

A* 算法距离评估函数在不同障碍物下的性能分析与改进

兰 坤

(长治医学院, 山西 长治 046000)

摘 要: 本文主要介绍利用带启发信息的 A* 算法进行二维地图中目标的最短路径搜索, 使用主流 cocos2dx 游戏引擎搭建计算机软件的实验平台, 对 A* 算法中的距离评估函数进行性能测试。通过在不同类型障碍物下各种距离评估函数的时间复杂度和空间复杂度进行对比分析, 并在此基础上引出带权距离评估函数, 实现距离评估函数的合理利用和改进, 提高 A* 算法的搜索效率。

关键词: A* 算法; 距离评估函数; 性能分析与改进

中图分类号: TP315

文献标识码: A

文章编码: 1672-6251(2016)04-0022-05

Performance Analysis and Improvement on the Distance Evaluation Functions of A* Algorithm under Different Obstacles

LAN Kun

(Changzhi Medical College, Shanxi Changzhi 046000)

Abstract: This paper used A* algorithm with heuristic information to calculate the shortest path on 2D electric map, and established a computer software testing platform to test the performance of distance evaluation functions of A* algorithm. Based on the contrastive analysis of different distance evaluation functions' time and space complexities under different obstacle models, it created a weighted distance evaluation function to realize its rational utilization and improvement, and improved the searching effectiveness of A* algorithm.

Key words: A* algorithm; distance evaluation function; performance analysis and improvement

A* 算法作为一种主流启发式寻路算法具有效率高与执行速度快等特点, 是现在游戏开发中的主流寻径算法。然而距离评估函数的选择对 A* 算法的搜索效率有着巨大的影响作用。面对不同的游戏地图、地形, 选择合适的距离评估函数, 有助于游戏性能的优化, 提高用户的体验效果^[1]。本文通过主流跨平台游戏引擎 cocos2dx, 实现二维平面游戏开发中对不同路障情况下, 寻路时间效率和空间效率的对比分析, 帮助游戏开发人员在开发中选择合适的距离评估函数来提高游戏的整体性能, 并在此基础上引入带权距离评估函数, 实现进一步优化选择。

1 启发式 A* 算法

1.1 启发函数

A* 算法的启发函数表示为: $f(n) = g(n) + h(n)$ 。

其中: $f(n)$ 值表示节点 n 的估价函数, $g(n)$ 值表示节点 n 离起始节点的实际距离, $h(n)$ 值表示节点 n 到目标节点的距离评估值, $h(n)$ 的大小决定启发信息的大小。

启发函数可以控制 A* 算法的行为, 在有些情况下, 可以通过对 h 函数乘以加权系数的方法, 即: $(f(n) = g(n) + wh(n))$ 来实现启发能力的改变。 w 值大, $h(n)$ 作用大, 启发能力强, 搜索效率高, 但当 w 值很大时, 则会过分强调启发信息。在极端情况下, $g(n)$ 的作用减弱到可忽略不计, 只有 $h(n)$ 起作用时, A* 算法就演变成启发式搜索 (BFS) 算法, 速度最快, 但不一定能得到最短路径; w 值小, 启发能力弱, 搜索效率低, 但是有希望得到最短路径, 过小的 w 值则突出广度优先的特征。

作者简介: 兰坤(1977-), 女, 硕士, 讲师, 研究方向: 计算机基础教学、计算机应用。

收稿日期: 2016-04-08

所以说 A* 算法的距离估计值 $h(n)$ 对游戏中的时间效率和空间效率性能起着决定性作用。A* 算法需要一个距离评估函数来计算 $h(n)$ ，在不同情况下选择合适的距离评估函数，以提高搜索效率也是 A* 算法的精华所在。

1.2 三种常用的距离评估函数

曼哈顿距离 (Manhattan distance)，欧氏几何平面距离 (Euclidean distance) 和切比雪夫距离 (Chebyshev distance) 是距离评估函数中常用的三种。在没有障碍物的地图上三种距离评估函数得到的效果基本是一致的，但是在有障碍物的地图上三种距离评估函数略有差异^[2]。以下三个距离评估函数的定义都是针对二维平面的两个点 (x_1, y_1) 和 (x_2, y_2) 而言的。

1.2.1 曼哈顿距离

曼哈顿距离的命名原因是从规划为方形建筑区的城市 (曼哈顿) 中寻找最短行车路径问题引入的命名^[2]。是指两点间的横向直线距离加纵向直线距离^[3]。

$$D = |x_1 - x_2| + |y_1 - y_2|$$

1.2.2 欧式几何平面距离

欧式几何平面距离又称欧式距离或者欧几里得距离，即两点之间的空间直线距离。

$$D = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

1.2.3 切比雪夫距离

切比雪夫距离又称对角线距离，即两个点的横向直线距离和纵向直线距离的最大值。

$$D = \max(|x_1 - x_2|, |y_1 - y_2|)$$

2 障碍物类型

各种游戏种类繁多，地图模式千姿百态，无法逐一测试具体的地图障碍物类型。但是根据对主流游戏的分析，可把障碍物大致分为三种类型：①点线式障碍物，如 PC 端英雄联盟中一对一模式召唤师 (玩家) 要以最短路径绕过自己的防御塔来到对面防御塔所经过的障碍物，可以简单理解为点、线式结构；②凹形障碍物，如在手游王者荣耀中五对五模式中野区地图大致可分解成多个凹形障碍物组合；③蜂窝状障碍物，如炸弹超人中墙和石头组成的障碍物。

下面将针对这三种类型障碍物进行距离评估函数的性能影响对比。

3 部分逻辑实现

3.1 节点的数据结构

class PathSprite : public cocos2d::Sprite //继承自 Cocos2dx 游戏引擎中精灵节点类

```
{
    PathSprite () :Sprite ()
    { //构造方法
        m_parent = NULL;
        m_child = NULL;
        m_costToSource = 0;
        m_FValue = 0;
    };
public:
    static PathSprite* create (const char* ch)
    {
        PathSprite *pRet = new PathSprite ();
        if (pRet)
        {
            pRet->initWithFile (ch);
            pRet->autorelease ();
            return pRet;
        }
        else
        {
            delete pRet;
            pRet = NULL;
            return NULL;
        }
    }
    PathSprite* m_parent; //父节点 方便保存最终的路径
    PathSprite* m_child; //子节点
    float m_costToSource; //到起始点的距离
    int m_x; //地图坐标
    int m_y;
    float m_FValue;
};

3.2 主要寻路类数据结构
class PathSearchInfo //寻路类 (主要负责寻路的参数和逻辑)
{public:
    static int m_startX; //开始点
    static int m_startY;
    static int m_endX; //结束点
    static int m_endY;
    static vector<PathSprite*> m_openList; //开放列表 (里面存放相邻节点)
    static vector<PathSprite*> m_inspectList; //检测列
```

表 (里面存放除了障碍物的节点)

```
static vector<PathSprite*> m_pathList; //路径列表
.....
static void inspectTheAdjacentNodes ( PathSprite*
node, PathSprite* adjacent, PathSprite* endNode) //把
相邻的节点放入开放节点中
{
    if (adjacent)
    {
        float _x = abs (endNode->m_x - adja-
cent->m_x) ;
        float _y = abs (endNode->m_y - adja-
cent->m_y) ;
        float F, G, H1, H2, H3;
        adjacent ->m_costToSource =node ->
m_costToSource + calculateTwoObjDistance ( node, adja-
cent) ; //获得累计的路程
        G = adjacent->m_costToSource;
        H1 = _x + _y; //曼哈顿距离
        H2 = hypot (_x, _y) ; //欧氏几何平面距离
        H3 = max (_x, _y) ; //切比雪夫距离
        H4 = _x + _y*K; //带权的曼哈顿距离
        H5 = hypot (_x, _y*K) ; //带权的欧氏几何平
面距离
        F = G + H1; //以曼哈顿距离为例
        adjacent->m_FValue = F;
        adjacent->m_parent = node; //设置父节点
        adjacent ->setColor ( Color3B::ORANGE) ; //
搜寻过的节点设为橘色
        node->m_child = adjacent; //设置子节点
        PathSearchInfo::removeObjFromList ( Path-
SearchInfo::m_inspectList, adjacent) ; //把检测过的点从
```

检测列表中删除

```
PathSearchInfo::m_openList.push_back
(adjacent) ; //加入开放列表
}
}
.....
};
```

4 三种常用距离评估函数在不同路障下的性能

4.1 性能分析表

前提: 在分别设置三种类型障碍物的二维平面地图上, 寻径的起点和终点相同。性能分析见表 1。

4.2 总体分析比较

(1) 曼哈顿距离:

平均搜索节点数= (87+171+81) /3=113

平均花费时间= (12+24+9) /3=15

(2) 欧氏距离:

平均搜索节点数= (97+205+91) /3=131

平均花费时间= (14.5+28+9) /3=17.2

(3) 切比雪夫距离:

平均搜索节点数= (131+205+96) /3=144

平均花费时间= (20+30+9) /3=19.7

4.3 三种距离评估函数下搜索范围对比图

其中, h1 为曼哈顿距离, h2 为欧氏几何平面距离, h3 为切比雪夫距离。

(1) 点线状 (图 1)。

(2) 凹形状 (图 2)。

(3) 蜂窝状 (图 3)。

根据以上图表分析, 可以发现在二维平面游戏中三种距离评估函数在搜索空间和搜索时间方面曼哈顿距离总体最优, 在凹形状地图中曼哈顿距离虽然是搜索空间最优, 但是没有找到一条最短路径, 而欧式距

表 1 三种距离评估函数的性能分析

障碍物类型	距离评估函数	平均搜索节点数	平均花费时间 (ms)	是否最短路径
点线状	曼哈顿距离	87	12	是
	欧式距离	97	14.5	是
	切比雪夫距离	131	20	是
凹形状	曼哈顿距离	172	24	否
	欧式距离	205	28	是
	切比雪夫距离	205	30	否
蜂窝状	曼哈顿距离	81	9	是
	欧式距离	91	9	是
	切比雪夫距离	96	9	是

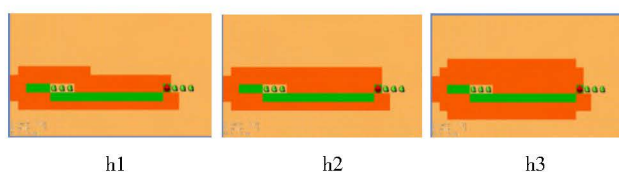


图 1 点线状

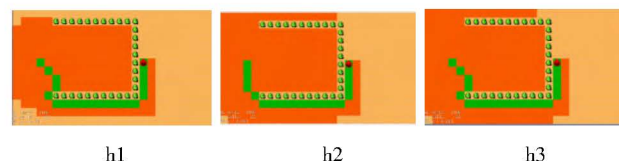


图 2 凹形状

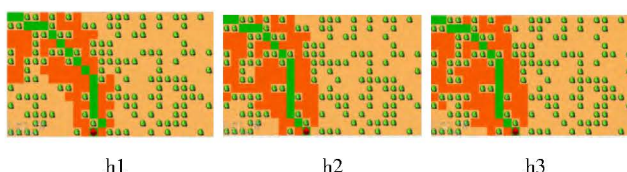


图 3 蜂窝状

离虽然搜索空间大，但是找到了最短路径。那么有没有一种更优的距离评估函数呢？答案是肯定的。

5 带权距离评估函数的分析

不同的地图场景中，有时候纵向与横向距离对启发函数的影响不同^[4]。所以在三种常用的距离评估函数中对 x 、 y 坐标加上权重值，借以提高启发函数的效率。如在曼哈顿距离评估函数中引入权重值 K ，则曼哈顿距离函数变为： $h(n) = |x_1 - x_2| + K * |y_1 - y_2|$ ， K 即为横纵坐标权重值，当 $K=1$ 时， x 与 y 权重相同，即为普通的曼哈顿距离；当 $K>1$ 时，纵轴对距离评估函数影响大；当 $K<1$ 时，横轴对距离评估函数影响大。

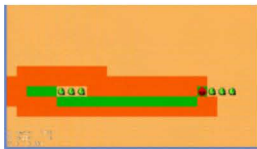
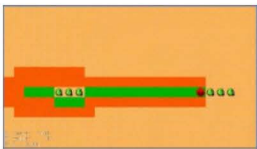
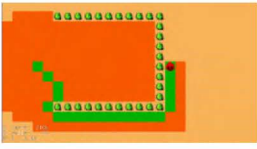
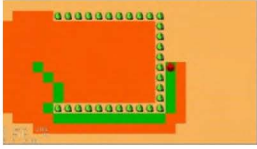
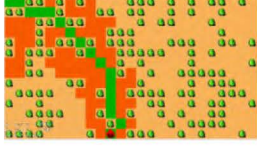
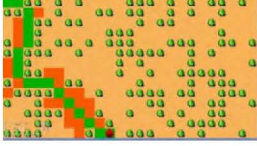
可以根据不同的地图障碍物类型计算出适合该地图的权重值 K ，进一步提高路径搜索效率。下面以曼哈顿距离为例进行定量分析。

由表 1 和表 2 对比可知，带权距离评估函数在 K 取一定范围的值要优于普通距离评估函数，只要根据具体地图测量出合适的 K 值，路径搜索效率就会明显提高。如蜂窝状地图中，普通曼哈顿距离搜索节点数为 81，而当采用带权曼哈顿距离，取 $k=1.2$ 时，搜索节点数为 52，搜索节点数明显小于前者。表 3 是取最优 k 值情况下，带权距离评估函数与不带权距离评估

表 2 带权曼哈顿距离评估函数在不同障碍物下性能分析

	K 值范围	搜索结点数
点线状	0.1~0.9	99
	1	87
	≥ 1.1	72
凹形状	< 0.1	239
	0.1~0.6	239, 234, 227, 209, 196, 193
	0.7~0.9	171
	1	172
	1.1~2	170
	> 2.0	172
蜂窝状	< 0.5	110
	0.9	103
	1	81
	1.2	52
	1.5	46
	2~2.9	44
	≥ 3	51

表 3 带权距离评估函数与不带权距离评估函数的对比图

障碍物类型	普通曼哈顿距离评估函数	带权曼哈顿距离评估函数
点线状		
凹形状		
蜂窝状		

函数的对比图。

由表 3 也可以看出,当 K 值取一定范围的值时,带权距离评估函数都要优于不带权距离评估函数,尤其在蜂窝状地图环境中这种优势尤其明显。

6 结果分析

在普通不带权距离评估函数的选择中,根据图表分析可以发现,在二维平面游戏中三种距离评估函数在搜索空间和搜索时间方面曼哈顿距离最优。在点、线状地图中,如推塔类游戏一对一场景等优先使用曼哈顿距离评估函数;在凹形状地图中曼哈顿距离虽然是搜索空间最优,但不是最短路径,而欧式距离虽然搜索空间大,但是找到了最短路径。游戏开发中在凹形地形比较多的情况下,可以根据游戏需要选择曼哈顿距离或者欧氏距离。实验数据表明在二维平面游戏中切比雪夫距离没有表现出特别的优势,它更适合于三维游戏地图。因为大多数游戏属于静态地图(玩家每次进入同样关卡时,地图障碍物等不变),实际应用中只要根据不同关卡、不同地图,测量出适合的 k 值,那么带权距离评估函数将大大提高搜索效率。

综上,在实现路径搜索应用或游戏时,要根据不

同的地图障碍物类型,选择适合的距离评估函数来帮助我们提高搜索效率,提升用户体验。

7 结束语

A* 算法是一种智能搜索算法,不仅是各种游戏人工智能寻径的首选,而且对 A* 算法进行适当修改,可以广泛应用在交通网络、各类机器人路径搜索、路线导航等领域^[5]。在实际的寻径应用中,选择合理的距离评估函数可以大大提高算法的搜索效率,而设计新的、可行的距离评估函数将是下一步继续研究的重点。

参考文献

- [1] 荆东星.启发式 A* 算法在游戏寻路中的应用[J].电脑与信息技术,2010,(6).
- [2] 王晓华.算法的乐趣[M].北京:人民邮电出版社,2015.
- [3] 梁艺凡.A* 算法在计算机联锁上位机软件中的应用研究[D].兰州:兰州交通大学,2013.
- [4] 宋岩.基于 A-Star 算法的进路搜索研究[D].重庆:西南交通大学,2014.
- [5] 周小镜.基于改进 A* 算法的游戏地图寻径的研究[D].重庆:西南大学,2011.