



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)**

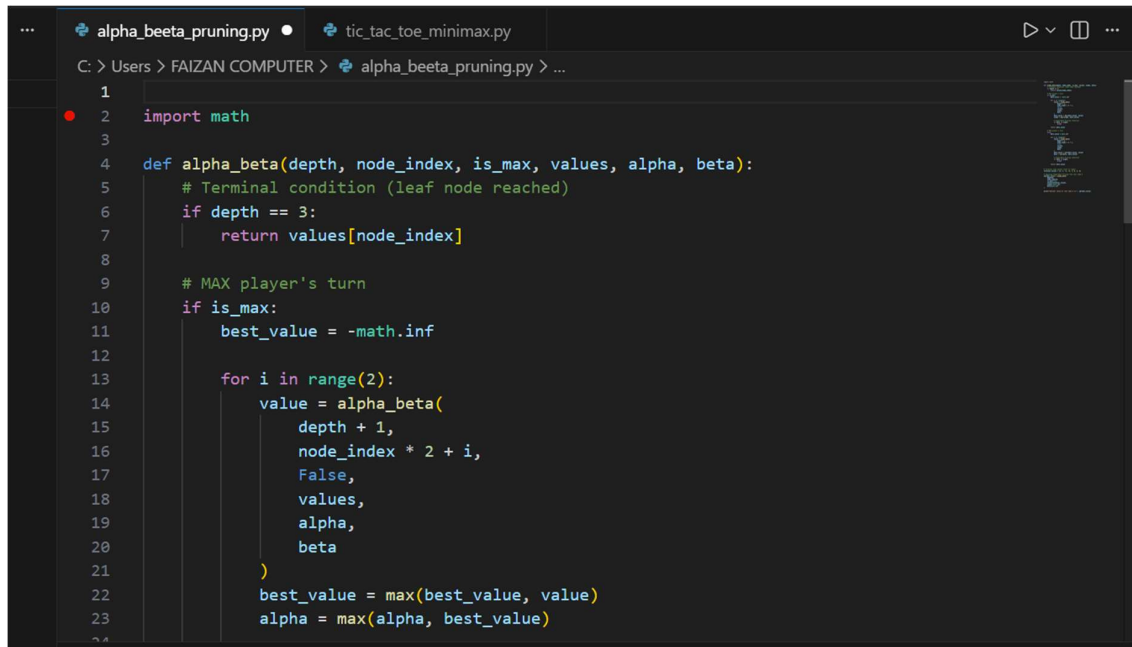
Department: **Cyber Security (A)**

BS CYBER SECURITY PROGRAM

DEPARTMENT OF COMPUTER SCIENCE HITEC UNIVERSITY TAXILA

TASK 1:

ScreenShot:



The screenshot shows a code editor with two tabs: 'alpha_beta_pruning.py' (active) and 'tic_tac_toe_minimax.py'. The active tab displays the following Python code:

```
1
2 import math
3
4 def alpha_beta(depth, node_index, is_max, values, alpha, beta):
5     # Terminal condition (leaf node reached)
6     if depth == 3:
7         return values[node_index]
8
9     # MAX player's turn
10    if is_max:
11        best_value = -math.inf
12
13        for i in range(2):
14            value = alpha_beta(
15                depth + 1,
16                node_index * 2 + i,
17                False,
18                values,
19                alpha,
20                beta
21            )
22            best_value = max(best_value, value)
23            alpha = max(alpha, best_value)
```



The screenshot shows the continuation of the Python code from the previous block, specifically the MIN player's turn and the pruning condition:

```
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
```


ScreenShot:



```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
alpha_beeta_pruning.py • tic_tac_toe_minimax.py •
C: > Users > FAIZAN COMPUTER > tic_tac_toe_minimax.py > ...

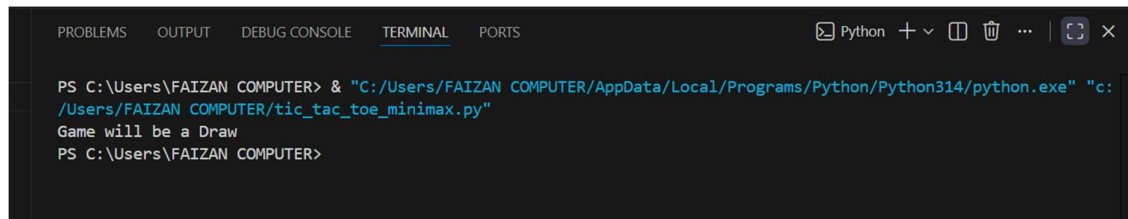
1
2  PLAYER_X = 'X' # Maximizing player
3  PLAYER_O = 'O' # Minimizing player
4  EMPTY = ' '
5
6  board = [
7      ['O', 'X', 'O'],
8      ['O', 'X', 'X'],
9      ['X', 'O', 'X']
10 ]
11
12 def check_winner(board):
13     # Rows, columns and diagonals
14     lines = board + list(zip(*board)) + [
15         [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     else:
```

```

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

```

Output:



```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
Python + - [] [X] ... [ ] [X]
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 learn how to applied 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 explain 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