**北京科技大学**

**人工智能课程实验报告**

学院： 计算机与通信工程学院 专业： 计算机科学与技术 信息安全

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **姓名** |  |  |  |  |  |
| **学号** |  |  |  |  |  |

**实验名称：**

基于博弈树α-β剪枝搜索的五子棋AI

**实验目的：**

通过这次实验了解蒙特卡洛树搜索、****策略以及α-β剪枝的原理；

并深入理解这些概念的编码方法。

**实验原理：**

本实验原理分为三部分讲述：

1. 概述蒙特卡洛树搜索基本原理；
2. 概述minmax算法；
3. 概述α-β剪枝的原理。
4. **蒙特卡洛树搜索**

蒙特卡洛树搜索是一种基于树结构的蒙特卡洛方法，所谓的蒙特卡洛树搜索就是基于蒙特卡洛方法在整个（等于决策次数，即树深度）空间中进行启发式搜索，基于一定的反馈寻找出最优的树结构路径（可行解）。概括来说就是，MCTS是一种确定规则驱动的启发式随机搜索算法。

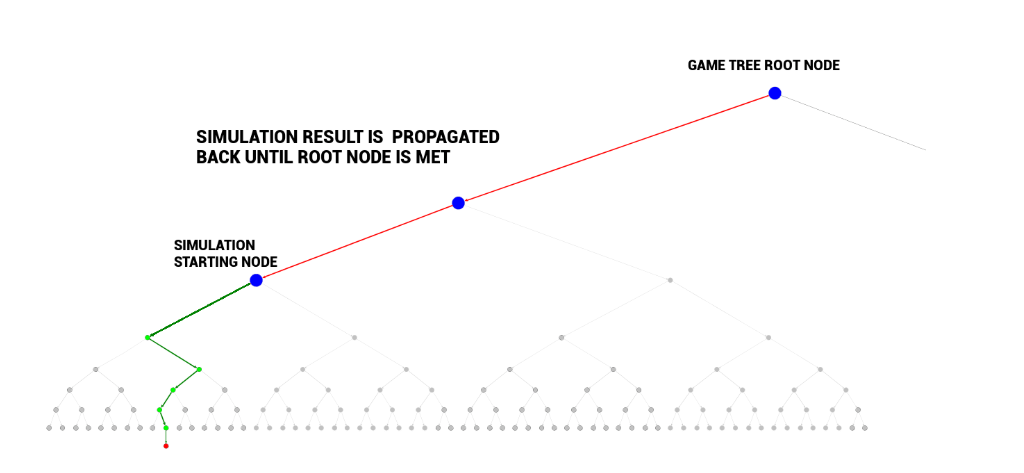
MCTS有5个主要核心部分：

1）树结构：树结构定义了一个可行解的解空间，每一个叶子节点到根节点的路径都对应了一个解（solution），解空间的大小为。

2）蒙特卡洛方法：MSTC不需要事先给定打标样本，随机统计方法充当了驱动力的作用，通过随机统计实验获取观测结果。

3）损失评估函数：有一个根据一个确定的规则设计的可量化的损失函数（目标驱动的损失函数），它提供一个可量化的确定性反馈。

4）反向传播线性优化：每次获得一条路径的损失结果后，采用反向传播（Backpropagation）对整条路径上的所有节点进行整体优化，优化过程连续可微。

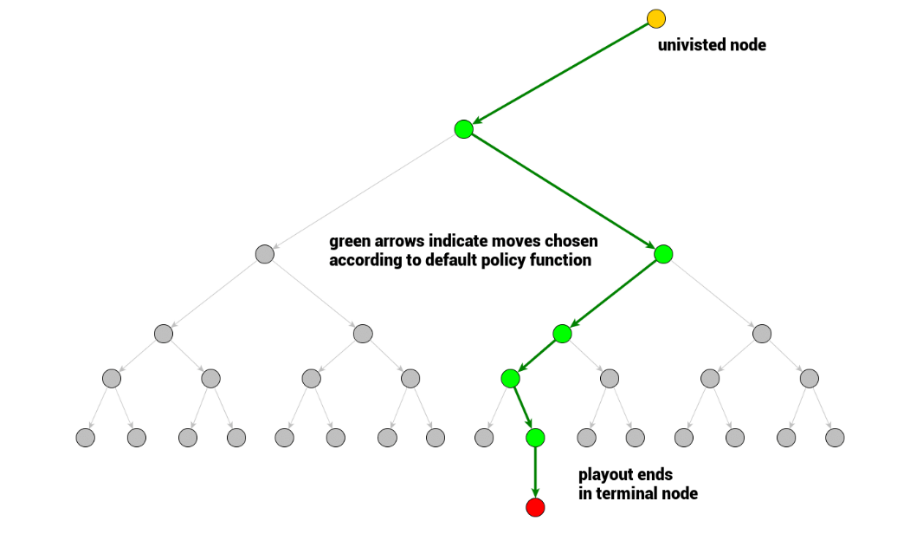


**图1-1 反向传播**

如上图所示，反向传播是从叶节点（模拟开始）到根节点的遍历。模拟结果被传送到根节点，并更新反向传播路径上每个节点的统计信息。反向传播保证每个节点的统计信息能够反映该节点所有后代的模拟结果。

5）启发式搜索策略：算法遵循损失最小化的原则在整个搜索空间上进行启发式搜索，直到找到一组最优解或者提前终止。

算法的优化核心思想是：在确定方向的渐进收敛（树搜索的准确性）和随机性（随机模拟的一般性）之间寻求一个最佳平衡。



**图1-2 蒙特卡洛树搜索的一次模拟**

如上图所示，我们按照蒙特卡洛树搜索所进行了一次模拟。模拟最简单的形式是一个从给定游戏状态到终端的随机移动序列。总是会产生一个结果，对于游戏来说就是获胜、失败或平局，但是广义上来说，这次模拟的合法结果可以是任意值。

1. **算法**

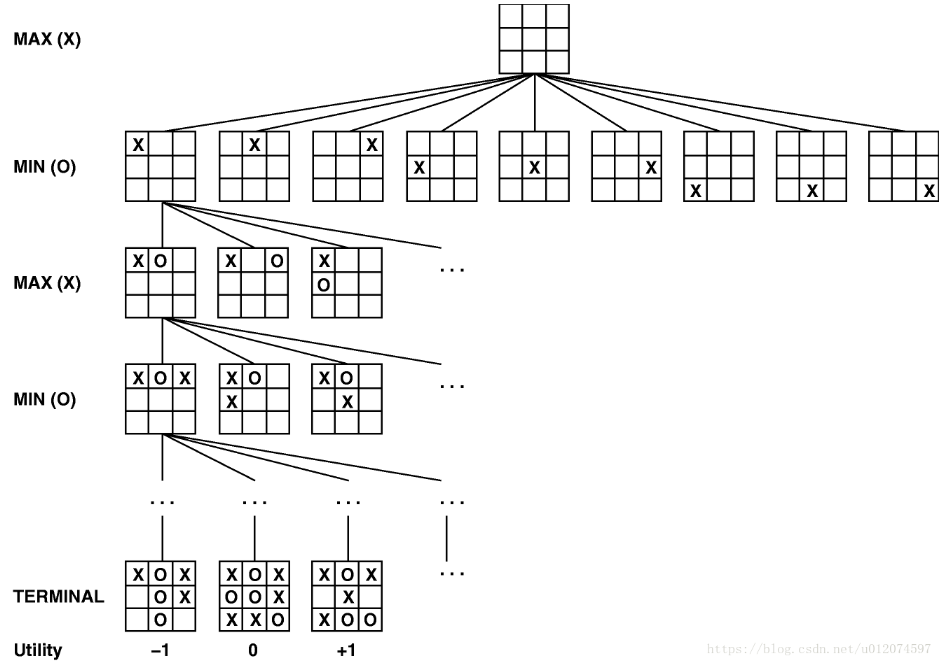
****算法又名极小化极大算法，是一种找出失败的最大可能性中的最小值的算法。该算法是一种零总和算法，即一方要在可选的选项中选择将其优势最大化的选择，而另一方则选择令对手优势最小化的方法。

****算法可以用以下递归表达式来描述：

其中，和是玩家A和B的效益函数，是给定当前状态和该状态下的动作产生下一个游戏状态的函数，是评估最终游戏状态的函数，是任意一个终端游戏状态。

简单来说，给定一个状态，你想要找到一个动作能够获得最大回报（假设你对手总是在最小化你的收益），这也是****算法的由来。我们所需要做的只是展开整个游戏树，并根据递归公式来进行反向传播。



**图1-3 算法举例**

如图1-3所示，假设有一个游戏，只有两个玩家参与。两个玩家便分别是MAX和MIN。每个玩家在各自的回合中，都有多个行动（actions）可选。MAX先手。图中，每条连线表示当前状态下可以采取的行动。当游戏进行到最后一轮时，可以评估该层各个状态的效用值（Utility），来表示玩家MAX的胜负多少，对于这一层的结果，MIN和MAX双方都是知道的。

MIN为了赢得游戏，一定会想办法让MAX的得分最低。游戏双方都会从游戏的结果开始回溯，来指导相应的状态下，自己该采取何种行动来最大化自己的目标。

若现在是MIN的回合，那么MIN在所有可选的actions中，一定会选择使得MAX的得分最少的行动。所以将子节点较小的效用值填写到相应的层中，表示游戏进行到相应的状态，MAX能达到的最大效用。从最后一层开始，一层层向上层回溯，这便是****算法的流程。

对于五子棋而言，就是统计目前的棋型，并累加分数。比如如果有4个子连起来了，那就给个很高的评分，因为下一步可能会赢。如果是3个子连起来，给个相对较低的评分，因为不一定就能赢，对手可能会堵，但是比只有2个子连在一起的得分要高。

对应评估局面上的分数，就是统计所有匹配的棋型得分并累加。这个分数的统计就叫做评估函数。****算法便是假设走某一个点后，计算局面的得分，然后取得分最大的那个点。

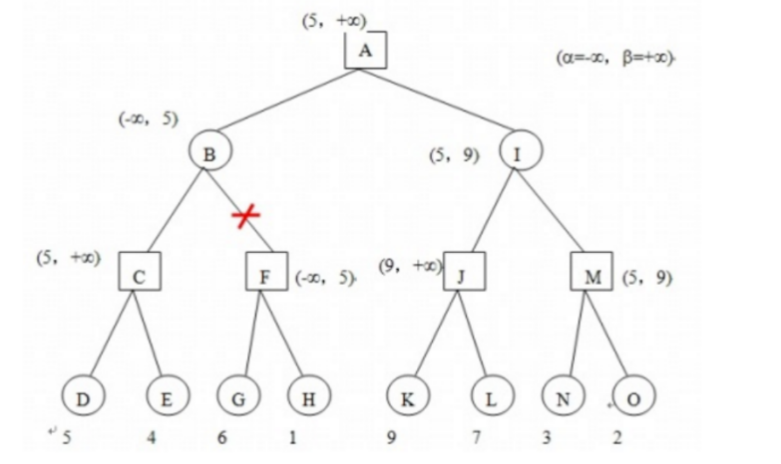
与上不同的是，本算法搜索深度为3，即搜索3步后得分最大的点。

1. **α-β剪枝**

α-β剪枝用于裁剪搜索树中不需要搜索的树枝，以提高运算速度。它基本的原理是：

当一个Min节点的β值≤任何一个父节点的α值时，剪掉该节点的所有子节点；

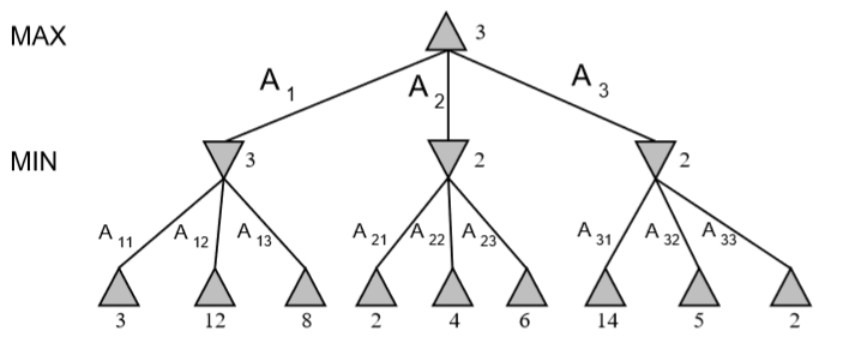
当一个Max节点的α值≥任何一个父节点的β值时，剪掉该节点的所有子节点。



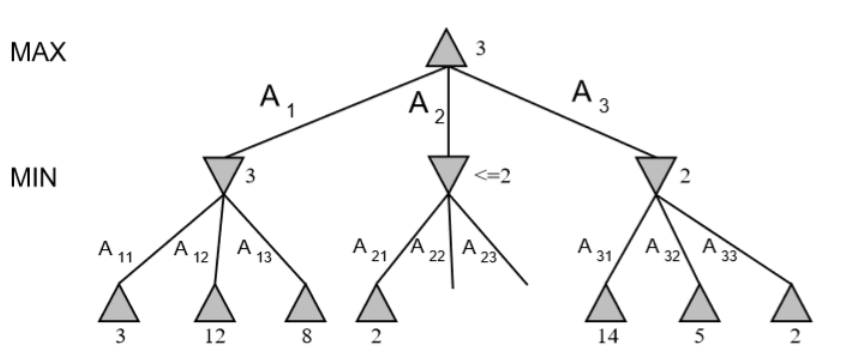
**图1-4 α-β剪枝举例**

如图1-4所示，α为已知的最大值，β为已知的最小值，分别初始化为-∞ 和+∞。

本文采用深度优先搜索。搜索到D的时候，局面得分是5，也就是说要搜索最大值，只可能会在（5，+∞）之间。同理，要搜索最小值，也只会在（-∞，5）之间。继续搜索，搜索到G时，F暂时等于6。因为F要找最大值，则F不可能再小于6。而B要找最小值，B的已知最小值是在（-∞，5）之间，则F不可能比6小。F这个分支可以不被搜索，这就是剪枝。同样对于另外一边的已知可能的极限范围β也是一样的情况，即遇到就算是搜索也是浪费时间的情况，就剪枝不进行搜索。这样就减少了很多不必要的搜索步骤，特别是若一开始就找到最有可能的极大极小值，则更能迅速的剪枝。



**（a）未使用α-β剪枝的算法**



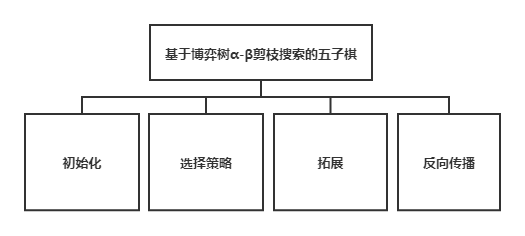
**（b）使用α-β剪枝的算法**

**图1-5 使用/未使用α-β剪枝的对比图**

如图1-5所示，α-β剪枝方法是对Minimax方法的优化，它们产生的结果是完全相同的，只不过运行效率不一样。在实际应用中，加上剪枝算法，计算机大约需要算个结果（其中，为分支数，为步数）。相比于仅用极小极大算法的，效率提高了很多。这也就意味着，如果在象棋比赛中仅使用极小极大的算法，计算机能往前评估7步，而加上α-β剪枝算法，计算机能往前评估14步。

**实验内容与步骤：**

本文的代码整体框架如下图所示：



**图2-1 整体框架图**

1）初始化模块

在开始阶段，搜索树只有一个节点，即根节点。搜索树中的每一个节点包含了三个基本信息：

* 1. 当前需要决策的局面R：即下一步可选的action list，action list是构成解空间的基本要素；
  2. 该节点被访问的次数：用于提供一个确定性的收敛方向判据；
  3. 累计评分：用于提供一个确定性的收敛方向判据。

2）选择策略模块

在选择策略模块，需要从父节点（首次选择从根节点开始），也就是要做决策的局面R出发向下选择出一个最急迫需要被拓展的节点N，即选择向哪个子节点方向生长。这部分采用****算法，具体的伪代码如下所示。

1. **int** MinMax(**int** depth) { // 函数的评估以白方的角度评估
2. **if** (SideToMove() == WHITE) { // 白方是“最大”者
3. **return** Max(depth);
4. } **else** { // 黑方是“最小”者
5. **return** Min(depth);
6. }
7. }
8. **int** Max(**int** depth) {
9. **int** best = -INFINITY;
10. **if** (depth <= 0) {
11. **return** Evaluate();
12. }
13. GenerateLegalMoves();
14. **while** (MovesLeft()) {
15. MakeNextMove();
16. val = Min(depth - 1);
17. UnmakeMove();
18. **if** (val > best) {
19. best = val;
20. }
21. }
22. **return** best;
23. }
24. **int** Min(**int** depth) {
25. **int** best = INFINITY;
26. **if** (depth <= 0) {
27. **return** Evaluate();
28. }
29. GenerateLegalMoves();
30. **while** (MovesLeft()) {
31. MakeNextMove();
32. val = Max(depth - 1);
33. UnmakeMove();
34. **if** (val < best) {
35. best = val;
36. }
37. }
38. **return** best;
39. }

3）拓展模块

在选择阶段结束时候，系统查找到了一个最迫切被拓展的节点N，以及一个尚未拓展的动作A。在搜索树中创建一个新的节点Nn作为N的一个新子节点。Nn的局面就是节点N在执行了动作A之后的局面。

1. **func (n \*MCTSNode) AddChild(move \*Pos, state \*State) \*MCTSNode {**
2. **child := &MCTSNode{**
3. **move, state.LastPlayer(), n, nil, 0, 0, state.GetMoves(),**
4. **}**
5. **n.Children = append(n.Children, child)**
6. **return child**
7. **}**

4）反向传播模块

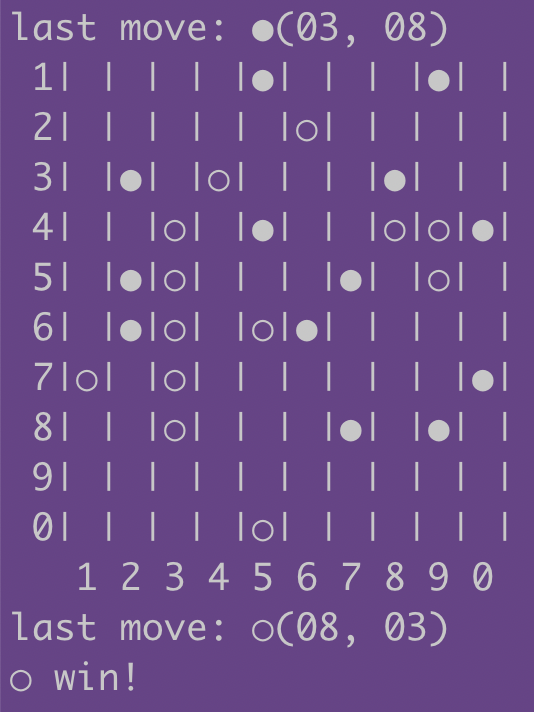
在Nn的模拟结束之后，它的父节点N以及从根节点到N的路径上的所有节点都会根据本次模拟的结果来修改自己的累计评分。注意，如果在选择环节中直接发现了一个游戏结局的话，根据该结局来更新评分。

每一次迭代都会拓展搜索树，随着迭代次数的增加，搜索树的规模也不断增加。当到了一定的迭代次数或者时间之后结束，选择根节点下最好的子节点作为本次决策的结果。

1. func UCT(root \*State, itermax int) (best \*Pos) {
2. // init a rootNode
3. rootNode := &MCTSNode{
4. LastMove: root.LastMove(), LastPlayer: root.LastPlayer(), UntriedMoves: root.GetMoves(),
5. }
6. // range itermax
7. for i := 0; i < itermax; i++ {
8. node := rootNode
9. state := root.Clone()
10. // 搜索到头了
11. for len(node.UntriedMoves) == 0 && len(node.Children) > 0 {
12. node = node.NCTSelectChild()
13. state.NextP(node.LastMove)
14. }
15. // 再往下一层
16. if len(node.UntriedMoves) > 0 {
17. m := node.PopTry()
18. state.NextP(m)
19. node = node.AddChild(m, state)
20. }
21. // 我是否赢得了比赛
22. winner, over := state.Winner()
23. for !over {
24. state.NextRandom()
25. winner, over = state.Winner()
26. }
27. for node != nil {
28. node.Update(winner)
29. node = node.Parent
30. }
31. }
32. options := rootNode.Children
33. sort.Slice(options, func(i, j int) bool {
34. return options[i].Visited < options[j].Visited
35. })
36. return options[len(options)-1].LastMove
37. }

**实验数据：**

我们将两个AI搜索深度为定义为30，两个AI经过N个回合的博弈，AI1赢得了比赛，结果如下。



**图2-2 博弈结果**

**实验数据分析：**

AI1落子X，然后AI2落子Y，并且两个AI交替随机，下棋，并且给下的棋打分，经过30层搜索之后，落子分数最高的点为真的落子的点。过程中使用α-β剪枝算法，如果满足特定条件的分支将不再对弈，因此将棋盘设置为15\*15=225也能在短时间内快速进行一局博弈。

**实验结论：**

在本次实验中，本小组完成了基于博弈树α-β剪枝搜索的五子棋AI，系统了解了蒙特卡洛树搜索、算法和α-β剪枝。并通过编程将三者结合起来，运行情况如实验数据所示。可以见得，本小组已完成这次实验相应功能。

**工作量和任务分配：**