Lab 9

CLO: 01, 02

Min-Max

The Min-Max algorithm is a pivotal concept in artificial intelligence, frequently employed in two-player games to strategize and make optimal decisions. Its primary purpose is to identify the best move for a player by exhaustively examining the game tree's various branches, evaluating the consequences of each potential move, and forecasting the game's ultimate outcome.

This algorithm finds applications in a wide array of two-player games, ranging from classics like chess, tic-tac-toe, and checkers to more modern board and card games. Its versatility makes it a cornerstone of AI research, empowering machines to challenge and even surpass human players in these strategic contests. As AI continues to evolve, so does the Min-Max algorithm's significance, shaping the landscape of competitive gameplay and decision-making.

```
function minimax(position, depth, maximizingPlayer)
        if depth == 0 or game over in position
                return static evaluation of position

        if maximizingPlayer
                maxEval = -infinity
                for each child of position
                        eval = minimax(child, depth - 1, false)
                        maxEval = max(maxEval, eval)
                return maxEval

        else
                minEval = +infinity
                for each child of position
                        eval = minimax(child, depth - 1, true)
                        minEval = min(minEval, eval)
                return minEval



// initial call
minimax(currentPosition, 3, true)
```

Question 1:

Algorithm:

```python
class TreeNode:
    def __init__(self, value, children=None):
        self.value = value
        self.children = children if children is not None else []
        self.is_maximizing = True

def minimax(node, depth, is_maximizing):
    if depth == 0 or not node.children:
        return node.value

    if is_maximizing:
        max_eval = -float('inf')
        for child in node.children:
            eval = minimax(child, depth - 1, False)
            max_eval = max(max_eval, eval)
        return max_eval
    else:
        min_eval = float('inf')
        for child in node.children:
            eval = minimax(child, depth - 1, True)
            min_eval = min(min_eval, eval)
        return min_eval
```

```python
root  = TreeNode(1)
node1 = TreeNode(3)
node2 = TreeNode(-4)
node3 = TreeNode(3)
node4 = TreeNode(5)
node5 = TreeNode(-4)
node6 = TreeNode(9)
node7=TreeNode(-1)
node8 = TreeNode(3)
node9=TreeNode(5)
node10=TreeNode(1)
node11=TreeNode(-6)
node12=TreeNode(-4)
node13=TreeNode(0)
node14=TreeNode(9)

root.children = [node1, node2]
node1.children = [node3, node4]
node2.children = [node5, node6]
node3.children=[node7, node8]
node4.children=[node9, node10]
node5.children=[node11, node12]
node6.children=[node13, node14]

best_value = minimax(root, depth=4, is_maximizing=True)
```

Alpha Beta Pruning:

Alpha-Beta pruning is an optimization of the Min-Max algorithm used in AI for two-player games. It reduces the number of evaluated nodes in a game tree by maintaining two values, alpha and beta, to identify promising branches and prune those that won't affect the final decision. The key steps are:

1. Initialize alpha to negative infinity and beta to positive infinity.
2. During tree evaluation, if alpha ≥ beta for Player Max or alpha ≤ beta for Player Min, prune the branch.
3. Update alpha and beta values as you explore the tree.
4. Propagate these values up the tree.
5. Choose the best move based on the final alpha values for Player Max.

```
best_value = alpha_beta(root, depth=3, alpha=-float('inf'), beta=float('inf'), is_maximizing=True)
print("Best value:", best_value)
```

Alpha-Beta pruning is crucial for optimizing the Min-Max algorithm in games with vast search spaces, such as chess, by eliminating unfruitful branches.

Question 2: