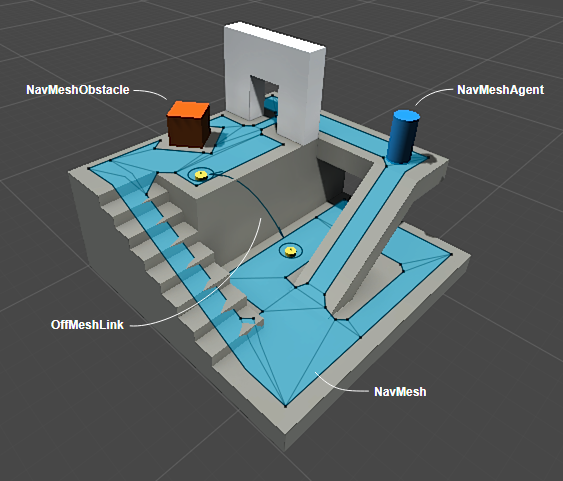
# 导航系统

## 概述

导航系统允许你创建一些在游戏场景里明智移动的角色，使用的导航网格在场景里将被自动创建。动态障碍物可在运行时修正角色的导航信息，网格链接（off-mesh links）可让你执行一些特定的行为，例如开门或者从平台上跳下。本章将详细描述导航和寻路系统。

这节内容会详细介绍如何为你的场景构建导航网格，创建网格代理，网格阻挡以及网格链接（off-mesh links）。



导航系统允许你在游戏场景里创建可导航的角色。这会让你的角色能够理解通过楼梯可以上二楼，或者从沟上可以跳过去。Unity的导航系统由一下几个部分组成：

**Navmesh （是Navigation Mesh的简称）**是描述场景里面可通过的面片以及能在场景里找到从一个可通过的位置移到另一个位置的一条路径。这个数据结构会根据水平几何自动的构建，或者烘焙。

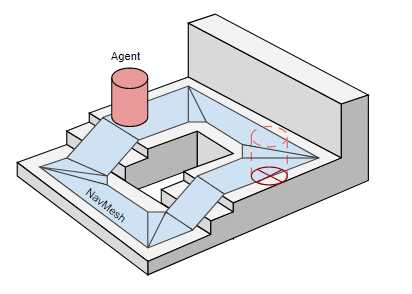
**NavMesh Agent（导航代理）** 组件帮助你创建那些当它们朝着自己目标移动时能避开其它代理。使用代理是因为游戏世界使用的导航网格并且知道像避开障碍物一样避开其他角色。

Off-Mesh Link 组件允许你去合成（链接）被切断的导航网格就是那些不能用可通过表面来表示的网格。比如，跳过沟或者栅栏，或者在你通过之前打开一扇门，都能够被描述成 Off-mesh links。

**NavMesh Obstacle（网格障碍）** 组件描述的是代理在场景中导航的时候应该避开的移动障碍。物理系统里面一个被控制的桶或者一个板条箱就是障碍的好例子。当障碍物处于移动的状态下代理会最好的避开它，一旦这个障碍物处于静止状态它将会在导航网格上挖一个洞，所以代理能围绕着这个洞修正它的寻路路径（就是绕着这个洞走），如果这个静态障碍完全阻挡了路径，那么代理会去找不同于当前的另一条路径。

## 导航系统内部工作机制

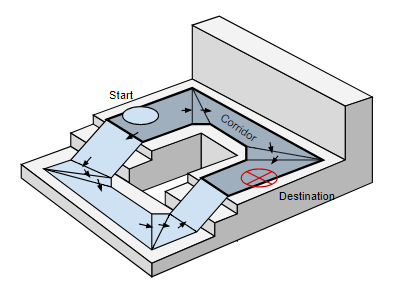
当你想要在游戏里聪明的移动你的角色（在AI系统中称其为代理），你不得不去解决下面的两个问题：如何找到目的地，然后是怎么移动到目的地。这两个问题紧密耦合，但在本质上完全不同。如何推理出关卡结构这个问题是全局的、静态的，因为要将这个问题考虑到整个场景中去。移动到目的地更加的局部性和动态性，它仅仅需要考虑移动的方向以及防止和其他移动的代理发生碰撞。



### 可行走区域

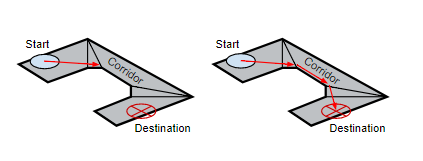
导航系统需要一些数据描述那些在场景中可行走的区域。代理可以在场景只能中站立和移动的区域叫 可行走区域（walkable area）。在Unity中代理被描述成圆柱体。可行走区域是在场景中可站立的位置计算出场景几何图形自动构建出来的。然后这些位置连接到几何图形的顶部表面。 这个表面我们就叫他 导航系统（简称 NvaMesh）。NavMesh 的表面是通过凸多边形存储的。凸多边形是一种很好的描述方式，因为我们知道在多边形的任意两个点之间是没有阻挡的。除了多边形的边界之外，我们还存储着相邻多边形之间的信息。这样我们就推理整个可行走区域。

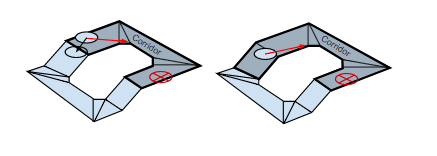
### 寻路算法



在场景中的两点找一条路径，首先我们要知道起点和终点之间相邻且最近的那些多边形。然后开始寻路，从起点开始遍历周围相邻的点，直到找到目的地所在的多边形。追踪这些多边形能够找到引导我们从起点到终点的多边形序列。一个通用的寻路算法叫A\*（读 A star），正是unity使用的寻路算法。

### 具体路径

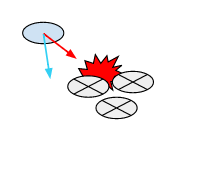




从起点到终点路径上的序列多边形我们叫它走廊。代理通过转向控制移动到达走廊的下一角的方式到达目的地。在场景中只有一个代理移动的简单游戏，它能很好的一举找到走廊上所有的角落以及控制角色沿着这些角落连成的线段移动。 当同时有多个代理在场景中移动的时候，它们在躲避其他角色时会偏离原来的路径。尝试着使用路径中的线段去修正偏离会变得很难且容易出现错误。

由于代理在每帧的运动时非常小的。我们可以用多边形的连接的方式去固定走廊防止代理绕路。然后快速找到下一个可见的角落并朝着它转向移动。

### 避开障碍



控制逻辑会找到下一个角落的位置并且给出一个需要达到目的地的预期方向和速度。使用预设的速度会导致和其他代理之间的碰撞。

避障选择了新的速度去平衡预期移动方向上和其他网格边缘上的将到来的碰撞。Unity使用了反向速度障碍（RVO）来预测防止碰撞 。

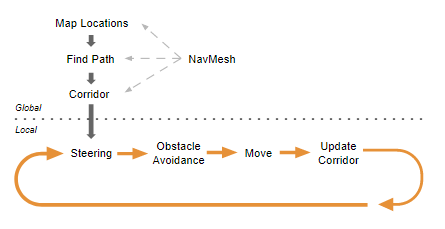
### 让Agent移动

最后通过转向和避障计算出的最终速度。Unity的代理模拟的简单动态模型，使用加速度让其移动得更加平滑和自然。

在这个阶段可以将模拟态代理的速度输入到Mecanim 动画系统中去控制角色移动，或者说让导航系统去处理。

一旦使用上面的任意一种方法来移动，模拟态代理的位置将被移到网格上。最后的这一小步对稳定的导航系统非常的重要。

### 全局导航与局部导航



最重要的事情是如何去理解全局导航和局部导航之间的差异。

全局导航用来查找穿过世界的走廊，在世界中寻路需要耗费处理性能和内存。

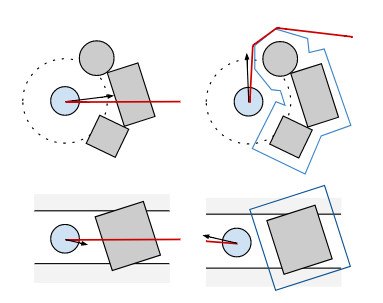
线性连接的多边形描述的路径就是代理可灵活转向的数据结构，在代理进行位置移动的时候可局部调整。局部导航试图找出如何有效的朝着下一个角落移动，且不与其他的代理或者移动物体发生碰撞。

### 障碍的两个例子

许多导航应用需要不同的障碍类型而不是不同的代理。就像是射击游戏和赛车里的那些箱子和桶。这些障碍物通过局部避障或者全局寻路解决。

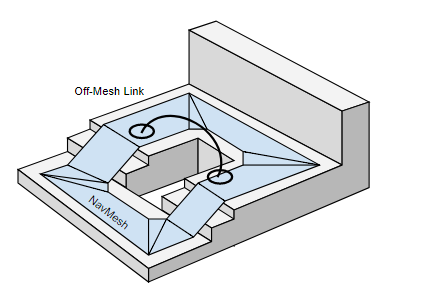
当障碍物时移动的时候，最好采用局部避障处理寻路。这样代理就能预测性的规避障碍。当障碍物处于静止状态时，会阻挡所有代理移动，这些障碍物就会影响到全局寻路，即导航网格。

改变导航网格被称其为雕刻（打洞）。程序处理了障碍物和导航网格接触的部分并且在接触部分的网格上挖了一个洞。这非常的消耗计算机性能，这是另外一个引人注目的重点，为啥移动的障碍物要使用碰撞规避.



局部碰撞规避经常被用来绕过疏散的障碍物，因为算法时局部的，它只会考虑下一步的即时碰撞，并不会绕过陷阱或者出路障碍阻塞路径的情况。这些情况呢可以用挖洞模式解决。

### 链接关系



导航网格之间的多边形连接其实质时寻路系统内部间链接。有时候必须让代理在不可通过区域间导航。比如，跳过栅栏，或者穿过关闭的门，这些情形需要知道行动位置（从那里起跳，着地点在哪里等等情形）。

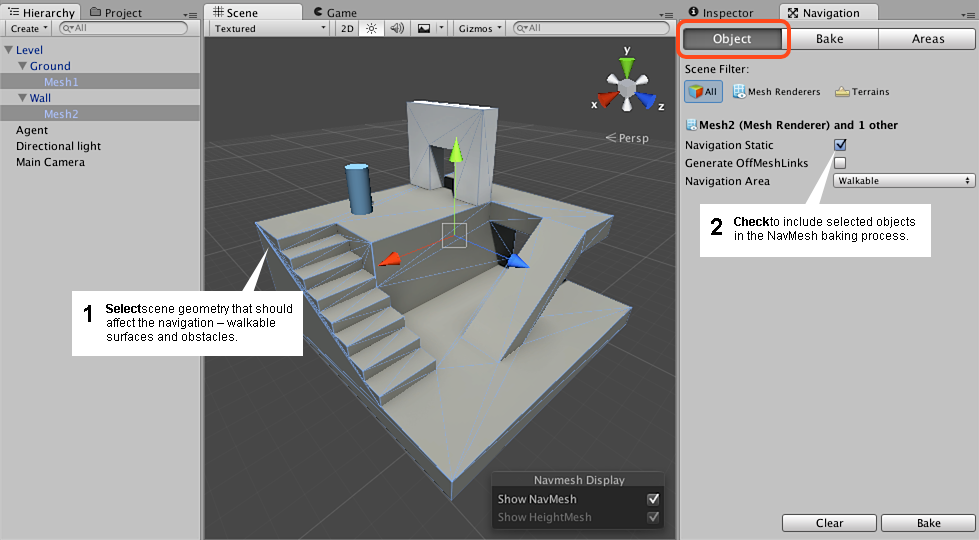
这些情形可以使用Off-Mesh Links 进行注释，它会告诉寻路着这里有一条指定的路径存在。这些链接在寻路中可以稍后访问，并且特殊的动作可以被执行。

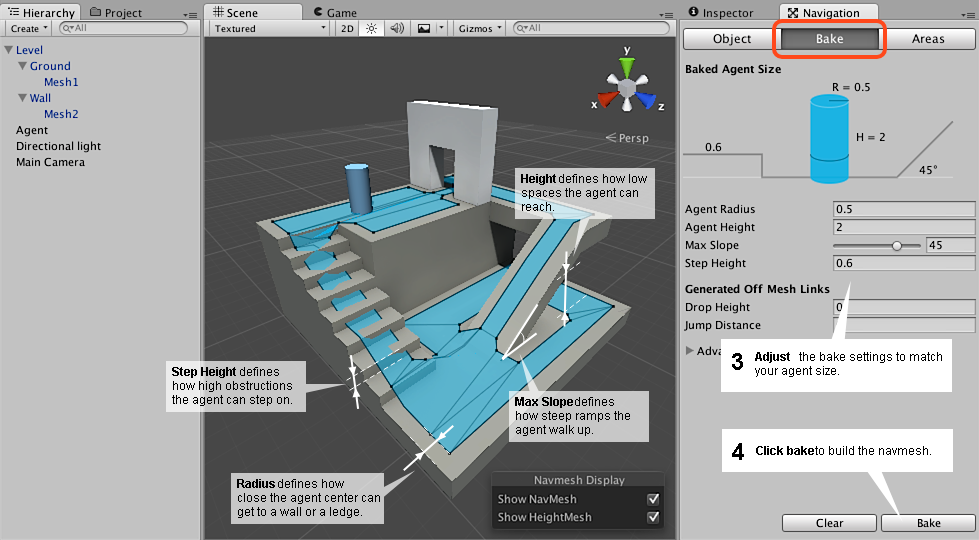
## 构建导航网格

从几何关卡创建导航网格的过程称其为导航网格烘焙。该过程收集被标记为Navigation Static 的网格渲染和地形，然后创建接近于水平表面可行走的导航网格。

在Unity里，导航网格生成可以在导航窗口里处理（菜单：Window> Navigation）

为你的场景生成导航网格可以通过如下4个快速步骤：





1. 选择能够影响几何场景导航的那些可通过区域和障碍物。
2. 焙处理被选中的对象需要勾选中 Navigation Static 选项框.
3. 调整烘焙设置去适应代理尺寸  
    a. 代理半径定义了代理可以靠近墙或者对象边缘的距离。

b. 代理高度定义了代理可以达到的空间有多低。

c. 最大坡度定义了代理可以在多大的斜坡上行走。

d. 台阶高度定义了代理可以踏上多高的障碍。

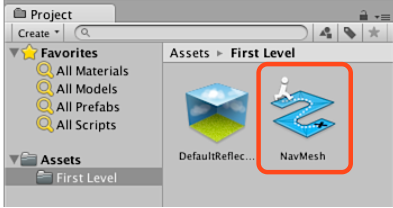
1. 点击烘焙（Bake）创建网格。

当导航窗口是打开并且可见，那么导航网格将会以叠加在水平几何体上的蓝色层显示在场景中。

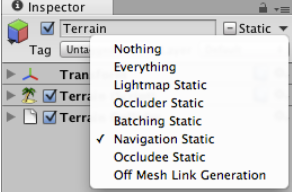
在上面的一些图片中你可能注意到了，生成的导航网格有一定的压缩。导网格所展现的区域是代理中心点能通过的区域。概念上讲，无论你放置的代理是作为一个点在压缩的导航网格上还是一个圆在全尺寸导航网格上都是等同的。代理作为一个点很好解释，它能有更好的运行效率并且能够让设计者立刻看见代理是否挤压通过一些空白区域且不用担心代理自身的半径。

另外需要注意的时导航网格时无限接近于可通过表面的。例如，在台阶生成的导航网格其实是一个平面 ，但是台阶本身确实有阶梯的。这要做是为了防止导航网格数据量小。还有其他的作用，有时候可能在几何关卡里需要一点额外的小空间以使允许代理通过狭窄的区域。

当烘焙完成以后，导航资源文件会被创建在Asset文件加下的同名目录里面。比如，如果Assets下你有一个场景叫First Level，导航的资源文件也会被创建在Assets>First Level 目录中，叫 NavMesh.asset。



### 额外的标记对象烘焙工作流程



另外在Navigation 窗口里标记对象为 Navigation Static，如上图所示，也可以使用Inspector面板的右上角的Static菜单。如果你没有打开Navigation窗口，上面那样做也很方便。

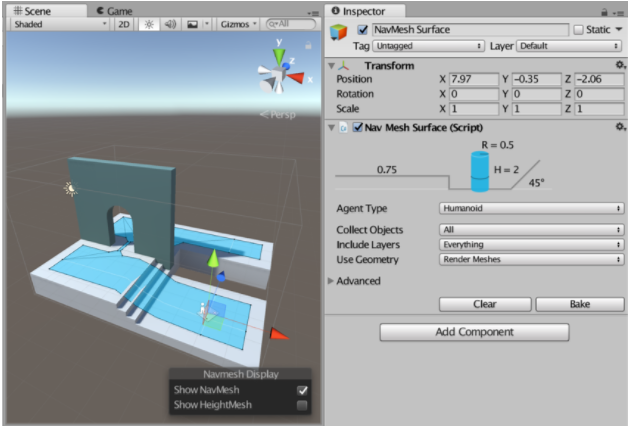
## 导航网格构建组件

### 导航网格表面组件（NavMesh Surface）

NavMesh Surface组件描述的是一个特殊的NavMesh Agent类型的可通过区域，它定义了整个场景中可被创建的导航网格部分。

NavMesh Surface并不在标准的的Unity安装包中，可以在资源商店中找到这个插件。

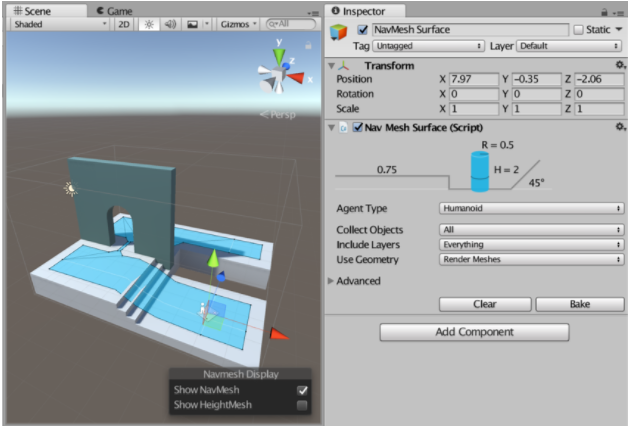
使用NavMesh Surface组件，要在GameObejct菜单栏中选中AI子选项在选中NavMeshSurface 即可。点击以后会创建一个空的节点，上面挂载着NavMesh Surface组件。一个场景里面可以挂载多个NavMesh Surface 组件。



（导航表面组件的效果）

你可以在任何场景对象中挂载NavMesh Surface 组件。当你想在Hierarchy视图中用挂载父节点的方式来决定那些场景对象可以被导航网格包含时，这很有用。

#### 属性功能



**代理类型**：代理类型运用在导航网格表面。用于烘焙设置以及寻路过程中代理和导航网格表面匹配。

1.人形状

2.怪物（非人形）

**Collect Objects**:定义那些对象将会被烘焙。

**All (全部)** - 使用所有的激活的游戏对象（这是默认选中的）

**Volume** - 使用与边界体积重叠的所有激活的游戏对象

**Children** - 使用子物体上挂载着NavMesh Surface组件的所有有激活的游戏对象。

**Include Layers**: 对象所在的层将会包含在烘焙中。除了Collect Objects 里指定的那些游戏对象以外，这允许进一步排除特定的对象（如，特效和动画角色）。默认选中的是所有的对象，但是你可以切换以下这些选项（用打勾表示）或者单独关闭。

Noting (自动不选中所有的选项，意思是关掉它们)

Everything(自动选中所有的选项，意思是打开它们)

Default 默认层

TransparentFX 层

Ignore Raycast 层

Water层

UI层

**Use geometry**:选择用于烘焙的几何体。

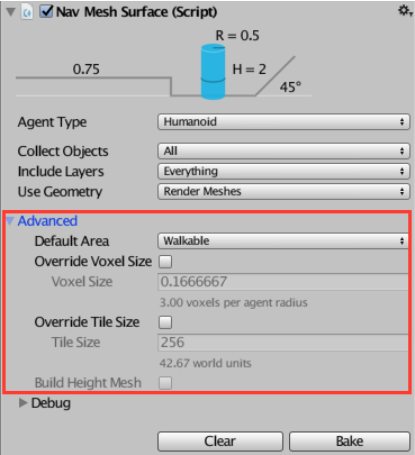
**Render Meshes** - 选择Render Mesh 组件和Terrain 组件几何体。

**Physics Colliders** - 选择挂载着碰撞器和Terrains（地形）组件的几何体。代理在当前环境中使用这个选项比使用Render Meshes 选项在移动时能够更加接近物理边界。

使用NavMesh Surface 组件主要设置可以大范围的过滤掉输入的结合体。Unity是如何精细处理每个对象上的几何体输入的呢，它是使用的NavMesh Modifier（导航网格修正） 组件实现的。

烘培处理会自动的排除挂有NavMesh Agent 组件和NavMesh Obstacle组件的对象。它们是是动态的NavMesh 用户，所有不会与导航网格绑定在一起。

#### 高级设置



高级设置部分允许您自定义以下附加参数:

**Property Function**(属性函数)

Default Area 定义构建NavMesh时生成的区域类型。

Walkable 可行走的区域（默认选中）

Not Walkable s不可行走的区域

Jump 可跳跃的区域

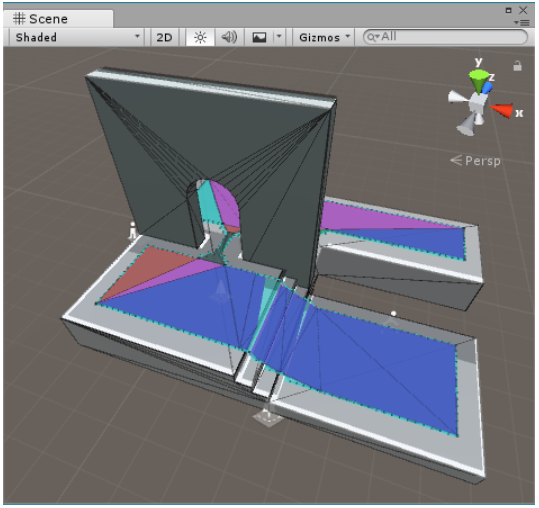
可以使用NavMesh Modifier 组件去修改区域类型的更多信息。

**Overried Voxel Size（覆盖体素大小）**：控制着Unity在烘焙导航网格时如何处理输入的几何信息（这是在速度和准确度之间折衷处理）.勾中复选框使其生效。默认是没有勾中的。每个代理半径3个体素（直径6个体素）允许捕获狭窄的通道，列如门，同时能够保持较快的烘焙时间。【对于较大的区域，半径使用1到2个体素能加速烘焙。室内紧密的斑点更适合较小的体素，如每半径4到6个体素。超过每半径8个体素并不会提供额外的附加效果。

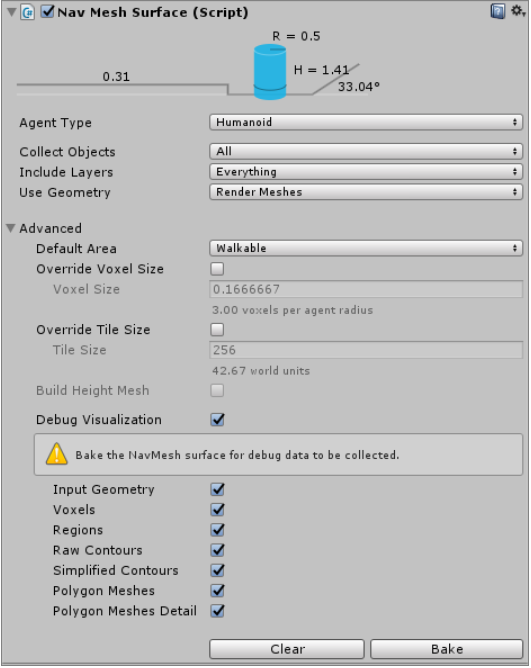
**Override Tile Siz（覆盖切片大小）**：为了使烘焙过程并行处理以及有效利用内存，被烘焙的场景会被分割成不同的小块去烘焙。在导航网格中那些可见的白色线条就是边界。默认的切片大小是256个体素，能够在内存利用和导航网个片段做很好的折衷。想要改变这个大小，需要狗钻中这个复选框，并且在Tile Size 字段里输入你想要体素值。切片越小，导航网格碎片越多，这可能导致一些非最佳的路径产生。导航网格雕刻(挖洞模式)也是使用的这些切片。如果你有多个障碍物，你可以用一个较小的值（如64 到128 之间）来加速雕刻，如果你想在运行时烘焙导航网格，使用一个较小的切片值来保持最大内存使用率最低。

**Build Height Mesh（构建高度网格）**: 不支持.

#### 可视化高级调试（Advanced Debug Visualization）



(输入的几何，区域，多边形网格细节，原始的轮廓显示在用调试选项构建NavMesh之后)



(NavMesh Surface组件在Inspector上的调试选项)

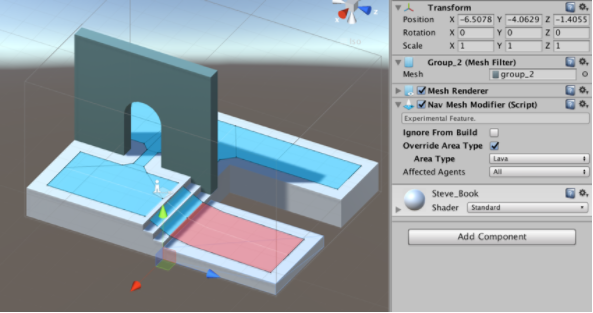
使用调试可视化部分中的设置来诊断在烘焙过程中遇到的任何问题。不同的复选框显示NvaMesh的构建过程，包括输入的场景体素化（输入几何），区域分割（区域），轮廓显示（轮廓），以及导航网格多边形（多边形网格）。

### 导航修正（NavMesh Modifier）

导航网格修正组件在运行状态烘焙导航网格中调整特定的对象行为。NavMesh Modifer 组件不在Unity的标准安装包中；可以查看文档中的高级NavMesh构建组件获取它的访问信息。

在菜单 GameObject>AI>NavMesh Modifier 中可以找到使用它。

下面的这张图片中，平台的右下角挂在了一个Modifier 组件并且把他的AreaType 设置成Lava.



NavMesh Modifier 组件分层影响游戏对象，也就是说如果它挂在到某个节点上那么所有的子节点也会受到影响。另外，如果有另一个NavMesh Modifier组件 挂在更下一层，那么下面这一层的NavMesh会覆盖上一层NavMesh组件的影响。

NacMesh Modifier 组件也会影响NavMesh的生成过程，意味着NavMesh必须更新以反映出NavMesh Modifier带来的改变。

**Property & Function**:**（属性&功能**）

**ignore From Build（构建忽略）**：勾中复选框在构建过程中将排除这个游戏对象及其全部内容。

**override Area Type（覆盖区域类型）**：选中复选框将改变对象的区域类型包含其所有子节点的区域类型。

**Area Type(区域类型)**：从下来菜单中选择一个新的区域类型。

**Affected Agents(受影响的代理)**：修改代理的选择。比如，特定的代理可以选择排除某些障碍。

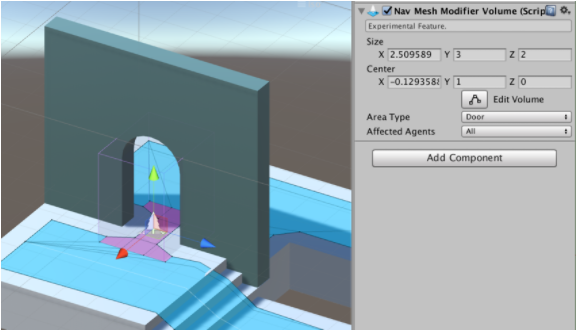
### 导航网格修改量（NavMesh Modifier Volume）

NavMesh Modifier Volume (导航网格修改量)组件不在Unity标准安装中; 请参阅高级NavMesh构建组件的文档以获取它的信息。

NavMesh Modifier Volume将定义区域标记为特定类型（例如Lava或Door）。 而NavMesh Modifier会将某些GameObjects标记为区域类型。 NavMesh Modifier Volume允许您根据特定的需求制定区域类型。

NavMesh Modifier Volume 标记不会被表示为单独几何体为可行走区域非常有用，如，危险区域。当然你也可以用它来标记一些不可行走的区域。

NavMesh Modifier Volume也会影响NavMesh生成过程，这意味着必须更新NavMesh以反映对NavMesh Modifier Volumes的任何更改。



Inspector面板中显示的NavMesh Modifier 组件

#### 属性和功能

|  |  |
| --- | --- |
| Size（尺寸） | 由XYZ值定义的NavMesh修改量的尺寸 |
| Center（中心） | 相对于GameObject中心的NavMesh Modifier Volume的中心，由XYZ测量值定义 |
| Area Type（区域类型） | 描述NavMesh Modifier Volume应用的区域类型 |
| -Walkable | 可行走（这是默认选项） |
| -Not walkable | 不可行走 |
| -Jump | 可跳跃 |
| Affected Agents | 可选择那些代理受影响。如，您可以选择仅将所选NavMesh设置为对特定代理类型的危险区域 |
| -None | 没有 |
| -All | 所有（默认选中） |
| -Humanoid | 人形 |
| -Ogre | 非人形 |

### 导航网格链接（NavMesh Link）

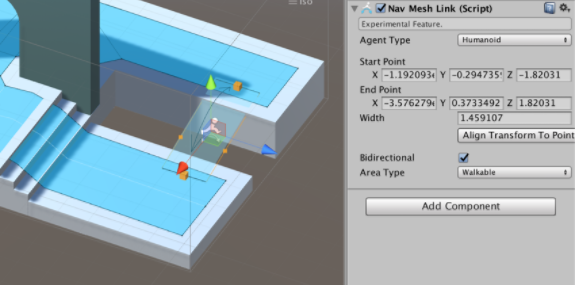
导航网格修正组件在运行状态烘焙导航网格中调整特定的对象行为。NavMesh Modifer 组件不在Unity的标准安装包中；可以查看文档中的高级NavMesh构建组件获取它的访问信息。

NavMesh链接在使用NavMeshes的两个位置之间创建可导航链接。

此链接可以是从一个点到另一个点，也可以跨越一个间隙，在这种情况下，Agent使用沿着入口边缘的最近位置穿过链接。

您必须使用NavMesh链接来连接不同的NavMesh曲面。

在菜单 GameObject>AI>NavMesh Link 中可以找到使用它。



属性和功能

Agent Type :可以使用链接的代理类型

- Humanoid （人形）

- Ogre（非人形）

Start Point:链接的起点，相对于GameObject。 由XYZ测量定义。

End Point:链接的终点，相对于GameObject。 由XYZ测量定义。

Align Transform To Points:点击这个按钮在连接的中心点移动游戏对象，并将变换的前进轴与终点对齐。

Bidirectional(双向)：在选中此复选框后，NavMesh Agent将以双向方式（从起点到终点，从终点返回起点）遍历NavMesh链接。当此复选框未被选中时，NavMesh链接仅具有一次性功能， 方式（仅从起点到终点）。

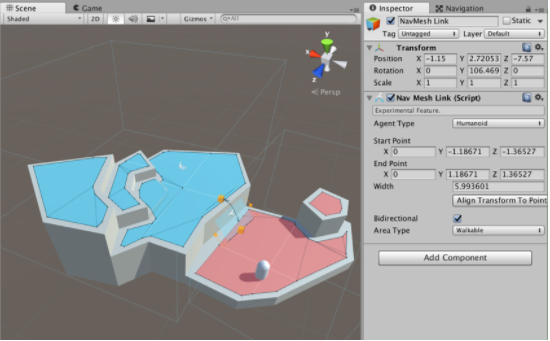
Area Type:NavMesh链接的区域类型（这会影响寻路成本）。

- Walkable (this is the default option)（可行走（这是默认选项）)）

- Not Walkable（不可行走）

- Jump（可跳跃）

#### 链接多个导航网格曲面



如果您希望Agent在场景中的多个NavMesh曲面之间移动，则必须使用NavMesh链接进行连接。

在上面的示例场景中，蓝色和红色的NavMeshes被定义在不同的NavMesh曲面中，并用NavMesh链接来连接它们。

您可以使用多个NavMesh链接连接NavMesh曲面。

Both the NavMesh Surfaces and the NavMesh Link must have same Agent type.

NavMesh曲面和NavMesh链接必须具有相同的Agent类型。

NavMesh链接的起点和终点必须只在一个NavMesh表面上。 如果在同一位置有多个NavMeshes，请小心。

如果您正在加载第二个NavMesh曲面，并且第一个场景中有未连接的NavMesh链接，请检查它们是否未连接到任何不需要的NavMesh曲面。

### 构建导航网格的API

NavMesh构建组件为您提供了用于构建（也称为烘焙）以及在运行时和Unity编辑器中使用NavMeshes的其他控件。

NavMesh Modifiers 不在Unity标准安装中; 请参阅高级NavMesh构建组件的文档以获取如何访问它的信息。

#### 导航表面（Navmesh Surface）

**Properties(属性)**

agentTypeID (代理类型ID)--描述构导航网格为代理类型构建的ID。

collectObject(采集对象)定义输入的几何对象是如何从场景里面被收集（UnityEgine.AI.CollectOBjects 之一）起来的

All (全部)--使用场景中所有的对象。

Volume(体积）--使用场景中触摸边界体积的实例化对象（查看大小和中心）

size (大小)--决定构建的体积。这个大小不会受到缩放的影响

center (中心) -- 构建的体积的中心相对于transform 的中心。

layerMask (遮罩层)-- 位掩码定义了必须将对象包含中烘焙中的层

useGeometr-使用几何图形--定义了用于烘焙的几何（UnityEngine.AI.NavMeshCollectGeometry 之一））

RenderMeshes 网格渲染--使用网格渲染和地形里面的几何体。

PhysicsColliders物理碰撞-- 使用碰撞体和地形提供的几何体。

defaultArea 默认区域--所有输入几何体的默认区域类型，除非特殊指定。

ignoreNavMeshAgent  忽略的网格导航代理--为True时挂有NavMesh Agent组件的游戏对象在输入时将会被忽略。

ignoreNavMeshObstacle 忽略导航网格障碍-- 为True时挂有NavMeshObstacle组建的游戏对象在输入时将会呗忽略。

overrideTileSize 覆盖切片（图块）大小 -- 为True时切片（图快）大小将会被设置。

tileSize 平铺大小 -- 体素中的切片大小（这个组件描述如何选择切片大小信息）

overrideVoxelSize 覆盖体素大小 -- 为True时设置体素大小。

voxelSize 体素大小 -- 世界单位体素的大小（组件描述的是如何选择切片大小的信息）

buildHeightMesh 构建指定高度网格 -- 未实现

bakedNavMeshData 烘焙的导航网格数据 -- 导航网格表面使用时引用的导航网格数据，为null 时 不生效。

activeSurfaces生效的曲面 -- 所有激活的NavMeshSurfaces 组件列表

注意：上述值会影响烘焙的结果，所以你必须调用Bake()去包含它们（就是各种设置以后，必须调用一下Bake()函数，点击Bake按钮即可）。

**Public Functions**

void Bake ()

void类型的Bake函数

给予NavMesh Surface 足组件上的参数设置烘培出新的导航啊网格数据。这些数据通过遍历 bakedNavMeshzData 来访问。

#### NavMesh Modifier

**Properties（属性）**

overrideArea 覆盖区域 -- 为True时 Modifier 组件会重写区域类型。

area 区域 -- 要应用的新区域类型

ignoreFromBuild 构建时忽略 -- 为True时，对象包含Modifier 组件以及它的子节点 不应用于NavMesh 烘焙

activeModifiers 生效的Modifiers -- 所有激活的NavMeshModifiers组件构成的列表

**Public Functions（公有函数）**

bool AffectsAgentType(int agentTypeID)

返回true,如果modifier 应用指定的代理类型，否则为false.

#### NavMesh Modifier Volume

**Properties(属性)**

size 大小 -- 本地空间单元中边界卷的大小。 变换会影响大小。

center 中心 - 本地空间单元中边界卷的中心。 变换影响中心。

area 区域 - 要应用边界卷内的NavMesh区域的区域类型。

**Public Functions**

bool AffectsAgentType(int agentTypeID)

Modifier用于指定的代理类型时返回true。

#### 导航网格链接

**Properties 属性**

agentTypeID- 可以使用链接的代理的类型。

startPoint - 本地空间单元中链接的起始点。 transform会影响位置

endPoint - 本地空间单元中链接的结束点。 transform 会影响位置。

width - 以世界长度单位表示的链接宽度。

bidirectional双向 - 如果为true，则可以通过两种方式遍历链接。 如果为false，则只能从开始到结束遍历链接。

autoUpdate - 如果为true，则链接更新端点以遵循每个帧的GameObject变换。

area 区域 - 链接的区域类型（用于寻路成本）。

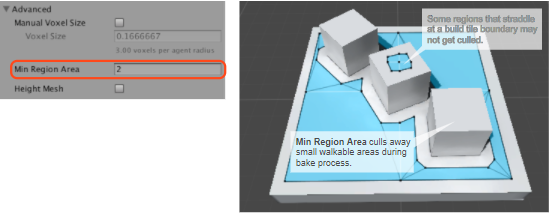
**Public Functions**

void UpdateLink()

更新链接以匹配关联的转换。 这对于更新链接很有用，例如在更改Transform位置之后，但如果启用了autoUpdate属性，则不需要。 但是，如果您很少更改链接转换，则调用UpdateLink可以使性能影响小得多。

## 高级导航网格烘焙设置

**Min Region Area**



Min Region Area高级构建设置允许您清除小型未连接的NavMesh区域。 表面积小于指定值的NavMesh区域将被删除。

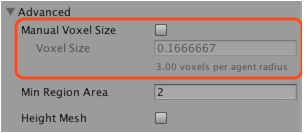
请注意，尽管有最小区域设置，但有些区域可能无法删除。 NavMesh并行构建为一个网格。 如果某个区域跨越切片边界，则该区域不会被移除。 其原因是区域修剪发生在构建过程中不能访问周围区块的阶段。

**Voxel Size**

手动设置体素大小允许您更改烘烤过程的准确性

NavMesh烘焙过程使用体素化从任意级别的几何构建NavMesh。 在算法的第一遍中，场景被光栅化成体素，然后提取可行走的曲面，最后将可行走的曲面变成导航网格。 体素大小描述了生成的NavMesh如何准确地表示场景几何。

默认精度设置为每个代理半径有3个体素，也就是说，整个代理宽度是6个体素。 这是精度和烘烤速度之间的良好折衷。 将体素大小减半将使内存使用量增加4倍，并且构建场景需要花费4倍的时间。



通常，您不需要调整体素大小，但有两种情况可能需要：构建较小的代理半径或更精确的NavMesh。

**Smaller Agent Radius**

当您烘烤手动设置较小的代理半径时，NavMesh烘烤系统也会减小体素尺寸。 如果其他代理维度保持不变，则可能不需要增加NavMesh构建分辨率。

The easiest way to do that is as follows:（译：最简单的方法如下：）

1.将代理半径设置为实际代理半径。

2.打开手动体素大小，这将采用当前体素大小并“冻结”。

3.人为设置更小的代理半径，因为您检查了手动体素大小，体素大小不会改变。

**More Accurate NavMesh**

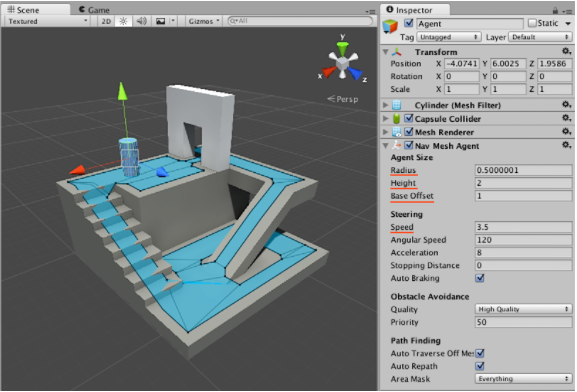
如果你的关卡有很多密集点，你可能希望通过缩小体素来提高准确性。 体素大小下的标签显示体素大小和代理半径之间的关系。 一个很好的范围是2-8之间，比这更进一步，通常会导致非常长的构建时间。

当你有意在你的游戏中建立紧密的走廊时，请注意除了代理半径以外，你应该至少保留4 \* voxelSize 清楚空间，特别是当走廊有角度时。

如果您需要小于NavMesh烘焙支持的走廊，请考虑使用“Off-Mesh Links”。 这些还有额外的好处，您可以在使用它们时检测到它们，例如可以播放特定的动画。

## 创建导航网格代理

一旦你有一个适合你的关卡的NavMesh，现在是时候创建一个可以导航场景的角色了。 我们打算从一个圆柱体上构建我们的原型代理并将其设置为运动。 这是使用NavMesh Agent组件和一个简单的脚本完成的。



首先让我们创建角色：

1.创建一个圆柱体：GameObject> 3D Object> Cylinder。

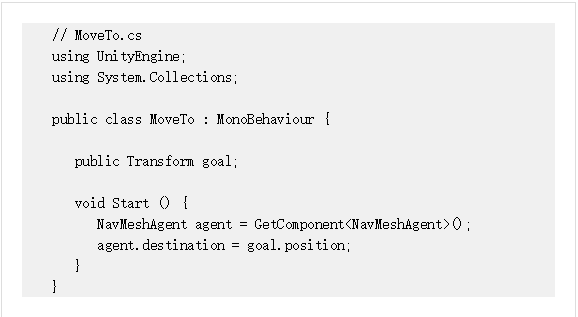
2.默认的圆柱体尺寸（高度2和半径0.5）适用于人形造型剂，所以我们将保留它们原样。

3.添加NavMesh代理组件：组件>导航> NavMesh代理。

现在你有了简单的NavMesh Agent准备好接收命令！

当你开始试用NavMesh Agent时，你很可能会根据你的角色大小和速度来调整它的尺寸。

NavMesh Agent组件处理一个角色的路径查找和移动控制。 在您的脚本中，导航可以像设置所需的目标点一样简单 - NavMesh Agent可以从此处处理所有内容。



接下来，我们需要构建一个简单的脚本，它允许您将角色发送到由另一个游戏对象指定的目的地，以及一个Sphere，它将成为移动到的目标：

1.创建一个新的C＃脚本（MoveTo.cs）并用上面的脚本替换它的内容。

2.将MoveTo脚本分配给您刚创建的角色。

3.创建一个球体，这将是代理将移动到的目的地。

4.将球体从角色移动到靠近NavMesh表面的位置。

5.选择角色，找到MoveTo脚本，并将球体分配给Goalproperty。

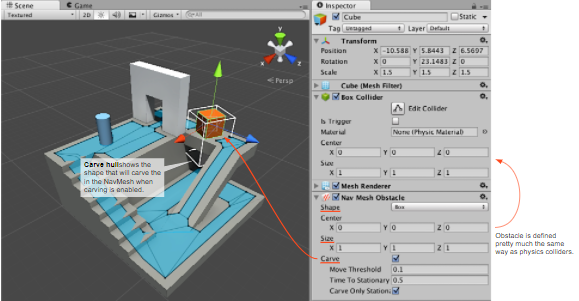
6.按下播放; 您应该看到代理程序导航到球体的位置。

综上所述，在脚本中，您需要获取对NavMesh代理组件的引用，然后设置代理运行，只需将位置分配给其目标属性即可。 导航如何使用NavMesh Agent为您提供有关如何解决常见游戏场景的更多示例。

## 创建导航阻挡

NavMesh障碍物组件可用于描述代理在导航时应避免的障碍。 例如，代理应该避免物理控制对象，例如移动时的板条箱和桶。

我们将添加一个箱子来阻止关卡顶部的通路。



1.首先创建一个立方体来描述箱子：Game Object > 3D Object > Cube。

2.将立方体移动到顶部的平台上，立方体的默认尺寸对于一个箱子是有益的，因此请保持原样。

3.将一个NavMesh障碍物组件添加到立方体。 从检查器中选择添加组件，然后选择 Navigation > NavMesh Obstacle。

4.将障碍物的形状设置为盒子，更改形状将自动将中心和大小适合渲染网格。

5.向障碍物添加刚体。 从检查器中选择Add Component，然后选择Physics> Rigid Body。

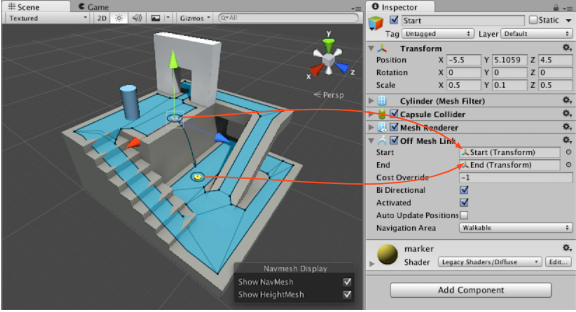
6.最后从NavMesh Obstacle检查器开启Carve设置，以便代理知道在障碍物周围找到路径。

现在我们有一个物理控制的盒子，AI在航行时知道如何避免。

## 创建导航网格链接

Off-Mesh Links用于创建跨越可行走的导航网格表面的路径。 例如，跳过沟渠或围栏，或在穿过它之前打开门，都可以被描述为离网链接。

我们将添加一个Off-Mesh Link组件来描述从上层平台到地面的跳转。



1.首先创建两个柱面：Game Object > 3D Object > cylinder.

2.您可以将圆柱缩放到（0.1,0.5,0.1）以便更容易地使用它们。

3.移动顶部平台边缘的第一个圆柱体，靠近NavMesh表面。

4.将第二个圆柱体放在地面上靠近NavMesh的地方，该地点应该位于链路应该着陆的位置。

5.选择左侧的圆柱体并添加一个离网连接组件。 从巡视器中选择添加组件，然后选择 Navigation > Off Mesh Link.。

6.在开始字段中分配最左侧的圆柱，在结束字段中分配最右侧的圆柱。

现在你已经开始运作 Off-Mesh Link了！ 如果通过 Off-Mesh Link的路径比沿着导航网走过的路径短，则将使用 Off-Mesh Link。

您可以使用场景中的任何游戏对象来保持关闭Off-Mesh link组件，例如，网格预制可能包含Off-Mesh link组件。 同样，你可以使用任何具有Transform的游戏对象作为开始和结束标记。

NavMesh烘烤过程可以自动检测并创建常见的跳转和下拉链接。 请参阅Building Off-Mesh Links Automatically以获取更多详细信息。

## 自动构建导航网格链接

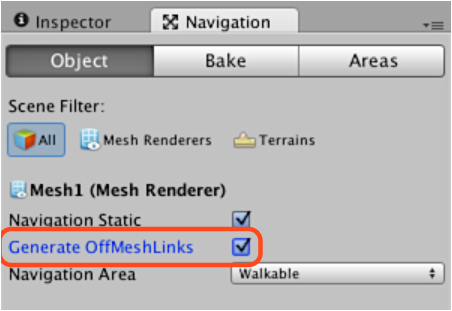
可以自动检测一些用于Off-Mesh Links的用例。 最常见的两个是：下落和跳跃。

Drop-Down links被创建为从平台掉下。

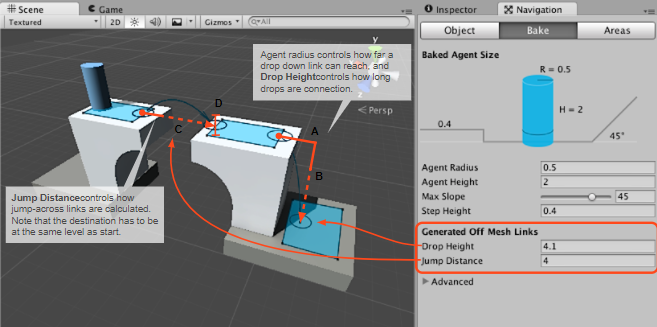
Jump-Across links 被创建为跳过缝隙

为了自动查找跳转位置，构建过程沿着NavMesh的边缘行进，并检查跳转的着陆位置是否在NavMesh上。 如果跳跃轨迹不受阻碍，则创建Off-Mesh link。Let’s set up automatic Off-Mesh Link generation. If you’re not familiar with NavMesh baking, take a look at Building a NavMesh.

让我们设置自动Off-Mesh Link生成。 如果您对NavMesh烘焙不熟悉，请查看构建NavMesh。



首先，需要标记跳转可以从哪里开始的场景中的对象。 这是通过在“Objects”选项卡下的“Navigation Window ”中检查“Generate Off-Mesh Linksoption”选项完成的。



第二步是设置下落和跳转轨迹：

Drop-Down link生成由Drop Height参数控制。 该参数控制将要连接的最高点是什么，将该值设置为0将禁止生成)

drop-down link的轨迹被定义为使得水平行进（A）是：2 \* agentRadius + 4 \* voxelSize。 也就是说，下降将会降落到平台边缘之外。 此外，垂直行程（B）需要超过烘烤设置的“步高”（否则我们可以下降）并且小于“下降高度”。 通过体素大小进行调整是为了在体素化过程中出现任何舍入误差并不妨碍链接的生成。 您应该将Drop Height设置为比您在关卡中测量的值大一点的值，以便链接正确连接。

Jump-Across链接生成由跳转距离参数控制。 该参数控制将连接的最远距离。 将该值设置为0将禁用生成。

jump-across链接的轨迹被定义为使得水平行程（C）大于2 \* agentRadius且小于跳跃距离。 另外，着陆位置（D）不得超过起始位置的水平的v哦邪恶了Size。

现在标记了对象并调整了设置，现在是时候按Bake了，并且您将自动生成 off-mesh links！ 当你改变场景并烘烤时，旧的链接将被丢弃，并且基于新的场景创建新的链接。

Trouble Shooting（故障排除）

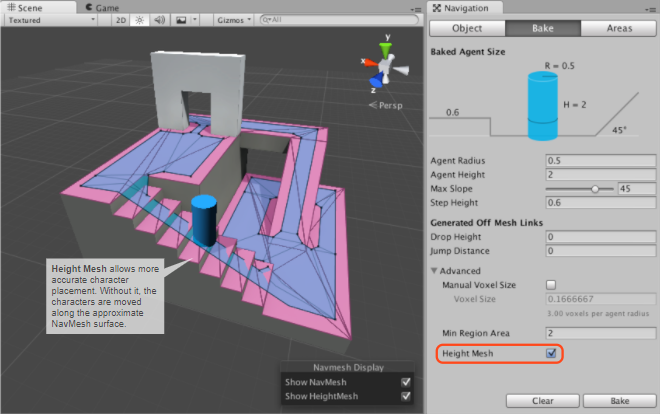
需要注意的是，如果在您希望的位置没有生成off-mesh link，请执行以下操作：）

落差高度应该比您的关卡中测得的实际距离大一点。 这确保了NavMesh烘烤过程中发生的小偏差不会阻止链接被连接。

跳跃距离应该比您的关卡中测得的实际距离长一点。 跳转距离是从NavMesh上的一个位置到NavMesh上的另一个位置测量的，这意味着您应该添加2 \* agentRadius（加上一点）以确保裂缝交叉。

## 建立高度网格（Height Mesh）

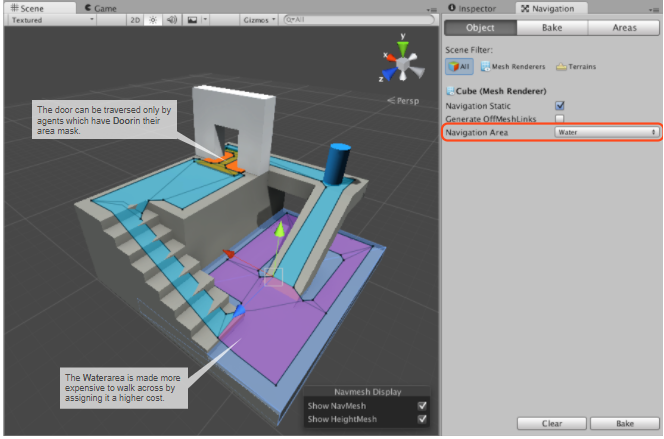
高度网格允许您将角色更精确地放置在步行表面上。



导航时，NavMesh Agent被约束在NavMesh的表面上。 由于NavMesh是可行走空间的近似值，因此在构建NavMesh时会将某些功能均匀化。 例如，楼梯可能在NavMesh中显示为斜坡。 如果您的游戏需要准确放置代理，则应在启动NavMesh时启用高度网格构建。 该设置可以在导航窗口中的高级设置下找到。 请注意，构建Height Mesh将在运行时占用内存和处理时间，并且需要花费一点时间才能烘焙NavMesh。

## 导航区域和移动成本

导航区域定义了跨越特定区域的难度，在路径查找期间较低成本的区域将是优选的。 另外，每个NavMesh Agent都有一个区域掩码，可以用来指定代理可以移动的区域。



在上面的例子中，区域类型用于两种常见用例

通过分配更高的成本，水区的走路成本更高，以处理在浅水区行走较慢的情况。

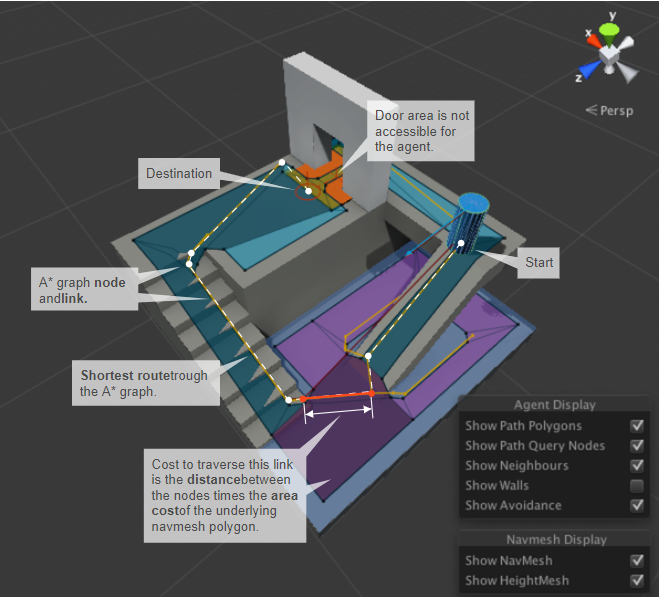
门区可以通过特定角色访问，以创建人类可以穿过门的场景，但僵尸不能。

区域类型可以分配给包含在NavMesh烘焙中的每个对象，另外，每个Off-Mesh Link都有一个属性来指定区域类型。

**Pathfinding Cost（寻路成本）**

简而言之，成本允许您控制探路者寻找路径时偏好的区域。 例如，如果将区域的成本设置为3.0，则跨越该区域的行程被认为是替代路线的三倍。

为了完全理解成本的工作机制，让我们看一下寻路路径是怎么工作的。



在寻路过程中访问的节点和链接。

Unity使用A \*计算NavMesh上的最短路径。 A \*适用于连接节点的图形。 该算法从最近的节点开始到路径开始，并访问连接节点，直到达到目的地。

由于Unity导航表示是一个多边形网格，因此探路者需要做的第一件事是在每个多边形上放置一个点，这是该节点的位置。 然后计算这些节点之间的最短路径。

上图中的黄色圆点和线条显示节点和链接如何放置在NavMesh上，以及它们在A \*期间以何种顺序遍历。

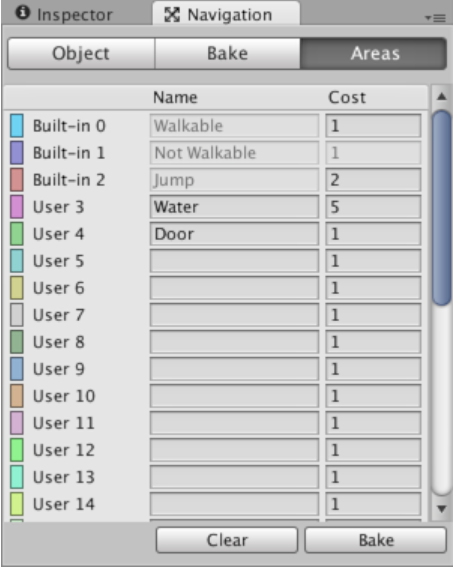
在两个节点之间移动的成本取决于行进距离以及与链接下多边形的面积类型相关的成本，即距离\*成本。 实际上，这意味着，如果一个区域的成本是2.0，跨多边形的距离看起来会长一倍。 A \*算法要求所有成本必须大于1.0。

成本对结果路径的影响可能难以调整，特别是对于较长的路径。 处理成本的最佳方式是将它们视为提示。 例如，如果您希望代理不经常使用“关闭网格链接”，则可能会增加成本。 但调整代理人更喜欢在人行道上行走的行为可能具有挑战性。

另一件你可能会注意到的问题是，探路者并不总是选择最短的路径。 这是节点布局的原因。 在大型开放区域紧挨着小型障碍物的情况下，效果可能会很明显，导致导航网格与非常大和小的多边形相关联。 在这种情况下，大多边形上的节点可能会放置在大多边形中的任何位置，从探路者的角度来看，它看起来像是绕道而行。

每个区域类型的成本可以在“Areas”选项卡中全局设置，也可以使用脚本为每个代理覆盖它们。

**Area Types**



区域类型在导航窗口的区域选项卡中指定。 有29种自定义类型，以及3种内置类型：可走，不可走和跳。

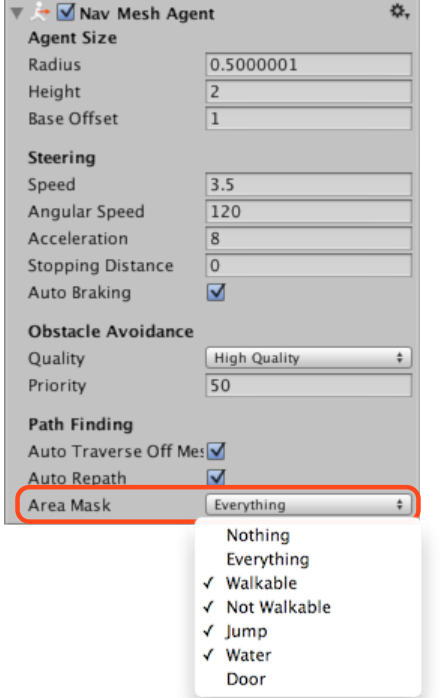
可走是一种通用的区域类型，它指定该区域可以走路。

不可走是一种防止导航的通用区域类型。 对于想要将某个对象标记为障碍物的情况非常有用，但不要将NavMesh放在其上。

跳转是分配给所有自动生成的关闭网格链接的区域类型。

如果多个不同区域类型的对象重叠，则导致的导航区域类型通常是索引最高的那个。 但是有一个例外：不可行走始终优先。 如果您需要屏蔽某个区域，这可能会有所帮助。

**Area Mask**



每个代理都有一个区域掩码，用于描述导航时可以使用的区域。 区域掩码可以在代理属性中设置，或者位掩码可以在运行时使用脚本进行操作。

当您只希望某些类型的字符能够穿过某个区域时，区域掩码很有用。 例如，在僵尸逃生游戏中，您可以用门区类型标记每个门下方的区域，并从僵尸角色的区域掩码中取消选中门区域。

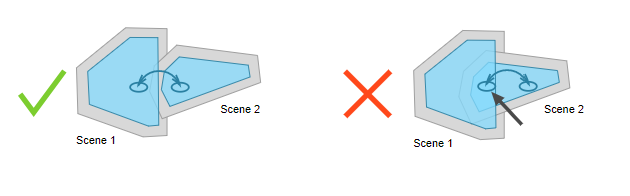
## 额外加载多个导航网格

不同场景中的NavMes默认情况下不会连接。 当使用Application.LoadLevelAdditive()加载另一个级别时，您将需要使用Off-Mesh链接连接不同场景中的导航。



在这个例子中，我们有场景1和场景2.场景1有一个Off-Mesh Link，在一个可走的区域开始，并在场景2的一个可走的区域着陆。根据需要，可以有多个连接场景的Off-Mesh Link连接。

在创作时，连接Off-Mesh links的场景的另一端点未连接。 加载新场景后，Off-Mesh links将重新连接。



如果多个场景在同一区域具有NavMesh重叠，则位置选取可以是该位置处的任意NavMesh。 这适用于代理，离网连接和使用NavMesh API进行位置选择。 您应该创建场景穿越关闭网格链接，以便它们仅在一个NavMesh上清晰地开始和结束。 重叠的NavMesh区域不会自动连接。

## 与其它组件一起使用的问题

您也可以将NavMesh Agent，NavMesh Obstacle和Off Mesh Link组件与其他Unity组件一起使用。 以下列出了将不同组件混合在一起时的注意事项。

### 导航网格代理与物理组件

1、您不需要将物理碰撞器添加到NavMesh Agent中，用来实现躲避

也就是说，导航系统会模拟代理及其对障碍物和静态世界的反应。这里的静态世界是烘焙的NavMesh。

2、如果你想要一个NavMesh Agent来推动物理对象或使用物理触发器：

a. 添加碰撞组件

b. 添加Rigidbody组件

c. 打开运动学开关，重要。

d. 运动学意味着刚体由不由物理系统直接控制

如果NavMesh Agent和Rigidbody（非运动学）在同一时间都处于活动状态，就是一种冲突状态，会发生未定义的行为。

您可以使用NavMesh代理移动例如 一个玩家角色，不需要物理组件

将玩家代理人的回避优先权设置为少量（高优先级），以便玩家通过人群刷新

使用NavMeshAgent.velocity移动角色代理，以便其他代理可以预测角色的移动以避免当前角色。

### 导航网格与动画组件

具有Root Motion的NavMesh Agent和Animator会导致竞争状况，两个组件都试图在每一帧中移动变换。有两种可能的解决方法。

信息应该始终朝一个方向流动，也就是说几个系统的工作应该有严格的顺序。比如让代理移动角色、驱动动画，或者动画根据导航结果移动角色。如果产生循环反馈，将会难以调试。

**1、动画跟随导航代理移动**

使用NavMeshAgent.velocity作为Animator的输入，以大致匹配代理的动作和动画

强大且易于实施，将导致动画无法匹配速度的情况下滑动脚步

**2、导航代理跟随动画运动**

禁用NavMeshAgent.updatePosition和NavMeshAgent.updateRotation以将导航行为与物体的位置分离。用代理计算的位置（NavMeshAgent.nextPosition）和动画根的位置（Animator.rootPosition）之间的差异来控制角色的移动。

**3、导航网格代理与障碍物**

二者不能在同一个物体上混合使用。

启用这两者都会使代理试图避开自己。

**4、导航障碍与物理组件**

如果你想让物理控制对象影响NavMesh Agent的行为，将NavMesh Obstacle组件添加到代理应该避开的对象上。

如果一个游戏对象附有一个刚体和一个导航障碍组件，障碍物的速度将自动从Rigidbody中获得。这使得NavMesh Agent能够预测和避免移动的障碍物。