AIML - EXPERIMENT 4

NAME: AYUSH BODADE

UID: 2021300015

BATCH: A1

Problem Definition:

The problem addressed in this code is to find the optimal value for a given game state in a two-player game, such as chess or tic-tac-toe. The goal is to determine the best possible outcome for the player making a move, assuming that both players play optimally. The code implements the Alpha-Beta Pruning algorithm to efficiently search through the game tree and identify the best move.

Theory:

The Alpha-Beta Pruning algorithm is a search algorithm commonly used in two-player games to reduce the number of nodes evaluated in the game tree. It is based on the Mini-max algorithm, which aims to find the optimal move for a player by considering all possible future moves and their outcomes.

Here's a brief overview of how the code applies the Alpha-Beta Pruning algorithm:

1. **Initialization:** The code initializes two constants, `POSITIVE_INFINITY` and `NEGATIVE_INFINITY`, representing positive and negative infinity values. These are used for initializing the alpha and beta values.
2. **Recursive Mini-max Function:** The `alphabeta` function is a recursive function that evaluates the game tree. It takes parameters such as the current depth, node index, a boolean indicating whether it's the maximizing player's turn, the values associated with each node, and alpha and beta values that represent the best values found for the maximizing and minimizing players so far.
3. **Termination Condition:** If the depth of the tree reaches a predefined limit (in this case, 3), the function returns the value associated with the current node.
4. **Maximizing Player's Turn:** When it's the maximizing player's turn, the code iterates through the possible child nodes, calculating their values recursively. It updates the `bestValue` and `alpha` as it goes along. The algorithm prunes branches when it finds a `beta <= alpha` condition, indicating that further exploration of that branch is unnecessary.
5. **Minimizing Player's Turn:** When it's the minimizing player's turn, the code again iterates through child nodes, calculating their values and updating `bestValue` and `beta`. It also prunes branches when it finds `beta <= alpha`.
6. **Optimal Value:** The optimal value for the root node is determined by calling the `alphabeta` function with appropriate initial values for alpha and beta, representing negative and positive

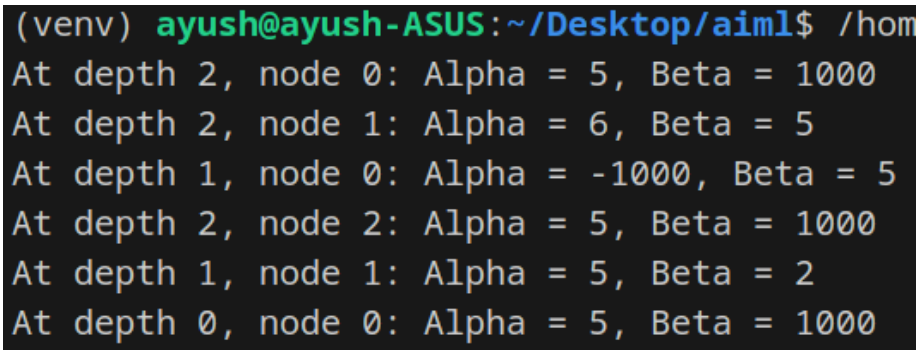
infinity, respectively. The algorithm efficiently prunes branches that will not affect the final result, leading to improved performance in evaluating game positions.

Code:

```
POSITIVE_INFINITY, NEGATIVE_INFINITY = 1000, -1000
def alphabeta(depth, nodeIndex, isMaximizingPlayer,
            values, alpha, beta):
    if depth == 3:
        return values[nodeIndex]
    if isMaximizingPlayer:
        bestValue = NEGATIVE_INFINITY
        for i in range(0, 2):
            childValue = alphabeta(depth + 1, nodeIndex * 2 + i,
                                   False, values, alpha, beta)
            bestValue = max(bestValue, childValue)
            alpha = max(alpha, bestValue)
            if beta <= alpha:
                break
        print(f"At depth {depth}, node {nodeIndex}: Alpha = {alpha}, Beta = {beta}")
        return bestValue
    else:
        bestValue = POSITIVE_INFINITY
        for i in range(0, 2):
            childValue = alphabeta(depth + 1, nodeIndex * 2 + i,
                                   True, values, alpha, beta)
            bestValue = min(bestValue, childValue)
            beta = min(beta, bestValue)
            if beta <= alpha:
                break
        print(f"At depth {depth}, node {nodeIndex}: Alpha = {alpha}, Beta = {beta}")
        return bestValue

if __name__ == "__main__":
    values = [3, 5, 6, 9, 1, 2, 0, -1, 8, -3, 7, 4, -2, -4, 10, 12]
    print("The optimal value is :", alphabeta(0, 0, True, values, NEGATIVE_INFINITY, POSITIVE_INFINITY))
```

Output:



Conclusion:

This experiment demonstrates the Alpha-Beta Pruning algorithm's efficient application in finding optimal game moves. Key points to take away:

- Alpha-Beta Pruning significantly reduces search tree exploration, making it vital for game AI.
- It follows the Mini-max strategy, aiming to maximize the player's outcome while considering the opponent's best moves.
- Alpha and beta values help prune branches, optimizing the search.
- Termination conditions limit the tree depth for practical use.

In summary, Alpha-Beta Pruning is an essential tool for AI in two-player games, enhancing decision-making efficiency.