AI-hw4

张芷苒 PB21081601

- 5.8
- 5.9
- 5.13

5.8

> 5.8 Consider the two-player game described in Figure 5.17. a. Draw the complete game tree, using the following conventions: - Write each state as (s_A, s_B), where s_A and s_B denote the token locations. - Put each terminal state in a square box and write its game value in a circle. - Put loop states (states that already appear on the path to the root) in double square boxes. Since their value is unclear, annotate each with a "?" in a circle. b. Now mark each node with its backed-up minimax value (also in a circle). Explain how you handled the "?" values and why. c. Explain why the standard minimax algorithm would fail on this game tree and briefly sketch how you might fix it, drawing on your answer to (b). Does your modified algorithm give optimal decisions for all games with loops? d. This 4-square game can be generalized to n squares for any $n > 2$. Prove that A wins if n is even and loses if n is odd.

Part (a): Complete Game Tree

1. Initial state: $(1, 2)$
2. Player A moves to 2: $(2, 2)$
3. Player B can only move back to 1: $(2, 1)$
4. Player A moves to 3: $(3, 1)$
5. Player B moves to 2: $(3, 2)$
6. Player A jumps to 1: $(1, 2)$ – a loop state.
7. Player A moves to 4: $(4, 2)$ – a terminal state, A wins (+1).
8. If A jumps back to 1 initially: $(1, 2)$ – another loop state.

- Draw each non-terminal state as (s_A, s_B).
- Terminal states: Put in a square box, with the value (+1 or -1) in a circle.
- Loop states: Double square boxes with a "?" in a circle.

Part (b): Minimax Values

- For terminal states, the value is the result of the game (+1 for A win, -1 for B win).
- For non-terminal states, consider the best achievable outcome for the player whose turn it is, assuming the opponent plays optimally.
- Loop states ("?") are problematic because they can lead to infinite loops. For this exercise, a heuristic approach could be to assign a value based on who benefits from the loop or to enforce a rule such as a draw after a certain number of loops.

Part (c): Standard Minimax Algorithm Issue and Fix

The standard minimax algorithm fails because of the loops, potentially causing an infinite recursion. The modified algorithm could include:

- A visitation count to detect loops.
- Assign a value to loop states, possibly a draw (0) or based on which player the loop favors.
- Use a depth limit to force a decision after a certain number of moves.

Part (d): Generalization to (n) Squares

Proof:

- If n is even, A wins:
  - A always makes the last move to n, since both players move alternately.
- If n is odd, A loses:
  - B always makes the last move to n, as A starts but there is an odd number of moves to make.

For an even n, A controls the last move, and for an odd n, B controls the last move, given optimal play from both sides.

5.9

> 5.9 This problem exercises the basic concepts of game playing, using tic-tac-toe (noughts and crosses) as an example. We define X_n as the number of rows, columns, or diagonals with exactly $nX's$ and no $O's$. Similarly, O_n is the number of rows, columns, or diagonals with just $nO's$. The utility function assigns +1 to any position with $X_3 = 1$ and -1 to any position with $O_3 = 1$. All other terminal positions have utility 0 . For nonterminal positions, we use a

> linear evaluation function defined as
> $\text{Eval}(s) = 3X_2(s) + X_1(s) - (3O_2(s) + O_1(s))$.
>
> a. Approximately how many possible games of tic-tac-toe are there?
>
> b. Show the whole game tree starting from an empty board down to depth 2 (i.e., one X and one O on the board), taking symmetry into account.
>
> c. Mark on your tree the evaluations of all the positions at depth 2.
>
> d. Using the minimax algorithm, mark on your tree the backed-up values for the positions at depths 1 and 0 , and use those values to choose the best starting move.
>
> e. Circle the nodes at depth 2 that would not be evaluated if alpha-beta pruning were applied, assuming the nodes are generated in the optimal order for alpha-beta pruning.

Part (a): Number of Possible Games

For tic-tac-toe, there are approximately 255,168 possible game sequences considering all possible moves and turn orders, without accounting for symmetry or early wins. Accounting for these, there are about 26,830 unique positions.

Part (b): Game Tree to Depth 2

- **Depth 0**: Empty board.
- **Depth 1**: 'X' moves in the center or a corner.
- **Depth 2**: 'O' responds:
  - 'X' in center: 'O' in a corner or on an edge.
  - 'X' in a corner: 'O' in the center, adjacent edge, or opposite corner.

Part (c): Evaluations at Depth 2

- **X in Center, O in Corner/Edge**:
  - **Eval(s)**: $3 \times 0 + 4 - (3 \times 0 + 4) = 0$
- **X in Corner, O in Center/Adjacent Edge/Opposite Corner**:
  - **Eval(s)**: 0 (Similar to the above scenario, no immediate threats from either player)

Part (d): Minimax Values

- **Depth 1** (X to move): Choose the move leading to the highest evaluation.

- **Depth 0** (O to move): Choose the move leading to the lowest maximum score from depth 1.

Part (e): Alpha-Beta Pruning

Assuming optimal order of node generation, nodes at depth 2 that do not improve upon previously discovered scores can be pruned, depending on the actual node values and sequence of moves.

5.13

> 5.13 Develop a formal proof of correctness for alpha-beta pruning. To do this, consider the situation shown in Figure 5.18. The question is whether to prune node n_j, which is a maxnode and a descendant of node n_1. The basic idea is to prune it if and only if the minimax value of n_1 can be shown to be independent of the value of n_j. > > a. Mode n_1 takes on the minimum value among its children: $n_1 = \min\left(n_2, n_{21}, \ldots, n_2 b_2\right)$. Find a similar expression for n_2 and hence an expression for n_1 in terms of n_j. > > b. Let l_i be the minimum (or maximum) value of the nodes to the left of node n_i at depth i, whose minimax value is already known. Similarly, let r_i be the minimum (or maximum) value of the unexplored nodes to the right of n_i at depth i. Rewrite your expression for n_1 in terms of the l_i and r_i values. > > c. Now reformulate the expression to show that in order to affect n_1, n_j must not exceed a certain bound derived from the l_i values. > > d. Repeat the process for the case where n_j is a min-node.

Part (a): Expression for n_1 in Terms of n_j

- Node n_1 takes the minimum value among its children.
- Node n_2 is a max-node and will take on the maximum value of its children: $n_2 = \max(n_{21}, n_{22}, \ldots, n_j, \ldots, n_{2b_2})$.
- Therefore, n_1 can be expressed as: $n_1 = \min(\max(n_{21}, n_{22}, \ldots, n_j, \ldots, n_{2b_2}), n_{11}, \ldots, n_{1b_1})$.

Part (b): Expression for n_1 in Terms of l_i and r_i

- Let l_2 be the maximum value among the nodes to the left of n_j at depth 2, whose minimax value is already known.
- Let r_2 be the maximum value among the unexplored nodes to the right of n_j at depth 2.
- Rewrite n_1 as: $n_1 = \min(\max(l_2, n_j, r_2), l_1)$, where l_1 is the minimum value of nodes to the left of n_1 at depth 1 that are already evaluated.

Part (c): Bound for n_j to Affect n_1

- For n_j to affect n_1, it must be such that $\max(l_2, n_j, r_2)$ is lower than the next best alternative (here represented as l_1). This is critical because n_1 is taking the minimum of these maximums.
- Therefore, n_j can only affect n_1 if $n_j < l_1$. Any value for n_j greater than or equal to l_1 would make $\max(l_2, n_j, r_2) \geq l_1$, which would not be the minimum value influencing n_1.

Part (d): Case Where n_j is a Min-Node

- When n_j is a min-node, the expression for n_2 becomes: $n_2 = \min(n_{21}, \ldots, n_j, \ldots)$, which means that n_2 is now a minimum among its children.
- n_j can only affect the value of n_1 if n_j is greater than the known maximums of the nodes to the left of n_2 at depth 2, denoted as l_2. This is because n_1 will be the minimum of these values and n_2's value.
- Specifically, n_j can influence n_1 if $n_j > l_2$, making it a potentially lower value than what l_2 can offer, thus potentially lowering n_1 if n_2 influences n_1's value.