Devesh Vengurlekar
Roll No: 9766
TE Comps A

AI Experiment 1

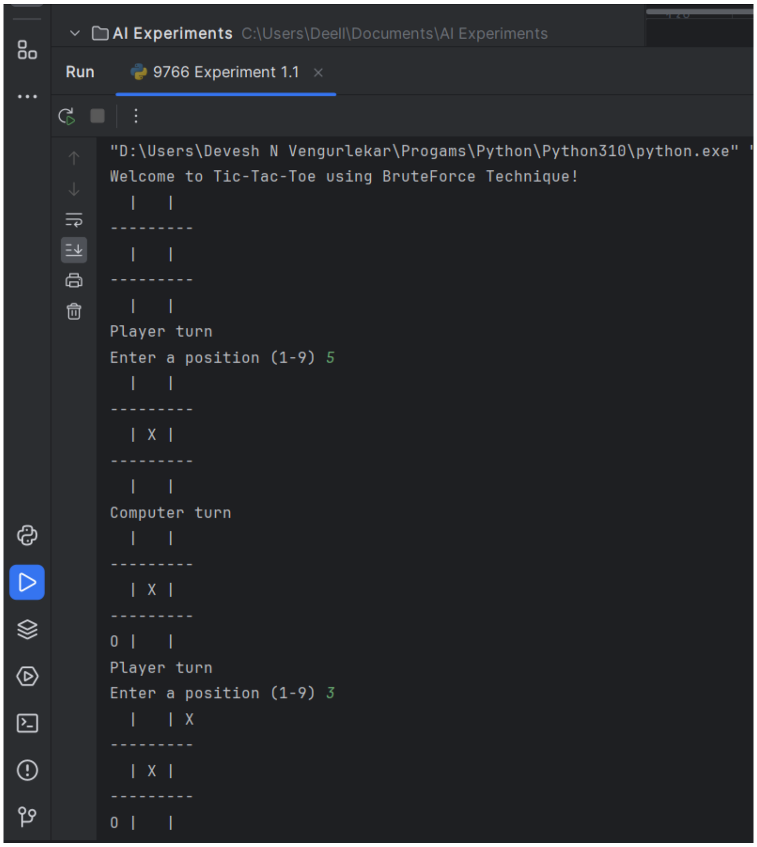
Tic Tac Toe game implementation by

a) Brute Force Method

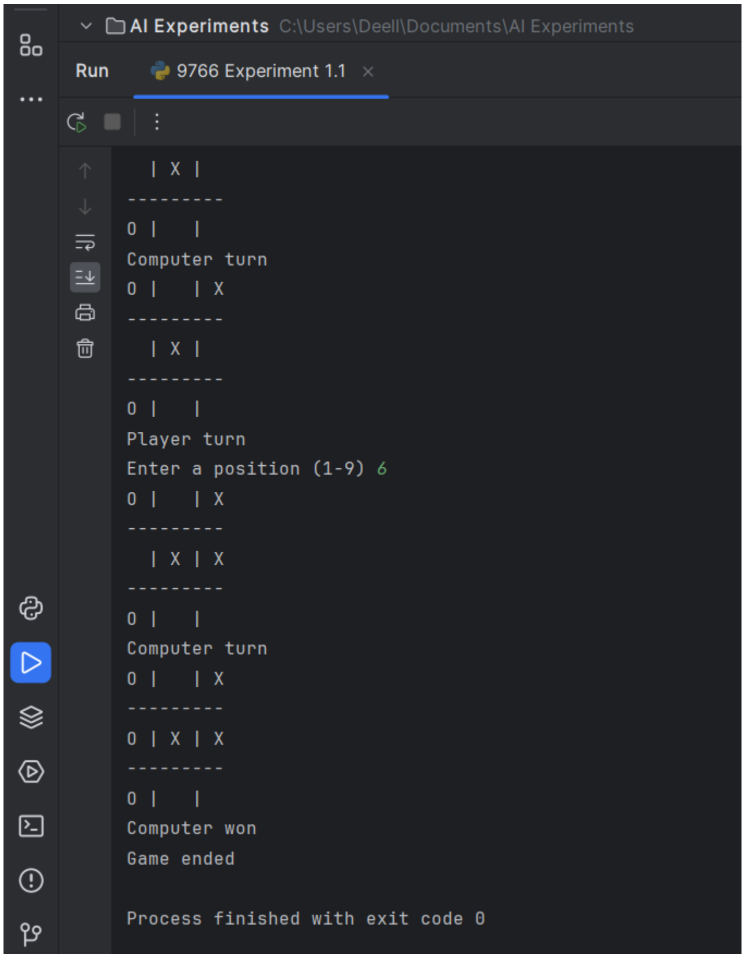
Program:

```
# Devesh Vengurlekar
# Roll No: 9766
# TE Comps A

import random

board = [' ' for x in range(9)]

def main():
    print('Welcome to Tic-Tac-Toe using BruteForce Technique!')

    print_board()

    game_end = False


    while not game_end:
        print('Player turn')
        player_turn()
        print_board()
        if check_winner(board):
            print('Player won')
            game_end = True
            break

        print('Computer turn')
        computer_move = computer_turn()
        if computer_move != -1:
            board[computer_move] = 'O'
            print_board()
            if check_winner(board):
                print('Computer won')
                game_end = True
                break


        if board.count(' ') < 1:
            print('Tie game')
            game_end = True

    print('Game ended')

def print_board():
    print(board[0] + ' | ' + board[1] + ' | ' + board[2])
    print('---------')
    print(board[3] + ' | ' + board[4] + ' | ' + board[5])
    print('---------')
    print(board[6] + ' | ' + board[7] + ' | ' + board[8])

def check_winner(board):
    # rows
    if ((board[0] == board[1] == board[2] != ' ') or
        (board[3] == board[4] == board[5] != ' ') or
        (board[6] == board[7] == board[8] != ' ')):
        return True

    # columns
    if ((board[0] == board[3] == board[6] != ' ') or
        (board[1] == board[4] == board[7] != ' ') or
        (board[2] == board[5] == board[8] != ' ')):
        return True

    # diagonals
    if ((board[0] == board[4] == board[8] != ' ') or
        (board[2] == board[4] == board[6] != ' ')):
        return True

    return False

def player_turn():
    made_move = False

    while not made_move:
        player_input = input('Enter a position (1-9) ')
        try:
            player_move = int(player_input)
            if player_move < 1 or player_move > 9:
                print('Enter a valid position')
            else:
                player_position = player_move - 1 # player index in board
                if board[player_position] != ' ':
                    print('Position is already taken')
                else:
                    board[player_position] = 'X'
                    made_move = True
        except:
            print('Enter a valid number')

def computer_turn():
    available_moves = [pos for pos, value in enumerate(board) if value == ' ']
    move = -1


    for i in available_moves:
        new_board = board[:]
        new_board[i] = 'O'
        if check_winner(new_board):
            move = i
            return move


    for i in available_moves:
        new_board = board[:]
        new_board[i] = 'X'
        if check_winner(new_board):
            move = i
            return move


    avalable_corners = []
    for i in available_moves:
        if i in [0, 2, 6, 8]:
            avalable_corners.append(i)
    if len(avalable_corners) > 0:
        random_index = random.randrange(0, len(avalable_corners))
        move = avalable_corners[random_index]
        return move
    if 4 in available_moves:
        move = 4
        return move


    avalable_edges = []
    for i in available_moves:
        if i in [1, 3, 5, 7]:
            avalable_edges.append(i)
    if len(avalable_edges) > 0:
        random_index = random.randrange(0, len(avalable_edges))
        move = avalable_edges[random_index]
        return move


    return move

if __name__ == '__main__':
    main()
```

Output:



```
"D:\Users\Devesh N Vengurlekar\Progams\Python\Python310\python.exe" "
Welcome to Tic-Tac-Toe using BruteForce Technique!
  |   |
---------
  |   |
---------
  |   |
Player turn
Enter a position (1-9) 5
  |   |
---------
  | X |
---------
  |   |
Computer turn
  |   |
---------
  | X |
---------
0 |   |
Player turn
Enter a position (1-9) 3
  |   | X
---------
  | X |
---------
0 |   |
```



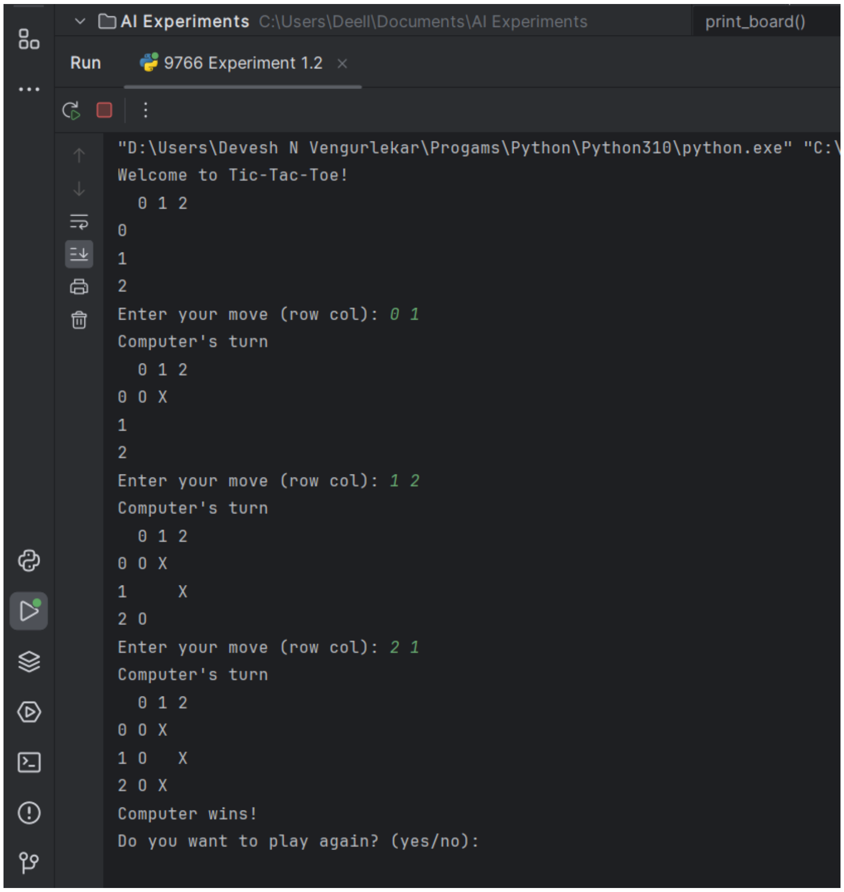
```
  | X |
---------
O |   |
Computer turn
O |   | X
---------
  | X |
---------
O |   |
Player turn
Enter a position (1-9) 6
O |   | X
---------
  | X | X
---------
O |   |
Computer turn
O |   | X
---------
O | X | X
---------
O |   |
Computer won
Game ended

Process finished with exit code 0
```

b) Heuristic Approach

Program:

```
import random

def print_board(board):
    print("  0 1 2")
    for i, row in enumerate(board):
        print(i, " ".join(row))


def check_winner(board, player):
    # Check rows, columns, and diagonals for a win
    for i in range(3):
        if all(board[i][j] == player for j in range(3)) or all(board[j][i] == player for j in
range(3)):
            return True
    if all(board[i][i] == player for i in range(3)) or all(board[i][2 - i] == player for i in
range(3)):
        return True
    return False


def evaluate(board):
    # Heuristic evaluation function
    if check_winner(board, 'X'):
        return -1  # Player X wins
    elif check_winner(board, 'O'):
        return 1  # Player O wins
    else:
        return 0  # It's a draw


def is_board_full(board):
    return all(board[i][j] != ' ' for i in range(3) for j in range(3))


def get_available_moves(board):
    return [(i, j) for i in range(3) for j in range(3) if board[i][j] == ' ']


def alphabeta(board, depth, alpha, beta, maximizing_player):
    if depth == 0 or check_winner(board, 'X') or check_winner(board, 'O') or
is_board_full(board):
        return evaluate(board)

    available_moves = get_available_moves(board)

    if maximizing_player:
        max_eval = float('-inf')
        for move in available_moves:
            i, j = move
            board[i][j] = 'O'
            eval = alphabeta(board, depth - 1, alpha, beta, False)
            board[i][j] = ' '  # Undo the move
            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)
            if beta <= alpha:
                break  # Beta cut-off
        return max_eval
    else:
        min_eval = float('inf')
        for move in available_moves:
            i, j = move
            board[i][j] = 'X'
            eval = alphabeta(board, depth - 1, alpha, beta, True)
            board[i][j] = ' '  # Undo the move
            min_eval = min(min_eval, eval)
            beta = min(beta, eval)
            if beta <= alpha:
                break  # Alpha cut-off
        return min_eval

def get_best_move(board):
    available_moves = get_available_moves(board)
    best_move = None
    best_eval = float('-inf')
    alpha = float('-inf')
    beta = float('inf')

    for move in available_moves:
        i, j = move
        board[i][j] = 'O'
        eval = alphabeta(board, 5, alpha, beta, False)  # Adjust depth as needed
        board[i][j] = ' '  # Undo the move

        if eval > best_eval:
            best_eval = eval
            best_move = move

    return best_move

def play_game():
    board = [[' ' for _ in range(3)] for _ in range(3)]
    game_end = False

    print('Welcome to Tic-Tac-Toe!')
    while not game_end:
        print_board(board)

        # Player's turn
        while True:
            try:
                player_move = tuple(map(int, input('Enter your move (row col): ').split()))
                if board[player_move[0]][player_move[1]] == ' ':
                    board[player_move[0]][player_move[1]] = 'X'
                    break
                else:
                    print('Invalid move. Try again.')
            except (ValueError, IndexError):
                print('Invalid input. Please enter row and column numbers separated by
space.')

        # Check if the player wins
        if check_winner(board, 'X'):
            print_board(board)
            print('You win!')
            break

        # Check for a draw
        if is_board_full(board):
            print_board(board)
            print('It\'s a draw!')
            break

        # Computer's turn
        print('Computer\'s turn')
        computer_move = get_best_move(board)
        board[computer_move[0]][computer_move[1]] = 'O'

        # Check if the computer wins
        if check_winner(board, 'O'):
            print_board(board)
            print('Computer wins!')
            break

        # Check for a draw again
        if is_board_full(board):
            print_board(board)
            print('It\'s a draw!')
            break


def main():
    while True:
        play_game()
        choice = input('Do you want to play again? (yes/no): ').lower()
        if choice != 'yes':
            break


if __name__ == "__main__":
    main()
```

Output:



```
"D:\Users\Devesh N Vengurlekar\Progams\Python\Python310\python.exe" "C:\
Welcome to Tic-Tac-Toe!
  0 1 2
0
1
2
Enter your move (row col): 0 1
Computer's turn
  0 1 2
0 O X
1
2
Enter your move (row col): 1 2
Computer's turn
  0 1 2
0 O X
1     X
2 O
Enter your move (row col): 2 1
Computer's turn
  0 1 2
0 O X
1 O   X
2 O X
Computer wins!
Do you want to play again? (yes/no):
```