

OBJECTIFS : Ce TP est la suite du Tp11_B. On continue à compléter le code déjà réalisé en créant d'autres évènements qui vont permettre de déplacer ou modifier les images dans le *canvas*. Comme dans le Tp précédent, l'évaluation du travail est basée sur le rendu du fichier *.py* qui sera constitué.

⇒ Pour débiter, se placer dans le répertoire utilisé dans le Tp11_B précédent et ouvrir le fichier *tp11.py* que vous avez écrit à l'issue de la partie B . Néanmoins, pour être sûr de démarrer sur de bonnes bases, vous pouvez partir du code écrit dans le fichier *tp11.txt* disponible sur *nsibrantly.fr* (ouvrir le fichier *.txt* et copier-coller dans *pyzo*).

1. Exercice : Déplacer avec la souris, les images affichées dans le canvas

On commence cet exercice en créant un évènement lié cette fois-ci à un click de souris dans le *canvas*. La syntaxe est proche des évènements clavier vus dans le tp11_B. Comme le click souris doit se faire à l'intérieur du canvas, la méthode *bind()* doit être ici appliquée au **canvas**. On obtient par exemple le code basique, donné ci-contre :

```
def deplace(event):
    print("tu as cliqué")
```

Fonction appelée lorsque l'évènement se produit. Cette fonction est un « callback ».

```
# Main -----
pic = {}
picsDansCanvas = []
n = 0

fenetre = creer_fenetre()
zone_graphique, mon_texte, champ_saisie, bouton_valider = creer_widgets()

creation_dictionnaire_pic()

fenetre.bind("<Down>", efface_image)
fenetre.bind("<Up>", efface_tout)
zone_graphique.bind("<ButtonPress-1>", deplace)
```

Création de l'évènement « souris »

⇒ Exécuter ce code et vérifier son bon fonctionnement.

Lorsque l'on clique, il est intéressant de connaître les coordonnées du point cliqué. On récupère ces informations en relevant les propriétés *x* et *y* de l'**objet évènement** crée par python et dont le nom est *event* . Si on affiche les coordonnées du point cliqué dans le shell, la fonction *deplace()* devient :

```
def deplace(event):
    x_click = event.x
    y_click = event.y
    print(x_click , y_click)
```

Les coordonnées sont stockées ici dans 2 variables nommées *x_click* et *y_click*

⇒ Exécuter ce code.

⇒ Compléter à présent, le code ci-dessous qui permet de déterminer le numéro de la dernière image insérée dans le *canvas*. On teste bien sûr avant, s'il y a une image dans le canvas.

```
def deplace(event):
    x_click = event.x
    y_click = event.y
    print(x_click , y_click)
    if            != [] :
        num_derniere_image = picsDansCanvas[-1]
        print(num_derniere_image)
```

⇒ Exécuter ce code et vérifier son fonctionnement

On continue à améliorer le code de cette « callback » de cet évènement souris :

```
def deplace(event):
    x_click = event.x
    y_click = event.y
    print(x_click , y_click)
    if picsDansCanvas != [] :
        num_derniere_image = picsDansCanvas[-1]
        print(num_derniere_image)
        zone_graphique.itemconfigure(num_derniere_image , image = pic['z'])
```

La dernière ligne permet d'appliquer la méthode *itemconfigure()* au *canvas* nommée *zone_graphique*
Au même titre que la ligne

```
num = zone_graphique.create_image(200 , 300 , anchor = "nw", image = pic['z'])
```

permet d'**insérer** une image dans le *canvas* en utilisant la méthode *create_image()* , la ligne
`zone_graphique.itemconfigure(num_derniere_image , image = pic['z'])`
permet de **modifier l'attribut image** d'une image **déjà insérée** avec le numéro *num_derniere_image* .

⇒ Exécuter ce code et vérifier son fonctionnement

On améliorer encore une dernière fois ensemble ce code, en rajoutant la ligne :

```
def deplace(event):
    x_click = event.x
    y_click = event.y
    print('coordonnées click :' , x_click , y_click)

    if picsDansCanvas != [] :
        num_derniere_image = picsDansCanvas[-1]
        print('numéro dernière image :' , num_derniere_image)

        zone_graphique.itemconfigure(num_derniere_image , image = pic['z'])
        zone_graphique.coords(num_derniere_image , x_click , y_click)
```

Cette dernière ligne permet d'appliquer la méthode *coords()* au *canvas*. Au même titre que la ligne
`num = zone_graphique.create_image(200 , 300 , anchor = "nw", image = pic['z'])`
permet d'**insérer** une image dans le *canvas* en utilisant la méthode *create_image()* , la ligne
`zone_graphique.coords(num_derniere_image , x_click , y_click)`
permet de **modifier les coordonnées** d'une image **déjà insérée** avec le numéro *num_derniere_image* .

⇒ Exécuter ce code et vérifier son fonctionnement

⇒ Pour conclure cet exercice, on vous demande de compléter le code final de la fonction *deplace()* donné ci-après. Il devra permettre au premier click, de déplacer la première image sur l'endroit du click, au second click la seconde image, au troisième la troisième, etc Lorsque le nombre de click dépasse le nombre d'image, on revient sur la première image, etc

```
def deplace(event):
    global compteur_click
    # coordonnées du click
    [redacted]

    # on repère le numéro de l'image dans la liste picsDansCanvas[]
    if picsDansCanvas != [] :
        [redacted]

    # on incrémente le compteur de click
    compteur_click = compteur_click + 1
    if compteur_click == len([redacted]) :
        compteur_click = 0

# Main -----
pic = {}
picsDansCanvas = []
n = 0
compteur_click = 0

fenetre = creer_fenetre()
zone_graphique, mon_texte, champ_saisie, bouton_valider = creer_widgets()

creation_dictionnaire_pic()

fenetre.bind("<Down>", efface_image)
fenetre.bind("<Up>", efface_tout)
zone_graphique.bind("<ButtonPress-1>", deplace)
```

2. Exercice : Déplacer aléatoirement les images affichées dans le canvas

On souhaite créer un évènement clavier lié à la touche ⇨ (Right) du clavier. En appuyant sur cette touche, les images déjà insérées dans le *canvas* devront se déplacer toutes sur des coordonnées x , y déterminées **aléatoirement** avec la fonction `randint()`. On aura : $x = \text{randint}(0, 940)$ car la largeur du canvas est de 1000 px et que celle d'une image est de 60 px. De la même façon, $y = \text{randint}(0, 520)$. La « callback » de l'évènement sera appelée `deplace_aleatoire()`.

⇨ Exécuter dans le shell la ligne `>>> from random import randint` puis plusieurs fois la ligne `>>> randint(0, 930)`. Le résultat est-il conforme avec l'attendu ?

⇨ Créer l'évènement et sa fonction associée.

3. Exercice : Effacer l'image qui est cliquée avec un clic droit.

⇒ Créer un évènement souris `zone_graphique.bind("<ButtonPress-3>", efface_au_click)` qui appelle la fonction `efface_au_click(event)`. Tester dans un premier temps, le fonctionnement de la fonction `efface_au_click()` en y affichant simplement un message dans le shell.

Dans cet exercice, on souhaite obtenir le comportement suivant : « l'image qui est cliquée avec un clic droit doit disparaître ». Il est ainsi nécessaire lors d'un clic droit, de comparer les coordonnées du point cliqué qui sont données par les propriétés `event.x` et `event.y` de l'évènement souris `event`, à celles qui délimitent les bords gauche, droit, haut et bas d'une image.

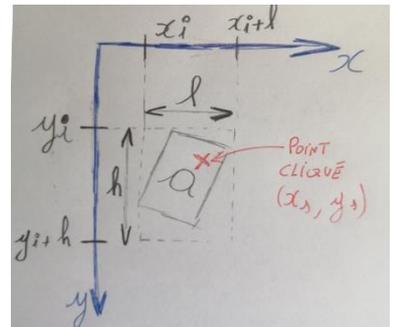
Pour connaître les coordonnées des images qui ont été insérées dans le canvas, on opère ainsi :

Lorsqu'une image a été insérée dans le canvas `zone_graphique`, on a écrit une instruction du type :
`num = zone_graphique.create_image(200 , 300 , anchor = "nw", image = pic['z'])`

Tkinter permet de récupérer les coordonnées de cette image identifiée par son numéro `num`, en écrivant la ligne : `x , y = zone_graphique.coords(num)`

Cette instruction consiste à appliquer la méthode `coords()` sur l'objet `canvas`, ce qui permet de retourner les coordonnées `x , y` courantes de l'ancre de l'image identifiée par `num`.

Cette ancre correspond ici au coin NW . Connaissant la largeur $l \approx 60 \text{ px}$ et la hauteur $h \approx 80 \text{ px}$ des images utilisées ici, on peut définir les coordonnées x_{min} , x_{max} , y_{min} , y_{max} qui délimitent la zone du `canvas` dans laquelle se trouve l'image.



Si les coordonnées du point cliqué se trouvent dans cette zone, c'est que l'image analysée est présente à l'endroit du clic.

⇒ Compléter le code de la fonction `efface_au_click()` donné ci-dessous. Il permet d'afficher le numéro de l'image cliquée avec un clic droit :

```
def efface_au_click(event) :
    [ ]
    for i in range(len(picsDansCanvas)) :
        [ ]
        if x_click > xmin and x_click < xmax and y_click > ymin and y_click < ymax :
            print(num)
```

⇒ Exécuter ce code et vérifier son fonctionnement

⇒ En s'inspirant du code incomplet ci-dessous, finaliser le script pour qu'il puisse réaliser ce qui a été initialement prévu, c'est-à-dire effacer l'image cliquée en clic droit. Pour cela, il faudra retirer l'image du *canvas* et supprimer son numéro de la liste `picsDansCanvas[]` (voir partie `tp11_b`). Exécuter ce code et vérifier son fonctionnement.

```
def efface_au_click(event) :  
    x_click = event.x  
    y_click = event.y  
    i_a_effacer = -1  
    for i in range(len(picsDansCanvas)) :  
          
        if x_click > xmin and x_click < xmax and y_click > ymin and y_click < ymax :  
            i_a_effacer = i  
    if i_a_effacer != -1 :  
        num = picsDansCanvas[i_a_effacer]  
        zone_graphique.delete(num)  
        del(picsDansCanvas[i_a_effacer])
```

Il ne faut pas réaliser un `del(l[i])` à l'intérieur d'une boucle qui parcourt cette liste l , cela provoquerait une erreur d'exécution. En effet en supprimant un élément de la liste, les index repérant chacun des éléments suivants, baissent de 1. Ainsi le dernier index prévu initialement d'être lu lorsque la boucle `for` démarre, n'existe plus. D'où une erreur.

4. Exercice : Modifier l'image qui est cliquée avec un clic molette.

⇒ Créer un évènement souris `zone_graphique.bind("<ButtonPress-2>", modifie_au_click)` qui permette de remplacer l'attribut image d'une image cliquée avec un « *clic mollette* », par l'image contenue dans le fichier `l_26.png`. On utilise à cet effet la méthode `itemconfigure()` vue dans le `tp11_B` :

```
zone_graphique.itemconfigure(   
    pic[' ' ] )
```

5. Exercice : Créer une animation lorsqu'on appuie sur la touche du clavier ⇐

⇒ Créer un évènement clavier lié à l'appui sur la touche ⇐ (Left) du clavier. La fonction callback sera nommée *tombe_image()*. Tester dans un premier temps le fonctionnement de la fonction en y affichant simplement un message dans le shell.

⇒ Compléter ensuite le script de *tombe_image()* donné ci-contre, pour que toutes les images insérées dans le *canvas* se déplacent de 5 px vers le bas lorsque l'on appuie une fois sur la touche ⇐.

```
def tombe_image(event) :  
    tombe()  
  
def tombe() :  
    for i in range(len(picsDansCanvas)) :  
        num =   
        x , y =   
        y = y + 5  
        zone_graphique.coords(num ,
```

⇒ Exécuter ce code et vérifier son fonctionnement en appuyant plusieurs fois sur la touche ⇐

Pour éviter de devoir appuyer plusieurs fois sur la touche, il est possible de créer une animation automatique. Pour cela, on peut rajouter une instruction à la fin du script de la fonction *tombe()*. Cette instruction est :

```
fenetre.after(50 , tombe)
```

Elle consiste à appliquer sur l'objet *fenetre* de *Tkinter*, la méthode *after()* qui permet de répéter l'exécution de la fonction *tombe()* toutes les 50 millisecondes.

⇒ Ajouter cette ligne en fin de script et vérifier le fonctionnement en appuyant sur la touche ⇐

Avec ce procédé, les images continuent à se déplacer, même si elles sortent du *canvas* visible. Il s'agit de pouvoir arrêter le processus lorsque plus aucune image n'est visible. On peut par exemple les retirer de la liste *picsDansCanvas[]* et ne plus les afficher dans le *canvas* (*zone_graphique.delete()*).

⇒ Réaliser cette amélioration et vérifier le fonctionnement en appuyant sur la touche ⇐

----- FIN de cette 3^{ème} partie -----

DOCUMENT A RENDRE :

Cette 3^{ème} partie vous a permis de déplacer des images dans le *canvas*, manuellement ou automatiquement.

Transférer le fichier *tp11.py* par l'intermédiaire de l'onglet transfert du site <https://nsibrantly.fr> en utilisant le code : **tp11**.