UNIVERSITY INSTITUTE OF ENGINEERING AND TECHNOLOGY

PANJAB UNIVERSITY - CHANDIGARH

AI LAB PRACTICALS

Dept. of Information Technology

5th Semester 2023

Prepared by:

Abhay kumar

IT 5th Semester

UE218002

Submitted to:

Ms. Shewangi

IT Department

INDEX

S.NO	Practical	Page no.	Signature & Remark
1	Write a program to implement Water-Jug Problem using Breadth First Search algorithm		
2	Write a program to implement Tic-Tac-Toe Game :		
3	Write a program to implement n-queens problem		
4	Write a program to implement Missionary Cannibal problem		
5	Write a program to implement 8-puzzle problem using A* algorithm		

Practical 1

Write a program to implement Water-Jug Problem using Breadth First Search algorithm :

Code :-

```
def BFS(a, b, target):

    m = {}
    isSolvable = False
    path = []
    q = deque()
    q.append((0, 0))

    while (len(q) > 0):

        u = q.popleft()
         #print(f"u = {u}")
        if ((u[0], u[1]) in m):
            continue
        if ((u[0] > a or u[1] > b or
             u[0] < 0 or u[1] < 0)):
            continue

        path.append([u[0], u[1]])

        m[(u[0], u[1])] = 1

        if (u[0] == target or u[1] == target):
            isSolvable = True

            if (u[0] == target):
                if (u[1] != 0):
                    path.append([u[0], 0])
            else:
                if (u[0] != 0):

                    # Fill final state
                    path.append([0, u[1]])

            sz = len(path)
            for i in range(sz):
                print("(", path[i][0], ",",
                      path[i][1], ")")
            break

        q.append([u[0], b])  # Fill Jug2
        q.append([a, u[1]])  # Fill Jug1

        for ap in range(max(a, b) + 1):
```

```
            c = u[0] + ap
            d = u[1] - ap
            if (c == a or (d == 0 and d >= 0)):
                q.append([c, d])
            c = u[0] - ap
            d = u[1] + ap
            if ((c == 0 and c >= 0) or d == b):
                q.append([c, d])
        q.append([a, 0])
        q.append([0, b])
    if (not isSolvable):print("No solution")
BFS(4,3,2)
```

Output :-

```
( 0 , 0 )
( 0 , 3 )
( 4 , 0 )
( 4 , 3 )
( 3 , 0 )
( 1 , 3 )
( 3 , 3 )
( 4 , 2 )
( 0 , 2 )
```

Practical 2

Write a program to implement Tic-Tac-Toe Game :

User VS User :-

Code :-

```
from IPython.display import clear_output

def display_board(board):
    clear_output()
    print(board[7]+'|'+board[8]+'|'+board[9])
    print('-'+'|'+'-'+'|'+'-')
    print(board[4]+'|'+board[5]+'|'+board[6])
    print('-'+'|'+'-'+'|'+'-')
    print(board[1]+'|'+board[2]+'|'+board[3])

def player_input():
    marker = ''
    while marker not in ['X','O']:
        marker = input("Player 1 , choose X or O :").upper()
        player_1 = marker
        if marker not in ['X','O']:
            print("Invalid Value")
    if player_1 == 'X':
        player_2 = 'O'
    else :
        player_2 = 'X'
    return (player_1,player_2)

def place_marker(board, marker, position):
    board[position] = marker

def win_check(board, mark):
    return ((board[1]==board[2]==board[3]==mark) or
        (board[4]==board[5]==board[6]==mark) or
        (board[7]==board[8]==board[9]==mark) or
        (board[1]==board[5]==board[9]==mark) or
        (board[3]==board[5]==board[7]==mark) or
        (board[1]==board[4]==board[7]==mark) or
        (board[2]==board[5]==board[8]==mark) or
        (board[3]==board[6]==board[9]==mark))

import random
def choose_first():
    a = random.randint(0,1)
    if a == 0:
        return "Player 1"
    else:
        return "Player 2"
```

```
def space_check(board, position):
    return board[position]==' '

def full_board_check(board):
    for i in [1,2,3,4,5,6,7,8,9]:
        if space_check(board,i):
            return False

    return True

def player_choice(board):
    position = 0
    while position not in [1,2,3,4,5,6,7,8,9] or not space_check(board,position):
        position = int(input("Choose from 1 to 9 only :"))
    return position

def replay():
    game_on = input("Do you want to play again (Y/N):")
    return game_on == 'Y'

print('Welcome to Tic Tac Toe!')

while True:
    the_board = [' '] * 10
    p1,p2 = player_input()
    turn = choose_first()
    print(turn + " gets to play first :) ")

    play = input("Ready to play (Y/N) : ").upper()
    if play=='Y':
        game = True
    else :
        game = False
    while game:
        if turn == "Player 1":
            display_board(the_board)
            position = player_choice(the_board)
            place_marker(the_board,p1,position)

            if win_check(the_board,p1):
                display_board(the_board)
                print("Player 1 has won !!!!")
                game = False
            else :
                if full_board_check(the_board):
                    display_board(the_board)
                    print("TIE !!")
                    break
                else :
                    turn = 'Player 2'
```

```
        # Player2's turn.
        else :
            display_board(the_board)
            position = player_choice(the_board)
            place_marker(the_board,p2,position)

            if win_check(the_board,p2):
                display_board(the_board)
                print("Player 2 has won !!!!")
                game = False
            else :
                if full_board_check(the_board):
                    display_board(the_board)
                    print("TIE !!")
                    break
                else :
                    turn = 'Player 1'
            #pass
    if not replay():
        break
```

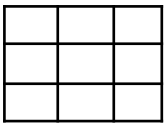
Output :-

Welcome to Tic Tac Toe!

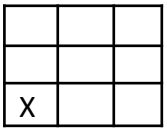
Player 1 , choose X or O : X

Player 1 gets to play first :)

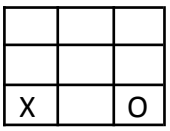
Ready to play (Y/N) : Y



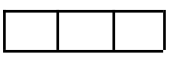
Choose from 1 to 9 only : 1



Choose from 1 to 9 only : 3



Choose from 1 to 9 only : 2



X	X	O

Choose from 1 to 9 only : 6

		O
X	X	O

Choose from 1 to 9 only : 7

X		
		O
X	X	O

Choose from 1 to 9 only : 9

X		O
		O
X	X	O

Player 2 has won !!!!

Machine VS User :

Code :-

```
from copy import deepcopy

class Tic_Tac_Toe:
    def __init__(self, size):
        self.size = size

    def Display_Current_State(self, curr_state):
        print("\n")
        print(curr_state[0][0]+'|'+ curr_state[0][1]+'|'+ curr_state[0][2])
        print(curr_state[1][0]+'|'+ curr_state[1][1]+'|'+ curr_state[1][2])
        print(curr_state[2][0]+'|'+ curr_state[2][1]+'|'+ curr_state[2][2])


    def Success(self, state):
        for i in range(self.size):
            if (state[i][0] == 'X' and state[i][1] == 'X' and state[i][2] == 'X'):
                return True
        for i in range(self.size):
            if (state[0][i] == 'X' and state[1][i] == 'X' and state[2][i] == 'X'):
                return True
        if (state[0][0] == 'X' and state[1][1] == 'X' and state[2][2] == 'X'):
            return True
```

```
    if (state[0][2] == 'X' and state[1][1] == 'X' and state[2][0] == 'X'):
        return True
    return False

def Lose(self, state):
    for i in range(self.size):
        if (state[i][0] == 'O' and state[i][1] == 'O' and state[i][2] == 'O'):
                            return True
    for i in range(self.size):
        if (state[0][i] == 'O' and state[1][i] == 'O' and state[2][i] == 'O'):
            return True
    if (state[0][0] == 'O' and state[1][1] == 'O' and state[2][2] == 'O'):
        return True
    if (state[0][2] == 'O' and state[1][1] == 'O' and state[2][0] == 'O'):
        return True
    return False

     def Draw(self, state):
    for i in range(self.size):
        for j in range(self.size):
            if(state[i][j] == '_'):
                return False
    return True

def Computer_Move(self, state):
    minv = 1
    mini = 0
    index = 0
    for i in range(self.size):
        for j in range(self.size):
            index += 1
            if (state[i][j] == '_'):
                temp_state = deepcopy(state)
                temp_state[i][j] = 'O'
                val = self.Best_Move(temp_state, 'min')
                if (val < minv):
                    minv = val
                    mini = index
    return mini

def Best_Move(self, state, player):
    if(self.Success(state)):
        return 1
    if(self.Lose(state)):
        return -1
    if(self.Draw(state)):
        return 0
    if(player == 'min'):
        maxv = -1
                    for i in range(self.size):
            for j in range(self.size):
```

```
                if(state[i][j] == '_'):
                    temp_state = deepcopy(state)
                    temp_state[i][j] = 'X'
                    val = self.Best_Move(temp_state, 'max')
                    if(val > maxv):
                        maxv = val
            return maxv
        if(player == 'max'):
            minv = 1
            for i in range(self.size):
                for j in range(self.size):
                    if (state[i][j] == '_'):
                        temp_state = deepcopy(state)
                        temp_state[i][j] = 'O'
                        val = self.Best_Move(temp_state, 'min')
                        if (val < minv):
                            minv = val
            return minv

    def Start_Game(self):
        curr_state = [['_','_','_'] for i in range(self.size)]
        self.Display_Current_State(curr_state)
        for i in range(9):
            if(i%2 == 0):
                user_move = int(input("\nUser's turn : "))
                while(curr_state[int((user_move-1)/self.size)][int((user_move-1)%self.size)] != '_'):
                    user_move = int(input("\nEnter a valid move :"))
                curr_state[int((user_move-1)/self.size)][int((user_move-1)%self.size)] = 'X'
                self.Display_Current_State(curr_state)

            else:
                print("\nSystem's turn ...")
                system_move = self.Computer_Move(curr_state)
                curr_state[int((system_move-1)/self.size)][int((system_move-1)%self.size)] = 'O'
                self.Display_Current_State(curr_state)

            if (self.Success(curr_state)):
                print("\nYou win the game !!")
                return
            elif(self.Lose(curr_state)):
                print("\nYou lose the game !!")
                return
            elif(self.Draw(curr_state)):
                print("\nMatch draw !!")
                return

t = Tic_Tac_Toe(3)
t.Start_Game()
```

Output :-

_|_|_
_|_|_
_|_|_

User's turn : 1
X|_|_
_|_|_
_|_|_

System's turn ...
X|_|_
_|O|_
_|_|_

User's turn : 3
X|_|X
_|O|_
_|_|_

System's turn ...
X|O|X
_|O|_
_|_|_

User's turn : 4
X|O|X
X|O|_
_|_|_

System's turn ...
X|O|X
X|O|_
_|O|_
You lose the game !!

Practical 3

Write a program to implement n-queens problem :

Code :-

```
class QueenChessBoard:
    def __init__(self, size):
        self.size = size
        self.columns = []

    def place_in_next_row(self, column):
        self.columns.append(column)

    def remove_in_current_row(self):
        return self.columns.pop()

    def is_this_column_safe_in_next_row(self, column):
        row = len(self.columns)
        for queen_column in self.columns:
            if column == queen_column:
                return False

        for queen_row, queen_column in enumerate(self.columns):
            if queen_column - queen_row == column - row:
                return False
         for queen_row, queen_column in enumerate(self.columns):
            if ((self.size - queen_column) - queen_row
                == (self.size - column) - row):
                return False

        return True

    def display(self):
        for row in range(self.size):
            for column in range(self.size):
                if column == self.columns[row]:
                    print('Q', end=' ')
                else:
                    print('.', end=' ')
            print()


def solve_queen(size):
    board = QueenChessBoard(size)
    number_of_solutions = 0

    row = 0
    column = 0
```

```
    while True:

        while column < size:
            if board.is_this_column_safe_in_next_row(column):
                board.place_in_next_row(column)
                row += 1
                column = 0
                break
            else:
                column += 1
        if (column == size or row == size):
            if row == size:
                board.display()
                print()
                        board.remove_in_current_row()
                row -= 1          try:
                prev_column = board.remove_in_current_row()
            except IndexError:
                break

            row -= 1
            column = 1 + prev_column

    print('Number of solutions:', number_of_solutions)

n = int(input('Enter n: '))
solve_queen(n)
```

Output :-

```
Enter n : 4
. Q . .
. . . Q
Q . . .
. . Q .

. . Q .
Q . . .
. . . Q
. Q . .

Number of solutions : 2
```

Practical 4

Write a program to implement Missionary Cannibal problem :

Code :-

```
print("\n")
print("\tGAME START\nNow the task is to move all of them to right side of the river")
print("Rules:\n1. The boat can carry at most two people\n2. If cannibals num greater than missionaries then the cannibals would eat the missionaries\n3. The boat cannot cross the river by itself with no people on board")
lM = 3
lC = 3
rM=0
rC=0
userM = 0
userC = 0
k = 0
print("\nM M M C C C |   --- | \n")
try:
   while(True):
      while(True):
         print("Left side -> right side river travel")

         uM = int(input("Enter number of Missionaries travel => "))
         uC = int(input("Enter number of Cannibals travel => "))

         if((uM==0)and(uC==0)):
            print("Empty travel not possible")
            print("Re-enter : ")
         elif(((uM+uC) <= 2)and((lM-uM)>=0)and((lC-uC)>=0)):
            break
         else:
            print("Wrong input re-enter : ")
      lM = (lM-uM)
      lC = (lC-uC)
      rM += uM
      rC += uC

      print("\n")
      for i in range(0,lM):
         print("M ",end="")
      for i in range(0,lC):
         print("C ",end="")
      print("| --> | ",end="")
      for i in range(0,rM):
         print("M ",end="")
      for i in range(0,rC):
         print("C ",end="")
      print("\n")
```

```
        k +=1

        if(((lC==3)and (lM == 1))or((lC==3)and(lM==2))or((lC==2)and(lM==1))or((rC==3)and (rM
== 1))or((rC==3)and(rM==2))or((rC==2)and(rM==1))):
            print("Cannibals eat missionaries:\nYou lost the game")

            break

        if((rM+rC) == 6):
            print("You won the game : \n\tCongrats")
            print("Total attempt")
            print(k)
            break
        while(True):
            print("Right side -> Left side river travel")
            userM = int(input("Enter number of Missionaries travel => "))
            userC = int(input("Enter number of Cannibals travel => "))

            if((userM==0)and(userC==0)):
                print("Empty travel not possible")
                print("Re-enter : ")
            elif(((userM+userC) <= 2)and((rM-userM)>=0)and((rC-userC)>=0)):
                break
            else:
                print("Wrong input re-enter : ")
        lM += userM
        lC += userC
        rM -= userM
        rC -= userC

        k +=1
        print("\n")
        for i in range(0,lM):
            print("M ",end="")
        for i in range(0,lC):
            print("C ",end="")
        print("| <-- | ",end="")
        for i in range(0,rM):
            print("M ",end="")
        for i in range(0,rC):
            print("C ",end="")
        print("\n")



        if(((lC==3)and (lM == 1))or((lC==3)and(lM==2))or((lC==2)and(lM==1))or((rC==3)and (rM
== 1))or((rC==3)and(rM==2))or((rC==2)and(rM==1))):
            print("Cannibals eat missionaries:\nYou lost the game")
            break
except EOFError as e:
    print("\nInvalid input please retry !!")
```

Output :-

GAME START
Now the task is to move all of them to right side of the river
Rules:
1. The boat can carry at most two people
2. If cannibals num greater than missionaries then the cannibals would eat the missionaries
3. The boat cannot cross the river by itself with no people on board

M M M C C C | --- |

Left side -> right side river travel
Enter number of Missionaries travel => 0
Enter number of Cannibals travel => 2

M M M C | --> | C C

Right side -> Left side river travel
Enter number of Missionaries travel => 0
Enter number of Cannibals travel => 1

M M M C C | <-- | C

Left side -> right side river travel
Enter number of Missionaries travel => 0
Enter number of Cannibals travel => 2

M M M | --> | C C C

Right side -> Left side river travel
Enter number of Missionaries travel => 0
Enter number of Cannibals travel => 1

M M M C | <-- | C C

Left side -> right side river travel
Enter number of Missionaries travel => 2
Enter number of Cannibals travel => 0

M C | --> | M M C C

Right side -> Left side river travel
Enter number of Missionaries travel => 1
Enter number of Cannibals travel => 1

M M C C | <-- | M C

Left side -> right side river travel
Enter number of Missionaries travel => 2
Enter number of Cannibals travel => 0

C C | --> | M M M C

Right side -> Left side river travel
Enter number of Missionaries travel => 0
Enter number of Cannibals travel => 1

C C C | <-- | M M M

Left side -> right side river travel
Enter number of Missionaries travel => 0
Enter number of Cannibals travel => 2

C | --> | M M M C C

Right side -> Left side river travel
Enter number of Missionaries travel => 0
Enter number of Cannibals travel => 1

C C | <-- | M M M C

Left side -> right side river travel
Enter number of Missionaries travel => 0
Enter number of Cannibals travel => 2

| --> | M M M C C C

You won the game :
    Congrats
Total attempt
11

Practical 5

Write a program to implement 8-puzzle problem using A* algorithm :-

Code :-

```
from queue import PriorityQueue

# Define the goal state
goal_state = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]

# Define the heuristic function
def heuristic(state):
    count = 0
    for i in range(3):
        for j in range(3):
            if state[i][j] != goal_state[i][j]:
                count += 1
    return count

# Define the A* algorithm
def solve_8_puzzle(start_state):
    # Define the moves: up, down, left, right
    moves = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    move_names = ['Up', 'Down', 'Left', 'Right']

    # Define the priority queue for the open set
    open_set = PriorityQueue()
    open_set.put((heuristic(start_state), 0, start_state))
    closed_set = set()

    while not open_set.empty():
        _, g, current_state = open_set.get()
        closed_set.add(tuple(map(tuple, current_state)))

        # Print the current state
        print_state(current_state, g)

        # Check if the current state is the goal state
        if current_state == goal_state:
            return current_state

        # Generate possible next moves
```

```
        zero_pos = next((i, j) for i, row in enumerate(current_state) for j, val in enumerate(row)
if val == 0)
        for move, move_name in zip(moves, move_names):
            new_pos = zero_pos[0] + move[0], zero_pos[1] + move[1]
            if 0 <= new_pos[0] < 3 and 0 <= new_pos[1] < 3:
                new_state = [row.copy() for row in current_state]
                new_state[zero_pos[0]][zero_pos[1]], new_state[new_pos[0]][new_pos[1]] =
new_state[new_pos[0]][new_pos[1]], new_state[zero_pos[0]][zero_pos[1]]
                if tuple(map(tuple, new_state)) not in closed_set:
                    open_set.put((heuristic(new_state) + g + 1, g + 1, new_state))

    # No solution found
    return None

# Function to print the state
def print_state(state, step):
    print(f"Step: {step}")
    for row in state:
        print(row)
    print()

# Define the initial state
initial_state = [[1, 2, 3], [0, 4, 6], [7, 5, 8]]

# Solve the puzzle
result = solve_8_puzzle(initial_state)
if result is None:
    print("No solution found.")
else:
    print("Puzzle solved!")
```

Output :-

```
Step: 0
[1, 2, 3]
[0, 4, 6]
[7, 5, 8]

Step: 1
[1, 2, 3]
[4, 0, 6]
[7, 5, 8]

Step: 2
[1, 2, 3]
[4, 5, 6]
[7, 0, 8]
```

Step: 3
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]

Puzzle solved!