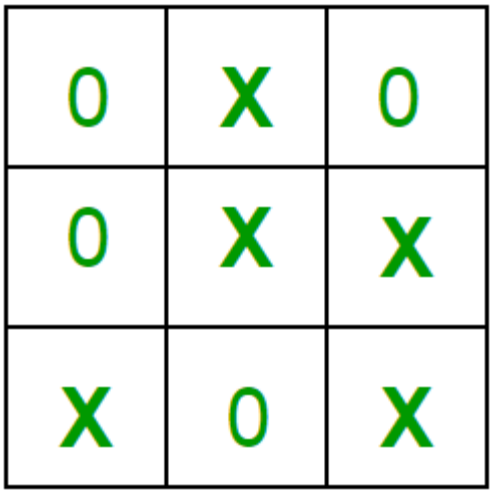
**Exp 3: Implementation of Breadth First Search for Tic-Tac-Toe problem.**

**Rules of the Game:**

- The game is to be played between two people (in this program between HUMAN and COMPUTER).
- One of the player chooses 'O' and the other 'X' to mark their respective cells.
- The game starts with one of the players and the game ends when one of the players has one whole row/ column/ diagonal filled with his/her respective character ('O' or 'X').
- If no one wins, then the game is said to be draw.




**Implementation:** In the program the moves taken by the computer and the human are chosen randomly. We use rand() function for this. What more can be done in the program? The program is in not played optimally by both sides because the moves are chosen randomly. The program can be easily modified so that both players play optimally (which will fall under the category of Artificial Intelligence). Also the program can be modified such that the user himself gives the input (using scanf() or cin). The above changes are left as an exercise to the readers. Winning Strategy – An Interesting Fact If both the players play optimally then it is destined that you will never lose ("although the match can still be drawn"). It doesn't matter whether you play first or second.In another ways – " Two expert players will always draw ". Isn't this interesting ?

**Program:**

```
# Set up the game board as a list
board = ["-", "-", "-",
    "-", "-", "-",
    "-", "-", "-"]

# Define a function to print the game board
def print_board():
  print(board[0] + " | " + board[1] + " | " + board[2])
  print(board[3] + " | " + board[4] + " | " + board[5])
  print(board[6] + " | " + board[7] + " | " + board[8])

# Define a function to handle a player's turn
```

```
def take_turn(player):
  print(player + "'s turn.")
  position = input("Choose a position from 1-9: ")
  while position not in ["1", "2", "3", "4", "5", "6", "7", "8", "9"]:
    position = input("Invalid input. Choose a position from 1-9: ")
  position = int(position) - 1
  while board[position] != "-":
    position = int(input("Position already taken. Choose a different
position: ")) - 1
  board[position] = player
  print_board()

# Define a function to check if the game is over
def check_game_over():
  # Check for a win
  if (board[0] == board[1] == board[2] != "-") or \
  (board[3] == board[4] == board[5] != "-") or \
  (board[6] == board[7] == board[8] != "-") or \
  (board[0] == board[3] == board[6] != "-") or \
  (board[1] == board[4] == board[7] != "-") or \
  (board[2] == board[5] == board[8] != "-") or \
  (board[0] == board[4] == board[8] != "-") or \
  (board[2] == board[4] == board[6] != "-"):
    return "win"
  # Check for a tie
  elif "-" not in board:
    return "tie"
  # Game is not over
  else:
    return "play"

# Define the main game loop
def play_game():
  print_board()
  current_player = "X"
  game_over = False
  while not game_over:
    take_turn(current_player)
    game_result = check_game_over()
    if game_result == "win":
      print(current_player + " wins!")
      game_over = True
    elif game_result == "tie":
      print("It's a tie!")
      game_over = True
    else:
      # Switch to the other player
      current_player = "O" if current_player == "X" else "X"
```

```
# Start the game
play_game()
```

**Using BFS:**

To implement Breadth-First Search (BFS) for the Tic-Tac-Toe problem, you can modify your code to include a BFS algorithm that explores possible game states. The BFS algorithm will be used to find the optimal move for the computer player (assuming the computer plays as "O").

```
import random

# Set up the game board as a list
board = ["-", "-", "-",
         "-", "-", "-",
         "-", "-", "-"]

# Define a function to print the game board
def print_board():
    print(board[0] + " | " + board[1] + " | " + board[2])
    print(board[3] + " | " + board[4] + " | " + board[5])
    print(board[6] + " | " + board[7] + " | " + board[8])

# Define a function to check if the game is over
def check_game_over():
    # Check for a win
    if (board[0] == board[1] == board[2] != "-") or \
            (board[3] == board[4] == board[5] != "-") or \
            (board[6] == board[7] == board[8] != "-") or \
            (board[0] == board[3] == board[6] != "-") or \
            (board[1] == board[4] == board[7] != "-") or \
            (board[2] == board[5] == board[8] != "-") or \
            (board[0] == board[4] == board[8] != "-") or \
            (board[2] == board[4] == board[6] != "-"):
        return "win"
    # Check for a tie
    elif "-" not in board:
        return "tie"
    # Game is not over
    else:
        return "play"

# Define a function to check if the current player has won
def check_win(player):
    return (board[0] == board[1] == board[2] == player) or \
           (board[3] == board[4] == board[5] == player) or \
           (board[6] == board[7] == board[8] == player) or \
           (board[0] == board[3] == board[6] == player) or \
           (board[1] == board[4] == board[7] == player) or \
```

```
            (board[2] == board[5] == board[8] == player) or \
            (board[0] == board[4] == board[8] == player) or \
            (board[2] == board[4] == board[6] == player)

# Define a function to handle a player's turn
def take_turn(player):
    print(player + "'s turn.")
    if player == "X":
        position = input("Choose a position from 1-9: ")
        while position not in ["1", "2", "3", "4", "5", "6", "7", "8",
"9"] or board[int(position) - 1] != "-":
            position = input("Invalid input or position already taken.
Choose a position from 1-9: ")
        position = int(position) - 1
    else:
        # Computer's turn (O)
        available_positions = [i for i in range(9) if board[i] == "-"]
        position = random.choice(available_positions)

    board[position] = player
    print_board()

# Define the main game loop
def play_game():
    print_board()
    current_player = "X"
    game_over = False
    while not game_over:
        take_turn(current_player)
        game_result = check_game_over()
        if game_result == "win":
            print(current_player + " wins!")
            game_over = True
        elif game_result == "tie":
            print("It's a tie!")
            game_over = True
        else:
            # Switch to the other player
            current_player = "O" if current_player == "X" else "X"

# Start the game
play_game()
```

BFS (DQUEUE):

```
from collections import deque
import copy
import random
```

```
# Set up the game board as a list
board = ["-", "-", "-",
         "-", "-", "-",
         "-", "-", "-"]

# Define a function to print the game board
def print_board():
    print(board[0] + " | " + board[1] + " | " + board[2])
    print(board[3] + " | " + board[4] + " | " + board[5])
    print(board[6] + " | " + board[7] + " | " + board[8])

# Define a function to check if the game is over
def check_game_over():
    # Check for a win
    if (board[0] == board[1] == board[2] != "-") or \
            (board[3] == board[4] == board[5] != "-") or \
            (board[6] == board[7] == board[8] != "-") or \
            (board[0] == board[3] == board[6] != "-") or \
            (board[1] == board[4] == board[7] != "-") or \
            (board[2] == board[5] == board[8] != "-") or \
            (board[0] == board[4] == board[8] != "-") or \
            (board[2] == board[4] == board[6] != "-"):
        return "win"
    # Check for a tie
    elif "-" not in board:
        return "tie"
    # Game is not over
    else:
        return "play"

# Define a function to check if the current player has won
def check_win(player):
    return (board[0] == board[1] == board[2] == player) or \
           (board[3] == board[4] == board[5] == player) or \
           (board[6] == board[7] == board[8] == player) or \
           (board[0] == board[3] == board[6] == player) or \
           (board[1] == board[4] == board[7] == player) or \
           (board[2] == board[5] == board[8] == player) or \
           (board[0] == board[4] == board[8] == player) or \
           (board[2] == board[4] == board[6] == player)

# Define a function to handle a player's turn
def take_turn(player):
    print(player + "'s turn.")
    if player == "X":
        position = input("Choose a position from 1-9: ")
```

```
        while position not in ["1", "2", "3", "4", "5", "6", "7", "8",
"9"] or board[int(position) - 1] != "-":
            position = input("Invalid input or position already taken.
Choose a position from 1-9: ")
        position = int(position) - 1
    else:
        # Computer's turn (O) using BFS
        position = make_computer_move()

    board[position] = player
    print_board()

# Define a function to make a move using BFS for computer player (O)
def make_computer_move():
    queue = deque([(copy.deepcopy(board), "O")])

    while queue:
        current_board, current_player = queue.popleft()

        for i in range(9):
            if current_board[i] == "-":
                new_board = current_board[:]
                new_board[i] = current_player

                if check_win(current_player):
                    return i

                queue.append((copy.deepcopy(new_board), "X" if
current_player == "O" else "O"))

    # If no winning move is found, make a random move
    available_positions = [i for i in range(9) if board[i] == "-"]
    return random.choice(available_positions)

# Define the main game loop
def play_game():
    print_board()
    current_player = "X"
    game_over = False
    while not game_over:
        take_turn(current_player)
        game_result = check_game_over()
        if game_result == "win":
            print(current_player + " wins!")
            game_over = True
        elif game_result == "tie":
            print("It's a tie!")
            game_over = True
```

```
        else:
            # Switch to the other player
            current_player = "O" if current_player == "X" else "X"

# Start the game
play_game()
```