Mise en place d'un outil de suivi d'imagerie clinique

Notice du code

Auteur : Kenza Benkirane

Tuteur : Stan Durand

Table des matières

Table des figures

I. Plan du code

1. Importations et modules

```
from tkinter import * ; import tkinter as tk

from PIL import ImageTk as itk; from PIL import Image

import subprocess; import os ;

import cv2

import numpy as np

import pandas as pd;from pandastable import Table,TableModel;
```

- Le module Tkinter sert pour *l'Interface graphique.* Utiliser le diminutif « tk » permet d'éviter les erreurs mineures et de reconnaître les modules Tkinter. Une aide python est disponible ici.
- Le module PIL (pillow) permet d'importer des images et de les inclure dans des canvas Tkinter, d'où le itk : ImageTk
- Les modules subprocess et os permettent de rechercher des fichiers dans un dossier
- CV2 correspond à Open CV qui sert pour le traitement d'image. Il n'a pas été utilisé dans la suite du code mais peut s'avérer utile pour une modification postérieure.
- Le module numpy est un module fondamental de python qui est souvent nécessaire pour exécuter d'autres modules. Il sert principalement pour la manipulation de matrices ou tableaux.
- Les modules pandas et pandastable servent à la création de bases de données à partir du traitement d'image.

2. Variables et options

```
TITLE_FONT = ("Arial", 25, "bold")

violet_bg='#ccb6e4'

white_bg='#FFFFFF'
```

```
cross_color=(255,255,255)

V=['C1', 'C2', 'C3', 'C4', 'C5', 'C6',
'C7','T1','T2','T3','T4','T5','T6','T7','T8','T9''T10','T11','T12','L1','L2','L3','L4','L5']

Titre_Application="X-Rays Measurements Tracking Platform"

from numpy import *

global radio

radio=None
```

Ces variables sont utilisées à plusieurs reprises pour l'esthétique de l'application et

3. Directories à modifier

```
20  #Directories --> A MODIFIER !!!!
21  icone='/Users/KenzaBenkirane/Desktop/Skairos/Skairos_logo.ico'
22  logo='/Users/KenzaBenkirane/Desktop/Skairos/Skairos_logo.jpg'
23  radios_directory='/Users/KenzaBenkirane/Desktop/Skairos/Radios'
```

Figure 1 : Directories pour le logo

Ces directories permettent de donner un logo à l'application, mais aussi de mettre le logo de Skairos sur l'ensemble des pages de l'application. Lors de la transformation de l'application en fichier exécutable, ce logo sera intégré directement dans le dossier 'read me' et n'aura pas besoin d'être stocké à l'extérieur

II. Interface graphique

1. Pourquoi Tkinter ?

Le plugin choisit pour l'interface graphique est Tkinter, pour les raisons suivantes :

- Simplicité d'utilisation : il a déjà été utilisé et pourra être réutilisé assez rapidement afin de pouvoir gagner du temps sur les étapes supérieures
- Disponible en open source
- Très souvent utilisé donc il existe énormément de ressources et références sur internet qui peuvent être utiles en cas de doute

2. Créer une page : le constructeur

```
class Application(object):

    """Défini l'objet application qui est notre application de suivi d'imageries cliniques"""




    def __init__ (self):

        """Construteur de la fenêtre principale"""



        #Paramètres de la fenêtre

        self.fen=Tk()

        global frame; frame=self.fen

        self.fen.configure(background='white')

        self.fen.title(Titre_Application)

        self.fen.iconbitmap(icone)

        self.fen.minsize(1000,400)
```

```
#Affichage du logo sur toutes les pages

        global image;global photo

        image = Image.open(logo)

        image = image.resize((80, 50), Image.ANTIALIAS)

        photo = itk.PhotoImage(image)

        label = Label(frame,image=photo,bg=white_bg)
```

```
label.image = photo # keep a reference!

label.pack(side='bottom', fill='x')



self.fen.iconphoto(False,photo)

self.fen.iconbitmap(icone)
```

La classe Application permet de créer l'objet Application. A celui ci est toujours associé le constucteur, la fonctio__init__, dans laquelle on définit les éléments de base de notre objet :

Il s'agit d'un GUI Tkinter : `self.fen =Tk()`

- `frame=self.fen` : ce nom est utilisé par soucis de simplification
- On utilise le titre de l'application défini précédemment
- On lui donne un icône
  On donne une taille minimale à la fenêtre : `self.fen.minsine(longueur,largeur)`
- Donner un icône à l'application self.fen.iconephoto(False,file='yourfile.png'), en utilisant la fonction PhotoImage

On pourrait aussi :

- Lui donner une taille maximale : self.fen.maxsize(1000,500)

```
#Initialisation des frames

global f; f = tk.Frame(self.fen,bg=white_bg) #frame pour la base de données

global f2;f2 = tk.Frame(self.fen,bg=white_bg) #frame pour les boutons

f.pack(fill='y',expand=0,side='right')

f2.pack(fill='y',expand=0,side='left')
```

Les deux frames f et f2 serviront à « porter » les boutons de contrôle ainsi que les tableaux de coordonnées. Ici expand = 0 pour qu'elles ne s'affichent que si un bouton

y est « packed », ce qui n'a lieu que lorsque l'on choisit une radio, soit après avoir appuyé sur le bouton « Parcourir », donc après avoir activé la fonction « openradio ».

La fonction global permet de les utiliser hors de la fonction __init__, c'est à dire de faire de ces frames des variables globales.

```
    tk.Label(frame, text="\n Plateforme de suivi d'imageries cliniques", font=("Arial", 20,
"bold"),bg=white_bg).pack(fill='x')

    global Description_lbl; Description_lbl=tk.Label(frame, text="Description \n", font=("Arial", 12,
"italic"),bg=white_bg); Description_lbl.pack(expand='yes')



    self.drawing_tool = "line" #initalisation du drawing_tool : paramètre par défaut : ligne

    global Start_bttn; Start_bttn= tk.Button(frame,text="Commencer",command=lambda:start(self),
bg=white_bg)

    Start_bttn.pack(expand='yes')
```

Figure 2 : Page d'accueil de l'application

Les fonctions Label permettent d'afficher du text, \n permet de sauter du ligne. Nommer les label permet de les supprimer ensuite : Le label « Description_lbl » sera supprimé lorsque l'on appuie sur le bouton Parcourir , à l'aide de la méthode Description_lbl.pack_forget()

Ici on initialise la variable self.drawing_tool en lui associant la valeur 'line' comme paramètre de défaut : dès qu'on ouvrira l'image, on pourra lui déssiner des lignes dessus

Le bouton start Permet d'acceder à la page de déscription des étapes. Il affichera le bouton parcourir et tous les textes en conséquence

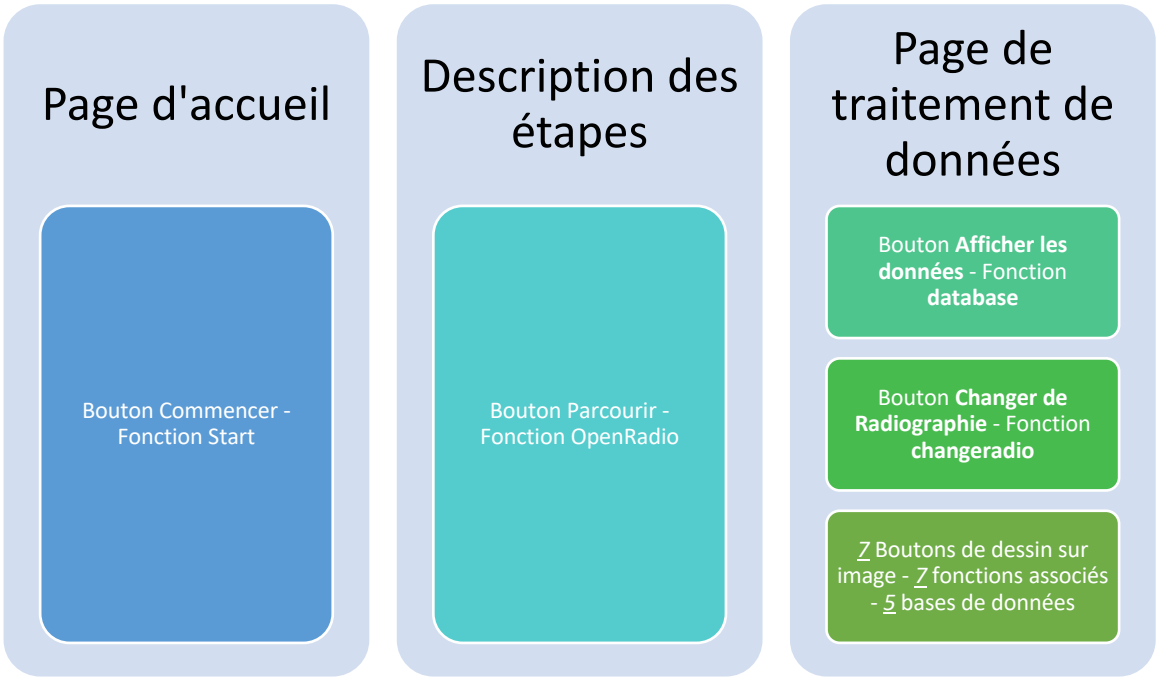

Figure 3 : Architecture des boutons et fonctions associées

III. Fonctions

1. Openradio : Page de saisi d'informations sur image

Cette fonction s'exécute lorsque l'on appuie sur le bouton Parcourir sur la deuxième page.

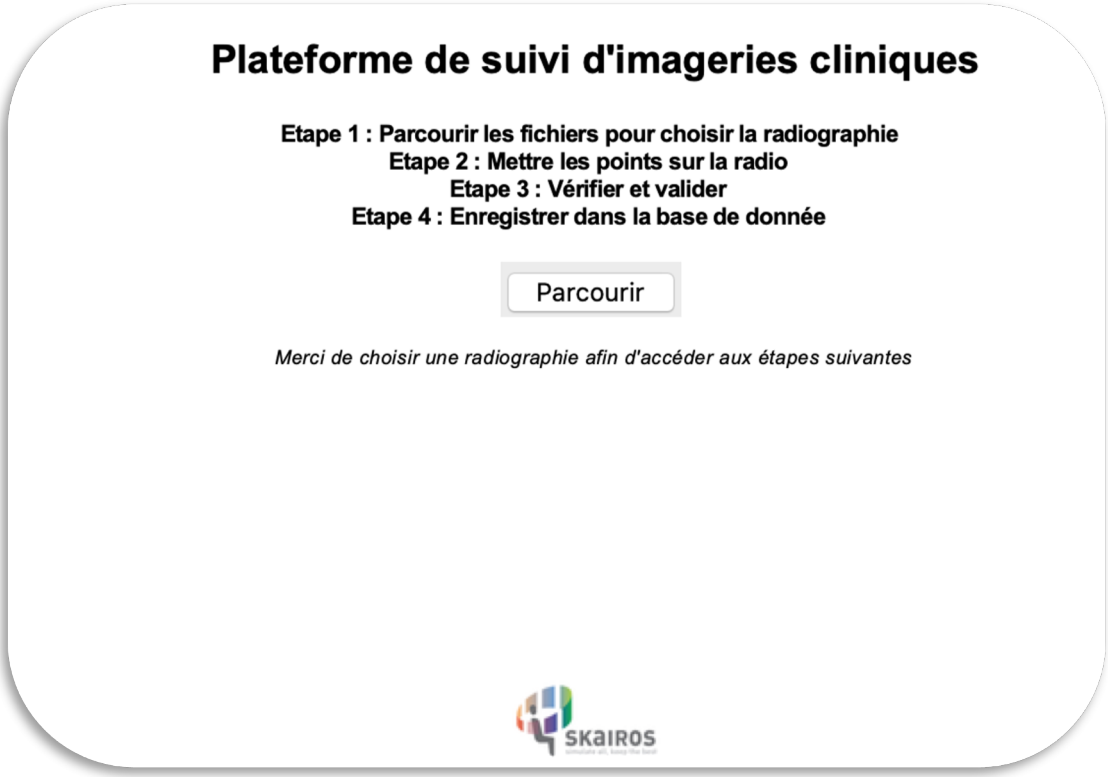

Figure 4 : Deuxième Page de l'application

```
global openradio

    def openradio(self):

        """S'execute lorsque l'on appuie sur le bouton "Parcourir" : affiche l'image, créer les boutons pour choisir les formes à créer et la base de données à afficher """

        #Visuel de la page modifié

        Choisir_Label.pack_forget(); etapes.pack_forget(); #On supprime l'explication des étapes

        Parcourir['text'] = 'Changer de radiographie' # "Parcourir" devient "Changer de radiographie" mais conserve la même fonction

        global Database_bttn

        Database_bttn=Button(frame,text='Afficher le tableau des données',bg=white_bg,command=lambda: database(self))

        Database_bttn.pack(); basededonnee=1;
```

```
    ##### Ouvrir la radio et l'afficher

    radio_filename = tk.filedialog.askopenfilename(initialdir=radios_directory, title="Select A File", filetypes=(("jpg files", "*.jpg"),("all files", "*.*")))
```

La ligne précédente va générer une fenêtre pour ouvrir un fichier

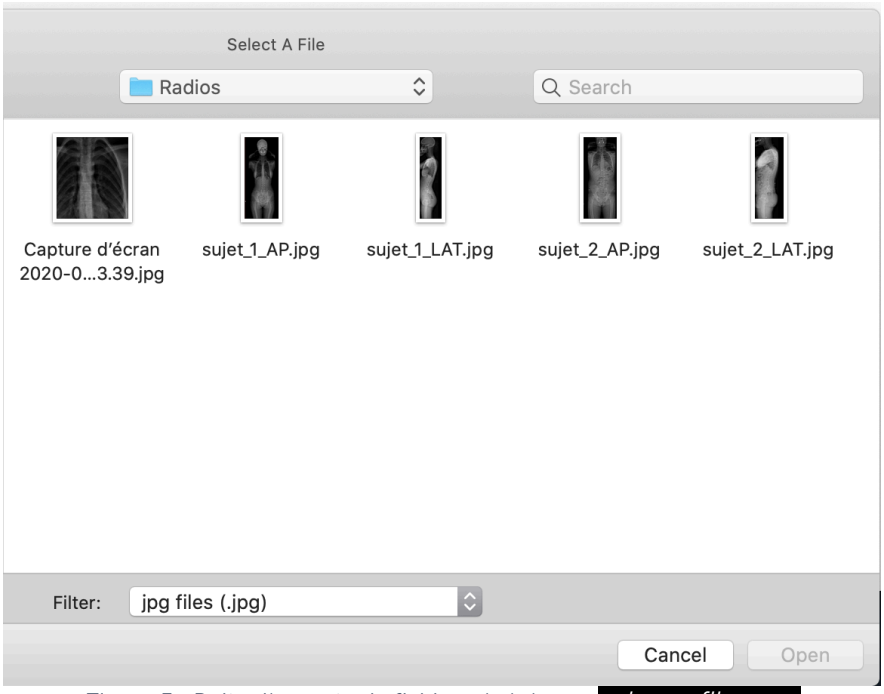

Figure 5 : Boite d'ouverte de fichier générée par askopenfilename

```
    radio_img=cv2.imread(radio_filename)

    radio_img_tk= Image.fromarray(radio_img) # Convert the Image object into a TkPhoto object

    radio = itk.PhotoImage(image=radio_img_tk)
```

Les lignes précédentes permettent d'afficher l'image choisie dans le canva créée, intitulé drawing_area.

```
    global drawing_area

    drawing_area = tk.Canvas(frame, width=radio.width(), height=radio.height(),bg='black') #à améliorer
```

```
        global Image_to_canva;

        Image_to_canva=drawing_area.create_image(0, 0,image=radio,anchor="nw") #Associate the
image to the canva

        drawing_area.bind("<Motion>", self.motion)

        drawing_area.bind("<ButtonPress-1>", self.left_but_down)

        drawing_area.bind("<ButtonRelease-1>", self.left_but_up)
```

Les trois lignes précédentes permettent d'associer :

- Le mouvement de la souris à la fonction `self.motion` → <Motion> : voir lien
- Le fait d'appuyer sur la souris à la fonction `self.left_but_down`
- Le fait de lâcher la souris à la fonction `self.left_but_up`

```
        drawing_area.pack(side='top',fill="y")

        #Ajout des boutons de contrôle pour choisir la forme à dessiner

        global DTool; DTool=tk.Label(f2, text=" \n L'outil de dessin est :
\n"+str(self.drawing_tool)+"\n", font=("Arial", 15),bg=white_bg)

        DTool.pack(side="top")

        Button(f2, text="Lignes",font=("Arial", 12),bg=white_bg,command=lambda: setline(self)).pack()

        Button(f2, text="Arc",font=("Arial", 12),bg=white_bg,command=lambda: setarc(self)).pack()

        Button(f2, text="Oval",font=("Arial", 12),bg=white_bg,command=lambda: setoval(self)).pack()

        Button(f2, text="Rectangle",font=("Arial", 12),bg=white_bg,command=lambda:
setrectangle(self)).pack()

        Button(f2, text="Point",font=("Arial", 12),bg=white_bg,command=lambda: setpoint(self)).pack()

        Label(f2, text="",bg=white_bg).pack()

        Button(f2, text="Crayon",font=("Arial", 12),bg=white_bg,command=lambda:
setpencil(self)).pack()

        Button(f2, text="Legender les vertèbres",font=("Arial", 12),bg=white_bg,command=lambda:
settext(self)).pack()
```

Cette partie sert à afficher les boutons de contrôle des fonctions associés aux differentes formes, et donc d'obtenir l'affichage suivant :

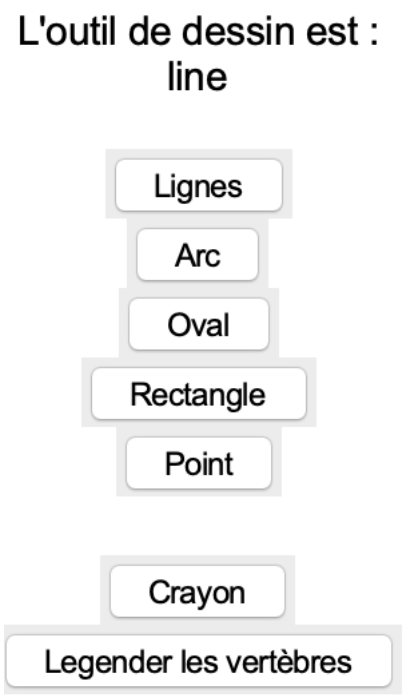

Figure 6 : Affichage dans la frame 2 f2 en Page de traitement d'images

```
#On initalise les base de données

global coordo_pointeur; coordo_pointeur=[] #Coordonnées des points

global coordo_ligne; coordo_ligne=[] #Coordonnées des lignes

global coordo_oval; coordo_oval=[] #Coordonnées des cercles : centre du cercle pour l'instant

global coordo_rectangle; coordo_rectangle=[] #Coordonnées des rectangles

global coordo_arc; coordo_arc=[] #Coordonnées des arc
```

Permet d'initaliser les listes. Celles-ci sont le fondement des bases de données qui constitueront les tableaux qui seront affichés dans l'application.

2. Fonctions d'insertion d'informations sur image

```
global setline

def setline(self):

    self.drawing_tool = "line"



global setarc

def setarc(self):

    self.drawing_tool = "arc"



global setoval

def setoval(self):

    self.drawing_tool = "oval"



global setrectangle

def setrectangle(self):

    self.drawing_tool = "rectangle"



global settext

def settext(self):

    self.drawing_tool = "text"
```

```
    global setpoint

    def setpoint(self):

        self.drawing_tool="point"



    global setpencil

    def setpencil(self):

        self.drawing_tool="pencil"
```

Ces fonctions sont associées à tous les boutons de la figure : ils permettent de choisir la figure à dessiner. Ils sont donc associée à la variable self.drawing_tool qui change de valeur, qui associé à la fonction left_but_up suivante :

```
    def left_but_up(self, event=None):

    #Méthode s'executant lorsque l'on lache la souris, qui permet de prendre les coordonnées du point où on lache la souris.



        self.left_but = "up"



        # Reset the line

        self.x_pos = None

        self.y_pos = None



        # Set x & y when mouse is released
```

```
        self.x2_line_pt = event.x
        self.y2_line_pt = event.y


        # If mouse is released and line tool is selected
        # draw the line
        if self.drawing_tool == "line":
            self.line_draw(event)
        elif self.drawing_tool == "arc":
            self.arc_draw(event)
        elif self.drawing_tool == "oval":
            self.oval_draw(event)
        elif self.drawing_tool == "rectangle":
            self.rectangle_draw(event)
        elif self.drawing_tool == "text":
            self.text_draw(event)
        elif self.drawing_tool=="point":
            self.pointeur(event)
```

Elle prend pour argument event : le cas où où lache la souris, et activera une des fonctions suivantes :

```
# ---------- DRAW PENCIL ----------


    def pencil_draw(self, event=None):
```

```
        if self.left_but == "down":



            # Make sure x and y have a value

            if self.x_pos is not None and self.y_pos is not None:

                event.widget.create_line(self.x_pos, self.y_pos,  event.x, event.y,
smooth=TRUE,fill="white")



            self.x_pos = event.x

            self.y_pos = event.y



    # ---------- DRAW LINE ----------



    def line_draw(self, event=None):



        # Shortcut way to check if none of these values contain None

        if None not in (self.x1_line_pt, self.y1_line_pt, self.x2_line_pt, self.y2_line_pt):

            event.widget.create_line(self.x1_line_pt, self.y1_line_pt, self.x2_line_pt, self.y2_line_pt,
smooth=TRUE, fill="red")



            #Database Lignes

            coordo_ligne.append([self.x1_line_pt,self.y1_line_pt, self.x2_line_pt, self.y2_line_pt])

            self.df_lines=pd.DataFrame(coordo_ligne,columns=['x1','y1','x2','y2'])

            self.df=self.df_lines
```

```
# ---------- DRAW ARC ----------

def arc_draw(self, event=None):

    # Shortcut way to check if none of these values contain None
    if None not in (self.x1_line_pt, self.y1_line_pt, self.x2_line_pt,self.y2_line_pt):

        coords = self.x1_line_pt, self.y1_line_pt, self.x2_line_pt,self.y2_line_pt

        # start : starting angle for the slice in degrees
        # extent : width of the slice in degrees
        # fill : fill color if needed
        # style : can be ARC, PIESLICE, or CHORD
        event.widget.create_arc(coords, start=0, extent=150,outline='white',
                                style=ARC)

        #Database Arc
        coordo_arc.append([self.x1_line_pt, self.y1_line_pt,self.x2_line_pt, self.y2_line_pt])
        self.df_arc=pd.DataFrame(coordo_arc,columns=['x1','y1','x2','y2'])
        self.df=self.df_arc
```

```
    # ---------- DRAW OVAL ----------


    def oval_draw(self, event=None):

        if None not in (self.x1_line_pt, self.y1_line_pt, self.x2_line_pt, self.y2_line_pt):



            # fill : Color option names are here http://wiki.tcl.tk/37701

            # outline : border color

            # width : width of border in pixels



            event.widget.create_oval(self.x1_line_pt, self.y1_line_pt,
self.x2_line_pt, self.y2_line_pt,

                                     fill="blue",

                                     outline="white",

                                     width=2)



            #Database Cercle

            coordo_oval.append([self.x1_line_pt, self.y1_line_pt])

            self.df_oval=pd.DataFrame(coordo_oval,columns=['x','y'])

            self.df=self.df_oval



    # ---------- DRAW RECTANGLE ----------
```

```
    def rectangle_draw(self, event=None):

        if None not in (self.x1_line_pt, self.y1_line_pt, self.x2_line_pt,self.y2_line_pt):

            event.widget.create_rectangle(self.x1_line_pt, self.y1_line_pt, self.x2_line_pt,
self.y2_line_pt,

                outline="white",

                width=2)



            #Database Rectangle

            coordo_rectangle.append([self.x1_line_pt, self.y1_line_pt,self.x2_line_pt, self.y2_line_pt])

            self.df_rectangle=pd.DataFrame(coordo_rectangle,columns=['x1','y1','x2','y2'])

            self.df=self.df_rectangle



    # ---------- DRAW TEXT ----------



    def text_draw(self, event=None):

        k=0

        if None not in (self.x1_line_pt, self.y1_line_pt):



            event.widget.create_text(self.x1_line_pt, self.y1_line_pt,

                                     fill="white",

                                     font=("Arial", 10),

                                     text=V[self.k])

            self.k+=1
```

```
    # ---------- DRAW POINTS ----------



    def pointeur(self,event=None):



        event.widget.create_oval(self.x1_line_pt, self.y1_line_pt,
self.x2_line_pt, self.y2_line_pt,

                                 fill="white",

                                 outline="white",

                                 width=1)



        #Database Pointeur

        coordo_pointeur.append([self.x1_line_pt, self.y1_line_pt]) #ajoute les coordonnées des points à
la liste

        self.df_points=pd.DataFrame(coordo_pointeur,columns=['x','y'])

        self.df=self.df_points
```

A la fin de chaque base de données, deux lignes permettent d'ajouter les nouveaux widget déssinés à la liste correspondante, puis à la dataframe qui sera affichée grace à la fonction database.

Les paramètres des formes sont modifiables :

- Pour la couleur, il suffit de choisir une autre couleur à la place de white : toutes les couleurs possibles sont disponibles ici : http://wiki.tcl.tk/37701
- outline représente la couleur du bord
- On peut aussi choisir la couleur avec les HTML colors RGB, en mettant (255,255,255) par exemple pour le blanc. Vous pouvez trouver la votre ici : https://htmlcolorcodes.com
- Pour la largeur, width represente la largeur des bords, en pixels
- Pour les arc,
    - le style peut : ARC, PIESLICE, or CHORD. A vous de tester les différentes possibilités pour choisir celle qui vous convient.
    - extent : width of the slice in degrees
    - start : starting angle for the slice in degrees

3. Fonction database

```
global database

def database(self):

    Database_bttn['text'] = 'Mettre les données à jour'

    DTool['text']="\n Les données affichées \n dans le tableau sont : \n "+self.drawing_tool+"\n"

    print(self.drawing_tool)

    print(self.df)

    # tk.Label(f, text="Table of the"+self.drawing_tool, font=("Arial", 11),bg=white_bg).pack()

    self.table = pt = Table(f, dataframe=self.df,

                        showtoolbar=True, showstatusbar=True)

    pt.show()
```

La base de données affichées est celle pour laquelle on a déssiné un objet pour la dernière fois

IV. Propositions d'amélioration et idées

- La Scrollbar

Elle permetterait de voir l'ensemble de la radio et de
Voilà une source de code qui peut être utile pour créer une scrollbar :

```
# canvas=tk.Canvas(SaisiInfo_frame,bg=white_bg)

    # canvas.config(scrollregion=canvas.bbox('all'))

    # yscrollbar = tk.Scrollbar(self,orient='vertical')

    # xscrollbar = tk.Scrollbar(self,orient='horizontal')

    # yscrollbar.config(command=canvas.yview)

    # xscrollbar.config(command=canvas.xview)



    # SaisiInfo_frame.bind(

    #     "<Configure>",

    #     lambda e: canvas.configure(

    #         scrollregion=canvas.bbox("all")

    #     )

    # )



    # canvas.create_window((0, 0), window=SaisiInfo_frame, anchor="nw")

    # canvas.configure(xscrollcommand=xscrollbar.set,yscrollcommand=yscrollbar.set)
```

```
    # canvas.pack(side="left", fill="both", expand=True)

    # yscrollbar.pack( side = 'right', fill = 'y')

    # xscrollbar.pack( side = 'bottom', fill = 'x')
```

- Fonctions zoom in et out

Quelques morceaux de code sont disponibles ici et ici

- Choix des couleurs pour les différentes formes, ajouter une image dans la frame f2 en guise de légende.
- Une option pour changer de radio une fois qu'on a fini de travailler sur la première.
- Une fois qu'on aura une base de données construite, l'option d'ouvrir des radios sur lesquels on a déjà effectué les tracés
- Un bouton retour : pour annuler la dernière action.