游戏开发中一种新的高效寻路算法
——B* 寻路算法

■ 文 / 赵清松

与传统的A*寻路算法相比，本文提出的B*寻路算法具备更高的效率。

在网络游戏中，寻路已经成为一种自然的AI需求，但在游戏服务器端，除非有独立的AI处理线程或独立的AI服务器模块，否则无法允许可能消耗大量时间的寻路搜索，即使是业界公认最有效的A*寻路算法。

我自主设计了一种B*寻路算法，它非常适用于游戏服务器端的自动寻路，效率是A*算法的几十到上百倍。

算法原理

本算法启发于自然界中真实动物的寻路过程，并加以改善以解决各种阻挡问题。寻路过程中探索节点（Open节点）分为两种状态：自由探索节点和爬绕探索节点，起始探索节点为自由的，从原点朝着目标前进，当遇到障碍时，沿着障碍分叉为左右两个分支，每个分支分别构成一个爬绕的探索节点，试图绕过障碍，绕过障碍后，爬绕节点又变为自由节点向目标前进，此过程迭代进行，直到到达目标寻路成功或探索节点消失表明没有可达路径。

算法描述

地图中节点状态分为以下几种：

```
enum CellState
{
  cell_state_none, //空，寻路
                   可以通过
  cell_state_balk, // 障碍
  cell_state_origin, // 原点
  cell_state_target, // 目标点
  cell_state_close, // Close 已
经走过
  cell_state_open, // Open 当前
探索节点
  cell_state_path, // 完成寻路后构
成路径
  cell_state_check, // Check 当前
检查节点
  cell_state_next, // Next 下一检
查节点
};
```

索节点分以下两种爬绕状态。

自由非爬绕状态：此种状态的探索节点，自由地朝着目标节点前进。

爬绕状态：当自由探索节点遇到障碍后，会沿着障碍分叉为左右两个独立分支，每个分支构成一个爬绕状态的探索节点，从两个方向试图绕过障碍，绕过后重新变为自由非爬绕探索节点。

算法过程

1. 起始，探索节点为自由非爬绕节点，从原点出发，向目标点前进。

2. 判断前面是否为障碍：不是障碍，向目标前进一格，仍为自由非爬绕节点；是障碍，沿障碍分叉为左右两个分支，从两个方向分别试图绕过障碍，这两个分支节点即成为爬绕探索节点。

3. 爬绕的探索节点绕过障碍后，又成为自由非爬绕节点，回到2。

4. 探索节点前进后，判断当前地图格子是否为目标格子，如果是则寻路成功，根据寻路过程构造完整路径。

5. 寻路过程中，如果探索节点没有了，则寻路结束，目标格子不可达。

图1~5演示了算法的寻路过程。

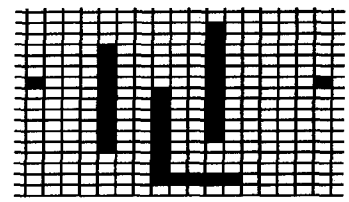


图1 起始（绿为起点，红为终点，灰为阻挡）

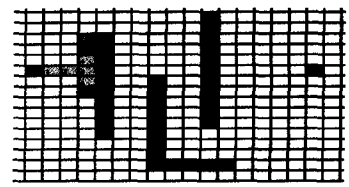


图2 遇到障碍分叉为两个分支

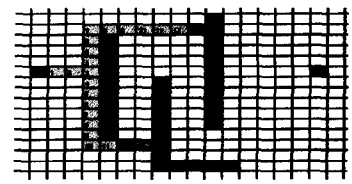


图3 爬绕节点绕过阻挡，又成为自由节点

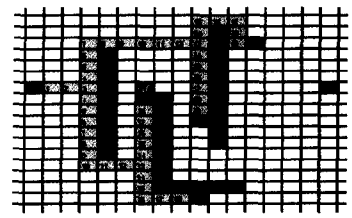


图4 两个自由节点遇到障碍又分叉

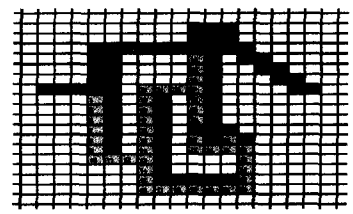


图5 某一探索节点到达目标，寻路成功

表1　四种情况的寻路次数比较

寻路次数比较（5秒钟寻路次数）				
阻挡类型	测试回数	A*基本算法	A* 优化算法	B*算法
无阻挡	1	2500	15617	99970
	1	2391	14823	118780
	平均	2445	15220	109375
	B*是其倍数	44	7	
线型阻挡	1	2954	3955	82901
	1	2908	3987	81700
	平均	2931	3971	82300
	B*是其倍数	28	20	
环型阻挡	1	352	339	45747
	1	302	305	40884
	平均	327	322	43315
	B*是其倍数	132	134	
封闭阻挡	1	159	133	84276
	1	126	130	80890
	平均	142	131	82583
	B*是其倍数	581	630	

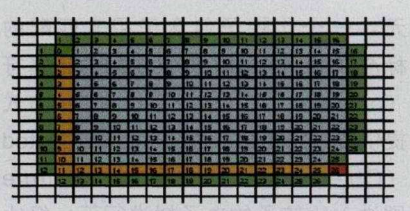


图6　A*基本算法–无阻挡

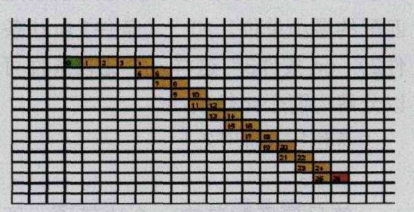


图7　B*算法–无阻挡

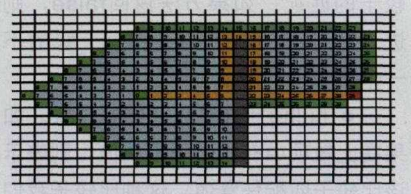


图8　A*基本算法–线型阻挡

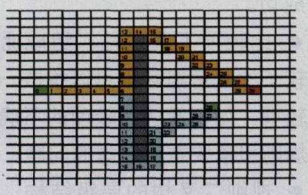


图9　B*算法–线型阻挡

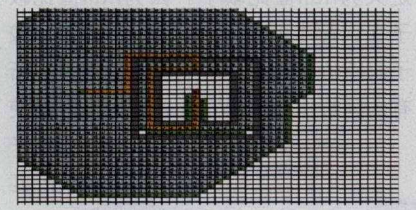


图10　A*基本算法–环型阻挡

图11　B*算法–环型阻挡

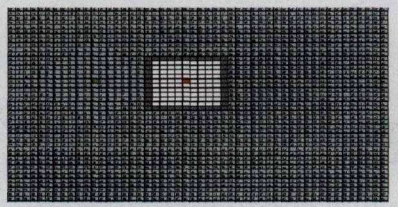


图12　A*基本算法–封闭阻挡

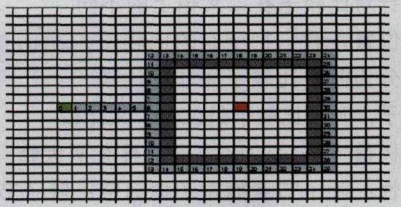


图13　B*算法–封闭阻挡

B*算法与A*算法的性能比较

为了全面比较B*算法与A*算法在性能上的关系，我对A*算法进行了方向上的优化，使A*在简单地图上更加高效，且寻找路径更为平滑，而不是简单地走出所谓直角三角形。

为了完整表现地图阻挡情况，本测试包括四种阻挡情况：无阻挡、线型阻挡、环型阻挡、封闭阻挡。表1为测试结果。由表1可见，无论地图阻挡情况如何，B*算法其效率是A*基本算法（包括优化）的几十倍甚至几百倍。

图6~13是四种阻挡测试的演示。

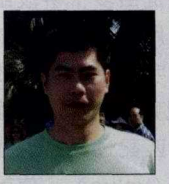
A*算法是“水满则溢”的思想，根据对探索节点的预估函数值进行探索遍历，随着地图阻挡的复杂化，遍历的节点趋向于构成一个平面，所以算法的时间复杂度趋向于O（n^2）。

B*算法则与“水往低处流”相仿，它的时间复杂度无论地图阻挡多么复杂，都基本与探索路径长度相当，所以算法的时间复杂度为O（n）（n为路径长度）。

总结

可见，B*算法具备更加适合游戏的特征：高效率使其更好地满足游戏服务器端对性能的要求；尽管最后寻到的路线可能不是最近的，但路线却更加贴近自然的寻路特征；若找到的不是最短路径，可以进行优化，比如每次寻路记住一个关键点，使下次寻路更近，以达到Boss智能学习的效果，提高游戏策划的策划空间。

作者简介



赵清松，系统分析师，现就职于北京金山游戏七尘斋工作室，任程序副经理。

■ 责任编辑：董世晓（dongsx@csdn.net）