Fr. Conceicao Rodrigues College of Engineering
Fr. Agnel Ashram, Bandstand, Bandra (W), Mumbai - 400050

**Department of Computer Engineering**
**Academic Term II: 23-24**

**Class: B.E (Computer), Sem – VI** **Subject Name: Artificial Intelligence**
**Student Name: Siddhesh Pradhan** **Roll No: 9632**

| | |
|---|---|
| **Practical No:** | **1** |
| **Title:** | Tic Tac Toe game implementation by<br>a) Brute Force Method<br>b) Heuristic Approach |
| **Date of Performance:** | 29/01/2024 |
| **Date of Submission:** | 13/02/2024 |

## 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:**

Fr. Conceicao Rodrigues College of Engineering
Fr. Agnel Ashram, Bandstand, Bandra (W), Mumbai - 400050

# Experiment No: 1

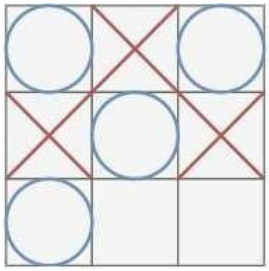
**Title**: Tic Tac Toe game implementation by

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 diagonal

### 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 $3^9$ elements, each element of which is a nine-element vector representing board position.
3) MT is a vector of $3^9$ elements, 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.

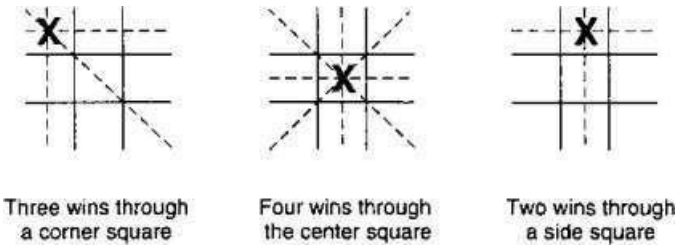
      c. Set board equal to that vector.

## b) Heuristic Approach

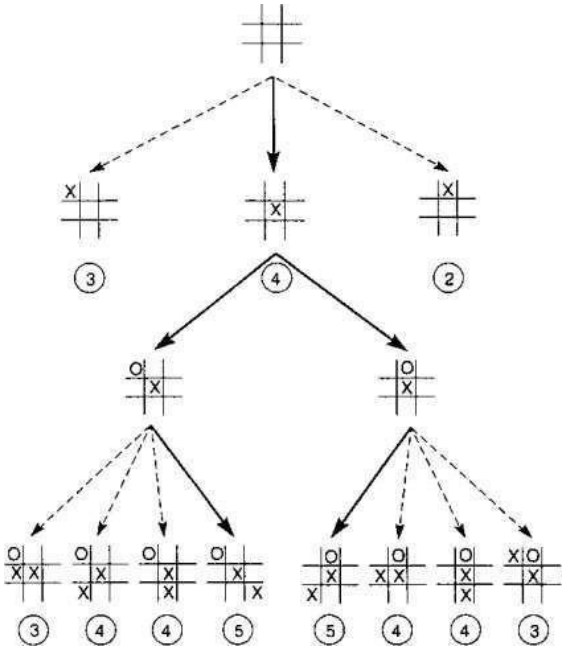
Heuristics are essentially problem-solving tools that can be used for solving non-routine and challenging problems. A heuristic method is a practical approach for a short-term goal, such as solving a problem. The approach might not be perfect but can help find a quick solution to help move towards a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



Three wins through a corner square | Four wins through the center square | Two wins through a side square



**OUTPUT:**

**expt1_BruteForce.py**

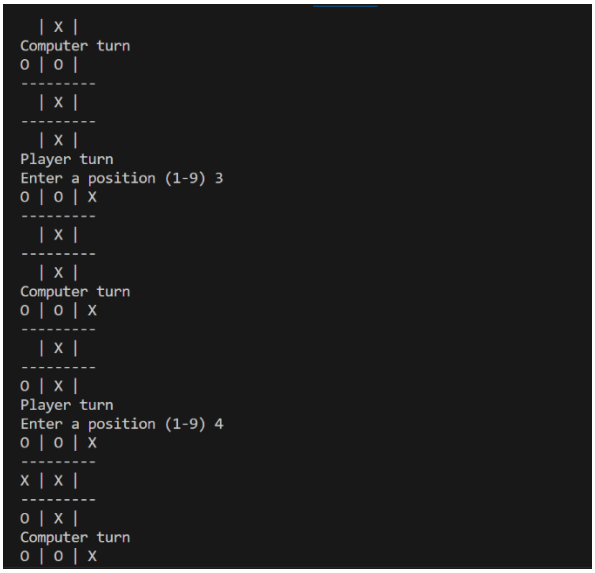
```
# Program for Tic-Tac-Toe using BruteForce Technique

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):
```

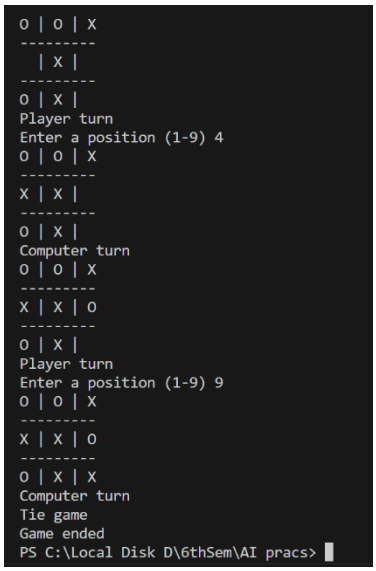
```
                print('Computer won')
                game_end = True
                break


        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()
```

```
PS C:\Local Disk D\6thSem\AI pracs> python3 expt1_BruteForce.py
Welcome to Tic-Tac-Toe using BruteForce Technique!
  |   |
---------
  |   |
---------
  |   |
Player turn
Enter a position (1-9) 5
  |   |
---------
  | X |
---------
  |   |
Computer turn
O |   |
---------
  | X |
---------
  |   |
Player turn
Enter a position (1-9) 8
O |   |
---------
  | X |
---------
  | X |
Computer turn
O | O |
```




**expt1_Heuristic.py**

```
# Program for Tic-Tac-Toe using Heuristic Technique

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()
```

```
PS C:\Local Disk D\6thSem\AI pracs> python3 expt1_Heuristic.py
Welcome to Tic-Tac-Toe!
  0 1 2
0
1
2
Enter your move (row col): 1 1
Computer's turn
  0 1 2
0 O
1   X
2
Enter your move (row col): 2 0
Computer's turn
  0 1 2
0 O   O
1   X
2 X
Enter your move (row col): 0 1
Computer's turn
  0 1 2
0 O X O
1   X
2 X O
Enter your move (row col): 2 2
Computer's turn
  0 1 2
0 O X O
1 O X
```

```
0 O
1   X
2
Enter your move (row col): 2 0
Computer's turn
  0 1 2
0 O   O
1   X
2 X
Enter your move (row col): 0 1
Computer's turn
  0 1 2
0 O X O
1   X
2 X O
Enter your move (row col): 2 2
Computer's turn
  0 1 2
0 O X O
1 O X
2 X O X
Enter your move (row col): 1 2
  0 1 2
0 O X O
1 O X X
2 X O X
It's a draw!
Do you want to play again? (yes/no): no
PS C:\Local Disk D\6thSem\AI pracs>
```

**Post Lab Assignment:**

1. What is the easiest trick to win Tic Tac Toe?

Ans:- The easiest trick to win Tic Tac Toe is to play as the first player and start in the center square. This gives you the greatest chance of winning or forcing a draw if your opponent plays perfectly.

2. What is the algorithm to follow to win a 5*5 Tic Tac Toe?

Ans:- To win a 5x5 Tic Tac Toe game, you can follow a simple algorithm:

1. Start in the center: Begin by placing your first mark in the center square.

2. Block the opponent's attempts: Try to block your opponent from forming a winning line while creating your own.

3. Build your own winning line: Continue to place your marks to create a line of 5 in a row horizontally, vertically, or diagonally.

If both players play optimally, the game will likely end in a draw, but this algorithm can give you the best chance of winning if your opponent makes a mistake.

3. Is there a way to never lose at Tic-Tac-Toe?

Ans:- Yes, there is a way to never lose at Tic-Tac-Toe if you play perfectly. This involves following a specific strategy known as the "perfect strategy" or "optimal strategy," which ensures that you either win or draw the game. The key to this strategy is to always make the best possible move based on the current state of the game, which requires analyzing all possible future moves and outcomes.

4. What can tic-tac-toe help you with?

Ans:- Tic-tac-toe can help develop several skills and abilities, including:

1. Strategic Thinking: It encourages players to think ahead and anticipate their opponent's moves.
2. Problem-Solving: Players need to find the best move in different situations, which can improve their problem-solving skills.
3. Spatial Awareness: The game involves placing marks on a grid, which can help improve spatial awareness and visualization skills.
4. Decision-Making: Players must make decisions based on the current state of the game and potential future outcomes.
5. Pattern Recognition: Recognizing patterns can help players predict their opponent's moves and plan their own strategy.
6. Critical Thinking: Players need to analyze the game board and make decisions based on logical reasoning.
7. Persistence: The game can teach perseverance and the importance of not giving up, even in challenging situations.

Overall, tic-tac-toe is a simple game that can provide valuable lessons and skills that are applicable in various aspects of life.