Théo CARY - RT2

Python et Tkinter TP 2

Exercice 1 : Réalisation d'un outil pour dessiner des réseaux

- ### Génération des éléments et création de boutons

J'ai commencé par importer des images de switch, routeur, pc et téléphone au format png depuis google.

Pour imiter le logiciel **packet tracer** j'ai voulu créer une barre de tâche où il est possible de sélectionner les différents équipements sous forme d'icône pour créer un schéma .

```
21    barre = Frame(root, bg='blue')
22    barre.pack(fill=X)
```

J'utilise **Frame** qui permet de créer une zone rectangulaire où je vais placer mes icônes. Pour les icônes je compresse les images des équipements réseaux que j'ai sélectionné auparavant :

```
30    img10 = Image.open(r"C:\Users\33659\Desktop\image\switch.png")
31    image10=img10.resize((40,30))
32    uImg10 = ImageTk.PhotoImage(image10)
33
```

Les icônes sont en fait des boutons, j'associe à chaque bouton l'image correspondante et j'utilise **grid** pour placer le bouton sur ma barre d'icônes :

```
boutonR=barre1(barre, text= 'ROUTEUR', image=uImg9)
boutonR.grid(row=0, column=1)
boutonR['command'] = apparition
```

le ***barre1*** ci dessus est une classe que je devais créer obligatoirement et l'associer à chaque bouton pour utiliser **grid** sinon une erreur s'affichait.

```
class barre1(Button):
    def test(self):
        print("")
```

J'appelle ensuite la fonction apparition qui permet d'invoquer l'image avec **create_image** sur le canvas.

```
48  def apparition():
49
50      global image1
51      c.create_image(100,200, image = image1)
```

J’ai créé une fonction pour chaque bouton, c’est répétitif mais je n’ai pas trouvé de solution pour créer une fonction générale commune à tous les boutons.

- ## Raccourcis clavier

J’ai ensuite créé des raccourcis clavier.
Pour la création d’un nouvel item par raccourcis clavier j'ai utilisé la fonction **lambda** car au début il me manquait un **argument** dans ma fonction "apparition" et quand je voulais ajouter un argument je ne pouvais plus utiliser le bouton permettant de faire apparaître mon routeur par exemple.

```
118  root.bind_all("<R>", lambda a: apparition()) # routeur
119  root.bind_all("<P>", lambda a: apparition2()) #pc
120  root.bind_all("<S>", lambda a: apparition3()) #switch
121  root.bind_all("<T>", lambda a: apparition4()) #telephone
```

Avec la fonction lambda je peux utiliser le raccourcis clavier et garder mon bouton car cette fonction permet d’avoir un nombre quelconque d'arguments.

- ## Déplacement des images

Pour le déplacement des images, j’ai utilisé la méthode objet avec self, cette méthode est très efficace car elle permet de réduire considérablement le code au lieu de tout faire avec des fonctions.

La méthode objet étant assez complexe pour moi, mon code ne fonctionnait pas correctement alors j’ai cherché des codes sur internet pour m’aider et je m’en suis inspiré pour l’exercice.

Je crée une classe et j'utilise la méthode __init__ avec self en premier argument puis je définis les valeurs par défaut **self.__x,y,image,move**.

```
class MainFrame:

    def __init__( self, image2):
        self.__image2 = image2
        self.__x, self.__y = 250,250
        self.__picture0 = c.create_image( self.__x, self.__y,image =  self.__image2)
        self.__move = False
        c.bind( "<Button-1>", self.startMovement )
        c.bind( "<ButtonRelease-1>", self.stopMovement )
        c.bind( "<Motion>", self.movement )
```

Il y a aussi la méthode bind qui permet d'associer un événement à une fonction, ici quand j'exécute un clic gauche (**startMovement)**, quand je relache (**stopMovement)** et Motion permet de déplacer l'image en suivant les coordonnées du pointeur de la souris.

La fonction **startMovement** permet de faire la conversions des coordonnées x et y du pointeur de la souris en coordonnée sur le canvas :

```
def startMovement( self, event ):
    self.__move = True
    self.initi_x = c.canvasx(event.x)
    self.initi_y = c.canvasy(event.y)
    #print('startMovement init', self.initi_x, self.initi_y)
    self.movingimage = c.find_closest(self.initi_x, self.initi_y, halo=5)
```

La fonction **movement** permet de voir l'image se déplacer en suivant le pointeur.

end_x et **end_y** indique les coordonnés en temps réel et **deltax deltay** permet de réaliser la différence afin de faire une mise à jour avec une nouvelle localisation des coordonnées.

```
148     def movement( self, event ):
149         if self.__move:
150             end_x = c.canvasx(event.x)
151             end_y = c.canvasy(event.y)
152             # mise à jour avec une nouvelle localisation
153             deltax = end_x - self.initi_x
154             deltay = end_y - self.initi_y
155             self.initi_x = end_x
156             self.initi_y = end_y
157             #print('movement init', self.initi_x, self.initi_y)
158             c.move(self.movingimage, deltax, deltay)
```

c.move : déplacement de l'objet.

- ## Reset

Création d'un nouveau bouton dans la barre à côté des icônes

Pour réinitialiser le schéma :
class ***reset1*** obligatoire pour utiliser grid et placer le bouton dans la colonne 8 de la barre.

c.delete va supprimer toutes les images et lignes présentes.

```
105   class reset1(Button):
106       print("")
107
108   def reset():
109       c.delete(ALL)
110
111   boutonReset=reset1(barre, text="RESET")
112   boutonReset.grid(row=0, column=8)
113   boutonReset['command'] = reset
```

- ## Tracé de ligne

Cette fonction permet de tracer des lignes afin de relier les items entre eux comme sur paint.

J'utilise **create_ovale** avec les coordonnées qui suivent le pointeur de la souris.

```
def ligne(event):
    color='black'
    x1, y1=(event.x-1),(event.y-1)
    x2, y2=(event.x+1),(event.y+1)
    c.create_oval(x1,y1,x2,y2, fill=color, outline=color)

c.bind("<B3-Motion>", ligne)
```

B3-motion désigne le clic droit de la souris. J'ai essayé d'utiliser le clic gauche pour tracer les lignes quand un bouton est sélectionné mais cela provoque des erreurs car le clic gauche est déjà associé et je ne peux pas déplacer image **et** tracer des lignes.

- **Bouton help**

J'utilise la classe **label** pour afficher l'aide dans une nouvelle fenêtre.

```
131  def help():
132      fen=Tk()
133      fen.title("Aide")
134      fen.geometry("800x50")
135      texteLabel = Label(fen,
136      text = "Pressez les touches
137      texteLabel.pack()
138      fen.mainloop()
```

Exercice 2 : Des traits bien droits !

Pour tracer des traits droits j'ai essayé de créer une fonction où je clic à deux endroits différents du schéma, la fonction récupère ensuite les coordonnés des deux points et trace une ligne entre ces deux points.

Mais j'arrive seulement à récupérer les coordonnées comme pour
la fonction startMovement : print('startMovement **init', self.initi_x, self.initi_y**)