(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 105999689 A
(43)申请公布日 2016.10.12

(21)申请号 201610369682.2

(22)申请日 2016.05.30

(71)申请人 北京理工大学
地址 100081 北京市海淀区中关村南大街5号

(72)发明人 史继筠 宋鑫 赵峻瑶 孙云霄

(51)Int.Cl.
A63F 3/00(2006.01)
A63F 13/60(2014.01)

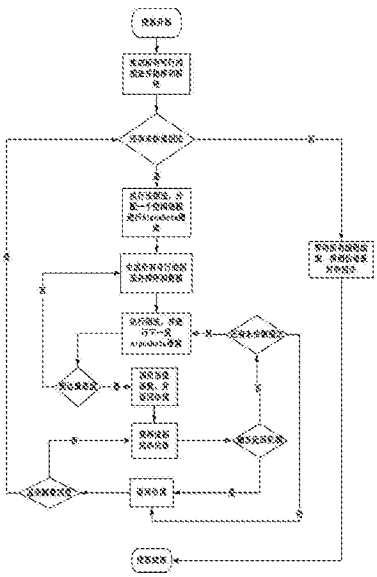
权利要求书7页 说明书9页 附图2页

(54)发明名称

一种机器博弈的亚马逊棋AI算法

(57)摘要

一种机器博弈的亚马逊棋AI算法包括搜索算法和估计算法；其中，搜索算法将生成的可行招法排序，最大可能的将能引发剪枝或能得到较好效果的招法：通过为每一个效果赋值以判定招法的好坏，排在前面；搜索算法还体现在不同阶段采用不同策略：搜索深度越小的层次，生成的招法会被裁剪的越小；而在较深的层次需要展开全招法，即不对招法进行裁剪；估值算法通过加入区域占领参数occupy，使对占领格子的判断更准确；在不同阶段采用不同估值参数，基于步数判断棋局进行到了哪个程度，再决定使用哪种参数，再根据局面对估值参数进行动态调整；此外，还采用优化的多线程：使共享资源尽量少以及使各线程间有互斥地共享有价值的资源。



CN 105999689 A

1.一种机器博弈的亚马逊棋AI算法，其特征在于：

一种机器博弈的亚马逊棋AI算法包括搜索算法和估值算法两方面；

其中，搜索算法方面，将生成的可行招法排序，最大可能的将能引发剪枝或能得到较好效果的招法，具体的，通过为每一个效果赋值以判定招法的好坏，排在前面；搜索算法还体现在不同阶段采用不同策略：在搜索未到达最底层时，生成的招法会被裁剪；而当搜索到达最底层时需要展开全招法，即不对招法进行裁剪；

估值算法方面，一方面，通过加入区域占领参数occupy，使对占领格子的判断更准确；另一方面，在不同阶段采用不同估值参数，基于步数判断棋局进行到了哪个程度，再决定使用哪种参数，再根据局面对估值参数进行动态调整；

此外，搜索算法中还采用优化的多线程：一方面使共享资源尽量少，以防止等待资源消耗的时间；另一方面，使各线程间有互斥地共享有价值的资源，从而提升AlphaBeta算法的效率；具体优化参数主要包括全局变量AlphaB，MultiThread函数中变量i，将变量best引入AlphaBeta函数，以及改进AlphaBeta算法中对返回和更新beta值的机制；

另外，在AlphaBeta的最底层采用更新排序，其它层采用估值排序。

2.如权利要求1所述的一种机器博弈的亚马逊棋AI算法，其特征还在于：

一种机器博弈的亚马逊棋AI算法，主要涉及如下函数：

函数名称	函数功能	输入参数	输出参数	函数流程描述
search_a_goodmove	每下一步棋的入口函数；	无	第一步的所有的可行找法数	该函数作为一个入口，在该函数内调用createMove生成第一步的可行招法，

				然后调用MultiThread
MultiThread	处理多线程，每个线程执行一个MultiThread	*x指向调用时传入的参数：包括AlphaBeta搜索的最大深度depth，第一步的可行招法数stacknum和线程标号thread	无	各线程分别调用AlphaBeta
AlphaBeta	实现AlphaBeta算法的函数	depth表示当前深度，color表示下一步的行棋方，alpha,beta为AlphaBeta算法中的基本参数 thread代表运行于第几个线程；	该函数返回一个浮点型的数据代表对调用函数时执行的招法的估值	在该函数内先调用createMove生成招法，执行一个招法，然后再调用下一层AlphaBeta，直至depth=0时，调用Evaluate，经过不断的嵌套调用后可获得刚执行招法的估值，利用该估值及AlphaBeta机制修改参数alpha,beta

createMove	用于生成所有合规的招法；	depth,color，thread三个参数的,意义与AlphaBeta中输入参数的意义相同，start表示生成招法时的出发点，stacknum表示已生成的招法数；	已生成的招法数	对四个棋子分别按规则生成所有可行招法，返回值为每次生成一个棋子的招法后与之前棋了招法数的总和
createMove_eval	用于生成所有合规的招法，对每一个生成的招法估值；	输入参数同createMove	已生成的招法数	在createMove的基础上对每个生成的招法估值
createMove_select	用于生成所有合规的招法，并对每一个生成的招法进行一次检查，去除不符合要求的招法。	输入参数同createMove	已生成的招法数	在createMove的基础上选择符合筛选条件的招法
Evaluate	在执行一个招法后，对获得的	Color，thread含义与	该函数返回一个浮点型的数	按规则计算数据并加和以获得估值

	局面进行估值(局面的估值即为招法的估值)	AlphaBeta 中参数 color，thread 相同；	据代表对调用函数时执行的招法的估值	

3.如权利要求1所述的一种机器博弈的亚马逊棋AI算法，其特征还在于：

一种机器博弈的亚马逊棋AI算法，具体步骤如下：

步骤一、声明全局变量并初始化下棋步数为0；

步骤二、搜索并选择可用的招法；

步骤三、调用createMove生成可行招法，将满足条件的招法连同估值一起存入招法栈返回：

所述的满足条件是所产生的招法落子点或落障碍处，至少有一点满足五步以内有对方棋子；

步骤四、调用处理多线程的函数MultiThread，具体为：

步骤4.1声明变量i，为全局变量ST设置互斥锁，在互斥锁内将ST的值赋给i，并给ST加1；然后释放ST的互斥锁；

步骤4.2遍历步骤三生成的所有顶层招法；

步骤4.2.1对遍历到的每个招法，在模拟棋盘上执行AlphaBeta搜索，返回该招法的估值；再撤销在模拟棋盘上该招法的执行；

步骤4.2.2若步骤4.2.1返回招法的估值大于当前线程的最好估值，则修改当前线程最好招法，并记录新的最好估值；

步骤4.2.3为全局变量AlphaB设置互斥锁，在互斥锁内判断如果当前线程最好估值大于全局最好估值AlphaB，则修改全局最好招法及AlphaB，并更新全局最好招法BestMove，然后释放AlphaB的互斥锁；

步骤五、调用AlphaBeta函数进行搜索；

步骤5.1判断搜索深度，进行相应操作：

5.11若搜索到达最底，即传入的参数depth＝＝0，则调用估值函数Evaluate，并返回估值；

5.12否则，调用createMove_selecteval或createMove_eval生成招法，同时，对每个生成招法调用Evaluate进行估值，生成全部招法后根据估值，即Evaluate返回值对招法进行排序，具体为：

5.12.1在20步以内有选择性的生成招法，即调用createMove_selecteval函数来生成招法；

5.12.2否则生成全部招法，即调用createMove_eval函数来生成招法；

步骤5.2根据搜索深度对招法进行裁剪，具体为：

5.2.1当搜索深度depth为1时不裁剪招法；

5.2.2当搜索深度depth为2且招法数大于K6，则保留排序在前K6的招法；

5.2.3当搜索深度depth为3且招法数大于K7，则保留排序在前K7的招法；

5.2.4当搜索深度depth为4且招法数大于K8,则保留排序在前K8的招法;

其中,优选的K6为300,也可以是大于250小于350的整数;

其中,优选的K7为200,也可以是大于150小于250的整数;

其中,优选的K8为100,也可以是大于80小于120的整数;

步骤5.3判断当前招法数是否为0,并进行相应操作:

5.31若生成的招法数为0,说明棋局已经结束,返回-INF+(maxdepth-depth);

5.32若生成的招法数不为0,则继续步骤5.4;

步骤5.4对产生的招法估值,并返回该层的最好估值或第一个大于beta的估值;

步骤六、调用估值函数Evaluate;

步骤6.1调用Evaluate该函数计算qstep和kstep;

步骤6.2再计算四个基础变量:t1、t2、c1、c2;这四个值在每次调用估值函数时都初始化为0,具体包含如下操作:

步骤6.2.1通过a和b确定t1的值:

a.当某个格子双方qstep值相同时,对于当前行棋颜色给予t1x分奖励,非当前行棋给予-t1x分的惩罚,再将此奖励或惩罚的分数加在t1上;

其中,所述的t1x分数范围为0.0到1.0;

b.当我方qstep值较小时,使t1获得一个加分score,反之获得一个负分-score;的值为能以qstep所代表的步数到达该格子的棋子的个数,即对于一个可以以最少步数到达的格子,记录能以最少步数到达的棋子的数量;

其中,所述的score分数范围为大于1的整数,原则上不设上限;

a、b同时运用以防止出现以一敌多的局面;

步骤6.2.2计算t2的值,具体为:遍历所有kstep,当我方kstep值较小则t2加1,当对方kstep值较小则t2减1;

步骤6.2.3计算c1及c2的值,具体为:

c1+=(pow(2.0,-qstep[thread][i][j][0])-pow(2.0,-qstep[thread][i][j][1]));

c2+=min(1,max(-1,(kstep[thread][i][j][1].st-kstep[thread][i][j][0].st)/6.0));

注:thread表示当前线程数,i、j对应棋盘上格子,最后一维为0代表我方,1代表对方;

步骤6.3声明计算变量occupy,并赋occupy初值为1,当某格子只有我方可到达则将occupy加一,否则将occupy减1,即occupy记录了我方当前局面已经占领的空格子数量;

步骤6.4声明并计算变量sumA[2];

至此,经过步骤一到步骤六,完成了一种机器博弈的亚马逊棋AI算法。

4.如权利要求3所述的一种机器博弈的亚马逊棋AI算法,其特征还在于:

一种机器博弈的亚马逊棋AI算法,步骤一中声明的全局变量主要如下:

1.1声明AlphaB用于记录多线程中综合的最好估值;

1.2声明maxdepth定义搜索深度;

1.3声明MOVEi记录各线程各自的最好招法的编号;

1.4声明BestMove记录全局的最好招法;

1.5声明qstep和kstep用于估值函数;

1.6声明ST；

1.7声明totalTime记录一盘棋开始到目前的总用时；

1.8声明movestack用于存储招法，movestack又称招法栈；

其中，qstep的含义为四个棋子中沿单一方向连续行进到达的某一格子所需的最少步数；

kstep的含义为四个棋子中方向任意地单步行进到达的某一格子所需的最少步数；ST的含义为记录最顶层搜索到了第几个招法。

5.如权利要求3所述的一种机器博弈的亚马逊棋AI算法，其特征还在于：

步骤二又具体包括如下步骤：

步骤2.1调用入口函数search_a_goodmove；

步骤2.2在函数search_a_goodmove中初始化部分全局变量：其中所述的部分全局变量主要包括全局变量AlphaB；

所述的AlphaB初始值为-INF；

步骤2.3根据下棋步数得出搜索深度值M；

步骤2.4调用createMove_eval生成招法；

步骤2.5对生成的每一个招法调用Evaluate函数，并对所有招法调用Evaluate函数的返回值进行排序；

步骤2.6根据步数和对步骤2.5调用createMove_eval生成的招法进行裁剪；

步骤2.7当总用时totalTime超过14分钟，则将搜索深度值M设为4；

步骤2.8调用MultiThread函数启动多线程；

步骤2.9等待直至步骤2.7的所有线程运行结束，删除多线程并返回。

6.如权利要求5所述的一种机器博弈的亚马逊棋AI算法，其特征还在于：

步骤2.3具体为：

步骤2.3.1当步数step小于K1，则搜索深度值M为4；

步骤2.3.2当步数step大于等于K1小于K2，则搜索深度值M为5；

步骤2.3.3当步数step大于等于K2，则搜索深度值M为6；

其中，优选的K1为32，也可以是大于20小于38的整数；

其中，优选的K2为48，也可以是大于30小于60的整数。

7.如权利要求3所述的一种机器博弈的亚马逊棋AI算法，其特征还在于：

具体的，步骤三的执行过程包括：

步骤3.1对本方四个棋子中的每一个棋子做如下3.11和3.12两个操作，得出一个可行招法；

3.11从棋子出发遍历上下左右四个方向，再遍历每个方向上最多10个可行步，找到符合落子的格子，再遍历所有符合落子的格子，对每一格子执行3.12；

3.12从每一可落子的格子出发，遍历上下左右四个方向，再遍历每个方向上最多10个可行步，找到可以设置障碍的格子；

步骤3.2根据步骤3.1得出的可行招法分别做如下四类操作：

步骤3.2.1调用createMove得到可行招法后直接存入招法栈，继续搜索其他招法；

步骤3.2.2调用createMove_eval执行招法，调用估值函数，得到执行招法后的估值(可

认为即为该招法的估值),将估值和招法都存入招法栈;

步骤3.2.3调用createMove_select在产生招法后先对招法进行检查,是否满足如下条件:

所产生的招法落子点或落障碍处,至少有一点满足五步以内有对方棋子;

再将满足条件的招法加入招法栈;

步骤3.2.4createMove_selecteval结合3.2.2和3.2.3,将满足条件的招法连同估值一起存入招法栈;

步骤3.3遍历结束后,返回。

8.如权利要求3所述的一种机器博弈的亚马逊棋AI算法,其特征还在于:

步骤5.4具体为:

步骤5.4.1声明变量best,并初始化为-1,将当前最好估值赋值给best;

步骤5.4.2遍历执行步骤5.1和5.2产生的所有招法并进入下一层AlphaBeta,递归直至返回估值;

步骤5.4.3得到步骤5.4.1的估值后,若估值大于best,则将得到的估值赋给best;

步骤5.4.4若估值大于beta,则返回估值;

步骤5.4.5遍历结束后,若有更好的招法,则记录并更新;

步骤5.4.6返回best。

9.如权利要求5所述的一种机器博弈的亚马逊棋AI算法,其特征还在于:

步骤2.6具体为:

2.6.1当步数step小于K3且招法数大于K4,则保留排序在前K4的招法;

2.6.2当步数step大于等于K3且招法数大于K5,则保留排序在前K5的招法;

其中,优选的K3为32,也可以是大于6小于36的整数;

其中,优选的K4为300,也可以是大于200小于400的整数;

其中,优选的K5为400,也可以是大于300小于500的整数。

10.如权利要求3所述的一种机器博弈的亚马逊棋AI算法,其特征还在于:

步骤6.4声明并计算变量sumA[2];

其中,所述的sumA[2]代表双方棋子的灵活度,计算方法如下:

步骤6.4.1对于每一个棋子计算其灵活度:寻找该棋子一步即可到达的所有空格子,对于每一个空格子记录棋子与空格子的距离记为i,再计算该空格子周围8个格子中为空的格子数量记为N;计算A=N*pow(2.0,-i);对于所有棋子一步可到达的空格子都能得到这样一个A,棋子的灵活度AM即为所有这样的A的和;

步骤6.4.2sumA为四个棋子各自AM的和;对于双方都计算灵活度,可以得到双方各自的sumA。

一种机器博弈的亚马逊棋AI算法

技术领域

[0001] 本发明主要涉及搜索算法、多线程运用以及估值函数，尤其涉及一种机器博弈的亚马逊棋AI算法，属于人工智能博弈、搜索算法、估值算法领域。

背景技术

[0002] 1.机器博弈

[0003] 计算机博弈(也称机器博弈)，是一个挑战无穷、生机勃勃的研究领域，是人工智能领域的重要研究方向，是机器智能、兵棋推演、智能决策系统等人工智能领域的重要科研基础。机器博弈被认为是人工智能领域最具挑战性的研究方向之一。国际象棋的计算机博弈已经有了很长的历史，并且经历了一场波澜壮阔的“搏杀”，“深蓝”计算机的胜利也给人类留下了难以忘怀的记忆。时至今日，每年仍有很多不同棋类的人机对战以及机器间博弈在包括中国的世界各地定期举行。我们研究的亚马逊棋作为国内计算机博弈锦标赛以及国际计算机博弈竞赛(ICGA)的重要比赛项目已有多年的竞赛积累，两项比赛的冠军亦在不断地更迭中不断进步。

[0004] 亚马逊棋(Game of the Amazons)，是由阿根廷人Walter Zamkauska在1988年推出的两人棋类，是奥林匹亚电脑游戏程式竞赛的比赛指定棋类-。

[0005] 由于局面过于复杂，仅第一步就有两千七百多种走法，故该棋类多不用于人类之间比赛，而是用于计算机博弈相关方面的比赛与研究。

[0006] 10*10方格棋盘。

[0007] 每方各4枚棋子，以黑白两色区分敌我，称为亚马逊(Amazons)；92枚棋子双方共用，称为箭(Arrows)。

[0008] 初始布置后，双方轮流动一己子，方向可朝纵横斜八方，距离无限制至空棋位，中途不得有子。在移动至目标棋位后，从该棋位检视其可移动范围，然后在这范围的任一空格放置一枚箭。箭放定后，就不得移动或移除。

[0009] 直至一方无法行棋则输掉此游戏。

[0010] 2.AlphaBeta剪枝算法

[0011] AlphaBeta剪枝算法(AlphaBeta pruning algorithm)是一个搜索算法旨在减少在其搜索树中，被极大极小(Minimax)算法评估的节点数。这是一个常用人机游戏对抗的搜索算法。它的基本思想是根据上一层已经得到的当前最优结果，决定目前的搜索是否要继续下去。AlphaBeta剪枝算法是对Minimax方法的优化，它们产生的结果是完全相同的，只不过运行效率不一样。AlphaBeta剪枝算法的前提假设与Minimax也是一样的，具体体现在如下三方面：

[0012] 1)双方都按自己认为的最佳着法行棋；

[0013] 2)对给定的盘面用一个分值来评估，这个评估值永远是从一方(搜索程序)来评价的，己方有利时给一个正数，对方有利时给一个负数；当红方走棋时，如果红方有利时返回正数，当轮到黑方走棋时，评估值又转换到黑方的观点，如果认为黑方有利，也返回正数；

[0014] 3)一般来说分值大的数表示对己方有利,而对于对方来说,它会选择分值小的着法。

[0015] AlphaBeta算法严重依赖于着法的寻找顺序,若总是先去搜索最坏的着法,那么Beta截断就不会发生,即AlphaBeta算法最终会找遍整个搜索树,就像最小最大算法一样,效率非常低。

[0016] 若程序总是能挑最好的着法来首先搜索,那么数学上有效分枝因子就接近于实际分枝因子的平方根,这是AlphaBeta算法可能达到的最好的情况。

[0017] 3.多线程技术

[0018] 多线程(multithreading)是指从软件或者硬件上实现多个线程并发执行的技术。具有多线程能力的计算机因有硬件支持而能够在同一时间执行多于一个线程,进而提升整体处理性能。在一个程序中,这些独立运行的程序片段叫“线程”(Thread),利用它编程的概念就叫做“多线程”或“多线程处理”(Multithreading processing)。

[0019] 4.估值函数

[0020] 估值函数用于估计一个局面的效果(正值表示对我方有利,越大越好,负值相反),估值函数是AlphaBeta算法不可或缺的重要部分,估值函数的好坏一方面决定了剪枝效率,另一方面也决定了最终招法的优劣程度。

[0021] 针对亚马逊棋搜索算法的研究,经过资料检索,大约有如下几类:

[0022] a)东北大学张柳在2010年发表的“题目为基于极大极小搜索算法的亚马逊棋博弈系统的研究”的学位论文中,利用经典的AlphaBeta算法、纳什均衡原理以及基于棋子位置的估值函数设计了一个亚马逊棋的博弈程序。但是该程序的搜索效率依然较低。而且估值函数较简单,虽然可节省时间,但是其获得的估值不能准确反映招法的真实价值。

[0023] b)在Julien Kloetzer等人发表于Computer Games Workshop 2007论坛上的题目为“The Monte-Carlo Approach in Amazons”的论文中,他们利用机器博弈领域另一经典算法,蒙特卡洛算法设计实现了一个亚马逊棋博弈程序。同样的,虽然采用了不同的算法,其搜索效率并没有得到显著提升;且相对于AlphaBeta算法,采用uct算法的程序中对估值函数难于掌握,不易编写一个适应搜索深度变化的估值函数,即估值的效果不稳定。

[0024] 其次,针对亚马逊棋估值函数的研究,经过资料检索,除上述论文外,还有以下这种主流估值:

[0025] c)在Jens Lieberum发表于Theoretical Computer Science,Vol.349,No.2的题目为“An evaluation function for the game of amazons”的论文中,采用了基于领地和位置的估值,并引入了灵活度的概念。该估值算法具有一定准确性,但该算法对占领格子与被棋子围堵的情况考虑不到位,导致与对方抢占地盘时容易失利,或出现被对方围堵在较小范围内的不利局面;并且该算法对棋子的影响范围的估计不到位,容易出现某棋子被孤立,形成以一敌多的局面。

[0026] 本申请致力于克服上述效率较低及估值不准确的问题,旨在提出更高效率的搜索算法,以及更准确的用于亚马逊棋博弈的估值函数。

发明内容

[0027] 本发明的目的是克服现有搜索算法搜索效率较低、估值不准确、估值效果不稳定、

以及对占领格子与棋子围堵情况考虑不周的问题，提出了一种机器博弈的亚马逊棋AI算法。

[0028] 一种机器博弈的亚马逊棋AI算法包括搜索算法和估值算法两方面；

[0029] 其中，搜索算法方面，将生成的可行招法排序，最大可能的将能引发剪枝或能得到较好效果的招法，具体的，通过为每一个效果赋值以判定招法的好坏，排在前面；搜索算法还体现在不同阶段采用不同策略：在搜索未到达最底层时，生成的招法会被裁剪；而当搜索到达最底层时需要展开全招法，即不对招法进行裁剪；

[0030] 估值算法方面，一方面，通过加入区域占领参数occupy，使对占领格子的判断更准确；另一方面，在不同阶段采用不同估值参数，基于步数判断棋局进行到了哪个程度，再决定使用哪种参数，再根据局面对估值参数进行动态调整；

[0031] 此外，搜索算法中还采用优化的多线程：一方面使共享资源尽量少，以防止等待资源消耗的时间；另一方面，使各线程间有互斥地共享有价值的资源，从而提升AlphaBeta算法的效率；具体优化参数主要包括全局变量AlphaB，MultiThread函数中变量i，将变量best引入AlphaBeta函数，以及改进AlphaBeta算法中对返回和更新beta值的机制；

[0032] 另外，在AlphaBeta的最底层采用更新排序，其它层采用估值排序；

[0033] 一种机器博弈的亚马逊棋AI算法，主要涉及如下函数：

[0034]

函数名称	函数功能	输入参数	输出参数	函数流程描述
search_a_goodmove	每下一步棋的入口函数；	无	第一步的所有的可行找法数	该函数作为一个入口，在该函数内调用createMove生成第一步的可行招法，然后调用MultiThread
MultiThread	处理多线程，每个线程执行一个MultiThread	*x指向调用时传入的参数：包括AlphaBeta搜索的最大深度depth，第一步的可行招法数stacknum和线程标号thread	无	各线程分别调用AlphaBeta
AlphaBeta	实现AlphaBeta算法的函数	depth表示当前深度，color表示下一步的行棋方，alpha，beta为	该函数返回一个浮点型的数据代表对调用函数时执行的招法的估值	在该函数内先调用createMove生成招法，执行一个招法，然后再调用下一层AlphaBeta，直至

[0035]

		AlphaBeta 算法中的基本参数 thread 代表运行于第几个线程；		depth=0 时，调用 Evaluate，经过不断的嵌套调用后可获得刚执行招法的估值，利用该估值及 AlphaBeta 机制修改参数 alpha, beta
createMove	用于生成所有合规的招法；	depth,color ，thread 三个参数的，意义与 AlphaBeta 中输入参数的意义相同，start 表示生成招法时的出发点，stacknum 表示已生成的招法数；	已生成的招法数	对四个棋子分别按规则生成所有可行招法，返回值为每次生成一个棋子的招法后与之前棋子招法数的总和
createMove_eval	用于生成所有合规的招法，对每一个生成的招法估值；	输入参数同 createMove	已生成的招法数	在 createMove 的基础上对每个生成的招法估值
createMove_select	用于生成所有合规的招法，并对每一个生成的招法进行一次检查，去除不符合要求的招法。	输入参数同 createMove	已生成的招法数	在 createMove 的基础上选择符合筛选条件的招法
Evaluate	在执行一个招法后，对获得的局面进行估值（局面的估值即为招法的估值）	Color，thread 含义与 AlphaBeta 中参数 color，thread 相同；	该函数返回一个浮点型的数据代表对调用函数时执行的招法的估值	按规则计算数据并加和以获得估值

[0036] 一种机器博弈的亚马逊棋AI算法，具体步骤如下：

[0037] 步骤一、声明全局变量并初始化下棋步数为0；

[0038] 具体的，步骤一中声明的全局变量主要如下：

[0039] 1.1声明AlphaB用于记录多线程中综合的最好估值；

[0040] 1.2声明maxdepth定义搜索深度；

[0041] 1.3声明MOVEi记录各线程各自的最好招法的编号；

[0042] 1.4声明BestMove记录全局的最好招法；

[0043] 1.5声明qstep和kstep用于估值函数；

[0044] 1.6声明ST；

[0045] 1.7声明totalTime记录一盘棋开始到目前的总用时；

[0046] 1.8声明movestack用于存储招法，movestack又称招法栈；

[0047] 其中，qstep的含义为四个棋子中沿单一方向连续行进到达的某一格子所需的最少步数；

[0048] kstep的含义为四个棋子中方向任意地单步行进到达的某一格子所需的最少步数；ST的含义为记录最顶层搜索到了第几个招法；

[0049] 步骤二、搜索并选择可用的招法，又具体包括如下步骤：

[0050] 步骤2.1调用入口函数search_a_goodmove；

[0051] 步骤2.2在函数search_a_goodmove中初始化部分全局变量：其中所述的部分全局变量主要包括全局变量AlphaB；

[0052] 所述的AlphaB初始值为-INF；

[0053] 步骤2.3根据下棋步数得出搜索深度值M；

[0054] 具体步骤为：

[0055] 步骤2.3.1当步数step小于K1，则搜索深度值M为4；

[0056] 步骤2.3.2当步数step大于等于K1小于K2，则搜索深度值M为5；

[0057] 步骤2.3.3当步数step大于等于K2，则搜索深度值M为6；

[0058] 其中，优选的K1为32，也可以是大于20小于38的整数；

[0059] 其中，优选的K2为48，也可以是大于30小于60的整数；

[0060] 步骤2.4调用createMove_eval生成招法；

[0061] 步骤2.5对生成的每一个招法调用Evaluate函数，并对所有招法调用Evaluate函数的返回值进行排序；

[0062] 步骤2.6根据步数和对步骤2.5调用createMove_eval生成的招法进行裁剪，具体为：

[0063] 2.6.1当步数step小于K3且招法数大于K4，则保留排序在前K4的招法；

[0064] 2.6.2当步数step大于等于K3且招法数大于K5，则保留排序在前K5的招法；

[0065] 其中，优选的K3为32，也可以是大于6小于36的整数；

[0066] 其中，优选的K4为300，也可以是大于200小于400的整数；

[0067] 其中，优选的K5为400，也可以是大于300小于500的整数；

[0068] 步骤2.7当总用时totalTime超过14分钟，则将搜索深度值M设为4；

[0069] 步骤2.8调用MultiThread函数启动多线程；

[0070] 步骤2.9等待直至步骤2.7的所有线程运行结束，删除多线程并返回；

[0071] 步骤三、调用createMove生成可行招法，将满足条件的招法连同估值一起存入招法栈返回：

[0072] 所述的满足条件是所产生的招法落子点或落障碍处，至少有一点满足五步以内有对方棋子；

[0073] 具体的，步骤三的执行过程包括：

[0074] 步骤3.1对本方四个棋子中的每一个棋子做如下3.11和3.12两个操作，得出一个可行招法；

[0075] 3.11从棋子出发遍历上下左右四个方向，再遍历每个方向上最多10个可行步，找

到符合落子的格子，再遍历所有符合落子的格子，对每一格子执行3.12；

[0076] 3.12从每一可落子的格子出发，遍历上下左右四个方向，再遍历每个方向上最多10个可行步，找到可以设置障碍的格子；

[0077] 步骤3.2根据步骤3.1得出的可行招法分别做如下四类操作：

[0078] 步骤3.2.1调用createMove得到可行招法后直接存入招法栈，继续搜索其他招法；

[0079] 步骤3.2.2调用createMove_eval执行招法，调用估值函数，得到执行招法后的估值(可认为即为该招法的估值)，将估值和招法都存入招法栈；

[0080] 步骤3.2.3调用createMove_select在产生招法后先对招法进行检查，是否满足如下条件：

[0081] 所产生的招法落子点或落障碍处，至少有一点满足五步以内有对方棋子；

[0082] 再将满足条件的招法加入招法栈；

[0083] 步骤3.2.4createMove_selecteval结合3.2.2和3.2.3，将满足条件的招法连同估值一起存入招法栈；

[0084] 步骤3.3遍历结束后，返回；

[0085] 步骤四、调用处理多线程的函数MultiThread，具体为：

[0086] 步骤4.1声明变量i，为全局变量ST设置互斥锁，在互斥锁内将ST的值赋给i，并给ST加1；然后释放ST的互斥锁；

[0087] 步骤4.2遍历步骤三生成的所有顶层招法；

[0088] 步骤4.2.1对遍历到的每个招法，在模拟棋盘上执行AlphaBeta搜索，返回该招法的估值；再撤销在模拟棋盘上该招法的执行；

[0089] 步骤4.2.2若步骤4.2.1返回招法的估值大于当前线程的最好估值，则修改当前线程最好招法，并记录新的最好估值；

[0090] 步骤4.2.3为全局变量AlphaB设置互斥锁，在互斥锁内判断如果当前线程最好估值大于全局最好估值AlphaB，则修改全局最好招法及AlphaB，并更新全局最好招法BestMove，然后释放AlphaB的互斥锁；

[0091] 步骤五、调用AlphaBeta函数进行搜索；

[0092] 步骤5.1判断搜索深度，进行相应操作：

[0093] 5.11若搜索到达最底，即传入的参数depth＝＝0，则调用估值函数Evaluate，并返回估值；

[0094] 5.12否则，调用createMove_selecteval或createMove_eval生成招法，同时，对每个生成招法调用Evaluate进行估值，生成全部招法后根据估值，即Evaluate返回值对招法进行排序，具体为：

[0095] 5.12.1在20步以内有选择性的生成招法，即调用createMove_selecteval函数来生成招法；

[0096] 5.12.2否则生成全部招法，即调用createMove_eval函数来生成招法；

[0097] 步骤5.2根据搜索深度对招法进行裁剪，具体为：

[0098] 5.2.1当搜索深度depth为1时不裁剪招法；

[0099] 5.2.2当搜索深度depth为2且招法数大于K6，则保留排序在前K6的招法；

[0100] 5.2.3当搜索深度depth为3且招法数大于K7，则保留排序在前K7的招法；

[0101] 5.2.4当搜索深度depth为4且招法数大于K8，则保留排序在前K8的招法；

[0102] 其中，优选的K6为300，也可以是大于250小于350的整数；

[0103] 其中，优选的K7为200，也可以是大于150小于250的整数；

[0104] 其中，优选的K8为100，也可以是大于80小于120的整数；

[0105] 步骤5.3判断当前招法数是否为0，并进行相应操作：

[0106] 5.31若生成的招法数为0，说明棋局已经结束，返回-INF+(maxdepth-depth)；

[0107] 5.32若生成的招法数不为0，则继续步骤5.4；

[0108] 步骤5.4对产生的招法估值，并返回该层的最好估值或第一个大于beta的估值，具体为：

[0109] 步骤5.4.1声明变量best，并初始化为-1，将当前最好估值赋值给best；

[0110] 步骤5.4.2遍历执行步骤5.1和5.2产生的所有招法并进入下一层AlphaBeta，递归直至返回估值；

[0111] 步骤5.4.3得到步骤5.4.1的估值后，若估值大于best，则将得到的估值赋给best；

[0112] 步骤5.4.4若估值大于beta，则返回估值；

[0113] 步骤5.4.5遍历结束后，若有更好的招法，则记录并更新；

[0114] 步骤5.4.6返回best；

[0115] 步骤六、调用估值函数Evaluate；

[0116] 步骤6.1调用Evaluate该函数计算qstep和kstep；

[0117] 步骤6.2再计算四个基础变量：t1、t2、c1、c2；这四个值在每次调用估值函数时都初始化为0，具体包含如下操作：

[0118] 步骤6.2.1通过a和b确定t1的值：

[0119] a.当某个格子双方qstep值相同时，对于当前行棋颜色给予t1x分奖励，非当前行棋给予-t1x分的惩罚，再将此奖励或惩罚的分数加在t1上；

[0120] 其中，所述的t1x分数范围为0.0到1.0；

[0121] b.当我方qstep值较小时，使t1获得一个加分score，反之获得一个负分-score；的值为能以qstep所代表的步数到达该格子的棋子的个数，即对于一个可以以最少步数到达的格子，记录能以最少步数到达的棋子的数量；

[0122] 其中，所述的score分数范围为大于1的整数，原则上不设上限；

[0123] a、b同时运用以防止出现以一敌多的局面；

[0124] 步骤6.2.2计算t2的值，具体为：遍历所有kstep，当我方kstep值较小则t2加1，当对方kstep值较小则t2减1；

[0125] 步骤6.2.3计算c1及c2的值，具体为：

[0126] c1+=(pow(2.0,-qstep[thread][i][j][0])-pow(2.0,-qstep[thread][i][j][1]))；

[0127] c2+=min(1,max(-1,(kstep[thread][i][j][1].st-kstep[thread][i][j][0].st)/6.0))；

[0128] 注：thread表示当前线程数，i、j对应棋盘上格子，最后一维为0代表我方，1代表对方；

[0129] 步骤6.3声明计算变量occupy，并赋occupy初值为1，当某格子只有我方可到达则

将occupy加一，否则将occupy减1，即occupy记录了我方当前局面已经占领的空格子数量；

[0130] 步骤6.4声明并计算变量sumA[2]；

[0131] 其中，所述的sumA[2]代表双方棋子的灵活度，计算方法如下：

[0132] 步骤6.4.1对于每一个棋子计算其灵活度：寻找该棋子一步即可到达的所有空格子，对于每一个空格子记录棋子与空格子的距离记为i，再计算该空格子周围8个格子中为空的格子数量记为N；计算A＝N*pow(2.0,-i)；对于所有棋子一步可到达的空格子都能得到这样一个A，棋子的灵活度AM即为所有这样的A的和；

[0133] 步骤6.4.2sumA为四个棋子各自AM的和；对于双方都计算灵活度，可以得到双方各自的sumA；

[0134] 至此，经过步骤一到步骤六，完成了一种机器博弈的亚马逊棋AI算法。

[0135] 有益效果

[0136] 本发明一种机器博弈的亚马逊棋AI算法与现有亚马逊棋AI算法相比，具有如下有益效果：

[0137] 1.本发明所述算法在多线程策略中，全局变量AlphaB不止在AlphaBeta的最顶层被修改，而是在内层触发某种条件时就能被修改，此策略的应用显著地提高了搜素速度，即提升了算法效率；

[0138] 2.本发明所述算法在多线程策略中，声明变量i并为其设置互斥锁，保证同一时刻里只有一个线程能访问该变量；i使得多个线程可以像一个线程运行时一样，从0到n逐个搜索招法，在有招法排序的前提下，此策略可以提高剪枝效率；

[0139] 3.本发明所提算法，在全盘运行时间为15分钟以内的情况下，可实现将AlphaBeta搜索的最低层数由3层提升至4层，即表示本算法可以在全盘运行时间不允许超过15分钟的情况下，比同类算法的对手多考虑一步；

[0140] 4.本发明所提算法在有益效果1的基础上，后期的最深思考层数可达6层，经测试层数继续加深依然可以；

[0141] 5.本发明所提估值算法中，代表灵活度的值sumA的计算方法的改进使得估值函数对占领格子，与被围堵的判断更精准，使得在这两个情况出现时，最终给出的招法更合理有效。

附图说明

[0142] 图1为本发明一种机器博弈的亚马逊棋AI算法的流程图；

[0143] 图2为本发明一种机器博弈的亚马逊棋AI算法及其实施例中使用的亚马逊棋盘。

[0144] 实施方式

[0145] 为使本发明实施例的目的、技术方案和优点更加清楚，下面将结合本发明实施例中的附图，对本发明实施例中的技术方案进行清楚、完整的描述，显然，所描述的实施例是本发明的一部分实施例，而不是全部实施例。

[0146] 需要说明的是程序中有全局变量step，记录双方从开局到当前局面共执行的步数，初值为零。

[0147] 实施例1

[0148] 当step大于等于6小于10时，设置默认搜索深度M为4；带估值的生成招法，按估值

排序招法栈，剪枝留下排序在前K4的招法；交付多线程运行。

[0149] 各线程分别调用AlphaBeta函数，采用选择性生成方式生成下一层招法，按估值排序，然后剪枝，根据深度不同使招法分别不超过K6，K7，K8个。

[0150] 遍历生成的招法，执行之，再次调用AlphaBeta进入下层。

[0151] 搜索深度在递归过程中递减，直至为0，则调用估值函数，并返回；不再调用AlphaBeta。该估值采用对灵活度更敏感的参数。

[0152] 根据规则更新alpha，beta，best值，修改AlphaB。遍历所有招法后再向上层返回。

[0153] 直至返回到顶层，在MultiThread函数中根据规则更新该线程最好招法。

[0154] 所有线程结束后，综合各线程最好招法选出全局最好招法。

[0155] 实施例2

[0156] 当step大于等于20小于32，在实施例3的基础上采用更全面的招法生成，即在生成招法时没有选择性。

[0157] 实施例3

[0158] 当step大于等于32小于48，设置默认搜索深度M为5；带估值的生成招法，按估值排序招法栈，剪枝留下排序在前K5的招法；后续步骤与实施例1相同。

[0159] 综上所述，以上仅为本发明的较佳实例而已，并非用于限定本发明的保护范围。凡在本发明的精神和原则之内，所作的任何修改、等同替换、改进等，均应包含在本发明的保护范围之内。

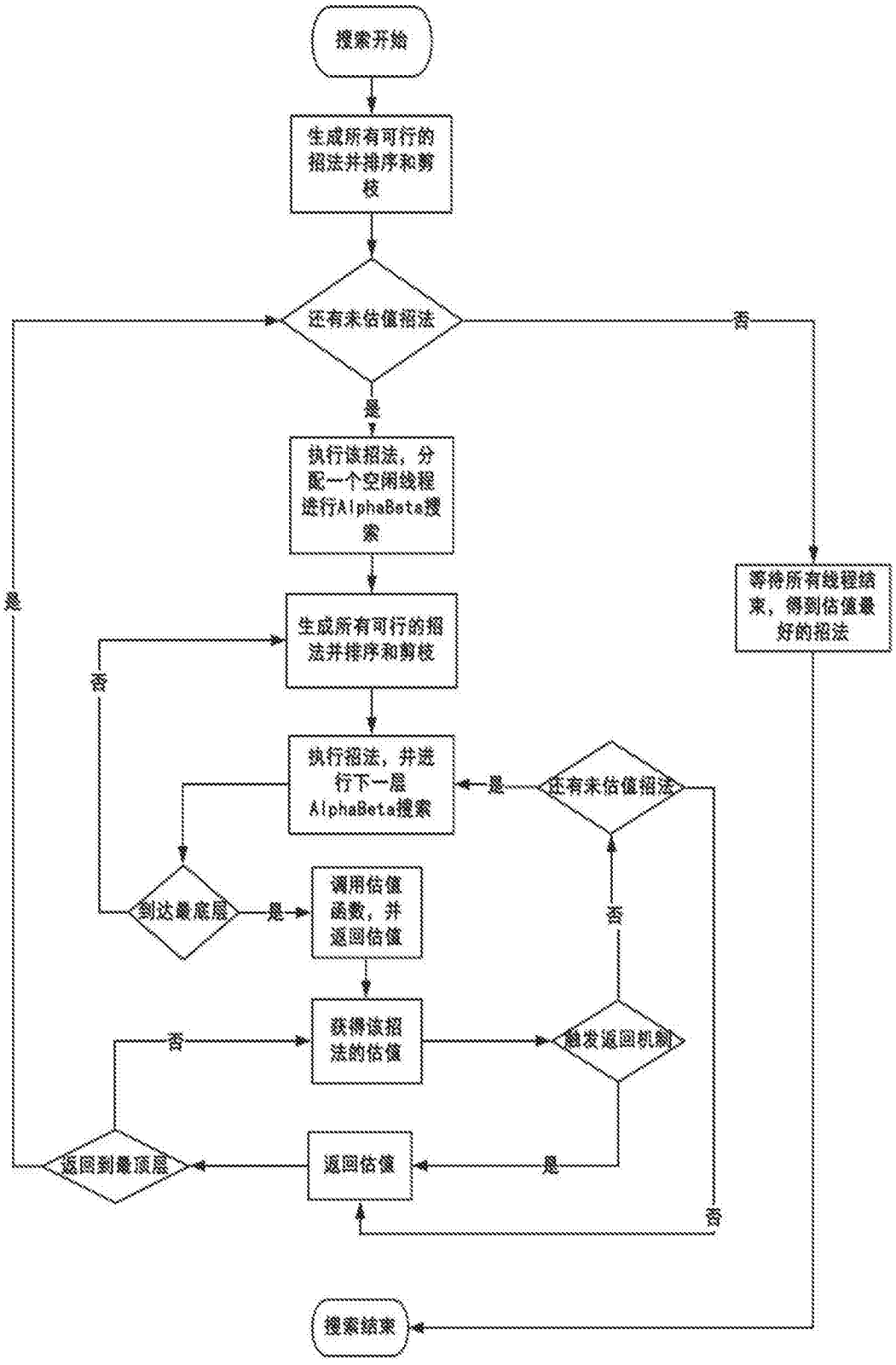

搜索开始
生成所有可行的招法并排序和剪枝
还有未估值招法
否
是
执行该招法，分配一个空闲线程进行AlphaBeta搜索
等待所有线程结束，得到估值最好的招法
是
生成所有可行的招法并排序和剪枝
否
执行招法，并进行下一层AlphaBeta搜索
是
还有未估值招法
到达最底层
是
调用估值函数，并返回估值
否
获得该招法的估值
触发返回机制
否
返回到最顶层
返回估值
是
否
搜索结束

图1

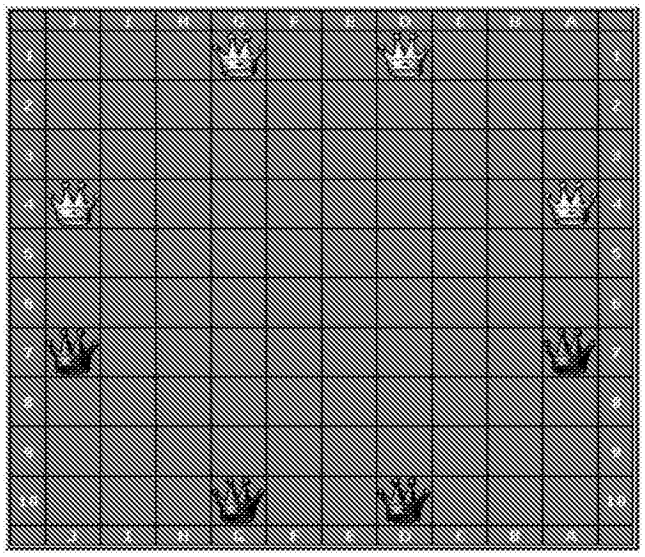


图2