ASSIGNMENT: 01

Name: Abhijeet Biswas
SRN: 201900400
Roll No: 05
Div: B

Question 1.1: Implement non AI Technique for Tic Tac Toe problem.

Code:

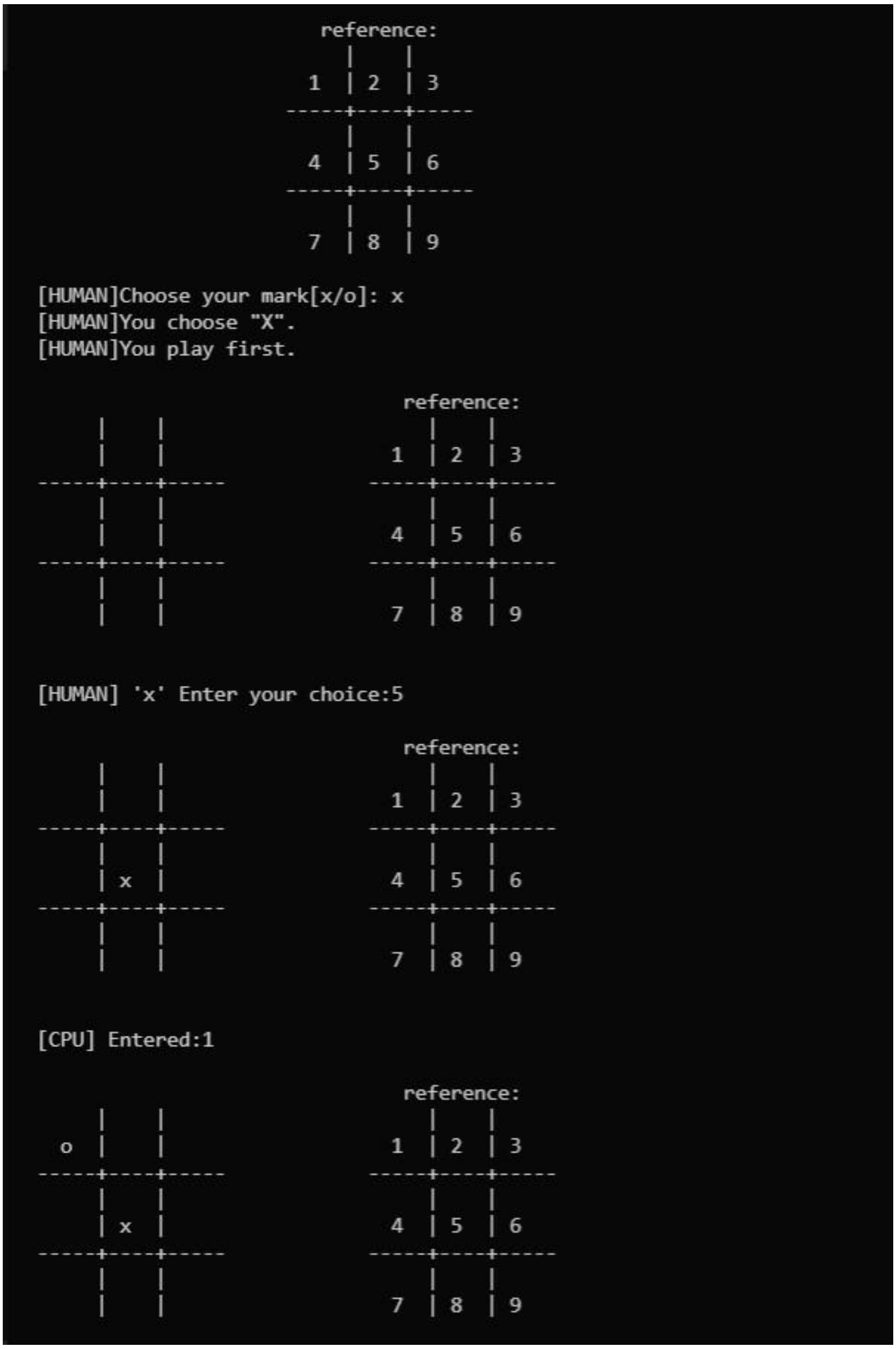
```
print(20 * ' ', "   reference:    ")

print(20 * ' ', '     |     |      ')

print(20 * ' ', '  1  | 2  | 3    ')

print(20 * ' ', "-----+----+----- ")

print(20 * ' ', "     |     |      ")

print(20 * ' ', "  4  | 5  | 6    ")

print(20 * ' ', "-----+----+----- ")

print(20 * ' ', "     |     |      ")

print(20 * ' ', "  7  | 8  | 9    \n")



def display_board():

    print()

    print('                              reference:')

    print('     |     |     ', 10 * ' ', '     |     |   ', )
    print('  ' + board[1] + '  | ' + board[2] + '  | ' + board[3] + '   ', 10 * ' ', '  1  | 2  | 3  ')
    print('-----+----+-----', 10 * ' ', "-----+----+-----")
    print('     |     |     ', 10 * ' ', "     |     |     ")
    print('  ' + board[4] + '  | ' + board[5] + '  | ' + board[6] + '   ', 10 * ' ', "  4  | 5  | 6   ")
    print('-----+----+-----', 10 * ' ', "-----+----+-----")
    print('     |     |     ', 10 * ' ', "     |     |      ")
    print('  ' + board[7] + '  | ' + board[8] + '  | ' + board[9] + '   ', 10 * ' ', "  7  | 8  | 9   \n\n")


def human_input(mark):
    while True:
        inp = input(f"[HUMAN] '{mark}' Enter your choice:")
        if inp.isdigit() and int(inp) < 10 and int(inp) > 0:
            inp = int(inp)
            if board[inp] == " ":
                return inp
            else:
                print(f"[HUMAN] '{mark}' place already taken.")
        else:
            print(f"[HUMAN] '{mark}' Enter valid option (1 - 9).")



def winning(mark, board):
    winning_place = [[1, 2, 3], [4, 5, 6], [7, 8, 9], [1, 4, 7], [2, 5, 8], [3, 6, 9], [1, 5, 9], [3, 5, 7]]
    for win_place in winning_place:
        if board[win_place[0]] == board[win_place[1]] == board[win_place[2]] == mark:
            return True


def win_move(i, board, mark):
    temp_board = list(board)
    temp_board[i] = mark
    if winning(mark, temp_board):
        return True
    else:
        return False


def cpu_input(cpu, human, board):
    for i in range(1, 10):
        if board[i] == ' ' and win_move(i, board, cpu):
            return i
    for i in range(1, 10):
        if board[i] == ' ' and win_move(i, board, human):
            return i
    for i in [5, 1, 7, 3, 2, 9, 8, 6, 4]:
        if board[i] == ' ':
            return i
```

```
def new_game():
    while True:
        nxt = input('[HUMAN] Do you want to play again?(y/n):')
        if nxt in ['y', 'Y']:
            again = True
            break
        elif nxt in ['n', 'N']:
            print('Have a great day')
            again = False
            break
        else:
            print('Enter correct input')
    if again:
        print('___________NEW GAME___________')
        main_game()
    else:
        return False


def win_check(human, cpu):
    winning_place = [[1, 2, 3], [4, 5, 6], [7, 8, 9], [1, 4, 7], [2, 5, 8], [3, 6, 9], [1, 5, 9], [3, 5, 7]]
    for win_place in winning_place:
        if board[win_place[0]] == board[win_place[1]] == board[win_place[2]] == human:
            print('[HUMAN] wins the match!')
            if not new_game():
                return False
        elif board[win_place[0]] == board[win_place[1]] == board[win_place[2]] == cpu:
            print('[CPU] wins the match!')
            if not new_game():
                return False
    if ' ' not in board:
        print('MATCH DRAW!!')
        if not new_game():
            return False
    return True



def user_choice():
    while True:
        inp = input('[HUMAN]Choose your mark[x/o]: ')
        if inp in ['x', 'X']:
            print('[HUMAN]You choose "X".\n[HUMAN]You play first.')
            return 'x', 'o'
        elif inp in ['O', 'o']:
            print('[HUMAN] You choose "O".\n[HUMAN] CPU plays first.')
            return 'o', 'x'
        else:
            print('[HUMAN] Enter correct input!')
```

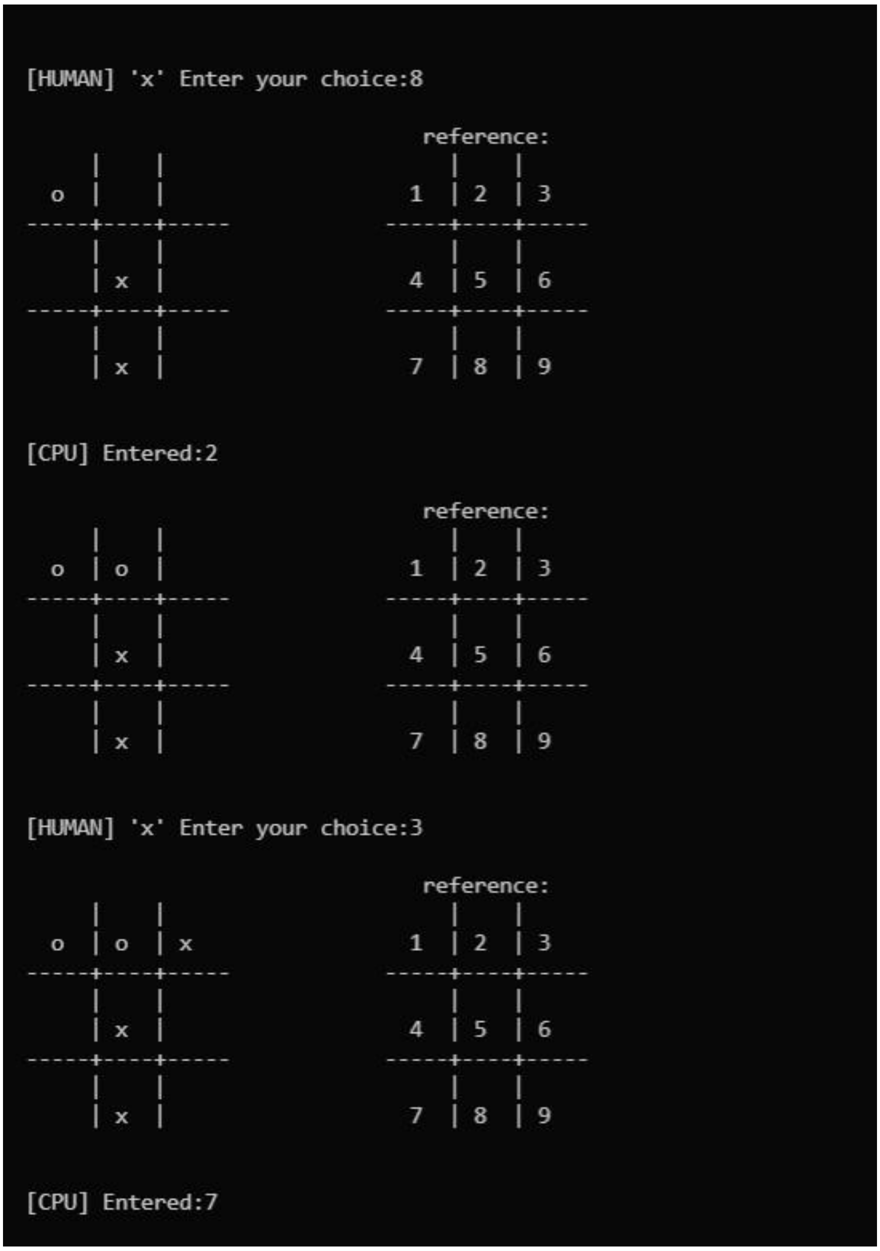
```
def main_game():
    global board
    play = True
    board = ['', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
    human, cpu = user_choice()
    display_board()
    while play:
        if human == 'x':
            x = human_input(human)
            board[x] = human
            display_board()
            play = win_check(human, cpu)
            if play:
                o = cpu_input(cpu, human, board)
                print(f'[CPU] Entered:{o}')
                board[o] = cpu
                display_board()
                play = win_check(human, cpu)
        else:
            x = cpu_input(cpu, human, board)
            print(f'[CPU] Entered:{x}')
            board[x] = cpu
            display_board()
            play = win_check(human, cpu)
            if play:
```

```
        o = human_input(human)

        board[o] = human

        display_board()

        play = win_check(human, cpu)



if __name__ == '__main__':

    main_game()
```

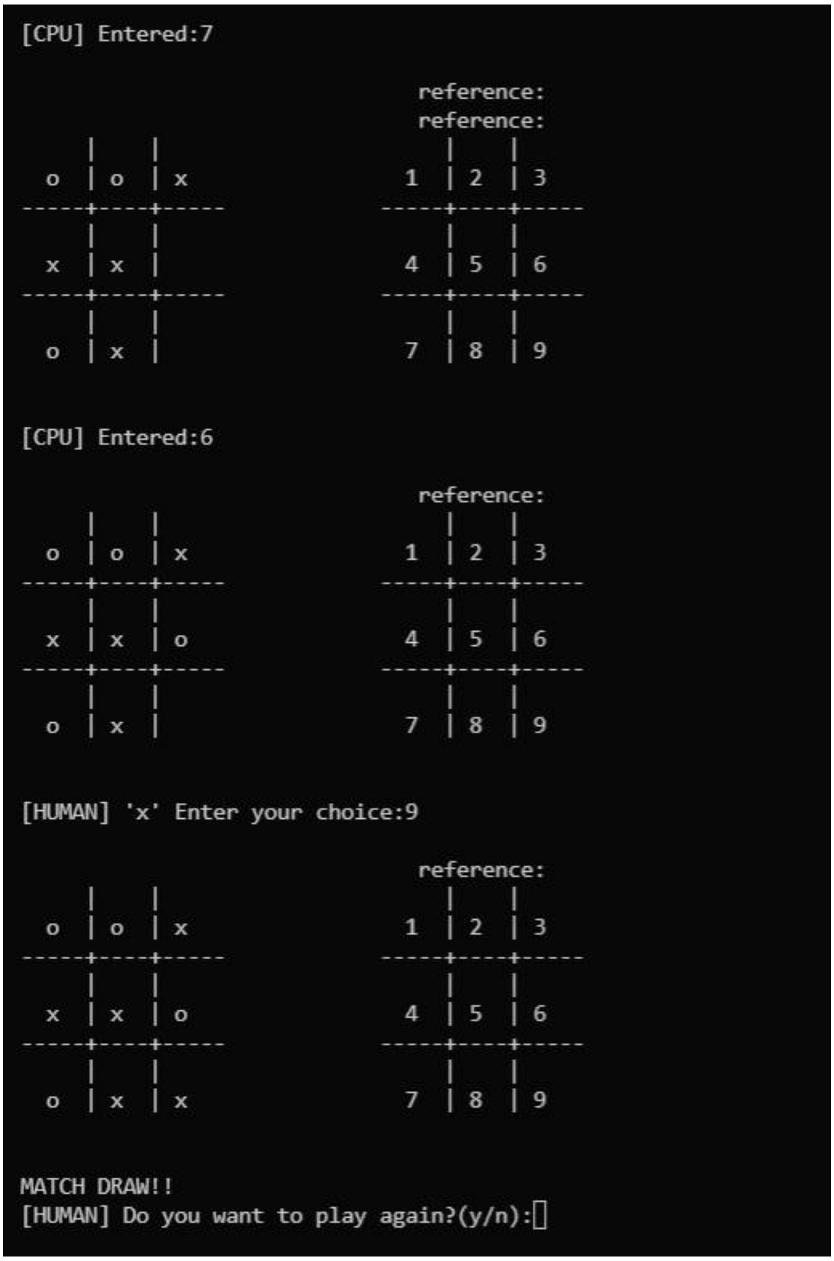
Output:



```
                         reference:
                           |     |
                        1  | 2   | 3
                      -----+-----+-----
                           |     |
                        4  | 5   | 6
                      -----+-----+-----
                           |     |
                        7  | 8   | 9

[HUMAN]Choose your mark[x/o]: x
[HUMAN]You choose "X".
[HUMAN]You play first.

                               reference:
     |     |                     |     |
     |     |                  1  | 2   | 3
-----+-----+-----           -----+-----+-----
     |     |                     |     |
     |     |                  4  | 5   | 6
-----+-----+-----           -----+-----+-----
     |     |                     |     |
     |     |                  7  | 8   | 9


[HUMAN] 'x' Enter your choice:5

                               reference:
     |     |                     |     |
     |     |                  1  | 2   | 3
-----+-----+-----           -----+-----+-----
     |     |                     |     |
     |  x  |                  4  | 5   | 6
-----+-----+-----           -----+-----+-----
     |     |                     |     |
     |     |                  7  | 8   | 9


[CPU] Entered:1

                               reference:
     |     |                     |     |
  o  |     |                  1  | 2   | 3
-----+-----+-----           -----+-----+-----
     |     |                     |     |
     |  x  |                  4  | 5   | 6
-----+-----+-----           -----+-----+-----
     |     |                     |     |
     |     |                  7  | 8   | 9
```



```
[HUMAN] 'x' Enter your choice:8

                               reference:
     |    |                      |    |
  o  |    |                   1  | 2  | 3
-----+----+-----            -----+----+-----
     |    |                      |    |
     | x  |                   4  | 5  | 6
-----+----+-----            -----+----+-----
     |    |                      |    |
     | x  |                   7  | 8  | 9

[CPU] Entered:2

                               reference:
     |    |                      |    |
  o  | o  |                   1  | 2  | 3
-----+----+-----            -----+----+-----
     |    |                      |    |
     | x  |                   4  | 5  | 6
-----+----+-----            -----+----+-----
     |    |                      |    |
     | x  |                   7  | 8  | 9

[HUMAN] 'x' Enter your choice:3

                               reference:
     |    |                      |    |
  o  | o  | x                 1  | 2  | 3
-----+----+-----            -----+----+-----
     |    |                      |    |
     | x  |                   4  | 5  | 6
-----+----+-----            -----+----+-----
     |    |                      |    |
     | x  |                   7  | 8  | 9

[CPU] Entered:7
```



```
[CPU] Entered:7

                              reference:
                              reference:
     |    |                        |    |
  o  | o  | x                  1   | 2  | 3
-----+----+-----           -----+----+-----
     |    |                        |    |
  x  | x  |                    4   | 5  | 6
-----+----+-----           -----+----+-----
     |    |                        |    |
  o  | x  |                    7   | 8  | 9


[CPU] Entered:6

                              reference:
     |    |                        |    |
  o  | o  | x                  1   | 2  | 3
-----+----+-----           -----+----+-----
     |    |                        |    |
  x  | x  | o                  4   | 5  | 6
-----+----+-----           -----+----+-----
     |    |                        |    |
  o  | x  |                    7   | 8  | 9


[HUMAN] 'x' Enter your choice:9

                              reference:
     |    |                        |    |
  o  | o  | x                  1   | 2  | 3
-----+----+-----           -----+----+-----
     |    |                        |    |
  x  | x  | o                  4   | 5  | 6
-----+----+-----           -----+----+-----
     |    |                        |    |
  o  | x  | x                  7   | 8  | 9


MATCH DRAW!!
[HUMAN] Do you want to play again?(y/n):
```

Question 1.2: Implement non AI technique for magic square problem.

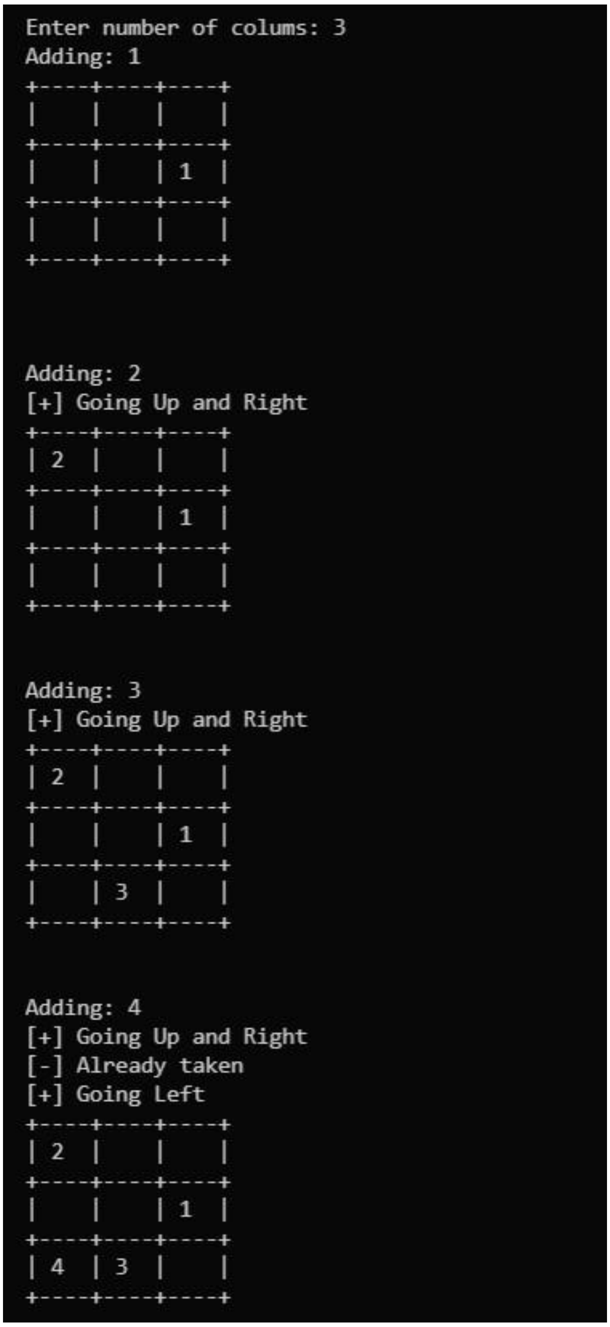
Code:

```
from time import sleep
def table(n,mSquare:dict):
        c=''
        num = 0
        for i in range(n):
                a='|'
                b='+'
                for j in range(n):
```

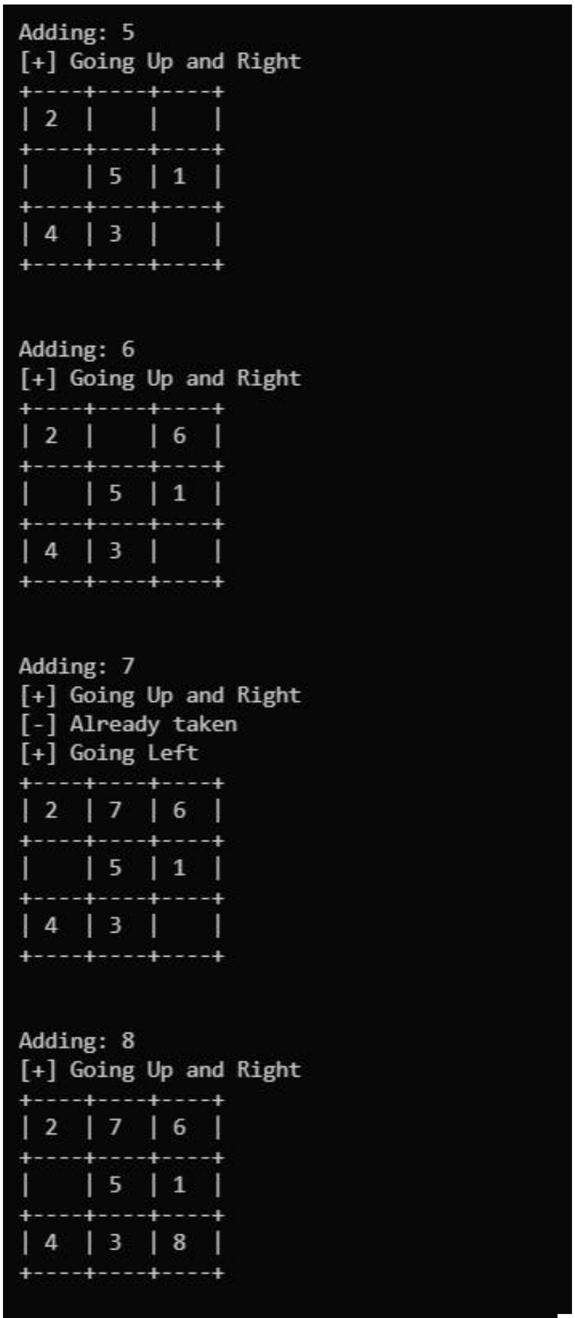
```
            if mSquare.get(i)[j] == 0: var = ''
            else: var = mSquare.get(i)[j]
            a = a+' {:<2} |'.format(var)
            b = b+'----+'
            num+=1
        print(b)
        print(a)
        c=b
    print(c)

n = int(input("Enter number of colums: "))
if n%2==0:
    print("Enter valid length")
    exit(0)

else:
    mSquare = {}
    for i in range(n):
        arr = []
        for j in range(n):
            arr.append(0)
        mSquare.update({i:arr})
    print("Adding: 1")
    x = n-1
    y = int((n-1)/2)
    mSquare.get(y)[x] = 1
    table(n,mSquare)
    print("\n\n")
    sleep(1)
    c=0
    while c<(n*n)-1:
        print(f"Adding: {c+2}")
        x = (x+1)%n
        y = (y-1)%n
        print("[+] Going Up and Right")
        if mSquare.get(y)[x] == 0:
            sleep(3)
            mSquare.get(y)[x] = c+2
            table(n,mSquare)
        else:
```

```
        print("[-] Already taken\n[+] Going Left")
        sleep(3)
        x = (x-2)%n
        y = (y+1)%n
        mSquare.get(y)[x] = c+2
        table(n,mSquare)
    print("\n")
    sleep(1)
    c+=1
```

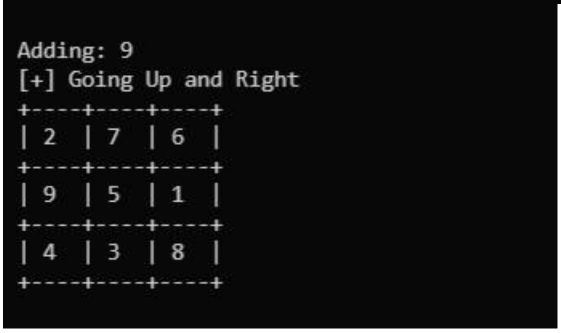
Output:



```
Enter number of colums: 3
Adding: 1
+----+----+----+
|    |    |    |
+----+----+----+
|    |    | 1  |
+----+----+----+
|    |    |    |
+----+----+----+



Adding: 2
[+] Going Up and Right
+----+----+----+
| 2  |    |    |
+----+----+----+
|    |    | 1  |
+----+----+----+
|    |    |    |
+----+----+----+


Adding: 3
[+] Going Up and Right
+----+----+----+
| 2  |    |    |
+----+----+----+
|    |    | 1  |
+----+----+----+
|    | 3  |    |
+----+----+----+


Adding: 4
[+] Going Up and Right
[-] Already taken
[+] Going Left
+----+----+----+
| 2  |    |    |
+----+----+----+
|    |    | 1  |
+----+----+----+
| 4  | 3  |    |
+----+----+----+
```



```
Adding: 5
[+] Going Up and Right
+----+----+----+
| 2  |    |    |
+----+----+----+
|    | 5  | 1  |
+----+----+----+
| 4  | 3  |    |
+----+----+----+


Adding: 6
[+] Going Up and Right
+----+----+----+
| 2  |    | 6  |
+----+----+----+
|    | 5  | 1  |
+----+----+----+
| 4  | 3  |    |
+----+----+----+


Adding: 7
[+] Going Up and Right
[-] Already taken
[+] Going Left
+----+----+----+
| 2  | 7  | 6  |
+----+----+----+
|    | 5  | 1  |
+----+----+----+
| 4  | 3  |    |
+----+----+----+


Adding: 8
[+] Going Up and Right
+----+----+----+
| 2  | 7  | 6  |
+----+----+----+
|    | 5  | 1  |
+----+----+----+
| 4  | 3  | 8  |
+----+----+----+
```



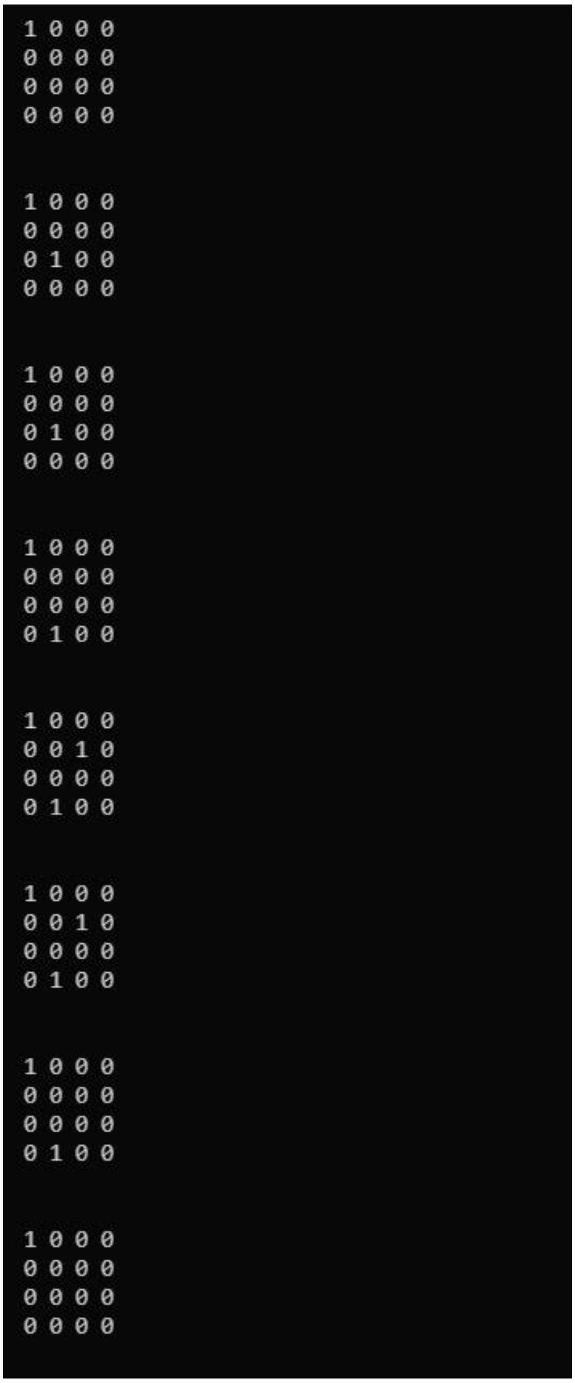
```
Adding: 9
[+] Going Up and Right
+----+----+----+
| 2  | 7  | 6  |
+----+----+----+
| 9  | 5  | 1  |
+----+----+----+
| 4  | 3  | 8  |
+----+----+----+
```

Question 1.3: Implement non AI technique for N Queens problem.

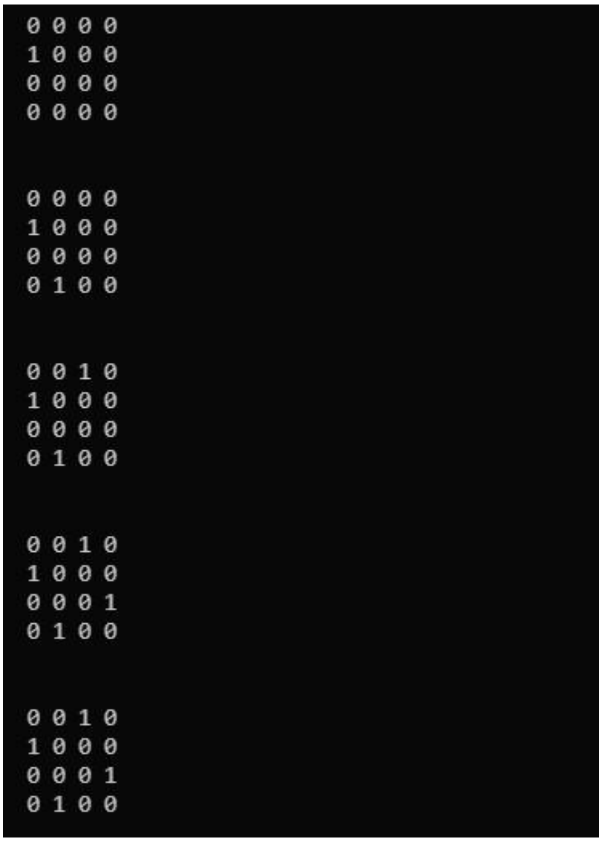
Code:

```
# Python program to solve N Queen
# Problem using backtracking

global N
N = 4

def printSolution(board):
	for i in range(N):
		for j in range(N):
			print (board[i][j],end=' ')
		print()


# A utility function to check if a queen can
# be placed on board[row][col]. Note that this
# function is called when "col" queens are
# already placed in columns from 0 to col -1.
# So we need to check only left side for
# attacking queens
def isSafe(board, row, col):

	# Check this row on left side
	for i in range(col):
		if board[row][i] == 1:
			return False

	# Check upper diagonal on left side
	for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
		if board[i][j] == 1:
			return False

	# Check lower diagonal on left side
	for i, j in zip(range(row, N, 1), range(col, -1, -1)):
		if board[i][j] == 1:
			return False

	return True

def solveNQUtil(board, col):
    # base case: If all queens are placed
    # then return true
    if col >= N:
        return True

    # Consider this column and try placing
    # this queen in all rows one by one
    for i in range(N):

        if isSafe(board, i, col):
            # Place this queen in board[i][col]
            board[i][col] = 1
            printSolution(board)
            print("\n")

            # recur to place rest of the queens
            if solveNQUtil(board, col + 1) == True:
                return True

            # If placing queen in board[i][col
            # doesn't lead to a solution, then
            # queen from board[i][col]
            board[i][col] = 0
    printSolution(board)
    print("\n")

    # if the queen can not be placed in any row in
    # this column col then return false
    return False

# This function solves the N Queen problem using
# Backtracking. It mainly uses solveNQUtil() to
# solve the problem. It returns false if queens
# cannot be placed, otherwise return true and
# placement of queens in the form of 1s.
# note that there may be more than one
# solutions, this function prints one of the
# feasible solutions.
def solveNQ():
    board = [ [0, 0, 0, 0],
                    [0, 0, 0, 0],
                    [0, 0, 0, 0],
                    [0, 0, 0, 0]
                    ]

        if solveNQUtil(board, 0) == False:
                print ("Solution does not exist")
                return False

        printSolution(board)
        print("\n")
        return True

# driver program to test above function
solveNQ()
```

Output:



```
1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0


1 0 0 0
0 0 0 0
0 1 0 0
0 0 0 0


1 0 0 0
0 0 0 0
0 1 0 0
0 0 0 0


1 0 0 0
0 0 0 0
0 0 0 0
0 1 0 0


1 0 0 0
0 0 1 0
0 0 0 0
0 1 0 0


1 0 0 0
0 0 1 0
0 0 0 0
0 1 0 0


1 0 0 0
0 0 0 0
0 0 0 0
0 1 0 0


1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
```



```
0 0 0 0
1 0 0 0
0 0 0 0
0 0 0 0


0 0 0 0
1 0 0 0
0 0 0 0
0 1 0 0


0 0 1 0
1 0 0 0
0 0 0 0
0 1 0 0


0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0


0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
```

Question 1.4: Implement Minimax algorithms for Tic Tac Toe problem.

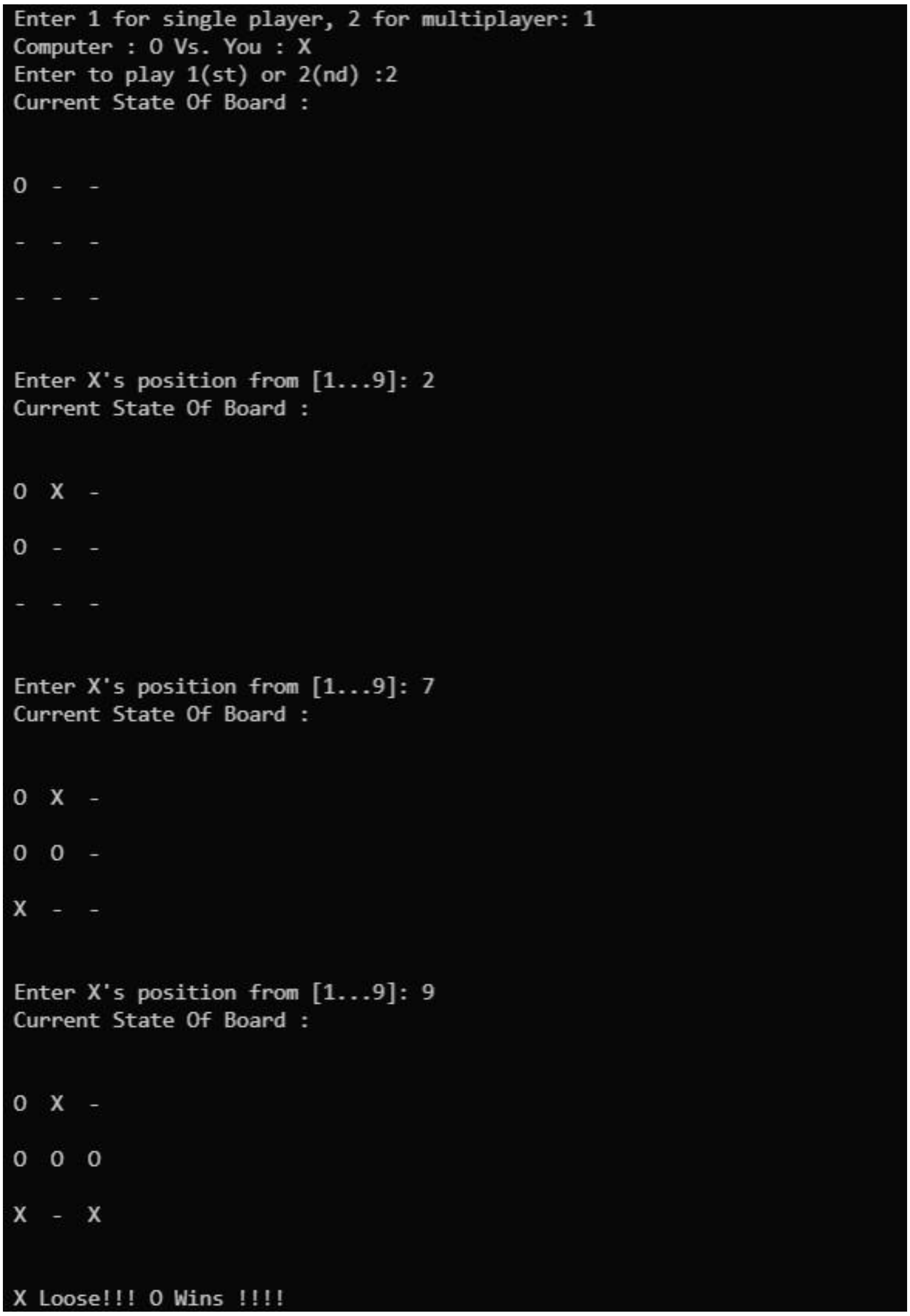
Code:

```
def ConstBoard(board):
    print("Current State Of Board : \n\n");
    for i in range(0, 9):
        if ((i > 0) and (i % 3) == 0):
            print("\n");
        if (board[i] == 0):
            print("- ", end=" ");
        if (board[i] == 1):
            print("O ", end=" ");
        if (board[i] == -1):
            print("X ", end=" ");
    print("\n\n");
```

```
# This function takes the user move as input and make the required changes on the board.
def User1Turn(board):
    pos = input("Enter X's position from [1...9]: ");
    pos = int(pos);
    if (board[pos - 1] != 0):
        print("Wrong Move!!!");
        exit(0);
    board[pos - 1] = -1;


def User2Turn(board):
    pos = input("Enter O's position from [1...9]: ");
    pos = int(pos);
    if (board[pos - 1] != 0):
        print("Wrong Move!!!");
        exit(0);
    board[pos - 1] = 1;


# MinMax function.
def minimax(board, player):
    x = analyzeboard(board);
    if (x != 0):
        return (x * player);
    pos = -1;
    value = -2;
    for i in range(0, 9):
        if (board[i] == 0):
            board[i] = player;
            score = -minimax(board, (player * -1));
            if (score > value):
                value = score;
                pos = i;
            board[i] = 0;

    if (pos == -1):
        return 0;
    return value;
```

```
# This function makes the computer's move using minmax algorithm.
def CompTurn(board):
    pos = -1;
    value = -2;
    for i in range(0, 9):
        if (board[i] == 0):
            board[i] = 1;
            score = -minimax(board, -1);
            board[i] = 0;
            if (score > value):
                value = score;
                pos = i;

    board[pos] = 1;


# This function is used to analyze a game.
def analyzeboard(board):
    cb = [[0, 1, 2], [3, 4, 5], [6, 7, 8], [0, 3, 6], [1, 4, 7], [2, 5, 8], [0, 4, 8], [2, 4, 6]];

    for i in range(0, 8):
        if (board[cb[i][0]] != 0 and
                board[cb[i][0]] == board[cb[i][1]] and
                board[cb[i][0]] == board[cb[i][2]]):
            return board[cb[i][2]];
    return 0;


# Main Function.
def main():
    choice = input("Enter 1 for single player, 2 for multiplayer: ");
    choice = int(choice);
    # The broad is considered in the form of a single dimentional array.
    # One player moves 1 and other move -1.
    board = [0, 0, 0, 0, 0, 0, 0, 0, 0];
    if (choice == 1):
        print("Computer : O Vs. You : X");
        player = input("Enter to play 1(st) or 2(nd) :");
        player = int(player);
        for i in range(0, 9):
            if (analyzeboard(board) != 0):
                break;
            if ((i + player) % 2 == 0):
                CompTurn(board);
            else:
                ConstBoard(board);
                User1Turn(board);
    else:
        for i in range(0, 9):
            if (analyzeboard(board) != 0):
                break;
            if ((i) % 2 == 0):
                ConstBoard(board);
                User1Turn(board);
            else:
                ConstBoard(board);
                User2Turn(board);

    x = analyzeboard(board);
    if (x == 0):
        ConstBoard(board);
        print("Draw!!!")
    if (x == -1):
        ConstBoard(board);
        print("X Wins!!! Y Loose !!!")
    if (x == 1):
        ConstBoard(board);
        print("X Loose!!! O Wins !!!!")


# ---------------#
main()
# ---------------#
```

Output:



```
Enter 1 for single player, 2 for multiplayer: 1
Computer : O Vs. You : X
Enter to play 1(st) or 2(nd) :2
Current State Of Board :


O  -  -

-  -  -

-  -  -


Enter X's position from [1...9]: 2
Current State Of Board :


O  X  -

O  -  -

-  -  -


Enter X's position from [1...9]: 7
Current State Of Board :


O  X  -

O  O  -

X  -  -


Enter X's position from [1...9]: 9
Current State Of Board :


O  X  -

O  O  O

X  -  X


X Loose!!! O Wins !!!!
```