当我第一次在游戏中看到那些非玩家角色顺畅无阻地通过某些地形，到达指定的目的地，或是怪物们能在绕开障碍物的情况下追逐玩家，我觉得这真的很神奇：这些角色知道沿着什么样的路径前往目的地，就像人类一样。它们行走的路径可不是定死在程序中的，而是随着周围环境的改变而发生变化的。那么到底是什么让它们栩栩如生，好像会思考一样呢？这就是寻路算法(PathFinding)在起作用。

寻路算法，在计算机学科中是一种找寻两点之间最佳路径的算法。这句话说得似乎很简单，但是这包含了非常丰富的内容。最佳路径如果按照距离来衡量的话，那么两点之间的直线段就是最佳路径。但是真正的游戏开发中往往没有这么计算，原因是：在游戏环境中存在着很多不能穿过的障碍物，所以很少有直接走一条直线就到达目的地的情况；另一个比较重要的原因就是取两点之间的直线距离涉及到了开方计算，这是非常影响算法效率的，虽然现在的计算机运算速度都非常卓越，但是寻路算法的效率，仍是一个占有重要地位的因素（实际上目前流行的A星寻路算法正是因为其寻路效率高而受到人们青睐），现在人们对于该算法的空间开销不太在意，所以在能成功到达目的地的前提下，时间复杂度就是衡量寻路算法优劣的主要标准。

由于寻路算法在游戏开发中应用比较广泛，并且寻路算法的效率会很大程度上影响用户的游戏体验，再者不同类型的游戏包含的不同场景对寻路算法也有很大的影响，所以怎么开发出一个高效地算法是一个比较重要的问题。本论文将对当前主流的几种寻路算法：深度优先搜索算法（DFS）、广度优先搜索算法（BFS）、迪杰斯特拉算法（Dijkstra）、A星算法（A\*）等进行正确率和效率地研究与比较，并且模拟几个常见但是复杂的游戏场景，以此改进寻路算法。

本论文将使用Visual Studio 2010作为程序部分的开发工具，以及C++作为开发语言。首先将列出各寻路算法的伪代码，并用C++实现该算法，计算理论上该算法的最好情况和最坏情况下的时间复杂度O(n)，作为初步选用算法的大体参考。其次给出不同的游戏场景，讨论各算法在这些场景中的应用结果与效率。给出的游戏场景有：无障碍物2D场景、路径中存在障碍物的2D场景、复杂多单位无障碍物的2D场景、复杂多单位有障碍物的2D场景等。

对于上面介绍过的几种寻路算法，首先将使用简单的图形程序演示算法运行的概况。例如将一个二维场景平均划分成100\*100的方格，每个方格的单位长度都是1，将起点设定在(2,5)的位置，将终点设置在(76,82)的位置，场景的边界不可越过，在场景中可以设定一些方格不可越过。这样就模拟了一个简单的游戏场景，从起点起，将寻路算法运算“行走”的路径用不同于方格的颜色标记出来。这样等寻路结束后，就可以观察每种算法经过方格的数量来简单地衡量算法的效率。当然对于每种算法，必须约定场景不变。由于大部分游戏的寻路在同一时间同一目标点只牵扯到一个物体的寻路，而即时战略类的游戏则不同，由于玩家可操控的目标较多，而且可以控制多目标同时前往同一目标点，那么这时寻路就必须要考虑到动态的障碍物以及碰撞带来的影响，这部分将作为寻路算法在复杂游戏场景中改进来研究。

其后，对于上面提到的几类游戏场景，分别编写一个代表性的游戏段，即只有涉及到寻路算法的那一块，但是具备正常的游戏元素与用户体验。首先，对于无障碍物的2D游戏场景，这种情形最简单，可以认为就是起点与目的地之间的连线就是最佳路径（在方格图中就是只要起点到目的地的横纵直线距离相加最少的路径都是最佳路径），为了增加寻路过程的复杂性以展示各种算法的可用性，我将设定不同区域移动的消耗不同，通过这项设定，寻路算法将会产生路径，但是是不是最佳路径以及耗费时间的多少，将会决定算法的优劣。其次，对于有障碍物的2D游戏场景，可以说大部分2D游戏都包含了类似的场景，我将使用连连看来模拟各种寻路算法的可用性以及高效性。连连看其实非常具有代表性，因为它本身就是一个方格图，不需要我们去另外划分它的区域；其次，它的游戏规则也就是不断地从起点连向目标点，只不过增加了一些规则。由于我要使用连连看来演示各种寻路算法的效率，故我会将程序改为自动完成所有连接步骤，即不用我们自己去找，然后记录下来完成的总时间来大体代表一个算法的效率。为了保证结果的可靠性，对于每种算法的演示，将使用同样的随机排列。总进行3次比较，取完成时间的平均值作为衡量算法效率的结果。再次，考虑复杂多单位无障碍物的2D场景，我想对于寻路算法增加的考验就是增加了动态的障碍物，那么前一次得到的路径将很有可能不再是最佳路径甚至是无法到达目的地。简单的说，就是一开始沿着起点与目标之间的直线走向目标（为了方便而取直线），当途中的某一刻，有另外一个单位移动到了路径中，并且将要和原单位相碰撞，导致原单位无法继续前进，或许有人会说仅仅让原单位原地等待，直到可以行走为止，那么如果另一单位就卡死在了路径中将会导致原单位永远无法到达目的地，所以这时，我们需要添加策略，可以让原单位增加一个垂直于当时路径方向的速度，让其主动离开另一单位，并重新寻路，找寻最佳路径。这时，增加的新策略将会是影响寻路效率的关键因素，这里我将展示一个类似于即时战略游戏的场景，其中会有多个单位可以随意移动或者停止，并且移动速度与碰撞体积均有所不同，来展示改进的寻路算法的可行性以及效率。最后，对于多单位有障碍物的2D场景，其实只是在原来无障碍物的基础上，增加了一些静态的不可跨越的边界，虽是如此，但是可能会造成原单位即使向旁边偏移也无法达到目的地的情况，即被其他单位和障碍物一起堵死，这种情况我们也要争取避免，这里我将在原有的即时战略游戏场景中增加一些墙壁或者什么的障碍物来演示。关于此处算法的改进，就需要增加一个预判断，即在单位偏移的过程中，要预判该偏移是否可能造成自己被堵死，即观察其他单位的移动特征，从而改变偏移方向或者不变。这种需要吸收其他单位信息的寻路算法需要维护一张全局的单位信息表，并且从表中找寻可能会影响自己的那些单位，进行预判断。这种思路在小范围的寻路是可行的，大范围单位数量增多就需要进行进一步的研究改进，这里留作论文的深度研究内容。

寻路算法是人工智能的一部分，也是非常基础的一部分。给寻路增加一些额外的条件虽然会增加寻路的难度，但是却会增加游戏的可玩性。比如说，一些游戏中，怪物是有视野范围的，或者说没有探索过的地形，它们也不知道那里是什么样子的。如果玩家位于这些地形之内，怪物是无法准确定位玩家并且朝着玩家追来，这是不合理的。所以我们需要给电脑和玩家各维护一张可行动区域表，记录着可以移动的区域大小，这样在怪物寻路时就不能直接定位到玩家，而是会在可行动区域中徘徊探索，不断扩充可行动区域，直到找到玩家，当然也可以等待直到玩家自己走到它的行动区域后去寻路（事实上大多数游戏都是后者）。这里我也会写一个例子来演示在寻路算法中增加了“视野”的条件。