In [39]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import sklearn
import seaborn as sns
from sklearn.svm import SVC
from sklearn.metrics import r2_score
```

In [4]:
```
#Cette commande permettra d'importer les données contenues dans le fichier "data.csv"
#en utilisant la fonction read_csv de Pandas.
#La commande df.head(25) permet de visualiser les 25 premières lignes du DataFrame,
df = pd.read_csv("data.csv")
df.head(25)
```

Out[4]:

	P1	P2	P3	P4	P5	P6	P7	P8	P9	MOVE
0	0	0	0	0	0	0	-1	0	0	5
1	0	0	0	-1	1	0	-1	0	0	1
2	1	-1	0	-1	1	0	-1	0	0	9
3	0	0	0	0	0	0	0	0	0	1
4	1	0	0	0	0	0	0	-1	0	3
5	1	0	1	0	-1	0	0	-1	0	2
6	-1	0	0	0	0	0	0	0	0	5
7	-1	0	0	0	1	0	0	0	-1	2
8	-1	1	0	-1	1	0	0	0	-1	8
9	1	0	-1	0	0	0	0	0	0	4
10	1	0	-1	1	0	0	0	-1	0	7
11	1	0	0	0	0	-1	0	0	0	3
12	1	0	1	0	-1	-1	0	0	0	2
13	1	0	0	-1	1	-1	-1	0	0	9
14	-1	1	0	0	1	0	-1	0	-1	8
15	1	-1	0	0	0	0	0	0	0	4
16	1	-1	0	1	0	0	0	-1	0	7
17	1	-1	0	1	0	0	0	0	-1	7
18	-1	0	0	0	1	-1	0	0	0	2
19	-1	1	0	0	1	-1	0	-1	0	7
20	-1	1	-1	0	1	-1	1	-1	0	9
21	0	0	0	0	-1	0	0	0	0	1
22	1	0	0	0	-1	-1	0	0	0	4
23	1	0	-1	1	-1	-1	0	0	0	7
24	0	0	0	0	0	0	0	-1	0	2

In [40]:
```
#La commande x= df.drop('MOVE', axis = 1) permet desélectionner toutes les colonnes
#sauf celle qui a pour nom "MOVE"
# Et de  les stocker dans une variable x
#qui sera utilisée pour les exemples d'entraînement.
x= df.drop('MOVE', axis = 1)

#La commande y=df['MOVE'].values permet de sélectionner la colonne "MOVE"
#et de stocker les valeurs de cette colonne dans une variable y
#qui sera utilisée pour les valeurs désirées.
y=df['MOVE'].values
```

In [41]:
```
# x_train,x_test,y_train,y_test=train_test_split(x, y, test_size=0.05,random_state=0)
# permet de diviser les données en utilisant la fonction train_test_split.
#Le paramètre test_size=0.05 signifie que 5% des données seront pour les données de test,
# et 95% pour les données d'entraînement.
#Le paramètre random_state=0 permet de fixer un état aléatoire
#pour s'assurer que les données sont divisées de la même manière chaque fois que vous exécutez le code.

x_train,x_test,y_train,y_test=train_test_split(x, y, test_size=0.05,random_state=0)
```

In [42]:
```
# importe la classe MLPClassifier de la bibliothèque
#scikit-learn qui permet de créer et d'entraîner un MLP.
from sklearn.neural_network import MLPClassifier
```

In [43]:
```
# MLP: création + entrainement
#  Cette commande crée une instance de la classe MLPClassifier avec des paramètres spécifiques.
#  Le paramètre hidden layer_sizes=(30,20) spécifie que le MLP aura 2 couches cachées
#   avec 30 et 20 neurones respectivement.
#  Le paramètre random_state=1 permet de fixer un état aléatoire pour s'assurer que
#  les poids sont initialisés de la même manière chaque fois que vous exécutez le code.
# Le paramètre max_iter=2014 permet de définir le nombre maximal d'itérations lors de l'entraînement du MLP.
# Cela signifie que l'algorithme d'apprentissage s'arrêtera après 2014 itérations

mlp = MLPClassifier(hidden_layer_sizes=(30,20), random_state=1, max_iter=2014)
```

In [44]:
```
# Cette commande utilise la fonction fit() pour entraîner le MLP sur les données d'entraînement.
# La fonction fit prend en entrée les données d'entraînementpour les exemples (x_train.values)
# et les valeurs désirées (y_train)

mlp.fit(x_train.values, y_train)
```

Out[44]:
```
▾                        MLPClassifier
MLPClassifier(hidden_layer_sizes=(30, 20), max_iter=2014, random_state=1)
```

In [ ]:
```
# Cette commande utilise la fonction predict() pour effectuer des prédictions sur les données de test.
# La fonction predict() prend en entrée les données de test (x_test)
# et retourne les prédictions du modèle (pred) pour ces données.
pred=mlp.predict(x_test)
# Les prédictions sont des valeurs prédites pour les valeurs désirées (y_test)
#correspondant aux exemples de test (x_test).
```

In [45]:
```
# Cette commande crée un DataFrame qui contient les valeurs réelles y_test et les valeurs prédites pred.
# La fonction reshape(-1) est utilisée pour remettre les données sous forme de vecteur 1D.
a=pd.DataFrame({'Real':y_test.reshape(-1),'Predict':pred.reshape(-1)})
# En utilisant la fonction head(100), vous pouvez afficher les premières 100 lignes de ce DataFrame
# qui vous permet de comparer visuellement les valeurs réelles avec les prédictions du modèle.
a.head(100)
```

Out[45]:

	Real	Predict
0	6	6
1	3	3
2	1	2
3	2	2
4	8	8
5	4	5
6	9	9
7	2	2
8	4	8
9	2	2
10	1	1
11	8	8
12	5	5
13	2	2
14	7	7
15	6	6
16	3	3
17	7	7
18	4	4
19	5	2
20	9	9
21	5	5
22	2	2
23	8	8
24	8	8
25	4	4

In [46]:
```
# Cette commande utilise la fonction r2_score() de scikit-learn
# pour calculer la précision (ou l'accuracy) du modèle.

# La fonction r2_score() prend en entrée les valeurs réelles (y_test)
#et les valeurs prédites (pred) et retourne un score compris entre -1 et 1.

print("accuracy",r2_score(y_test,pred))

# Un score proche de 1 indique une performance élevée,
#c'est-à-dire que les prédictions sont proches des valeurs réelles
# tandis qu'un score proche de -1 indique une performance faible
#c'est-à-dire que les prédictions sont éloignées des valeurs réelles.
```

```
accuracy 0.8373870743571925
```

In [20]:
```
# test2  (un seul exemple)
y_pred = mlp.predict([ [1,1,-1,-1,-1,1,1,-1,0] ])
print(y_pred[0])
```

```
9
```

In [10]:
```
# ---------------------------------------------- Application ---------------------------------------------------------

#  La logique du jeu est implémentée à l'aide de plusieurs fonctions, comme :
# •     "next_turn" pour passer au prochain tour
# •     "check_win" pour vérifier si quelqu'un a gagné
# •      "check_empty_spaces" pour vérifier s'il reste des cases vides
# •      "start_new_game" pour redémarrer le jeu.
# •      Il y a aussi une partie de l'IA qui est utilisée pour jouer contre l'homme
#  cette partie utilise le modèle de réseau de neurones pour prédire le meilleur coup à jouer.
```

In [29]:
```
from glob import glob
from tkinter import *
import random
#La bibliothèque utilisée est « Tkinter » pour
#créer une fenêtre avec un tableau de boutons pour représenter les cases du jeu.
```

In [30]:
```
#  fonction pour faire la correspondance entre la position du boutton
# avec  le board qui sera envoyé au model
def position(row,col):
    if row==0 and col==0 : return 0
    elif row==0 and col == 1 : return 1
    elif row==0 and col == 2 : return 2
    elif row==1 and col == 0 : return 3
    elif row==1 and col == 1 : return 4
    elif row==1 and col == 2 : return 5
    elif row==2 and col == 0 : return 6
    elif row==2 and col == 1 : return 7
    elif row==2 and col == 2 : return 8
```

In [31]:
```
# fonction pour faire la correspondance entre la position de la  valeur prise
#par la machine et la position de notre tableau

def positionINV(pos):
    if pos==1  : return  [0,0]
    elif pos==2 : return  [0,1]
    elif pos==3 : return  [0,2]
    elif pos==4 : return  [1,0]
    elif pos==5 : return  [1,1]
    elif pos==6 : return  [1,2]
    elif pos==7 : return  [2,0]
    elif pos==8 : return  [2,1]
    elif pos==9 : return  [2,2]
```

In [32]:
```
# fonction de pour tester et faire tourner le jeu
def next_turn(row, col):
    global player
    global board
    #print(board)
    if game_btns[row][col]['text'] == "" and check_winner() == False:
        if player == players[0]:
            # Put player 1 sympol
            game_btns[row][col]['text'] = player
            board [position(row,col)]= -1
            if check_winner() == False:
                # switch player
                player = players[1]
                label.config(text=(players[1] + " turn"))
                #   prediction of  machine
                predMove = mlp.predict([ board ])
                predMoveCorr=positionINV(predMove[0])
                print(predMoveCorr)
                print(board)
                next_turn(predMoveCorr[0],predMoveCorr[1])

            elif check_winner() == True:
                label.config(text=(players[0] + " wins!"))

            elif check_winner() == 'null':
                label.config(text=(" No Winner!"))

        elif player == players[1]:
            # Put player 2 sympol
            game_btns[row][col]['text'] = player
            board [position(row,col)]= 1
            if check_winner() == False:
                # switch player
                player = players[0]
                label.config(text=(players[0] + " turn"))

            elif check_winner() == True:
                label.config(text=(players[1] + " wins!"))

            elif check_winner() == 'null':
                label.config(text=(" No Winner!"))
```

In [33]:
```
# pour tester s'il y a un gagnant

def check_winner():
    # tester les  3  conditions horizontal
    for row in range(3):
        if game_btns[row][0]['text'] == game_btns[row][1]['text'] == game_btns[row][2]['text'] != "":
            game_btns[row][0].config(bg="cyan")
            game_btns[row][1].config(bg="cyan")
            game_btns[row][2].config(bg="cyan")
            return True

    # tester les  3  conditions vertical
    for col in range(3):
        if game_btns[0][col]['text'] == game_btns[1][col]['text'] == game_btns[2][col]['text'] != "":
            game_btns[0][col].config(bg="cyan")
            game_btns[1][col].config(bg="cyan")
            game_btns[2][col].config(bg="cyan")
            return True

     # tester les  2  conditions restés (les diagonales)
    if game_btns[0][0]['text'] == game_btns[1][1]['text'] == game_btns[2][2]['text'] != "":
        game_btns[0][0].config(bg="cyan")
        game_btns[1][1].config(bg="cyan")
        game_btns[2][2].config(bg="cyan")
        return True
    elif game_btns[0][2]['text'] == game_btns[1][1]['text'] == game_btns[2][0]['text'] != "":
        game_btns[0][2].config(bg="cyan")
        game_btns[1][1].config(bg="cyan")
        game_btns[2][0].config(bg="cyan")
        return True

    # si il ya l'egalité
    if check_empty_spaces() == False:
        for row in range(3):
            for col in range(3):
                game_btns[row][col].config(bg='red')

        return 'null'

    else:
        return False
```

In [34]:
```
# fonction pour tester l'égalité
def check_empty_spaces():
    spaces = 9

    for row in range(3):
        for col in range(3):
            if game_btns[row][col]['text'] != "":
                spaces -= 1

    if spaces == 0:
        return False
    else:
        return True
```

In [35]:
```
# fonction pour commncer un nouveau game
def start_new_game():
    global player
    global board
    player = random.choice(players)
    label.config(text=(player + " turn"))
    board=[0,0,0,0,0,0,0,0,0]
    for row in range(3):
        for col in range(3):
            game_btns[row][col].config(text="", bg="#F0F0F0")
     # tester si la machine qui va jouer la premiere pour appeler le model
    if player==players[1] :
                predMove = mlp.predict([ board ])
                predMoveCorr=positionINV(predMove[0])
                print(predMoveCorr)
                print(board)
                next_turn(predMoveCorr[0],predMoveCorr[1])

board=[0,0,0,0,0,0,0,0,0]
window = Tk()
window.title("Tic-Tac-Toe ")
# on  aura une application de tictactoe HOMME-MACHINE
#dans notre cas H represente l'homme et M represente la machine
players = ["H", "M"]
# on choisit d'une maniere aleatoire qui va jouer en premier
player = random.choice(players)
#notre tableau
game_btns = [
    [0, 0, 0],
    [0, 0, 0],
    [0, 0, 0]
]

label = Label(text=(player + " turn"), font=('consolas', 30))
label.pack(side="top")

restart_btn = Button(text="restart", font=(
    'consolas', 15), command=start_new_game)
restart_btn.pack(side="top")

btns_frame = Frame(window)
btns_frame.pack()


for row in range(3):
    for col in range(3):
        game_btns[row][col] = Button(btns_frame, text="", font=('consolas', 40), width=4, height=1,
                                     command=lambda row=row, col=col: next_turn(row, col))
        game_btns[row][col].grid(row=row, column=col)

# tester si la machine qui va jouer la premiere pour appeler le model
if player==players[1] :
                predMove = mlp.predict([ board ])
                predMoveCorr=positionINV(predMove[0])
                print(predMoveCorr)
                print(board)
                next_turn(predMoveCorr[0],predMoveCorr[1])
```

```
[0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0]
```

In [38]:
```
window.mainloop()
```

In [ ]:

In [ ]:

In [ ]: