14. Write the python program to implement Apha & Beta pruning algorithm for gaming.

**AIM :** Apha & Beta pruning algorithm for gaming.

**ALGORITHM :**

1. The ‘print\_board’ function prints the current state of the Tic Tac Toe board.
2. The ‘evaluate’ function checks if the game is won by a player or if it's a draw.
3. The ‘is\_moves\_left’ function checks if there are any empty cells left on the board.
4. The ‘minimax\_alpha\_beta’ function implements the Minimax algorithm with Alpha-Beta Pruning.
5. The ‘ find\_best\_move\_alpha\_beta’ function finds the best move for the AI player using Alpha-Beta Pruning.
6. The ‘play\_tic\_tac\_toe\_alpha\_beta’ function contains the main game loop and player input handling.
7. The loop continues until there is a winner, a draw, or no more empty cells.

**PROGRAM :**

def print\_board(board):

for row in board:

print(" | ".join(row))

print("-" \* 9)

def evaluate(board):

for row in board:

if all(cell == "X" for cell in row):

return 10

elif all(cell == "O" for cell in row):

return -10

for col in range(3):

if all(board[row][col] == "X" for row in range(3)):

return 10

elif all(board[row][col] == "O" for row in range(3)):

return -10

if all(board[i][i] == "X" for i in range(3)) or all(board[i][2 - i] == "X" for i in range(3)):

return 10

elif all(board[i][i] == "O" for i in range(3)) or all(board[i][2 - i] == "O" for i in range(3)):

return -10

return 0

def is\_moves\_left(board):

return any(cell == " " for row in board for cell in row)

def minimax\_alpha\_beta(board, depth, is\_maximizing, alpha, beta):

score = evaluate(board)

if score == 10:

return score - depth

if score == -10:

return score + depth

if not is\_moves\_left(board):

return 0

if is\_maximizing:

best\_score = float("-inf")

for i in range(3):

for j in range(3):

if board[i][j] == " ":

board[i][j] = "X"

best\_score = max(best\_score, minimax\_alpha\_beta(board, depth + 1, False, alpha, beta))

board[i][j] = " "

alpha = max(alpha, best\_score)

if beta <= alpha:

break

return best\_score

else:

best\_score = float("inf")

for i in range(3):

for j in range(3):

if board[i][j] == " ":

board[i][j] = "O"

best\_score = min(best\_score, minimax\_alpha\_beta(board, depth + 1, True, alpha, beta))

board[i][j] = " "

beta = min(beta, best\_score)

if beta <= alpha:

break

return best\_score

def find\_best\_move\_alpha\_beta(board):

best\_move = None

best\_value = float("-inf")

alpha = float("-inf")

beta = float("inf")

for i in range(3):

for j in range(3):

if board[i][j] == " ":

board[i][j] = "X"

move\_value = minimax\_alpha\_beta(board, 0, False, alpha, beta)

board[i][j] = " "

if move\_value > best\_value:

best\_value = move\_value

best\_move = (i, j)

alpha = max(alpha, best\_value)

return best\_move

def play\_tic\_tac\_toe\_alpha\_beta():

board = [[" " for \_ in range(3)] for \_ in range(3)]

current\_player = "X"

while True:

print\_board(board)

if current\_player == "X":

row, col = map(int, input("Enter your move (row col): ").split())

if board[row][col] != " ":

print("Invalid move. Try again.")

continue

board[row][col] = "X"

else:

print("AI's move:")

best\_move = find\_best\_move\_alpha\_beta(board)

board[best\_move[0]][best\_move[1]] = "X"

if evaluate(board) == 10:

print\_board(board)

print("Player X wins!")

break

elif evaluate(board) == -10:

print\_board(board)

print("Player O wins!")

break

elif not is\_moves\_left(board):

print\_board(board)

print("It's a draw!")

break

current\_player = "O" if current\_player == "X" else "X"

play\_tic\_tac\_toe\_alpha\_beta()

**OUT PUT :**

| |

X | |

| |

-----------

| |

| O |

| |

-----------

| |

| | X

Enter your move (row col): 1 0

| |

X | |

| |

-----------

X | |

| O |

| |

-----------

| |

| | X

AI's move:

| |

X | |

| |

-----------

X | |

| O |

| |

-----------

| | O

Enter your move (row col): 0 2

| | X

X | |

| |

-----------

X | |

| O |

| |

-----------

| | O

AI's move:

| | X

X | |

| |

-----------

X | |

| O |

| |

-----------

O | | O

Enter your move (row col): 2 0

| | X

X | |

| |

-----------

X | |

| O |

| |

-----------

O | | O

AI's move:

| | X

X | |

| |

-----------

X | |

| O |

| |

-----------

O | X | O

Enter your move (row col): 1 2

| | X

X | |

| |

-----------

X | |

| O |

| |

-----------

O | X | O

AI's move:

| | X

X | |

| |

-----------

X | O |

| O |

| |

-----------

O | X | O

Enter your move (row col): 2 2

| | X

X | |

| |

-----------

X | O |

| O |

| |

-----------

O | X | O

It's a draw!