BE #4 : Application de dessin vectoriel

Ceci est la "version enseignants" du BE #4 incluant les solutions aux questions posées. Ce BE propose d'implémenter une application de dessin vectoriel à la souris pour utiliser les formes développées au BE #2.

L'objectif de ce BE est d'apprendre à manipuler quelques composants du module *Python Tkinter* permettant de créer des interfaces graphiques. Vous allez créer une application simple de dessin vectoriel, qui permettra de tracer à la souris les formes définies dans le BE #2.

Quelques éléments sur *Tkinter* (45 min.)

Le module *Tkinter* (*"Tk interface"*) permet de créer des interfaces graphiques. Il contient de nombreux composants graphiques (ou *widgets*), tels que les boutons (classe **Button**), les cases à cocher (classe **CheckButton**), les étiquettes (classe **Label**), les zones d'entrée de texte (classe **Entry**), les menus (classe **Menu**), ou les zones de dessin (classe **Canvas**).

Durant ce BE, nous vous recommandons de conserver le lien vers une documentation sur *Tkinter* ouverte dans un onglet de votre navigateur. Elle contient des exemples de code qui vous seront utiles pour utiliser chacun des *widgets*.

Voici un premier exemple de code *Tkinter* :

```
from tkinter import *
from random import randint

class FenPrincipale(Tk):
    def __init__(self):
        Tk.__init__(self)

        # paramètres de la fenêtre
        self.title('Tirage aléatoire')
        self.geometry('300x100+400+400')

        # constitution de son arbre de scène
        boutonLancer = Button(self, text='Tirage')
        boutonLancer.pack(side=LEFT, padx=5, pady=5)
        self.__texteResultat = StringVar()
        labelResultat = Label(self, textvariable=self.__texteResultat)
        labelResultat.pack(side=LEFT, padx=5, pady=5)
        boutonQuitter = Button(self, text='Quitter')
        boutonQuitter.pack(side=LEFT, padx=5, pady=5)

        # association des widgets aux fonctions
        boutonLancer.config(command=self.tirage) # appel "callback" (pas de parenthèses)
        boutonQuitter.config(command=self.quit) # idem

    # tire un entier au hasard et l'affiche dans self.__texteResultat
    def tirage(self):
        nb = randint(1, 100)
        self.__texteResultat.set('Nombre : ' + str(nb))


if __name__ == '__main__':
    app = FenPrincipale()
    app.mainloop()
```

Exercice 1 - Copiez le code suivant dans un fichier appelé *Exo1.py* et exécutez-le pour observer le résultat.

Attention, utilisateurs de Mac : l'association *Spyder+Tkinter* ne fonctionne pas bien sous Mac ! Lorsque vous quitterez l'interface (par le biais du bouton *Quitter*), la fenêtre va se bloquer (*freeze*). Deux solutions:

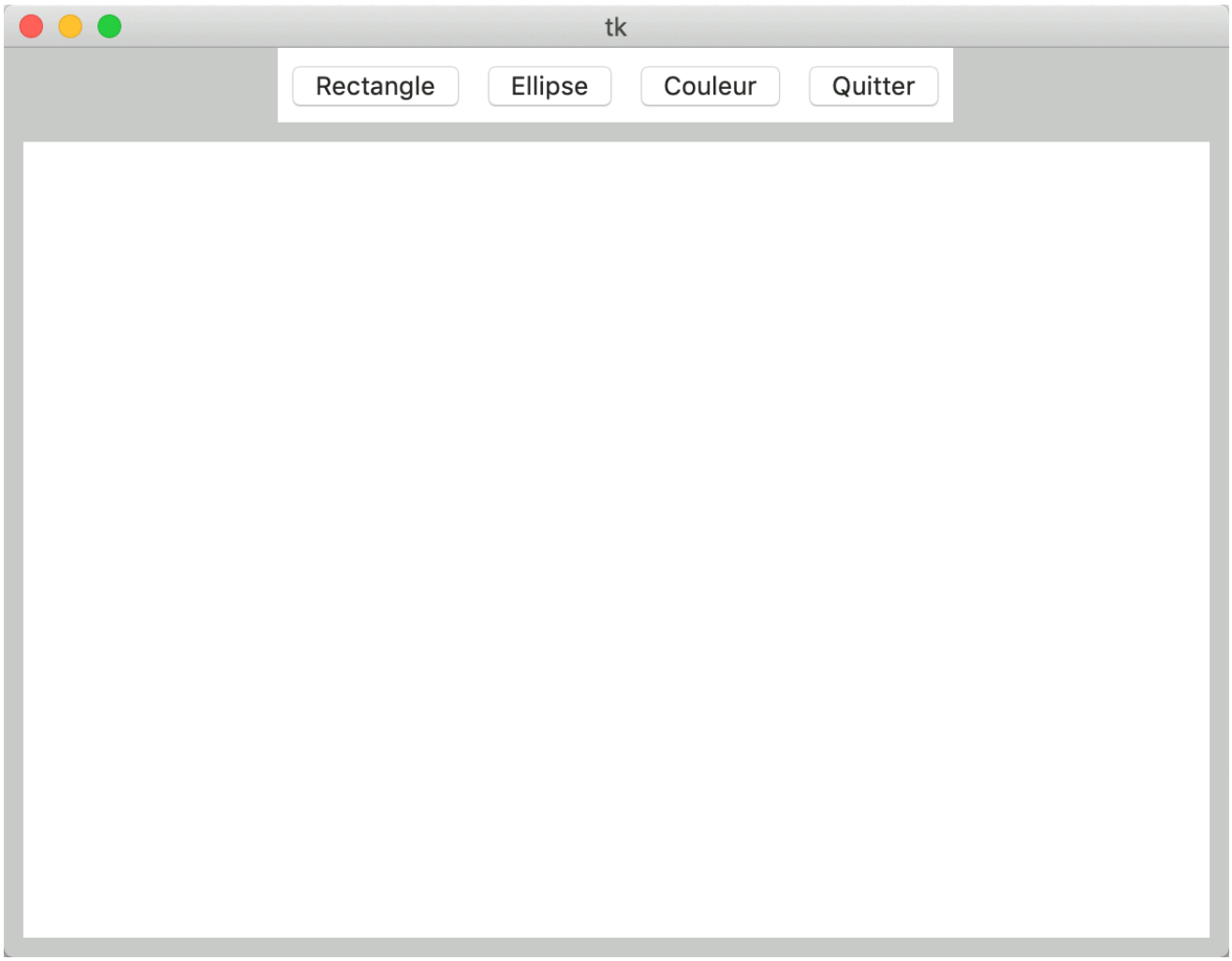
- soit vous forcez l'application à s'arrêter à chaque fois (utilisez le menu contextuel sur l'icône de l'application concernée dans la barre d'outils);
- soit vous exécutez votre programme en ligne de commande. Pour cela, ouvrez un terminal dans le répertoire de travail (clic-droit sur le répertoire → Nouveau terminal au dossier). Puis lancer la commande : `python3 Exo1.py` . Vous devriez pouvoir quitter l'application sans difficulté. N'oubliez pas de sauvegarder votre fichier sous *Spyder* avant toute exécution de cette manière !

Prenez le temps d'étudier cet exemple, et répondez aux questions suivantes :

- Combien d'éléments contient l'arbre de scène ?
- Que se passe-t-il lorsqu'on clique sur le bouton `Tirage` ?
- Comment peut-on inverser les positions des deux boutons ?
- Comment peut-on augmenter l'espace à gauche et à droite du label ?
- Comment peut-on colorier le texte du label en rouge ?

Squelette de l'application de dessin (60 min.)

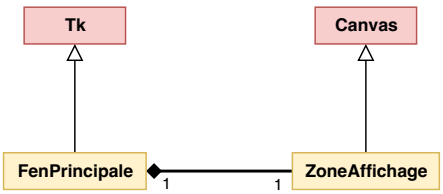
On souhaite obtenir l'interface ci-dessous, dans laquelle les utilisateurs sélectionneront le type de forme à dessiner avec les boutons, et créeront une forme en cliquant dans la zone située sous la barre d'outils (*widget* **Canvas** de *Tkinter*). On a donné une couleur grise au fond de la fenêtre pour vous aider à déterminer les différents *widgets* présents.



Une pratique courante dans les interfaces graphiques est d'intégrer le code de l'application dans l'arbre de scène, en créant des classes qui s'y intégreront comme des nœuds. Ces classes héritent des classes de *Tkinter* (pour êtres autorisées à les remplacer dans l'arbre), et nous leur ajouterons des attributs et méthodes spécifiques à leurs responsabilités dans l'application de dessin. Nous allons ainsi introduire deux classes :

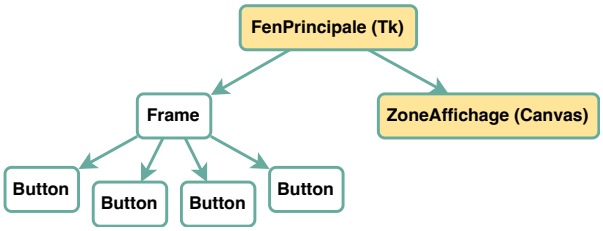
- la classe **ZoneAffichage**, qui hérite de **Canvas** et gère toutes les opérations de dessin spécifiques à votre application.
- la classe **FenPrincipale**, qui hérite de **Tk** et gère l'initialisation de l'arbre de scène et des *callbacks* des *widgets*.

Voici le diagramme UML correspondant :



Exercice 2 - Dessinez l'arbre de scène correspondant à la capture d'écran. Pour chaque nœud vous indiquerez la classe, et s'il y a lieu sa classe parente également.

Éléments de réponse pour les enseignants



Exercice 3 - Complétez le code ci-dessous avec l'initialisation de votre arbre de scène. Vous utiliserez une instance de **ZoneAffichage** pour implémenter le canevas. À ce stade, on ne vous demande pas de programmer les actions, uniquement de mettre en place le design de l'interface. Vous trouverez des exemples d'utilisation de chacun des *widgets* dans la documentation référencée plus haut.

```
from tkinter import *

class ZoneAffichage(Canvas):
    def __init__(self, parent, largeur, hauteur):
        Canvas.__init__(self, parent, width=largeur, height=hauteur)

class FenPrincipale(Tk):
    def __init__(self):
        Tk.__init__(self)
        self.configure(bg="grey")
        # L'initialisation de l'arbre de scène se fait ici

if __name__ == "__main__":
    fen = FenPrincipale()
    fen.mainloop()
```

Éléments de réponse pour les enseignants

Une difficulté ici est de penser à passer `self` comme argument au constructeur de **ZoneAffichage**, pour que cette zone d'affichage référence la fenêtre dans laquelle elle a été incluse.

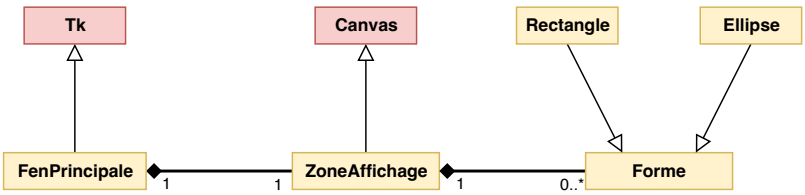
```
from tkinter import *

class ZoneAffichage(Canvas):
    def __init__(self, parent, largeur, hauteur):
        Canvas.__init__(self, parent, width=largeur, height=hauteur)

class FenPrincipale(Tk):
    def __init__(self):
        Tk.__init__(self)
        self.configure(bg="grey")

        barreOutils = Frame(self)
```

```
        barreOutils.pack(side=TOP)

        boutonRectangle = Button(barreOutils, text="Rectangle")
        boutonRectangle.pack(side=LEFT, padx=5, pady=5)
        boutonEllipse = Button(barreOutils, text="Ellipse")
        boutonEllipse.pack(side=LEFT, padx=5, pady=5)
        boutonCouleur = Button(barreOutils, text="Couleur")
        boutonCouleur.pack(side=LEFT, padx=5, pady=5)
        boutonQuitter = Button(barreOutils, text="Quitter")
        boutonQuitter.pack(side=LEFT, padx=5, pady=5)

        canevas = ZoneAffichage(self, 600, 400)
        canevas.pack(side=TOP, padx=10, pady=10)

if __name__ == "__main__":
    fen = FenPrincipale()
    fen.mainloop()
```

Dessin de formes dans le canevas (75 min.)

Vous trouverez dans le dossier de ce BE le fichier formes.py développé durant le BE #2. Nous avons agrémenté les classes **Rectangle** et **Ellipse** pour qu'elles reçoivent un canevas en argument et se dessinent dessus lors de leur initialisation. Téléchargez ce fichier dans votre répertoire de travail.

Les classes seront intégrées selon le diagramme UML suivant :



Exercice 4 - À l'aide de la méthode `bind` vue en cours, reliez les clics de la souris dans le canevas (événements `<ButtonRelease-1>`) à une nouvelle méthode de **ZoneAffichage** qui imprime les coordonnées de chaque clic avec `print` .

Exercice 5 - Modifiez cette méthode pour créer un nouveau **Rectangle** centré sur la souris chaque fois que la méthode est exécutée (choisissez des dimensions arbitraires). N'oubliez pas de stocker ce rectangle dans **ZoneAffichage** !

Exercice 6 - Lorsqu'on clique sur le bouton `Ellipse` , l'outil "Ellipse" est sélectionné et tous les futurs clics dans le canevas doivent créer une nouvelle **Ellipse**. Lorsqu'on clique ensuite sur le bouton `Rectangle` , les clics suivants créeront un **Rectangle**. L'outil sélectionné par défaut est "Rectangle". Modifiez votre code pour implémenter ce comportement.

Éléments de réponse pour les enseignants

On a utilisé l'événement `<ButtonRelease-1>` plutôt que `<Button-1>` pour ne pas interférer avec l'appui-déplacement proposé dans les exercices bonus. Enfin une difficulté ici est de penser à définir un attribut (privé) `self.__canevas` , car cet attribut est utilisé dans `clic_canevas(...)` .

```
from tkinter import *
from formes import *

class ZoneAffichage(Canvas):
    def __init__(self, master, largeur, hauteur):
        Canvas.__init__(self, master, width=largeur, height=hauteur)
        self.__formes = []
        self.__type_forme = 'rectangle'

    def selection_rectangle(self):
```

```
        self.__type_forme = 'rectangle'

    def selection_ellipse(self):
        self.__type_forme = 'ellipse'

    def ajout_forme(self, x, y):
        # Notez qu'on aurait aussi pu ajouter ce code en méthodes de Rectangle/Ellipse.
        if self.__type_forme == 'rectangle':
            f = Rectangle(self, x-5, y-10, 10, 20, "brown")
        elif self.__type_forme == 'ellipse':
            f = Ellipse(self, x, y, 5, 10, "brown")
        self.__formes.append(f)


class FenPrincipale(Tk):
    def __init__(self):
        Tk.__init__(self)

        # arbre de scène
        self.configure(bg="grey")
        barreOutils = Frame(self)
        barreOutils.pack(side=TOP)
        boutonRectangle = Button(barreOutils, text="Rectangle")
        boutonRectangle.pack(side=LEFT, padx=5, pady=5)
        boutonEllipse = Button(barreOutils, text="Ellipse")
        boutonEllipse.pack(side=LEFT, padx=5, pady=5)
        boutonCouleur = Button(barreOutils, text="Couleur")
        boutonCouleur.pack(side=LEFT, padx=5, pady=5)
        boutonQuitter = Button(barreOutils, text="Quitter")
        boutonQuitter.pack(side=LEFT, padx=5, pady=5)
        self.__canevas = ZoneAffichage(self, 600, 400)
        self.__canevas.pack(side=TOP, padx=10, pady=10)

        # commandes
        boutonRectangle.config(command=self.__canevas.selection_rectangle)
        boutonEllipse.config(command=self.__canevas.selection_ellipse)
        boutonQuitter.config(command=self.destroy)
        self.__canevas.bind("<ButtonRelease-1>", self.release_canevas)

    def release_canevas(self, event):
        self.__canevas.ajout_forme(event.x, event.y)


if __name__ == '__main__':
    fen = FenPrincipale()
    fen.mainloop()
```

Opérations de dessin supplémentaires (60 min.)

Nous allons à présent intégrer deux commandes simples dans l'application de dessin :

- Lorsqu'on clique sur une forme en maintenant la touche CTRL enfoncée, elle doit s'effacer du canevas.
- Lorsqu'on clique sur le bouton *Couleur*, un sélecteur de couleurs apparaît pour choisir la couleur de l'outil de dessin.

Exercice 7 - Implémentez l'effacement des formes avec CTRL-clic (événement `<Control-ButtonRelease-1>`). Vous pourrez faire appel aux méthodes `contient_point(...)` des classes **Rectangle** et **Ellipse** pour déterminer si la position de la souris au moment de l'événement est dans le périmètre d'une forme donnée, ainsi qu'à la méthode `effacer(...)` de la classe **Forme**.

Exercice 8 - À l'aide du module *colorchooser* de *Tkinter* (`from tkinter import colorchooser`), liez les clics sur le bouton *Couleur* à l'affichage d'un sélecteur de couleur, et utilisez la couleur renvoyée pour tous les ajouts de formes suivants.

Éléments de réponse pour les enseignants

Il est important d'arriver à l'exercice 7 qui traite pleinement du polymorphisme (et impossible de finauder pour l'éviter). Dans ce même exercice, pour éviter de modifier la liste des formes pendant qu'on itère dessus, on peut recommander de faire en deux temps : (i) chercher l'index d'une forme contenant le point, (ii) la retirer de la liste si trouvée.

Lors du test du CTRL-clic, il faut bien vérifier que lorsque deux formes se chevauchent on supprime uniquement celle du dessus.

```
from tkinter import *
from tkinter import colorchooser # doit être importé manuellement
from formes import *

class ZoneAffichage(Canvas):
    def __init__(self, master, largeur, hauteur):
        Canvas.__init__(self, master, width=largeur, height=hauteur)
        self.__formes = []
        self.__type_forme = 'rectangle'
        self.__couleur = 'brown'

    def selection_rectangle(self):
        self.__type_forme = 'rectangle'

    def selection_ellipse(self):
        self.__type_forme = 'ellipse'

    def selection_couleur(self):
        self.__couleur = colorchooser.askcolor()[1]

    def ajout_forme(self, x, y):
        # Notez qu'on aurait aussi pu ajouter ce code en méthodes de Rectangle/Ellipse.
        if self.__type_forme == 'rectangle':
            f = Rectangle(self, x-5, y-10, 10, 20, self.__couleur)
        elif self.__type_forme == 'ellipse':
            f = Ellipse(self, x, y, 5, 10, self.__couleur)
        self.__formes.append(f)

    def suppression_forme(self, x, y):
        try:
            i = next(i for i in range(len(self.__formes)-1, -1, -1)
                       if self.__formes[i].contient_point(x, y))
            f = self.__formes.pop(i) # on retire la forme de la liste
            f.effacer() # puis on l'efface du canevas
        except:
            pass


class FenPrincipale(Tk):
    def __init__(self):
        Tk.__init__(self)

        # arbre de scène
        self.configure(bg="grey")
        barreOutils = Frame(self)
        barreOutils.pack(side=TOP)
        boutonRectangle = Button(barreOutils, text="Rectangle")
        boutonRectangle.pack(side=LEFT, padx=5, pady=5)
        boutonEllipse = Button(barreOutils, text="Ellipse")
        boutonEllipse.pack(side=LEFT, padx=5, pady=5)
        boutonCouleur = Button(barreOutils, text="Couleur")
        boutonCouleur.pack(side=LEFT, padx=5, pady=5)
        boutonQuitter = Button(barreOutils, text="Quitter")
        boutonQuitter.pack(side=LEFT, padx=5, pady=5)
        self.__canevas = ZoneAffichage(self, 600, 400)
        self.__canevas.pack(side=TOP, padx=10, pady=10)

        # commandes
        boutonRectangle.config(command=self.__canevas.selection_rectangle)
        boutonEllipse.config(command=self.__canevas.selection_ellipse)
        boutonCouleur.config(command=self.__canevas.selection_couleur)
        boutonQuitter.config(command=self.destroy)
        self.__canevas.bind("<Control-ButtonRelease-1>", self.control_clic_canevas)
```

```
        self.__canevas.bind("<ButtonRelease-1>", self.release_canevas)

    def control_clic_canevas(self, event):
        self.__canevas.suppression_forme(event.x, event.y)

    def release_canevas(self, event):
        self.__canevas.ajout_forme(event.x, event.y)


if __name__ == '__main__':
    fen = FenPrincipale()
    fen.mainloop()
```

Exercices bonus

Il n'y a pas d'ordre prédéfini pour ces trois exercices supplémentaires, choisissez celui dont la fonctionnalité vous semble la plus intéressante.

Bonus 1 - Dans tout programme de dessin respectable, on doit pouvoir dessiner des formes de tailles arbitraires (pas prédéfinies). À l'aide des types d'événements `<Button-1>`, `<B1-Motion>` et `<ButtonRelease-1>`, faites qu'un mouvement de souris avec le bouton enfoncé dessine une forme en tirant ses coins (lorsqu'il ne déplace pas une forme existante). Vous pourrez utiliser les méthodes `redimension_par_points` des classes **Rectangle** et **Ellipse**.

Bonus 2 - Il serait aussi pratique de pouvoir déplacer les formes présentes sur le canevas. À l'aide des types d'événements `<Button-1>`, `<B1-Motion>` et `<ButtonRelease-1>`, implémentez la translation des formes lors des actions d'appui-déplacement de la souris. Comment faire pour qu'elles n'interfèrent pas avec la création de nouvelles formes ?

Bonus 3 - Maintenant que votre programme de dessin vectoriel est fonctionnel, on devrait pouvoir exporter chaque image produite dans un fichier. On utilise pour cela le format SVG, qui est un fichier texte contenant des instructions de dessin. Il suffit d'écrire `<svg width=600 height=400 xmlns=http://www.w3.org/2000/svg>` au début du fichier, `</svg>` à la fin, et d'insérer des balises `rect` et `ellipse` entre les deux. C'est à vous de jouer !