人工智能导论第三次作业 四子棋实验报告

算法介绍

本次实验中，我使用了讲义上的信心上限树算法（UCT）来实现四子棋AI。该算法的核心思想是，通过模拟多次游戏，来估计每个状态的胜率，然后选择胜率最高的行动。在每次模拟中，算法会选择一个行动，然后随机选择行动，直到游戏结束。在每次模拟结束后，算法会根据游戏结果来更新每个状态的胜率。在选择行动时，算法会根据每个状态的胜率和访问次数来计算一个置信上限，然后选择置信上限最高的行动。算法的伪代码如下：

```
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←TREEPOLICY(v_0);
                    Δ←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← BESTCHILD(v,c)
                return v

    function EXPAND(v)
            选择行动a∈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'∈children of v)
((Q(v'))/(N(v'))+c√((2ln(N(v)))/(N(v'))))

    function DEFAULTPOLICY(s)
             while s不是终止状态 do:
                     以等概率选择行动a∈A(s)
                      s←f(s,a)
             return 状态s的收益

     function BACKUP(v,Δ)
               while v≠NULL do:
                       N(v)←N(v)+1
                       Q(v)←Q(v)+Δ
                       Δ←1-Δ
                       v←v的父节点
```

其中v包含四项基本信息，分别为其所对应的状态s(v)，所对应的来自父节点的行为a(v)，随机模拟收益Q(v)（例如获胜次数），以及节点的被访问次数N(v)。

代码框架

由于代码核心部分为UCT节点，因此可将UCT数的方法挂载于根节点下。故在框架基础上，我仅添加了$UCTNode.h$与$UCTNode.cpp$两个文件；单个节点存储了棋盘状态的静态变量及当前状态、可走节点、父子关系、胜负评估的成员变量，同时具备判断自己胜负情况、拓展状态的节点方法及UCT相关的方法。
代码仓库：

```
链接：https://git.tsinghua.edu.cn/saiblo/connect4/2023s/connect4-
-2023IAI_2020010916--cpp/-/tree/master
commit-id：2424c2c106d1a257be791ebc8653494ec22a9125
所在分支：master
```

算法优化

在扩展节点的Expand函数中，加入了对当前节点的必胜点、必走点判断：

- 必胜点：若当前节点存在己方三子相连且余下一子位置可下，则将该子下在余下位置，以确保胜利。
- 必走点：若当前节点存在对方三子相连且余下一子位置可下，则将该子下在余下位置，以防止对手胜利。
- 时间优化：若已对当前节点进行过拓展，则不再进行必胜点、必走点判断，以节省时间。
- 必败点：若对于己方的某个扩展，对方存在必胜点（该必胜点必定位于己方点位之上，仅需对对方下该点的情况进行判断），则该扩展为必败点，不再进行拓展。
- 故大致流程为：若未对当前节点进行过拓展，则进行必胜点、必走点判断，否则无需此判断，之后在可拓展节点中任选一点进行拓展，若该点为必败点，则不进行拓展并重新选择，除非该点为唯一可拓展点。

结果测试

在平台上使用批量测试功能与编号00-100中偶数的AI进行100场对战，结果如下：

测试编号37997

总结

在本次实验中，我成功地开发了一种基于UCT算法的重力四子棋AI。该算法结合了蒙特卡罗树搜索和启发式评估函数，使得AI能够在重力四子棋游戏中做出智能的决策。
通过此次实验，我对UCT算法有了更深的理解：通过模拟大量的游戏对局，评估每个行动的胜率，并使用Upper Confidence Bounds的策略来进行探索与利用的平衡，使AI能够在不完全信息的情况下，做出具有高胜率潜力的决策。
同时，我也认识到了剪枝的重要性：在实验中，我发现，若不对胜负点进行剪枝，AI会在必败点上浪费

大量时间，导致无法在规定时间内完成决策。因此，我在实验中加入了剪枝，使得AI能够在规定时间内完成决策。

然而我也发现，过多的剪枝会导致AI的效率下降，因此我仅在扩展节点时进行了剪枝，而在随机模拟时未进行剪枝。

此外，我对C++的构造析构、内存管理与时间控制有了更深刻的认识。

然而，我也意识到该AI仍存在大量不足，例如开局时由于搜索深度过深、能搜索的次数就会偏少。因此，仍存在着不少改进方向，例如添加启发式评估函数，使其更准确地评估游戏状态；尝试使用更高级的搜索算法或引入神经网络来提高AI的棋力；还可以通过增加更多的对战测试来验证AI的性能和鲁棒性。

总之，通过本次实验，我成功地设计和实现了基于UCT的重力四子棋AI，并验证了其在游戏中的性能优势。这为研究类似问题的AI算法提供了有价值的参考，同时也为我今后的研究工作提供了宝贵的经验。