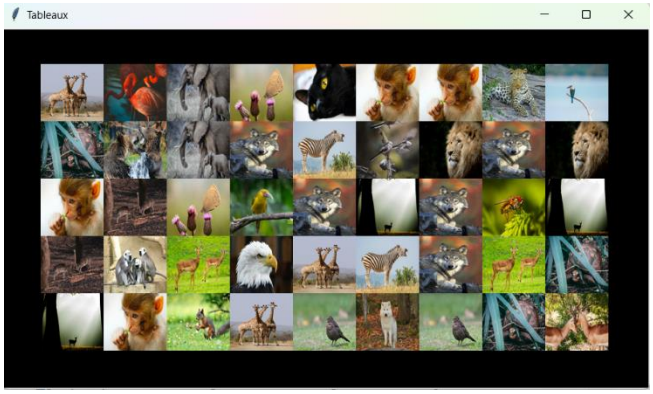
Info 15-C - Tkinter – Tableau dans Canvas



OBJECTIFS : On prolonge ici le tp15-B. On reprend les mêmes objectifs avec la différence suivante : l'évènement souris que l'on déclenche provoque un zoom de l'image cliquée *« qui se déplace vers le centre de la fenêtre en s'agrandissant »*.

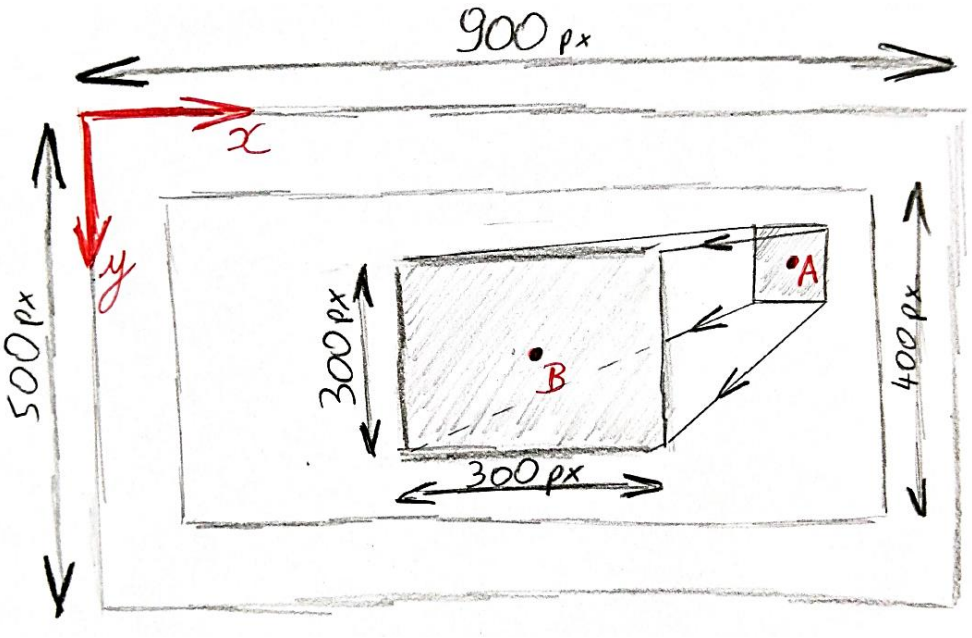
1. DEMARRAGE :

⇨ Ouvrir un nouveau fichier, toujours dans le répertoire *tp15* et y copier-coller le script donné dans ce lien texte : fichier démarrage . Sauvegardez sous le nom *tp15C.py*. Ce script python est identique à celui obtenu à l'issue du tp précédent. Il sera modifié et complété dans la suite.

⇨ Exécuter ce fichier. On retrouve la mosaïque d'image du tp précédent. En cliquant sur l'une d'entre elle, celle-ci est remplacée, par une image rouge.

2. QUE VA-T-ON FAIRE EXACTEMENT ICI ? :

Comme l'illustre le croquis ci-contre, en cliquant sur une image centrée sur le point A, on souhaite que cette image se déplace vers le centre B de la fenêtre en s'agrandissant. La taille finale sera de 300 par 300 px.
Inversement, en recliquant sur l'image zoomée, celle-ci devra revenir progressivement sur son emplacement.



Pour réaliser cela, il est nécessaire :

- de créer des copies de tailles différentes, pour chaque image,
- de réaliser une animation qui modifie progressivement la position et le fichier image lié à une image déjà insérée.
- de réaliser une seconde animation, qui permette à l'image zoomée de rétrécir pour revenir sur l'emplacement initial.

On voit tout cela ensemble, dans la suite ….

3. CREATION DES COPIES DE TAILLES DIFFERENTES POUR CHAQUE IMAGE :

Dans le code déjà mis au point, une opération de redimensionnement est déjà réalisée dans la fonction *creerImagesTk()* . Les fichiers images du type *'im...jpg'* y sont convertis en objet *pillow*, puis redimensionnés à la taille indiquée par les variables *largeur* et *hauteur*, pour enfin être encore convertis une seconde fois, en objet manipulable par *Tkinter*.

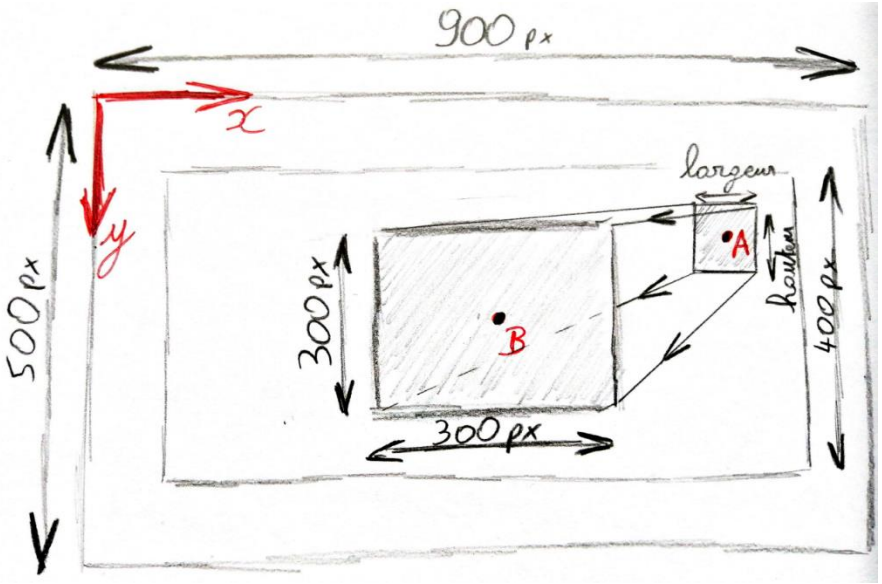
Après exécution de cette fonction *creerImagesTk()* , le dictionnaire *dicImagesTk* est rempli.
⇨ Exécuter le fichier *tp15C.py*, puis fermer la fenêtre graphique. Dans la console, exécuter `>>> dicImagesTk` afin d'afficher le contenu de ce dictionnaire…. Ce contenu est difficilement défrichable car trop dense. Exécuter alors `>>> dicImagesTk['im0.jpg']` , afin d'afficher uniquement le contenu lié à la clé *'im0.jpg'* . On obtient :

```
>>> dicImagesTk['im0.jpg']
<PIL.ImageTk.PhotoImage object at 0x0000026A349A6B20>
```

Ce contenu est un objet manipulable par *Tkinter*. Il contient toutes les informations de l'image.

On se propose ici, de redimensionner chacune des 33 images :

- comme avant, avec les dimensions *largeur* x *hauteur* (point A),
- puis, avec une dimension 300 x 300 px (point B),
- et enfin pour encore 4 tailles, intermédiaires entre celle au point A et celle au point B.



On aura ainsi pour chacun des 33 fichiers images, 6 objets images. On se propose de tous les mémoriser dans le dictionnaire *dicImagesTk* . Le contenu lié à chacune des clés sera non plus un objet image, mais une liste de 6 de ces objets images. Par exemple pour le fichier image *'im0.jpg'*, le contenu du dictionnaire *dicImagesTk* avec la clé *'im0.jpg'* sera une liste du type :

*dicImagesTk['*im0.jpg*'] = [obj0 , obj1 , obj2 , obj3, obj4 , obj5]* avec :

- *obj0 : image de taille largeur x hauteur*
- *obj1 : image de taille*
  - *(300-largeur)x0,20 + largeur*
  - *(300-hauteur)x0,20 + hauteur*
- *obj2 : image de taille*
  - *(300-largeur)x0,40 + largeur*
  - *(300-hauteur)x0,40 + hauteur*

- *obj3 : image de taille*
  - *(300-largeur)x0,60 + largeur*
  - *(300-hauteur)x0,60 + hauteur*
- *obj4 : image de taille*
  - *(300-largeur)x0,80 + largeur*
  - *(300-hauteur)x0,80 + hauteur*
- *obj5 : image de taille 300 x 300 px*

On propose ainsi de modifier la fonction *creerImagesTk()* de la manière suivante :

```
def creerImagesTk(largeur,hauteur) :
    for i in range(nbPhotos) :
        fichier = "im"+str(i)+".jpg"
        dicImagesTk[fichier] = [0,0,0,0,0,0]
        objPil = Image.open(fichier)
        for j in range(6) :
            coef = j * 0.20
            width = (300-largeur)*coef + largeur
            width = int(width)
            height = (300-hauteur)*coef + hauteur
            height = int(height)
            taille = (width,height)
            objPilRedim = objPil.resize(taille)
            objTk = ImageTk.PhotoImage(objPilRedim , master = fenetre)
            dicImagesTk[fichier][j] = objTk
```

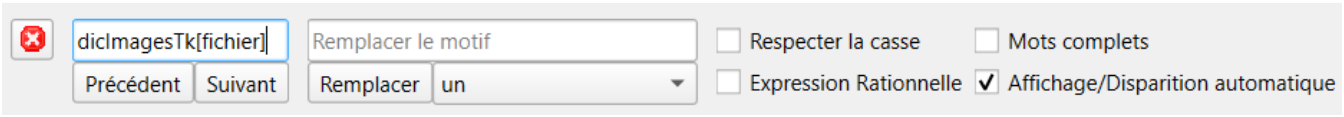
On crée ici une liste de 6 valeurs quelconques que l'on modifiera ensuite (évite d'avoir à utiliser *append*() ensuite)

On modifie la liste qui contenait initialement 6 valeurs égales à 0

Pour que cette modification soit compatible avec le reste du code, on remplace à chaque fois `dicImagesTk[fichier]` par : `dicImagesTk[fichier][0]` …. mais pas toujours ..

Pour être sûr de ne rien oublier, on utilise le « *Ctrl F* » disponible sur la plupart des logiciels :

⇨ Taper en même temps sur les touches *Ctrl* et *F*. La fonctionnalité *Rechercher* / *Remplacer* vous est alors proposée en bas de l'écran :



Compléter le champ de saisi « *Rechercher un motif* » et cliquer sur *suivant*, afin de pointer successivement toutes les lignes où apparait le motif ''*dicImagesTk[fichier]*''. Cette façon de procéder est « dangereuse » car on peut en 1 click faire « *beaucoup de dégâts* », il s'agit donc d'être vigilent. Il y a 3 lignes à modifier :

```
def creer_graphisme() :
    tab = []
    for i in range(nLignes) :
        ligne = []
        for j in range(nColonnes) :
            xA = margeX + j*largeur + largeur//2
            yA = margeY + i*hauteur + hauteur//2
            fichier = aleaFichier()
            num = zone_graphique.create_image(xA,yA,anchor="center",image = dicImagesTk[fichier][0] )
            ligne.append([num,fichier,False])
        tab.append(ligne)
    return  tab
```

Et :

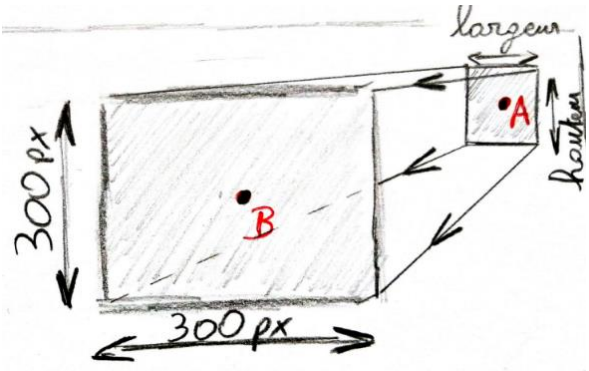
```
def gestionClick(event):
    x_click = event.x
    y_click = event.y
    i = (y_click - margeY) // hauteur
    j = (x_click - margeX) // largeur
    if 0<=i and i<nLignes and 0<=j and j<nColonnes :
        num = tab[i][j][0]
        if tab[i][j][2] == False :
            zone_graphique.itemconfigure(num , image = dicImagesTk["im0.jpg"][0])
            tab[i][j][2] = True
        else :
            fichier = tab[i][j][1]
            zone_graphique.itemconfigure(num , image = dicImagesTk[fichier][0])
            tab[i][j][2] = False
```

⇨ Exécuter le code complet pour vérifier qu'il fonctionne comme avant ….

Mission accomplie, on dispose à présent de toutes les images avec 6 tailles différentes.

4. STRATEGIE POUR REALISER L'OPERATION DE ZOOM :

Pour éviter tout problème de **chevauchement** d'images, une stratégie consiste à insérer dans le Canvas, en dernier, une image transparente. Etant insérée en dernier, elle **recouvrira** toutes les autres déjà insérées.



Ainsi, lorsque l'évènement click sera réalisé :

- L'image cliquée reste en place mais est remplacée par une image rouge
- L'image transparente prend successivement, l'apparence des 6 fichiers images de tailles différentes qui sont mémorisés dans le dictionnaire *dicImagesTk* pur la clé concernée,
- En même temps les coordonnées de cette image transparente sont modifiées pour obtenir le déplacement du point A vers le point B.

Pour réaliser les 2 opérations précédentes, il sera nécessaire de créer une animation.

On commence par s'occuper de cette image transparente.

5. IMAGE TRANSPARENTE :

Cliquer sur le lien *transparent.png* afin d'ouvrir ce fichier qui contient une image entièrement transparente de 300 px par 300 px. Avec un click droit, « *enregistrer l'image sous* » dans ton dossier de travail *tp15* sur **U:** sous le nom *'transparent.png'*. Elle se rajoute aux 32 autres images nommées *'im…jpg'*.

⇨ On modifie dans un premier temps la fonction *creerImagesTk()* afin de créer l'objet image associé à ce fichier *'transparent.png'*. Cet objet sera placé dans le dictionnaire *dicImagesTk* avec la clé

```
def creerImagesTk(largeur,hauteur) :
    for i in range(nbPhotos) :
        fichier = "im"+str(i)+".jpg"
        dicImagesTk[fichier] = [0,0,0,0,0,0]
        objPil = Image.open(fichier)
        for j in range(6) :
            coef = j * 0.20
            width = (300-largeur)*coef + largeur
            width = int(width)
            height = (300-hauteur)*coef + hauteur
            height = int(height)
            taille = (width,height)
            objPilRedim = objPil.resize(taille)
            objTk = ImageTk.PhotoImage(objPilRedim , master = fenetre)
            dicImagesTk[fichier][j] = objTk
    imTransparente = ImageTk.PhotoImage(Image.open("transparent.png") , master = fenetre)
    dicImagesTk["transparent.png"]  = imTransparente
```

Pour l'instant cette image transparente n'a pas encore été insérée dans le Canvas. On le fait dans la fonction *creer_graphisme().* ⇨ Modifier cette fonction :

```
def creer_graphisme() :
    tab = []
    for i in range(nLignes) :
        ligne = []
        for j in range(nColonnes) :
            xA = margeX + j*largeur + largeur//2
            yA = margeY + i*hauteur + hauteur//2
            fichier = aleaFichier()
            num = zone_graphique.create_image(xA,yA,anchor="center" ,image = dicImagesTk[fichier][0] )
            ligne.append([num,fichier,False])
        tab.append(ligne)
    numImTransparente = zone_graphique.create_image(W//2,H//2,anchor="center",image = dicImagesTk["transparent.png"])
    return  tab , numImTransparente
```

W est la largeur de l'écran et H sa hauteur

Et ajouter le retour dans l'appel de cette fonction dans la partie programme principal :

```
tab,numImTransparente = creer_graphisme()
```

⇨ Tester le code pour vérifier qu'il fonctionne encore.

6. MODIFICATION DE L'EVENEMENT « CLICK SOURIS » :

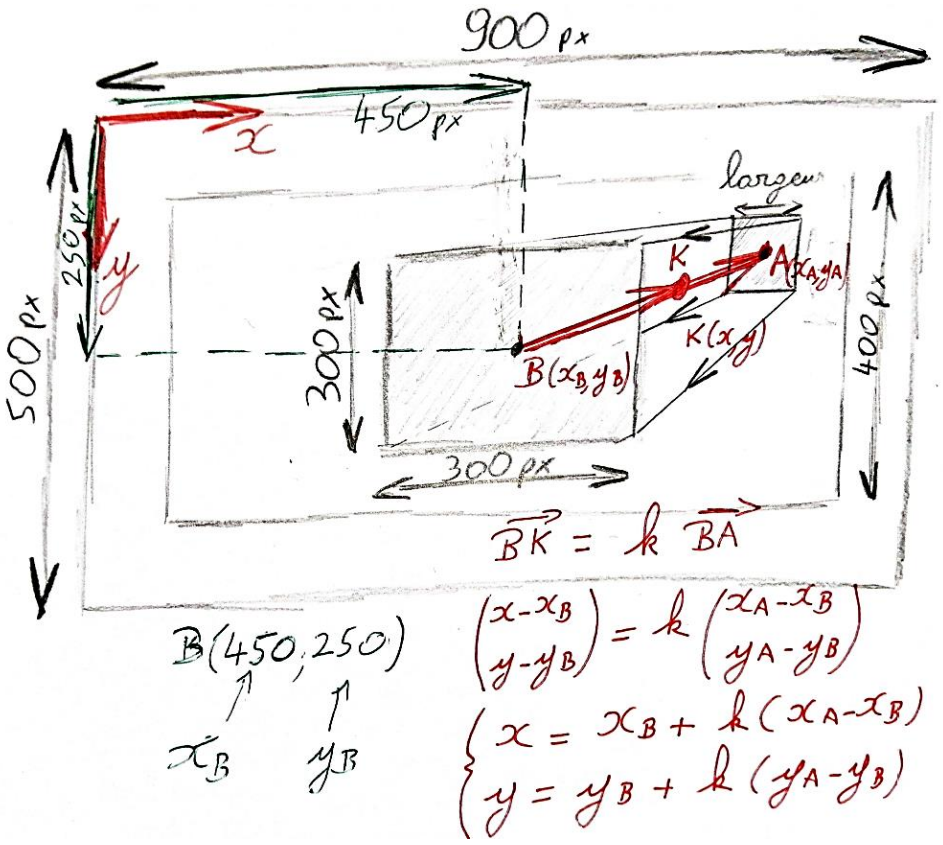
Dans le code du tp précédent, la fonction callback liée à l'évènement click de la souris était *gestionClick()*.

```
def gestionClick(event):
    x_click = event.x
    y_click = event.y
    i = (y_click - margeY) // hauteur
    j = (x_click - margeX) // largeur
    if 0<=i and i<nLignes and 0<=j and j<nColonnes :
        num = tab[i][j][0]
        if tab[i][j][2] == False :
            zone_graphique.itemconfigure(num , image = dicImagesTk["im0.jpg"][0])
            tab[i][j][2] = True
        else :
            fichier = tab[i][j][1]
            zone_graphique.itemconfigure(num , image = dicImagesTk[fichier][0])
            tab[i][j][2] = False
```

Partie à **supprimer**. Elle permettait de remplacer l'image par une image rouge

Il s'agit à présent en cas de click, de créer une animation :

- pour modifier l'objet lié à cette image : *on utilisera la méthode tkinter qui s'appelle itemconfigure(),*
-
- pour modifier les coordonnées de l'ancre de cette image *: on utilisera la méthode tkinter qui s'appelle coords()*



Pour déterminer les coordonnées du point K qui sera l'ancre de l'image du zoom qui se déplace, on réalise des calculs de géométrie vectorielle. Le point K se situe entre B et A. Pour gérer sa position, on utilise un un coefficient k compris entre 0 et 1 défini par la relation $\overrightarrow{BK} = k\ \overrightarrow{BA}$.

- Si $k = 0$, on a $\overrightarrow{BK} = 0 \times \overrightarrow{BA} = \vec{0}$. Le point K se trouve alors en B.
- Si $k = 1$, on a $\overrightarrow{BK} = 1 \times \overrightarrow{BA} = \overrightarrow{BA}$. Le point K se trouve alors en A.
- Si $k = 0.5$, le point K se trouve au milieu du segment [AB].

Comme $\overrightarrow{BK} = k\ \overrightarrow{BA}$ (voir calcul sur croquis ci-dessus), les coordonnées (x, y) du point K sont :

$$\begin{cases} x = x_B + k(x_{A.} - x_B) \\ y = y_B + k(y_{A.} - y_B) \end{cases}$$

On se propose de modifier le code de la fonction *gestionClick()* étape par étape :

⇨ Etape 1 :

```
def gestionClick(event):
    x_click = event.x
    y_click = event.y
    i = (y_click - margeY) // hauteur
    j = (x_click - margeX) // largeur
    if 0<=i and i<nLignes and 0<=j and j<nColonnes :
        num = tab[i][j][0]
        fichier = tab[i][j][1]
        animation(num,fichier,0,1)

def animation(num,fichier,i,k) :
    # l'image cliquée devient noire
    zone_graphique.itemconfigure(num , image = dicImagesTk["im0.jpg"][0])
    # on récupère les coordonnées de l'ancre de l'image cliquée (point A)
    xA , yA = zone_graphique.coords(num)
    print(xA,yA)
```

⇨ Tester le code …

⇨ Etape 2 :

```
def animation(num,fichier,i,k) :
    # l'image cliquée devient noire
    zone_graphique.itemconfigure(num , image = dicImagesTk["im0.jpg"][0])
    # on récupère les coordonnées de l'ancre de l'image cliquée (point A)
    xA , yA = zone_graphique.coords(num)
    # animation
    xB = W // 2
    yB = H // 2
    x = xB + k*(xA-xB)
    y = yB + k*(yA-yB)
    zone_graphique.coords(numImTransparente , x , y)
    zone_graphique.itemconfigure(numImTransparente , image = dicImagesTk[fichier][i])
```

⇨ Tester le code … … cela ne fonctionne pas correctement : c'est normal

⇨ Etape 3 :

```
def animation(num,fichier,i,k) :
    # l'image cliquée devient noire
    zone_graphique.itemconfigure(num , image = dicImagesTk["im0.jpg"][0])
    # on récupère les coordonnées de l'ancre de l'image cliquée (point A)
    xA , yA = zone_graphique.coords(num)
    # on déplace l'image transparente repérée par numImTransparente
    xB = W // 2
    yB = H // 2
    x = xB + k*(xA-xB)
    y = yB + k*(yA-yB)
    zone_graphique.coords(numImTransparente , x , y)
    # on affecte l'image en ième position à l'image transparente
    zone_graphique.itemconfigure(numImTransparente , image = dicImagesTk[fichier][i])
    # on réexécute la fonction 6 fois
    if i < 5 :
        i = i + 1
        k = k - 0.2
        fenetre.after(100,animation,num,fichier,i,k)
```

⇨ Tester le code …cela fonctionne …. mais si on clique sur plusieurs images, elles sont à chaque fois remplacées par une image rouge ….

7. AMELIORATION POUR BLOQUER L'EVENEMENT SI UNE IMAGE EST DEJA ZOOMEE :

On améliore le code en créant un variable booléenne DRAPEAU dans le programme principal. Sa valeur par défaut sera égale à *True*. Si une image est zoomée, cette variable prendra la valeur *False*. En parallèle, l'évènement click précédent ne pourra être réalisé uniquement si DRAPEAU a la valeur *True*. Cela donne :

⇨ Dans le programme principal :

```
# Variables globales --------
W = 900
H = 500
nLignes = 5
nColonnes = 9
nbPhotos = 33
dicImagesTk = {}
DRAPEAU = True
```

⇨ Dans la fonction gestionClick() :

```
def gestionClick(event):
    global DRAPEAU
    if DRAPEAU == True :
        x_click = event.x
        y_click = event.y
        i = (y_click - margeY) // hauteur
        j = (x_click - margeX) // largeur
        if 0<=i and i<nLignes and 0<=j and j<nColonnes :
            DRAPEAU = False
            num = tab[i][j][0]
            fichier = tab[i][j][1]
            animation(num,fichier,0,1)
```

8. AMELIORATION EN PERMETTANT LE DEZOOMAGE :

On désire encore améliorer cette application en permettant à l'utilisateur de refermer la fenêtre zoomée lorsqu'il clique dessus. Dans ce cas, le zoom se réduit progressivement pour retrouver la position de l'image initiale. Le mouvement d'ouverture précédent se fera ici à l'envers.

Pour ne pas trop être amené à trop modifier le code précédent, on propose de créer dans le programme principal une liste qui pourra recevoir les index *i* et *j* de la cellule qui a été zoomée. Cette liste sera accessible en lecture et en écriture dans les autres fonctions.

```
# Variables globales ------------
W = 900
H = 500
nLignes = 50
nColonnes = 90
nbPhotos = 33
dicImagesTk = {}
DRAPEAU = True
etatImTransparente = [0,0]
```

On propose ensuite de modifier la fonction *gestionClick()* de la façon suivante :

```
def gestionClick(event):
    global DRAPEAU
    x_click = event.x
    y_click = event.y
    if DRAPEAU == True :
        i = (y_click - margeY) // hauteur
        j = (x_click - margeX) // largeur
        if 0<=i and i<nLignes and 0<=j and j<nColonnes :
            DRAPEAU = False
            num = tab[i][j][0]
            fichier = tab[i][j][1]
            etatImTransparente[0] = i
            etatImTransparente[1] = j
            animation(num,fichier,0,1,"go")
    elif DRAPEAU == False :
        if  (W//2 - 150) < x_click < (W//2 + 150) and (H//2 - 150) < y_click < (H//2 + 150)  :
            DRAPEAU = True
            i = etatImTransparente[0]
            j = etatImTransparente[1]
            num = tab[i][j][0]
            fichier = tab[i][j][1]
            animation(num,fichier,5,0,"back")
```

Si on clique sur une cellule de la mosaïque

On sauvegarde les index i et j de la cellule cliquée

Si on clique sur la fenêtre zoom

On récupère les index i et j de la cellule cliquée

Un argument a été ajouté à la fonction *animation()* pour différencier la phase de création du zoom et sa phase de rétractation. Cette fonction devient alors :

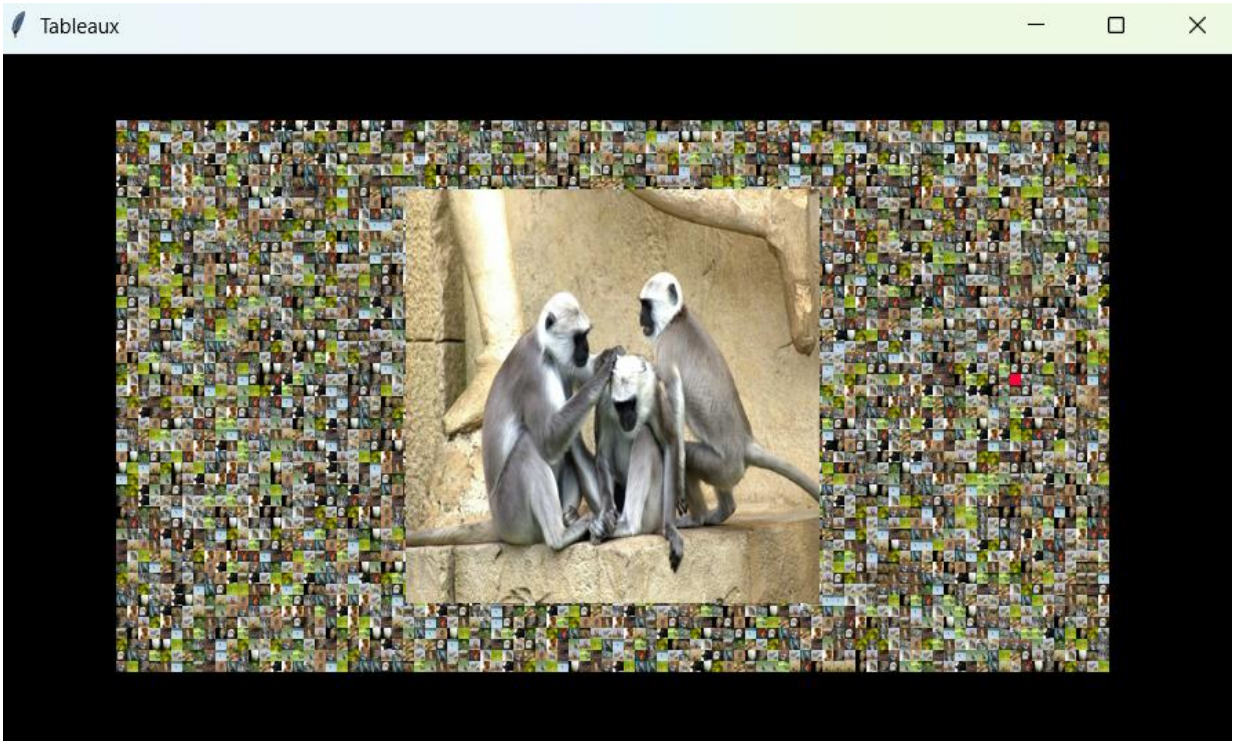
```
def animation(num,fichier,i,k,sens) :
    if sens == "go" :
        # l'image cliquée devient noire
        zone_graphique.itemconfigure(num , image = dicImagesTk["im0.jpg"][0])
    # on récupère les coordonnées de l'ancre de l'image cliquée (point A)
    xA , yA = zone_graphique.coords(num)
    # on déplace l'image transparente repérée par numImTransparente sur le point K
    xB = W // 2
    yB = H // 2
    x = xB + k*(xA-xB)
    y = yB + k*(yA-yB)
    zone_graphique.coords(numImTransparente , x , y)
    # on affecte l'image en ième position à l'image transparente
    zone_graphique.itemconfigure(numImTransparente , image = dicImagesTk[fichier][i])
    if sens == "back" and i == 0 :
        # l'image noire redevient celle initiale
        zone_graphique.itemconfigure(num , image = dicImagesTk[fichier][0])
    # on réexécute la fonction 6 fois
    if sens == "go" and i < 5 :
        i = i + 1
        k = k - 0.2
        fenetre.after(100,animation,num,fichier,i,k,sens)
    elif sens == "back" and i > 0 :
        i = i - 1
        k = k + 0.2
        fenetre.after(100,animation,num,fichier,i,k,sens)
```

⇨ Modifie ton script, toujours en donnant du sens à ce que tu écris … et teste …

⇨ Teste aussi l'ensemble avec

```
nLignes = 50
nColonnes = 90
```

. C'est ce qui est intéressant en informatique, lorsqu'un script est mis au point, on peut l'utiliser dans situations « *extrêmes* ». On obtient normalement :



⇨ Uploade le fichier *tp15C.py* sur nsibranly.fr, toujours avec le code ***tp13***.

Remarque : On peut créer un mouvement de rotation de l'image sur elle-même lors du déplacement en rajoutant uniquement les 2 lignes surlignées ci-dessous :

```
def creerImagesTk(largeur,hauteur) :
    for i in range(nbPhotos) :
        fichier = "im"+str(i)+".jpg"
        dicImagesTk[fichier] = [0,0,0,0,0,0]
        objPil = Image.open(fichier)
        for j in range(6) :
            coef = j * 0.20
            width = (300-largeur)*coef + largeur
            width = int(width)
            height = (300-hauteur)*coef + hauteur
            height = int(height)
            taille = (width,height)
            objPilRedim = objPil.resize(taille)
            if j == 1 or j == 3 : objPilRedim = objPilRedim.rotate(20)
            if j == 2 or j == 4 : objPilRedim = objPilRedim.rotate(-20)
            objTk = ImageTk.PhotoImage(objPilRedim , master = fenetre)
            dicImagesTk[fichier][j] = objTk
    imTransparente = ImageTk.PhotoImage(Image.open("transparent.png") , master = fenetre)
    dicImagesTk["transparent.png"]  = imTransparente
```