**1简述NavMesh的设置方法。**

NavMesh设置方法：

1首先，游戏场景中有一个“小人”，我们希望这个小人有自动寻路的能力。

首先给他添加一个Nav Mesh Agent组件。

2Nav Mesh Agent组件相当于用一个圆柱体代替这个小人，来进行寻路计算。

最基本的，需要设置Radius和Height（代理圆柱的半径和高度），还有下面的Stopping Distance,就是距离目标多远算导航结束。

3然后在Windows菜单下找到Navigation窗口

或者在右面切换到Navigation面板（已打开）

4刚才的Nav Mesh Agent设置是设置的“小人”怎么用“地图”。

现在则要设置导航网格的计算设置。要计算导航网格，需要设置假设的物体大小，物体能上的台阶高度，物体能爬的坡度，物体能跳下的高度，物体能跳的距离。这些都设置好后，点击Bake烘焙导航网格，如图。

5要使用导航网格，最简单的写一个脚本：

1. 在Start或者Awake里面找到Nav Mesh Agent组件

2. 在Update里面运用SetDestination设置目标地点即可

**2.NavMeshAgent属性参数的使用方法**

1. Radius，Height，Base Offset，分别为Agent的半径，高度，偏移值。

在添加了NavMeshAgent组件的对象上直观的影响。

2.调节以上参数，将会影响外部的Agent线框。

Base Offset指的是Y轴上的偏移。

3.

Steering

Speed：在NavMesh上移动的最大速度

Angular Speed：角速度

Acceleration： 物体从静止到最大速度的加速度

Stopping Distance：停止距离（常与remainingDistance一起使用）

Auto Braking：是否自动减速。（不勾选的话如果速度太快，物体会冲过去目标点以一定的惯性返回目的地）

4. Obstacle Avoidance

Quality（质量）（枚举）

None ： 无

Low Quality：低质量

Medium Quality：中等质量

Good Quality： 高质量

High Quality： 高质量

Priority： 优先值，低优先值的会给高优先值的让路。

5.Path Finding

Auto Traverse Off Mesh Linked：是否自动分离网格。（勾选时到达分离点时会从该分离点移动到另外一个分离点。）（一般不勾选，因为要配套爬墙或者跳跃的动画）

Auto Repath：是否自动重新规划路线。

Area Mask：网格遮罩。（枚举，规定物体可以通过的区域）

6.在检视面板中看到的Agent参数都可以用过脚本获取并控制。

一些常用的参数。

destination属性：可以设置目标点（SetDestination(target)方法相同效果）

nextPosition:获取或设置模拟navmesh代理的位置，可以手动的驱动角色进去移动（配合动画使用）

remainingDistance：物体当前位置与目标点的剩余距离。（设置了StopDistance后会出现一个问题，到时可以用这两个参数去调节）

UpdatePosition，UpdateRotation两个值都为bool值，可以通过更改他们俩，停止使用Agent组件来驱动物体移动和旋转。

isOnNavMesh： 判断物体是否在导航网格上。

isOnOffMeshLink： 判断物体是否在分离网格链接上。

7. SetDestination(Vector3 position)： 设置目标点。

Stop(): 停止寻路。

ResetPath（）：清空当前路径。

Resume（）： 停止游戏角色进行移动，并重新开始移动（原路径）。

Wrap（Vector3 position）：将物体瞬移到目标点。

**3 寻路过程中路网烘焙过程**

1 .window中打开Navigation窗口准备烘焙地形

2.inspector界面中将场景中的障碍物勾选为静态（ navigation static）

3 Navigation窗口中进行导航网个的烘焙

Navigation中的主要参数

//Areas（区域）：相当于障碍物的标签，可以在寻路物体身上的Navmeshagent组件中的AreaMask中屏蔽某一区域

    //Bake（烘焙）：生成指定的寻路网格

        //Agent Radius： 寻路物体的半径

        //Agent Heigh： 驯鹿物体的高度

        //Max Slope： 寻路物体可前进的最大坡度（最大为60度）

        //Step Heigh： 寻路物体前进可跨过的最大高度（最大不能超过寻路物体的身高）

        //Drop Heigh： 寻路物体下落的最大高度

        //Jump Distance 驯鹿物体跳跃的最大距离

   //Object（对象，障碍物）：设置指定游戏物体的寻路烘焙属性

   //Navigation Static:设置当前游戏物体为寻路静态状态

   //Generate offMeshLinks：是否生成网格连接，如果勾选，网格之间会指定跳点烘焙

   //Navigation Area：设置当前游戏物体的导航区域

4.设置完成后，直接点击Bake场景中会生成导航网格。

5.生成导航网格后，在需要自动导航的物体上添加NavMeshAgent组件

其中几个主要参数:

Agent Type ：导航代理的类型，这个参数下一节会用到

Base Offset：导航代理和当前游戏物体位置的差值

speed：移动速度

Angular Speed:转向速度

Acceleration：移动加速度

Radiuse： 导航代理的半径

Heigh：导航代理的高度

6接下来在代码中动态设置导航的终点

使用NavMeshAgent类中的函数SetDestination(Vector3 Target);函数来指定导航的终点就可以了

**4 对于寻路过程中的障碍物绕行应该怎么处理：**

一、重新计算路径

我们希望游戏世界随着时间改变。一条一段时间之前发现的路径，可能不再是现在的最优路径。用新的信息更新旧路径是值得的。下面列出的是一些用来判断决定是否需要重新计算路径的标准：

1 每N步一次：这样保证用来计算路径的信息不会旧于N步。

2 当额外的CPU时间可用时：这样可以实现路径质量的动态调整。即使使用了更多的游戏单位，或者是在一台较慢的电脑上运行游戏，每个游戏单位的CPU使用率都可以降低。

3当游戏单位转弯或者通过一个关键路径点的时候。

4当游戏单位附近的世界发生改变的时候。重新计算路线的主要缺点在于有很多路径信息被丢弃了。例如，如果路径长100步，并且每十步进行一次重新计算，那么路径的总步数是100+90+80+70+60+50+40+30+20+10 = 550。对于一个长M步的路径，总共大概进行了M^2步计算。因此，如果你想要得到很多条长路径，重新计算路线并不是一个好主意。重复使用路线信息，而非丢弃，这样会是更好的办法。

二、路径剪接：

路径剪接

当一条路径需要被重新计算时，意味着世界正在改变。给定一个变化中的世界，地图上的邻近部分比远处的部分更好了解。我们可以遵循一个局部修正策略：找到附近的一条好路径，并且假定较远的路径直到我们靠近它了才需要重新计算。我们可以仅仅重新计算路径的前M步，而不是整条路径：

1 设p[1]..p[N] 是路径剩余部分（N步）

2 计算一条从p[1]到p[M]新路径

3 剪接新路径到旧路径中，通过移除p[1]..p[M]然后插入新路径到这些空位置上。

因为p[1]和p[M]相距不到M步，所以新路线不太可能长。不幸的是，像新路径又长又不是非常好的情况，也有可能发生。上图展示了这样一个情况。原始的红色路径是1-2-3-4，棕色区域是障碍。如果我们到达2然后发现从2到3的路径被阻挡了，路径剪接会用绿色的路径2-5-3来替换2-3，导致这个单位循着路径1-2-5-3-4移动。我们可以看到这并不是一个好的路径，蓝色路径1-2-5-4是更好的选择。不好的路径经常可以通过计算新路径的长度来判断。如果它明显比M长，它就可能是不好的路径。一个简单的解决方法，给寻路算法添加一个上限（最大路径长度）。如果找不到一条短的路径，这个算法返回一个错误代码，在这种情况下，使用重新计算路线而不是路线剪接来得到一条像1-2-5-4这样的路径。

对于没有涉及到这类情况的例子，对于一条N步的路径，路线剪接会计算2N到3N路径步数，取决于一条新路径进行剪接插入的频繁程度。这是一个相对低的代价，使得算法能对世界的改变作出反应。意想不到的是，这个花费的代价大小取决于M，也就是进行剪接的路径步数。M控制一个反馈和路径质量的平衡，而不是CPU时间的影响。如果M的值较大，单位的移动将不会快速地对地图的改变作出反馈。如果M太小，被剪接的路径可能太短，以至于不能得到可以绕过障碍的替换路径：更不优的路径（例如1-2-5-3-4）可能会被找到，尝试M的不同取值和不同的剪接标准（例如每隔3/4 M步），来看看怎样做最适合你的地图。

路径剪接比重新计算路径明显地快了许多，但它对于路径的重大改变并不能很好地应对。不过很多这种情况可以容易地发现，并直接使用重新计算路径来代替路径剪接。它同样有几个可以调整的变量，比如M和进行新路径的寻找的时间，所以它可以被调整为适合不同的情况（即使是在运行的时候）。但是路径剪接并不能处理游戏单位需要确定位置进而来互相穿过的情况。

三、监视地图的改变：

选择重新计算全部或部分路径在特定的时间间隔，是对地图的改变来触发重新计算。地图可以分成不同的区域，每个游戏单位可以在特定的区域表现出兴趣。（所有包括部分路径的区域都可能是感兴趣的，或者仅仅是邻近的包含部分路径的区域）无论障碍进入或离开某个区域，那个区域就标记为已经改变，然后所有对那个区域感兴趣的游戏单位都会被通知，所以路径可以在考虑障碍发生变化这一前提下被重新计算。

这一技术有很多可能的变化。例如，我们可以仅仅在特定的时间间隔通知游戏单位而不是立即通知。并且多次改变可以被组合成一次通知，所以不再需要过多的进行重新计算路径。另一个变化是让游戏单位来查询地区的状态，而不是让地区来通知游戏单位。

**5 NavMeshAgent组件的代理器移动到给定目标点需要利用哪几个函数？该函数有几个参数？其含义分别是什么含义？**

函数:

1.CalculatePath

类型：bool CalculatePath(Vector3 targetPosition, NavMeshPath path)

参数：

targetPositon 路径要求的重点位置。

path 作为结果的路径

CompleteOffMeshLink

类型：void CompleteOffMeshLink()

2

结束当前OffMeshLink上的移动。

代理会移动到离当前OffMeshLink另一端最近的有效的位置。

如果代理不在一个OffMeshLink上，CompleteOffMeshLink()没有任何效果。

当autoTraverseOffMeshLink不可用时，代理会暂停在一个off-mesh link上直到这个函数被调用。当用户自己控制通过OffMeshLinks时会用到它。

3FindClosestEdge

类型：bool FindClosestEdge(out NavMeshHit hit);

参数：

hit 掌握定位结果的性能。

4.GetAreaCost

类型：float GetAreaCost(int areaIndex)

参数：areaIndex 区域索引

5.Move

类型：void Move(Vector3 offset)

参数：

offset 相对的移动向量。

6 .Raycast

类型：bool Raycast(Vector3 targetPosition, out NavMeshHit hit)

参数：targetPosition 将要移动到的终点

hit 被射线发现的障碍物的属性(如果有)

7.SamplePathPosition

类型：bool SamplePathPosition(int areaMask, float maxDistance, out NavMeshHit hit)

参数：areaMask 一个位域标志，指定了寻路是哪个区域可以通过。

maxDistance 超出这段距离则终止搜索路径。

hit 保存结果地理位置的属性。

8.SetAreaCost

类型：void SetAreaCost(int areaIndex, float areaCost)

参数：areaIndex 区域索引

areaCost 指定的区域的新的消耗

9.SetPath

类型：bool SetPath(NavMeshPath path)

参数：path 新的寻路路径

10.Stop

类型：void Stop()

停止代理沿着当前路径移动。