code:-

```
from collections import defaultdict, deque
def bfs(graph, start, goal):
    visited = set()
    queue = deque([(start, [start])])
    while queue:
        node, path = queue.popleft()
        if node == goal:
            return path
        if node not in visited:
            visited.add(nodes)
            for neighbor in graph[node]:
                queue.append((neighbour, path + [neighbor]))
    return None
graph = defaultdict(list)
num_nodes = int(input("Enter the number of nodes:"))
num_edges = int(input("Enter the number of edges:"))
for _ in range(num_edges):
    u, v = input("Enter edge (u v):").split()
    graph[u].append(v)
    graph[v].append(u)
nodes = list(graph.keys())
print("Available nodes:", nodes)
start_node = input("Enter the start node:")
goal_node = input("Enter the goal node:")
if start_node not in nodes or goal_node not in nodes:
    print("Invalid nodes entered.")
else:
    path = bfs(graph, start_node, goal_node)
    if path:
        print("Path from", start_node, "to", goal_node, ":", "→"
              .join(path))
    else:
        print("No path found from", start_node, "to", goal_node)
```

OUTPUT:-

```
Enter the number of nodes : 25
Enter the number of edges : 27
Enter edge (U V) : A B
Enter edge (U V) : A C
Enter edge (U V) : B D
Enter edge (U V) : B E
Enter edge (U V) : C F
Enter edge (U V) : C G
Enter edge (U V) : D H
Enter edge (U V) : D I
Enter edge (U V) : E J
Enter edge (U V) : E K
Enter edge (U V) : F L
Enter edge (U V) : F M
Enter edge (U V) : G N
Enter edge (U V) : G O
Enter edge (U V) : H P
Enter edge (U V) : H Q
Enter edge (U V) : I R
Enter edge (U V) : I S
Enter edge (U V) : J T
Enter edge (U V) : J U
Enter edge (U V) : K V
Enter edge (U V) : K W
Enter edge (U V) : L X
Enter edge (U V) : L Y
Enter edge (U V) : M X
Enter edge (U V) : M Y
Enter edge (U V) : N O
Available nodes : ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K',
                  'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V',
                  'W', 'X', 'Y']

Enter start node : A
Enter goal node : Y
Path from A to Y : A -> C -> F -> L -> Y
```

```
Enter the number of nodes : 25
Enter the number of edges : 27
Enter edge (U V) : A B
Enter edge (U V) : A C
Enter edge (U V) : B D
Enter edge (U V) : B E
Enter edge (U V) : C F
Enter edge (U V) : C G
Enter edge (U V) : D H
Enter edge (U V) : D I
Enter edge (U V) : E J
Enter edge (U V) : E K
Enter edge (U V) : F L
Enter edge (U V) : F M
Enter edge (U V) : G N
Enter edge (U V) : G O
Enter edge (U V) : H P
Enter edge (U V) : H Q
Enter edge (U V) : I R
Enter edge (U V) : I S
Enter edge (U V) : J T
Enter edge (U V) : J U
Enter edge (U V) : K V
Enter edge (U V) : K W
Enter edge (U V) : L X
Enter edge (U V) : L Y
Enter edge (U V) : M X
Enter edge (U V) : M Y
Enter edge (U V) : N O
Available nodes : ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
                  'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
                  'U', 'V', 'W', 'X', 'Y']

Enter start node : A
Enter goal node : Z
Invalid node entered
```

code:-

```
from collections import defaultdict
def bfs(graph, current_node, goal_node, visited, path):
    if current_node == goal_node:
        return path
    visited.add(current_node)
    for neighbor in graph[current_node]:
        if neighbor not in visited:
            new_path = dfs(graph, neighbor, goal_node, visited, path+[neighbor])
            if new_path:
                return new_path

    return None
graph = defaultdict(list)
num_nodes = int(input("Enter the number of nodes :"))
num_edges = int(input("Enter the number of edges :"))
for _ in range(num_edges):
        u,v = input("Enter edge (u v): ").split()
        graph[u].append(v)
        graph[v].append(u)
nodes = list(graph.keys())
print("Available nodes :", nodes)
start_node = input("Enter the start node :")
goal_node = input("Enter the goal node :")
if start_node not in nodes or goal_node not in nodes:
        print("Invalid nodes entered")

else:
    path = dfs(graph, start_node, "to", goal_node, set(), [start_node])
    if path:
        print("Path from", start_node, "to", goal_node, ":", " -> ".join(path))

    else:
        print("No path found from", start_node, "to", goal_node)
```

OUTPUT:-

```
Enter the number of nodes : 25
Enter the number of edges : 27
Enter edge (U V) : A B
Enter edge (U V) : A C
Enter edge (U V) : B D
Enter edge (U V) : B E
Enter edge (U V) : C F
Enter edge (U V) : C G
Enter edge (U V) : D H
Enter edge (U V) : D I
Enter edge (U V) : E J
Enter edge (U V) : E K
Enter edge (U V) : F L
Enter edge (U V) : F M
Enter edge (U V) : G N
Enter edge (U V) : G O
Enter edge (U V) : H P
Enter edge (U V) : H Q
Enter edge (U V) : I R
Enter edge (U V) : I S
Enter edge (U V) : J T
Enter edge (U V) : J U
Enter edge (U V) : K V
Enter edge (U V) : K W
Enter edge (U V) : L X
Enter edge (U V) : L Y
Enter edge (U V) : M X
Enter edge (U V) : M Y
Enter edge (U V) : N O
Available nodes : ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K',
'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U',
'V', 'W', 'X', 'Y']

Enter the start node : A
Enter the goal node : V
Path from A to V : A -> C -> F -> L -> X -> M -> Y
```

Enter the number of nodes : 25
Enter the number of edges : 27
Enter edge (u v) : A B
Enter edge (u v) : A C
Enter edge (u v) : B D
Enter edge (u v) : B E
Enter edge (u v) : C F
Enter edge (u v) : C G
Enter edge (u v) : D H
Enter edge (u v) : D I
Enter edge (u v) : E J
Enter edge (u v) : E K
Enter edge (u v) : F L
Enter edge (u v) : F M
Enter edge (u v) : G N
Enter edge (u v) : G O
Enter edge (u v) : H P
Enter edge (u v) : H Q
Enter edge (u v) : I R
Enter edge (u v) : I S
Enter edge (u v) : J T
Enter edge (u v) : J U
Enter edge (u v) : K V
Enter edge (u v) : K W
Enter edge (u v) : L X
Enter edge (u v) : L Y
Enter edge (u v) : M X
Enter edge (u v) : M Y
Enter edge (u v) : N O
Available nodes : ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y']

Enter the start node : A
Enter the goal node : Z
Invalid nodes entered

Code :-

```
import random
def TicTacToe():
    Print("Welcome to TicTacToe game!")
    def PrintBoard():
        Print()
        Print(' ', board[0], "|", board[1], "|", board[2])
        Print("---|---|---")
        Print(' ', board[3], "|", board[4], "|", board[5])
        Print("---|---|---")
        Print(' ', board[6], "|", board[7], "|", board[8])
        print()
    def GetRow():
        while True:
            row = input("choose row (1,2,3): ")
            if row in ['1', '2', '3']:
                return int(row)
            else:
                Print("In Invalid input. choose row between 1 and 3.")

    def GetColumn():
        while True:
            column = input("choose column (1,2,3): ")
            if column in ['1', '2', '3']:
                return int(column)
            else:
                Print("In Invalid input choose column between 1 and 3.")

    def GetNumber():
        while True:
            number = input()
            try:
                number = int(number)
                if number in range(1,10):
                    return number
                else:
                    Print("In Number not in board")
            except value error:
                Print("In that's not a number. try again")
                continue
```

```
def getBoardCopy(board):
    dupeBoard = []
    for i in board:
        dupeBoard.append(i)
    return dupeBoard

def isSpaceFree(board1, move):
    return board1[move - 1] == ' '

def makeMove(board1, letter, move):
    board1[move - 1] = letter

def chooseRandomMoveFromList(board, movesList):
    possibleMoves = []
    for i in movesList:
        if isSpaceFree(board, i):
            possibleMoves.append(i)
    if len(possibleMoves) != 0:
        return random.choice(possibleMoves)
    else:
        return None

def computerChoice():
    for i in range(1, 10):
        copy = getBoardCopy(board)
        if isSpaceFree(copy, i):
            makeMove(copy, 'O', i):
            if checkWin(copy, 'O'):
                return i

    for i in range(1, 10):
        copy = getBoardCopy(board)
        if isSpaceFree(copy, i):
            makeMove(copy, 'X', i)
            if checkWin(copy, 'X'):
                return i
    if isSpaceFree(board, 5):
        return 5
        move = chooseRandomMoveFromList(board, [1, 3, 7, 9])
        if move != none:
            return move
    return chooseRandomMoveFromList(board, [2, 4, 6, 8])
```

```
def Turn(Player):
    placing_index = GetNumber() - 1
    if board[placing_index] == 'X' or board[placing_index] == 'O':
        Print('In Box already occupied. Try another one')
        Turn(Player)
    else:
        board[placing_index] = Player
def Turn1(move):
    board[move -1] = 'O'
def checkWin(board1, Player):
    for x in range(9):
        for y in range(9):
            for z in range(9):
                if x != y and y != z and z != x:
                    if board1[x] == Player and board1[y]
                    == Player and board1[z] == Player:
                        if MagicSquare[x] + MagicSquare
                        [y] + MagicSquare = 15 :
                            return True

def isBoardFull(board):
    count = 0
    for a in range(9):
        if board[a] == "X" or board[a] == "O":
            count += 1
    if count == 9:
        Print(" The game ends in a Tie \n")
        return True
Print (" who Plays first? (C for computer / H for Human) :")
first_Player = input().upper()
while True:
    board = [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
    MagicSquare = [4, 9, 2, 3, 5, 7, 8, 1, 6]
    if first_Player == 'C':
        while True:
            printBoard()
            end = checkWin(board, "O")
```

```
if end :
    print("computer wins the game")
    break
else :
    if isBoardFull(board):
        break
move = computerchoice()
turn1(move)
printBoard()
end = checkwin(board, "X")
if end :
    print("you win the game")
    break
else :
    if isBoardFull(board):
        break
print("choose a box player X")
row = GetRow()
column = Get column()
if board[(row-1)*3 + column-1] == "X" or board[(row-1)*
3 + column-1] == "O":
        print("In Box already occupied. Try another one")

else :
    board[(row-1)*3 + column-1] = "X"

elif first_player == 'H':
    while True :
        printBoard()
        row = GetRow()
        column = Get column()
        if board[(row-1)*3 + column-1] == "X" or board[(
        row-1)*3 + column-1] == "O":
            print("In Box already occupied. Try another one")
            continue
        else :
            board[(row-1)*3 + column-1] = "X"
        end = checkwin(board, "X")
        if end :
            print("you win the game")
            break
```

```
        else:
            if isBoardFull(board):
                break
        move = computerchoice()
        run1(move)
        printBoard()
        end = checkwin(board, "O")
        if end:
            print("computer wins the game")
            break
        else:
            if isBoardFull(board):
                break
play_again = input("Do you want to play again? (Y/N): ").upper()
if play_again != 'Y':
    break

TicTacToe()
```

output:-

```
welcome to TicToc Toe game!
who plays first? (C for computer / H for Human):
H
```

```
   |   |
---|---|---
   |   |
---|---|---
   |   |
```

```
choose row (1,2,3): 1
choose column (1,2,3): 1
```

```
 X |   |
---|---|---
   |   |
---|---|---
   |   |

 X |   |
---|---|---
   | O |
---|---|---
   |   |
```

```
choose row (1,2,3): 1
choose column (1,2,3): 2
```

```
 X | X |
---|---|---
   | O |
---|---|---
   |   |

 X | X | O
---|---|---
   | X |
---|---|---
   |   |
```

```
choose row (1,2,3): 3
choose column (1,2,3): 1
```

```
 X | X | O
---|---|---
   | O |
---|---|---
 X |   |
```

```
 X | X | O
---|---|---
 O | O |
---|---|---
 X |   |
```

choose row (1,2,3) : 2
choose column (1,2,3) : 3

```
 X | X | O
---|---|---
 O | O | X
---|---|---
 X |   | O
```

```
 X | X | O
---|---|---
 O | O | X
---|---|---
 X |   | O
```

choose row (1,2,3) : 3
choose column (1,2,3) : 2

```
 X | X | O
---|---|---
 O | O | X
---|---|---
 X | X | O
```

The game ends in a tie

Do you want to play again (Y/N) : y

```
Welcome to Tic Tac Toe game!
Who plays first? (C for computer / H for Human):
H

     |   |
  --1--1--
     |   |
  --1--1--
     |   |

Choose row (1,2,3): 1
Choose column(1,2,3): 1

  X  |   |
  --1--1--
     |   |
  --1--1--
     |   |

  X  |   |
  --1--1--
     | O |
  --1--1--
     |   |

Choose row (1,2,3): 2
Choose column(1,2,3): 1

  X  |   |
  --1--1--
  X  | O | O
  --1--1--
     |   |

  X  |   |
  --1--1--
  X  | O | O
  --1--1--
     |   |

Choose row (1,2,3): 3
Choose column(1,2,3): 1

  X  |   |
  --1--1--
  X  | O | O
  --1--1--
  X  |   |

  X  |   | O
  --1--1--
  X  | O | O
  --1--1--
  X  |   |

Human wins the game
Do you want to play again (Y/N): N
```

```
   |   |
---+---+---
   |   |
---+---+---
   |   |
```

choose row(1,2,3) : 2
choose column(1,2,3) : 2

```
   |   |
---+---+---
   | X |
---+---+---
   |   |
```

```
   |   |
---+---+---
   | X |
---+---+---
   |   O
```

choose row(1,2,3) : 1
choose column(1,2,3) : 3

```
   |   | X
---+---+---
   | X |
---+---+---
   |   O
```

```
   |   | X
---+---+---
   | X |
---+---+---
O  |   O
```

choose row(1,2,3) : 2
choose column(1,2,3) : 3

```
   |   | X
---+---+---
   | X | X
---+---+---
O  |   O
```

```
   |   | X
---+---+---
   | X | X
---+---+---
O  | O | O
```

computer wins the game
Do you want to play again? (Y/N) : N