<u>LAB-10</u> - Tic Tac Toe using Min-Max.

<u>Code:</u>

```
import math

# Constants for players
HUMAN = 'O'  # Minimizer
AI = 'X'     # Maximizer

# Initialize empty board
def create_board():
    return [[' ' for _ in range(3)] for _ in range(3)]

# Check if there are any moves left on the board
def is_moves_left(board):
    for row in board:
        if ' ' in row:
            return True
    return False

# Check for a win condition
def evaluate(board):
    # Rows, columns, diagonals check
    for row in board:
        if row[0] == row[1] == row[2] and row[0] != ' ':
            return 1 if row[0] == AI else -1

    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] and board[0][col] != ' ':
            return 1 if board[0][col] == AI else -1

    if board[0][0] == board[1][1] == board[2][2] and board[0][0] != ' ':
        return 1 if board[0][0] == AI else -1

    if board[0][2] == board[1][1] == board[2][0] and board[0][2] != ' ':
        return 1 if board[0][2] == AI else -1

    return 0  # No winner
```

```
# Minimax algorithm with Alpha-Beta Pruning
def minimax(board, depth, is_maximizing, alpha, beta):
    score = evaluate(board)

    # Terminal condition
    if score == 1:  # AI wins
        return score - depth  # Prefer quicker wins
    if score == -1:  # Human wins
        return score + depth  # Prefer slower losses
    if not is_moves_left(board):  # Draw
        return 0

    if is_maximizing:
        best = -math.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == ' ':
                    board[i][j] = AI
                    best = max(best, minimax(board, depth + 1, False, alpha, beta))
                    board[i][j] = ' '
                    alpha = max(alpha, best)
                    if beta <= alpha:
                        break
        return best
    else:
        best = math.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == ' ':
                    board[i][j] = HUMAN
                    best = min(best, minimax(board, depth + 1, True, alpha, beta))
                    board[i][j] = ' '
                    beta = min(beta, best)
                    if beta <= alpha:
                        break
        return best

# Find the best move for the AI
def find_best_move(board):
```

```
    best_val = -math.inf
    best_move = (-1, -1)
    for i in range(3):
        for j in range(3):
            if board[i][j] == ' ':
                board[i][j] = AI
                move_val = minimax(board, 0, False, -math.inf, math.inf)
                board[i][j] = ' '
                if move_val > best_val:
                    best_val = move_val
                    best_move = (i, j)
    return best_move

# Print the board
def print_board(board):
    for row in board:
        print('|'.join(row))
        print('-' * 5)

# Example usage
if __name__ == '__main__':
    board = create_board()
    while is_moves_left(board):
        print_board(board)
        # Human makes a move
        row, col = map(int, input("Enter row and column (0, 1, 2): ").split())
        if board[row][col] == ' ':
            board[row][col] = HUMAN
        else:
            print("Invalid move! Try again.")
            continue

        if evaluate(board) != 0 or not is_moves_left(board):
            break

        # AI makes a move
        print("AI is making a move...")
        ai_move = find_best_move(board)
        board[ai_move[0]][ai_move[1]] = AI
```

```
    if evaluate(board) != 0 or not is_moves_left(board):
        break

# Final result
print_board(board)
result = evaluate(board)
if result == 1:
    print("AI wins!")
elif result == -1:
    print("Human wins!")
else:
    print("It's a draw!")
```

Output:

```
 | | 
-----
 | | 
-----
 | | 
-----
Enter row and column (0, 1, 2): 0 0
AI is making a move...
O|X| 
-----
 | | 
-----
 | | 
-----
Enter row and column (0, 1, 2): 1 2
AI is making a move...
O|X|X
-----
 | |O
-----
 | | 
-----
Enter row and column (0, 1, 2): 1 0
AI is making a move...
O|X|X
-----
O| |O
-----
X| | 
-----
Enter row and column (0, 1, 2): 1 1
O|X|X
-----
O|O|O
-----
X| | 
-----
Human wins!
```

Code: Alpha-Beta pruning

```
def is_safe(board, row, col):
    """
    Check if it's safe to place a queen at board[row][col].
    """
    # Check for queen in the same column
    for i in range(row):
        if board[i][col] == 1:
            return False

    # Check for queen in the left diagonal
    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    # Check for queen in the right diagonal
    for i, j in zip(range(row, -1, -1), range(col, len(board))):
        if board[i][j] == 1:
            return False

    return True


def solve_with_alpha_beta(board, row, alpha, beta):
    """
    Solve the 8-Queens problem using Alpha-Beta Pruning.
    """
    if row >= len(board):  # All queens placed successfully
        return True

    for col in range(len(board)):
        if is_safe(board, row, col):
            # Place the queen
            board[row][col] = 1

            # Recursive call to place the next queen
            if solve_with_alpha_beta(board, row + 1, alpha, beta):
                return True
```

```
            # Backtrack if placing the queen here leads to failure
            board[row][col] = 0

        # Update alpha and beta for pruning (though not strictly necessary for 8-Queens)
        alpha = max(alpha, col)
        if beta <= alpha:
            break  # Prune

    return False


def solve_8_queens():
    """
    Solves the 8-Queens problem and prints the solution.
    """
    n = 8
    board = [[0 for _ in range(n)] for _ in range(n)]

    # Start solving with Alpha-Beta Pruning
    if solve_with_alpha_beta(board, 0, -float('inf'), float('inf')):
        print("Solution:")
        for row in board:
            print(' '.join('Q' if cell == 1 else '.' for cell in row))
    else:
        print("No solution found.")


# Execute the solver
if __name__ == "__main__":
    solve_8_queens()
```

Output:

```
Solution:
Q . . . . . . .
. . . . Q . . .
. . . . . . . Q
. . . . . Q . .
. . Q . . . . .
. . . . . . Q .
. Q . . . . . .
. . . Q . . . .
```