四子棋实验报告

计54 陈宇 2015011343

实验描述

针对重力四子棋游戏，设计一个AI算法，打败尽量多的已有AI。

蒙特卡洛搜索树(MCST)

为什么选MCST

考虑到目标是设计一个AI算法，所以最优的可选方案是MCST和ab剪枝；但是考虑到本四子棋的特殊性（棋盘上并不是所有空白点都可落子，并且还有障碍点），所以人工设置一个估价函数是比较困难的；相反，如果选用MCST，则**可以免去设置估价函数的尴尬**，故最会选中的方案是MCST。

MCST是怎样工作的

在MCST算法中，存在一颗由不同局面组成的一颗“树”，这颗“树”的根节点就是当前局面，其子节点就是当前局面在某一处落子之后的局面，也是对手的局面，其他节点依次类推；对于每个节点，记录了该节点及其子节点一共被模拟了多少次，其中获胜多少次。

MCST的算法流程：

1. 当前节点为根节点
2. 如果当前节点可以扩展，则随机选择一个可扩展的局面，对该局面进行模拟，跳到5
3. 如果当前节点有子节点，则根据“信心上界”选择最优的一个节点作为当前节点，跳到2
4. 选择当前节点进行模拟
5. 根据模拟的结果，更新当前节点到根路径上所有节点的信息
6. 完成

当上面算法运行了足够多的次数，即可以选择根节点的所有子节点中期望胜率最大的一个节点作为最优操作。

对于MCST算法的一些修改

优化模拟

在传统的MCST算法中，模拟这部分是采用的纯随机的方式，这种方式的好处是简单，方便代码实现，速度快；但是缺点就是纯随机不太能够代表人的操作；为此，在尽量减少对速度的影响下，我添加了**必胜特判**和**必败特判**，即：如果当前某一步操作能够导致自己直接获胜，则一定下这个位置，否则，如果有某个位置不下会导致对方下一步获胜，则我方先一步下这个位置，如果以上两种情况都没有出现，则随机落子。

“信心上界”参数选取

信心上界公式：

$$UCB = \frac{win_i}{total_i} + C * \sqrt{\frac{2 * ln(total)}{total_i}}$$

由于第二项有参数C，所以考虑可以将根号中的数字2移除，使得公式变成了这样：

$$UCB = \frac{win_i}{total_i} + C * \sqrt{\frac{ln(total)}{total_i}}$$

在实际测试中，选用$C = 0.7$是最优方案。

常数优化

1. 胜利判定

   只判定最后一个落子有没有构成4连

2. 内存分配

   考虑到每次运行所用内存的最大值是可以预估的，所以采用内存池的方式进行优化；并且，将申请的内存放在全局变量中，这样只需要在整个程序结束的时候进行释放即可，节省了每次决策申请和释放内存的时间。

3. “信心上界”计算

   对于浮点数，每次计算其除法、对数和根号均非常费时；仔细分析“信心上界”的公式，发现我们可以将其分成两部分，一部分只和节点自身的信息有关，另一部分只和全局有关，拆分如下：

$$A_i = \frac{win_i}{total_i}, B_i = \sqrt{\frac{1.0}{total_i}}$$

$$X = C * \sqrt{ln(total)}$$

$$UCB = A_i + B_i * X$$

   其中 A_i, B_i 只和自身节点信息有关， X 只和全局有关，故每个节点使用记录 A_i, B_i 即可。

4. 局面储存

   前面已经说过，MCST中每个节点代表一个局面，但是如果每个节点都储存当前局面棋盘信息的话，会占用很大空间，而且相邻局面之间绝大部分是相同的；考虑到这颗MCST树其实从根到每个节点的路径都表示一种两人的对抗过程，所以每个节点的局面只用记录从父节点到当前节点多了哪个落子，然后就可以通过当前节点到根节点的路径以及根节点的局面合成出当前的局面。

5. 时间

   在每次决策过程中，不断执行蒙特卡洛树算法，直到运行时间达到了2.5秒，或者总次数超过了5000000次，则结束。

代码使用说明

AI的全部核心代码均在 `AIEngine.h` 和 `AIEngine.cpp` 中，这两个文件实现了 `AIEngine` 类：

`AIEngine(int MAX_M, int MAX_N)` 构造函数，两个参数分别表示可能的最大行和最大列。

`void reload(const int M, const int N, const int* _board, const int noX, const int noY)` 传入当前局面。

`Point getAction()` 获取决策，每次调用该函数必须首先调用 `reload` 函数传入局面。

与现有AI的测试结果

对战AI	平局数量	胜利数量	失败数量
2.dll	0	8	0
4.dll	0	8	0
6.dll	0	8	0
8.dll	0	8	0
10.dll	0	8	0
12.dll	0	8	0

对战AI	平局数量	胜利数量	失败数量
14.dll	0	8	0
16.dll	0	8	0
18.dll	0	8	0
20.dll	0	8	0
22.dll	0	8	0
24.dll	0	8	0
26.dll	0	8	0
28.dll	0	8	0
32.dll	0	8	0
34.dll	0	7	1
36.dll	0	8	0
38.dll	0	8	0
40.dll	0	8	0
42.dll	0	8	0
44.dll	0	8	0
46.dll	0	8	0
48.dll	0	8	0
50.dll	0	8	0
52.dll	0	8	0
54.dll	1	6	1
56.dll	0	8	0
58.dll	0	8	0
60.dll	0	8	0
62.dll	0	8	0
64.dll	0	8	0
68.dll	0	7	1
70.dll	0	8	0
72.dll	0	8	0
74.dll	0	6	2
76.dll	0	8	0
78.dll	0	8	0
80.dll	0	8	0
82.dll	0	6	2
84.dll	0	8	0
86.dll	0	8	0
88.dll	0	7	1
90.dll	0	7	1
92.dll	0	5	3
94.dll	0	8	0
96.dll	0	5	3
98.dll	0	7	1

对战AI	平局数量	胜利数量	失败数量
100.dll	0	6	2