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

**Department of Computer Engineering**

**Academic Term II: 23-24**

**Class: T.E (Computer), Sem – VI Subject Name: Artificial Intelligence Student Name: Roll No: 9538**

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

**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(submitte  d) |  |
| **Total** | | | | |  |

**Signature of the Teacher:**

a) Brute Force Method

#exp1\_9538\_BruteForce

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:

Game started

| |

---------

| |

---------

| |

Player turn

Enter a position (1-9) 5

| |

---------

| X |

---------

| |

Computer turn

| |

---------

| X |

---------

| | O

Player turn

Enter a position (1-9) 3

| | X

---------

| X |

---------

| | O

Computer turn

| | X

---------

| X |

---------

O | | O

Player turn

Enter a position (1-9) 8

| | X

---------

| X |

---------

O | X | O

Computer turn

| O | X

---------

| X |

---------

O | X | O

Player turn

Enter a position (1-9) 4

| O | X

---------

X | X |

---------

O | X | O

Computer turn

| O | X

---------

X | X | O

---------

O | X | O

Player turn

Enter a position (1-9) 1

X | O | X

---------

X | X | O

---------

O | X | O

Computer turn

Tie game

Game ended

b) Heuristic Approach

#HeuristicApproach

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()

OUTPUT:

Tic-Tac-Toe Game

Enter your move (row col): 2 0

Computer's turn

O

X

Enter your move (row col): 1 1

Computer's turn

O O

X

X

Enter your move (row col): 2 2

Computer's turn

O O O

X

X X

Computer wins!