**第二次作业**

**1.简述NavMesh的设置方法。**

答：在你的场景中构建NavMesh只需要4个步骤：

1. 选择场景中需要生成寻路的几何体-可行走表面和障碍物。

2. 在NavMesh面板中选择需要烘焙寻路的物体，检测是否勾选Navigation Static.

3. 根据你的agent大小来调整bake 面板的设置。

1. Agent Radius : agent可以距离墙体 ，窗户或边缘多近的距离。
2. Agent Height : agent可以通过的最低的空间高度。
3. Max Slope : agent可以直接行走上去的最小坡度。
4. Step Height:  agent可以踩上（走上）的障碍物最高高度。

4. 点击bake按钮烘焙NavMesh。

**2.简述NavMeshAgent属性参数的使用方法。**

答：agent.updateRotation = false; //不允许NavMesh来旋转角色

agent.updatePosition = true; //允许NavMesh来移动角色

agent.velocity.magnitude 这个也是速度,

GetComponent().SetFloat(“Speed”, agent.velocity.magnitude);speed 移动速度

Angular Speed 转角速度，转身速度角速度： 最高转速（度/秒）。

Acceleration 加速度,启动时的最大加速度。

Stopping Distance 停止距离 ,制动距离：制动距离。到目的地的距离小于这个值，代理减速。

Auto Traverse OffMesh Link 自动遍历OffMesh链接：自动移动并关闭OffMeshLinks

Auto Repath 自动重新寻路：如果现有的部分已失效，获得新的路径。

Height 高度：代理的高度（用于调试图形）。

Base offset 基本偏移：碰撞几何体相对于实际几何体垂直的偏移。

Obstacle Avoidance Type 障碍躲避类型 ：躲避的质量水平。

NavMesh Walkable 导航网格行走：指定代理可以遍历的导航网格层类型。

**3.简述寻路过程中路由烘焙过程。**

答：1、在window中打开Navigation窗口准备烘焙地形

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

3、在Navigation窗口中进行导航网格的烘焙

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.对于寻路过程中的障碍物绕行应该怎样处理？**

答：一个寻路算法会围绕静止的障碍物计算路径，但是如果障碍物移动了呢？一个单位到达一个指定点的过程中，一个障碍物也许已经不在那了，或是一个新的障碍物出现在那。如果这种典型的障碍可以绕过，使用一个单独的避障算法在你的探路者上。这个探路者会找到所需路径，然后按照它来绕过障碍物。但是如果障碍物导致路径发生明显的改变，则考虑使用探路者来避开障碍物。

**重计算路径**

随着时间的推移，我们期望游戏世界有所改变。一条路径被发现前可能不再是最佳路径。按照新的信息对旧的路径更新更具有价值。下面列出了一些标准，有助于确定何时需要重新计算：

每N步：保证用于计算路径的信息不比N步旧。

每当额外的CPU时间可用时：这允许路径质量的动态调整；像更多的单位部署在游戏中，或是游戏运行在较慢的电脑上，每个单位的CPU占用绿可以被降低。

每当单位转一个弯或通过一个航路点时。

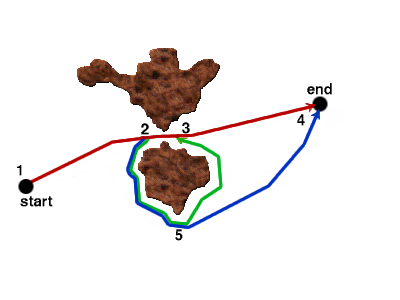
每当单位附近的世界发生了改变时。

重新计算路径的主要缺点是大量的路径信息被丢掉。例如，有条100步长的路径，每10步就需要重新计算，路径的总步数是 100+90+80+70+60+50+40+30+20+10=550。对于一条M步的路径，随着时间大约M^2的路径步数被计算。因此，如果你期望得到很多长路径，重新计算路径并不是一个好办法。相对于把它扔了重用路径信息反而更好。

**路径拼接**

当一条路径需要被重新计算时，这意味着世界正在改变。对于不断变化的世界，对附近地图区域的了解比地图远处区域更多。我们可以遵循一个本地维修策略：在附近找到一条好的路径，并假设远处的路径不需要重新计算直到我们靠近他。与重新计算整个路径不同，我们可以先重新计算M步：

设p[1].. p[N]为路径的其余部分（N个步骤）

从p[1]到p[M]计算一条新的路径

把这条新的路径拼接到老的路径上，用新的路径代替p[1]..p[M]

因为p[1]和p[M]比M步少，看上去新路径不太可能会很长。不幸的是，新路径很长并不怎么好的情况是可能出现的。上图显示了这样一种情况。原始是红色路径1-2-3-4；如果我们到达2时发现2-3的路径被封锁了，路径拼接将使用绿色路径2-5-3代替2-3，结果会沿着路径1-2-5-3-4移动。我们能看出这条路径并不好；显然是蓝色路径1-2-5-4更好一点。

通过观察新路径的长度能够检测到坏的路径。如果他明显比M长，他可能是坏的。一种简单的解决方法是给这个寻路算法加一个限制（最大路径长度）。如果没找到一条短的路径，算法就返回一个错误代码；在这个例子中，使用重新计算路径法代替路径拼接法得到路径1-2-5-4。

对于不涉及这些的情况，一条N步的路径，路径拼接会计算2N到3N的路径步数，这取决于拼接新路径的频率。这个应对世界改变的能力代价向当低。令人吃惊的是这个代价于拼接步数M无关。M控制权衡响应速度和路径质量，而不影响CPU时间。如果M很高，单位的移动将不会对地图的改变做出快速的响应。如果M太低，拼接出的路径可能太短以至于不能绕过障碍物；更多的次优路径（例如1-2-5-3-4）会被找到。尝试不同的M值和不同的拼接标准（例如每隔3/4M步），看看什么适合的地图。

路径拼接明显比路径重新计算快，但他没有很好地回应路径的主要变化。可能会发现许多这种情况并使用重新计算路径替换。也可以调整几个变量，例如M和找到新路径的时机，以对不同条件作出调整（甚至在运行时）。路径拼接也不适用于那些单位必须相互间协调通过的情况。

实现注意：

反向保存路径：更容易地删除路径的开始部分并用不同长度的新路径拼接，因为这两个操作都将在数组的末尾进行。本质上你可以把这个数组看成是堆栈因为顶部的元素总是下一个要使用的。

**监视地图变化**

替代在一定时间间隔内重新计算所有或部分路径的一种方法，是在地图改变时触发一次重计算。地图可以被划分成多个区域，并且每个单位都可以对某些区域感兴趣。（包含路径部分的所有区域，或仅仅是包含路径部分的邻近区域。）每当障碍物进入或离开一个区域，这个区域会被标记为改变，并且所有对这个区域感兴趣的单位都会被通知，使路径可以被重新计算，以考虑障碍物的改变。

这种技术存在许多变种。例如，只在固定的时间间隔通知单位，而不是立即通知。许多变化可以被归纳到一个通知，所以不需要过多的重新计算路径。又如用单位轮询区域代替区域通知单位。

在地图上的障碍物不发生改变时，监视地图变化可以避免单位重新计算路径，所以如果你有许多区域不会经常变化，就可以考虑这种方法。

**预测障碍物移动**

如果障碍物的移动可以被预测，就有可能为寻路考虑到障碍物的未来位置。一个诸如A\*的算法有一个代价函数来确定通过地图上某一点的难度。A\*可以被修改来追踪到达某一点所需时间（由当前路径长度决定），并且这个时间可以被传入到代价函数。这个代价函数可以考虑到这个时间，并在那个时间使用预测的障碍物的位置，通过确定地图上的区域是否能够通过。这个修改并不完美，但是，因为它并不考虑在某个点等待障碍物自动离开的可能性，同时A\*并不是用来区分相同路线的路径的，而是不同的路线。

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

答：NavMeshAgent.Move 移动

NavMeshAgent.acceleration 加速度

NavMeshAgent.angularSpeed 角速度

NavMeshAgent.areaMask 区域遮挡

NavMeshAgent.autoBraking 自动制动

NavMeshAgent.autoRepath 自动重新获取路径

NavMeshAgent.desiredVelocity 需求速度

NavMeshAgent.destination 目的地

NavMeshAgent.FindClosestEdge 寻找最近边缘

NavMeshAgent.hasPath 有路径

NavMeshAgent.height 高度

NavMeshAgent.isOnNavMesh 是否在导航网格上

NavMeshAgent.steeringTarget 转向目标

NavMeshAgent.hasPath 有路径