# EXPERIMENT 2

## PROBLEM TITLE

Write a Program to Implement Breadth-First Search (for Tic-Tac-Toe Problem).

## CODE

```
from collections import deque

class TicTacToe:
    def __init__(self):
        self.board = [' ' for _ in range(9)]

    def is_goal(self):
        # Check rows, columns, and diagonals for a win
        for i in range(3):
            if self.board[i*3] == self.board[i*3+1] == self.board[i*3+2] != ' ':
                return True
            if self.board[i] == self.board[i+3] == self.board[i+6] != ' ':
                return True
        if self.board[0] == self.board[4] == self.board[8] != ' ':
            return True
        if self.board[2] == self.board[4] == self.board[6] != ' ':
            return True
        return False

    def available_moves(self):
        return [i for i, x in enumerate(self.board) if x == ' ']

    def make_move(self, move, player):
        self.board[move] = player

def bfs_tic_tac_toe():
    initial_state = TicTacToe()
    queue = deque([(initial_state, 'X')])
    visited = set()

    while queue:
        state, player = queue.popleft()
        if state.is_goal():
            return state.board
        for move in state.available_moves():
            new_state = TicTacToe()
            new_state.board = state.board.copy()
            new_state.make_move(move, player)
            if tuple(new_state.board) not in visited:
                visited.add(tuple(new_state.board))
                queue.append((new_state, 'O' if player == 'X' else 'X'))

# Run BFS on Tic-Tac-Toe
result = bfs_tic_tac_toe()
print("BFS Resulting Board:", result)
```

**EXPECTED OUTPUT**

```
BFS Resulting Board: ['X', 'X', 'X', ' ', ' ', ' ', ' ', ' ', ' ']
```