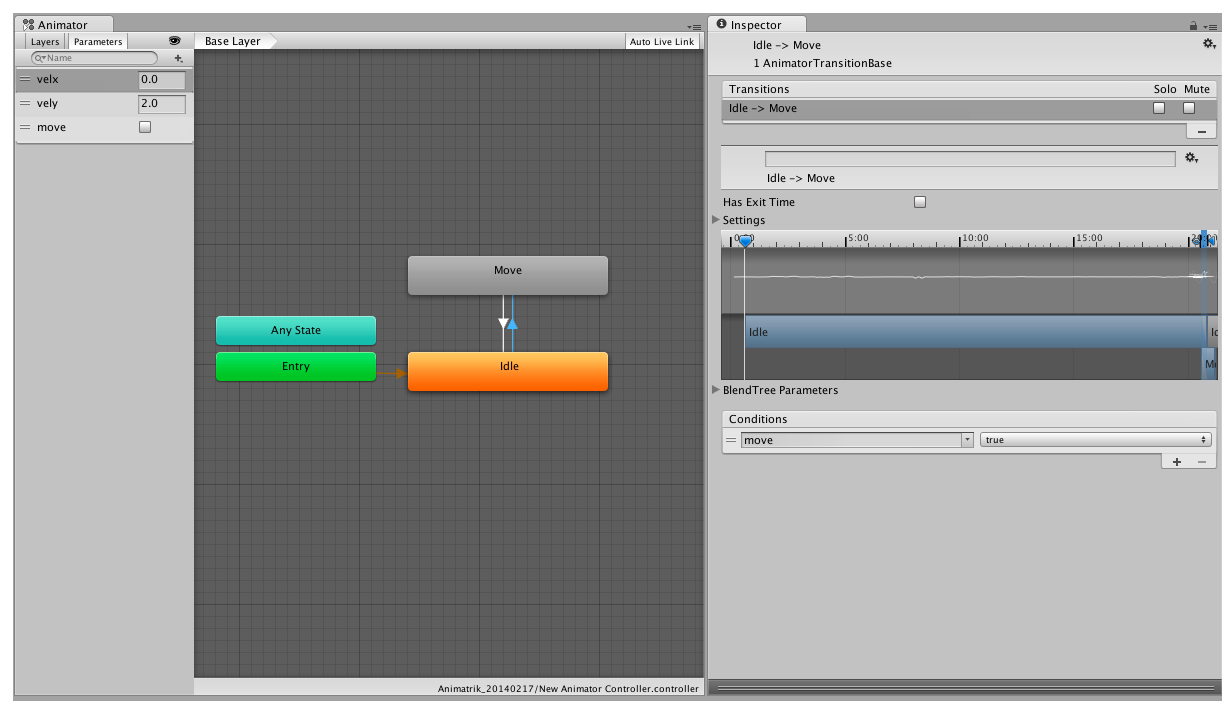
除了移动动画，我们还需要一段站立角色的动画。

我们继续将扫射集组织在 2D 混合树中；选择混合类型：**2D Simple Directional**，并使用 **Compute Positions > Velocity XZ** 放置动画

为进行混合控制，我们添加两个浮点参数 **velx** 和 **vely**，并将它们分配给混合树。

在这里，我们将放置 7 段奔跑动画，每段都有不同的速度。除了前进（+左/右）和后退（+左/右），我们还使用了原地奔跑的动画剪辑。后者在下面的 2D 混合图的中心位置进行了突出显示。采用原地奔跑动画有两个原因，首先，该动画可在与其他动画混合时保持奔跑风格；其次，该动画可以防止混合时出现脚滑。

然后，我们在空闲节点 (**Idle**) 本身中添加空闲动画剪辑。 我们现在有两个独立动画状态，我们将它们与 2 个过渡耦合。

为了控制移动状态和空闲状态之间的切换，我们添加一个布尔值控制参数 **move**。然后，对过渡禁用 **Has Exit Time** 属性。如此便可在动画期间的任何时间触发过渡。为获得快速响应的过渡，过渡时间应设置为约 0.10 秒。

现在将新创建的动画控制器放在要移动的角色上。

按 Play 并在 **Hierarchy 窗口**中选择该角色。现在可在 **Animator 窗口**中手动控制动画值，并更改移动状态和速度。

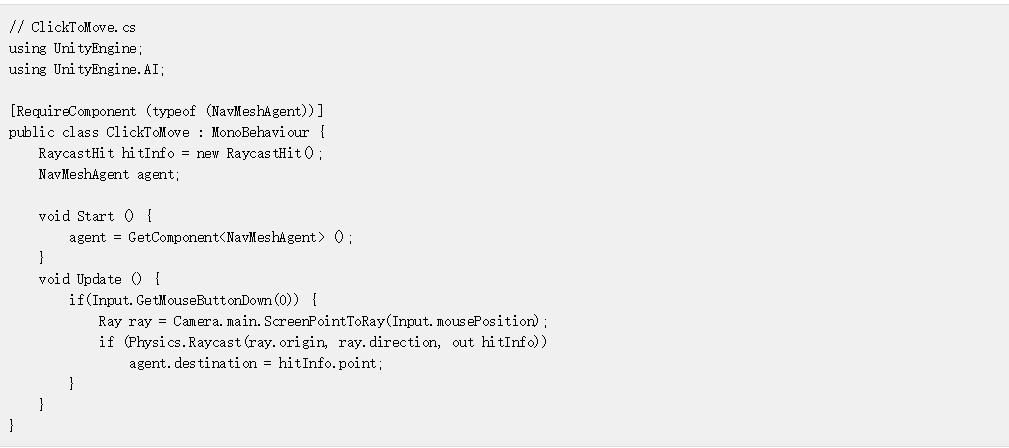
下一步是创建

## 导航控制

在角色上放置一个 **NavMeshAgent** 组件，调整半径和高度，并匹配角色（另外更改速度属性以匹配动画混合树中的最大速度）。

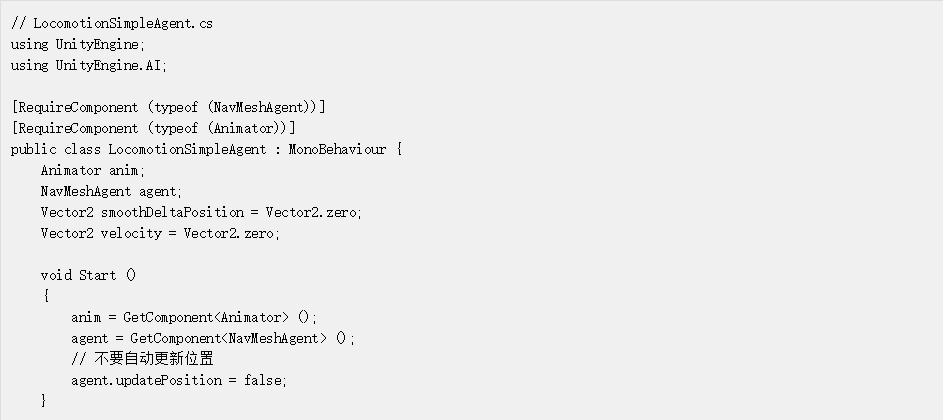
为放入角色的场景创建导航网格。

接下来，我们需要告诉角色要导航的目标。此设置通常与具体应用有非常大的关联性。在这里，我们选择“单击进行移动”(click to move) 行为：根据用户点击屏幕的位置，角色移动到世界中的相应点。

其他控制动画参数的方法。

现在按下 Play，然后在场景中单击，便会看到角色在场景中移动。但是，动画与动作完全不符。我们需要将代理的状态和速度传达给动画控制器。

为了将代理的速度和状态信息传输到动画控制器，我们将添加另一个脚本。





对于此脚本，需要进行一点说明。此脚本放置在角色上，而角色已附加 **Animator** 和 **NavMeshAgent** 组件以及上面的 click to move 脚本。

首先，脚本告诉代理不要自动更新角色位置。我们处理脚本中最后的位置更新。方向由代理进行更新。

通过读取代理速度来控制动画混合。该速度转换为相对速度（基于角色方向），然后经过平滑。然后，转换后的水平速度分量将传递到 **Animator**，另外，空闲状态和移动状态之间的状态切换由速度（即速度幅度）进行控制。

在 OnAnimatorMove() 回调中，我们更新角色的位置以便与 **NavMeshAgent** 匹配。

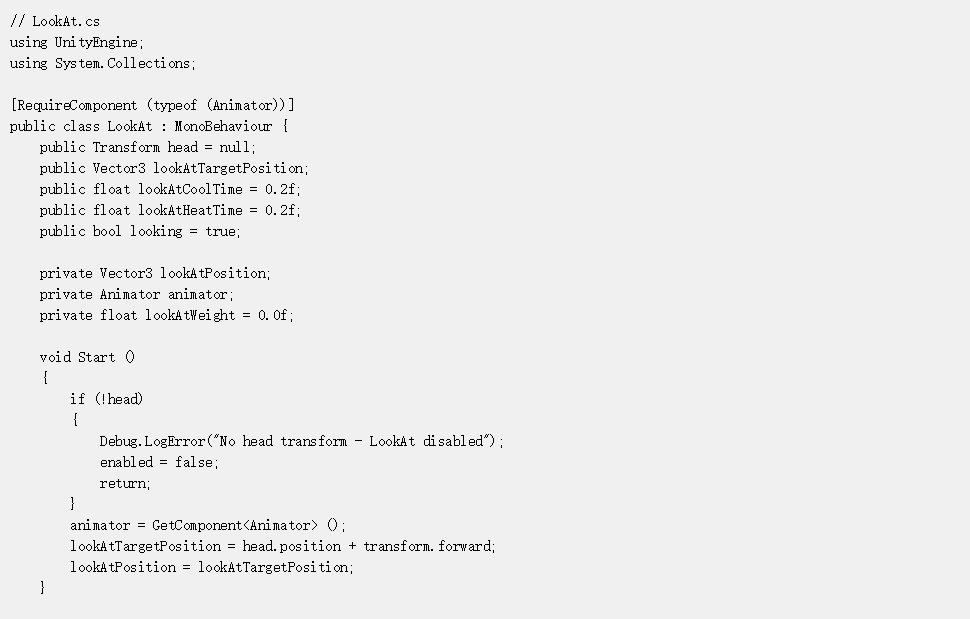
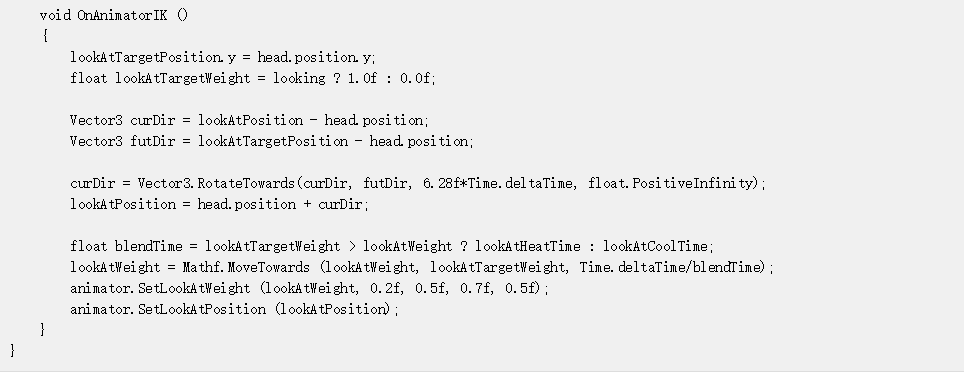
再次播放场景显示动画在最大限度上与动作匹配。

## 提高导航角色的质量

为了提高动画和导航角色的质量，我们将探索几个可能性。

### 注视

让角色注视和转向兴趣点对于表现注意力和期待效果十分重要。我们将使用动画系统 lookat API。因此需要另一个脚本。

将该脚本添加到角色，并将 head 属性分配给角色变换层级视图中的 head 变换。LookAt 脚本没有导航控制的概念；所以为了控制注视位置，我们回到 **LocomotionSimpleAgent.cs** 脚本，并添加几行代码来控制注视。在 Update() 末尾添加：



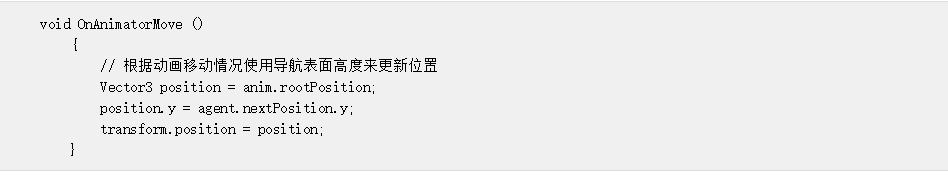
这样就会告诉 **LookAt** 脚本将兴趣点设置为沿路径的大致下一个角点，或者如果没有角落，设置为路径的末端。

自己尝试一下。

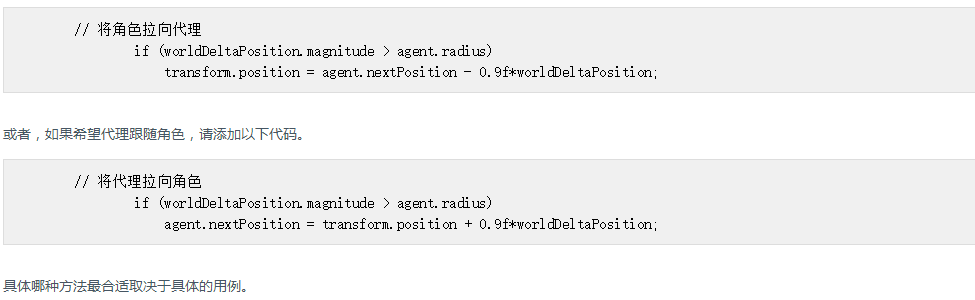
### 使用导航的动画驱动角色

到目前为止，角色完全由代理指定的位置控制。这确保了对其他角色和障碍物的躲避直接转换为角色位置。但是，如果动画未跟上建议的速度，则可能导致脚滑现象。在这里，我们将稍微放松一下对角色的约束。大体上，我们将用躲避质量换取动画质量。

将 **LocomotionSimpleAgent.cs** 脚本上的 OnAnimatorMove() 回调行替换为以下代码



尝试运行此代码时，您可能会注意到，角色现在可以游离于代理位置（绿色线框圆柱体）。您可能需要限制该角色动画游离问题。为此，可将代理拉向角色，或者将角色拉向代理位置。在 **LocomotionSimpleAgent.cs** 脚本上的 Update()方法末尾添加以下代码。



## 结论

我们已经设置一个使用导航系统移动的角色并相应地设置了动画。调整混合时间数字、注视权重等可以改善视觉效果，也是进一步探索此设置的好方法。