**LAB 08**

**Open Ended Lab-1**

**GAME SEARCH**

**Objective:**

One of the biggest areas of research in modern Artificial Intelligence is in making computer players

for popular games. Make your own four depth binary tree. Solve the decision tree by applying gaming

theory of your own choice (either apply Minimax or Alpha beta pruning etc. orany other technique),

make a choice of your own values ranging from 0 to 50:

• The game will be two player game, one will be user and other will be AI agent

• Name and design of levels must be distinct for each student‘s game binary tree.

**Theory**

**Introduction to Game Theory**

Game theory is a mathematical framework for analyzing competitive situations where the outcomes depend on the actions of multiple agents (players). In two-player games, each player aims to maximize their own payoff while minimizing the opponent's payoff.

**Alpha-Beta Pruning**

Alpha-Beta pruning is an optimization technique for the minimax algorithm, which is used in decision-making and game theory. It reduces the number of nodes evaluated in the search tree by eliminating branches that cannot possibly influence the final decision.

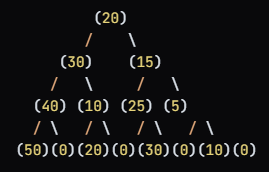
* **Alpha**: The best value that the maximizer (AI) can guarantee at that level or above.
* **Beta**: The best value that the minimizer (User ) can guarantee at that level or above.

**Game Design**

* **Game Name**: Treasure Hunt
* **Players**:
  + User: Tries to maximize their score.
  + AI: Tries to minimize the User's score while maximizing its own.
* **Binary Tree Structure**: Each node represents a treasure value ranging from 0 to 50. The internal nodes represent decisions made by the players.

## Diagram of the Binary Tree

Here is a visual representation of the binary tree used in the game:

****

**Source Code:**

class Node:

    def \_\_init\_\_(self, value):

        self.value = value

        self.left = None

        self.right = None

def alpha\_beta(node, depth, alpha, beta, maximizing\_player):

    # Base case: if we reach a leaf node or the maximum depth

    if depth == 0 or (node.left is None and node.right is None):

        return node.value

    if maximizing\_player:

        max\_eval = float('-inf')

        if node.left:

            max\_eval = max(max\_eval, alpha\_beta(node.left, depth - 1, alpha, beta, False))

        if node.right:

            max\_eval = max(max\_eval, alpha\_beta(node.right, depth - 1, alpha, beta, False))

        alpha = max(alpha, max\_eval)

        if beta <= alpha:

            return max\_eval

        return max\_eval

    else:

        min\_eval = float('inf')

        if node.left:

            min\_eval = min(min\_eval, alpha\_beta(node.left, depth - 1, alpha, beta, True))

        if node.right:

            min\_eval = min(min\_eval, alpha\_beta(node.right, depth - 1, alpha, beta, True))

        beta = min(beta, min\_eval)

        if beta <= alpha:

            return min\_eval

        return min\_eval

# Constructing the binary tree

root = Node(20)

root.left = Node(30)

root.right = Node(15)

root.left.left = Node(40)

root.left.right = Node(10)

root.right.left = Node(25)

root.right.right = Node(5)

root.left.left.left = Node(50)

root.left.left.right = Node(0)

root.left.right.left = Node(20)

root.left.right.right = Node(0)

root.right.left.left = Node(30)

root.right.left.right = Node(0)

root.right.right.left = Node(10)

root.right.right.right = Node(0)

# Running the Alpha-Beta pruning algorithm

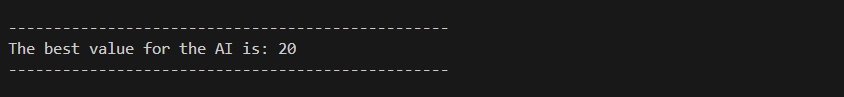
best\_value = alpha\_beta(root, 4, float('-inf'), float('inf'), True)

print("\n-------------------------------------------------")

print("The best value for the AI is:", best\_value)

print("-------------------------------------------------\n")

**Output:**

****

## Conclusion

The "Treasure Hunt" game demonstrates the application of game theory and the Alpha-Beta pruning technique in a two-player