NavMesh 是一种基于凸多边形网格的寻路，其整个寻路流程至少分为导航网格构建（bake）和寻路算法两个部分。

导航网格构建：

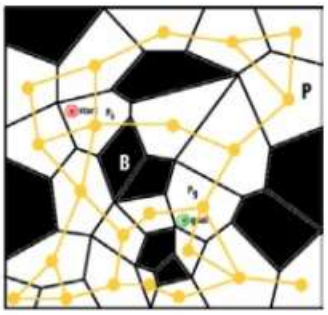

Figure.2 Transform polygons to nodes

黑色是不可通过的地块，黄点是每个地块的中心点，黄线是将每个可通过地块的中心点进行了连接。连接条件是：如果两个可通过地块有邻边，则连接

寻路算法：

如图，如果我们要从红点走到绿点，应该怎么寻路？首先更一般化地推广，若要对任意一对点（起点和终点）寻路，无非以下三种情况：

1. 两个点在同一个地块内，走直线；
2. 其中一个点不在合法地块内，此路不通；
3. 两个点在不同地块，需要寻路。

显然，回到图中来，我们要解决的是第三种情况，这也是大多数时候我们要解决的、最感兴趣的情况。针对处理后的地块网格，一个最简单的寻路方法是：A*寻路。

NavMesh 的生成过程，由 NavMeshGenerator 类提供接口。其基本流程如下：

体素化场景 - 将提供的源几何数据（通常是一整个场景，或关卡，对于动态生成世界的解决方案，由另一种生成算法来完成，超出这篇文章的讨论范围）通过体素化方式建立实体 Span，由 Heightfield 描述。

创建 Regions（区域） - 将 Heightfield 描述的实体 Span 转换为由 Floor(地板)和 Ceilling(天花板)定义的潜在可跨越的数据，虽然也可用 Heightfield 描述，但为了尽可能独立化，这里用另一个类似的 OHeightfield 类来存储。

创建 Contours（轮廓） - 检测 OHeightfiled 的可跨越 Span，将其生成为简单的轮廓数据。

创建 Polygon - 将轮廓转换为凸多边形（许多算法都假设在凸多边形基础上，因此，将复杂的轮廓转为多边形是简化问题的关键）。

创建 DetailMesh - 多每个凸多边形应用类三角剖分算法和高层信息（高度 y 或深度 z，取决于你使用的坐标系的 up 轴），将其转换为通用图形结构。