NATIONAL UNIVERSITY OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

ARTIFICIAL INTELLIGENCE LAB

NAME	Ayesha Imran
Class	CS-A
Oel	01
Course	Artificial Intelligence
Date	5-November-25
Submitted To	Lec. Ijlal Haider

IN LAB TASKS

TASK 01:

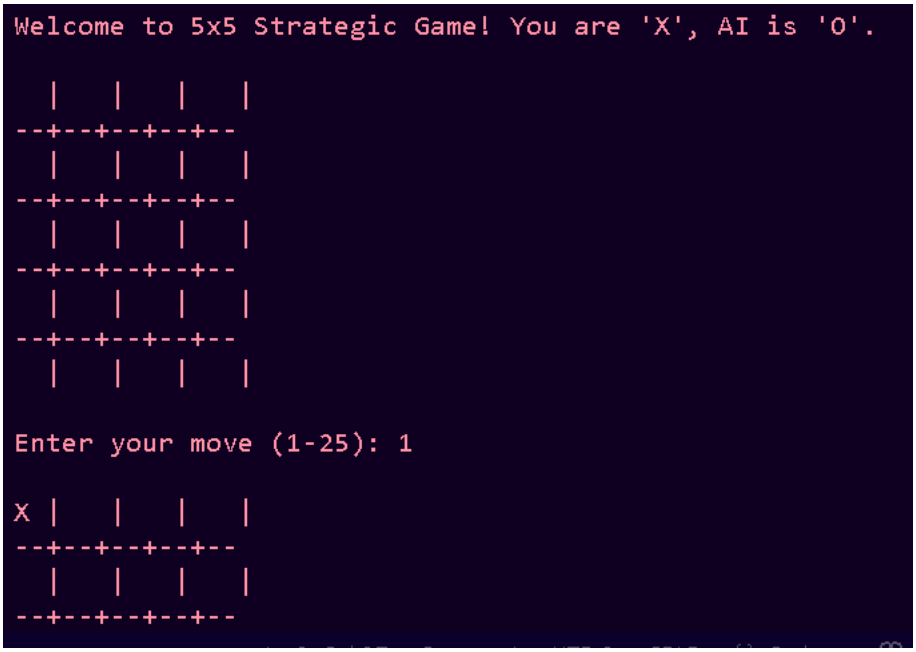
```
import math

# --- Constants ---
SIZE = 5
EMPTY = " "
PLAYER = "X"
AI = "O"

# --- Create 5x5 Board ---
board = [EMPTY for _ in range(SIZE * SIZE)]

# --- Print Board Function ---
def print_board(b):
    print()
    for i in range(SIZE):
        row = " | ".join(b[i * SIZE:(i + 1) * SIZE])
        print(row)
        if i < SIZE - 1:
            print("--+" * (SIZE - 1) + "--")
    print()

# --- Helper to Get Opponent ---
def get_opponent(player):
    return AI if player == PLAYER else PLAYER

# --- Generate All Rows, Columns, Diagonals ---
def get_lines():
    lines = []
    # Rows
    for r in range(SIZE):
        lines.append([r * SIZE + c for c in range(SIZE)])
    # Columns
    for c in range(SIZE):
        lines.append([r * SIZE + c for r in range(SIZE)])
    # Diagonals
    lines.append([i * SIZE + i for i in range(SIZE)])                # Main
diagonal
    lines.append([(i + 1) * (SIZE - 1) for i in range(SIZE)])        # Anti-
diagonal
    return lines
```

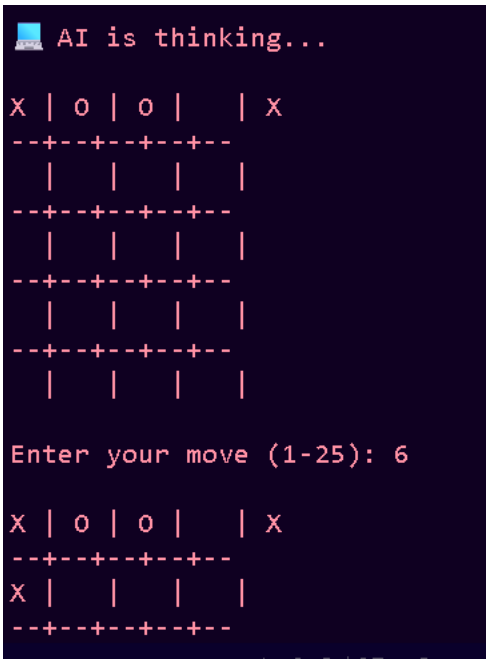
```python
LINES = get_lines()

# --- Scoring Function for a Single Player ---
def score_line(b, line, player):
    opp = get_opponent(player)
    p_count = sum(1 for i in line if b[i] == player)
    o_count = sum(1 for i in line if b[i] == opp)
    if o_count > 0:
        return 0
    if p_count == 3:
        return 5
    elif p_count == 4:
        return 10
    elif p_count == 5:
        return 20
    return 0

# --- Evaluate Board ---
def evaluate(b, player):
    opp = get_opponent(player)
    player_score = sum(score_line(b, line, player) for line in LINES)
    opp_score = sum(score_line(b, line, opp) for line in LINES)
    return player_score - opp_score

# --- Check Game End ---
def is_full(b):
    return EMPTY not in b

# --- Blocking Bonus (+3 when blocking opponent's potential 4) ---
def count_potential_fours(b, player):
    opp = get_opponent(player)
    count = 0
    for line in LINES:
        opp_count = sum(1 for i in line if b[i] == opp)
        empty_count = sum(1 for i in line if b[i] == EMPTY)
        p_count = sum(1 for i in line if b[i] == player)
        if p_count == 0 and opp_count == 3 and empty_count == 2:
            count += 1
    return count

# --- Get Available Moves ---
def available_moves(b):
    return [i for i, val in enumerate(b) if val == EMPTY]
```

```python
# --- Minimax with Alpha-Beta Pruning ---
def minimax(b, depth, alpha, beta, maximizing, player):
    if depth == 0 or is_full(b):
        return evaluate(b, player), None

    opp = get_opponent(player)
    if maximizing:
        max_eval = -math.inf
        best_move = None
        for move in available_moves(b):
            before = count_potential_fours(b, player)
            b[move] = player
            after = count_potential_fours(b, player)
            bonus = (before - after) * 3

            eval_score, _ = minimax(b, depth - 1, alpha, beta, False, player)
            eval_score += bonus
            b[move] = EMPTY

            if eval_score > max_eval:
                max_eval = eval_score
                best_move = move
            alpha = max(alpha, eval_score)
            if beta <= alpha:
                break
        return max_eval, best_move
    else:
        min_eval = math.inf
        best_move = None
        for move in available_moves(b):
            b[move] = opp
            eval_score, _ = minimax(b, depth - 1, alpha, beta, True, player)
            b[move] = EMPTY
            if eval_score < min_eval:
                min_eval = eval_score
                best_move = move
            beta = min(beta, eval_score)
            if beta <= alpha:
                break
        return min_eval, best_move

# --- AI Move ---
def ai_move(b):
    print("💻 AI is thinking...")
```

```
    score, move = minimax(b, depth=3, alpha=-math.inf, beta=math.inf,
maximizing=True, player=AI)
    return move

# --- Game Loop ---
print("Welcome to 5x5 Strategic Game! You are 'X', AI is 'O'.")
print_board(board)

while True:
    # Player Move
    try:
        move = int(input(f"Enter your move (1-{SIZE*SIZE}): ")) - 1
        if move in available_moves(board):
            board[move] = PLAYER
        else:
            print("Invalid move. Try again.")
            continue
    except ValueError:
        print("Please enter a valid number.")
        continue

    print_board(board)

    if is_full(board):
        print("Game over! Board full.")
        break

    # AI Move
    ai_best = ai_move(board)
    board[ai_best] = AI
    print_board(board)

    if is_full(board):
        print("Game over! Board full.")
        break
```

OUTPUT:


Welcome to 5x5 Strategic Game! You are 'X', AI is 'O'.
Enter your move (1-25): 1
X


AI is thinking...
X | O | O | | X
Enter your move (1-25): 6
X | O | O | | X
X


X | O | O | O | X
X | O | O | O | X
X | X | X | O | O
O | X | O | O | X
O | X | X | X | X
Game over! Board full.

END