Week 10

Alpha Beta Pruning

Code:

```
# Alpha-Beta Pruning Implementation
def alpha_beta_pruning(node, alpha, beta, maximizing_player):
    # Base case: If it's a leaf node, return its value (simulating
evaluation of the node)
    if type(node) is int:
        return node

    # If not a leaf node, explore the children
    if maximizing_player:
        max_eval = -float('inf')
        for child in node:  # Iterate over children of the maximizer node
            eval = alpha_beta_pruning(child, alpha, beta, False)
            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)  # Maximize alpha
            if beta <= alpha:  # Prune the branch
                break
        return max_eval
    else:
        min_eval = float('inf')
        for child in node:  # Iterate over children of the minimizer node
            eval = alpha_beta_pruning(child, alpha, beta, True)
            min_eval = min(min_eval, eval)
            beta = min(beta, eval)  # Minimize beta
            if beta <= alpha:  # Prune the branch
                break
        return min_eval

# Function to build the tree from a list of numbers
def build_tree(numbers):
    # We need to build a tree with alternating levels of maximizers and
minimizers
    # Start from the leaf nodes and work up
    current_level = [[n] for n in numbers]

    while len(current_level) > 1:
```

```
        next_level = []
        for i in range(0, len(current_level), 2):
            if i + 1 < len(current_level):
                next_level.append(current_level[i] + current_level[i + 1])
# Combine two nodes
            else:
                next_level.append(current_level[i])  # Odd number of
elements, just carry forward
        current_level = next_level

    return current_level[0]  # Return the root node, which is a maximizer

# Main function to run alpha-beta pruning
def main():
    # Input: User provides a list of numbers
    numbers = list(map(int, input("Enter numbers for the game tree
(space-separated): ").split()))

    # Build the tree with the given numbers
    tree = build_tree(numbers)

    # Parameters: Tree, initial alpha, beta, and the root node is a
maximizing player
    alpha = -float('inf')
    beta = float('inf')
    maximizing_player = True  # The root node is a maximizing player

    # Perform alpha-beta pruning and get the final result
    result = alpha_beta_pruning(tree, alpha, beta, maximizing_player)

    print("Final Result of Alpha-Beta Pruning:", result)

if __name__ == "__main__":
    main()
```

Output:

```
Enter numbers for the game tree (space-separated): 10 9 14 18 5 4 50 3
Final Result of Alpha-Beta Pruning: 50
```

17/12/24 WEEK-10 :

ADVERSARIAL SEARCH - Implement Alpha-Beta pruning

Algorithm:

```
function ALPHA-BETA-SEARCH(state) returns an action
    v ← MAX-VALUE(state, -∞, ∞)
    return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state
    v ← -∞
    for each a in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v

function MIN-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← +∞
    for each a in ACTIONS(state) do
        v ← MIN(v, MAX-VALUE(RESULT(s,a), α, β))
        if v ≤ α then return v
        β ← MIN(β, v)
    return v
```

classmate
Date
Page

Problem:

Apply the Alpha - Beta search algorithm to find value of root ~~node~~ to path to root node (MAX node). Identify the paths which are pruned (or cutoff) for exploration