

Usman Javed

Roll No: 023

Minimax Algorithm — Improved Code and Explanation

Step-by-step Explanation

1. Problem overview: Minimax is a recursive algorithm used for decision-making in game trees. It assumes two players: a maximizing player (tries to maximize the score) and a minimizing player (tries to minimize the score). Leaves of the tree contain final scores (utility values). 2. Tree structure: For a full binary tree of depth D, there are 2^D leaf nodes. Each internal node represents a game state where a player chooses between two moves (left or right child). 3. Base case: When recursion reaches the maximum depth (a leaf), the algorithm returns the leaf's score. This value represents the final outcome from that branch. 4. Maximizing step: If it is the maximizing player's turn at a node, the algorithm recursively computes the values of both children and returns the maximum of those two values. 5. Minimizing step: If it is the minimizing player's turn at a node, the algorithm recursively computes the values of both children and returns the minimum of those two values. 6. Propagation: Values propagate up the tree. Each parent node's value is determined by either max or min of its children, depending on the player's turn. The root's returned value is the optimal achievable value assuming perfect play from both sides. 7. Example (brief): For leaf scores [3, 5, 2, 9, 3, 5, 2, 9] and tree depth 3, the algorithm evaluates bottom-up to compute the optimal value at the root.

How it works (concise)

- Start at the root (depth 0) with the maximizing player. - Recursively explore both child nodes until reaching leaves. - At each leaf return its score. - At internal nodes, return $\max(\text{children})$ if maximizing, $\min(\text{children})$ if minimizing. - The root's result is the optimal value for the maximizing player.

What I improved in the code

- Clearer variable names: depth, index, is_maximizing, max_depth make the function intent obvious.
- Added docstring and structured comments for readability.
- Used math.log2 and explicit int conversion for tree depth calculation to avoid float issues.
- Encapsulated driver code inside the `if __name__ == "__main__":` block for best practice.
- Consistent indentation and preformatted code block for easy reading and copying.
- Removed unnecessary or unclear comments and kept only relevant explanations (PDF includes explanation separately).

Improved Code (ready to run):

```
import math

def minimax(depth, index, is_maximizing, scores, max_depth):
    """Recursive implementation of the Minimax algorithm."""
    # Base case: reached a leaf
    if depth == max_depth:
        return scores[index]

    if is_maximizing:
        # Maximizer chooses the child with the maximum value
        return max(
            minimax(depth + 1, index * 2, False, scores, max_depth),
            minimax(depth + 1, index * 2 + 1, False, scores, max_depth)
        )
    else:
        # Minimizer chooses the child with the minimum value
        return min(
            minimax(depth + 1, index * 2, True, scores, max_depth),
            minimax(depth + 1, index * 2 + 1, True, scores, max_depth)
        )
```

```
if __name__ == "__main__":
    scores = [3, 5, 2, 9, 3, 5, 2, 9]
    tree_depth = int(math.log2(len(scores)))
    optimal_value = minimax(0, 0, True, scores, tree_depth)
    print("The optimal value is:", optimal_value)
```