# Experiment No.-05

**Objective:** Implement MINI MAX Algorithm for game playing(ALPHA-BETA PRUNING)**.**

**Theory:-** The MiniMax algorithm with Alpha-Beta Pruning is an optimization of the basic MiniMax algorithm, used in game theory and artificial intelligence to determine the optimal move for a player in a two-player, zero-sum game.
Conceptual Overview:

- **Minimax Principle:** The algorithm explores the game tree, assuming both players play optimally. The maximizing player (Max) aims to maximize their score, while the minimizing player (Min) aims to minimize Max's score (which is equivalent to maximizing their own score in a zero-sum game).
- **Alpha-Beta Pruning:** This optimization prunes branches of the game tree that are guaranteed not to affect the final decision. It maintains two values:
- **Alpha (α):** The best (highest) value that the maximizing player can guarantee so far.
- **Beta (β):** The best (lowest) value that the minimizing player can guarantee so far.
- If at any point beta <= alpha, it means that a better move has already been found for the current player's opponent in a different branch, making the current branch irrelevant. The search can be terminated for that branch.

## Program:-

```
import math

def minimax_alpha_beta(node_index, depth, is_maximizing_player, scores, alpha, beta):

   # Base case: if at a leaf node (terminal state)     if
depth == 0:  # Assuming depth 0 means a terminal node
return scores[node_index]

   if is_maximizing_player:
max_eval = -math.inf
       # Simulate moves (assuming 2 children per node for simplicity)
for i in range(2):
           child_index = 2 * node_index + i + 1  # Calculate child index            eval
= minimax_alpha_beta(child_index, depth - 1, False, scores, alpha, beta)
max_eval = max(max_eval, eval)            alpha = max(alpha, eval)            if beta
<= alpha:
```

```
        break  # Alpha-Beta
Pruning         return max_eval
else:
      min_eval =
math.inf        #
Simulate moves
for i in range(2):
          child_index = 2 * node_index + i + 1            eval =
minimax_alpha_beta(child_index, depth - 1, True, scores, alpha, beta)
min_eval = min(min_eval, eval)            beta = min(beta, eval)            if beta <=
alpha:                break  # Alpha-Beta Pruning         return min_eval

# Example scores for leaf nodes (indices 3, 4, 5, 6 in a conceptual tree)
leaf_scores = [3, 5, 2, 9] # These are the scores at the deepest level of the tree.

full_scores_list = [0] * 7 # Placeholder for non-leaf
nodes full_scores_list[3] = 3 full_scores_list[4] = 5
full_scores_list[5] = 2 full_scores_list[6] = 9

optimal_value = minimax_alpha_beta(0, 2, True, full_scores_list, -math.inf, math.inf)
print(f"The optimal value is: {optimal_value}")
```

## Output:-

```
The optimal value is: 3
```