In [2]:

```
import tkinter as tk
import pickle
import re
```

0 - Recebe inputs do usuário (formação da letra)

In [64]:

```
class LetraApp:
    def __init__(self):
        self.board = [-1] * 49

        self.window = tk.Tk()
        self.window.title("Simulador de Letra")

        # Define a resolução da janela
        self.window.geometry("600x600")

        # Cria um canvas 6x6
        self.canvas = tk.Canvas(self.window, width=700, height=700)

        # Evita redimensionamento automático
        self.canvas.pack_propagate(False)
        self.canvas.pack()

        # Define o tamanho das colunas e linhas
        for i in range(7):
            self.canvas.grid_columnconfigure(i, weight=1)
            self.canvas.grid_rowconfigure(i, weight=1)

        # Cria botões para cada quadrado
        for i in range(7):
            for j in range(7):
                button = tk.Button(self.canvas, text="", height="100", width="600")
                button.grid(row=i, column=j, sticky="nsew")
                button.bind("<Button-1>", self.on_button_press)
                button.config(bg="white")

        # Cria um rótulo para exibir a letra simulada
        self.label = tk.Label(self.window, text="")
        self.label.pack()

        # Define a cor original do botão
        self.default_bg = "white"
        # Vincula a função 'on_close' ao evento de fechamento da janela
        self.window.protocol("WM_DELETE_WINDOW", self.on_close)

    def on_button_press(self, event):
        # Obtém a posição do botão na grade
        i = event.y // 100
        j = event.x // 100

        # Extrai a parte numérica do nome do widget
        button_id = re.findall(r'\d+', event.widget.winfo_name())


        # Se a lista não estiver vazia, converte o primeiro elemento para inteiro
        if button_id:
            position = int(button_id[0]) -1
        else:
```

```
            position = 0

        print(f"ButtonId:{button_id}, Position: {position}")


        # Atualiza a cor do botão
        if event.widget.cget("bg") == "red":
            event.widget.config(bg=self.default_bg)
        else:
            event.widget.config(bg="red")

        # Atualiza o valor no array
        if event.widget.cget("bg") == "red":
            self.board[position] = 1
        else:
            self.board[position] = -1

        # Atualiza o rótulo com a letra simulada
        self.label.config(text="Letra simulada")

    def on_close(self):
        # Salva o array para um arquivo (Para conseguir rodar o programa, rode este
        #e salve com o nome de arquivo 'letraX'. Execute novamente e crie a letra 'Z
        #Por último, faça o desenho da letra desejada para testar a IA e coloque o n
        with open("teste.pkl", "wb") as f:
            pickle.dump(self.board, f)

        # Fecha a janela
        self.window.destroy()


app = LetraApp()
app.window.mainloop()
```

```
ButtonId:[], Position: 0
ButtonId:['9'], Position: 8
ButtonId:['33'], Position: 32
ButtonId:['41'], Position: 40
ButtonId:['49'], Position: 48
ButtonId:['43'], Position: 42
ButtonId:['37'], Position: 36
ButtonId:['13'], Position: 12
```

2 - Encontrando os pesos (treinamento)

In [66]:

```
# Carregando o arquivo contendo os valores de cada posição da matriz do botões
with open("letraX.pkl", "rb") as letterInput:
    inputX = pickle.load(letterInput)
with open("letraZ.pkl", "rb") as letterInput:
    inputZ = pickle.load(letterInput)
#print(inputX)
#print(inputZ)

matrizEntrada = [inputX, inputZ]
print(matrizEntrada)

# Saida esperada -> 1 para X e -1 para Z
saidaEsperada = [1 , -1]
w = [0] * 49
deltaW = [0] * 49
b = 0
deltaB = 0
```

```
for linha in range(2):
    for coluna in range(49):
        deltaW[coluna] = matrizEntrada[linha][coluna] * saidaEsperada[linha]
        #print(f"linha: {linha}, coluna : {coluna}, deltaW: {deltaW[coluna]}, matriz
        w[coluna] = w[coluna] + deltaW[coluna]

    deltaB = saidaEsperada[linha]
    b += deltaB


print(w)
```

```
[[1, -1, -1, -1, -1, -1, -1, -1, 1, -1, -1, -1, 1, -1, -1, -1, 1, -1, 1, -1, -1, -1,
-1, -1, 1, -1, -1, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, -1, 1, -1, -
1, -1, -1, -1, 1], [1, 1, -1, 1, 1, 1, 1, 1, -1, -1, -1, -1, 1, -1, -1, -1, -1, -1, -
1, -1, -1, -1, -1, -1, 1, -1, -1, -1, -1, -1, 1, -1, -1, -1, -1, -1, 1, -1, -1, -1, -
1, 1, 1, -1, 1, 1, 1, -1, 1]]
[0, -2, 0, -2, -2, -2, -2, -2, 2, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -2, 0, 0, 0, 2, -2, 0, 0, -2, -2, -2, 0, 0]
```

3 - Teste o resultado (rode o bloco 01 novamente)

Obs.: Ao executar o bloco 01 novamente, salve o arquivo da matriz com o nome "teste"

In [65]:

```
with open("teste.pkl", "rb") as letterInput:
    letraTeste = pickle.load(letterInput)

print(letraTeste)
deltaTeste = 0;
for coluna in range(len(letraTeste)):
    deltaTeste += letraTeste[coluna] * w[coluna]
    #print(f"coluna : {coluna}, letraTeste : {letraTeste[coluna]}, w : {w[coluna]} ,
deltaTeste += b
print(deltaTeste)

if(deltaTeste > 0):
    print('Resultado -> Se aproxima da letra "X"')
else:
    print('Resultado -> Se aproxima da letra "Z"')
```

```
[1, -1, -1, -1, -1, -1, -1, -1, 1, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, -1, -1, -1, 1, -1, -1, -1, 1, -1, 1, -1, -
1, -1, -1, -1, 1]
18
Resultado -> Se aproxima da letra "X"
```