	Marathwada Mitra Mandal's	
	Institute of Technology, Lohgaon Pune - 47	
	Department of Artificial Intelligence and Data Science	

Semester -I A.Y.2025-26 Sub.: - Artificial Intelligence Lab Class: SE

Experiment Title:

Implementation of Alpha-Beta Pruning Algorithm for Game Tree Search

Objectives:

1. To understand the concept of game playing algorithms in Artificial Intelligence.
2. To implement alpha-beta pruning as an optimization to the Minimax algorithm.
3. To analyze how pruning reduces the number of nodes evaluated in a search tree.

Theory:

- Game Playing in AI:
  Game playing problems (like Chess, Tic-Tac-Toe) involve two players: MAX (tries to maximize the score) and MIN (tries to minimize the score).

- Minimax Algorithm:
  A recursive algorithm that explores all possible moves in a game tree to determine the best possible strategy.

- Problem with Minimax:
  It evaluates all nodes → very time consuming for large trees.

- Alpha-Beta Pruning:
  An enhancement of Minimax that prunes (cuts off) branches in the search tree that don't affect the final decision, reducing computation.

- Key Terms:

  - α (alpha): Best value that MAX can guarantee so far.

  - β (beta): Best value that MIN can guarantee so far.

  - If β ≤ α, the branch is pruned (stopped).

Algorithm (Pseudocode):

```
function alpha_beta(node, depth, α, β, maximizingPlayer):
 if depth = 0 or node is terminal:
 return heuristic_value(node)

 if maximizingPlayer:
 value = -∞
 for each child of node:
 value = max(value, alpha_beta(child, depth-1, α, β, False))
 α = max(α, value)
 if β ≤ α:
 break # β cut-off
 return value
 else:
 value = +∞
```

```
for each child of node:
value = min(value, alpha_beta(child, depth-1, α, β, True))
β = min(β, value)
if β ≤ α:
break # α cut-off
return value
```

Sample Output:

Optimal value (with Alpha-Beta Pruning): 5

Observation Table:

Parameter	Minimax	Alpha-Beta
Nodes evaluated	8	5
Optimal value	5	5

Conclusion:

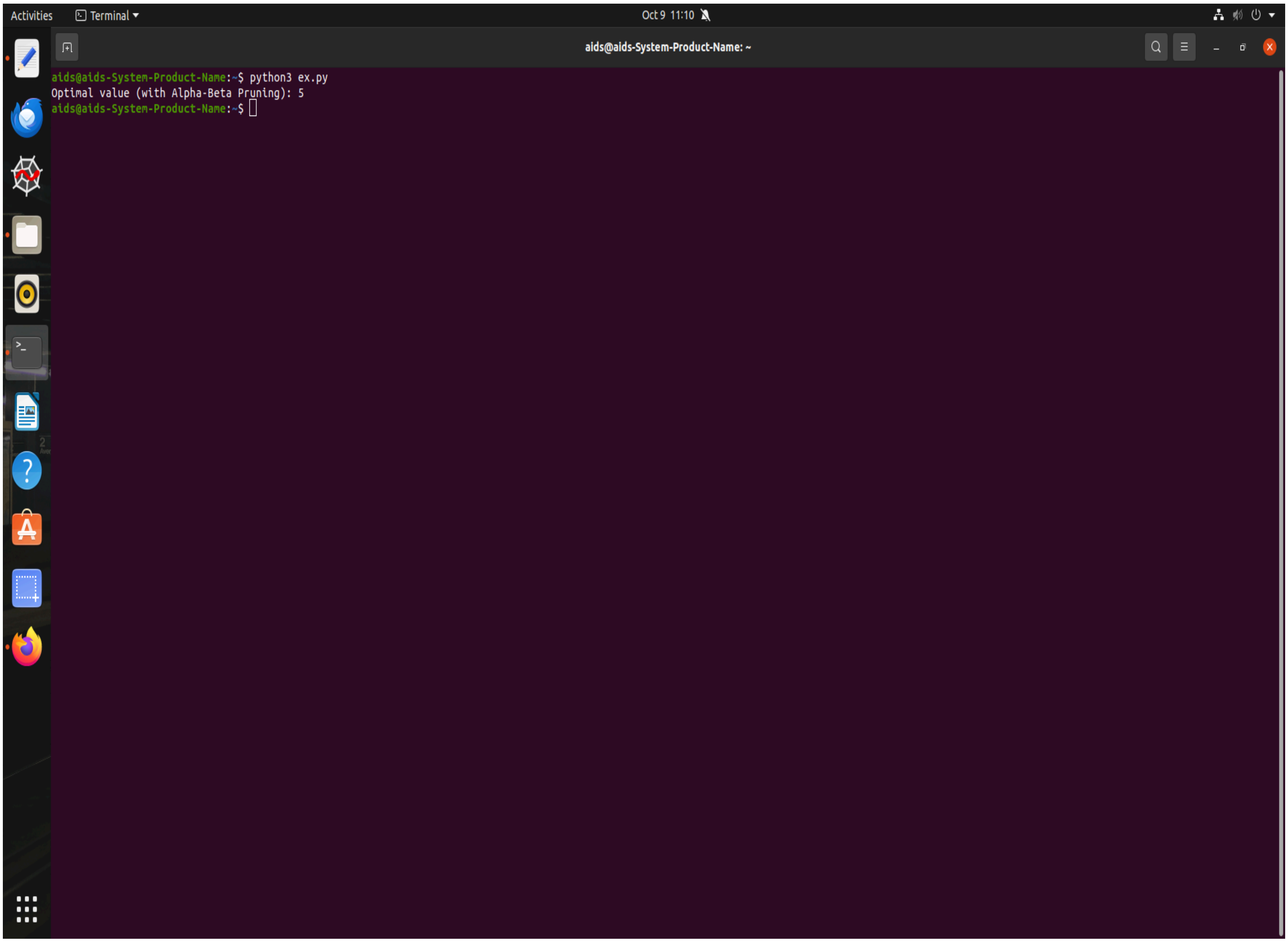
The Alpha-Beta Pruning algorithm significantly reduces the number of nodes evaluated compared to Minimax, while still giving the same optimal decision.

```
def alphabeta(node, depth, alpha, beta, maximizingPlayer, values,
index=0):
    # Leaf node condition
    if depth == 0 or index >= len(values):
    return values[index]

    if maximizingPlayer:
    best = float('-inf')
    for i in range(2):  # Two children for each node
    val = alphabeta(node*2+i, depth-1, alpha, beta, False, values,
index*2+i)
    best = max(best, val)
    alpha = max(alpha, best)
    if beta <= alpha:
        break  # Beta cut-off
    return best
    else:
    best = float('inf')
    for i in range(2):
    val = alphabeta(node*2+i, depth-1, alpha, beta, True, values,
index*2+i)
    best = min(best, val)
    beta = min(beta, best)
    if beta <= alpha:
        break  # Alpha cut-off
    return best


# Example: Game tree with depth = 3
values = [3, 5, 6, 9, 1, 2, 0, -1]  # Leaf node values
depth = 3
result = alphabeta(0, depth, float('-inf'), float('inf'), True, values)

print("Optimal value (with Alpha-Beta Pruning):", result)
```


Activities
Terminal
Oct 9 11:10
aids@aids-System-Product-Name: ~
aids@aids-System-Product-Name:~$ python3 ex.py
Optimal value (with Alpha-Beta Pruning): 5
aids@aids-System-Product-Name:~$