高级人工智能

本页介绍了如何设置高级 AI 以创建更有趣的敌人（或朋友）。

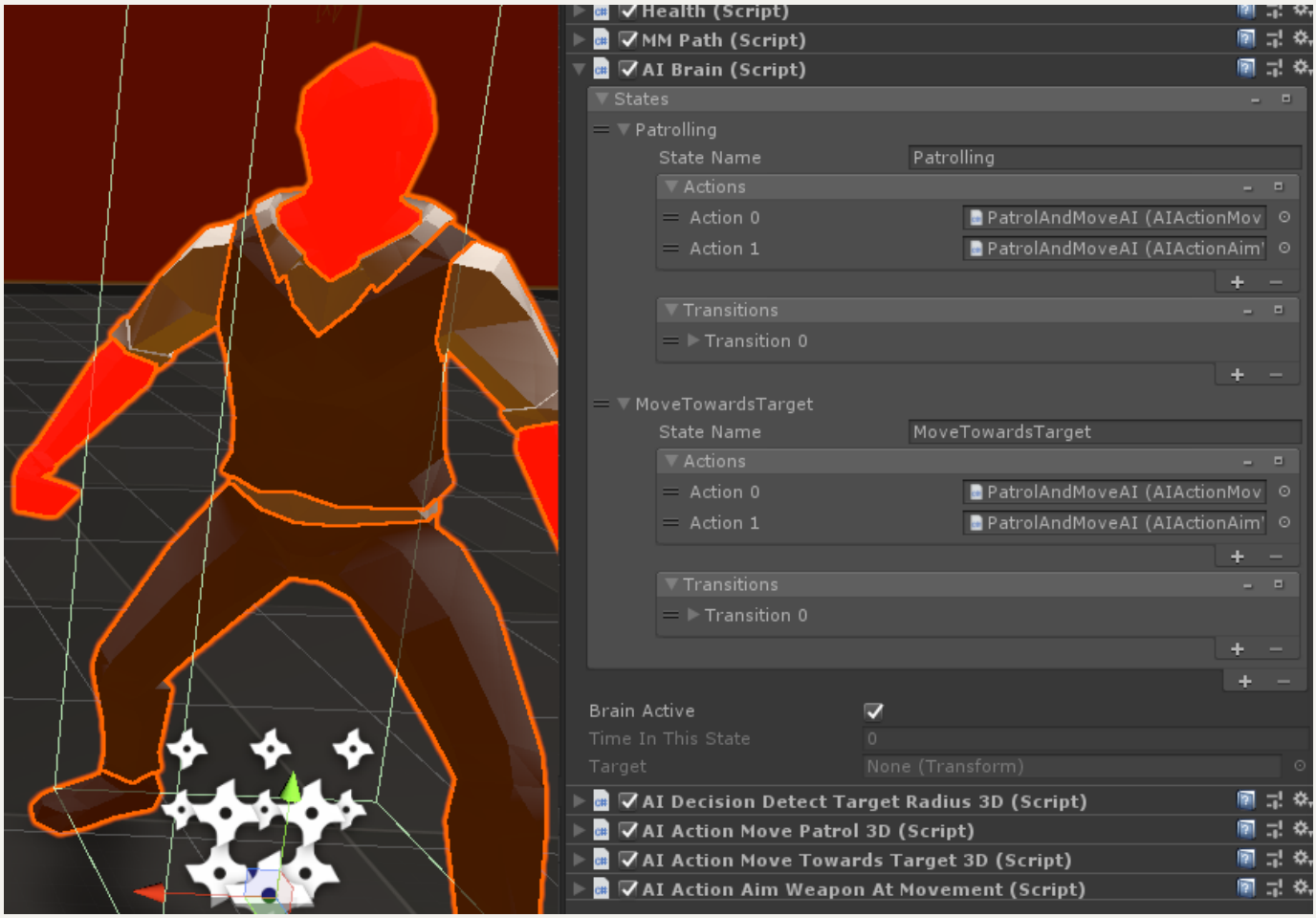
介绍

自上而下引擎提供了先进的人工智能系统，可以帮助您创建各种有趣的模式和态度。有关您可以使用它做什么的简要概述，您可以查看 MinimalAI 演示场景，其中将展示您可以使用该系统创建的一小部分内容。

核心概念

高级 AI 系统基于您可以在 MMTools/AI/ 文件夹中找到的一些核心类：

- **AIState**：状态是一个或多个动作和一个或多个转换的组合。状态的一个例子可以是"*巡逻直到敌人进入范围*"。
- **AIAction**：动作是行为，描述你的角色正在做什么。示例包括巡逻、射击、跳跃等。该引擎带有许多预定义的动作，而且很容易创建自己的动作。
- **AIDecision**：决策是将通过转换评估的组件，每一帧，并将返回 true 或 false。示例包括在某个状态中花费的时间、与目标的距离或区域内的对象检测。
- **AITransition**：转换是一个或多个决策和目标状态的组合，无论这些转换是真还是假。转换的一个例子可能是"*如果敌人进入射程，转换到射击状态*"。
- **AIBrain**：AI 大脑负责根据定义的转换从一种状态转到另一种状态。它基本上只是状态的集合，在这里您可以将所有操作、决策、状态和转换链接在一起。



高级 AI 驱动的角色检查员的示例，包括其大脑、三个动作和一个决定。

所有这些都将放在一个已经设置好的角色之上（查看专用文档了解更多信息）并且可以潜在地利用所有现有的能力。

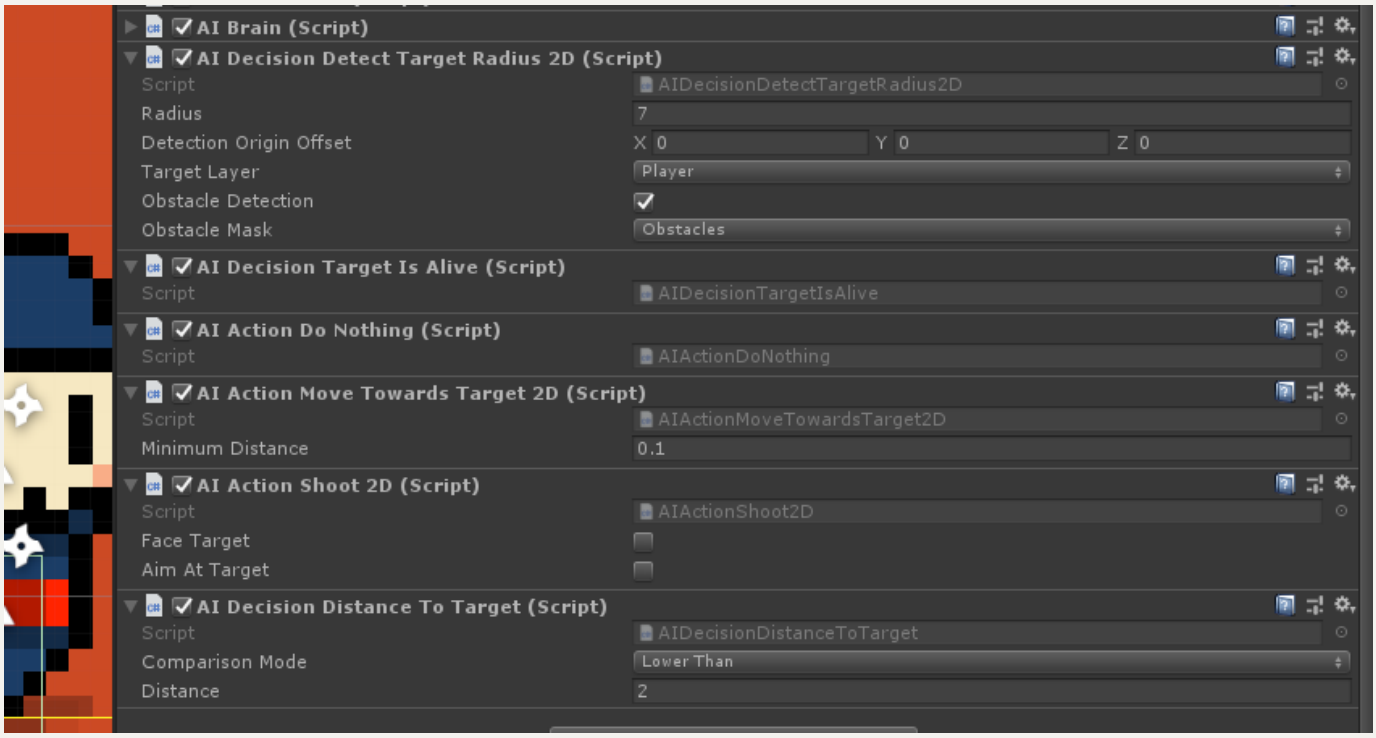
创建高级 AI 角色

我们将重新创建 KoalaDungeon 演示场景中可用的示例角色之一，NinjaSwordMaster。创建基础角色与创建任何角色没有什么不同，所以我们不会在这里再讨论。一旦该基本角色准备就绪，您将需要向其添加一个 AIBrain。

我们希望这个角色有以下行为：在敌人足够近之前什么都不做，如果是这样，就朝它移动，如果足够近就攻击。然后，如果目标超出范围，则回到什么都不做并等待目标。

为此，我们将使用引擎中可用的一些现成操作和决策。我们将添加以下组件：AIDecisionDetectTargetRadius2D、AIDecisionTargetIsAlive、AIActionDoNothing、AIActionMoveTowardsTarget2D、AIActionShoot2D、AIDecisionDistanceToTarget。每一个都带有一个检查器，我们需要在其中设置一些东西：

- **AIDecisionDetectTargetRadius2D**：我们定义我们的半径（示例中为 7），我们的目标层应该是 Player，我们想使用 Obstacles 层作为我们的 Obstacle 蒙版。
- **AIDecisionTargetIsAlive**：我们将使用它来确保我们的目标还活着。无需设置
- **AIActionDoNothing**：这里不需要设置
- **AIActionMoveTowardsTarget2D**：我们将最小距离设置为 0.1，这是角色认为其旅程完成时到目标的距离
- **AIActionShoot2D**：一切都应该取消选中
- **AIDecisionDistanceToTarget**：如果距离小于 2，我们希望此决策返回 true，因此我们将比较模式设置为"小于"，将距离设置为 2。



我们角色的检查员。

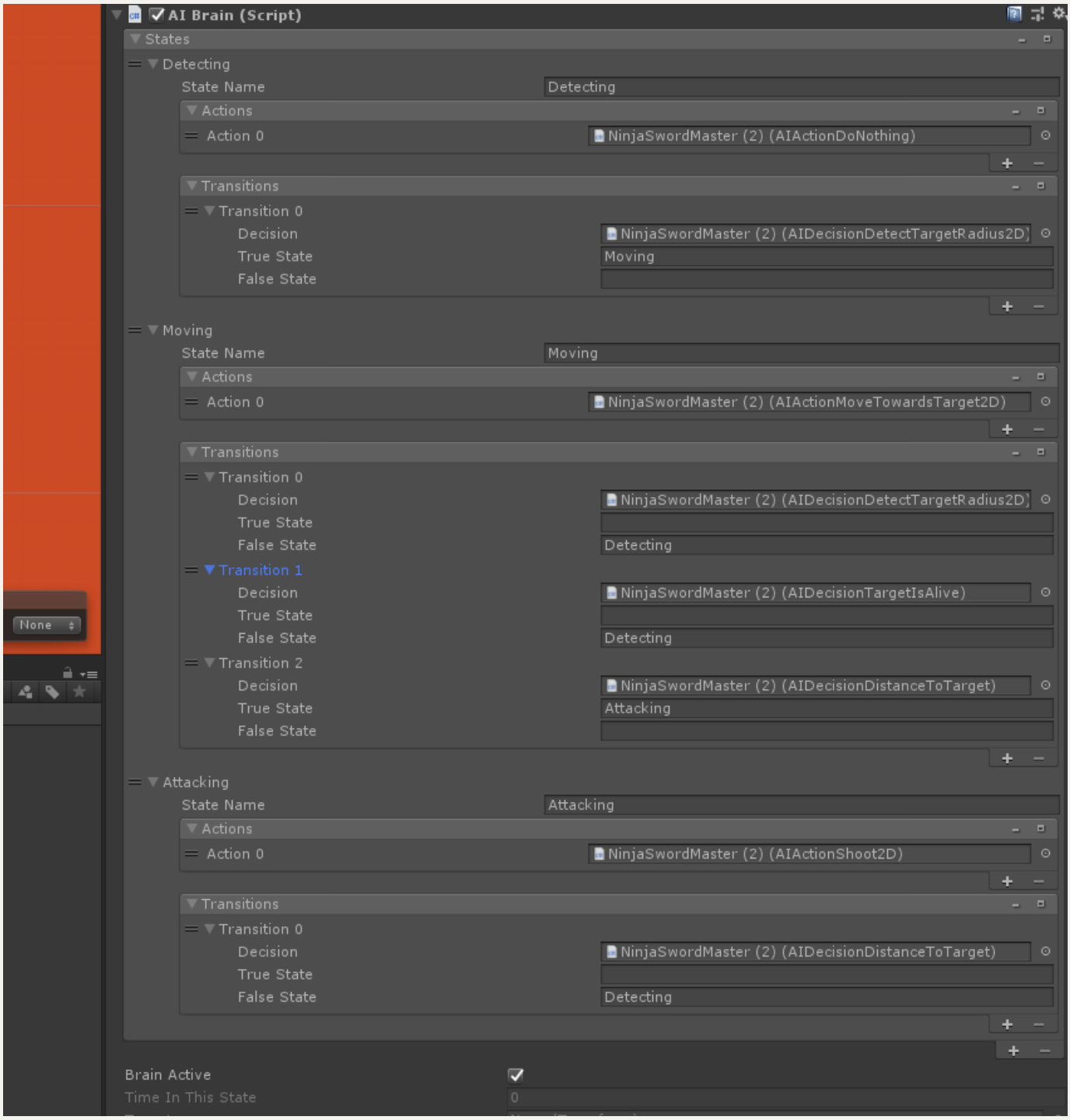
我们的角色现在拥有定义其行为所需的所有行动和决定。剩下要做的就是将所有这些插入它的大脑。如前所述，我们的角色将有两种状态：巡逻和攻击。因此，在其 AIBrain 组件中，我们只需按三下右下角的"+"号即可添加三个状态。我们将第一个状态命名为"检测"，第二个状态为"移动"，第三个状态为"攻击"。

命名我们的州。

现在在 Detecting 状态下，我们将只有一个动作和一个转换，因此我们将通过分别按下各自的"+"号来添加一个动作和一个决策。现在，我们将角色的 AIActionDoNothing 组件拖到 Actions 的数组槽中，并将 AIDecisionDetectTargetRadius2D 拖到 Transitions 的第一个 Decision 槽中。请注意，从 v1.10 开始，您还可以从下拉列表中选择该操作。我们希望在玩家在范围内时切换到移动，因此我们将"移动"放在过渡的*True State*字段中。

我们将以同样的方式处理我们的第二个状态：我们将添加一个动作，这次是三个转换，然后将我们的 AIActionMoveTowardsTarget2D 组件拖到我们正在移动状态的 Action 槽中，然后将我们的 AIDecisionDetectTargetRadius2D 移动到第一个转换的检测槽中，AIDecisionTargetIsAlive 进入第二个转换的槽，最后 AIDecisionDistanceToTarget 进入第三个转换的槽。当目标不再在范围内时，我们要返回到 Detecting，因此我们将 Detecting 置于第一个转换的 false 状态，并且我们还希望在目标死亡时返回到检测，因此我们将 Detecting在我们的第二个转换的错误状态槽中。如果到目标的距离足够小，我们想要攻击，所以我们会将 Attacking 放在第三个转换的真实状态槽中。

在我们的最后一个状态 Attacking 中，我们将添加一个动作和一个转换。然后我们想将我们的 AIActionShoot2D 拖到 Action 槽中，并将 AIDecisionDistanceToTarget 拖到过渡的决策槽中。如果距离太高，我们想回到 Detecting，所以我们将 Detecting 放在 false 状态槽中。



我们的大脑都设置好了！

为了确保我们的大脑正确设置，我们可以"阅读"它。从上到下，我们有一个角色什么都不做，等待目标进入其半径范围内。然后它开始移动，除非目标已经死亡或太远/看不见。但是，如果目标足够近，它就会发起攻击，然后返回检测目标是否足够远。

恭喜，您已经完成了使用自上而下引擎可能"最难"的事情，并且您现在知道如何创建各种角色行为。你当然可以在一个状态下有多个动作（例如跑步和跳跃），在一个状态下有多个转换，所以你真的可以想出复杂的东西。

行动

该引擎带有所有这些预定义的操作，随时可以在您自己的角色中使用：

- **AIActionAimObject**：将让您将对象的轴（通常是向右或向前）瞄准角色的移动方向或其瞄准方向。
- **AIActionAimWeaponAtMovement**：将强制武器瞄准移动方向。
- **AIActionChangeWeapon**：用于强制您的角色切换到另一种武器。只需将武器预制件拖入其 NewWeapon 插槽即可。
- **AIActionCrouchStart**：让你的角色蹲下
- **AIActionCrouchStop**：让你的角色停止蹲伏
- **AIActionDoNothing**：顾名思义，什么都不做。就在那里等着。
- **AIActionMMFeedbacks**：让您播放绑定到其检查器中的任何 MMFeedbacks
- **AIActionFaceTowardsTarget2D**：这个 AI 动作可以让你改变一个 AI 的 CharacterOrientation2D 的面向方向，以及它在 KoalaDungeon 演示场景中的一个例子
- **AIActionMoveAwayFromTarget2D/3D**：使角色向与目标相反的方向移动
- **AIActionMovePatrol2D/3D**：将让您的角色在 MMPath 组件中定义的一组节点中巡逻。确保将 MMPath 的循环选项设置为 Loop 或 BackAndForth（在 OnlyOnce 模式下不能沿着路径巡逻，它必须是连续的）。
- **AIActionInvertPatrolDirection**：让您反转目标 2D 或 3D 巡逻的方向
- **AIActionMoveRandomly2D/3D**：使角色随机移动，直到它在其路径中找到障碍物，在这种情况下它会随机选择一个新方向
- **AIActionMoveRandomlyGrid**：在网格上随机移动，无论是 2D 还是 3D
- **AIActionMoveTowardsTarget2D/3D**：指示 CharacterHorizontalMovement 能力向目标方向移动。
- **AIActionPathfinderToPatrol3D**：将使用 Pathfinding3D 能力移回它的最后一个巡逻点
- **AIActionPathfinderToTarget3D**：如果可以找到路径，将使用 Pathfinding3D 能力移动到目标
- **AIActionReload**：使代理重新加载其当前武器
- **AIActionRotateConeOfVision2D**：将这个 AI 的 ConeOfVision2D 旋转到 AI 的运动或其武器瞄准方向
- **AIActionRotateTowardsTarget2D/3D**：将使 CharacterOrientation3D 能力将角色旋转到大脑的目标
- **AIActionRunStart**：将导致代理开始运行（需要 CharacterRun 能力）
- **AIActionRunStop**：将导致代理停止运行（需要 CharacterRun 能力）
- **AIActionSetLastKnownPositionAsTarget**：将目标的最后已知位置设置为新目标

- **AIActionShoot2D/3D**：使用当前装备的武器进行射击。如果你的武器处于自动模式，会一直射击直到你退出状态，并且在半自动模式下只会射击一次。您可以选择让角色面对（左/右）目标，然后瞄准它（如果武器有 WeaponAim 组件），或者选择不同的原点来定义如何计算瞄准方向。
- **AIActionSwapBrain**：让你为另一个角色的当前大脑大脑。例如，对于 Boss 阶段之类的事情很有用
- **AIActionUnityEvents**：允许您触发绑定到其检查器的任何 Unity 事件

就像引擎中的其他一切一样，我们鼓励您创建自己的。让我们看一下 Action 的代码，看看它是如何工作的：

```
public class AIActionJump : AIAction
{
    public int NumberOfJumps = 1;
    protected CharacterJump _characterJump;

    protected int _numberOfJumps = 0;

    protected override void Initialization()
    {
        _characterJump = this.gameObject.GetComponent<CharacterJump>
();
    }

    public override void PerformAction()
    {
        Jump();
    }

    protected virtual void Jump()
    {
        if (_numberOfJumps < NumberOfJumps)
        {
            _characterJump.JumpStart();
            _numberOfJumps++;
        }
    }

    public override void OnEnterState()
```

```
    {
        base.OnEnterState();
        _numberOfJumps = 0;
    }
}
```

如您所见，此类覆盖了一些方法：

- **Initialization**，其中我们将做任何我们需要做的事情来初始化我们的 Action，在这种情况下，我们存储 CharacterJump 能力以备将来使用。
- **PerformAction**：每当我们的角色处于这个动作所处的状态时，都会调用它。在这种情况下，我们将只调用我们的 Jump 方法，如果我们的条件满足，它将依次调用 CharacterJump 能力的 JumpStart 方法。
- **OnEnterState**：每次我们返回包含此 Action 的状态时，我们都希望重置当前的跳转次数。

决定

- **AIDecisionDetectTargetConeOfVision2D/3D**：使用视野来确定目标是否在视野中。如果是这种情况，则返回 true。
- **AIDecisionDetectTargetLine2D**：如果其 TargetLayer 图层蒙版上的任何对象进入其视线，则返回 true。它还会将大脑的目标设置为该对象。您可以选择让它处于光线模式，在这种情况下，它的视线将是一条实际的线（光线投射），或者让它更宽（在这种情况下，它将使用球面投射）。您还可以指定射线原点的偏移量，以及阻挡它的障碍层蒙版。
- **AIDecisionDetectTargetLine3D**：允许您在 3D 上下文中使用光线投射或箱线投射检测目标
- **AIDecisionTargetRadius2D/3D**：如果 TargetLayer 图层蒙版上的对象在其指定半径内，则返回 true，否则返回 false。它还会将大脑的目标设置为该对象。
- **AIDecisionDistanceToTarget**：如果当前大脑的目标在指定范围内，则返回 true，否则返回 false。
- **AIDecisionGrounded** : 如果字符接地，则返回 true，否则返回 false。
- **AIDecisionHealth**：如果满足指定的健康条件，将返回 true。您可以让它低于、严格低于、等于、高于或严格高于指定值。
- **AIDecisionHit**：如果角色在此帧中被击中，或在达到指定的击中次数后，则返回 true。

- **AIDecisionLineOfSightToTarget2D/3D** : 如果一条线可以从指定的偏移量追踪到目标并且没有碰到指定的层，则返回 true
- **AIDecisionNextFrame** : 当进入这个 Decision 所在的状态时将返回 true。
- **AIDecisionRandom**：如果结果低于或等于其检查器中指定的赔率值，则掷骰子并返回 true
- **AIDecisionReloadNeeded**：如果在评估它的帧需要重新加载，则返回 true
- **AIDecisionTargetFacingAI2D**：如果大脑的当前目标面对这个角色，将返回真。是的，它是 Ghosts 特有的。但是，嘿，现在您也可以使用它了！
- **AIDecisionTargetIsAlive** : 如果目标还活着，将返回真
- **AIDecisionTargetIsNull** : 如果目标为空则返回 true
- **AIDecisionTimeInState**：将在指定的持续时间（以秒为单位）过去后返回 true，因为 Brain 一直处于此决策所处的状态。在完成其他事情 X 秒后使用它来做某事。
- **AIDecisionTimeSinceStart**：将在加载关卡后经过指定的持续时间（以秒为单位）后返回 true。

就像操作一样，创建自己的决策非常容易。

组织 AI 组件

在构建复杂角色时，角色的检查器组件列表可能会变得非常庞大，包括代理的核心类（角色、TopDownController 等）、能力以及大脑、动作和决策。幸运的是，v1.10 引入了将这些能力拆分到多个节点的能力，对于大脑、行动和决策也是如此。

您可以在 Loft3D 演示场景中看到一个角色设置示例，使用**PatrolSeekAndDestroyAI**预制件。在它的顶层，你会发现它的核心类。然后，在嵌套在其下的空游戏对象上，您将找到一个具有其能力 (Abilities) 的节点和一个具有其大脑 (AIBrain) 的节点。为了进一步推动事情，如果需要，还可以将操作和决策拆分到更多节点。此设置唯一重要的事情是确保您已将 Brain 绑定到 Character 的检查器，在其**CharacterBrain**字段中。如果你不这样做，它会自动在它自己的级别上寻找一个。对于能力，您可以在文档的专用部分中了解更多相关信息。