蚂蚁的人工智能。指南和文档

由安东·卡尔洛夫（ant-karlov.ru）创建

内容

概述

蚂蚁人工智能是一个基于GOAP（面向目标的行动计划）的人工智能创建库。该算法的特点是，在开发人工智能时，不需要创建一个严格的分支树来进行决策。所有可能的操作都是分开设置，行动计划是基于当前状态和目标动态构建的。

该算法的一个优点是，游戏世界或对象的状态的描述由一个逻辑变量列表给出，其中只能有两个值：真或假。例如：“角色有武器：真”，“角色有弹药：假”——根据这些条件，规划者可以决定角色应该寻找弹药。

你可以在这里阅读更多关于GOAP理论的信息：面向目标的行动规划(GOAP)蚂蚁人工智能的关键特点：

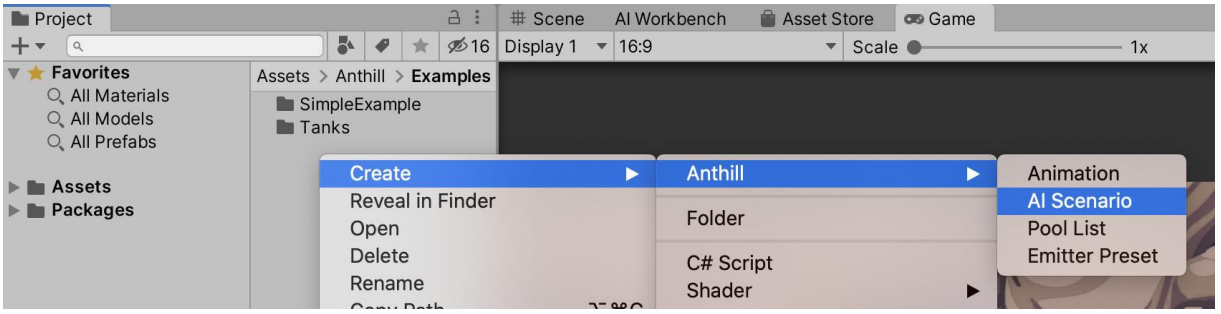
- 可以创建有设置条件、行动和目标的人工智能场景；
- 有可能对多个游戏实体同时使用相同的行为场景；
- 使用方便的编辑器来编辑和调试行为场景；
- 简单的标准人工智能规划师的实现；
- 自定义人工智能规划器，你可以传输一个条件列表，只得到输出的操作列表；
- 适用于任何类型的视图：3D，2D，自上而下，等距e.t.c.；
- 与任何平台一起工作；
- 丰富的例子。

人工智能场景

场景是一个脚本表对象，它包含计划者工作的所有必要数据。您可以将一个场景的实例分配给必须遵循相同策略的不同单元。例如，如果游戏中存在交付单元，那么所有交付实体都可以使用一个场景。

场景创建

在“项目”窗口中，打开上下文菜单，选择“创建”>”选集“>”“AI场景”。之后，将创建一个脚本表对象，它将包含计划者工作所需的所有必要数据。

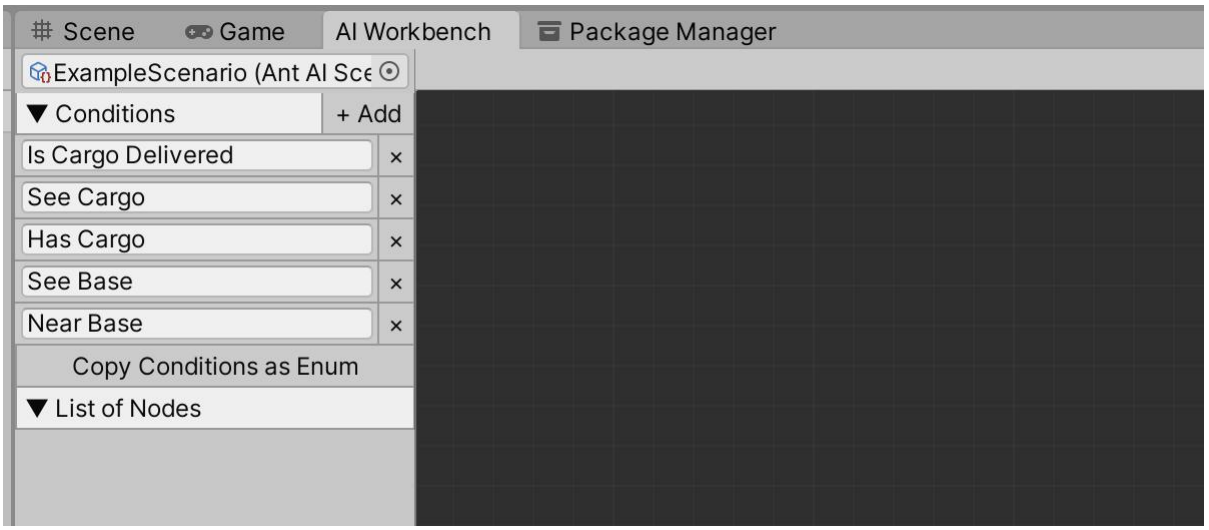


选择已创建的脚本表对象，并在“检查器”窗口中单击“打开AI工作台”。以一种方便您的方式安排“AI工作台”窗口。

条件

条件是一个逻辑变量的列表，它只能取两个值：“真”或“假”。条件是创建场景的基础，并用于描述“世界状态”，以及描述操作的输入和输出数据。

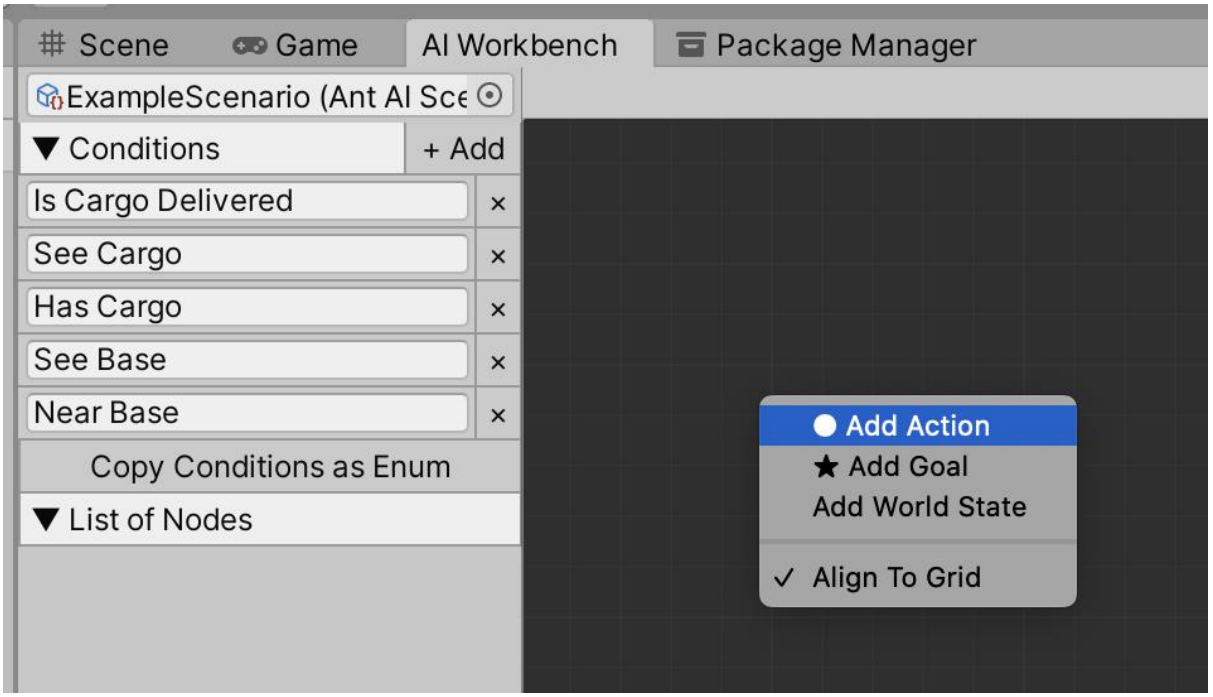
例如，添加以下条件：“货物是否已交付”、“查看货物”、“已交付货物”、“查看基地”、“接近基地”——此条件列表将允许我们为交付单元创建一个简单的场景。



行动

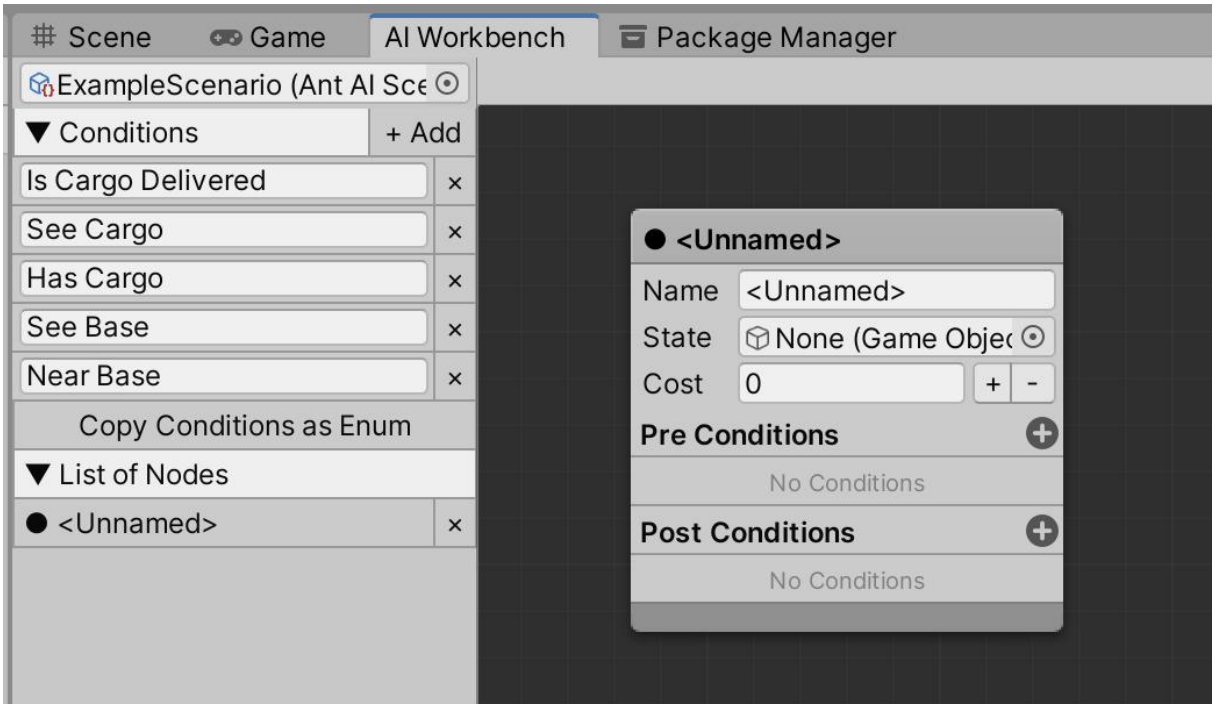
行动是我们可以执行的，例如：“装载货物”、“交付货物”、“卸载货物”。

要创建新操作，请通过点击编辑器的背景打开上下文菜单，并选择“添加操作”。



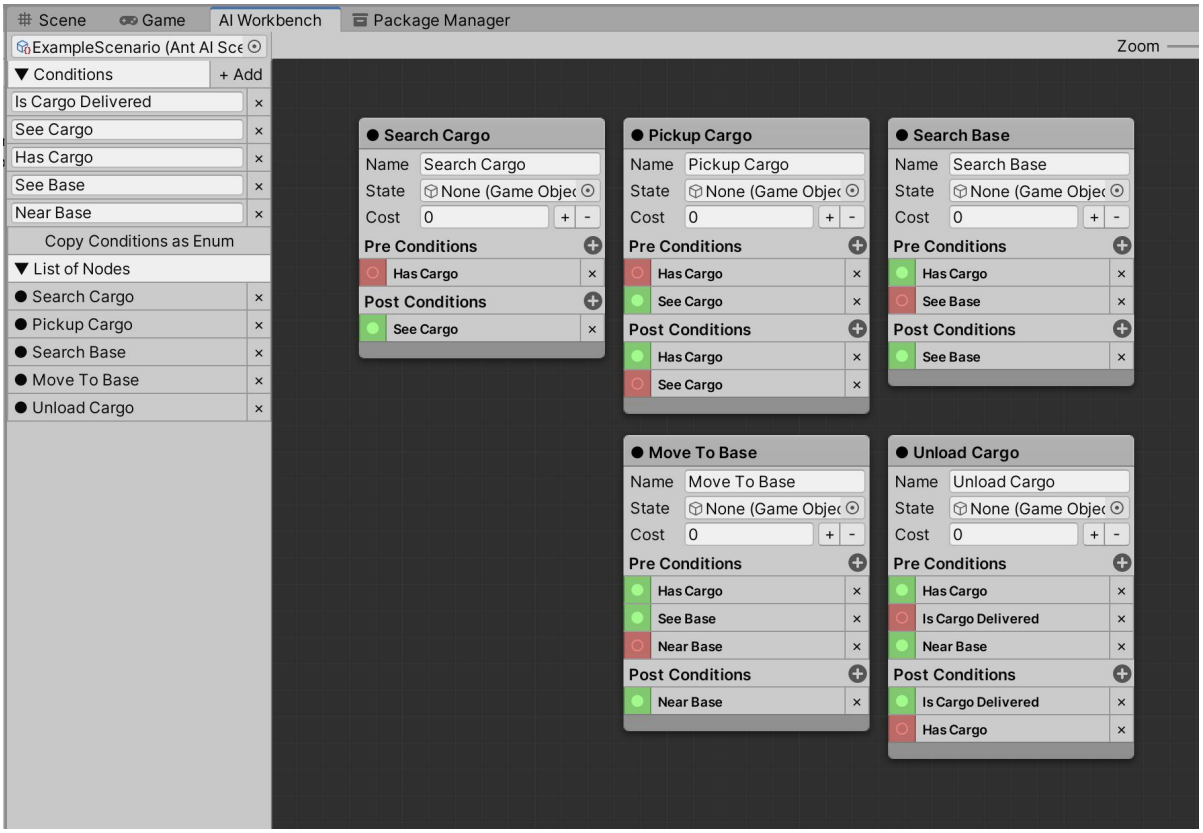
添加该操作后，在编辑器窗口中将显示一张新卡。您可以在编辑器窗口中自由地重新排列卡片。

要删除一张卡，请选择它并打开其上下文菜单，然后选择“删除”项。



应指定操作名称，应附上状态脚本（可选），应注明使用成本，以及必须处于和平状态的条件列表（前条件），以便计划者可以使用此操作，以及执行此操作后将更改的条件列表（后条件）。

为下面的示例创建一些操作，如下屏幕截图所示。



如果你仔细考虑所创建的卡片，你可以找到逻辑：如果该单位没有货物，那么它将开始寻找它。如果一个单位看到货物，那么它会接近并装载，之后它会寻找基地，当它看到时，它会接近并卸载货物。

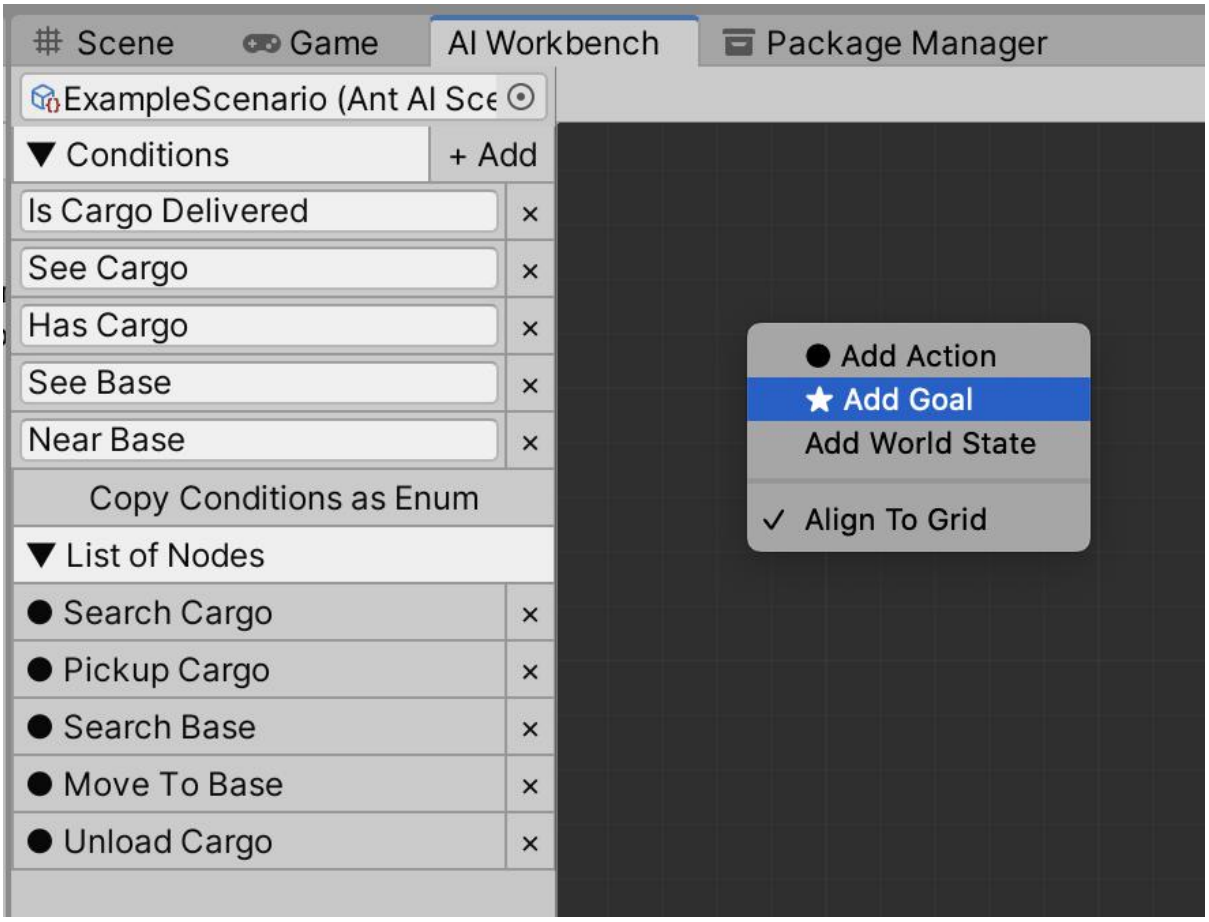
但如果计划者不知道他的计划，他将无法制定行动计划

球门

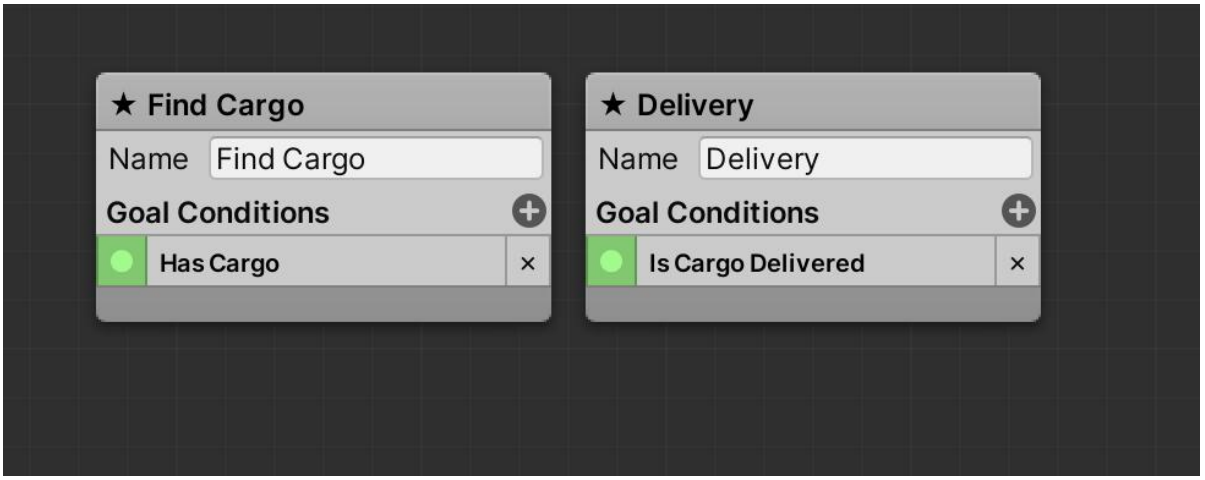
目标

目标是一种特殊的卡片，它可以让计划者知道它应该实现什么。如果在我们的场景列表中没有目标，那么计划者将无法构建一个行动计划。

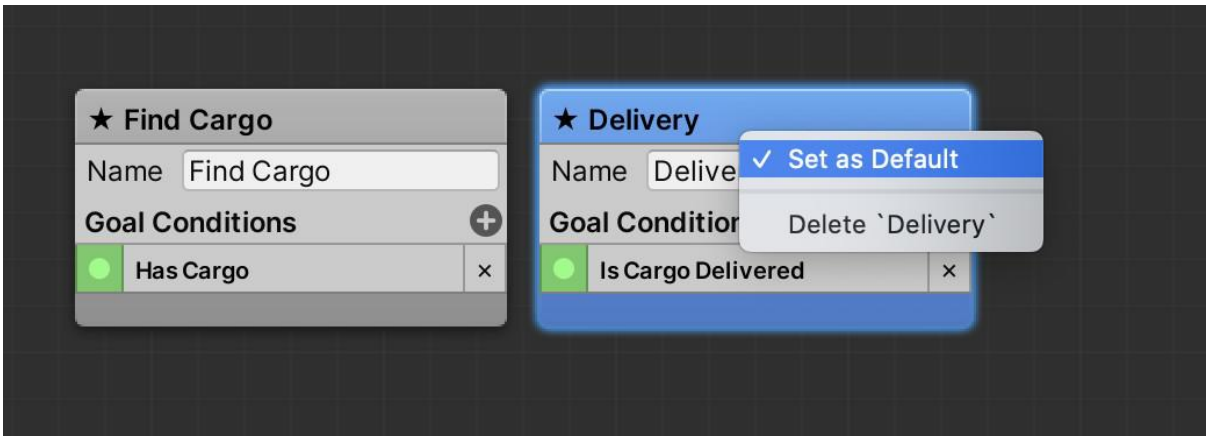
要创建目标，请调用编辑器窗口中的上下文菜单，并选择“添加目标”。



为我们的场景创建两个目标，如下的屏幕截图所示。将目标“查找货物”作为默认值——这意味着此目标在默认情况下将处于活动状态。如果有必要，如果需要更改计划者的策略，您可以从代码中切换目标。



要将“查找货物”目标设置为默认目标，请选择所需的卡，并在其上下文菜单中选择“设置为默认”菜单项。默认目标将显示为蓝色。



在目标卡“找到货物”中，我们指出“有货物”应该等于“真的”，所以规划者将试图制定一个行动计划，以实现这个条件。

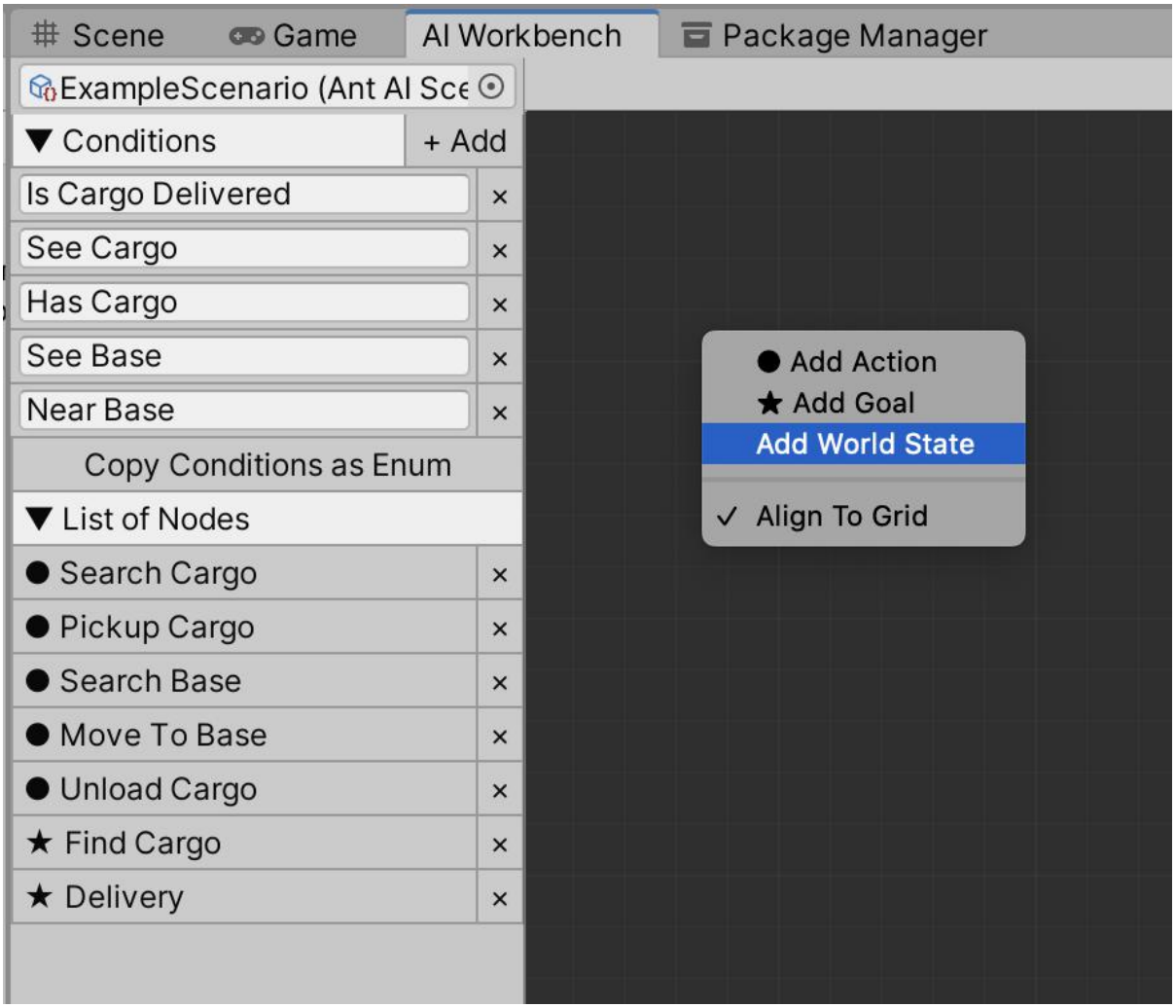
世界国家

世界状态是一个描述游戏对象当前状态的条件列表。世界状态允许规划者了解当前的情况，并在此基础上，计划者构建一个行动计划，改变世界状态到期望的状态。

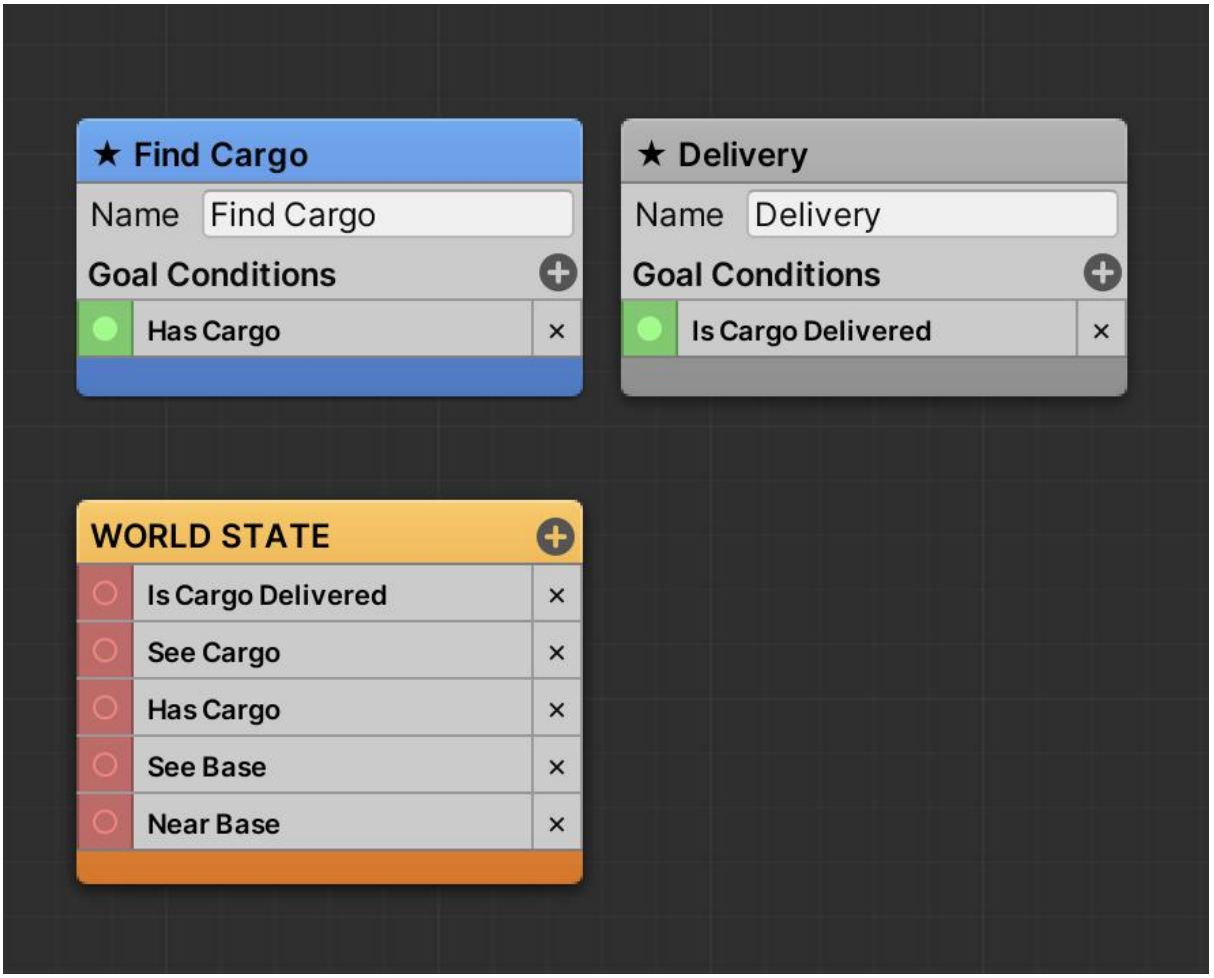
世界的状态并不是对游戏世界的完整描述，它通常是一个特定的游戏单元的世界状态。每个单元及其场景可能有一组不同的条件，只描述它们的世界状态。

世界状态是建立在该单元传感器的基础上的：视觉、触摸、听力、库存内容和e.t.c.单位的世界状况将如何建立，这取决于你。

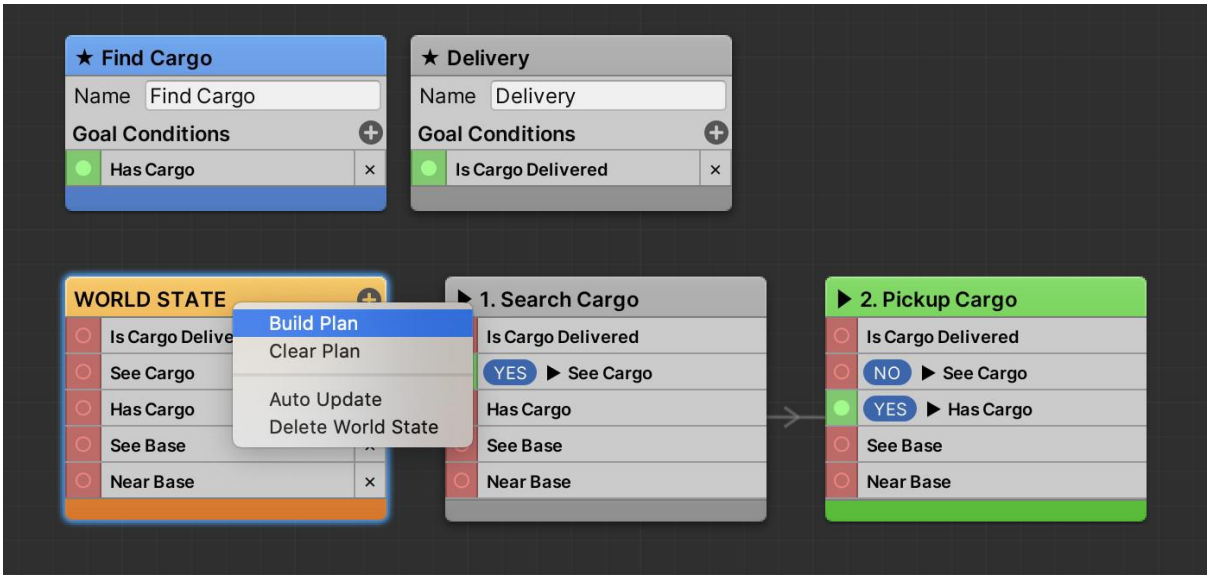
要创建用于测试场景的世界测试状态，请调用上下文菜单并选择菜单项“添加世界状态”。



向世界状态卡添加所有或几种条件。我们的测试条件应该如下图所示。



要从当前状态构建计划，请选择“世界状态”卡，并在上下文菜单中选择“构建计划”菜单项。或者，您可以启用自动更新选项，以在条件改变时自动重建计划。



我们设定了“找到货物”的当前目标——计划者的任务是将条件“有货物”，设置为“真”。规划者建立了一个包含两个行动的计划：“搜索货物”和“提货货物”——因为这些行动的顺序执行将把指定的条件更改为我们需要的值。

在每个操作中更改的条件用粗体标记，并有附加的标签“YES”或“NO”——这取决于值的变化方式。

最后一次行动的绿色意味着正是这一行动取得了成功，并实现了目标。在不可能实现目标的情况下，最后一张牌可能是红色的，这意味着规划者无法找到实现目标的解决方案。但即使是一个失败的计划也可能很有用，因为它可以包含世界状态可以改变的行动，而对该计划的进一步更新将导致成功。

您可以通过打开或关闭不同的条件并重建计划来实验世界的状态，看看场景在不同情况下的行为。

[重要]在场景中创建的世界状态仅用于测试，并且不影响在游戏中使用此场景的规划师的工作。

实施

这个库提供了两种类型的实现：标准实现和自定义实现。

标准实现包括一组组件，它们将游戏对象与计划者、场景以及实现操作的工作连接起来。

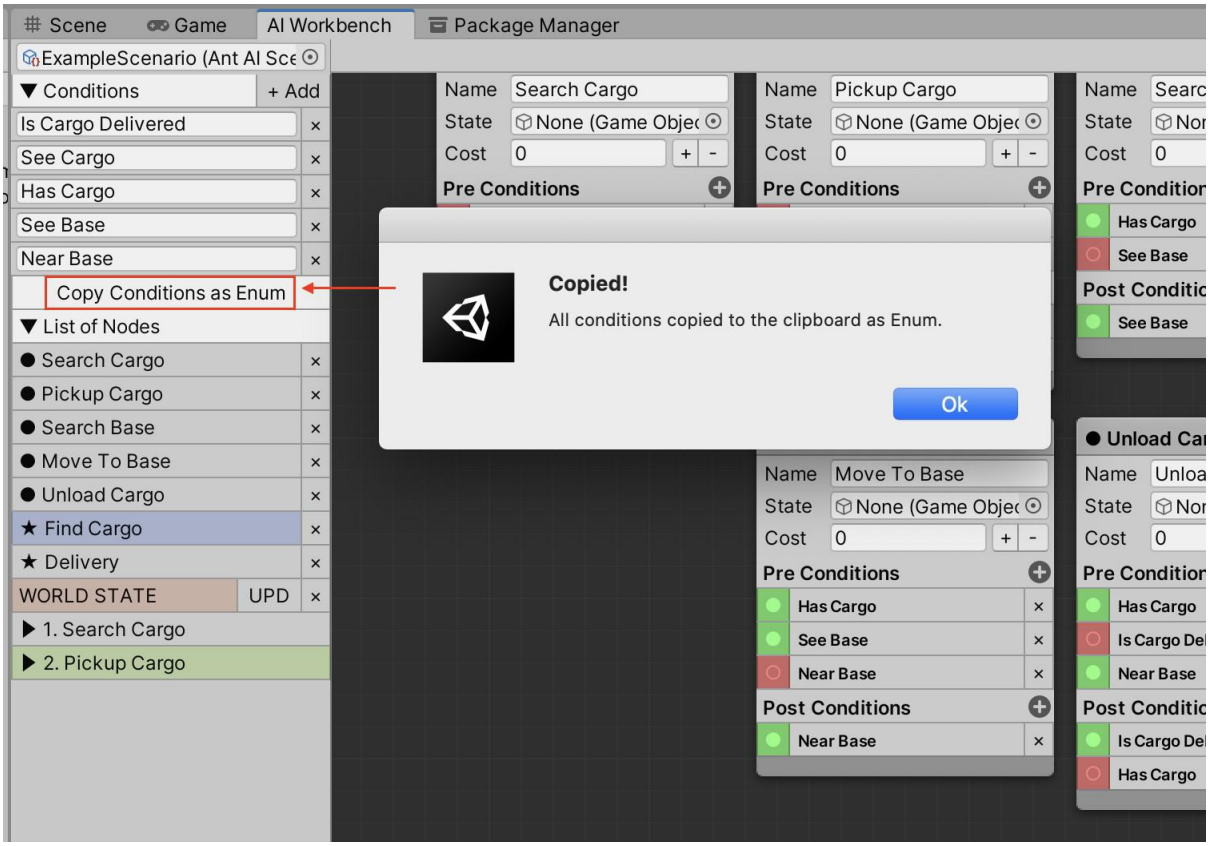
如果出于某种原因，您对规划器的标准实现不满意，并且/或者您知道如何更好地为项目实现状态/操作的工作——您可以使用自定义实现，在其中使用规划器的“原始”数据。

标准的实现

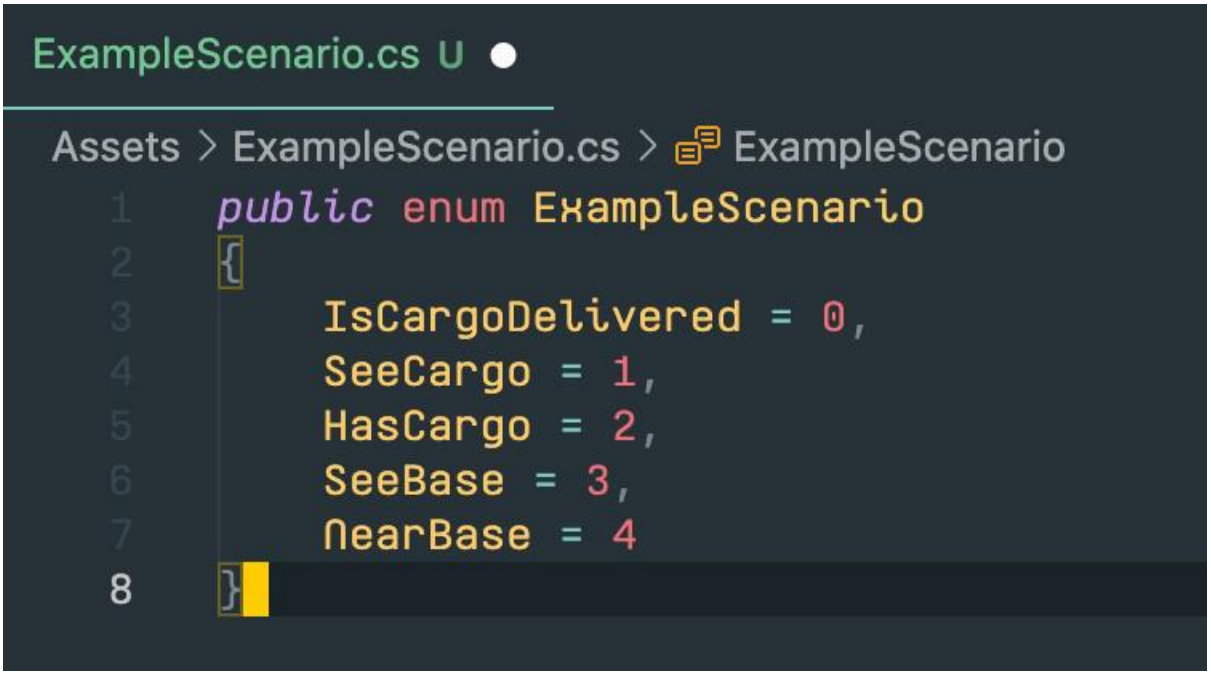
标准实现允许您通过“AI工作台”将脚本绑定到对象的操作（状态）。在本节中，我们将介绍如何实现这一点的最简单的示例。

（此示例的源代码可以在该文件夹中找到：
选集/示例/默认使用）

首先，按AI工作台中的“复制条件为枚举”按钮。



这是将所有条件作为枚举的形式复制到剪贴板中。创建新的*。cs文件，并粘贴到从剪贴板复制的代码：

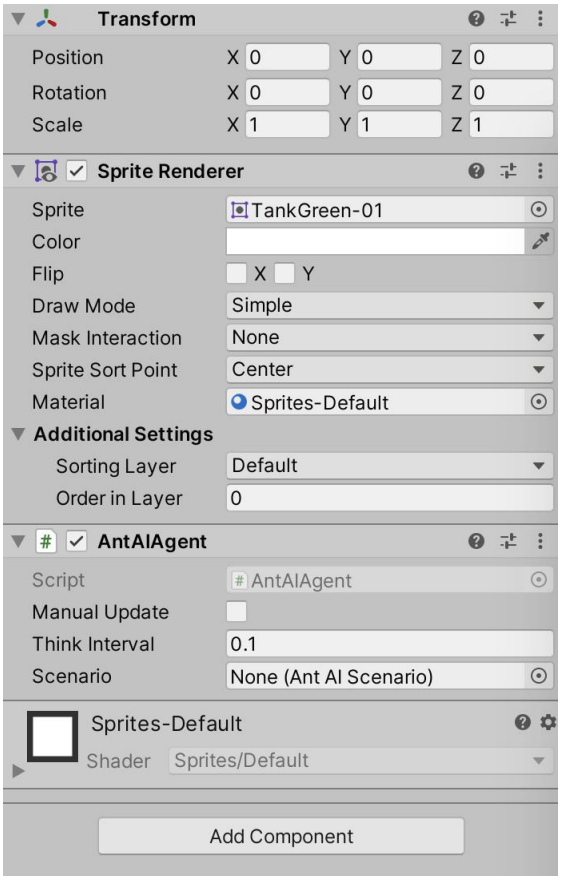


这允许您为代码中的条件放弃字符串名，这就消除了犯错误的可能性，并且您将更容易更新和添加条件，如果

您已经删除或更改了它们，因为更新后所有内容都会突出显示，并且您可以在枚举更新后轻松更正代码中的更改。

1. 创建一个新的场景，并向其添加任何将作为游戏单元的对象。

2. **将组件附加到单元对象：属性渲染器和天线代理，如下图所示。**



AntAIAgent是AntAI库的一个标准组件，它实现了游戏对象、场景和计划者之间的关系。

3. 在AntAIAgent组件的场景字段中，设置我们在本文档的前一节中创建的场景（脚本表对象）。

“手动更新”属性允许禁止自动更新计划，在这种情况下，您需要手动调用执行（删除时间、时间表）方法，以便AntAIAgent来更新计划。这个功能将是有用的，例如，如果你的游戏是回合制的，没有必要重建计划。

“思考间隔”属性允许您设置计划更新间隔。制定计划的过程并不是很复杂，但在制定计划之前，AntAIAgent呼吁该单元的传感器(ISensor)来更新世界的状态。如果

传感器的实现是复杂的，因此值得选择最优的延迟值，以便不经常重建计划。

4. 现在我们将创建脚本UnitControl.cs——这个脚本将负责该单元的库存。将此脚本添加到单元对象中。

```
//UnitControl.cs--使用联
合引擎;

公共单元控制：单一行为
{
    公共汽车，的货物
}
```

5. 接下来，我们需要创建一个脚本UnitSense.cs来实现该单元的所有传感器。这个脚本应该实现问题接口，以便AntAIAgent可以自动找到这个感觉器官，并在每次需要时为其调用集合条件()方法。此脚本也应该附加到单元对象上。

```
//UnitSense.cs——使用联
合引擎；使用国歌。人工智
能；使用Antil。使用；

公共集体意识：单一行为，问题
{
    私人单位控制_control；私人转换_t；
    私人转换_base；私人转换
    _cargo；

    私有空唤醒()
    {
        _control=获取组件<单元控制>()；
        _t=GetComponent<Transform>()；

        //保存机器人基础的参考。vargo=游戏对象。查
        找（“基础”）；
        调试。断言(go！=null，“场景中不存在基本对象！”)；
        _base=去了。Get组件<变换>()；

        //保存有关货物的参考资料。转到=游戏对
        象。查找（“Cargo”）；
        调试。断言(go！=null，“现场不存在货物对象！”)；
        _cargo=去了。Get组件<变换>()；
    }

    ///<summary>
    ///这种方法将被称为自动计算，每次当
```

```
    ///AntAIAgent决定更新该计划。您应该附加此脚本
    ///到相同的对象，其中是AntAIAgent。
    ///</summary>
    公共无效收集条件（代理，世界状态）
    {
        aWorldState.开始更新(aAgent。规划师）；
        {
            aWorldState.设置（示例场景。；一个世界上的国家。设置（示例场景。
            SeeCargo，IsSeeCargo())；aWorldState.设置（示例场景。哈斯卡戈，_control。
            HasCargo)；aWorldState.设置（示例场景。SeeBase，IsSeeBase())；
            aWorldState.设置（示例场景。NearBase，IsNearBase())；
        }
        aWorldState.EndUpdate()；
    }

    私人boolIsSeeCargo()
    {
        返回（AntMath。距离(_t。位置，_cargo。位置）<1.0f）；
    }

    私人boolIsSeeBase()
    {
        返回（AntMath。距离(_t。位置，_base。位置）<1.5f）；
    }

    私人boolIsNearBase()
    {
        返回（AntMath。距离(_t。位置，_base。位置）<0.1f）；
```

6. 现在，请创建一个脚本，以实现“搜索货物”的操作，并命名为该脚本SearchCargoState.cs，并添加以下代码：

```
//SearchCargoState.cs--使用联
合引擎；
使用选集。人工智能；使用
Antil。使用；

公共班级搜索
{
    私人浮速度=2.0f；

    私有转换_t；私人向量
    3_targetPos；私人浮动
    _targetAngle；

    公共覆盖void创建（游戏对象）
    {
        t=aGameObject.Get组件<变换>()；
```

```
    公共覆盖无效Enter()
    {
        //在地图上搜索货物。
        vargo=游戏对象。查找（“货物”）；如果
        (go！=null）
        {
            //保存货物的位置，以便移动。
            _targetPos=去了。变换位置；

            //Calc目标角度。
            _targetAngle=AntMath.AngleDeg(_t.位置，_targetPos)；
            _t.旋转=四元数。欧拉(_t。旋转x，_t.旋转y，
_targetAngle)；
        }
        其他的
        {
            调试。日志(“找不到卡戈！”)；完成
            ()；
        }
    }

    公共覆盖无效执行（浮动、删除时间、浮动、时间尺度）
    {
        //移动到货物上。
        varpos=_t.position；
        波斯。x+=速度**数学。代码(_targetAngle)；pos。+=速度*数学。罪
        （_targetAngle）；
        _t.位置=pos；

        //检查到货物的距离。
        如果(AntMath。Distance(pos，_targetPos)<=0.2f)
        {
            //我们到达了！
            //当前操作已完成。完成()；
        }
```

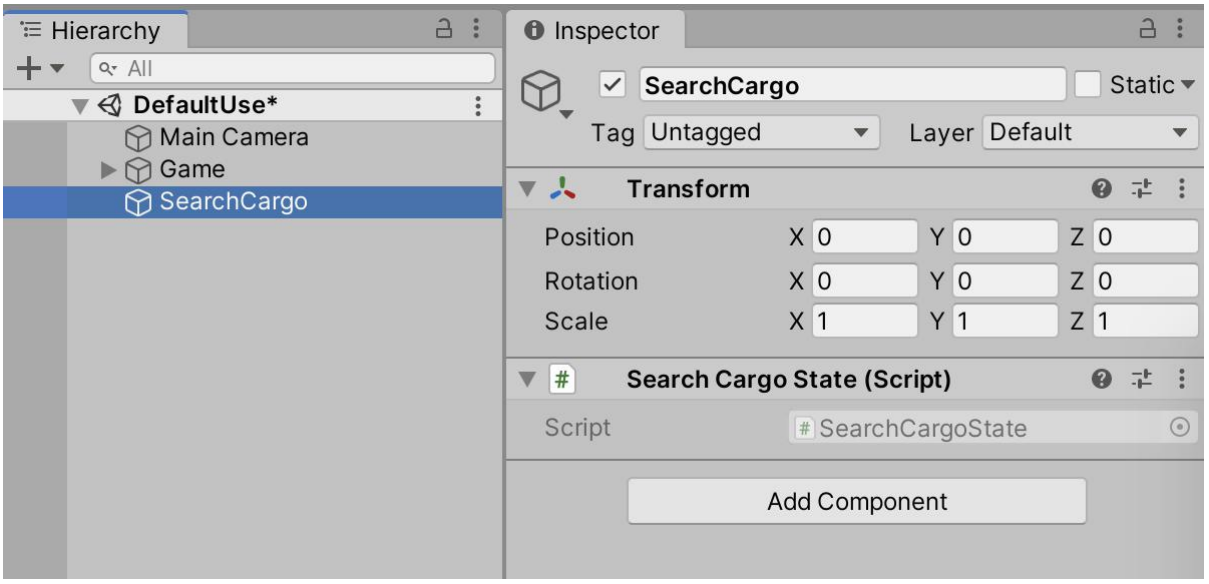
请注意，此脚本继承自AntAIState——这是一个实现处理状态所需功能的抽象类。每个状态脚本都必须从该类中继承这些脚本。

AntAIState有以下有用的方法，您可以覆盖它们来实现您所需要的行为：

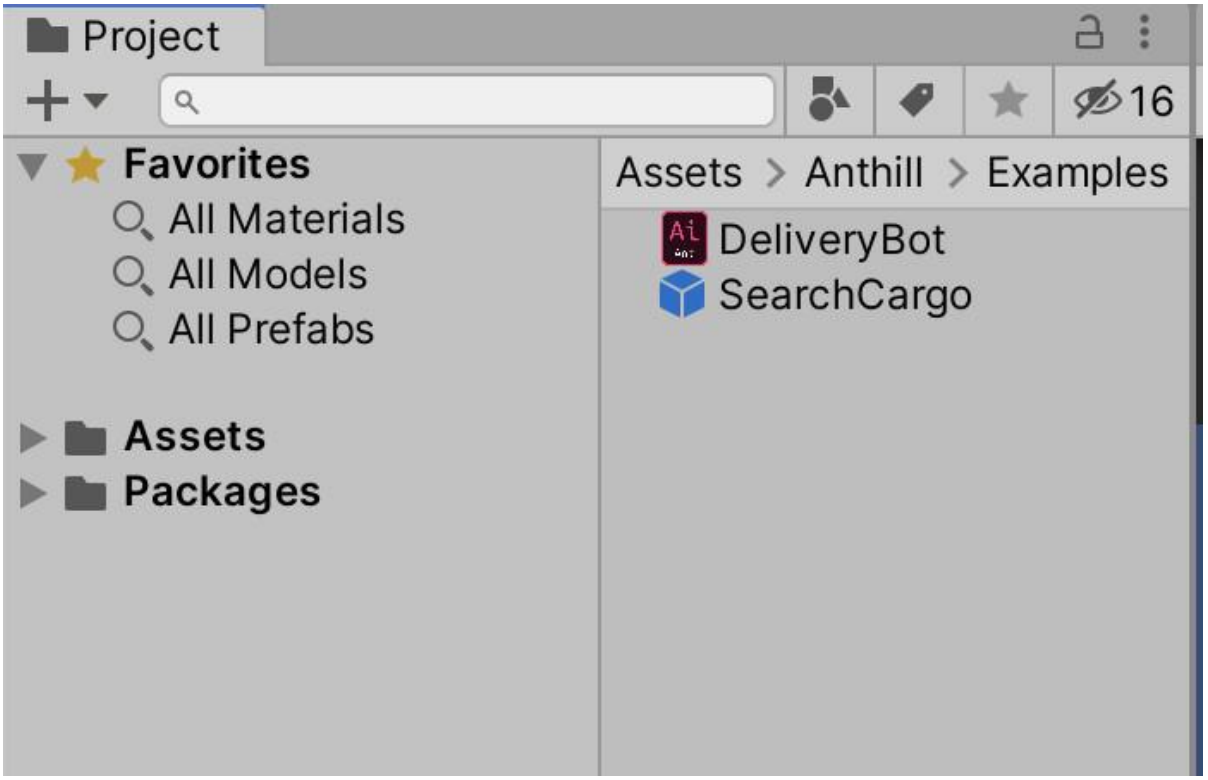
无效创建（游戏对象a游戏对象）—在游戏对象初始化并创建当前状态时调用一次。作为一个参数，一个游戏对象被传递到AntAIAgent上，这使您有机会获得游戏对象的所有必要组件，以便进一步使用它们。

Enter()-在激活之前调用一次。
无效执行（浮动时间、浮动时间）—当此操作活动时，每个游戏勾用。
空Exit()-在停用此操作之前调用一次。

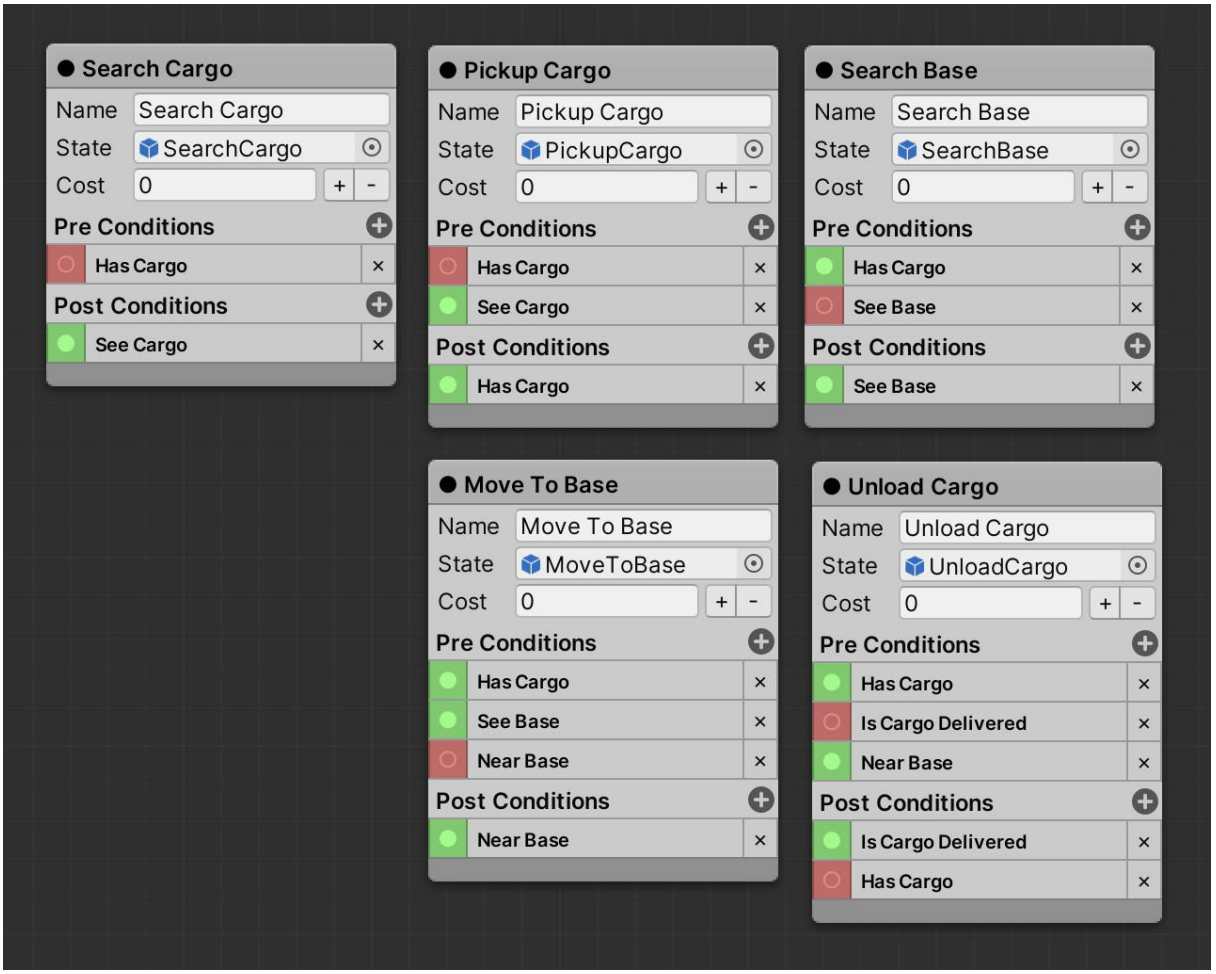
7. 要将创建的动作脚本添加到我们的场景中，您需要将其绑定到舞台上的一个空对象。在“层次结构”窗口中的“场景”上创建一个新对象，并向其添加SearchCargoScript.cs脚本，如下图所示。



8. 通过将创建的对象拖到脚本所在的同一文件夹中的“项目”窗口，将创建的对象保存为预览，如下图所示。

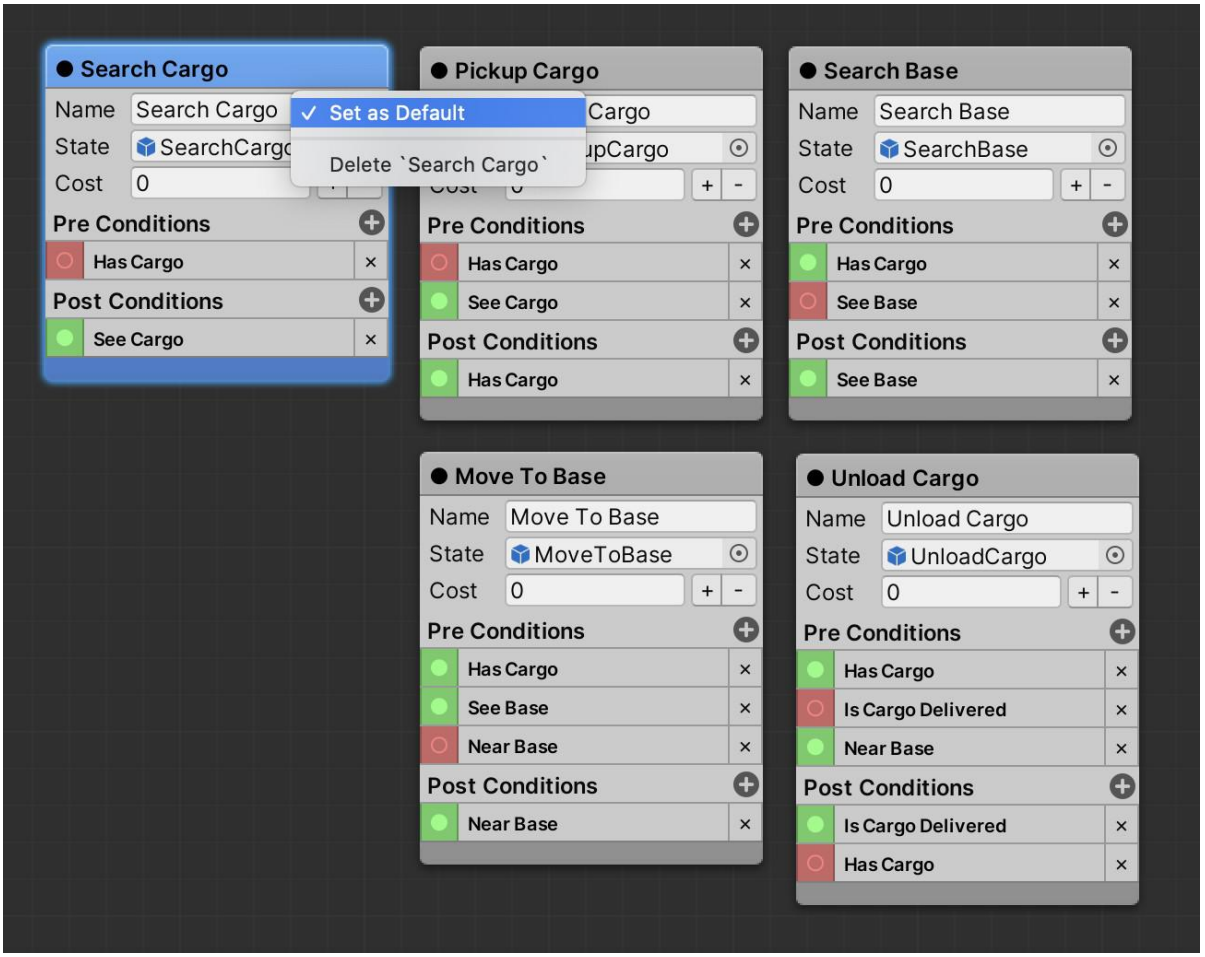


9. 打开“AI工作台”中的“交付Bot”场景，并将新创建的操作拖动到所有操作卡的状态字段中，如下图所示：

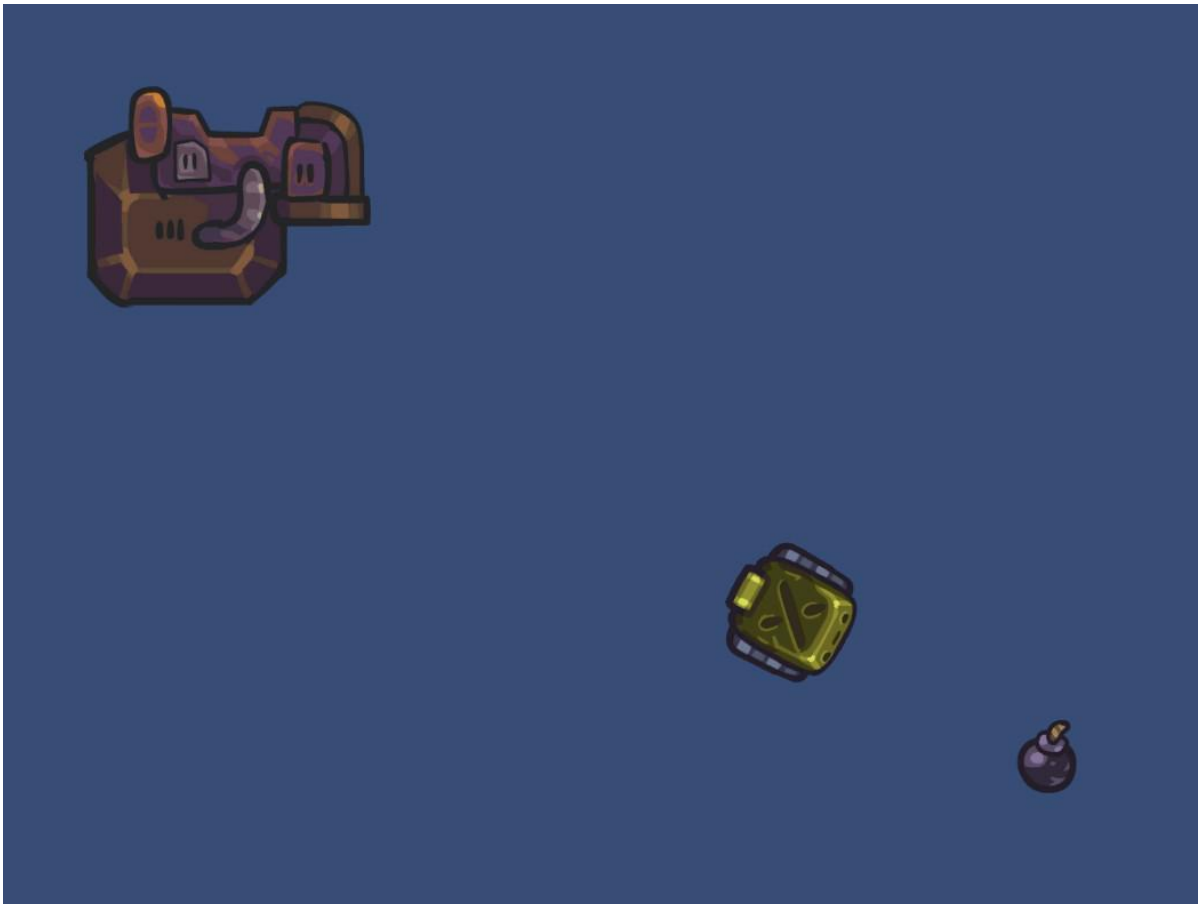


这看起来有点傻。但我们想尽快得到结果！AntAIAgent在确信场景中配置的所有操作都有一个状态脚本之前才能工作，所以这是一个欺骗它的好方法；）

10. 选择“搜索货物”卡，并将其设置为默认操作。因此，我们将让蚂蚁代理人知道，在任何难以理解的情况下，它都应该采取这一行动。在你的项目中，你可以为此创建一个单独的卡，它可以被称为“空闲”或“卡住”。



11. 开始游戏，看看部队是如何前往货物的，如下图所示是一个炸弹。一切都能正常工作，尽管在到达加载后，机器人会被困在它附近——这是因为我们没有“提货货物”的行动。



12. 现在创建脚本PickupCargoState.cs，并向其添加以下代码：

```
//PickupCargoState.cs--使用联
合引擎;
使用选集。ai;

公共交通中心
{
    私有Unit控制_control;

    公共覆盖void创建（游戏对象）
    {
        _control=aGame对象。Get组件<Unit控制>();
    }

    公共覆盖无效Enter()
    {
        //在地图上搜索货物并禁用它。vargo=游戏对象。查找
        （“Cargo”）;
        如果(go! =null)
        {
            走SetActive（假）;
        }
```

```
            完成();
        }
}
```

这个操作将在我们激活它后立即执行。就像上次一样，在舞台上创建一个空对象，将PickupCargoState.cs脚本附加到它上，并将预制件保存在“项目”窗口中。

13. 创建SearchBaseState.cs脚本，并向其添加以下代码：

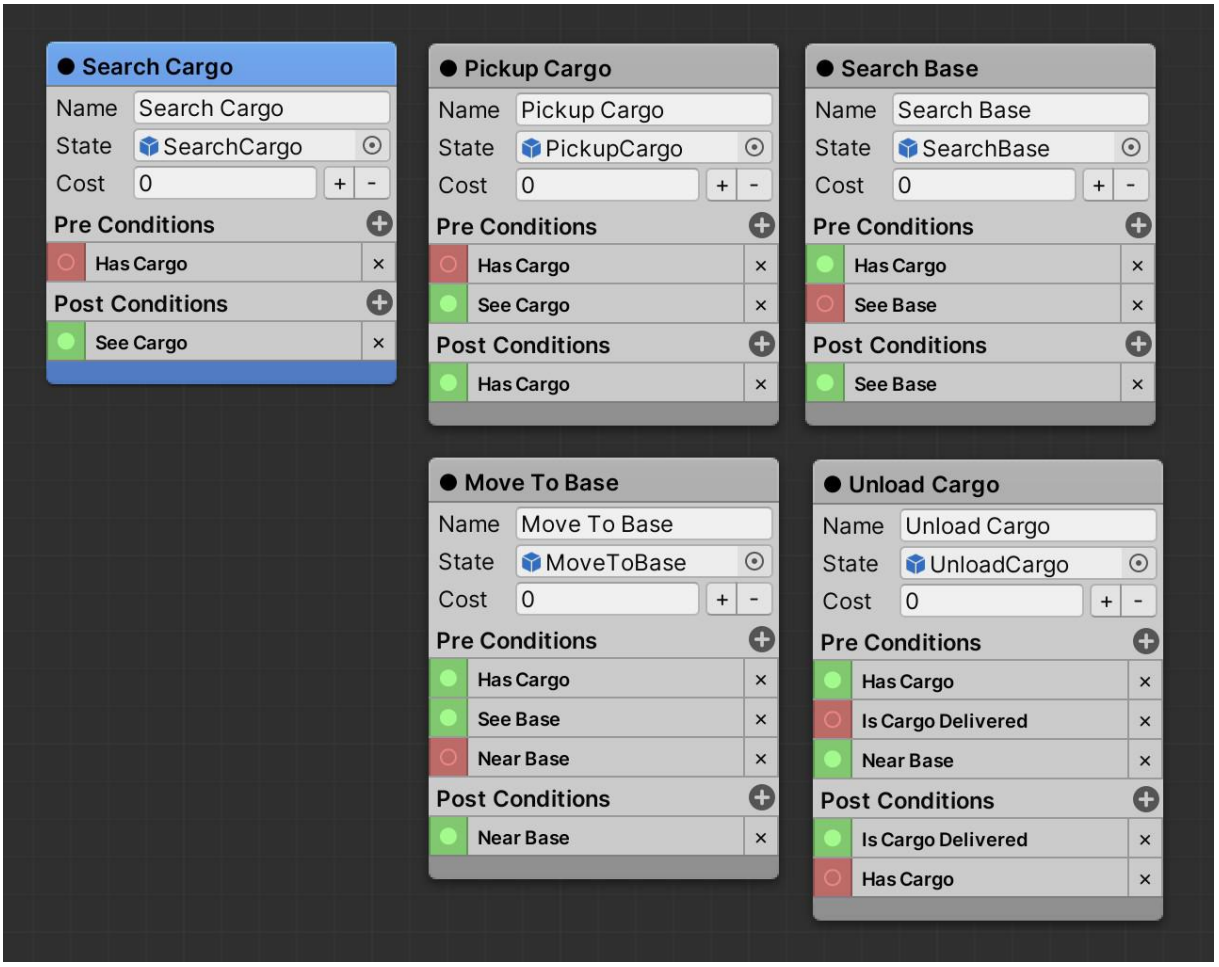
```
//SearchBaseState.cs--使用联
合引擎;
使用选集。人工智能；使用
Antil。使用;

公共班级搜索基础州
{
    私人浮速度=2.0f;

    私有转换_t；私人向量
    3_targetPos；私人浮动
    _targetAngle;

    公共覆盖void创建（游戏对象）
    {
        _t=aGameObject.Get组件<变换>();
    }

    公共覆盖无效Enter()
        //在地图上的搜索基础上。
        vargo=游戏对象。查找(“Base”)；如果
        (go! =null)
        {
            //如果我们看不到基础，就调用这个动作。
            //所以试着通过选择随机的位置来找到
            //在基地周围，直到我们发现它。varbasePos=去
            了。变换位置;
            _targetPos=新矢量3(
                basePos.x+AntMath.RandomRangeFloat(-2.0f    ,    2.0f)    ,
                basePos.y+AntMath.RandomRangeFloat(-2.0f, 2.0f), 0.0f
            );

            //Calc目标角度。
            _targetAngle=AntMath.AngleDeg(_t.位置, _targetPos);
            _t.旋转=四元数。欧拉(_t。旋转x，_t.旋转y,
_targetAngle);
            _targetAngle*=Mathf.Deg2广告;
```

```
            调试。日志(“找不到基础！””)；完
            成()；
        }
    }

    公共覆盖无效执行（浮动、删除时间、浮动、时间尺度）
    {
        //移动到基地。varpos=_t。
        位置；
        波斯。x+=速度**数学。代码(_targetAngle)；pos。+=速度*数学。罪
        （_targetAngle）；
        _t.位置=pos；

        //检查到底座的距离。
        如果(AntMath。Distance(pos，_targetPos)<=0.2f)
        {
            //我们到达了！
            //当前操作已完成。完成()；
        }
    }
```

请注意，这个脚本实际上重复脚本的代码SeachCargoState.cs——这表明它可能是值得把控制和移动单元的UnitControl.cs类，并在状态只给命令单元将移动和等待命令执行完成行动。

与上次一样，在“搜索库”场景中创建一个新对象，然后附加SearchBaseState.cs脚本到它并拖到“项目”窗口将其保存为文件。

14. 通过将预览文件从“项目”拖动到操作卡的状态字段中，将新创建的操作链接到场景中的操作，如下面的屏幕截图所示。



然后你可以运行这个示例，并确保当该单位拿起货物时，如果基地超出其视野，它就会走到基地附近的一个随机点。但当他找到一个基地并到达它时，然后一切都崩溃了。这是因为我们有一个“搜索货物”状态附加到“移动到基地”行动-我们将修复这个问题。

15. 创建一个新的脚本MoveToBaseState.cs，并向其添加以下代码：

```
//MoveToBaseState.cs--使用联
合引擎;
使用选集。人工智能；使用
Antil。使用；

公共阶级运动：公共阶级运动
{
    私人浮速度=2.0f；

    私有转换_t；私人向量
    3_targetPos；私人浮动
    _targetAngle；

    公共覆盖void创建（游戏对象）
```

```
        _t=aGameObject.Get组件<变换>();
    }

    公共覆盖无效Enter()
        //在地图上的搜索基础上。
        vargo=游戏对象。查找(“Base”)；如果
        (go！=null)
        {
            _targetPos=去了。变换位置;

            //Calc目标角度。
            _targetAngle=AntMath.AngleDeg(_t.位置，_targetPos);
            _t.旋转=四元数。欧拉(_t。旋转x，_t.旋转y,
_targetAngle);
            _targetAngle*=Mathf.Deg2广告;
        }
        其他的
        {
            调试。日志(“找不到基础！””)；完
            成();
        }
    }

    公共覆盖无效执行（浮动、删除时间、浮动、时间尺度）
    {
        //移动到基地。varpos=_t。
        位置;
        波斯。x+=速度**数学。代码(_targetAngle)；pos。+=速度*数学。罪
        （_targetAngle）；
        _t.位置=pos;

        //检查到底座的距离。
        如果(AntMath。Distance(pos，_targetPos)<=0.2f)
        {
            //我们到达了！
            //当前操作已完成。完成();
        }
```

同样，在舞台上创建一个空对象，并将MoveToBase.cs脚本绑定到它，并将其放到“项目”窗口进行预制，这样我们就可以将其附加到脚本中。

16. 创建另一个UnloadCargoState.cs脚本，并向其添加以下代码：

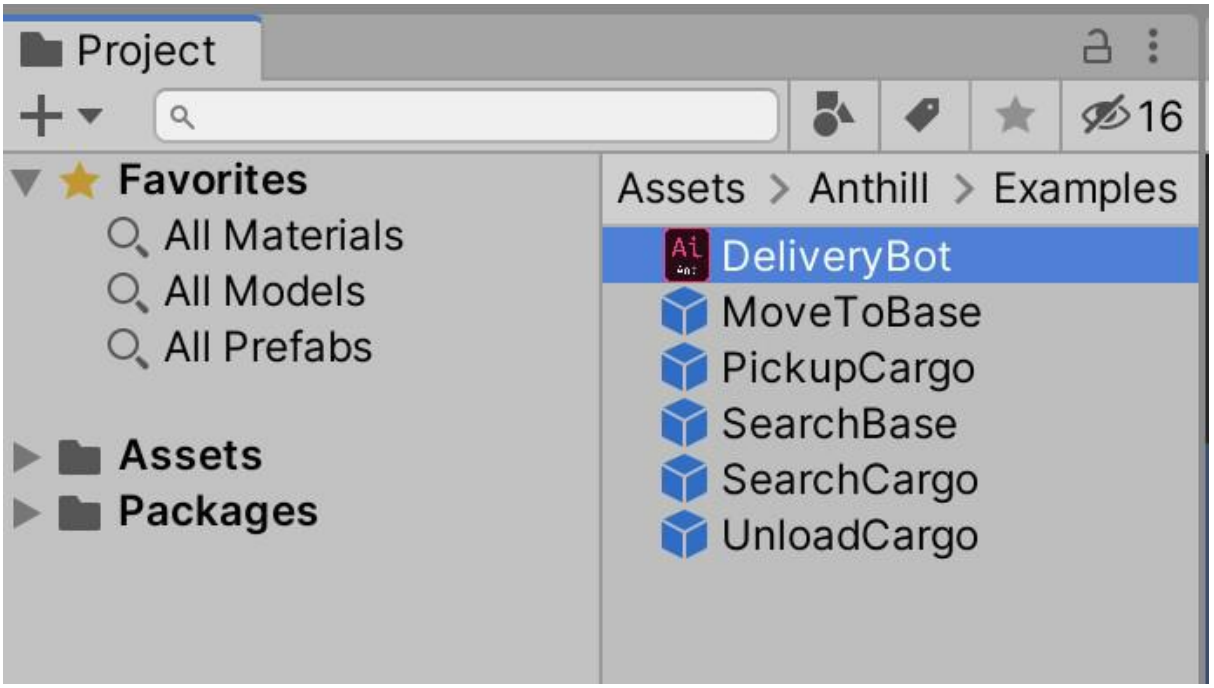
```
//UnloadCargoState.cs--使用联
合引擎;
使用选集。人工智能；使用
Antil。使用;
```

```
该州
{
    私有单元控制_control；私人游戏对象
    _cargoRef；私有向量3_initialPos；

    公共覆盖void创建（游戏对象）
    {
        //保存引用的货物重生时
        //单元完成交付。
        _cargoRef=GameObject.查找（“Cargo”）；
        调试。资产(_cargoRef！=null，“在现场没有找到货物！””)；
        _initialPos=_cargoRef.变换位置；
        _control=aGame对象。Get组件<Unit控制>()；
    }

    公共覆盖无效Enter()
    {
        //再次在随机的地图上产生货物的pos。
        _cargoRef.变换位置=新矢量3（
            _initialPos.x+AntMath.RandomRangeFloat(-2.0f，2.0f)，
            _initialPos.y+AntMath.RandomRangeFloat(-2.0f，2.0f)，0.0f
        )；
        _cargoRef.设置Active(true)；

        _control.HasCargo=假；完成()；
    }
}
```

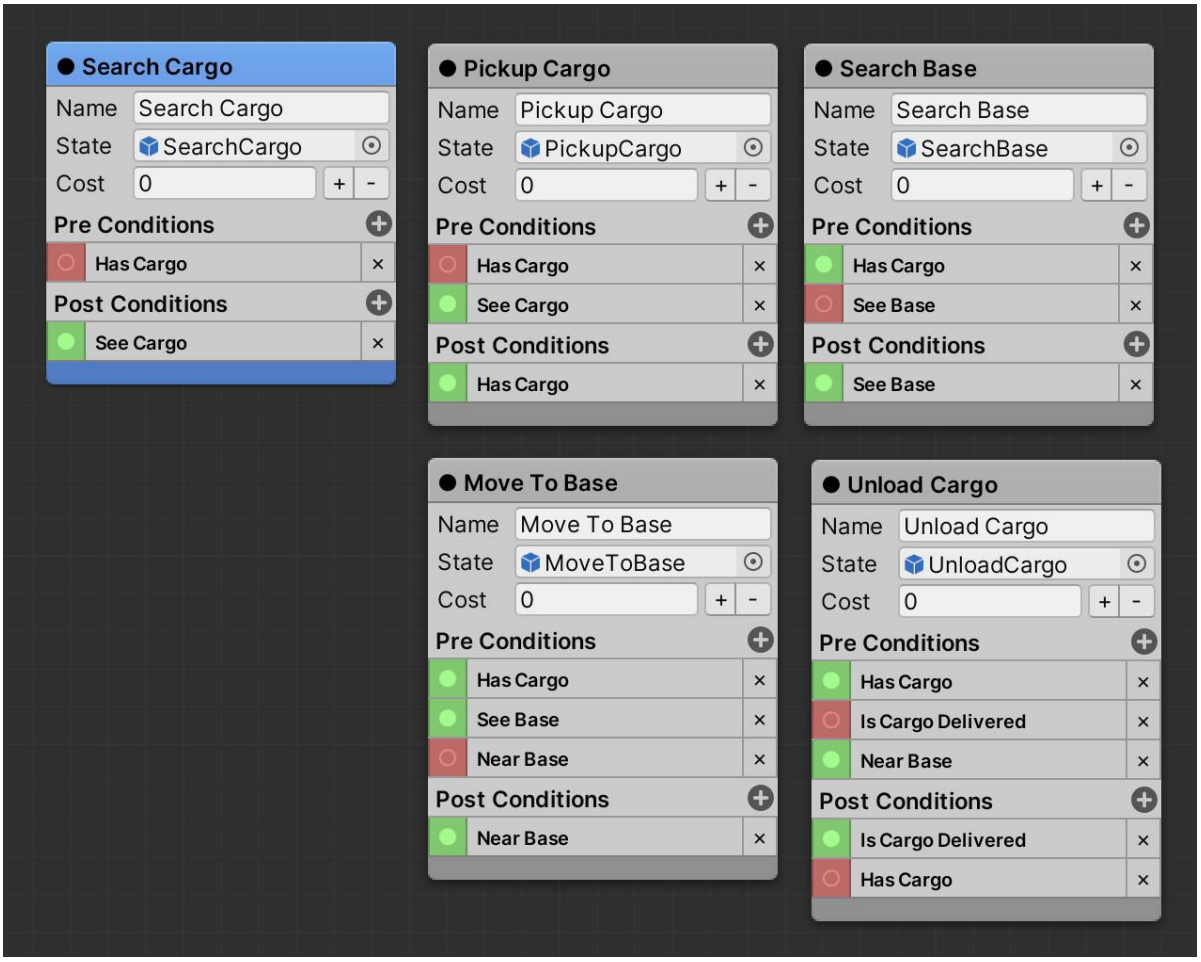
在此脚本中，在创建一个操作时，我们将始终保存对舞台上的货物对象的引用，以便在之前的货物交付时再次重新创建它。这不是一个正确的决定，但对于这个例子来说已经足够了。当该装置卸载其货物时，一个新的货物将出现在现场，这将无限期地发生。

我们在舞台上创建一个新的游戏对象，将UnloadCargoState.cs脚本附加到它上，将其放到“项目”窗口中，并附加到我们的脚本中。

17. 因此，我们应该采取一些行动：


Project
16
Favorites
All Materials
All Models
All Prefabs
Assets
Packages
Assets > Anthill > Examples
DeliveryBot
MoveToBase
PickupCargo
SearchBase
SearchCargo
UnloadCargo

它应附加到以下场景中：


Search Cargo
Name Search Cargo
State SearchCargo
Cost 0
Pre Conditions
Has Cargo
Post Conditions
See Cargo
Pickup Cargo
Name Pickup Cargo
State PickupCargo
Cost 0
Pre Conditions
Has Cargo
See Cargo
Post Conditions
Has Cargo
Search Base
Name Search Base
State SearchBase
Cost 0
Pre Conditions
Has Cargo
See Base
Post Conditions
See Base
Move To Base
Name Move To Base
State MoveToBase
Cost 0
Pre Conditions
Has Cargo
See Base
Near Base
Post Conditions
Near Base
Unload Cargo
Name Unload Cargo
State UnloadCargo
Cost 0
Pre Conditions
Has Cargo
Is Cargo Delivered
Near Base
Post Conditions
Is Cargo Delivered
Has Cargo

现在你可以运行一个例子，享受一个交付单位的工作，他将无休止地将货物运送到基地。

这是你需要知道的，以开始实现蚂蚁在你的AI！

自定义实现

要为规划师实施实施，并为您的单位制定行动计划，您只需要几件事情：

- 创建一个规划器的实例；
- 将场景加载到计划器中（计算机对象）；
- 设定世界的现状；
- 设置或更改一个目标（可选）。
- 建立一个计划，并使用它来组织你的单位的进一步行为。

最简单的自定义实现的代码如下的截图所示：

```
//SimpleExample.cs——使用
联合引擎；使用国歌。ai；

例如：单一行为
{
    公共AntAI场景场景；

    私有无效启动()
    {
        //1.创建人工智能规划器。
        //----------------------------------------------
        var规划师=新的天线规划师()；

        //为规划器提供的加载场景。
        调试。资产(场景！=null，“错过了场景！”)；规划师。加载场景
        （场景）；

        //2.创建世界状态。
        //--------------------------------------
        不同世界状态=新蚂蚁条件()；世界状态。BeginUpdate()；worldState.设置
        （示例场景。；世界上的状态。设置（示例场景。SeeCargo，假的)；世界状
        态。设置（示例场景。哈斯卡戈，假的)；世界状态。设置（示例场景。
        SeeBase，假的)；世界状态。设置（示例场景。NearBase，假的)；世界状
        态。EndUpdate()；

        //3.构建计划。
        //-------------------------
        var计划=新的AntAI计划()；
```

```
        规划师。制定计划(参考计划，世界状态，规划师。GetDefaultGoal());

        //输出计划。调试。日志(“<b>计
        划：</b>”);
        为(inti=0；i<计划。计数；i++)
        {
            调试。Log($“{(i+1)}{plan[i]}”)；.
        }

        //4.改变世界状态，重建计划。
        //
        ----------------------------------------------------------------world
        State.BeginUpdate(planner)；worldState.设置（示例场景。
        哈斯卡戈，真的)；世界状态。EndUpdate()；

        规划师。制定计划(参考计划，世界状态，规划师。FindGoal（“交付”）；

        //输出计划。调试。日志(“<b>计
        划：</b>”);
        为(inti=0；i<计划。计数；i++)
        {
            调试。Log($“{(i+1)}{plan[i]}”)；.
        }
```

本示例的源代码可以在这里找到：

选，简单的例子

反馈和支持

如果你有任何困难，你发现了错误，你需要建议或你有建议-请随时与我联系：

- 电子邮件 ant.anthill@gmail.com
- 脸书 https://www.facebook.com/groups/526566924630336/
- 断开 https://discord.gg/D9fASJ5