

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

Vous devez réaliser un logiciel permettant de dessiner des architectures réseau de façon simple et élégante. Le cahier des charges imposé par le client indique que votre logiciel doit permettre de représenter :

- des switches
- des clients (PC ou téléphone)
- des routeurs

Les différents éléments peuvent être présents de multiple fois dans l'interface. La création d'un nouvel item pourra se faire par un raccourci clavier ( C client, S switch, R routeur, par exemple ) ou par toolbar / menu, et il doit être possible de déplacer ou d'effacer les items à la souris.

Un clic droit sur un item doit permettre d'en modifier les propriétés :

- Nom
- Icône

Enfin, il doit être possible de dessiner des traits sur le schémas afin de représenter les liens réseau.

```
In [ ]: import tkinter as tk
from tkinter import simpledialog
from PIL import Image, ImageTk

class WishPacketTracer:
    def __init__(self, root):
        self.root = root
        self.root.title("Wish Packet Tracer")

        # 1)
        self.canvas = tk.Canvas(root, bg="ivory", width=800, height=600)
        self.canvas.pack(expand=tk.YES, fill=tk.BOTH)

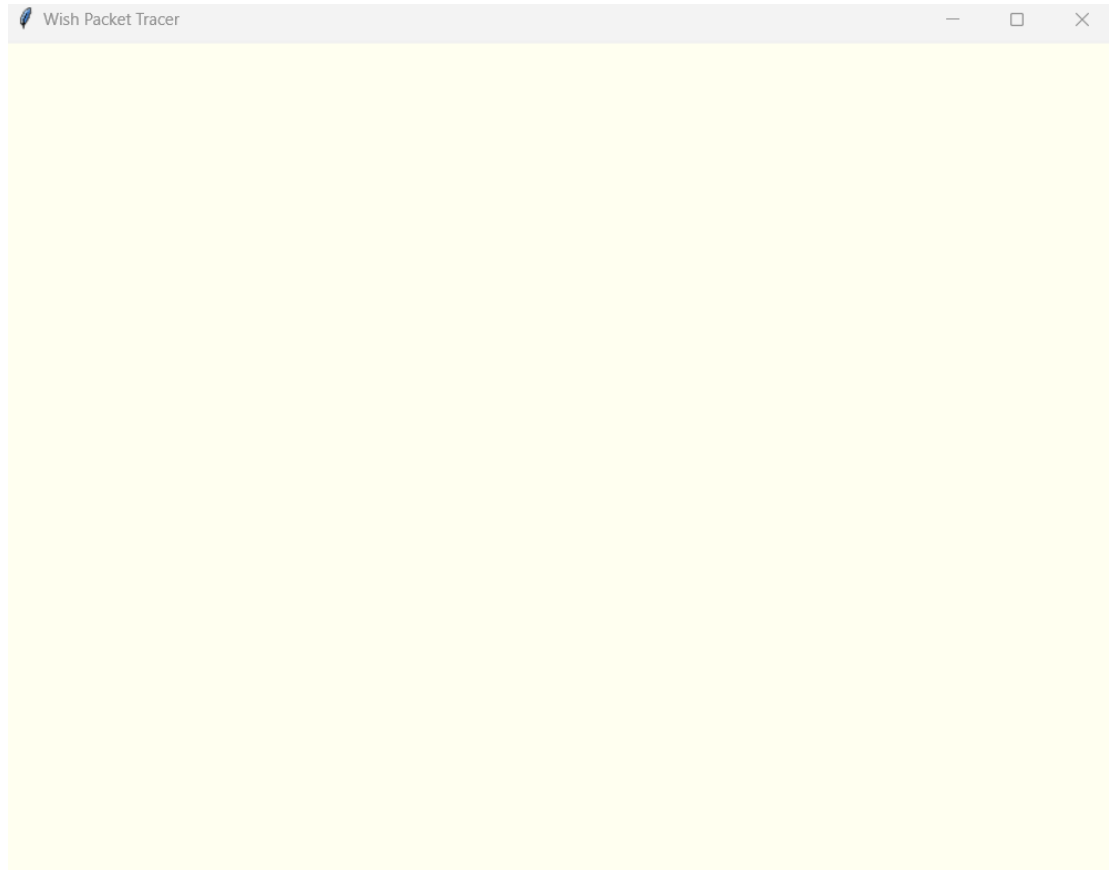
# 1)
root = tk.Tk()
app = WishPacketTracer(root)
root.mainloop()
```

J'ai dans mon premier commit, créer une classe qui permettra d'instancier les objets de l'application car il doivent être présents plusieurs fois en même temps dans la fenêtre.

A la fin de mon programme, je crée une fenêtre qui s'appelle root et à laquelle je créer une instance de la classe "WishPacketTracer".

Et donc le programme va créer un canvas basique de 800x600 qui va s'étendre càd prendre la place dans tout son espace (en l'occurrence la fenêtre) et doit s'élargir

horizontalement et verticalement à la fois avec le "pack(expand pour l'extension et fill pour l'élargissement)".



Puis j'ai ajouté à mon code, une barre d'outils dans laquelle, j'ai au préalable chargé mes 3 images (pc, routeur et switch) grâce à PIL et je les redimensionne et je les ai rendues cliquables à l'aide de la méthode "Button" dans ma barre d'outils et à l'appui du bouton (donc de l'image), une image va être créée dans mon canvas à l'aide de la méthode "create\_image".

```
In [ ]: import tkinter as tk
from tkinter import simpledialog
from PIL import Image, ImageTk

class WishPacketTracer:
    def __init__(self, root):
        self.root = root
        self.root.title("Wish Packet Tracer")

        self.canvas = tk.Canvas(root, bg="ivory", width=800, height=600)
        self.canvas.pack(expand=tk.YES, fill=tk.BOTH)

        # Barre d'outils
        self.toolbar = tk.Frame(root)
        self.toolbar.pack(side=tk.LEFT, fill=tk.Y)

        # Boutons de la barre d'outils (utilisant des images redimensionnées avec
        self.image_router = self.load_and_resize_image("./img/Router.png", 50, 50)
        self.button_router = tk.Button(self.toolbar, image=self.image_router, command=self.create_image)
        self.button_router.pack(side=tk.LEFT)

        self.image_switch = self.load_and_resize_image("./img/switch.png", 50, 50)
```

```

self.button_switch = tk.Button(self.toolbar, image=self.image_switch, co
self.button_switch.pack(side=tk.LEFT)

self.image_pc = self.load_and_resize_image("./img/pc.png", 50, 50)
self.button_client = tk.Button(self.toolbar, image=self.image_pc, comman
self.button_client.pack(side=tk.LEFT)

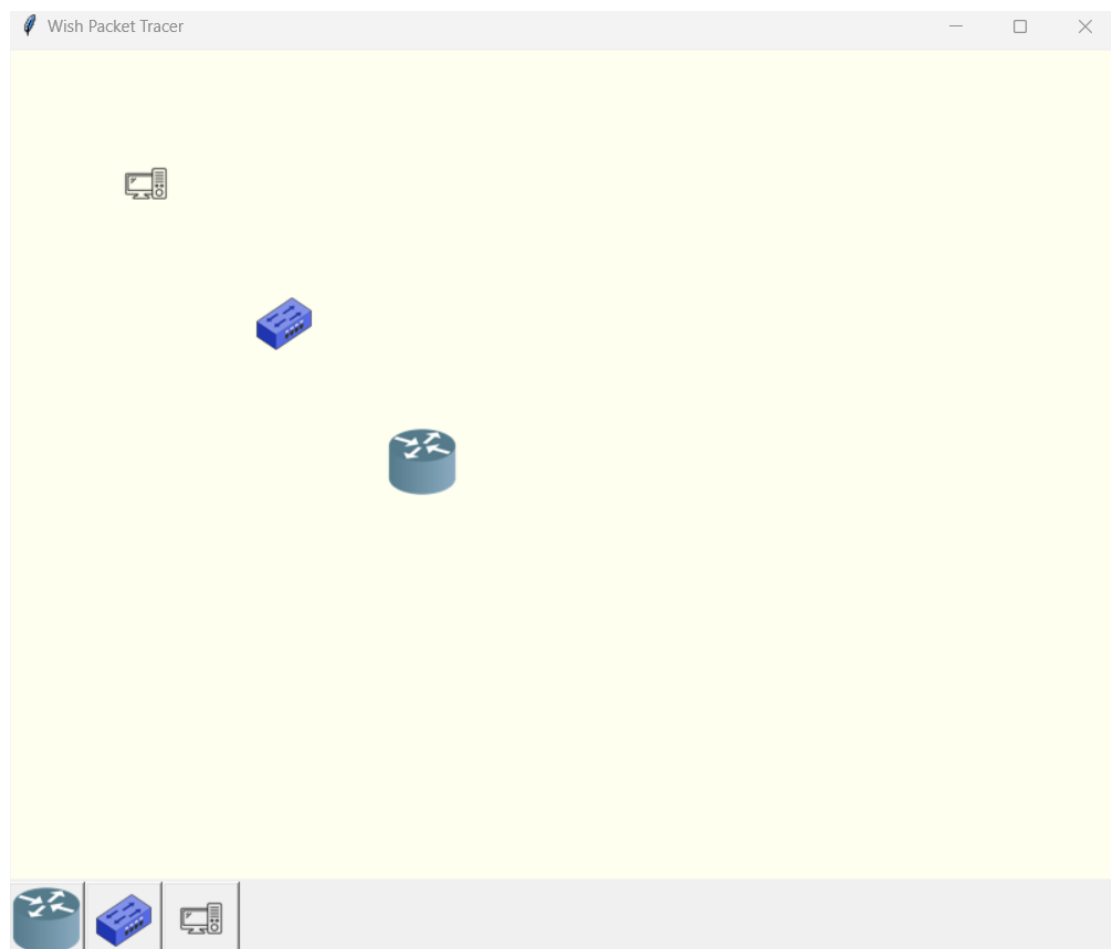
def create_client(self):
    item = self.canvas.create_image(100, 100, image=self.image_pc, tags="cli
def create_switch(self):
    item = self.canvas.create_image(200, 200, image=self.image_switch, tags=

def create_router(self):
    item = self.canvas.create_image(300, 300, image=self.image_router, tags=

def load_and_resize_image(self, filename, width, height):
    image = Image.open(filename)
    image = image.resize((width, height))
    photo = ImageTk.PhotoImage(image)
    return photo

root = tk.Tk()
app = WishPacketTracer(root)
root.mainloop()

```



Puis, j'ai implémenter la possibilité de pouvoir déplacer les différents items dans mon canvas à l'aide de deux écouteurs d'évènements qui vont se succéder :

1. "Button-1" Qui va écouter au clic gauche sur mon canvas et va déclencher une méthode "left\_click" (clic gauche)
  - Et cette méthode, prend en arguments "self" pour l'instance de la classe et "event" (pour le clic c-à-d les coordonnées en abscisses et ordonnées du clic).
  - Puis j'affecte à une variable item le résultat de la méthode "find\_closest" qui va rechercher l'élément le plus proche des coordonnées du clic (à l'aide des arguments "event.x" pour la coord en abscisse du clic et "event.y" pour la coord en ordonnée du clic).
  - Ensuite, une vérification pour savoir s'il y a un élément à l'emplacement du clic et si oui, on définit à une variable instanciée "self.current\_item", la 1ère valeur et unique de item.
2. "B1-Motion" Qui va écouter au clic gauche enfoncé sur mon canvas et déclencher une méthode "drag\_item" prenant en argument en + de l'instance, "event" pour récupérer les coordonnées du curseur enfoncé.
  - Il vérifie d'abord s'il y a une valeur dans "self.current\_item" et donc en ayant appuyé avant de pouvoir rester enfoncé, on a déclenché la méthode "left\_click" qui a attribué une valeur à "self.current\_item".
  - Il récupère alors les coordonnées actuelles de la souris en abscisses / ordonnées (event.x, event.y) puis va à chacun soustraire les coordonnées actuelles de l'élément sélectionné en abscisses "self.canvas.coords(self.current\_item)[0]" et puis en ordonnées "self.canvas.coords(self.current\_item)[1]"
  - Et la différence des coords actuelles de la souris moins les coords actuelles de l'élément sélectionné nous donne la distance parcourue par la souris avec en abscisses "dx" et en ordonnées "dy".

Ainsi, la méthode "move" permet de déplacer un élément (ici, "self.current\_item" donc l'élément sélectionné) de "dx" pixels (une distance horizontale) et de "dy" pixels (une distance verticale). C'est comme cela, qu'une image peut être "glissée-déposée" (drag-and-drop).

```
In [ ]: import tkinter as tk
from tkinter import simpledialog
from PIL import Image, ImageTk

class WishPacketTracer:
    def __init__(self, root):
        self.root = root
        self.root.title("Wish Packet Tracer")

        self.canvas = tk.Canvas(root, bg="ivory", width=800, height=600)
        self.canvas.pack(expand=tk.YES, fill=tk.BOTH)

        # Barre d'outils
        self.toolbar = tk.Frame(root)
        self.toolbar.pack(side=tk.LEFT, fill=tk.Y)
```

```

# Boutons de la barre d'outils (utilisant des images redimensionnées avec
self.image_router = self.load_and_resize_image("./img/Router.png", 50, 50)
self.button_router = tk.Button(self.toolbar, image=self.image_router, command=self.router)
self.button_router.pack(side=tk.LEFT)

self.image_switch = self.load_and_resize_image("./img/switch.png", 50, 50)
self.button_switch = tk.Button(self.toolbar, image=self.image_switch, command=self.switch)
self.button_switch.pack(side=tk.LEFT)

self.image_pc = self.load_and_resize_image("./img/pc.png", 50, 50)
self.button_client = tk.Button(self.toolbar, image=self.image_pc, command=self.client)
self.button_client.pack(side=tk.LEFT)

self.canvas.bind("<Button-1>", self.left_click)
self.canvas.bind("<B1-Motion>", self.drag_item)

def create_client(self):
    item = self.canvas.create_image(100, 100, image=self.image_pc, tags="client")

def create_switch(self):
    item = self.canvas.create_image(200, 200, image=self.image_switch, tags="switch")

def create_router(self):
    item = self.canvas.create_image(300, 300, image=self.image_router, tags="router")

def left_click(self, event):
    item = self.canvas.find_closest(event.x, event.y)
    if item:
        self.current_item = item[0]

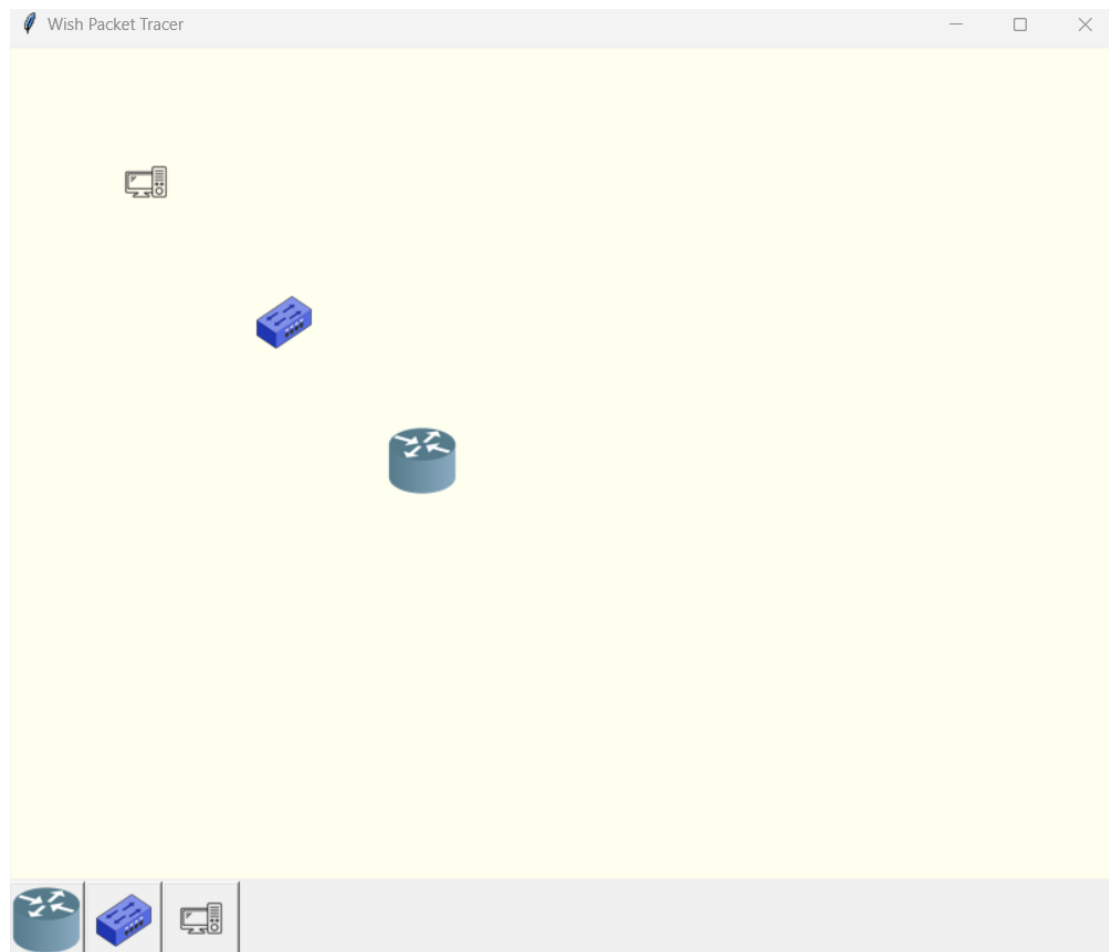
def drag_item(self, event):
    if self.current_item:
        dx = event.x - self.canvas.coords(self.current_item)[0]
        dy = event.y - self.canvas.coords(self.current_item)[1]
        self.canvas.move(self.current_item, dx, dy)

def load_and_resize_image(self, filename, width, height):
    image = Image.open(filename)
    image = image.resize((width, height))
    photo = ImageTk.PhotoImage(image)
    return photo

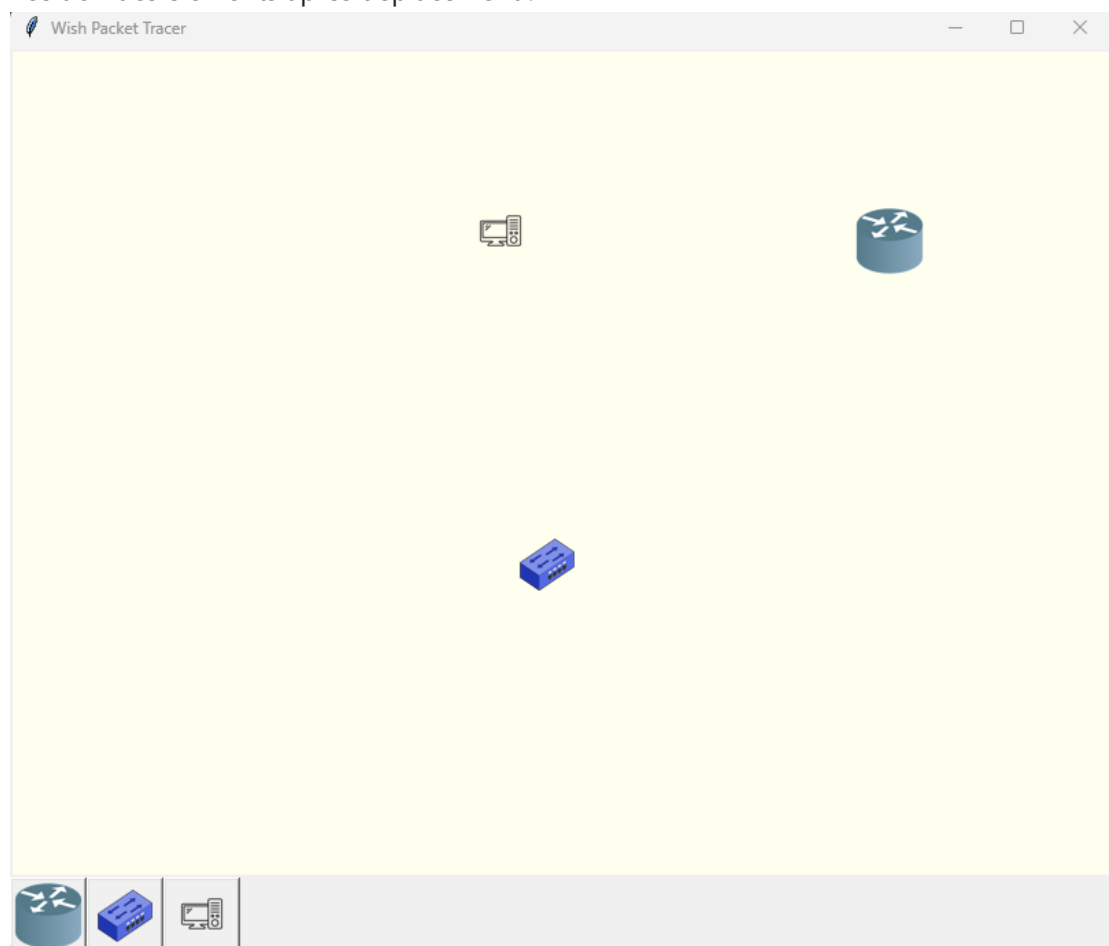
root = tk.Tk()
app = WishPacketTracer(root)
root.mainloop()

```

Position des éléments de base :



Position des éléments après déplacement :



Il faut également pouvoir créer routeurs / switches / pc à l'aide d'appuis sur le clavier et c'est pourquoi, j'ai ajouté un écouteur d'évènement "Key" qui va déclencher la méthode "key\_pressed".

"key\_pressed" est une méthode prenant en argument "event", qui signifie qu'une touche clavier a été tapée puis on y applique la méthode "char" pour retrouver la touche clavier qui a déclenché l'évènement (ex: on appuie sur la touche "A", "event.char" contiendra "a") et enfin on y applique la méthode "upper" pour s'assurer que la chaîne de caractère sera toujours en majuscules et on affecte le résultat à une variable "key".

Ainsi "key" sera une chaîne de caractère contenant la touche tapée (ex: on appuie sur la touche "S", key contiendra "S").

Puis on compare "key" à 3 chaînes de caractères "C" ou "S" ou "R" et qui vont créer à chacun un pc ou un switch ou un routeur. Si une autre touche a été tapée que ces 3 là alors rien ne se passera.

```
In [ ]: import tkinter as tk
from tkinter import simpledialog
from PIL import Image, ImageTk

class WishPacketTracer:
    def __init__(self, root):
        self.root = root
        self.root.title("Wish Packet Tracer")

        self.canvas = tk.Canvas(root, bg="ivory", width=800, height=600)
        self.canvas.pack(expand=tk.YES, fill=tk.BOTH)

        # Barre d'outils
        self.toolbar = tk.Frame(root)
        self.toolbar.pack(side=tk.LEFT, fill=tk.Y)

        # Boutons de la barre d'outils (utilisant des images redimensionnées avec
        self.image_router = self.load_and_resize_image("./img/Router.png", 50, 50)
        self.button_router = tk.Button(self.toolbar, image=self.image_router, command=self.create_router)
        self.button_router.pack(side=tk.LEFT)

        self.image_switch = self.load_and_resize_image("./img/switch.png", 50, 50)
        self.button_switch = tk.Button(self.toolbar, image=self.image_switch, command=self.create_switch)
        self.button_switch.pack(side=tk.LEFT)

        self.image_pc = self.load_and_resize_image("./img/pc.png", 50, 50)
        self.button_client = tk.Button(self.toolbar, image=self.image_pc, command=self.create_client)
        self.button_client.pack(side=tk.LEFT)

        # Evènements souris
        self.canvas.bind("<Button-1>", self.left_click)
        self.canvas.bind("<B1-Motion>", self.drag_item)
        # Evènements clavier
        self.root.bind("<Key>", self.key_pressed)

    def create_client(self):
        item = self.canvas.create_image(100, 100, image=self.image_pc, tags="cli
```

```

def create_switch(self):
    item = self.canvas.create_image(200, 200, image=self.image_switch, tags=
def create_router(self):
    item = self.canvas.create_image(300, 300, image=self.image_router, tags=

def left_click(self, event):
    item = self.canvas.find_closest(event.x, event.y)
    if item:
        self.current_item = item[0]

def drag_item(self, event):
    if self.current_item:
        dx = event.x - self.canvas.coords(self.current_item)[0]
        dy = event.y - self.canvas.coords(self.current_item)[1]
        self.canvas.move(self.current_item, dx, dy)

def load_and_resize_image(self, filename, width, height):
    image = Image.open(filename)
    image = image.resize((width, height))
    photo = ImageTk.PhotoImage(image)
    return photo

def key_pressed(self, event):
    key = event.char.upper()
    if key == "C":
        self.create_client()
    elif key == "S":
        self.create_switch()
    elif key == "R":
        self.create_router()

root = tk.Tk()
app = WishPacketTracer(root)
root.mainloop()

```

Il faut ensuite pouvoir modifier certaines propriétés d'un item sélectionné comme le nom et avoir la possibilité de lui attribuer à nouveau nom.

Du coup, d'abord je crée une liste instanciée "self.items" qui me permettra de stocker les éléments "dessinés" c'à-d les items présents sur mon canva.

Et je crée aussi une variable appelée "self.current\_item", qui va stocker l'élément que l'utilisateur a sélectionné.

```

...
self.items = [] # Liste pour stocker les éléments dessinés
self.current_item = None # Stocke l'élément actuellement sélectionné
...

```


J'ajoute aussi à mon programme un menu contextuel "self.context\_menu" avec la class "Menu" qui prend 2 argument :

- le premier est le widget dans lequel il se trouvera (ici root donc la fenetre principale)
- le second est "tearoff" qui est un argument permettant de gérer le détachement de mon menu contextuel (s'il est à 0 alors mon menu ne se détachera pas du tout et restera fixé dans ma fenetre principale, si par contre il a une autre valeur alors il

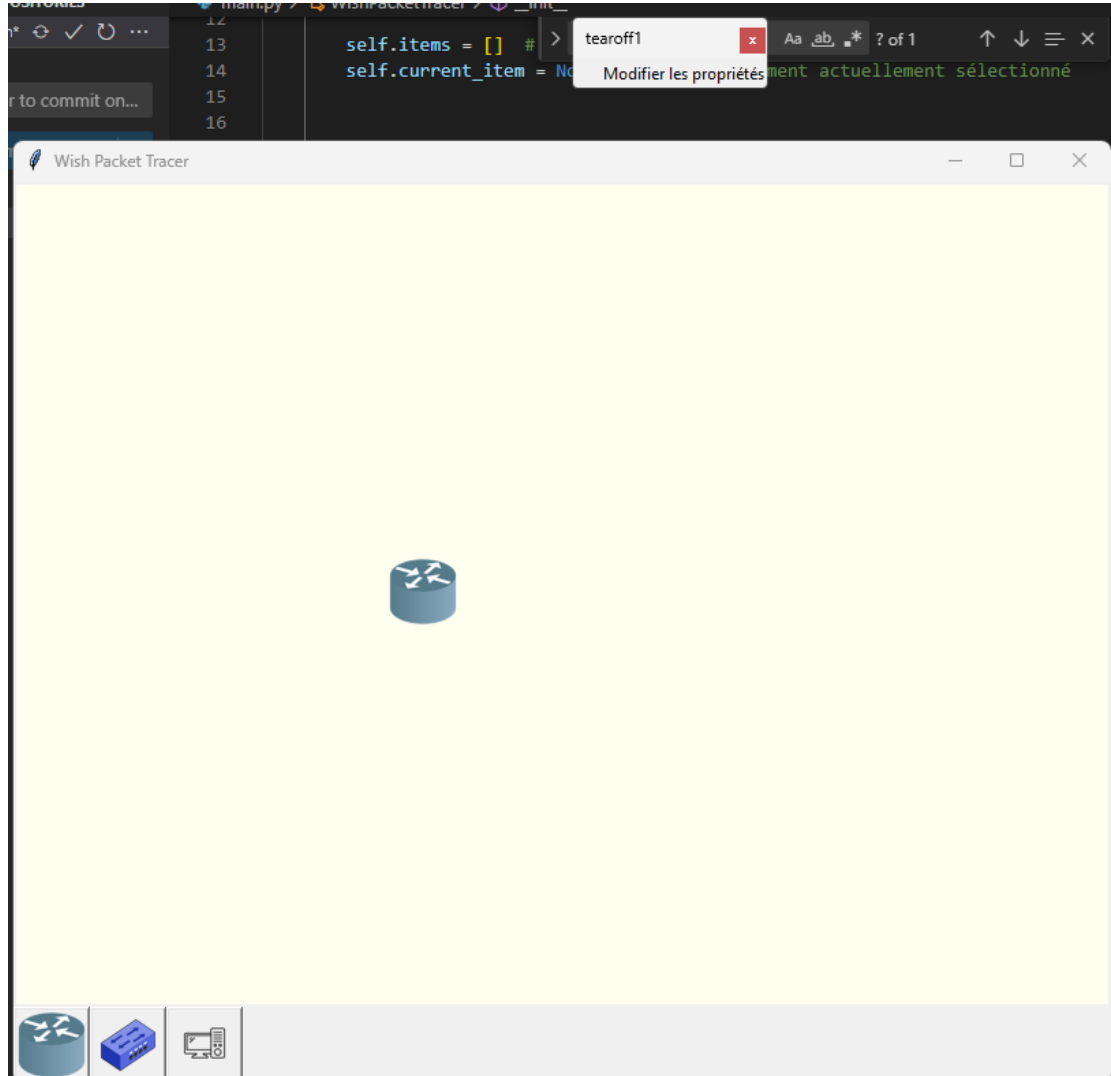


pourra se détacher c-à-d se trouver dans une fenêtre complètement indépendante de ma fenêtre principale).

Tearoff à 0 donc pas de détachement du menu

 No description has been provided for this image

Tearoff à 1 donc détachement du menu (il faut cliquer sur les "-----" pour détacher le menu) :

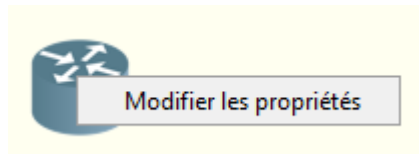


Et dans mon menu contextuel j'utilise la méthode "add\_command" qui permet d'ajouter une option dans mon menu qui prendra deux arguments, d'abord "label" pour afficher la description de l'option mais aussi le lancement de la méthode "edit\_proprietes" :

Sans aucune option dans mon menu (je suis obligé de mettre un tearoff pour pouvoir afficher le petit menu au clic droit sur l'item) :



Avec l'option "Modifier les propriétés" (et sans tearoff) :



# Menu contextuel

...

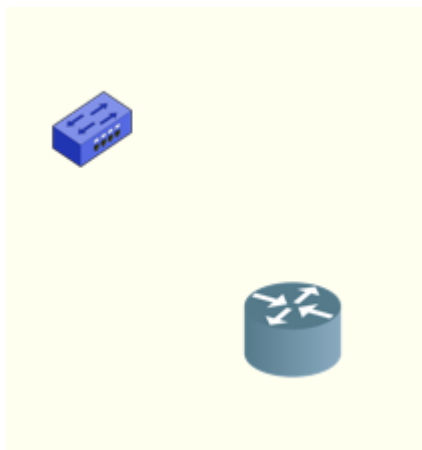
```
self.context_menu = tk.Menu(root, tearoff=0)
self.context_menu.add_command(label="Modifier les propriétés",
                              command=self.edit_proprietes)
...
```

Puis je rajoute dans les événements de souris l'écouteur d'événements "Button-3" permettant d'écouter si un clic droit a été fait et si oui qui va lancer la méthode "right\_click".

Ensuite j'ai ajouté dans mes méthodes de créations des items (routeurs/switchs/client), l'ajout dans ma liste "items" vu au-dessus d'un dictionnaire correspondant à l'item (en fait je les répertorie à leurs création par ordre de création ex: on crée un routeur, ce sera l'item 1 puis on crée un switch qui sera l'item 2 etc...) contenant ainsi plusieurs infos :

- item pour le numéro de l'item
- type pour le type d'item
- proprietes qui est un dictionnaire dans le dictionnaire et contiendra "name" pour le nom de l'item (par défaut "Router ou Switch ou Client") et "icon" pour son icône par défaut "C,R,S"

Ex: J'ai créé un routeur puis un switch, si j'affiche la liste "items" :



```
[{'item': 1, 'type': 'router', 'proprietes': {'name': 'Router', 'icon': 'R'}}, {'item': 2, 'type': 'switch', 'proprietes': {'name': 'Switch', 'icon': 'S'}}]
```

Ensuite, il faut donc créer la méthode "right\_click" qui ne se lance seulement si un clic droit à été fait sur un item prend un argument qui est "event" pour obtenir les coords du clic droit :

```
def right_click(self, event):
    item = self.canvas.find_closest(event.x, event.y)
    if item:
        self.current_item = item[0]
        self.context_menu.post(event.x_root, event.y_root)
```

J'affecte à la variable item l'élément le plus proche avec la méthode "find\_closest" à l'aide des coordonnées du clic, puis si il y a effectivement un élément dans item alors je défini l'élément sélectionné "self.current\_item" à le numéro de l'item (ex :si c'est le premier item créé que j'ai sélectionné alors il aura la valeur 1 )

Puis je fais apparaître mon menu contextuel créé au-dessus à l'aide de la méthode "post" qui va afficher au coordonnées du clic droit le menu contextuel.

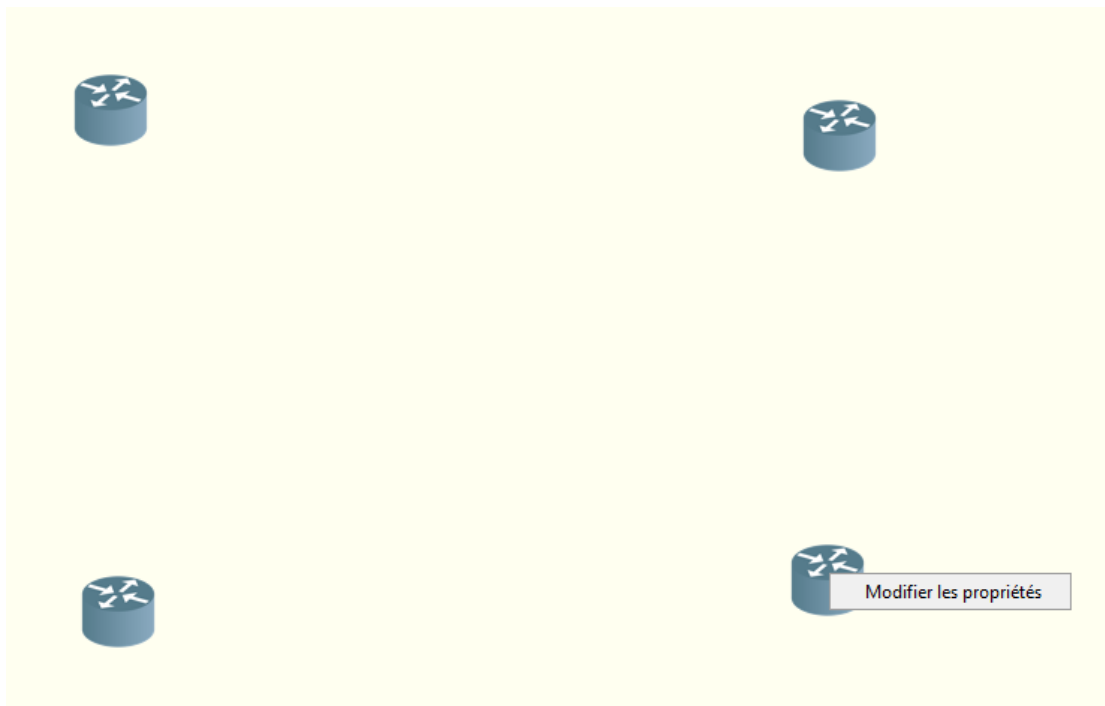
J'ai par la suite créé ma méthode "edit\_proprietes" :

```
def edit_proprietes(self):
    if self.current_item:
        item_type = self.canvas.gettags(self.current_item)[0]
        item = next(x for x in self.items if x["item"] ==
self.current_item )
        old_name=item["proprietes"]["name"]
        new_name = simpledialog.askstring("Modifier les
propriétés", "Nouveau nom:", initialvalue=old_name)
        if new_name:
            item["proprietes"]["name"] = new_name
```

Qui va chercher s'il y a un élément qui a été sélectionné donc s'il y a quelque chose dans "self.current\_item" et s'il y a bien un élément sélectionné alors :

- J'attribue à la variable item\_type, le type de l'item (si c'est un routeur, switch ou client) grâce à la méthode "gettags" qui va prendre le tag de l'image du canva, puisqu'à la création de l'image sur le canva, on lui a attribué un tag.
- Je parcours ensuite ma liste "items" contenant les items créés sur le canva et grâce à la fonction "next" en compréhension de liste qui parcourt et ,s'il trouve arrête de parcourir ma liste, sous une condition marquée par le "if" qui précise qu'il faut que le numéro de l'élément sélectionné corresponde bien à l'élément sélectionné (ex: si on créer 5 routeurs et que l'on cherche le 4e routeur alors il faut bien s'assurer que le routeur que l'on sélectionne soit le 4e routeur créé).

J'ai crée ci-dessous 4 routeurs que j'ai placé par ordre de création dans le sens des aiguilles d'une montre (le 1er routeur étant en haut à gauche) et dont j'ai sélectionné le 3e routeur



Ma méthode (en affichant "item" dans la méthode "edit\_proprietes") me donne bien l'item 3 correspondant au routeur 3 :

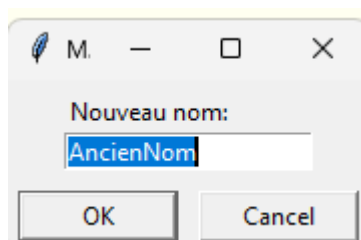
```
PS C:\Users\johnk\Desktop\r309-tp2-PhilippeLUU> & C:/Users/johnk/AppData/Local/Programs/Python/Python310/Python.exe -i  
{'item': 3, 'type': 'router', 'proprietes': {'name': 'Router', 'icon': 'R'}}
```

Puis je récupère l'ancien nom du routeur (par défaut est "Router")

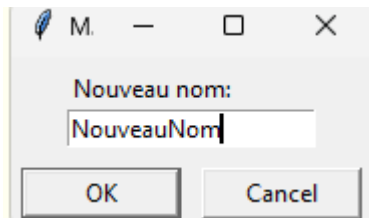
Et à l'aide du module "simplifiedialog" de tkinter et sa méthode "askstring", je crée une fenêtre dans laquelle, l'utilisateur peut attribuer un nouveau nom (initialvalue permet d'inscrire dans le champ à modifier, l'ancien nom de l'élément).

J'affecte la valeur du champ donc le nouveau nom inscrit par l'utilisateur à une variable new\_name et enfin je vérifie s'il un nouveau nom a été attribué et si oui alors je change dans le dictionnaire de l'élément sélectionné, le nom de l'élément.

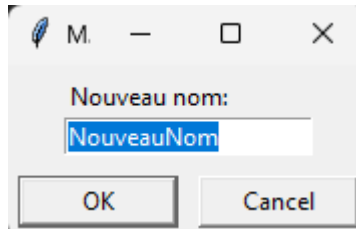
J'ai au préalable changé le nom d'un routeur pour vérifier si ma fenêtre de nouveau nom reprenait bien l'ancien nom de l'élément pour l'afficher (l'ancien nom étant "AncienNom") :



Puis je change le nom de cet élément en lui donnant "NouveauNom" comme nom :



Et si je revérifie en allant dans la fenêtre nouveau nom, je vois bien que le nom a été changé :



Et donc le code total à ce point :

```
In [ ]: import tkinter as tk
from tkinter import simpledialog
from PIL import Image, ImageTk

class WishPacketTracer:
    def __init__(self, root):
        self.root = root
        self.root.title("Wish Packet Tracer")

        self.canvas = tk.Canvas(root, bg="ivory", width=800, height=600)
        self.canvas.pack(expand=tk.YES, fill=tk.BOTH)

        self.items = [] # Liste pour stocker les éléments dessinés
        self.current_item = None # Stocke l'élément actuellement sélectionné

        # Barre d'outils
        self.toolbar = tk.Frame(root)
        self.toolbar.pack(side=tk.LEFT, fill=tk.Y)

        # Boutons de la barre d'outils (utilisant des images redimensionnées avec
        self.image_router = self.load_and_resize_image("./img/Router.png", 50, 50)
        self.button_router = tk.Button(self.toolbar, image=self.image_router, command=self.router_command)
        self.button_router.pack(side=tk.LEFT)

        self.image_switch = self.load_and_resize_image("./img/switch.png", 50, 50)
        self.button_switch = tk.Button(self.toolbar, image=self.image_switch, command=self.switch_command)
        self.button_switch.pack(side=tk.LEFT)

        self.image_pc = self.load_and_resize_image("./img/pc.png", 50, 50)
        self.button_client = tk.Button(self.toolbar, image=self.image_pc, command=self.client_command)
        self.button_client.pack(side=tk.LEFT)

        # Menu contextuel
        self.context_menu = tk.Menu(root, tearoff=0)
        self.context_menu.add_command(label="Modifier les propriétés", command=self.modify_properties)

        # Evénements souris
```

```

self.canvas.bind("<Button-1>", self.left_click)
self.canvas.bind("<Button-3>", self.right_click)
self.canvas.bind("<B1-Motion>", self.drag_item)
# Evènements clavier
self.root.bind("<Key>", self.key_pressed)

def create_client(self):
    item = self.canvas.create_image(100, 100, image=self.image_pc, tags="cli")
    self.items.append({"item": item, "type": "client", "proprietes": {"name": ""}})

def create_switch(self):
    item = self.canvas.create_image(200, 200, image=self.image_switch, tags="swi")
    self.items.append({"item": item, "type": "switch", "proprietes": {"name": ""}})

def create_router(self):
    item = self.canvas.create_image(300, 300, image=self.image_router, tags="rou")
    self.items.append({"item": item, "type": "router", "proprietes": {"name": ""}})

def left_click(self, event):
    item = self.canvas.find_closest(event.x, event.y)
    if item:
        self.current_item = item[0]

def right_click(self, event):
    item = self.canvas.find_closest(event.x, event.y)
    if item:
        self.current_item = item[0]
        self.context_menu.post(event.x_root, event.y_root)

def edit_proprietes(self):
    if self.current_item:
        item_type = self.canvas.gettags(self.current_item)[0]
        item = next(x for x in self.items if x["item"] == self.current_item)
        old_name = item["proprietes"]["name"]
        new_name = simpledialog.askstring("Modifier les propriétés", "Nouveau nom: ")
        if new_name:
            item["proprietes"]["name"] = new_name

def drag_item(self, event):
    if self.current_item:
        dx = event.x - self.canvas.coords(self.current_item)[0]
        dy = event.y - self.canvas.coords(self.current_item)[1]
        self.canvas.move(self.current_item, dx, dy)

def load_and_resize_image(self, filename, width, height):
    image = Image.open(filename)
    image = image.resize((width, height))
    photo = ImageTk.PhotoImage(image)
    return photo

def key_pressed(self, event):
    key = event.char.upper()
    if key == "C":
        self.create_client()
    elif key == "S":
        self.create_switch()

```

```

        elif key == "R":
            self.create_router()

root = tk.Tk()
app = WishPacketTracer(root)
root.mainloop()

```

Ensuite, il faut pouvoir supprimer les items à l'aide du clic droit, pour ce faire, j'ai ajouté une nouvelle option lors du clic droit d'un item qui est "Supprimer" et va lancer la méthode "delete\_item" :

```

self.context_menu.add_command(label="Supprimer",
                              command=self.delete_item)

...

def delete_item(self):
    if self.current_item:
        item = next(x for x in self.items if x["item"] ==
self.current_item)
        self.items.remove(item)
        self.canvas.delete(self.current_item)
    ...

```

La méthode delete\_item va d'abord vérifier s'il un item a été sélectionné donc en regardant s'il y a une valeur dans "self.current\_item" et puis il va encore parcourir la liste des items qui contiendra un dictionnaire avec comme clés : pour faire correspondre le numéro de l'item sélectionné avec l'item sélectionné (ex: si on veut supprimer le 2e routeur créé, il faut vérifier que le routeur sur lequel on a fait clic droit porte bien le numéro 2).

- "item" et sa valeur est le numéro de création de l'élément
- "type" et sa valeur est un string s'autodécrivant
- "proprietes" un autre dico à l'intérieur qui contiendra les clés "name" pour le nom de l'élément

Et une fois trouvé, il va l'affecter à une variable "item" qui va nous servir à retirer les infos de l'élément sélectionné (si on supprime le switch 2 il faut que les infos du switch 2 seulement soient retirées de la liste des items).

On retire également l'image de l'élément sur le canva à l'aide de la méthode delete.

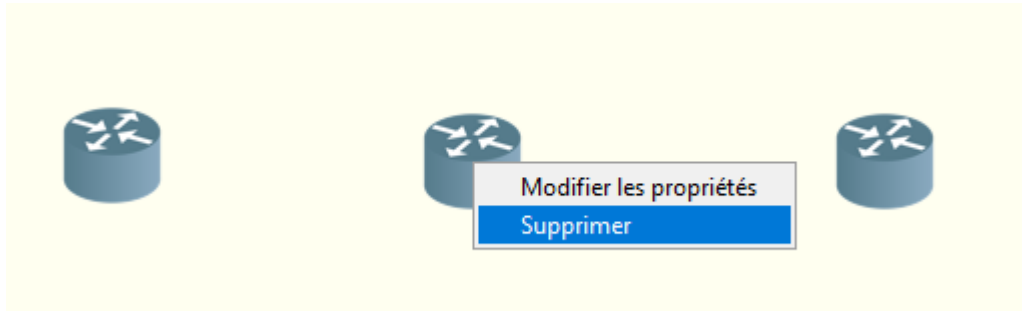
Ex: J'ai créé 3 routeurs et j'affiche la liste des items à chaque clic droit ;



dans la liste des items on a donc 3 dictionnaires avec items "1,2 et 3" ;

```
[{'item': 1, 'type': 'router', 'proprietes': {'name': 'Router', 'icon': 'R'}}, {'item': 2, 'type': 'router', 'proprietes': {'name': 'Router', 'icon': 'R'}}, {'item': 3, 'type': 'router', 'proprietes': {'name': 'Router', 'icon': 'R'}}]
```

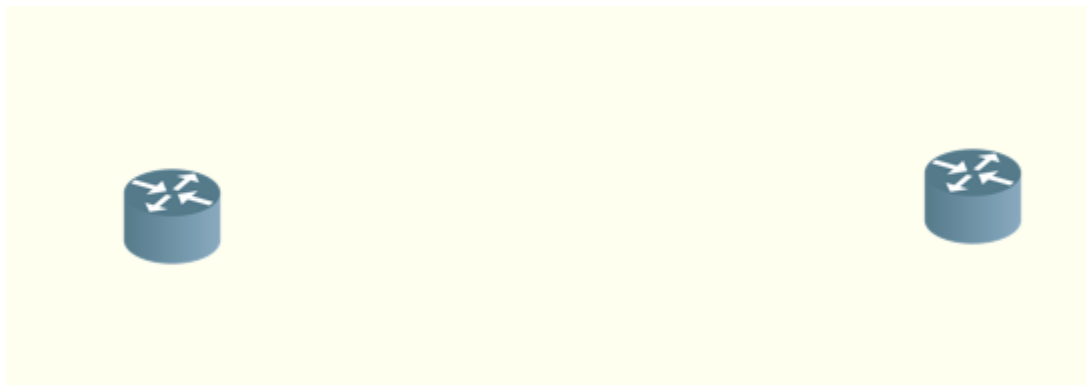
Puis je supprime le second item :



Et j'ai bien le dico du second routeurs qui a été retiré de la liste des items (passant de l'item 1 à l'item 3) :

```
[{'item': 1, 'type': 'router', 'proprietes': {'name': 'Router', 'icon': 'R'}}, {'item': 3, 'type': 'router', 'proprietes': {'name': 'Router', 'icon': 'R'}}]
```

Et j'ai bien seulement 2 routeurs sur mon canva :



Donc le code à ce point est :

```
In [ ]: import tkinter as tk
from tkinter import simpledialog
from PIL import Image, ImageTk

class WishPacketTracer:
    def __init__(self, root):
        self.root = root
        self.root.title("Wish Packet Tracer")

        self.canvas = tk.Canvas(root, bg="ivory", width=800, height=600)
        self.canvas.pack(expand=tk.YES, fill=tk.BOTH)

        self.items = [] # Liste pour stocker Les éléments dessinés
        self.current_item = None # Stocke l'élément actuellement sélectionné

        # Barre d'outils
        self.toolbar = tk.Frame(root)
        self.toolbar.pack(side=tk.LEFT, fill=tk.Y)

        # Boutons de la barre d'outils (utilisant des images redimensionnées avec
        self.image_router = self.load_and_resize_image("./img/Router.png", 50, 50)
        self.button_router = tk.Button(self.toolbar, image=self.image_router, command=self.delete_item)
        self.button_router.pack(side=tk.LEFT)
```



```

self.image_switch = self.load_and_resize_image("./img/switch.png", 50, 50)
self.button_switch = tk.Button(self.toolbar, image=self.image_switch, command=self.button_switch.pack(side=tk.LEFT))

self.image_pc = self.load_and_resize_image("./img/pc.png", 50, 50)
self.button_client = tk.Button(self.toolbar, image=self.image_pc, command=self.button_client.pack(side=tk.LEFT))

# Menu contextuel
self.context_menu = tk.Menu(root, tearoff=0)
self.context_menu.add_command(label="Modifier les propriétés", command=self.edit_proprietes)
self.context_menu.add_command(label="Supprimer", command=self.delete_item)

# Evènements souris
self.canvas.bind("<Button-1>", self.left_click)
self.canvas.bind("<Button-3>", self.right_click)
self.canvas.bind("<B1-Motion>", self.drag_item)
# Evènements clavier
self.root.bind("<Key>", self.key_pressed)

def create_client(self):
    item = self.canvas.create_image(100, 100, image=self.image_pc, tags="client")
    self.items.append({"item": item, "type": "client", "proprietes": {"name": "Client 1", "x": 100, "y": 100}})

def create_switch(self):
    item = self.canvas.create_image(200, 200, image=self.image_switch, tags="switch")
    self.items.append({"item": item, "type": "switch", "proprietes": {"name": "Switch 1", "x": 200, "y": 200}})

def create_router(self):
    item = self.canvas.create_image(300, 300, image=self.image_router, tags="router")
    self.items.append({"item": item, "type": "router", "proprietes": {"name": "Router 1", "x": 300, "y": 300}})

def left_click(self, event):
    item = self.canvas.find_closest(event.x, event.y)
    if item:
        self.current_item = item[0]

def right_click(self, event):
    item = self.canvas.find_closest(event.x, event.y)
    if item:
        self.current_item = item[0]
        self.context_menu.post(event.x_root, event.y_root)

def edit_proprietes(self):
    if self.current_item:
        item_type = self.canvas.gettags(self.current_item)[0]
        # print(self.items)
        item = next(x for x in self.items if x["item"] == self.current_item)
        old_name=item["proprietes"]["name"]
        new_name = simplifiedialog.askstring("Modifier les propriétés", "Nouveau nom: ")
        if new_name:
            item["proprietes"]["name"] = new_name

def drag_item(self, event):

```

```

        if self.current_item:
            dx = event.x - self.canvas.coords(self.current_item)[0]
            dy = event.y - self.canvas.coords(self.current_item)[1]
            self.canvas.move(self.current_item, dx, dy)

    def load_and_resize_image(self, filename, width, height):
        image = Image.open(filename)
        image = image.resize((width, height))
        photo = ImageTk.PhotoImage(image)
        return photo

    def key_pressed(self, event):
        key = event.char.upper()
        if key == "C":
            self.create_client()
        elif key == "S":
            self.create_switch()
        elif key == "R":
            self.create_router()

    def delete_item(self):
        if self.current_item:
            item = next(x for x in self.items if x["item"] == self.current_item)
            self.items.remove(item)
            self.canvas.delete(self.current_item)

root = tk.Tk()
app = WishPacketTracer(root)
root.mainloop()

```

Enfin, il faut que l'on puisse changer l'icône de notre élément, et pour cela j'ai fais :

```

from tkinter import filedialog
...

self.item_images = {}
...
self.context_menu.add_command(label="Changer l'icône",
command=self.change_icon) # Ajouter l'option pour changer l'icône
...

def change_icon(self):
    if self.current_item:
        self.filename = filedialog.askopenfilename(filetypes=
[("Image files", "*.png")]) # Boîte de dialogue pour sélectionner un
fichier PNG
        if self.filename:

self.photo=self.load_and_resize_image(self.filename,50,50)
        self.item_images[self.current_item] = self.photo
        self.canvas.itemconfig(self.current_item,
image=self.photo) # Changer l'image de l'élément actuellement
sélectionné

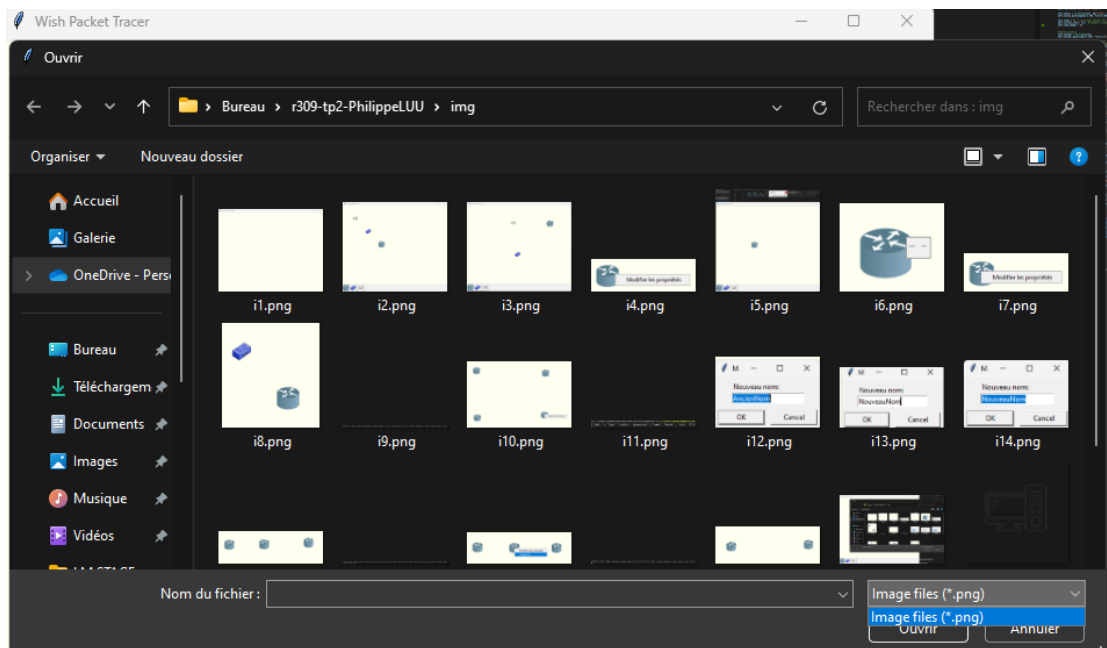
```

D'abord j'importe un module présent dans tkinter qui est "filedialog" qui permet la sélection de fichier dans une fenêtre.

Ensuite je crée un dictionnaire instancié "item\_image" qui contiendra pour le numéro de l'item correspondant (ex: si le 1er élément crée un routeur alors l'item 1 sera le routeur crée) une image en format tkinter (converti depuis PNG par "ImageTk.PhotoImage") et que j'ai redimensionné à l'aide de ma méthode "load\_and\_resize\_image".

Puis j'ajoute l'option "Changer l'icône" lors d'un clic droit sur un élément, qui va lancer une méthode "change\_icon".

La méthode "change\_icon", vérifie d'abord si un élément à été sélectionné, puis il va utiliser la méthode "askopenfilename" du module "filedialog" pour permettre l'affichage d'une fenêtre à l'utilisateur lui permettant de sélectionner une image au format PNG seulement :



L'image sélectionnée va donc être affectée à une variable instanciée "filename", et ensuite on refait une vérification pour savoir s'il y a quelque chose dans "filename" (si non alors rien ne se passe) et si oui alors on utilise une méthode de notre classe qui est "load\_and\_resize\_image" et qui va ainsi prendre en arguments : l'image puis les dimensions en hauteur et largeur de l'image (ici 50 pour les deux).

Puis, on insère dans notre dico "item\_images", d'abord la clé de l'image qui correspond à au numéro de création de l'élément (1 pour l'item 1 par exemple) et dont la valeur sera l'image PNG au format tkinter.

Ci-dessous, j'ai changé l'image de deux éléments différents et on voit bien le dico se remplir :

```
{1: <PIL.ImageTk.PhotoImage object at 0x000001ECD41EA7D0>}
```

```
{1: <PIL.ImageTk.PhotoImage object at 0x000001ECD41EA7D0>, 2: <PIL.ImageTk.PhotoImage object at 0x000001ECD8015CD0>}
```

Et enfin, on modifie l'image de l'élément sélectionné dans le canva en la remplaçant par l'image chargée et redimensionnée.



Bien sûr, les images d'origines, n'étant pas toutes de mêmes dimensions et de mêmes qualité, lors du redimensionnement, il se peut que les images se retrouvent déformées. Mais si aucun redimensionnement n'était fait, alors les images prendraient toute la fenêtre.

- J'ai aussi rajouté le fait de compter les items ("self.nb\_router" par exemple) qui pour l'utilisateur permettra de se repérer si jamais il ne cherche pas à renommer les items (ex: router0 par défaut, puis router1, router2 etc...)

Et donc le code complet :

```
In [ ]: import tkinter as tk
from tkinter import simpledialog
from PIL import Image, ImageTk
from tkinter import filedialog

class WishPacketTracer:
    def __init__(self, root):
        self.root = root
        self.root.title("Wish Packet Tracer")

        self.canvas = tk.Canvas(root, bg="ivory", width=800, height=600)
        self.canvas.pack(expand=tk.YES, fill=tk.BOTH)

        self.items = [] # Liste pour stocker Les éléments dessinés
        self.current_item = None # Stocke l'élément actuellement sélectionné
        self.item_images = {}

        self.nb_router=0
        self.nb_switch=0
        self.nb_client=0

        # Barre d'outils
        self.toolbar = tk.Frame(root)
        self.toolbar.pack(side=tk.LEFT, fill=tk.Y)

        # Boutons de la barre d'outils (utilisant des images redimensionnées avec
        self.image_router = self.load_and_resize_image("./img/Router.png", 50, 50)
        self.button_router = tk.Button(self.toolbar, image=self.image_router, command=self.add_router)
        self.button_router.pack(side=tk.LEFT)

        self.image_switch = self.load_and_resize_image("./img/switch.png", 50, 50)
        self.button_switch = tk.Button(self.toolbar, image=self.image_switch, command=self.add_switch)
        self.button_switch.pack(side=tk.LEFT)
```

```

self.image_pc = self.load_and_resize_image("./img/pc.png", 50, 50)
self.button_client = tk.Button(self.toolbar, image=self.image_pc, command=self.button_client.pack(side=tk.LEFT)

# Menu contextuel
self.context_menu = tk.Menu(root, tearoff=0)
self.context_menu.add_command(label="Modifier les propriétés", command=self.modify_properties)
self.context_menu.add_command(label="Changer l'icône", command=self.change_image)
self.context_menu.add_command(label="Supprimer", command=self.delete_item)

# Evénements souris
self.canvas.bind("<Button-1>", self.left_click)
self.canvas.bind("<Button-3>", self.right_click)
self.canvas.bind("<B1-Motion>", self.drag_item)
# Evénements clavier
self.root.bind("<Key>", self.key_pressed)

def create_client(self):
    item = self.canvas.create_image(100, 100, image=self.image_pc, tags="client")
    self.items.append({"item": item, "type": "client", "proprietes": {"name": "Client", "x": 100, "y": 100}})
    self.nb_client+=1

def create_switch(self):
    item = self.canvas.create_image(200, 200, image=self.image_switch, tags="switch")
    self.items.append({"item": item, "type": "switch", "proprietes": {"name": "Switch", "x": 200, "y": 200}})
    self.nb_switch+=1

def create_router(self):
    item = self.canvas.create_image(300, 300, image=self.image_router, tags="router")
    self.items.append({"item": item, "type": "router", "proprietes": {"name": "Router", "x": 300, "y": 300}})
    self.nb_router+=1

def left_click(self, event):
    item = self.canvas.find_closest(event.x, event.y)
    if item:
        self.current_item = item[0]

def right_click(self, event):
    item = self.canvas.find_closest(event.x, event.y)
    if item:
        self.current_item = item[0]
        self.context_menu.post(event.x_root, event.y_root)

def edit_proprietes(self):
    if self.current_item:
        item_type = self.canvas.gettags(self.current_item)[0]
        print(self.items)
        item = next(x for x in self.items if x["item"] == self.current_item)
        old_name=item["proprietes"]["name"]
        new_name = simplifiedialog.askstring("Modifier les propriétés", "Nouveau nom: ")
        if new_name:
            item["proprietes"]["name"] = new_name

def drag_item(self, event):
    if self.current_item:
        dx = event.x - self.canvas.coords(self.current_item)[0]

```



```

def lien(self, item1, item2):
    x1, y1 = self.canvas.coords(item1)
    x2, y2 = self.canvas.coords(item2)
    line = self.canvas.create_line(x1, y1, x2, y2, fill="black",
width=2) # Dessiner une ligne entre les éléments sélectionnés
    self.links.append(line) # Ajouter le lien à la liste des liens

    # Créer un menu contextuel pour les liens avec uniquement
    l'option "Supprimer"
    link_menu = tk.Menu(self.root, tearoff=0)
    link_menu.add_command(label="Supprimer", command=lambda:
self.delete_link(line))
    self.canvas.tag_bind(line, "<Button-3>", lambda event,
menu=link_menu: self.affiche_menu_lien(event, menu))

def affiche_menu_lien(self, event, menu):
    menu.post(event.x_root, event.y_root)

def delete_link(self, line):
    self.links.remove(line) # Retirer le lien de la liste des
liens
    self.canvas.delete(line) # Supprimer graphiquement le lien
...
def middle_click(self, event):
    item = self.canvas.find_closest(event.x, event.y)
    if item:
        if not self.start_link: # Premier élément sélectionné
            self.start_link = item[0]
        else: # Deuxième élément sélectionné
            self.end_link = item[0]
            self.lien(self.start_link, self.end_link)
            self.start_link = None
    ...

```

J'ai d'abord rajouté une liste "self.items" pour les liens afin de les répertorier et donc de prévoir la fonctionnalité de suppression des liens. Puis j'ajoute aussi une nouvelle variable "start\_link" qui servira à savoir s'il y a un élément déjà sélectionné ou si c'est le second. Je rajoute un nouvel écouteur d'évènement qui permet de détecter si un clic molette est fait et si oui va lancer une méthode "middle\_click".

La méthode "middle\_click" va récupérer le numéro de création de l'item (si c'est l'item 1 alors c'est le premier élément créé du canva). Et ensuite il vérifie s'il y a bien un élément dans item donc que l'utilisateur a bien choisi un élément ( la méthode find\_closest va chercher l'élément le plus proche s'il y a quelque chose sur le canva, s'il n'y a rien alors aucun lien ne sera créé).

Et donc on vérifie s'il y a quelque chose dans la variable "start\_link" c'est à dire si c'est la 1ère sélection ou la seconde, et si c'est la première alors on lui affecte l'élément du canva. Si c'est la seconde sélection alors on donne à la variable "end\_link" l'élément sélectionné.

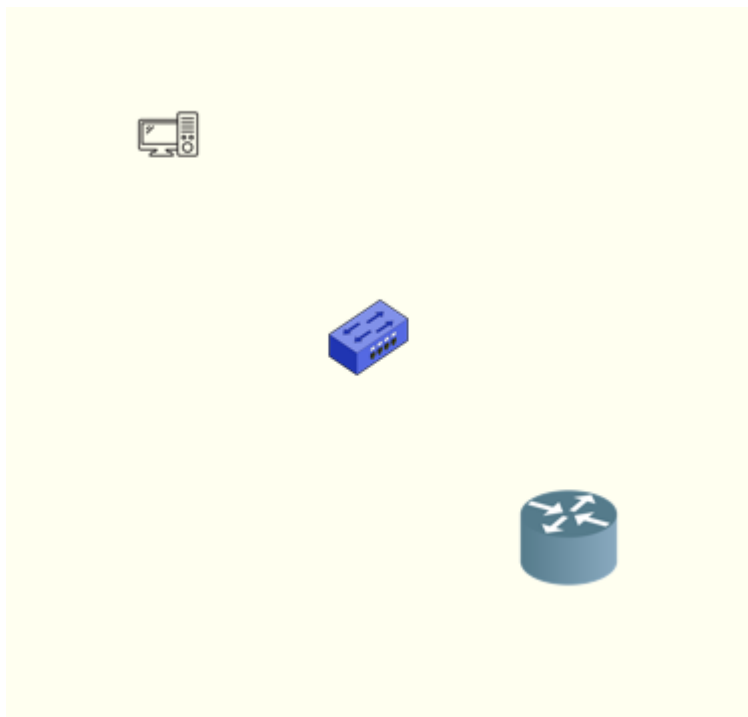
Et on utilise une autre méthode nommée "lien" avec comme argument (le 1er élément et le second) avant de redéfinir "start\_link" à None.

La fonction lien quand à elle, prend les coordonnées des deux éléments sur le canva. Puis il va tracer un trait suivant les coordonnées en abscisses et ordonnées des points A et B. Et on affecte l'identifiant du lien à une variable line (cet ID nous servira à modifier ultérieurement ce lien précis et non pas un autre si on souhaite par exemple le supprimer) que l'on va ajouter à la liste "links" créée précédemment.

Puis on va également mettre en place un menu contextuel (donc la boîte qui s'affiche lors d'un clic droit sur le lien) qui est propre aux liens que l'on va affecter à une variable "link\_menu". Et ce menu contextuel ne comportera logiquement qu'une seule option "Supprimer", et va lancer une méthode "delete\_link" en prenant en argument l'ID du lien que l'on a sélectionné (lambda permet de créer une fonction anonyme, ici sans argument qui ne sert qu'à instancier la méthode "delete\_link"). Puis on lie aussi un évènement sur le canva à un "tag" grâce à la méthode "tag\_bind" qui va au clic droit sur tous les éléments "line" (donc clic droit uniquement sur les liens), afficher le menu contextuel des liens permettant de supprimer les liens grâce à la méthode "affiche\_lien\_menu" qui ne sert qu'à afficher le menu contextuel.

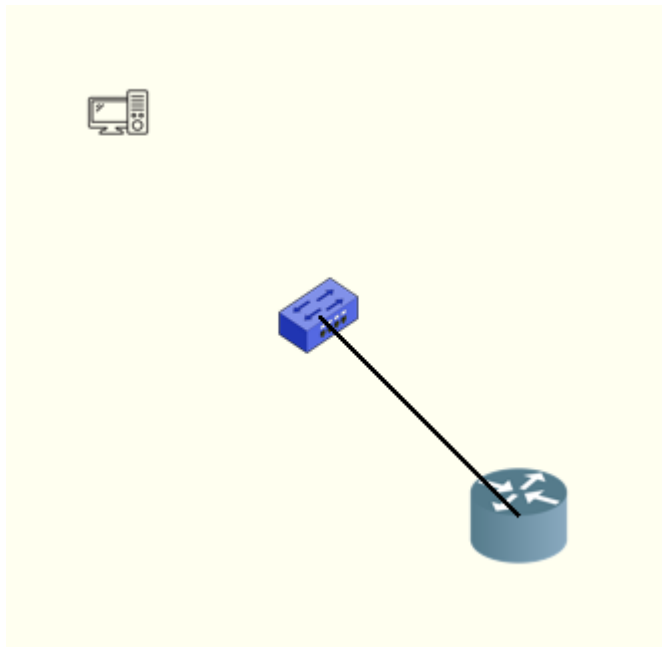
La méthode "delete\_link" servant à supprimer un lien, va prendre en argument l'ID d'une ligne et va aller chercher dans la liste des liens contenant l'ID de tous les liens (les rendant donc unique et manipulable indépendamment de chacun) et va retirer le lien de la liste grace à la méthode "remove" et puis va retirer le lien graphiquement grâce à la méthode "delete" sur le canva.

Ex: Je créé 3 éléments sur mon canva dans cet ordre : Routeur - Switch - Client



Je relie le routeur et le switch et affiche la liste des liens mais aussi l'ID des éléments sélectionnés :

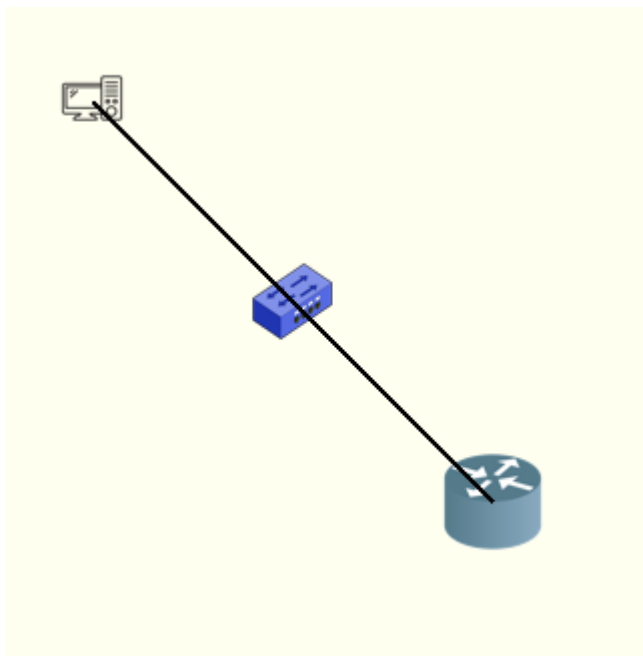




On voit bien que le 1er élément sélectionne porte l'ID d'item "1" car j'ai crée le routeur en premier puis en re cliquant sur le switch qui est le second item créer, j'affiche d'abord la liste des liens et donc le lien créé porte l'ID lien "4" et l'ID du switch est bien le "2"

```
PS C:\
1
[4]
2
```

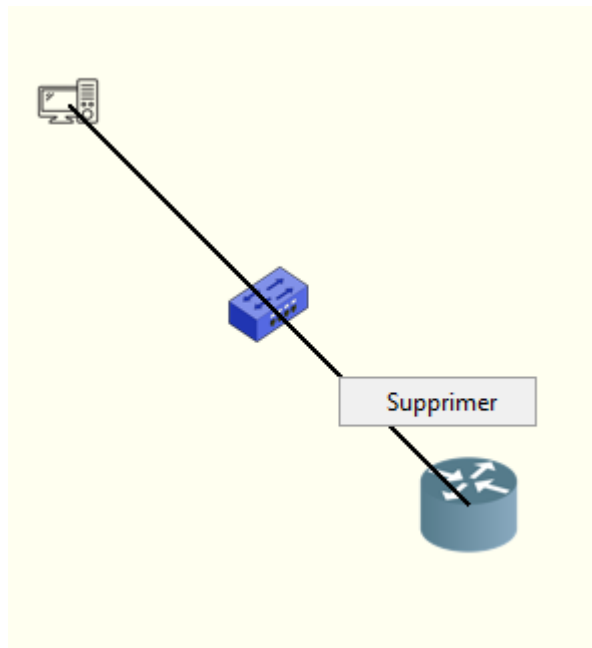
Ensuite je lie le switch au client :



Et dans mon terminal, j'ai bien l'ID item "2" donc le switch puis en le liant au client j'ai l'ID lien "5" qui s'est ajouté à ma liste et puisque j'ai cliqué sur le client, j'ai bien l'ID item "3" affiché car c'est le 3e élément créé.

```
2
[4, 5]
3
```

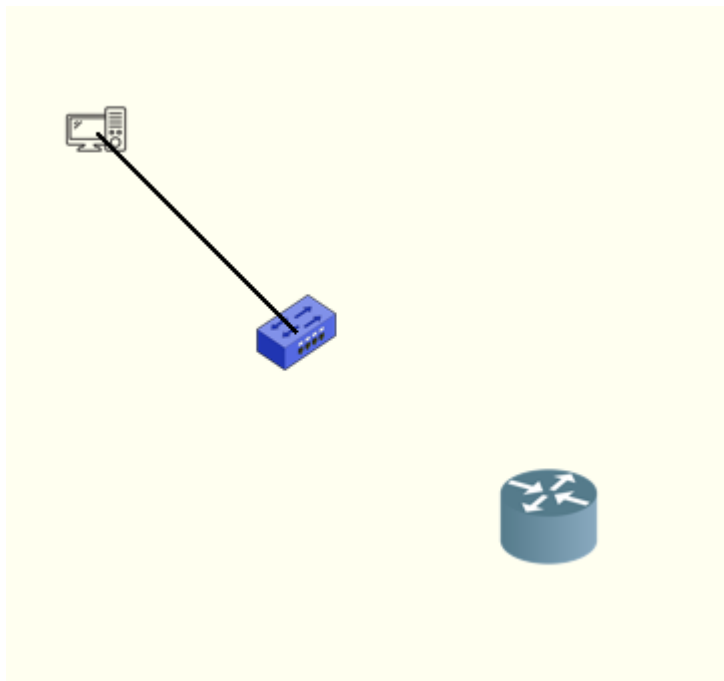
Enfin je fais un clic droit sur le lien Routeur - Switch (il faut bien viser le lien pour avoir le menu correspondant) :



Et je fais "Supprimer", je vois bien en affichant ma liste des liens, que je n'ai plus le lien "4" correspondant au lien que je viens de supprimer :



Enfin, graphiquement le lien s'enlève aussi :



Et donc le code complet pour l'exercice 1, disponible également dans exo1.py

```
In [ ]: import tkinter as tk
from tkinter import simpledialog
from PIL import Image, ImageTk
from tkinter import filedialog
```

```

class WishPacketTracer:
    def __init__(self, root):
        self.root = root
        self.root.title("Wish Packet Tracer")

        self.canvas = tk.Canvas(root, bg="ivory", width=800, height=600)
        self.canvas.pack(expand=tk.YES, fill=tk.BOTH)

        self.items = [] # Liste pour stocker Les éléments dessinés
        self.current_item = None # Stocke l'élément actuellement sélectionné
        self.item_images = {}
        self.links=[]

        self.start_link = None # Initialisation de self.start_link pour gérer l

        self.nb_router=0
        self.nb_switch=0
        self.nb_client=0

        # Barre d'outils
        self.toolbar = tk.Frame(root)
        self.toolbar.pack(side=tk.LEFT, fill=tk.Y)

        # Boutons de la barre d'outils (utilisant des images redimensionnées avec
        self.image_router = self.load_and_resize_image("./img/Router.png", 50, 50)
        self.button_router = tk.Button(self.toolbar, image=self.image_router, command=self.add_router)
        self.button_router.pack(side=tk.LEFT)

        self.image_switch = self.load_and_resize_image("./img/switch.png", 50, 50)
        self.button_switch = tk.Button(self.toolbar, image=self.image_switch, command=self.add_switch)
        self.button_switch.pack(side=tk.LEFT)

        self.image_pc = self.load_and_resize_image("./img/pc.png", 50, 50)
        self.button_client = tk.Button(self.toolbar, image=self.image_pc, command=self.add_client)
        self.button_client.pack(side=tk.LEFT)

        # Menu contextuel
        self.context_menu = tk.Menu(root, tearoff=0)
        self.context_menu.add_command(label="Modifier les propriétés", command=self.modify_properties)
        self.context_menu.add_command(label="Changer l'icône", command=self.change_icon)
        self.context_menu.add_command(label="Supprimer", command=self.delete_item)

        # Evènements souris
        self.canvas.bind("<Button-1>", self.left_click)
        self.canvas.bind("<Button-3>", self.right_click)
        self.canvas.bind("<B1-Motion>", self.drag_item)
        self.canvas.bind("<Button-2>", self.middle_click)

        # Evènements clavier
        self.root.bind("<Key>", self.key_pressed)

    def create_client(self):
        item = self.canvas.create_image(100, 100, image=self.image_pc, tags="client")
        self.items.append({"item": item, "type": "client", "proprietes": {"name": "Client", "nb": 1}})
        self.nb_client+=1

    def create_switch(self):

```

```

        item = self.canvas.create_image(200, 200, image=self.image_switch, tags=
self.items.append({"item": item, "type": "switch", "proprietes": {"name"
self.nb_switch+=1

def create_router(self):
    item = self.canvas.create_image(300, 300, image=self.image_router, tags=
self.items.append({"item": item, "type": "router", "proprietes": {"name"
self.nb_router+=1

def lien(self, item1, item2):
    x1, y1 = self.canvas.coords(item1)
    x2, y2 = self.canvas.coords(item2)
    line = self.canvas.create_line(x1, y1, x2, y2, fill="black", width=2) #
self.links.append(line) # Ajouter Le lien à la Liste des Liens
    print(self.links)

    # Créer un menu contextuel pour Les Liens avec uniquement L'option "Supp
    link_menu = tk.Menu(self.root, tearoff=0)
    link_menu.add_command(label="Supprimer", command=lambda: self.delete_lin
    self.canvas.tag_bind(line, "<Button-3>", lambda event, menu=link_menu: s

def affiche_menu_lien(self, event, menu):
    menu.post(event.x_root, event.y_root)

def delete_link(self, line):
    self.links.remove(line) # Retirer Le Lien de La Liste des Liens
    print(self.links)
    self.canvas.delete(line) # Supprimer graphiquement Le Lien

def left_click(self, event):
    item = self.canvas.find_closest(event.x, event.y)
    if item:
        self.current_item = item[0]

def right_click(self, event):
    item = self.canvas.find_closest(event.x, event.y)
    if item:
        self.current_item = item[0]
        self.context_menu.post(event.x_root, event.y_root)

def middle_click(self, event):
    item = self.canvas.find_closest(event.x, event.y)
    if item:
        if not self.start_link: # Premier élément sélectionné
            self.start_link = item[0]
            print(self.start_link)
        else: # Deuxième élément sélectionné
            self.end_link = item[0]
            self.lien(self.start_link, self.end_link)
            self.start_link = None
            print(self.end_link)

def edit_proprietes(self):
    if self.current_item:
        item_type = self.canvas.gettags(self.current_item)[0]
        item = next(x for x in self.items if x["item"] == self.current_item
        old_name=item["proprietes"]["name"]
        new_name = simpledialog.askstring("Modifier les propriétés", "Nouvea
        if new_name:

```

```

        item["proprietes"]["name"] = new_name

    def drag_item(self, event):
        if self.current_item:
            dx = event.x - self.canvas.coords(self.current_item)[0]
            dy = event.y - self.canvas.coords(self.current_item)[1]
            self.canvas.move(self.current_item, dx, dy)

    def load_and_resize_image(self, filename, width, height):
        image = Image.open(filename)
        image = image.resize((width, height))
        photo = ImageTk.PhotoImage(image)
        return photo

    def change_icon(self):
        if self.current_item:
            # Boîte de dialogue pour sélectionner un fichier PNG
            self.filename = filedialog.askopenfilename(filetypes=[("Image files",
                                                                    ".png")])
            if self.filename:
                self.photo=self.load_and_resize_image(self.filename,50,50)
                self.item_images[self.current_item] = self.photo
                # Changer l'image de l'élément actuellement sélectionné
                self.canvas.itemconfig(self.current_item, image=self.photo)

    def key_pressed(self, event):
        key = event.char.upper()
        if key == "C":
            self.create_client()
        elif key == "S":
            self.create_switch()
        elif key == "R":
            self.create_router()

    def delete_item(self):
        if self.current_item:
            item = next(x for x in self.items if x["item"] == self.current_item)
            self.items.remove(item)
            self.canvas.delete(self.current_item)

root = tk.Tk()
app = WishPacketTracer(root)
root.mainloop()

```

## Exercice 2 Des traits bien droits !

Faites en sorte que les lignes ne puissent être que horizontales ou verticales lorsque l'utilisateur maintient la touche CTRL enfoncée.

In [ ]:

## Exercice 3 Un client exigeant ...

Le client, satisfait par votre travail précédant, demande à ce que vous alliez plus loin. Il souhaite maintenant que les items aient des ports sur lesquels votre souris sera aimantée lorsque vous dessinez des traits. Le nombre de ports des routeurs et des switches sera paramétrable de 1 à 4. Les clients, eux, auront toujours un seul port

```
In [ ]: import tkinter as tk
from tkinter import simpledialog
from PIL import Image, ImageTk
from tkinter import filedialog

class WishPacketTracer:
    def __init__(self, root):
        self.root = root
        self.root.title("Wish Packet Tracer")

        self.canvas = tk.Canvas(root, bg="ivory", width=800, height=600)
        self.canvas.pack(expand=tk.YES, fill=tk.BOTH)

        self.items = [] # Liste pour stocker les éléments dessinés
        self.current_item = None # Stocke l'élément actuellement sélectionné
        self.item_images = {}
        self.links=[]

        self.start_link = None # Initialisation de self.start_link pour gérer l

        self.nb_router=0
        self.nb_switch=0
        self.nb_client=0

        self.ports = {} # Dictionnaire pour stocker les ports des appareils

        # Barre d'outils
        self.toolbar = tk.Frame(root)
        self.toolbar.pack(side=tk.LEFT, fill=tk.Y)

        # Boutons de la barre d'outils (utilisant des images redimensionnées avec
        self.image_router = self.load_and_resize_image("./img/Router.png", 50, 50)
        self.button_router = tk.Button(self.toolbar, image=self.image_router, command=self.add_router)
        self.button_router.pack(side=tk.LEFT)

        self.image_switch = self.load_and_resize_image("./img/switch.png", 50, 50)
        self.button_switch = tk.Button(self.toolbar, image=self.image_switch, command=self.add_switch)
        self.button_switch.pack(side=tk.LEFT)

        self.image_pc = self.load_and_resize_image("./img/pc.png", 50, 50)
        self.button_client = tk.Button(self.toolbar, image=self.image_pc, command=self.add_client)
        self.button_client.pack(side=tk.LEFT)

        # Menu contextuel
        self.context_menu = tk.Menu(root, tearoff=0)
        self.context_menu.add_command(label="Modifier les propriétés", command=self.modify_properties)
        self.context_menu.add_command(label="Changer l'icône", command=self.change_icon)
        self.context_menu.add_command(label="Supprimer", command=self.delete_item)

        # Evénements souris
        self.canvas.bind("<Button-1>", self.left_click)
        self.canvas.bind("<Button-3>", self.right_click)
```

```

self.canvas.bind("<B1-Motion>", self.drag_item)
self.canvas.bind("<Button-2>", self.middle_click)
# Evènements clavier
self.root.bind("<Key>", self.key_pressed)

def create_client(self):
    item = self.canvas.create_image(100, 100, image=self.image_pc, tags=("client",))
    self.items.append({"item": item, "type": "client", "proprietes": {"name": "Client", "x": 100, "y": 100}})
    self.nb_client+=1

def create_switch(self):
    item = self.canvas.create_image(200, 200, image=self.image_switch, tags=("switch",))
    self.items.append({"item": item, "type": "switch", "proprietes": {"name": "Switch", "x": 200, "y": 200}})
    self.nb_switch+=1

def create_router(self):
    item = self.canvas.create_image(300, 300, image=self.image_router, tags=("router",))
    self.items.append({"item": item, "type": "router", "proprietes": {"name": "Router", "x": 300, "y": 300}})
    self.nb_router+=1

def add_port(self, item):
    item_type = self.canvas.gettags(item)[0]
    if item_type == "client":
        self.canvas.itemconfig(item, tags=("client", f"port_client_{self.nb_client}"))
        self.nb_client += 1
    elif item_type == "switch":
        self.canvas.itemconfig(item, tags=("switch", f"port_switch_{self.nb_switch}"))
        self.nb_switch += 1
    elif item_type == "router":
        self.canvas.itemconfig(item, tags=("router", f"port_router_{self.nb_router}"))
        self.nb_router += 1

def lien(self, item1, item2):
    x1, y1 = self.canvas.coords(item1)
    x2, y2 = self.canvas.coords(item2)
    item1_type = self.canvas.gettags(item1)[0]
    item2_type = self.canvas.gettags(item2)[0]

    port_item1 = [tag for tag in self.canvas.gettags(item1) if tag.startswith("port_")]
    port_item2 = [tag for tag in self.canvas.gettags(item2) if tag.startswith("port_")]

    line = self.canvas.create_line(x1, y1, x2, y2, fill="black", width=2, tags=(item1_type, item2_type))
    self.links.append(line) # Ajouter le lien à la liste des liens
    print(self.links)

# Créer un menu contextuel pour les liens avec uniquement l'option "Supprimer"
link_menu = tk.Menu(self.root, tearoff=0)
link_menu.add_command(label="Supprimer", command=lambda: self.delete_link(line))
self.canvas.tag_bind(line, "<Button-3>", lambda event, menu=link_menu: self.delete_link(line))

def affiche_menu_lien(self, event, menu):
    menu.post(event.x_root, event.y_root)

def delete_link(self, line):
    self.links.remove(line) # Retirer le lien de la liste des liens
    print(self.links)
    self.canvas.delete(line) # Supprimer graphiquement le lien

```

```

def left_click(self, event):
    item = self.canvas.find_closest(event.x, event.y)
    if item:
        self.current_item = item[0]

def right_click(self, event):
    item = self.canvas.find_closest(event.x, event.y)
    if item:
        self.current_item = item[0]
        self.context_menu.post(event.x_root, event.y_root)

def middle_click(self, event):
    item = self.canvas.find_closest(event.x, event.y)
    if item:
        if not self.start_link: # Premier élément sélectionné
            self.start_link = item[0]
            print(self.start_link)
            self.add_port(self.start_link) # Ajouter un port au premier élément
        else: # Deuxième élément sélectionné
            self.end_link = item[0]
            self.add_port(self.end_link) # Ajouter un port au deuxième élément
            self.show_port_menu(event) # Afficher Le menu des ports

def show_port_menu(self, event):
    item = self.canvas.find_closest(event.x, event.y)
    if item:
        port_menu = tk.Menu(self.root, tearoff=0)
        item_tags = self.canvas.gettags(item)
        port_tags = [tag for tag in item_tags if tag.startswith("port")]
        if port_tags:
            for tag in port_tags:
                port_number = tag.split('_')[-1]
                port_menu.add_command(label=f"Port {port_number}", command=1)
            port_menu.post(event.x_root, event.y_root)

def connect_ports(self, port_number):
    # Connecter les ports en utilisant le numéro de port sélectionné
    print("test")

def edit_proprieties(self):
    if self.current_item:
        item_type = self.canvas.gettags(self.current_item)[0]
        item = next(x for x in self.items if x["item"] == self.current_item)
        old_name = item["proprietes"]["name"]
        new_name = simpledialog.askstring("Modifier les propriétés", "Nouveau nom")
        if new_name:
            item["proprietes"]["name"] = new_name

def drag_item(self, event):
    if self.current_item:
        dx = event.x - self.canvas.coords(self.current_item)[0]
        dy = event.y - self.canvas.coords(self.current_item)[1]
        self.canvas.move(self.current_item, dx, dy)

def load_and_resize_image(self, filename, width, height):
    image = Image.open(filename)

```



```
        image = image.resize((width, height))
        photo = ImageTk.PhotoImage(image)
        return photo

    def change_icon(self):
        if self.current_item:
            # Boîte de dialogue pour sélectionner un fichier PNG
            self.filename = filedialog.askopenfilename(filetypes=[("Image files",
                                                                    ".png")])
            if self.filename:
                self.photo=self.load_and_resize_image(self.filename,50,50)
                self.item_images[self.current_item] = self.photo
                # Changer l'image de l'élément actuellement sélectionné
                self.canvas.itemconfig(self.current_item, image=self.photo)

    def key_pressed(self, event):
        key = event.char.upper()
        if key == "C":
            self.create_client()
        elif key == "S":
            self.create_switch()
        elif key == "R":
            self.create_router()

    def delete_item(self):
        if self.current_item:
            item = next(x for x in self.items if x["item"] == self.current_item)
            self.items.remove(item)
            self.canvas.delete(self.current_item)

root = tk.Tk()
app = WishPacketTracer(root)
root.mainloop()
```