



ARTIFICIAL INTELLIGENCE LAB

BSCYS-3rd Semester

Fall 2025

LAB REPORT # 8

Submitted To:**Sir Mubashir Iqbal**

Submitted by:**M.Umer**

Reg No: **(24-cys-024)**

Department: **Cyber Security (A)**

BS CYBER SECURITY PROGRAM

DEPARTMENT OF COMPUTER SCIENCE HITEC UNIVERSITY TAXILA

TASK 1:

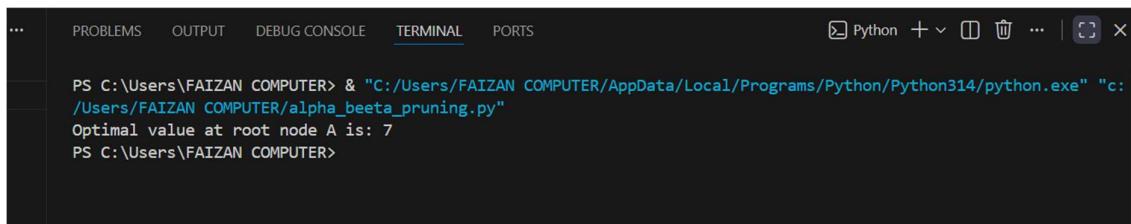
ScrenShot:

The screenshot shows a code editor window with two tabs: "alpha_beeta_pruning.py" and "tic_tac_toe_minimax.py". The current file is "alpha_beeta_pruning.py". The code implements the Alpha-Beta Pruning algorithm for a game tree with a maximum depth of 3. It uses a recursive function "alpha_beta" to calculate the best value for a player's turn (MAX) or the opponent's turn (MIN). The MAX player's turn starts at depth 0, while the MIN player's turn starts at depth 1. The code includes logic for terminal conditions (leaf node reached at depth 3), pruning based on the Alpha-Beta condition (beta <= alpha), and updating the best value and alpha/beta bounds.

```
1 import math
2
3 def alpha_beta(depth, node_index, is_max, values, alpha, beta):
4     # Terminal condition (leaf node reached)
5     if depth == 3:
6         return values[node_index]
7
8     # MAX player's turn
9     if is_max:
10        best_value = -math.inf
11
12        for i in range(2):
13            value = alpha_beta(
14                depth + 1,
15                node_index * 2 + i,
16                False,
17                values,
18                alpha,
19                beta
20            )
21            best_value = max(best_value, value)
22        alpha = max(alpha, best_value)
23
24
25    # Alpha-Beta Pruning condition
26    if beta <= alpha:
27        break
28
29    return best_value
30
31    # MIN player's turn
32 else:
33    best_value = math.inf
34
35    for i in range(2):
36        value = alpha_beta(
37            depth + 1,
38            node_index * 2 + i,
39            True,
40            values,
41            alpha,
42            beta
43        )
44        best_value = min(best_value, value)
45        beta = min(beta, best_value)
46
```

```
47     # Alpha-Beta Pruning condition
48     if beta <= alpha:
49         break
50
51     return best_value
52
53
54 # Terminal node values (left to right)
55 terminal_values = [4, 2, -3, -6, 7, 0, 5, 8]
56
57 # Applying Alpha-Beta Pruning from root node A
58 optimal_value = alpha_beta(
59     depth=0,
60     node_index=0,
61     is_max=True,
62     values=terminal_values,
63     alpha=-math.inf,
64     beta=math.inf
65 )
66
67 print("Optimal value at root node A is:", optimal_value)
68
```

Output:



The screenshot shows a terminal window in VS Code. The tab bar at the top includes 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is underlined), and 'PORTS'. To the right of the tabs are icons for Python, a plus sign, a dropdown arrow, a trash can, three dots, and a close button. The terminal itself displays the following text:

```
PS C:\Users\FAIZAN COMPUTER> & "C:/Users/FAIZAN COMPUTER/AppData/Local/Programs/Python/Python314/python.exe" "c:/Users/FAIZAN COMPUTER/alpha_beeta_pruning.py"
Optimal value at root node A is: 7
PS C:\Users\FAIZAN COMPUTER>
```

Task 2:

ScreenShot:

```
C: > Users > FAIZAN COMPUTER > tic_tac_toe_minimax.py > ...
1  PLAYER_X = 'X' # Maximizing player
2  PLAYER_O = 'O' # Minimizing player
3  EMPTY = '.'
4
5
6  board = [
7      ['O', 'X', 'O'],
8      ['O', 'X', 'X'],
9      ['X', 'O', 'X']
10 ]
11
12 def check_winner(board):
13     # Rows, columns and diagonals
14     lines = board + list(zip(*board)) + [
15         [board[i][i] for i in range(3)],
16         [board[i][2 - i] for i in range(3)]
17     ]
18
19     for line in lines:
20         if line.count(PLAYER_X) == 3:
21             return 1
22         if line.count(PLAYER_O) == 3:
23             return -1
24
25     return 0
26
27 def is_full(board):
28     return all(cell != EMPTY for row in board for cell in row)
29
30 def minimax(board, is_max):
31     score = check_winner(board)
32
33     if score != 0:
34         return score
35     if is_full(board):
36         return 0 # Draw
37
38     if is_max:
39         best = -1000
40         for i in range(3):
41             for j in range(3):
42                 if board[i][j] == EMPTY:
43                     board[i][j] = PLAYER_X
44                     best = max(best, minimax(board, False))
45                     board[i][j] = EMPTY
46             return best
47     else:
```

```
46
47     else:
48         best = 1000
49         for i in range(3):
50             for j in range(3):
51                 if board[i][j] == EMPTY:
52                     board[i][j] = PLAYER_O
53                     best = min(best, minimax(board, True))
54                     board[i][j] = EMPTY
55
56
57     return best
58
59 result = minimax(board, True)
60
61 if result == 1:
62     print("X will win")
63 elif result == -1:
64     print("O will win")
65 else:
66     print("Game will be a Draw")
```

Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + ×
```

PS C:\Users\FAIZAN COMPUTER> & "C:/Users/FAIZAN COMPUTER/AppData/Local/Programs/Python/Python314/python.exe" "c:/Users/FAIZAN COMPUTER/tic_tac_toe_minimax.py"
Game will be a Draw
PS C:\Users\FAIZAN COMPUTER>

Conclusion:

In this lab, I learned how to apply Alpha-Beta Pruning to a minimax tree and determined the optimal value at the root node A as 7, with pruning reducing unnecessary evaluations and improving efficiency. I also applied the Minimax algorithm to analyze a nearly full Tic-Tac-Toe board, where no winning moves were available, resulting in a draw with a Minimax value of 0. Overall, the lab explained how decision-making algorithms like Minimax and Alpha-Beta Pruning can be used to evaluate game trees and strategic scenarios efficiently, highlighting both optimal decision selection and computational optimization.