ARTIFICIAL INTELLIGENCE

Name :- Shreyash Tekade PRN No. :- 12110840 Roll No. : 71

Problem Statement :- Implement Tic-Tac-Toe using AI and Non-AI Technique

Code :-

```
import tkinter as tk
import copy
from tkinter import ttk
import numpy as np

class main:
    def __init__(self):
        self.matrix = [0 for i in range(9)]
        self.x_count = 0
        self.y_count = 0
        self.input()
        if (self.isValid()):
            print('The value is ::', self.calculate())
            target = int(input("Enter the target variable::"))
            self.generate_move(target)
            self.calculate_score(target)
            self.view()




    def input(self):
        self.x_ = int(input("Enter the number of X in the current board position::"))
        self.y_ = int(input("Enter the number of O in the current board position::"))
```

```
        for i in range(self.x_):
            pos = int(input("Enter the position for X::"))
            self.matrix[pos] = 1
            self.x_count  = self.x_count + 1


        for i in range(self.y_):
            pos = int(input("Enter the position for O::"))
            if(self.matrix[pos] != 0):
                print('Already occupied by X')
                i = i - 1
                continue
            self.matrix[pos] = 2
            self.y_count = self.y_count + 1
        print('Matrix Representation of the current position::')
        print(self.matrix)


    def isValid(self) -> bool:
        if abs(self.x_count - self.y_count) >= 2:
            print('The board position is not a valid one')
            return False
        # elif (self.matrix[0] == self.matrix[1] and self.matrix[1] == self.matrix[2]) or (self.matrix[3] == self.matrix[4] and self.matrix[4] == self.matrix[5]) or (self.matrix[6] == self.matrix[7] and self.matrix[7] == self.matrix[8]) or (self.matrix[0] == self.matrix[4] and self.matrix[4] == self.matrix[8]) or (self.matrix[2] == self.matrix[4] and self.matrix[4] == self.matrix[6]) or (self.matrix[0] == self.matrix[3] and self.matrix[3] == self.matrix[6]) or (self.matrix[1] == self.matrix[4] and self.matrix[4] == self.matrix[7]) or (self.matrix[2] == self.matrix[5] and self.matrix[5] == self.matrix[8]):
        #     print('The board position is not a valid position')
        #     return False
        else:
            print('This is a valid board position')
            return True
    def calculate(self) -> int:
        arr = self.matrix[::-1]
```

```
    count = 0
    sum = 0
    base = 1
    for i in range(len(arr)):
        sum = sum + arr[i] * base
        count = count + 1
        base *= 3
    return sum

def pow(number, power):
    sum = 0
    for i in range(power):
        sum = sum * number
    return sum



def view(self):
    self.window = tk.Tk()
    self.window.geometry('400x400')
    self.window.rowconfigure((0, 1, 2), weight = 1, uniform = 'a')
    self.window.columnconfigure((0, 1, 2), weight = 1, uniform = 'a')
    count = 0
    temp = ''
    for i in range(3):
        for j in range(3):
            if self.matrix[count] == 1:
                temp = 'x'
            elif self.matrix[count] == 2:
                temp = 'o'
            else:
                temp = ''
```

```
            ttk.Button(text = temp).grid(row = i, column = j, sticky = 'nsew')
            count = count + 1

    self.window.mainloop()

def generate_move(self, target):
    self.move_matrix = []
    i = 0
    m = 0
    print(self.matrix.count(0))
    while i < self.matrix.count(0):
        temp = copy.deepcopy(self.matrix)
        if temp[m] == 0:
            temp[m] = target
            self.move_matrix.append(temp)
            i = i + 1
        m = m + 1

    print(f"All the possible moves for the current matrix position for target {target} is ::")
    print(self.move_matrix)

def calculate_score(self, target):
    self.score = [0 for i in range(self.matrix.count(0))]
    self.reshaped_matrix = []
    for i in range(self.matrix.count(0)):
        self.reshaped_matrix.append(np.array(self.move_matrix[i]).reshape((3, 3)))

    for i in range(len(self.score)):
        for j in range(3):
            if target in self.reshaped_matrix[i][:, j]:
```

```
            self.score[i] += 1
            break

    for i in range(len(self.score)):
        for j in range(3):
            if target in self.reshaped_matrix[i][j]:
                self.score[i] += 1
                break

    self.diagonal_elements = []
    for i in range(len(self.score)):
        temp = []
        for j in range(3):
            temp.append(self.reshaped_matrix[i][j][j])

        self.diagonal_elements.append(temp)

    self.opposite_diagonal_elements = []
    for i in range(len(self.score)):
        self.opposite_diagonal_elements.append(self.reshaped_matrix[i][::-1, ::-1].diagonal())




    # print("The diagonal element matrix", self.diagonal_elements)
    for i in range(len(self.diagonal_elements)):
        if target in self.diagonal_elements[i]:
            self.score[i] += 1
        if target in self.opposite_diagonal_elements[i]:
            self.score[i] += 1
```

```
        print("Scores",self.score)


main()
```

Output :-