This code implements the Alpha-Beta pruning algorithm for a two-player game tree. It recursively explores the game tree to find the optimal move for the maximizing player, while pruning branches that cannot influence the final decision to improve efficiency. The tree is represented as a dictionary where internal nodes have child nodes, and leaf nodes contain game values. The function alternates between maximizing and minimizing players, updating the alpha (best max value) and beta (best min value) thresholds to prune unnecessary branches and return the optimal score for the root node.

```
def alpha_beta(node, depth, alpha, beta, maximizingPlayer, tree):
    # If node has no children (leaf node) or children are leaf values
(ints)
    if node not in tree:
        return node  # node is an integer leaf value

    # Check if children are leaf values (integers)
    if isinstance(tree[node][0], int):
        # Return max or min of these leaf values depending on player
        if maximizingPlayer:
            return max(tree[node])
        else:
            return min(tree[node])

    if maximizingPlayer:
        maxEval = float('-inf')
        for child in tree[node]:
            eval = alpha_beta(child, depth + 1, alpha, beta, False,
tree)
            maxEval = max(maxEval, eval)
            alpha = max(alpha, eval)
            print(f"Max Node {node}: alpha={alpha}, beta={beta}")
            if beta <= alpha:
                print(f"Pruned at Node {node}")
                break
        return maxEval

    else:
        minEval = float('inf')
        for child in tree[node]:
            eval = alpha_beta(child, depth + 1, alpha, beta, True,
tree)
            minEval = min(minEval, eval)
            beta = min(beta, eval)
            print(f"Min Node {node}: alpha={alpha}, beta={beta}")
            if beta <= alpha:
                print(f"Pruned at Node {node}")
                break
        return minEval
```

```
# Tree as you defined
tree = {
    "A": ["B", "C"],
    "B": ["D", "E"],
    "C": ["F", "G"],
    "D": [3, 5],
    "E": [6, 9],
    "F": [1, 2],
    "G": [0, -1]
}

result = alpha_beta("A", 0, float('-inf'), float('inf'), True, tree)
print("\nFinal Result at Root (A):", result)
```

```
Min Node B: alpha=-inf, beta=5
Min Node B: alpha=-inf, beta=5
Max Node A: alpha=5, beta=inf
Min Node C: alpha=5, beta=2
Pruned at Node C
Max Node A: alpha=5, beta=inf

Final Result at Root (A): 5
```

This is a Tic-Tac-Toe game that supports both single player (against the computer) and two-player modes. The board is represented as a 1D list of size 9, where each position can be:

0 → Empty

1 → Player O's move

-1 → Player X's move

```
# This function is used to draw the board's current state every time
the user's turn arrives.
def ConstBoard(board):
    print("Current State Of Board : \n\n")
    for i in range(0, 9):
        if (i > 0) and (i % 3) == 0:
            print("\n")
        if board[i] == 0:
            print("- ", end=" ")
        if board[i] == 1:
            print("O ", end=" ")
        if board[i] == -1:
            print("X ", end=" ")
    print("\n\n")


# This function takes the user move as input and makes the required
changes on the board.
```

```
def User1Turn(board):
    pos = input("Enter X's position from [1...9]: ")
    pos = int(pos)
    if board[pos - 1] != 0:
        print("Wrong Move!!!")
        exit(0)
    board[pos - 1] = -1


def User2Turn(board):
    pos = input("Enter O's position from [1...9]: ")
    pos = int(pos)
    if board[pos - 1] != 0:
        print("Wrong Move!!!")
        exit(0)
    board[pos - 1] = 1


# Minimax function.
def minimax(board, player):
    x = analyzeboard(board)
    if x != 0:
        return x * player
    pos = -1
    value = -2
    for i in range(0, 9):
        if board[i] == 0:
            board[i] = player
            score = -minimax(board, -player)
            if score > value:
                value = score
                pos = i
            board[i] = 0
    if pos == -1:
        return 0
    return value


# This function makes the computer's move using the minimax algorithm.
def CompTurn(board):
    pos = -1
    value = -2
    for i in range(0, 9):
        if board[i] == 0:
            board[i] = 1
            score = -minimax(board, -1)
            board[i] = 0
            if score > value:
                value = score
                pos = i
```

```
    board[pos] = 1


# This function is used to analyze a game.
def analyzeboard(board):
    cb = [
        [0, 1, 2],
        [3, 4, 5],
        [6, 7, 8],
        [0, 3, 6],
        [1, 4, 7],
        [2, 5, 8],
        [0, 4, 8],
        [2, 4, 6],
    ]
    for i in range(0, 8):
        if (
            board[cb[i][0]] != 0
            and board[cb[i][0]] == board[cb[i][1]]
            and board[cb[i][0]] == board[cb[i][2]]
        ):
            return board[cb[i][2]]
    return 0


# Main Function.
def main():
    choice = input("Enter 1 for single player, 2 for multiplayer: ")
    choice = int(choice)
    # The board is considered in the form of a single dimensional
array.
    # One player moves 1 and other move -1.
    board = [0, 0, 0, 0, 0, 0, 0, 0, 0]
    if choice == 1:
        print("Computer : O Vs. You : X")
        player = input("Enter to play 1(st) or 2(nd) :")
        player = int(player)
        for i in range(0, 9):
            if analyzeboard(board) != 0:
                break
            if (i + player) % 2 == 0:
                CompTurn(board)
            else:
                ConstBoard(board)
                User1Turn(board)
    else:
        for i in range(0, 9):
            if analyzeboard(board) != 0:
                break
            if i % 2 == 0:
```

```
                ConstBoard(board)
                User1Turn(board)
            else:
                ConstBoard(board)
                User2Turn(board)

    x = analyzeboard(board)
    if x == 0:
        ConstBoard(board)
        print("Draw!!!")
    if x == -1:
        ConstBoard(board)
        print("X Wins!!! You Lose!!!")
    if x == 1:
        ConstBoard(board)
        print("X Lose!!! O Wins!!!")


# --------------- #
main()
# --------------- #
```

```
Enter 1 for single player, 2 for multiplayer: 1
Computer : O Vs. You : X
Enter to play 1(st) or 2(nd) :2
Current State Of Board :


O  -  -

-  -  -

-  -  -


Enter X's position from [1...9]: 6
Current State Of Board :


O  -  O

-  -  X

-  -  -


Enter X's position from [1...9]: 8
Current State Of Board :


O  O  O
```

```
-  -  X
-  X  -

X Lose!!! O Wins!!!
```