第二次大作业报告

四字棋博弈——信心上限树算法

**#实验简介：**

近年来，AlphaGo因打败人类高手而大红，人工智能的发展，尤其对弈的算法，也因此备受关注。我们在课程学习了各种对弈的算法，比如极大极小、alpha-beta剪枝、蒙特卡罗树算法，并要借此作业来实现个人的AI算法来给，并以打败课程提供尽可能多的AI为目标。

**#对弈规则：**

基本规则是双方轮流落子、每次只能在列顶落子、先完成四连子（或四连子以上）者胜，若直到棋盘满尚未决出胜负，则为平局。

此外，为了防止必胜策略，此实验也被设置成：棋盘大小M行N列会随机产生在9到12的闭区间内；不可落子点（棋不能放的地方）会随机产生在这里面的任一点。

**#基本算法：**

考虑到alpha-beta剪枝要设置一个评价函数，或者为可能下的点评分，是很麻烦且需要极多相关下棋的专业知识的，不禁觉得是个很费劲的算法，而且这个方法，如果我们不是专家，或者说如果无法穷举所有的可能下法，可能就无法真正意义上成为高手，所以我从一开始就下定决心要写一个蒙特卡洛算法的演进——**信心上限搜索树算法**。这个是一个最符合人类高手思路的算法了，它通过乱数决定是否演练已知节点，或者要扩展新的未知节点，然后往最后的胜负结果前进，这是一个随机扩大的树，通过大量的模拟，它最后可以判断，下一个决策点，哪个点最好。

整个算法的灵魂，即选择节点的公式如下：

可以看到公式主要有两项以及，在模拟节点的初期，由于还未知哪一个点的reward最好，前一项会比较小，所以更容易扩展新节点，而不会扩展有胜率的点。但越到后面，即的几率渐渐累积了起来（可能到了几千到一万步的是情况），而变相的，前项会变得更重，而相应地会更多以更大的几率去扩展胜率大的节点，在最后的模拟阶段，我们将可以很大的比例地把胜率大的节点模拟数量成长起来，以真正确保这个节点是好的。

到了决定要下哪一步棋的情况，我们只管选最大胜率就好，也就是说，c = 0的情况（不需要再考虑新节点，或其他较少被模拟的节点）。

除此之外，为了防止出现被0除的情况（即第一次开展节点的情况），我们也对这个公式做了一些变化，公式如下：

这基本上是对分母做了一次平移，我所选取的err的量是0.001。从数学上来看，这都不是很重要的量，只要足够小，就可以达到我们的目的，真正决定性的是c参数。当c越大，则我们选择新节点的可能会更大。以打败最强的level100为目标，来调整我的参数，所以我模拟了跑了很多局来去得到最优的c值。各种c值与100.dll对弈的结果如下：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C值 | 0.5 | 0.55 | 0.5655 | 0.5851 | 0.6 | 0.6125 | 0.625 |
| 对弈轮数 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| 胜率 | 0.4 | 0.6 | 0.75 | 0.55 | 0.8 | 0.45 | 0.45 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C值 | 0.65 | 0.6563 | 0.6625 | 0.6688 | 0.675 | 0.6875 | 0.7 |
| 对弈轮数 | 10 | 7.5 | 10 | 10 | 10 | 10 | 10 |
| 胜率 | 0.55 | 0.73 | 0.8 | 0.8 | 0.65 | 0.6 | 0.5 |

最后我选择了c=0.6625，因为这个在c=0.6563到0.6688这个区间内，有比较稳定且高的胜率，c=0.6的点比较附近比较低，所以看来不是很稳定。

**#实验结果：**

接下来也与各个dll进行了模拟，结果如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 对手 | 2.dll | 24.dll | 50.dll | 74.dll | 90.dll |
| 对弈轮数 | 10 | 10 | 10 | 10 | 10 |
| 胜率 | 1 | 1 | 1 | 1 | 0.9 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 对手 | 92.dll | 94.dll | 96.dll | 98.dll | 100.dll |
| 对弈轮数 | 10 | 10 | 10 | 10 | 10 |
| 胜率 | 1 | 0.9 | 0.9 | 0.9 | 0.8 |

进行更多轮的对弈模拟，将可以更加确保我的算法性能的稳定。再者，我的所有模拟过程中，都没有出现过超时的过程，倒是发现100.dll出现过一次超时的情况。因此最后的结果是，也很期待我最后的算法可以有效地击败所有助教的dll。