Class: B.E (Computer), Sem – VI **Subject Name: Artificial Intelligence**

Student Name: Royce Dmello **Roll No:9533**

Practical No:	**1**
Title:	Tic Tac Toe game implementation by a) Brute Force Method b) Heuristic Approach
Date of Performance:	2/2/24
Date of Submission	9/2/24

Rubrics for Evaluation:

Sr. No	Performance Indicator	Excellent	Good	Below Average	Marks
1	On time Completion & Submission (01)	01 (On Time)	NA	00 (Not on Time)	
2	Logic/Algorithm Complexity analysis (03)	03(Correct)	02(Partial)	01 (Tried)	
3	Coding Standards (03): Comments/indention/Naming conventions Test Cases /Output	03(All used)	02 (Partial)	01 (rarely followed)	
4	Post Lab Assignment (03)	03(done well)	2 (Partially Correct)	1(submitted)	
Total					

Signature of the Teacher:

Experiment No: 1

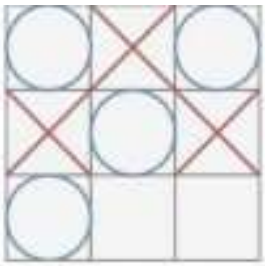
Title: Tic Tac Toe game implementation by

a) Brute Force Method

b) Heuristic Approach

Objective: To write a computer program in such a way that computer wins most of the time **Theory:**

This is a 2 players game where each player should put a cross or a circle on a 3 x 3 grid. The first player that has 3 crosses or 3 circles aligned (be it vertically, horizontally or diagonally) wins the game.



The blue player won because he aligned 3 blue circles on the diagona

a) Brute Force Method

A brute force approach is an approach that finds all the possible solutions to find a satisfactory solution to a given problem. The brute force algorithm tries out all the possibilities till a satisfactory solution is not found.

a) Consider a Board having nine element vectors.

b) Each element will contain

i) 0 for blank

ii) 1 indicating 'X' player move

iii) 2 indicating 'O' player move

c) Computer may play as an 'X' or O player.

d) First player always plays as 'X'.

2) MT is a vector of 3^9elements, each element of which is a nine-element vector representing board position.
3) MT is a vector of 3^9elements, each element of which is a nine-element vector representing board position.
   a) Move Table (MT) is a vector of 39 elements, each element of which is a nine element vector representing board position.

Index	Current Board position	New Board position
0	000000000	000010000
1	000000001	020000001
2	000000002	000100002
3	000000010	002000010
⋮		

   b) To make a move, do the following:

      a. View the vector (board) as a ternary number and convert it to its corresponding decimal number.

      b. Use the computed number as an index into the MT and access the vector stored there.

         i. The selected vector represents the way the board will look after the move.

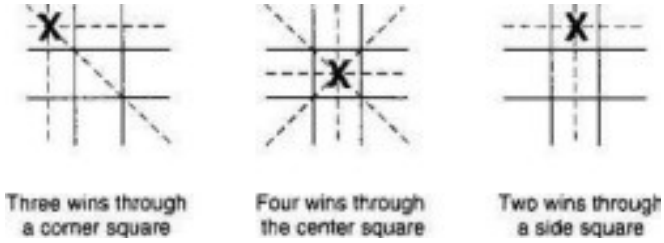
      c. Set board equal to that vector.

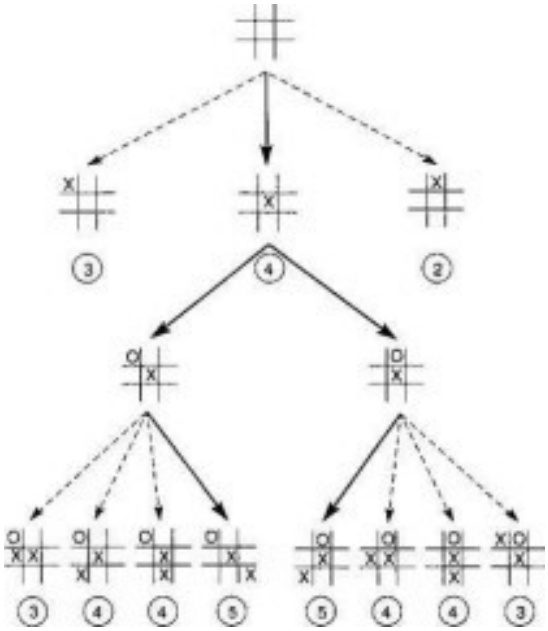
b) Heuristic Approach

Heuristics are essentially problem-solving tools that can be used for solving non-routine and challenging problems. A heuristic method is a practical approach for a short-term goal, such as solving a problem. The approach might not be perfect but can help find a quick solution to help move towards a reasonable way to resolve a problem.

Without considering symmetry the search space is 9! using symmetry the search space is 12 * 7! A simple heuristic is the number of solution paths still open when there are 8 total

paths (3 rows, 3 columns, 2 diagonals). Here is the search space using this heuristic. The total search space is now reduced to about 40, depending on the opponents play.

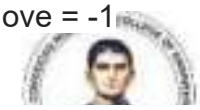




Code of Brute force approach

```
import random

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

def main():
    print('Game started')

    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:

BRUTE FORCE METHOD:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

Microsoft Windows [Version 10.0.22631.3085]
(c) Microsoft Corporation. All rights reserved.

C:\Users\SANJAY RAI\OneDrive\Desktop\TE_VI\AI_pracs>python TicTacToe_Brute_force.py
  0 1 2
0 - - -
1 - - -
2 - - -
Enter row (0, 1, or 2): 1
Enter column (0, 1, or 2): 1
  0 1 2
0 - - -
1 - X -
2 - - -
  0 1 2
0 O - -
1 - X -
2 - - -
Enter row (0, 1, or 2): 2
Enter column (0, 1, or 2): 2
  0 1 2
0 O - -
1 - X -
2 - - X
  0 1 2
0 O - O
1 - X -
2 - - X
Enter row (0, 1, or 2): 0
Enter column (0, 1, or 2): 1
```

```
Enter row (0, 1, or 2): 0
Enter column (0, 1, or 2): 1
  0 1 2
0 O X O
1 - X -
2 - - X
  0 1 2
0 O X O
1 - X -
2 - O X
Enter row (0, 1, or 2): 1
Enter column (0, 1, or 2): 2
  0 1 2
0 O X O
1 - X X
2 - O X
  0 1 2
0 O X O
1 O X X
2 - O X
Enter row (0, 1, or 2): 2
Enter column (0, 1, or 2): 1
Invalid move. Please try again.
Enter row (0, 1, or 2): 2
Enter column (0, 1, or 2): 0
  0 1 2
0 O X O
1 O X X
2 X O X
It's a draw!
Ln 143, Col 1   Spaces: 4   UTF-8   CRLF   Python   3.11.2 64-bit   Go Live
```

HEURISTIC METHOD: Ashram, Bandstand, Bandra (W), Mumbai - 400050

```
import random

def print_board(board):
    for row in board:
        print(' '.join(row))
    print()

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 minimax(board, depth, 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 = minimax(board, depth - 1, False)
            board[i][j] = ' '  # Undo the move
            max_eval = max(max_eval, eval)
        return max_eval
    else:
        min_eval = float('inf')
        for move in available_moves:
            i, j = move
            board[i][j] = 'X'
            eval = minimax(board, depth - 1, True)
            board[i][j] = ' '  # Undo the move
            min_eval = min(min_eval, eval)
```

```
        return min_eval

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

    for move in available_moves:
        i, j = move
        board[i][j] = 'O'
        eval = minimax(board, 2, False)  # You can adjust the depth for a more or less sophisticated AI
        board[i][j] = ' '  # Undo the move

        if eval > best_eval:
            best_eval = eval
            best_move = move

    return best_move

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

    print('Tic-Tac-Toe Game')

    while not game_end:
        print_board(board)

        # Player's turn
        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'
        else:
            print('Invalid move. Try again.')
            continue

        # 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

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

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

Microsoft Windows [Version 10.0.22631.3085]
(c) Microsoft Corporation. All rights reserved.

C:\Users\SANJAY RAI\OneDrive\Desktop\TE_VI\AI_pracs>python TicTacToe_Heuristic.py
  0 1 2
0 - - -
1 - - -
2 - - -
Enter your move (0-8): 5
  0 1 2
0 - - -
1 - - X
2 - - -
  0 1 2
0 O - -
1 - - X
2 - - -
Enter your move (0-8): 4
  0 1 2
0 O - -
1 - X X
2 - - -
  0 1 2
0 O - -
1 O X X
2 - - -
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

2 - - -
Enter your move (0-8): 6
  0 1 2
0 O - -
1 O X X
2 X - -
  0 1 2
0 O - O
1 O X X
2 X - -
Enter your move (0-8): 8
  0 1 2
0 O - O
1 O X X
2 X - X
  0 1 2
0 O O O
1 O X X
2 X - X
O wins!

C:\Users\SANJAY RAI\OneDrive\Desktop\TE_VI\AI_pracs>
```

Post Lab Assignment:

1. What is the easiest trick to win Tic Tac Toe?
2. What is the algorithm to follow to win a 5*5 Tic Tac Toe?
3. Is there a way to never lose at Tic-Tac-Toe?
4. What can tic-tac-toe help you with?

Post lab Assignment-1

1.) What is the easiest track to win Tic Tac To

→ The eaisct trick to win Tic-tae-Toe is as follows

i) Start by placing your first mark in the center square.

ii.) If your opponent does'nt pair their mar in the counter square, place your second ma in any corner

iii.) otherwise, place your second mark in t corner opposite to your first mark.

iv.) from your third move onwards, prior t completing rows, column or diagonsis while blacklog your opponent's moves

2.) what is the algorithm to follow to win a 5*5 tic tac Toe?

→ Algorithms:

1.) control the center square

2.) create two-in-a-row, three-in-a-row, four in a row combination horizontally, vertically and digonally.

3.) secure adjacent corner square to cre multiple winning paths.

4.) control edge square to add flexibility t combinations and block opponents moves

5.) Anticipate opponents move and block potential winning moves while advancing own strategy.

6.) Be flexible and adapt strategy based the current state of the board and opponent moves.

3.) Is there a way to never lose at Tic-Ta
→ 1.) Start in the center: Always begin w
the center square for more winning
opportunities in board games
2.) Create and block: prioritize forming
winning combination while blocking y
opponents moves to maintain control an
increase your chances of winning
3.) Adapt Strategy: Adjust your appr
based on the board state and oppon
moves to stay ahead and maximize yo
winning potentials.

4.) What can tic-tac-toe help you with?
→ Strategic thinking: planning and executi
moves to outnumber your opponent
Problem-solving: Analyzing the game s
and finding optimal moves to acheive vi
pattern recognization: Identifying patter
and potential winning combinations on bo
Score good grade: Studing tic-tac-toe wi
help to gain marks in AI.
Decision making: Evaluating different
options and selecting the best course of a
critical thinking: Accessing the consequen
of each move and predicting your oppone
responses.