人工智能导论——四子棋 实验报告

滕启成 计16 2021010837

实验目的

利用对抗搜索算法实现四子棋游戏AI，通过调整、完善策略获得更高的胜率。

算法介绍

蒙特卡洛方法

蒙特卡洛方法是二十世纪40年代中期S.M.乌拉姆和J.冯·诺伊曼提出的一种随机模拟方法，它以概率统计中的相关理论为指导，通过（伪）随机数来解决计算问题。蒙特卡洛方法主要用于解决随机过程模拟以及随机变量分布参数估计等问题，一个经典的例子是蒲丰投针实验估计π的值。

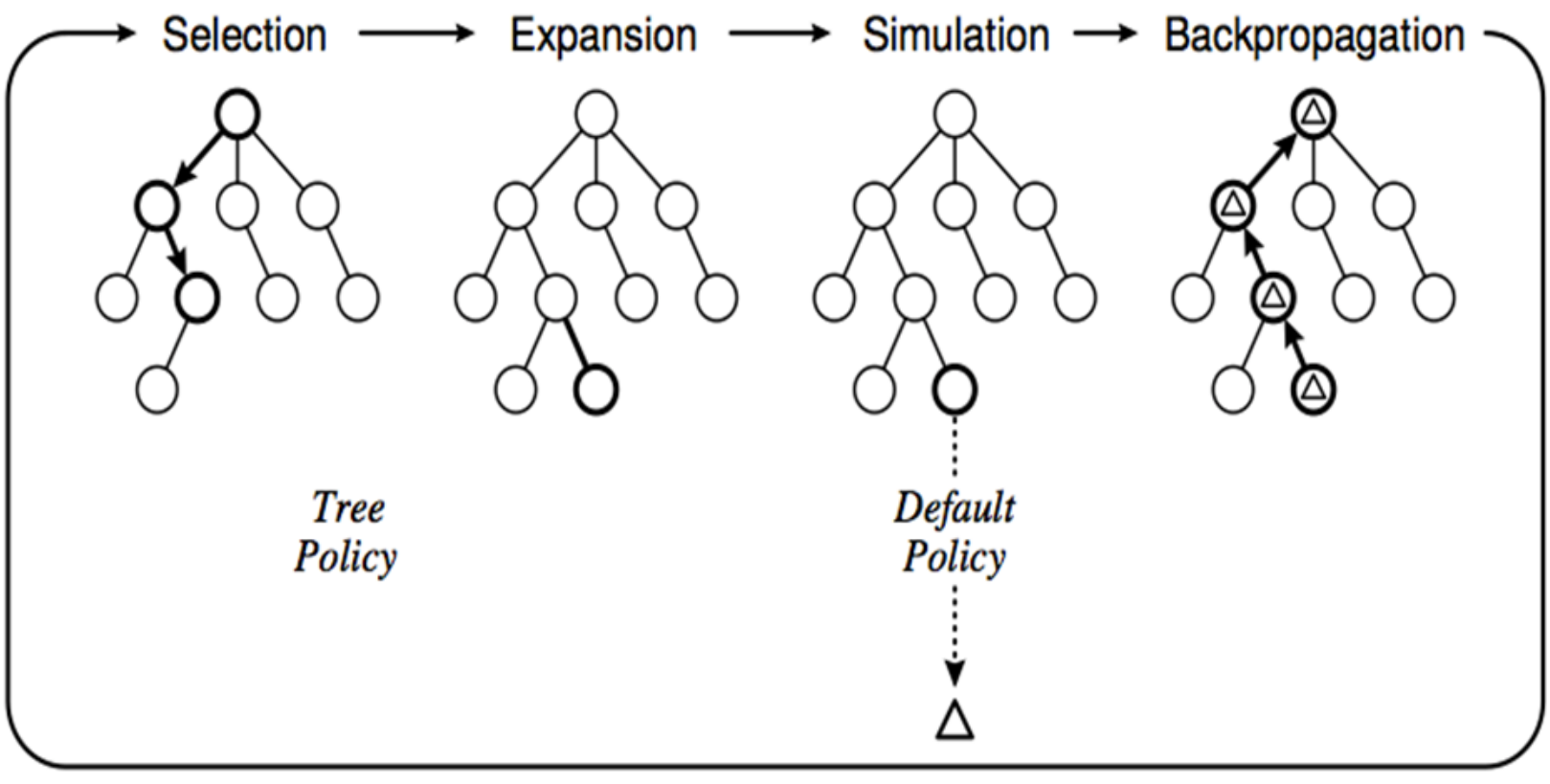
蒙特卡洛方法在棋类游戏中常常用于对局面进行评估。为了评估某一棋局，我们可以通过多次随机模拟得到胜率，从而根据胜率大小来判断局面的好坏。

蒙特卡洛树搜索（MCTS）与信心上限树（UCT）

蒙特卡洛树搜索算法将蒙特卡洛方法和搜索树结合起来，是一种用于决策的启发式搜索算法。蒙特卡洛树搜索算法主要包含以下五个步骤：

- 选择：对于已经充分扩展的节点，从所有子节点中选择一个当前最优的作为这一步的决策；
- 扩展：对于未充分扩展的节点，在未扩展的决策中随机选择一个，将该决策后的局面作为新节点插入当前节点的孩子中；
- 随机模拟：在新扩展的节点处进行随机模拟，得到胜率（收益）；
- 收益回传：模拟结束后，将收益从新节点一步步传到根节点，更新该路径上所有节点的收益信息；
- 决策：达到规定的时限后，比较根节点的所有决策，选择受益最大者作为最终决策。

过程图示如下（选自课件）：



在选择以及决策时，我们会遇到如何比较各个子节点好坏程度的问题。在选择过程中，我们偏向于扩展那些胜率高的节点，但同时也希望尽可能多地扩展决策信息相对较少节点；而决策过程中，我们则直接选择收益最大的决策即可。为了满足这一需要，我们引入如下的信心上限作为评估各个子节点的依据：

$$I_j = \bar{X}_j + c\sqrt{\frac{2\log N_j}{T_j(N)}} = 1 - \frac{Q_j}{N_j} + c\sqrt{\frac{2\log N_j}{\log N_{parent_j}}}$$

第一项表示胜率（由于子节点记录的是对方的胜率，因此应该取反），第二项用于扩展未知节点。参数c越大，表明越偏向扩展访问次数较少的节点，反之则越偏向扩展已知的高胜率节点。最终决策时取$c = 0$即可。

最终的UCT搜索算法总结如下：

算法 3：信心上限树算法（UCT）

function UCTSEARCH(s_0)
　以状态s_0创建根节点v_0;
　while 尚未用完计算时长 **do**:
　　$v_l \leftarrow$TREEPOLICY(v_0);
　　$\Delta \leftarrow$DEFAULTPOLICY(s(v_l));
　　BACKUP(v_l, Δ);
　end while
　return a(BESTCHILD($v_0, 0$));

function TREEPOLICY(v)
　while 节点v不是终止节点 **do**:
　　if 节点v是可扩展的 **then**:
　　return EXPAND(v)
　　else:
　　$v \leftarrow$ BESTCHILD(v, c)
　return v

function EXPAND(v)
　选择行动$a \in A(state(v))$中尚未选择过的行动
　向节点v添加子节点v'，使得$s(v')= f(s(v), a), a(v') = a$
　return v'

function BESTCHILD(v, c)
　return $argmax_{v' \in children\ of\ v}\left(\frac{Q(v')}{N(v')} + c\sqrt{\frac{2ln(N(v))}{N(v')}}\right)$

function DEFAULTPOLICY(s)
　while s不是终止状态 **do**:
　　以等概率选择行动$a \in A(s)$
　　$s \leftarrow f(s, a)$
　return 状态s的收益

function BACKUP(v, Δ)
　while $v \neq NULL$ **do**:
　　$N(v) \leftarrow N(v) + 1$
　　$Q(v) \leftarrow Q(v) + \Delta$
　　$\Delta \leftarrow -\Delta$
　　$v \leftarrow v$的父节点

实际实现时分别实现`Node`和`UCT`两个类，上述伪代码中的`BestChild`和`Expand`方法属于`Node`类接口，其余方法属于`UCT`类接口，`UCTSearch`方法返回当前局面的最终决策。

细节优化

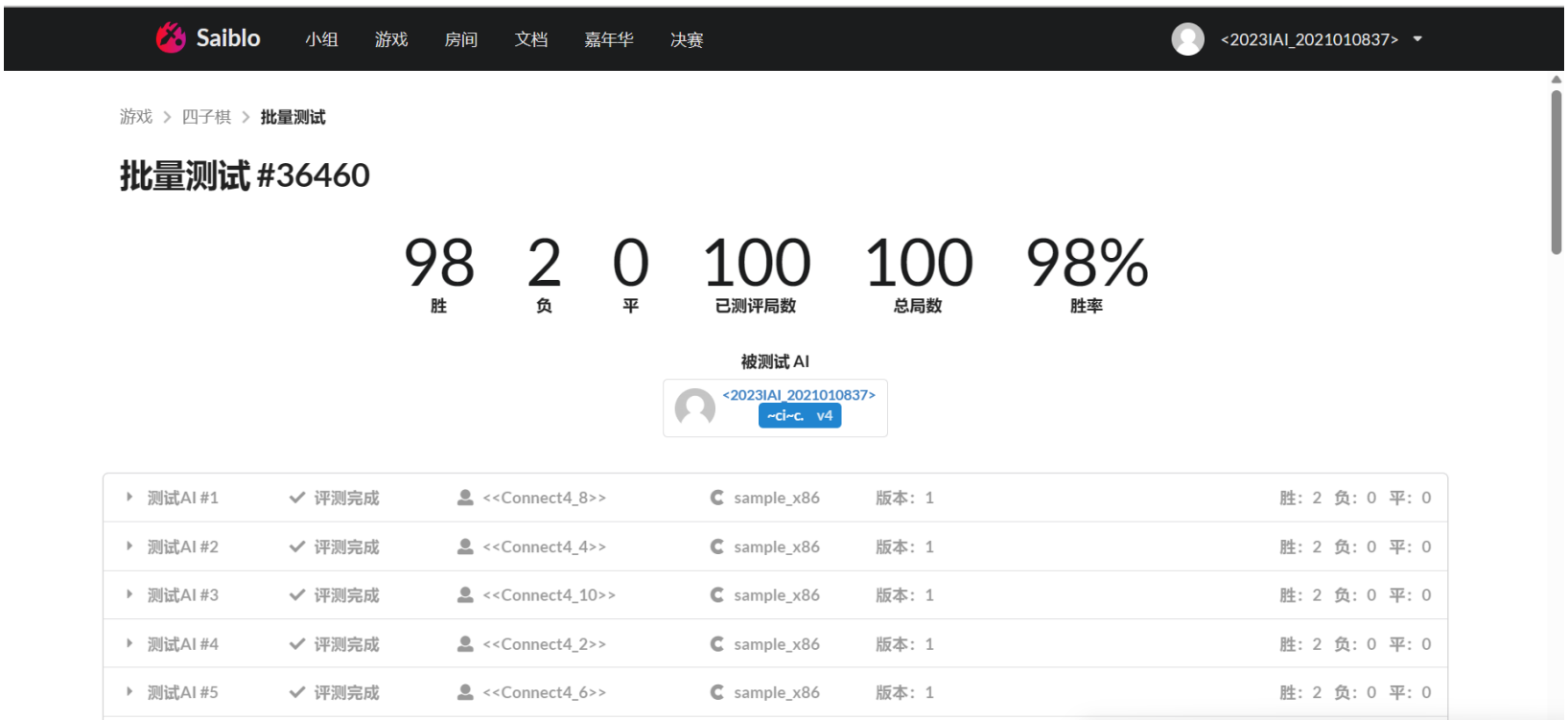
在上述策略的基础上，还有如下细节可以调整。

- 通过对棋局的观察可以发现，在中间落子的收益要大于在边缘落子。因此，我们可以在随机模拟时有意向棋盘中部靠拢，如随机一列发现不可落子时向中间寻找可落子的列，或者调高中间几列在随机模拟时的权重。
- 实际对战中棋手往往会特殊情况下采取如下贪心策略：当己方已经有三个子相连而且第四子为有效落子点，则直接骡子获得胜利（这个策略是显然的）；当对方已经有三个子相连而且第四子为有效落子点，则必须优先堵住对方。在一开始我曾经想过通过特判实现这一策略，但经过对于UCT搜索算法的测试后发现上述策略基本都会被搜索到并成为最终决策，因此在最终版本中并未进行特判。

- 关于参数c，在最终版本中取值为0.7，属于相对比较激进（偏向扩展访问次数少的节点）的策略。
- 关于时间上限，为了避免因为服务器性能波动问题超出3s的限制，最终版本中选择2s作为搜索的决策时限。实际测试中发现最长单步时间达到2.7s左右。时限过低容易导致搜索节点太少，而时限过高则可能TLE且容易造成过拟合，综合考虑两方面，2s是一个比较合适的时限。

测试结果

Saiblo平台上共进行三次测试，胜率分别为95%，92%，98%。胜率最高的截图如下：



链接：批量测试 #36460 - Saiblo

此外，针对最强的92~100号AI，也专门进行了测试，胜率如下：

对手AI	**92**	**94**	**96**	**98**	**100**
胜率	83%	36%	67%	77%	35%

总结

MCTS算法是一个权衡的过程：在有限的时间内，需要最大限度地找到增大模拟次数和实现各种细节策略的平衡。通过本次实验，我初步学习、了解了MCTS算法，实现了一个较为直观的对抗性游戏AI，并在策略编写和分析过程中体会这种权衡的思想方法。