开篇

这篇文章介绍找最短路径的一种算法，它的字我比较喜欢:启发式搜索。

标题上写的是翻译，只是觉得原文讲解的思路很清晰。这篇文章整体构思和原文相差不多，只是有些地方有小的改动，

我想的是用更容易理解的方式、更简洁的把A*算法的思想呈现出来。

文章中出现的词openlist，closelist我觉得用原文会更好故没有翻译，在文中会有解释。

原文地址http://www.gamedev.net/page/resources/_/technical/artificial-intelligence/a-pathfinding-for-beginners-r2003

各位也可以直接参考原文。

网上关于A*算法的文章还有很多，只是那些都需要有一定的基础，对于入门的好文章不多，而这篇文章就是为初学者而写的，很适合入门的一篇。文章定位：非专业性A*文章，很适合入门。

有图有真相，先给大家看个效果图吧：从图的左下角到右上角寻找最短路径，灰色部分是障碍物。

这是用一般的搜素方法，类似穷举的效果

下面的图是用A*搜素的效果，也就是本文要介绍的算法。

可以看出，用A*算法减少了许多计算量，它的效率有了显著的提高。

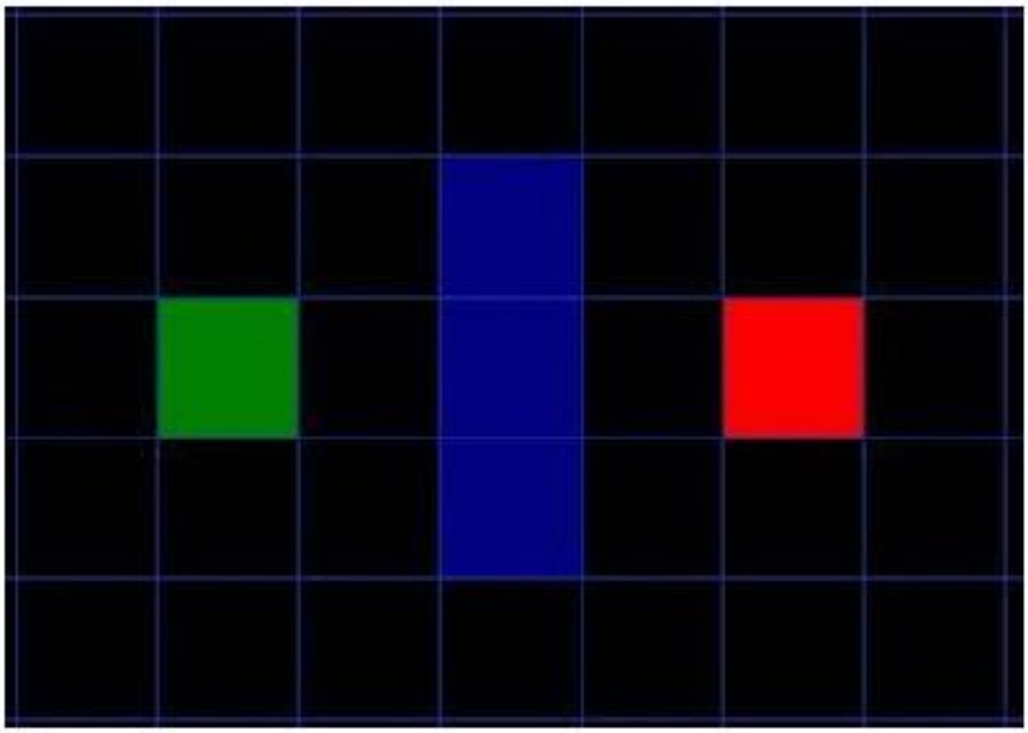
下面将为你解答上图中的算法是如何实现的。

图片来源：http://en.wikipedia.org/wiki/A*_search_algorithm

正文

搜索区域介绍

下图是这篇文章讨论的中心：



[Figure 1]

图中左边的绿色点是搜索的起点A，目标点是右边的红色点B，中间被障碍物挡住（蓝色部分）。

我们的目的是从起点出发，找到一条到达目标点的最短路径。

把整个图都分为一个个小方块只是为了这样方便讨论，更多的应用中，也可以把图分为其它方块的组合。

开始搜索

目标是找出从A点出发到B的最短路径，所以我们从A点开始搜索，直到找到目标B。

搜索的步骤是这样的：

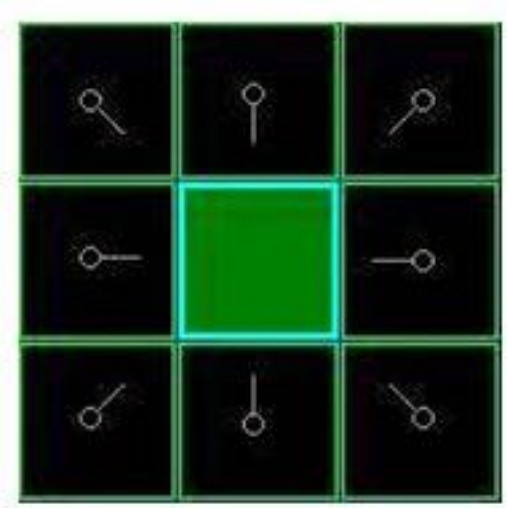
1、从起点A开始把A加入到openlist中。openlist解释：它是一个队列，里面元素是一些方块，它们有可能构成最短路径。现在队列中只有元 素A，以后会加入更多的元素。以后会对里的元素进行检查，从里面来找到构成最短路径的元素。

2、看起点A周围的元素是否可达（是否能从A到达它们）把从A可到达的元素加入到openlist中，并且加入到openlist中的节点维护一个指指针，指向他的

父亲，也即A点。如果A周围有障碍物就忽略它。从这个图看， A周围把个元素都可达，所以把它们都加到openlist中。

3、把起点A放入closelist中，在closelist中的点意味着以后不需要再去考虑它了。对于A节点，A可达的点都加入到了openlist中，以后也就不用考虑A的情况了。

经过以上三步操作后的效果图如下所示



[Figure 2]

图中被暗绿色包围的就是openlist中的点，一共八个，都是从起点A可达的点，并且他们中的每个都有一个指向他们父节点的指针（图中的小针方向）被高亮绿色包围的表示closelist中的点，可以看出起点A已经在closelist中。

路径选择

从起点出发 ，下一步可以走的点现在有八个，选取哪一个作为下一步的点呢?正常的思维是选取一个离目标值最进，且在这些点中离远点最近的点。

本文的思路也是这样的，文中用

F = G + H

表示，其中：

对于每个点，都有自己的G、H、F。

其中G表示从特定的点到起点的距离，H表示从该点到目标的估值，那么F就是经过该点路径的估值。

下面详细介绍

G:从起点到特定节点的距离，也就是G的父节点加上从G的父节点到起点A的距离g。图中是边长为10的正方形块，所以就是G的父节点的值g

加上10（上下左右相邻）或者加上14（斜块相邻、也就是对角线的长度，本来是14.14、、为了方便计算这里取近似值）

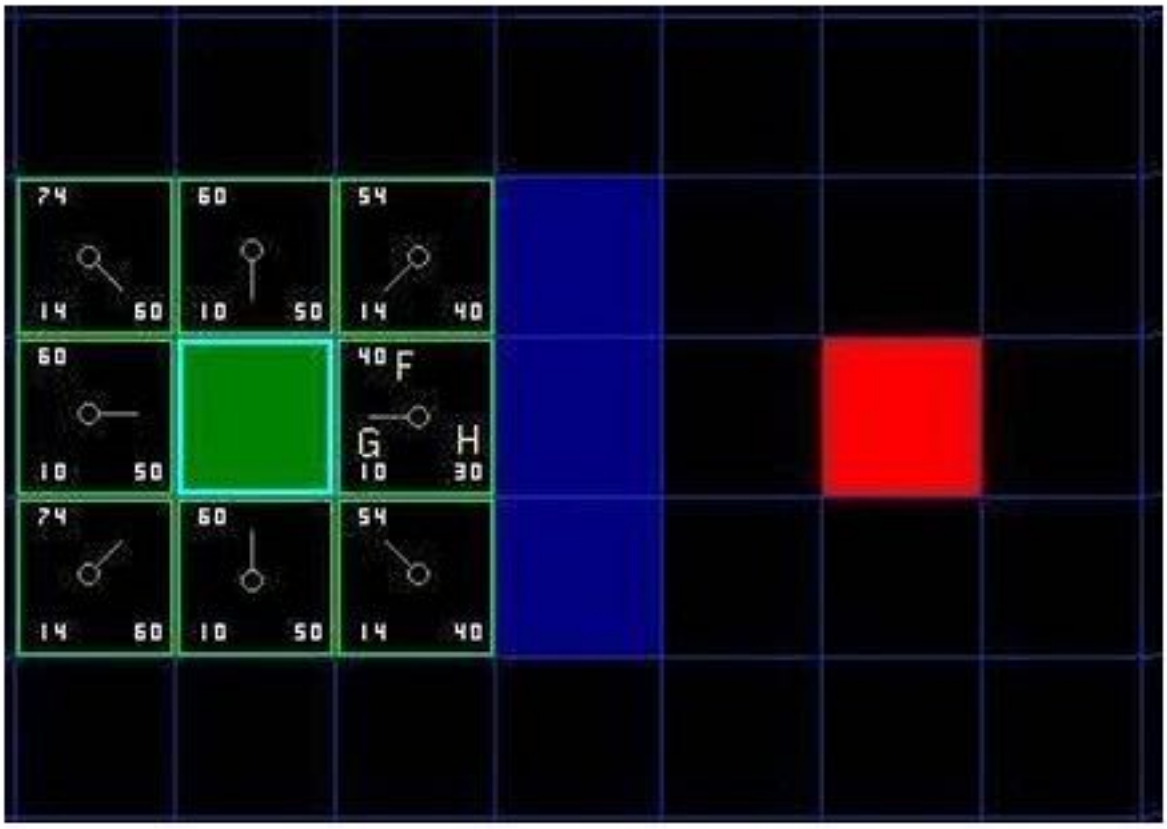
H：H能用很多方法得到估计值.这里用到的方法称为Manhattan method，H的值就是从考虑的点通过水平和垂直移动达到目标点的移动步数乘10（正方形块的边长为10）.注意只是水平和垂直移动，不走斜线。并且忽略图中的障碍物。

插一句：

看了对H的描述，你可能会怀疑这种估计的精确性，有一点是可以肯定的：估计值越接近真实值，算法就能更块的找出最短路径。我们用的这种方法确实是做了估计，只是这种估计准确性不高，就是说只是粗略的估计，因为这种方法容易理解，所以才采用这种方法。可以想到，太过接近的估值最后不一定能得到想要的结果。关于估值函数想了解更多请参见：http://www.policyalmanac.org/games/heuristics.htm

为了从openlist中选取一个点继续搜索，就要计算出openlist中的每个点的F、H、G的值然后选取F小的一个点，进行下一步的探索。

对于上图中的点，他们的F、G、H的值在图中都有标明。



[Figure 3]

F、H、G的位置在起点右边的点中已经有标注，其他点的位置同理。

现在看起点右边的点（也就是标有字母的点）G=10，因为在起点正左边。H=30，水平移动三个格子可以到目标点B。F=G+H=40

继续搜索

由于我们的目的是找最短路径 ， 下一步就从openlist中选取F最小的点做进一步的搜索，按如下步骤进行：

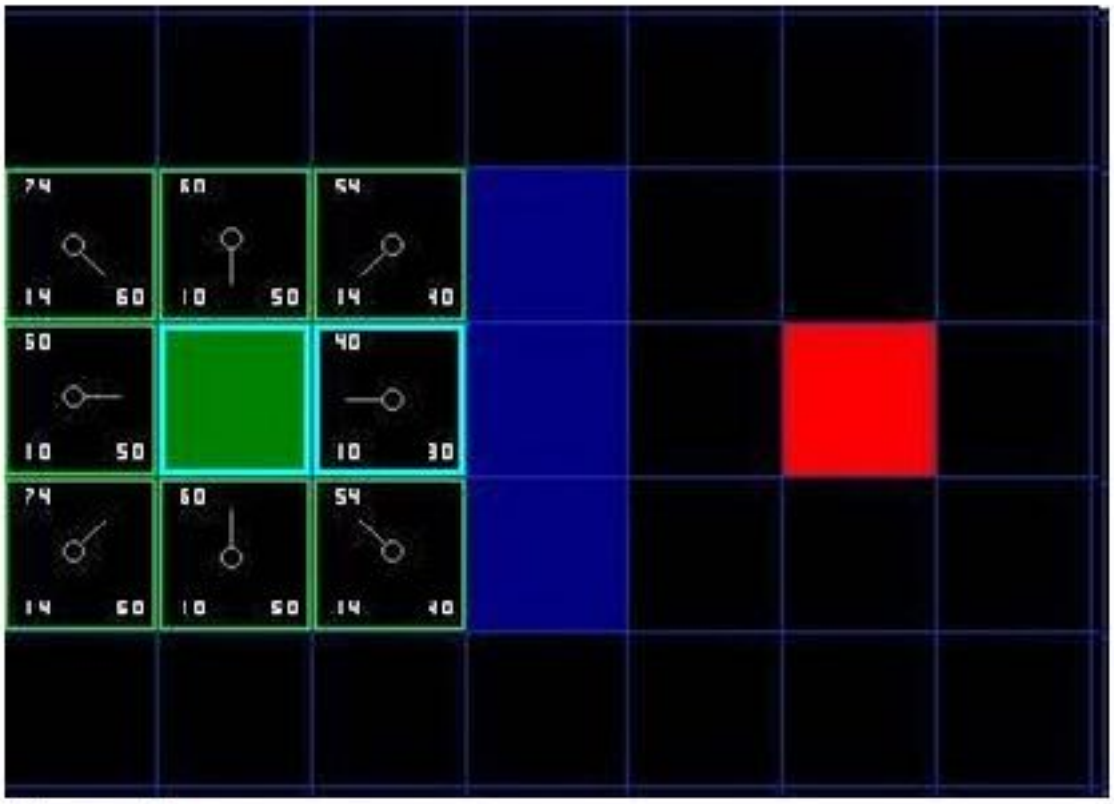
（为了方便描述，把选取的点成为点M）

1、检查M周围的点，在closelist中则忽略它，如果可达且不在openlist中，则加入openlist中，同理的维护一个指向父节点的指正，同时计算加入点的F H G 值。

2、如果M周围的点在openlist中，则看从起点A通过M到这类点的路径是不是小于他们的G值，如果是则更新他们的G、F值（更新为小的）。如果不是则不做任何操作。

3、把M从openlist中移除，加入closelist中。

对openlist中F最小的点（也就是起点左边的点）的处理效果如下图所示：



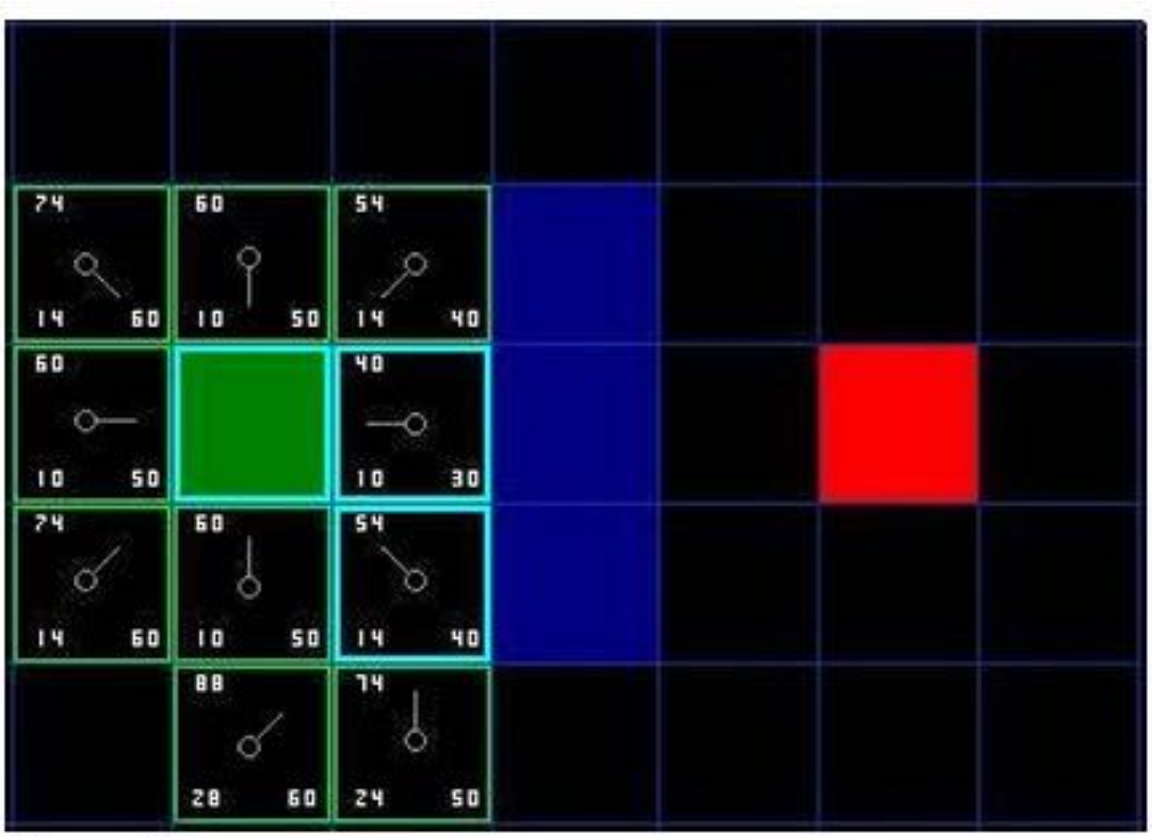
[Figure 4]

M的右边、右上、右下是障碍物，所以忽略他们。M的左边点在closelist中，也不去管他，剩下的是M的上、下、左上、左下的点。他们已经在openlist中，所以看从起点通过M到他们的距离是不是小于他们的G值。通过判断，都比他们的G值大，所以做任何操作。

可以看出，现在的closelist已有两个元素了（高亮绿色包围的块）

下一步的操作和上面叙述的一样，从openlist中找出F最小的，重复上的操作。从图中可以看出，现在的openlist中F最小的有两个，就是刚刚考虑的点的正上方和正下方，其实这里选哪个都无所谓，只是人们习惯于选择较晚加入到openlist中的元素，这里选择下方的点。

同理，处理效果如下图所示：



[Figure 5]

下面简单的说下处理过程：

暂且称现在处理的点为N吧。

N上方　在closelist中，不考虑。

N左方 在openlist中，看从原点通过N到它的距离为14+10大于10，不做操作，跳到下一步

N的左下方，下方 加入openlist中，同时记录F、G 、H的值还有指向父节点的指针。

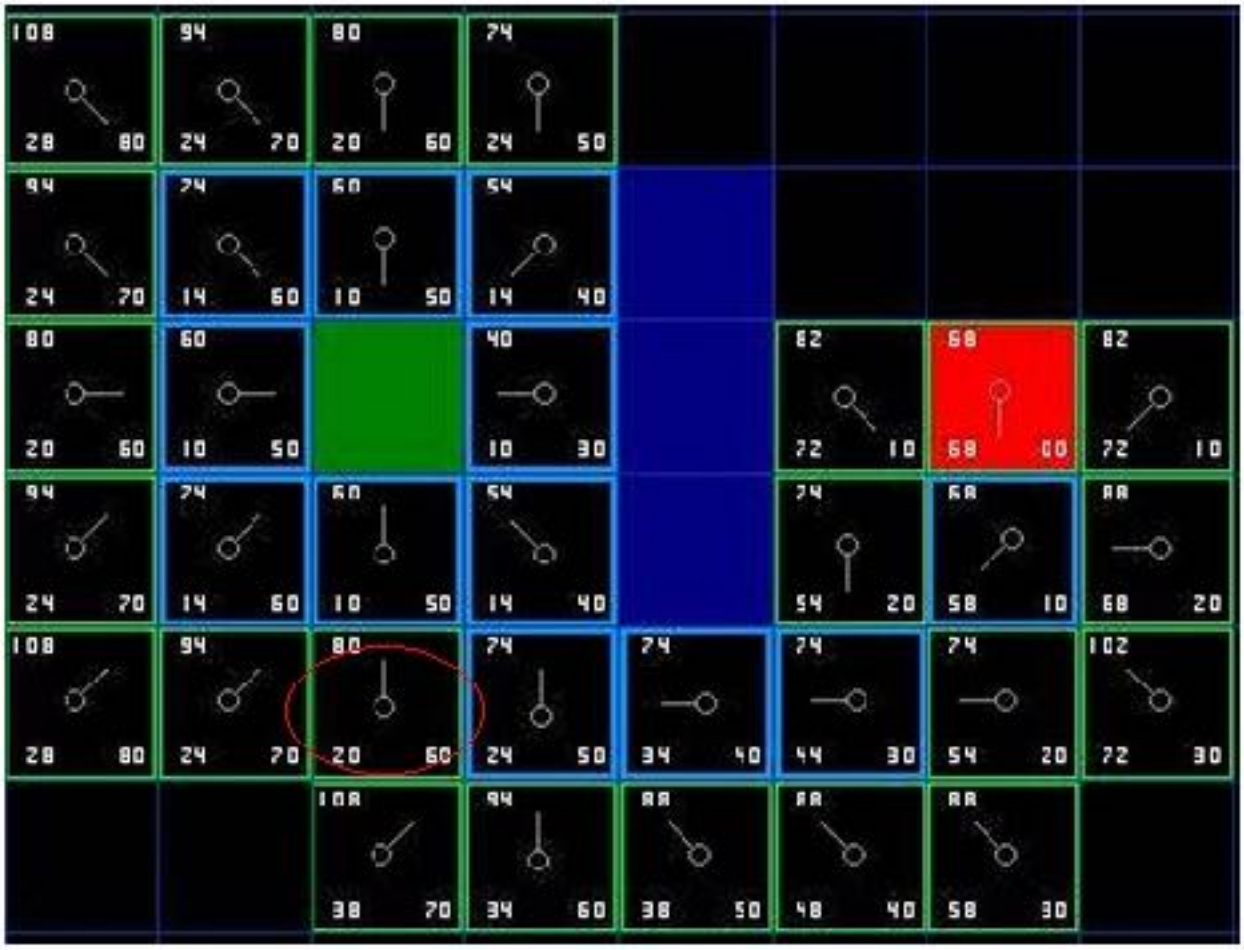
N的右下方这里看做“不可达的点”原因是这两个点都处于障碍物的对角上，当然这只是一种人为的规定。也可以取消这条规定就把它加入到openlist中。这只是一种规定，不必深究。

处理的结果是closelist中现在有三个元素，用高亮的蓝色标记，同样的，openlist中的元素用暗绿色标记出。

重复上的步骤，每次从openlist中选取F最小的点加入closelist中，同时处理这个点周围的元素。。

直到目标节点也被加入到closelist中停止。

处理的效果如下图所示：



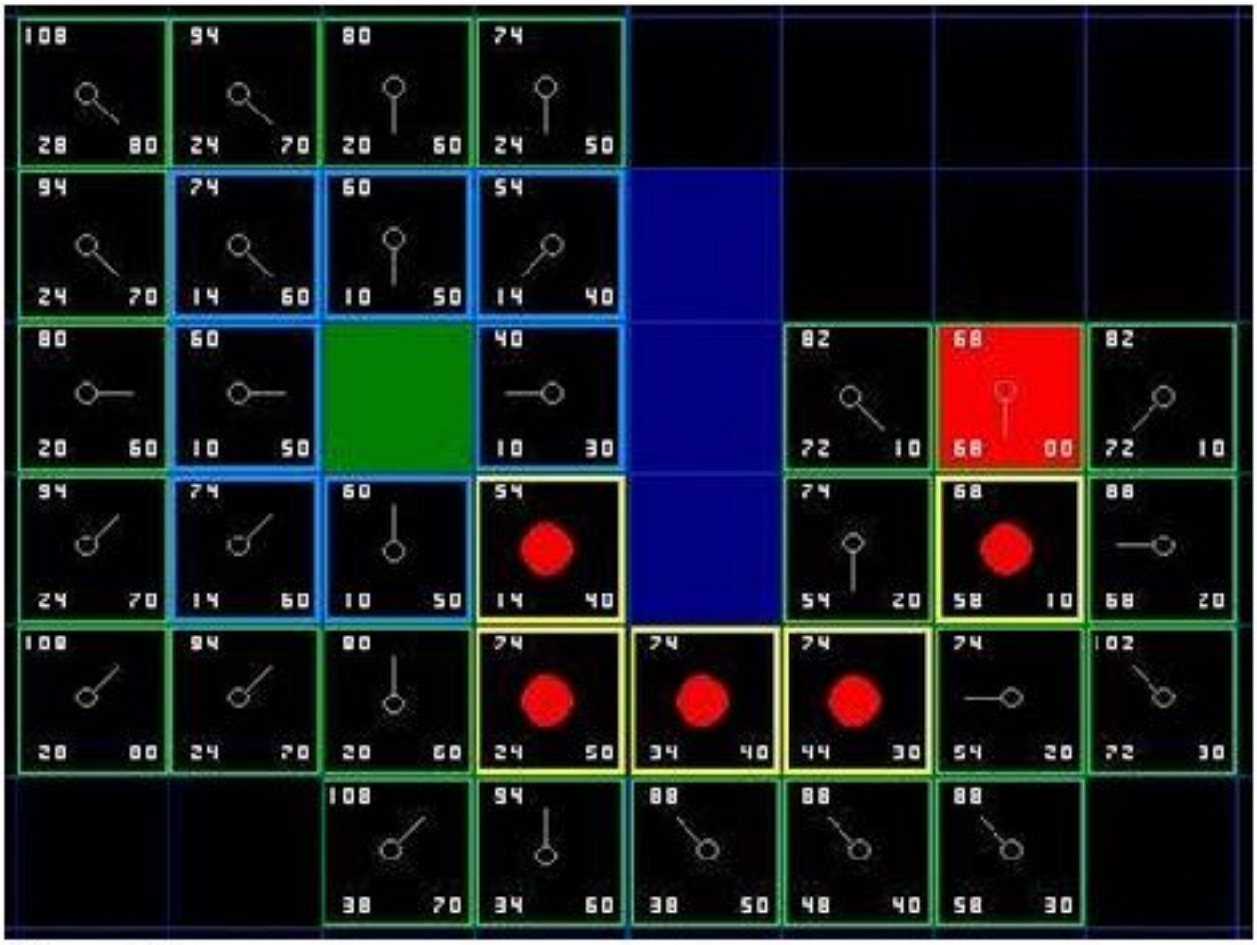
[Figure 6]

如果用心看、你也许已经发现了，在起点正下方两个点的G值，没错，就是图中用椭圆圈起来的点，之前的G=28，现在是20。这是在算法进行的时候更新的，可能 是这其中的某一步，处理这个点的时候，发现了一条更短的路径20，替换了原来的28。

到这里，问题已经基本解决了，最后的任务就是得到这条路径。

只要从目标点出发，沿着他们的父节点遍历，直到起点。就得到了一条最短路径。

如下图所示


[Figure 7]

总结

现在你应该对A*算法有一个初步的认识了吧，总结下算法的实现过程：

1、把起点加入到openlist中

2、重复以下步骤

a、从openlist中找出F最小的节点，并把它当做当前的操作节点

b、检查当前点周围的点，如果已经在openlist中看是否能通过当前点得到更小的G，如果能就更新那个点的G，F的值，如果在closelist中或者是障碍物（不可达）则忽略他们

c、把当前点从openlist中移除 ，加入closelist中

d、当目标点加入closelist中时停止

3、保存路径，从目标点出发，按照父节点指针遍历，直到找到起点。

后记

其实启发式搜索就是对穷举的一种优化，让每次搜索都更接近目标。这就要通过估值函数实现，对于这类问题，找到一个估值函数是关键。

估值函数：从当前点出发到目标点的花费。其实从这个理念上说，好像和分支界限法有些类似，都是在穷举的基础上对搜素优化。