四子棋 实验报告

致理书院 郭士尧 2022012406

1 基本思路

本实验要求实现四子棋游戏的 AI。为了避免使用必胜策略，游戏的棋盘尺寸随机，且棋盘会固定有一处随机障碍物。AI 需要在每一步三秒的时限内给出落子位置。

实现了两种模型：

- Alpha-Beta 剪枝：基于 Minimax 算法的搜索树剪枝算法
- MCTS 蒙特卡洛树搜索：基于随机模拟的搜索算法

2 模型实现

2.1 Alpha-Beta 剪枝

Minimax 树是决策树的一种，适用于两人对弈的零和博弈。其思想在于，其假设博弈双方均会最大化自己的收益，因此在搜索树的奇数层会选择最大值，而偶数层会选择最小值。实际上，如果将局面的收益定义的正负交替（即一方的收益为另一方的损失），则 Minimax 树可以简化（每一层节点都只需要最大化）。

但在实际应用中，Minimax 树的搜索空间非常大，因此需要进行剪枝。Alpha-Beta 剪枝算法在 Minimax 树的搜索过程中维护两个值（Alpha 和 Beta），用于修剪不必要的分支。

如果延续上面正规化分数的定义，则 Alpha-Beta 剪枝算法可以简化为：

```
function Alpha-Beta (state, α, β)
    result ← −∞
    foreach action in actions(state) do
        score ← Alpha-Beta (apply(state, action), −β, −α)
        result ← max(result, score)
        α ← max(α, score)
        if α ≥ β then
            break
    end
    return result
```

图 1 Alpha-Beta 剪枝算法伪代码

2.2 MCTS 蒙特卡洛树搜索

蒙特卡洛树搜索（MCTS）是一种基于随机模拟的搜索算法，适用于大规模的决策问题。其基本思想是通过随机模拟来评估每个局面的价值，并通过逐步扩展搜索树来找到最优解。

MCTS 的基本流程如下。这里针对编程实现对原版本做了一些简化：

```
function MCTS-Search (s_0)
    create root node v_0 with state s_0
    while within computational budget do
        Augment(v_0)
    end
    return Best-Child(s_0, 0)

function Update-Node (v, t)
    N(v) ← N(v) + 1
    Q(v) ← Q(v) + (t+1)/2

function Augment (v)
    if v is terminal then
        return R(v)
    if v is not fully expanded then
        a ← untried action from v
        v′ ← child of v with action a
        t ← −Default-Policy(v′)
    else
        v′ ← Best-Child (v, c_0)
        t ← −Augment(v′)
    end
    Update-Node(v, t)
    return t

function Best-Child (v, c)
    return argmax_{v′ is child of v} Q(v′)/N(v′) + c·sqrt(ln N(v) / N(v′))

function Default-Policy (s)
    while s is not terminal do
        choose a ∈ A(s) uniformly at random
        s ← apply(s, a)
    end
    return R(s)
```

图 2 MCTS 伪代码

其中 $N(v)$ 表示节点 v 的访问次数，$Q(v)$ 表示节点 v 的总胜场，$R(v)$ 为衡量节点胜负的函数（胜利为 1，失败为 -1，平局为 0），c 为探索系数，一般取 $c_0 = \sqrt{2}$。

3 优化尝试

3.1 估价函数设计

为了避免过深，Alpha-Beta 剪枝需要在一定的深度截断并返回当前界面的估价函数而非胜负。估价函数的设计是一个关键问题。

我使用的估价函数考虑了局面中的可能连线数。具体地，对局面上任意四子位置（横纵或斜向），若这四位置上仅有空白或某一方棋子，则该棋子方增加得分。有一个子加 1 分、两个加 10 分、三个加 100 分、四个加 20000 分。

此外，会统计棋盘上双方落子数量。为了鼓励在中间部分（非两侧边缘）落子，若落子在中间部分，则增加 2 分；否则加 1 分。

3.2 增量更新局面

每次骡子后无需重新扫描整个局面，只需要扫描落子所在的横纵与斜线即可判断胜负。类似地，上述的估价函数也可以增量地更新，不过需要考虑落子前后可能连线数的变化（可能需要减去原有的得分）。

3.3 落子顺序

在 Alpha-Beta 剪枝中，落子顺序会影响搜索树的剪枝效果。为了提高剪枝效率，我使用两个指针，初始都指向中间两列，然后逐渐向两侧扩展。例如，共有 7 列，则搜索顺序为 $3, 4, 2, 5, 1, 6, 0$。

3.4 MCTS 复用

在每次玩家行动后，MCTS 实际上是变为了原 MCTS 的一棵直接子树。为了避免重复计算，我在已经有上次计算的 MCTS 树的情况下，会尝试在子树中寻找是否有对应当前局面的节点。如果有，则直接从该节点开始进行搜索，而不是重新构建整个树。

3.5 MCTS 必胜子节点

若 MCTS 的某个子节点是终止节点（即胜利或失败），则该子节点应当成为父节点的唯一子节点，以避免在搜索过程中出现不必要的分支。具体地，在 Augment 函数中，若检查到当前节点是终止节点，则会将子节点清空并添加对应节点为唯一子节点。

3.6 `ArrayVec`

程序许多可变列表的长度都有可知的小上限（如行列最多 12），因此可以使用 `ArrayVec` 来避免不必要的堆分配。

3.7 `slab`

为了避免在 MCTS 中频繁地分配和释放内存，我使用了 `slab`（一种 Arena）来管理节点的内存。

4 实验结果

下面展示了不同模型不同参数对战 AI $2, 4, 6, ..., 100$ 的胜率：

模型	胜率
Alpha-Beta 剪枝	90%
MCTS, $c_0 = \sqrt{2}$	95%
MCTS, $c_0 = 2$	89%
MCTS, $c_0 = 1$	91%

综合选择了 MCTS 模型，$c_0 = \sqrt{2}$ 的参数。