四子棋对弈

2018011359 计84 乐阳

算法思路

本实验实现了一个四子棋对弈AI，主要采用的算法是蒙特卡洛搜索与信心上界树。历经三个版本的优化，最终实现与全部测例对抗胜率在97%以上、与100号测例对抗胜率在80%以上的成果，且在天梯中长期处于前三甲（账号：yiqunyang）。下面分别介绍三个版本AI的思路。

版本一：信心上界树

对抗100号测例胜率：40%

选用信心上界树算法而非\alpha-\beta剪枝的原因是前者实现优雅且避免了设计评估函数的困难。第一版AI完整复现了课上介绍的信心上界树算法。

信心上界（UCB）算法面对的问题是从多个收益不同的选择中，如何在有限次尝试中选择可能受益最大的决策。具体来讲，对于第 j 种选择，有如下的信心上界公式：

其中 I_j 就是所谓的“信心上界“，在算法开始时先将每个选择尝试一遍，然后每次选择信心上界最大的选择尝试。 n 为当前尝试的总次数，\overline{X_j}是尝试第 j 种选择获得收益的均值，T_j(n) 是已经尝试第 j 种选择的次数。C为一个可调的参数，用于权衡已知高胜率选择的利用和未充分了解的选择的探索。

蒙特卡洛搜索树结合信心上界就得到了**信心上界树（UCT）**。一次蒙特卡洛搜索有四个阶段：

1. 选择：从根节点出发自上而下选择子节点。按照极大极小过程，每次选择相对当前节点信心上界最大的子节点。直到访问到一个可扩展节点（有可能的子节点未被创建）或终止节点（已经分出胜负）。
2. 扩展：如果第1步得到的节点是可扩展节点，则创建该节点的一个新的子节点。
3. 模拟：对于新扩展出的节点，随机模拟下棋直到分出胜负，胜、负、平的收益分别为1,0,-1。
4. 回溯：向上传播收益，每向上传播一层收益取相反数（相邻层的节点属于对弈双方）。

以下代码段展示了一次蒙特卡洛模拟的过程。

```
MCST_Node* v = treePolicy(); //寻找待扩展节点并扩展
int gain = defaultPolicy(v); //蒙特卡洛模拟
v->Backward(gain); //反向传播收益
```

计算时长耗尽后，选择胜率最高的决策即可。UCT算法有相当的普适性，仅仅知道游戏规则而没有任何关于策略的知识（笔者就是这样）就能够写出一个性能不错的AI。源码中的`AI_Engine`类实现了信心上界树的规则，其中每个节点为`MCST_Node`类型。

在不改变算法逻辑的前提下，设计程序时采用了几个策略来优化性能。

- 节省内存：每个节点只存储一步落子而非整个局面。使用`short`类型存储棋子坐标。
- 提高效率：每一次落子不重新建树，而是将树根移动到对应的子节点上。存储终止节点的胜负信

息，当TreePolicy选择到一个终止节点时不需要模拟而直接返回收益。

版本二：考虑迫手的模拟

对抗100号测例胜率：50%

"迫手"是棋类游戏的一个重要的策略，对于四子棋这种规则简单的游戏更甚。具体来讲，迫手是指在轮到本方下棋时出现的以下两种情况：

1. 有可以一步获胜的落子位置。
2. 对手有可以一步获胜的落子位置，本方必须抢先下在该位置以避免失利。

显然，第一种情况的优先级大于第二种。

版本二的核心策略是在蒙特卡洛模拟时考虑迫手来提高模拟效率。虽然在模拟中依次检查所有位置的获胜情况会使模拟次数有所下降，但按照四子棋的规则每次可以下的位置很少，因此完全检查获胜情况的开销是可以接受的（经过统计模拟次数减少约五倍）。更重要的是，这种方法很大程度上避免了不合理的模拟比如送死、错过一步致胜。方法`AI_Engine::checkForce`实现了检查迫手的功能。

添加迫手模拟后，版本二对阵版本一几乎完胜，对抗测例的胜率也有所上升。模拟次数减少，但有更合理的指导性，可以使蒙特卡洛模拟的效率提高。

版本三：终止节点的进一步考虑

对抗100号测例胜率：80%

在版本二的基础上，版本三针对迫手增加了很多新的特性，使算法的棋力大幅度提高。

在模拟阶段，如果在模拟的每一步都为迫手局面直到分出胜负，则该节点也被标志为**终止节点**。如果从某局面开始，每一步都是迫手直到最后，则该局面的结局就已经唯一确定了，因此标记位终止节点是合理的。此举能进一步提高模拟效率，剔除模拟中的非随机部分。更重要的是，由于终止节点能够看到多步，能够减少许多不必要的节点扩展。

在此基础上，修改信心上界选择子节点的策略，对必败的子终止节点**直接跳过**（完全失去信心），而不再考虑尝试；如果该节点的子节点中有必胜的终止节点，则**一定选择**。如果一个节点的所有子节点都是必败的终止节点（走投无路），则该节点也被标记为必败终止节点；如果一个节点选到了必胜的子节点，则该节点也被标记为必胜终止节点。此谓**终止节点的向上传递**。以上的叙述中胜负的概念是对某一方而言的，具体实现中记录的是对当前节点而言的胜负。以下代码段展示了上述逻辑的实现。

```
//------MCST_Node::bestChild------
if (current->terminal == 1){
    //子节点显示必输，证明选该点必赢
    best = current;
    break;
}
else if (current->terminal == 3){
    //子节点必赢，说明一定不选
    current = current->next;
    continue;
}
//---------------------------------
```

```
//------AI_Engine::treePolicy------
MCST_Node* bchild = current->bestChild(1.0);
if(bchild == nullptr){
    current->terminal = 1;//该节点没有可能获胜
}
else if (bchild->terminal == 1){
    current->terminal = 3;//该节点必获胜
}
//---------------------------------
```

版本三做出的优化看似简单，但是对搜索树的结构有相当大的修改，性能的提升也非常明显。

对抗结果

在线上平台与所有偶数号AI正反手各对抗一次，胜率为98%。其中面对第84号测例后手告负，面对第100号测例后手告负。其余对局全胜，不再一一列出。

批量测试结果如下图。账号用户名：yiqunyang，绑定邮箱：m18519660881@163.com，测试编号#363。