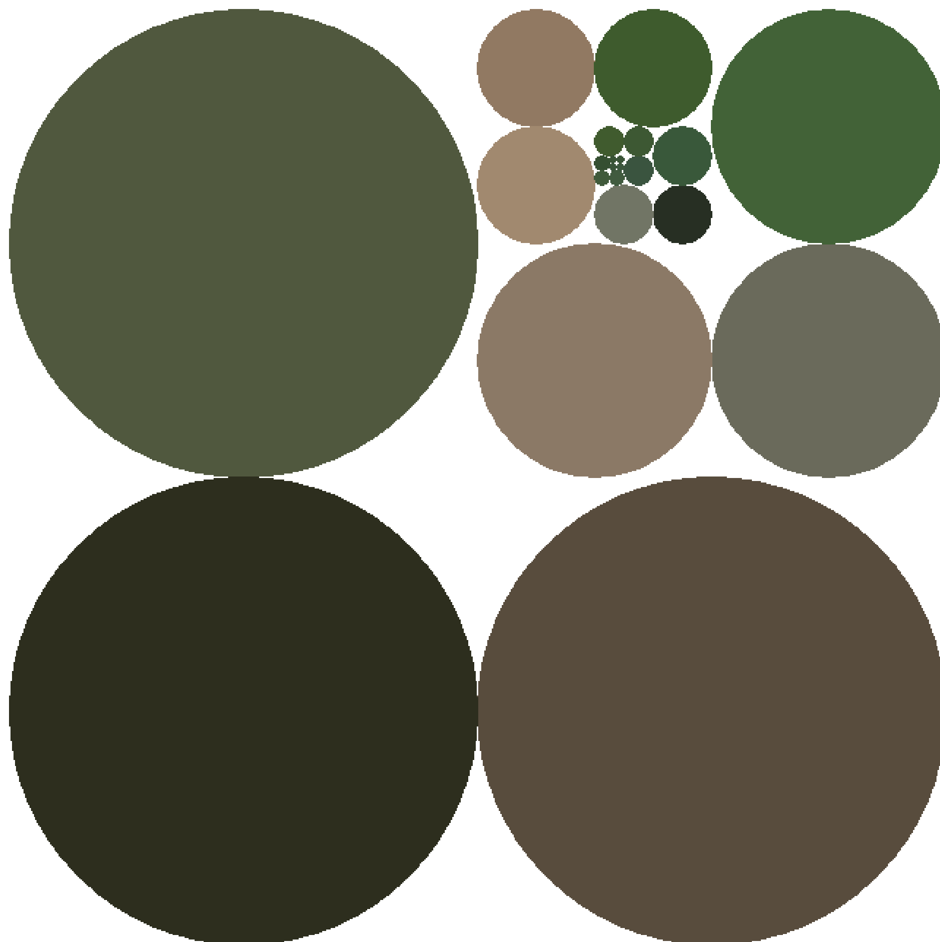
Braun Louison
Jacquemet Elise

2014/2015
TS4

Projet ISN :
Les Bons points



Lycée de la plaine de l'Ain

SOMMAIRE

Présentation du projet..1

Partie gestion..2

Idée de base..2

Séparation des cercles..2

Correspondance cercle/liste..4

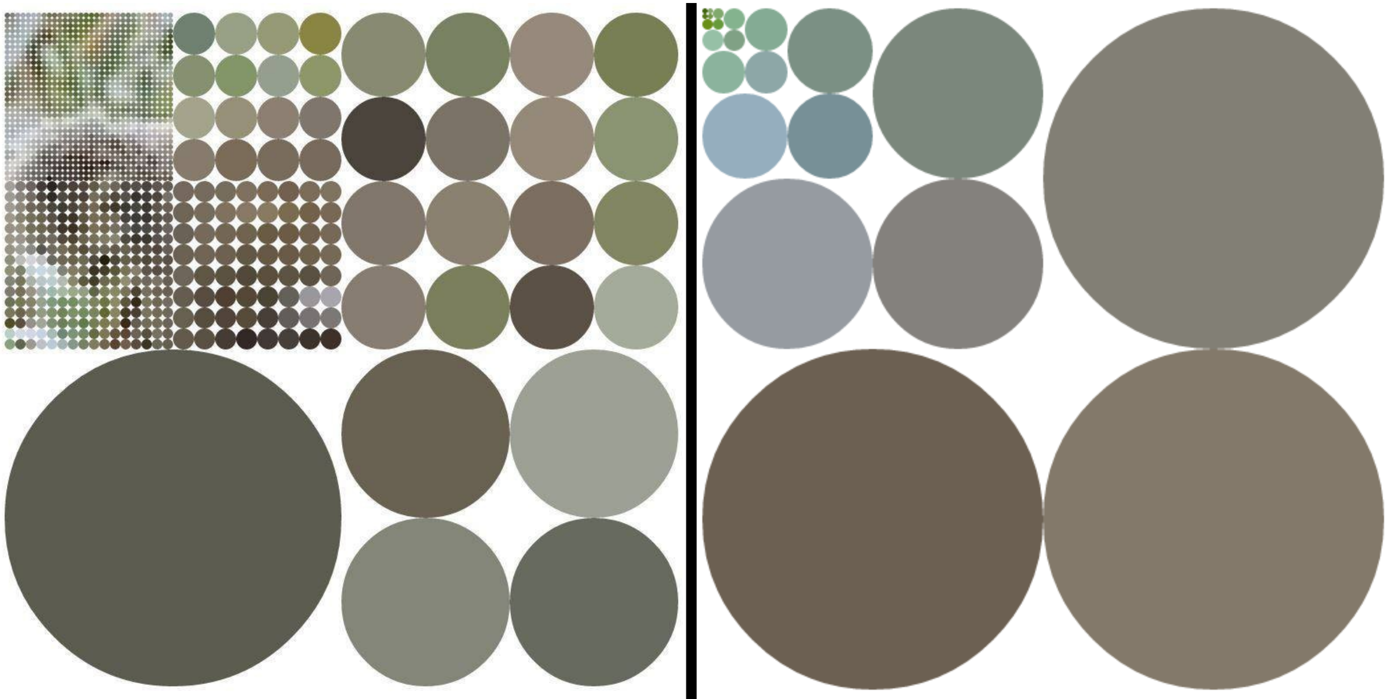
Détecter la fin..6

Conclusion 6

Annexes 7

Présentation du projet

Nous nous sommes inspirés du site koalastothemax.com. Ce site affiche un grand cercle et lorsque la souris passe dessus il se sépare en quatre cercles de rayon deux fois plus petit et d'une couleur différente. Ce cercles se séparent encore jusqu'à ce que les cerclent soient assez petits et affichent finalement une image. L'image doit donc être un carré avec une longueur qui est une puissance de 2 donc son nombre de pixels est une puissance de 4.



Captures du site koalastothemax.com

Nous avons voulu faire la même chose mais en python. Pour cela nous avons déduit plusieurs critères en observant le site :

- Chaque cercle est de la couleur moyenne des pixels de l'image finale qu'il contient.
- Il y a 8 tailles de cercle différentes, le nombre de cercles est donc multiplié 7 fois par deux, pour être au final égal à $2^7 = 128$. Nous travaillerons donc avec des images de 128*128 pixels.

Nous avons choisi d'avoir au final des cercles de 5 pixels de diamètre, l'image finale serait donc un carré de 640 pixels, ce qui est aussi le diamètre du premier cercle.

Nous avons séparé le travail en deux parties : Élise s'est occupée de la partie graphique et j'ai travaillé sur la partie de gestion des données.

Partie gestion

Idée de base

J'ai choisi de travailler avec une liste regroupant tous les pixels de l'image de départ. J'ai donc programmé une fonction qui récupère tous les pixels d'une image de longueur et de largeur de 128 pixels. C'est la fonction *RecupererPixels()* qui renvoie cette liste. Pour que le programme n'affiche pas la même image à chaque fois, la fonction tire au sort une image. Quatre images ont une chance sur 8 d'être choisies et les deux dernières ont deux chances sur 8 car c'est celles qui peuvent être remplacées par une image choisie par l'utilisateur (format jpg ou png). La liste des pixels était d'abord une liste de listes qui correspondait à un tableau, chaque ligne de pixels de l'image étant une liste.

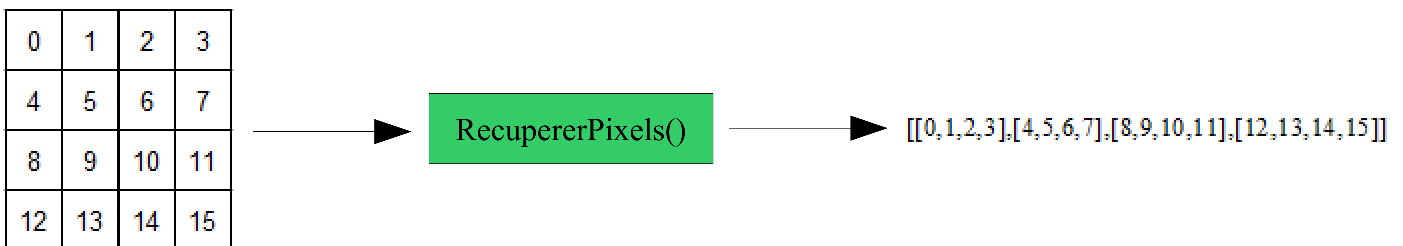


Illustration de la fonction *RecupererPixels*

Je ne pouvais pas facilement vérifier mes fonctions avec une liste de 16 384 pixels donc au lieu de travailler avec la liste retournée par *RecupererPixels()* j'ai travaillé sur des listes réduites (16 ou 64 éléments pour garder une puissance de 4) dans lesquelles chaque élément était égal à son indexation dans la liste, comme le tableau ci-dessus. Cela me permettait de vérifier rapidement le résultat des foncions.

L'inconvénient ici était que les cercles étaient composés de pixels répartis sur plusieurs lignes donc il était compliqué de retrouver quels pixels correspondaient à quels cercles. J'ai vite abandonné cette méthode pour une liste unique, contenant tous les pixels de l'image.

Séparation des cercles

Le principal problème de ma partie était de répartir les pixels à la séparation d'un cercle pour former les 4 cercles plus petits. En effet, à chaque séparation on doit regrouper les quatre coins de l'image mais les pixels étaient d'abord triés par liste de ligne. J'ai d'abord codé une fonction

decoupeEn4Liste(liste) qui séparait la liste de lignes en deux à sa moitié, ce qui correspond à une coupure horizontale de l'image, puis qui découpait chaque ligne en deux pour avoir une partie gauche et une partie droite. Enfin elle récupérait une liste sur deux dans chaque moitié verticale de l'image pour avoir les quatre quarts séparés et renvoyait une liste composée de ces quarts.

L'inconvénient de cette fonction est qu'elle doit être lancée à chaque séparation de cercle, ce qui pouvait ralentir le programme et gêner l'utilisateur. De plus, elle créait des listes de demi-lignes, et il était difficile de retrouver ensuite les pixels qui composaient les cercles.

J'ai pensé à trier les pixels une fois qu'ils étaient tous récupérés pour ensuite n'avoir qu'à découper une liste en quatre à chaque séparation. Pour cela j'ai d'abord modifié la fonction *decoupeEn4Liste(liste)* pour qu'elle découpe une liste unique. On découpe toujours la liste en deux pour avoir une séparation haut/bas mais il fallait déterminer le nombre de pixels consécutifs appartenant au même quart. C'est le rôle de la ligne suivante :

```
s=int(sqrt(len(listeCoupe[0])*2))//2
```

listeCoupe[0] est la première moitié de la liste donc len(listeCoupe[0])*2 est le nombre de pixels du carré à diviser. Sa longueur est égale à la racine carrée de ce nombre et on récupère à chaque fois la moitié gauche ou droite donc il faut encore diviser par deux. Le int() permet de convertir le nombre flottant renvoyé par sqrt() en un entier. Une fois ce chiffre obtenu il suffit de faire une boucle pour récupérer ce qui correspond à la moitié de chaque ligne.

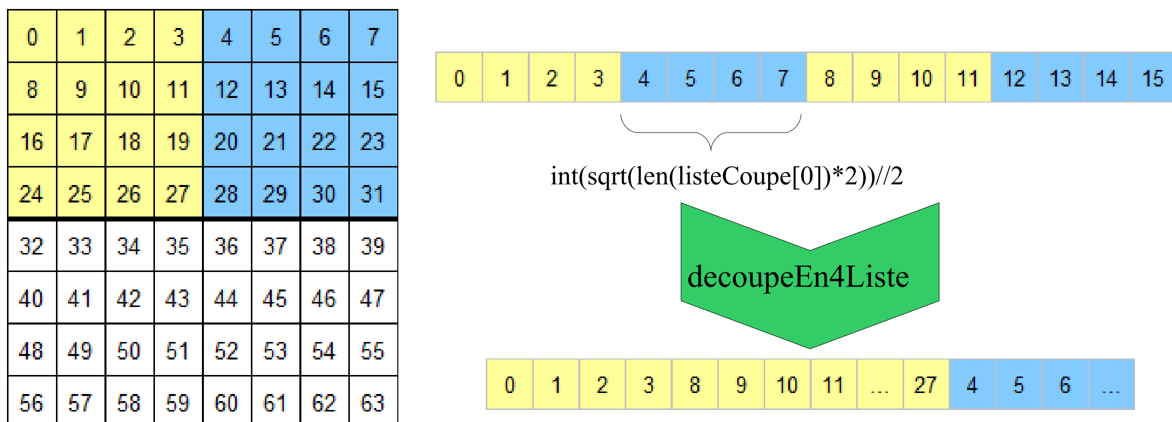
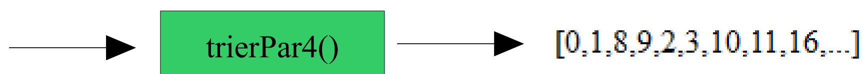


Illustration de la fonction *decoupeEn4Liste(liste)*

J'appelle cette fonction dans une autre fonction, *trierPar4(liste)*, qui est une fonction récursive. Si la longueur de la liste en argument est égale à 4 alors la fonction renvoie cette liste et sinon elle lui applique decoupeEn4Liste(), sépare les quatre quarts et renvoie (trierPar4(quart1) + trierPar4(quart2) + trierPar4(quart3) + trierPar4(quart4)). Ainsi la fonction sépare chaque carré de pixels en quatre jusqu'à ce qu'ils soient de taille minimale.

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63



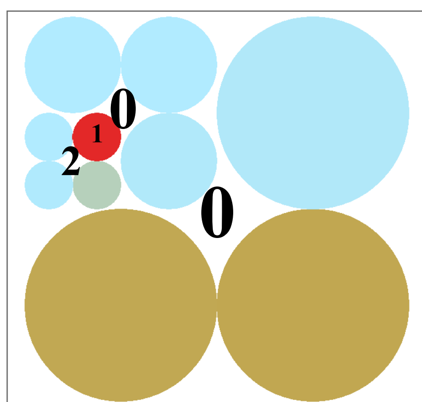
liste de départ — fin de la première étape — fin de la dernière étape

Illustration de la fonction *trierPar4*

Grâce à ce tri initial de la liste des pixels il suffit à chaque séparation de couper la liste correspondant au grand cercle en quatre pour obtenir les petits cercles.

Correspondance cercle/liste

Un autre problème de ma partie était de trouver un moyen d'identifier quelle partie de la liste de pixels correspondait à chaque cercle. J'ai pensé à utiliser des listes imbriquées et des tags pour les cercles. A chaque séparation, la liste correspondant au cercle deviendrait une liste de quatre listes, qui correspondent aux petits cercles. Chaque cercle possède un tag, celui du plus premier étant « 0 » et les cercles enfants héritent du tag de leur parent auquel on ajoute « 0 », « 1 », « 2 », ou « 3 » de sorte que le tag d'un cercle est aussi sa position dans les listes imbriquées. Ainsi, si un cercle possède le tag « 0021 », les pixels le composant se trouvent dans listeImage[0][0][2][1] (listeImage étant la liste qui contient les pixels), et le cercle est en rouge sur l'image :



Pour que le grand cercle ait un tag égal à « 0 » il faut que la liste de pixels soit de la forme [[pixels]]. Ensuite, il faut arriver à accéder à la liste qui correspond à un cercle pour le séparer ou pour déterminer sa couleur. C'est le rôle de ces lignes :

```
cercleInit = liste
for a in chaine:
        k = int(a)
        cercleInit = cercleInit[k]
```

« chaine » est le tag du cercle. La boucle permet de plonger dans les listes jusqu'à arriver à celle qui correspond au cercle. Cette méthode est utilisée dans la fonction *defColor(liste, chaine)* qui prend en entrée la liste des pixels et le tag du cercle et retourne la couleur moyenne des pixels en faisant la moyenne du rouge, du vert et du bleu. Cependant il a fallu modifier un peu les lignes de code pour la fonction *diviseCercle(liste, chaine)* car contrairement à la fonction *defColor* qui ne fait que récupérer des valeurs celle-ci doit modifier la liste pour en créer quatre nouvelles. Si on utilisait les mêmes lignes on créait les quatre petit cercles de cette manière:

```
cercleInit = [[quart1], [quart2], [quart3], [quart4]]
```

Or lorsque l'on assigne une valeur à une variable de type liste avec l'opérateur « = » on perd l'adresse initiale, ce qui signifie que le programme ne considérait plus que c'était une liste imbriquée quelque part dans la liste totale mais une nouvelle variable, à part. Pour éviter cela, j'ai fait une boucle qui plongeait un niveau moins profond et j'ai récupéré la valeur x = int(chaine[-1]) qui est le dernier élément du tag du cercle pour travailler avec cercleInit[x]. Grâce à cela, le programme comprenait que cette liste était contenue dans la liste totale.

Détecter la fin

Une fois tout cela terminé, j'ai codé une fonction qui devait détecter quand l'utilisateur avait complété l'image, c'est à dire quand tous les petits cercles avaient été formés. C'est la fonction *fini()*. Elle teste si il y a bien 16384 cercles affichés à l'écran. Si oui, elle affiche une fenêtre avec une image faite par Élise qui indique à l'utilisateur qu'il a terminé et qu'il peut lancer le programme avec une de ses propres images en la nommant « mon_image.jpg » et en la plaçant dans le dossier du script.

Au début, il y avait un problème d'affichage. La fenêtre de fin était vide et python indiquait « pyimage doesn't exists ». Cela était dû au fait qu'il y avait deux fenêtres Tkinter ouvertes en même temps, ce qui créait un conflit. Pour régler le problème il a fallu remplacer

```
fenetreFin = Tk()
```

par

```
fenetreFin = Toplevel()
```

car *Tk()* désigne la fenêtre principale et *Toplevel()* une autre fenêtre.

Conclusion

Ce projet nous a permis de travailler en équipe à la fois sur la gestion des listes et l'affichage de fenêtres et d'images. J'ai trouvé le fait de travailler en groupe intéressant car il fallait rechercher des solutions réalisables des deux côtés et ensuite réunir les parties de chacun pour avoir un programme fonctionnel. Nous avons aussi remarqué grâce au sujet qu'à partir du moment où les points qui composent une image sont assez petit nous la percevons des façon assez nette même si elle est composée de disques sur un fond blanc.

Annexes

Code du programme :

```
# Elise JACQUEMET - Louison BRAUN
# PROGRAMME FINAL

from tkinter import *
from random import randint
from copy import deepcopy
from math import sqrt
from PIL import Image


#--------------- SOUS-FONCTIONS --------------#

## Partie gestion, codée par Louison

def RecupererPixels():
	'''La fonction choisit au hasard une image parmi les 4 prédéfinies et
	renvoie la liste de ses pixels'''

	n = randint(1,8) #nombre aléatoire pour choisir une image
	if n == 1:
		img=Image.open("Pandas128.jpg")
	elif n==2:
		img=Image.open("Pikachu128.jpg")
	elif n==3:
		img=Image.open("Rilakkuma128.jpg")
	elif n==4:
		img=Image.open("Kiwi128.jpg")
	elif n==5 or n==6:
		img=Image.open("mon_image.png")
		img = img.resize((128,128))
	else:
		img=Image.open("mon_image.jpg")
		img = img.resize((128,128))

	listePixels=[]
	#on récupère les pixels de l'image dans listePixels
	for a in range (128):
		for b in range (128):
			listePixels.append(img.getpixel((b,a)))
	return(listePixels)


def defColor(liste, chaine):
	'''Récupère le tag correspondant au cercle et trouve les pixels
	associés dans la liste. Renvoie le tuple RGB correspondant à la moyenne
	des couleurs'''

	#on retrouve la liste correspondant au cercle grâce à son tag
	cercleInit=liste
	for a in chaine:
		k=int(a)
		cercleInit=cercleInit[k]
	#création des variables qui vont contenir la somme des valeurs de rouge, vert et bleu:
```

```
        TotR=0
        TotG=0
        TotB=0
        #on fait la moyenne des trois couleurs
        for pixel in cercleInit:
                TotR+=pixel[0]
                TotG+=pixel[1]
                TotB+=pixel[2]
        n=len(cercleInit)
        TotR=TotR/n
        TotG=TotG/n
        TotB=TotB/n
        return(TotR,TotG,TotB)


def decoupeEn4Liste(liste):
        '''La fonction prend en entrée une liste et la découpe en
        4 quarts correspondant à 4 carrés (haut à gauche en premier
        et bas à droite en dernier'''

        listeCoupe=[]
        #on commence par couper la liste en deux, ce qui correspond à une scission verticale sur l'image
        l1=liste[:len(liste)//2]
        l2=liste[len(liste)//2:]
        listeCoupe.append(l1)
        listeCoupe.append(l2)

        #on détermine le nombre de pixels consécutif qui appartiennent au meme quart
        s=int(sqrt(len(listeCoupe[0])*2))//2

        k=0
        quart1=[]
        quart2=[]
        quart3=[]
        quart4=[]

        #on répartit les pixels
        for k in range(0,s*s,2):
                quart1 += listeCoupe[0][k*s:k*s+s]
                quart2 += listeCoupe[0][(k+1)*s:(k+2)*s]
                quart3 += listeCoupe[1][k*s:k*s+s]
                quart4 += listeCoupe[1][(k+1)*s:(k+2)*s]

        listeFinale=quart1+quart2+quart3+quart4
        return listeFinale


def trierPar4(liste):
        '''Trie les pixels de la liste de façon à ce qu'ils soient faciles à
        séparer au cours du programme'''

        n=len(liste)
        if n == 4: return(liste)

        listeModif = decoupeEn4Liste(liste)
        m = len(listeModif)
        quart1 = listeModif[:m//4]
        quart2 = listeModif[m//4:m//2]
        quart3 = listeModif[m//2:3*m//4]
        quart4 = listeModif[3*m//4:]
        return(trierPar4(quart1)+trierPar4(quart2)+trierPar4(quart3)+trierPar4(quart4))
```

```
def diviseCercle(liste,chaine):
        '''Récupère le tag du grand cercle, retrouve la liste qui lui
        correspond pour la diviser en 4 listes et renvoie la liste ainsi
        modifiée'''

        #on récupère la liste qui correspond au cercle à séparer
        cercleInit=liste
        for a in chaine[:len(chaine)-1]:
                k=int(a)
                cercleInit=cercleInit[k]
        x=int(chaine[-1])
        #et on la coupe en 4 listes
        l=len(cercleInit[x])//4
        petitsCercles = [cercleInit[x][:l],cercleInit[x][l:2*l],cercleInit[x][2*l:3*l],cercleInit[x][3*l:]]
        cercleInit[x]=petitsCercles
        return liste



## Partie graphique, codée par Elise

def Clic(event):
        ''' Détecte le passage de la souris sur un des cercles du canvas et
        appelle la fonction pour le séparer   '''

        # position du pointeur de la souris
        X = event.x
        Y = event.y

        # recuperer le tag puis les coordonnées du cercle le + proche
        Cercle_a_separer = Canevas.find_closest(X,Y)[0]
        tag = Canevas.gettags(Cercle_a_separer)[0]

        [x1,y1,x2,y2] = Canevas.coords(Cercle_a_separer)

        # definition des coordonnées du centre
        xcentre=(x2+x1)/2
        ycentre=(y2+y1)/2

        # rayon du cercle
        r=(xcentre-x1)

        # si le curseur est bien dans le cercle le plus proche
        # et que celui-ci n'a pas la taille mini
        if  (X-xcentre)**2+(Y-ycentre)**2 <= r**2 and len(tag)<8:
                diviseCercle(listeImage,tag)
                separe(tag,x1,x2,y1,y2)
                Canevas.delete(Cercle_a_separer)
                fini() # pour vérifier si l'on a complété l'image


def separe(tag,x1,x2,y1,y2):
        ''' sépare le cercle initial (sur lequel se trouve la souris)
        et le sépare en 4 nouveaux cercles '''

        xcentre=(x2+x1)/2
        ycentre=(y2+y1)/2 # coord centre cercle
        # Création des quatres nouveaux cercles référencés via le tag du cercle dont ils sont issus
        nouv_tag= tag+"0"
        couleur0=remplissage(defColor(listeImage, nouv_tag))
```

```
        Canevas.create_oval(x1,y1,xcentre,ycentre,fill=couleur0, outline=couleur0, tags=nouv_tag)

        nouv_tag= tag+"3"
        couleur3=remplissage(defColor(listeImage, nouv_tag))
        Canevas.create_oval(xcentre,ycentre,x2,y2,fill=couleur3, outline=couleur3, tags=nouv_tag)

        nouv_tag= tag+"1"
        couleur1=remplissage(defColor(listeImage, nouv_tag))
        Canevas.create_oval(xcentre,y1,x2,ycentre,fill=couleur1, outline=couleur1, tags=nouv_tag)

        nouv_tag= tag+"2"
        couleur2=remplissage(defColor(listeImage, nouv_tag))
        Canevas.create_oval(x1,ycentre,xcentre,y2,fill=couleur2, outline=couleur2, tags=nouv_tag)



def remplissage(rgb):
        ''' convertit la couleur du cercle donnée en base rgb en un hexadécimal '''
        couleur='#%02x%02x%02x' % rgb
        return couleur


def reinit():
        ''' reinitialise la zone graphique '''
        global listeImage

        listeImage = RecupererPixels()
        listeImage = [trierPar4(listeImage)]
        Canevas.delete(ALL)

        # Creation d'un objet graphique, ici le cercle de départ
        d=640
        x1=Largeur//2-d//2
        x2=x1+d
        y1=Hauteur//2-d//2
        y2=y1+d

        couleur=remplissage(defColor(listeImage, "0"))
        Cercle = Canevas.create_oval(x1,y1,x2,y2,fill=couleur, outline=couleur , tags='0')


def fini():
        '''vérifie si tous les cercles ont été créés et affiche une nouvelle fenetre
        si oui, pour signaler à l'utilisateur qu'il a fini'''

        if len(Canevas.find_all()) == 4**7 :
                print('Bravo !')
                fenetreFin = Toplevel()
                fenetreFin.title("Bravo !")
                fond_fin = PhotoImage(file="mavis.png")
                w = fond_fin.width()
                h = fond_fin.height()
                CanevasFin = Canvas(fenetreFin,width=w,height=h)
                CanevasFin.pack()
                CanevasFin.create_image(w//2, h//2, image=fond_fin)
                fenetreFin.mainloop()
```

```
## EXECUTION

# Creation de la fenetre MENU
fenetre2 = Tk()
fenetre2.title("accueil")
fond_accueil = PhotoImage(file="tableau.gif")
w = fond_accueil.width()
h = fond_accueil.height()

fenetre2.geometry("%dx%d+0+0" % (w, h))
accueil = Label(fenetre2, image=fond_accueil)
accueil.pack(side='top', fill='both', expand='yes')

# Creation d'un widget Button (pour lancer le programme)
go = PhotoImage(file='button_go.png')
Button(accueil, image=go, command= fenetre2.destroy).pack(side='bottom', pady = 40)

fenetre2.mainloop()

# Creation de la fenetre principale
fenetre = Tk()
fenetre.title("LES BONS POINTS")

# Creation d'un widget Canvas
Largeur = 700
Hauteur = 660

Canevas = Canvas(fenetre,width=Largeur,height=Hauteur,bg ='white')
Canevas.pack(padx =5, pady =5)

listeImage=[]


reinit()

# La methode bind() permet de lier un evenement avec une fonction
Canevas.bind('<B1-Motion>',Clic) # evenement bouton gauche enfonce (hold down)
Canevas.focus_set()
Canevas.pack(padx=10,pady=10)

# Creation d'un widget Button (bouton Quitter)
photo1 = PhotoImage(file='button_quit.png')
Button(fenetre, image=photo1, command = fenetre.destroy).pack(side='right',padx=5,pady=0)

# Creation d'un widget Button (bouton Effacer)
photo2 = PhotoImage(file='button_reinit.png')
Boutonreinit = Button(fenetre, command = reinit, image=photo2)
Boutonreinit.pack(side ='left', padx = 5, pady = 0)
print("passez sur les cercles en cliquant")

fenetre.mainloop()
```

Image affichée lorsque tous les cercles ont été créés