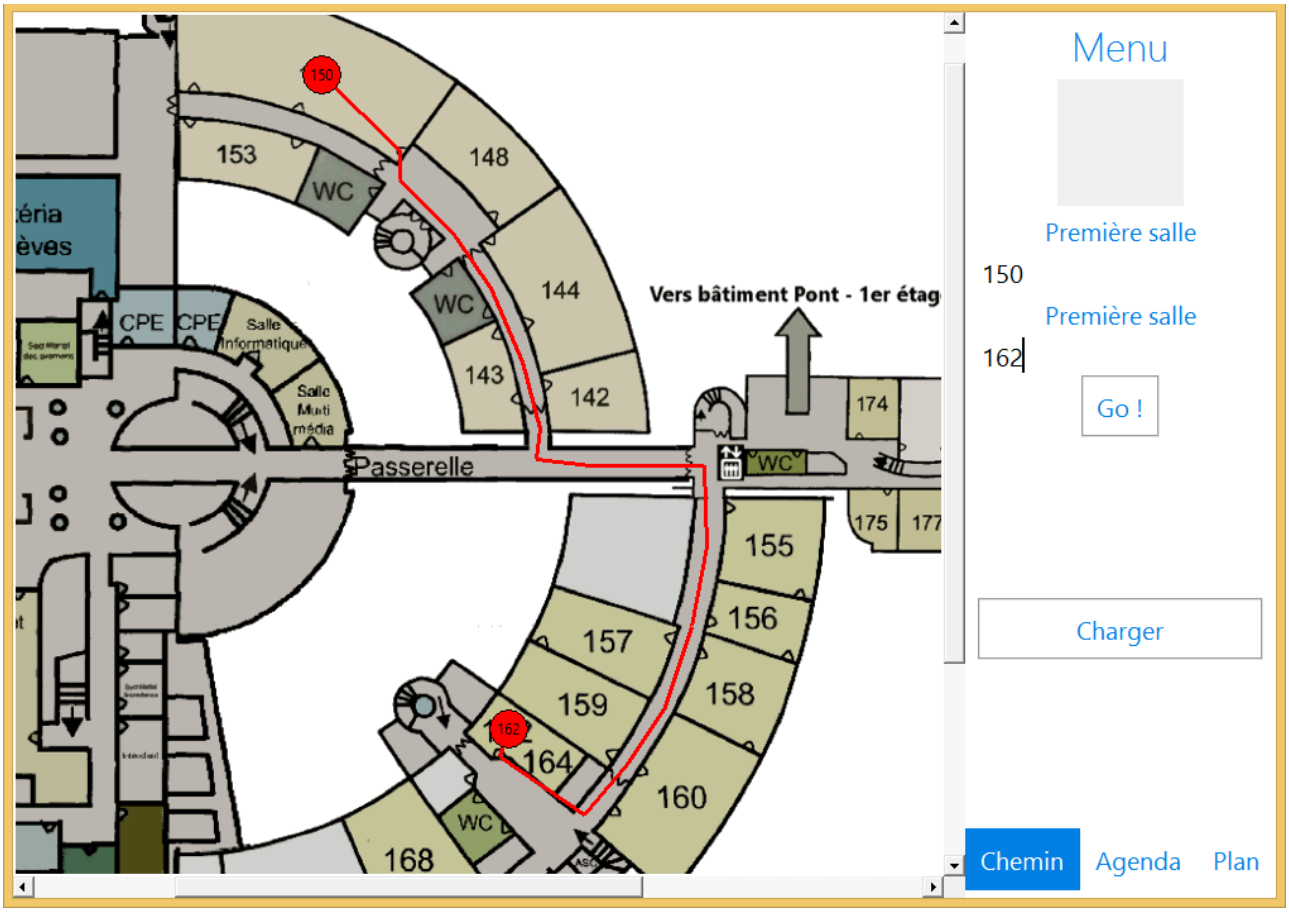
GPS Lycée

Rapport de notre projet d'ISN

Gilles DEVILLERS
Paul LECORRE
Armand RIBAULT



1. Présentation

Comment se déplacer efficacement au sein du lycée et optimiser ses trajets ?

Les lycées accueillent en général plus de mille élèves, et les infrastructures sont donc adaptées à ce nombre important de lycéens. On se retrouve donc rapidement avec des lycées très grands aux multiples bâtiments et étages.

Notre lycée comporte une centaine de salles de cours, sans compter les salles annexes comme le CDI, la salle réseau, de devoirs, de permanence, les bureaux de l'intendance, du secrétariat, de la vie scolaire…

Les élèves ont dans notre lycée environs 5 minutes entre chaque cours, pour leur laisser le temps de se déplacer d'une salle à l'autre, mais parfois en prenant du retard dans un cours, ou aux heures d'affluence, les couloirs deviennent rapidement encombrés et l'optimisation du trajet devient primordiale pour rester ponctuel.

Le but de notre application est d'indiquer rapidement le chemin le plus court à prendre pour se déplacer jusqu'à la prochaine salle de cours, afin de passer le moins de temps possible dans les couloirs.

2. L'application

L'application devra :

- Posséder une interface graphique simple et ergonomique et élégante.
- Organiser les plans de lycée sous forme de graphe.
- Être polyvalente, en permettant de modéliser simplement n'importe quel lycée ou autre infrastructure plutôt que d'être exclusif à un lycée.
- Sauvegarder et charger ces différentes modélisations à la volée dans un format personnalisé.
- Proposer un emploi du temps modifiable et un formulaire de connexion pour que chaque élève puisse facilement accéder à ses trajets d'une salle de cours à une autre.
- En plus des salles de cours de l'emploi du temps, permettre de chercher le trajet le plus court entre deux salles données.

Pour la réalisation de l'application, nous avons choisi d'utiliser le langage de programmation Python, un langage interprété, dynamiquement typé, avec une gestion automatique de la mémoire, **une syntaxe claire** et qui propose nativement des types de haut niveau, utiles pour **la représentation de graphes**.

La distribution Python contient également la **bibliothèque graphique** Tkinter préinstallée qui permet de créer des interfaces graphiques simplement et rapidement.

Une des caractéristiques les plus importantes est le **support multi-plateformes** de ces outils, qui ne rendent pas l'application exclusive à un système d'exploitation particulier.

Tous ces avantages permettent **un développement simple et rapide** d'applications nécessitant **une interface graphique**.

3. Répartition des tâches

Gilles DEVILLERS :

- Choix de la représentation interne du graphe et de l'algorithme de recherche du chemin le plus court.
- Réalisation de l'application de modélisation du lycée avec importation et exportation du graphe.
- Réalisation de l'algorithme de recherche du chemin le plus court ainsi que de l'affichage du trajet.
- Assemblage des plans des différents bâtiments du lycée pour obtenir un seul plan du lycée entier.

Paul LECORRE :

- Récupération des plans du lycée.
- Choix de la représentation interne de l'emploi du temps.
- Réalisation de l'application d'affichage et d'édition de l'emploi du temps.
- Réalisation du formulaire de connexion des élèves, ainsi que de la sauvegarde et le chargement des emplois du temps.

Armand RIBAULT :

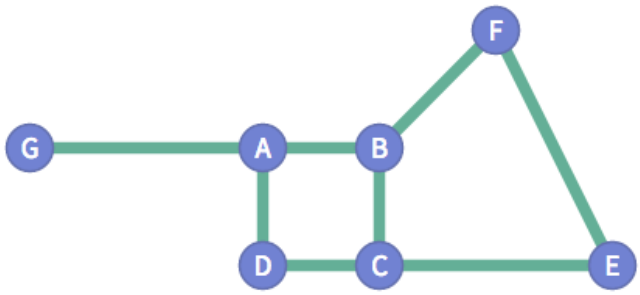
- Réalisation de la fenêtre d'accueil.
- Intégration de chaque partie (applications de modélisation, d'affichage du chemin le plus court et de l'emploi du temps) dans l'application finale.
- Réalisation de l'interface graphique de chaque partie (ergonomie, icônes, placement des divers éléments graphiques).

4. Réalisation

A. *Modélisation du lycée*

Pour comprendre le fonctionnement et l'utilisation des graphes ainsi que l'algorithme de recherche, je me suis aidé du site « Red Blob Games » d'Amit Patel qui présente de multiples articles à propos de diverses techniques de programmation utilisées principalement dans les jeux vidéos.

Comme dit précédemment, le lycée sera représenté sous forme de graphe. Les graphes sont des ensembles de nœuds, reliés entre eux par des arêtes. Dans notre cas, les nœuds représentent les salles du lycée et les arêtes des portions de couloir.



Exemple de graphe

On applique un « poids » aux arêtes, qui représente le temps de déplacement. Ici, plus la distance à parcourir sera longue, plus le coût de déplacement sera élevé. C'est ce qui permet à l'algorithme de recherche de trouver le chemin le plus court, peu importe la taille des arêtes.

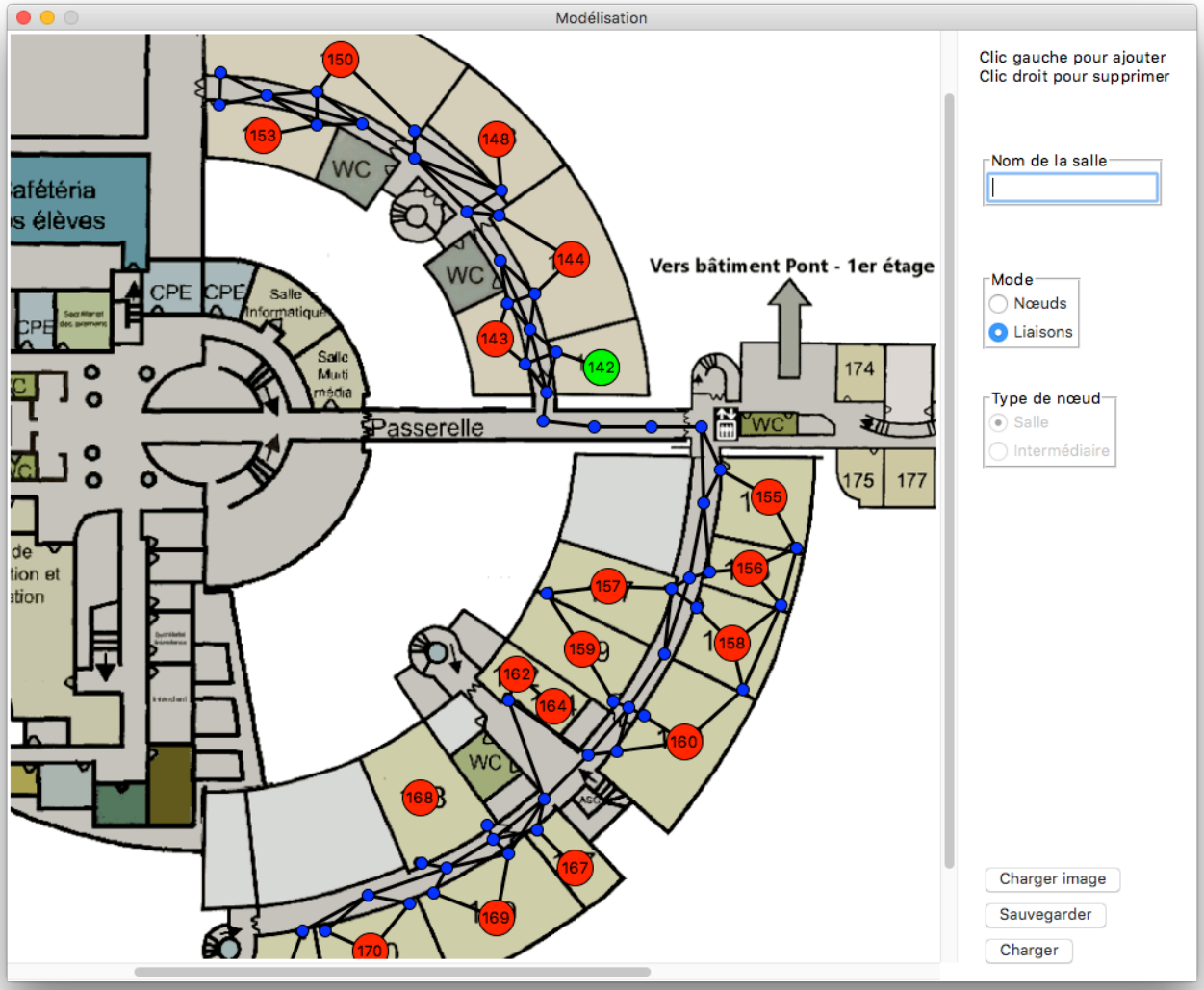
Dans le code, un graphe est représenté avec un dictionnaire de la forme :

```
graphe = {"noeud": "x": <coordonnée horizontale du point>,
                    "y": <coordonnée verticale du point>,
                    "voisin": <distance entre noeud et voisin>
          }
```

Par exemple, le graphe suivant comporte deux nœuds nommés « A » et « B » reliés entre eux avec une distance de 352 pixels les séparant :

```
graphe = {"A": {"x": 452, "y": 53, "B": 352},
          "B": {"x": 102, "y": 89, "A": 352}
         }
```

J'ai donc réalisé une application pour modéliser facilement un lycée et le transformer en graphe. En l'ouvrant, on charge une image ou un graphe et on peut directement commencer à modéliser un espace à partir d'un plan.



Il y a plusieurs modes d'édition :

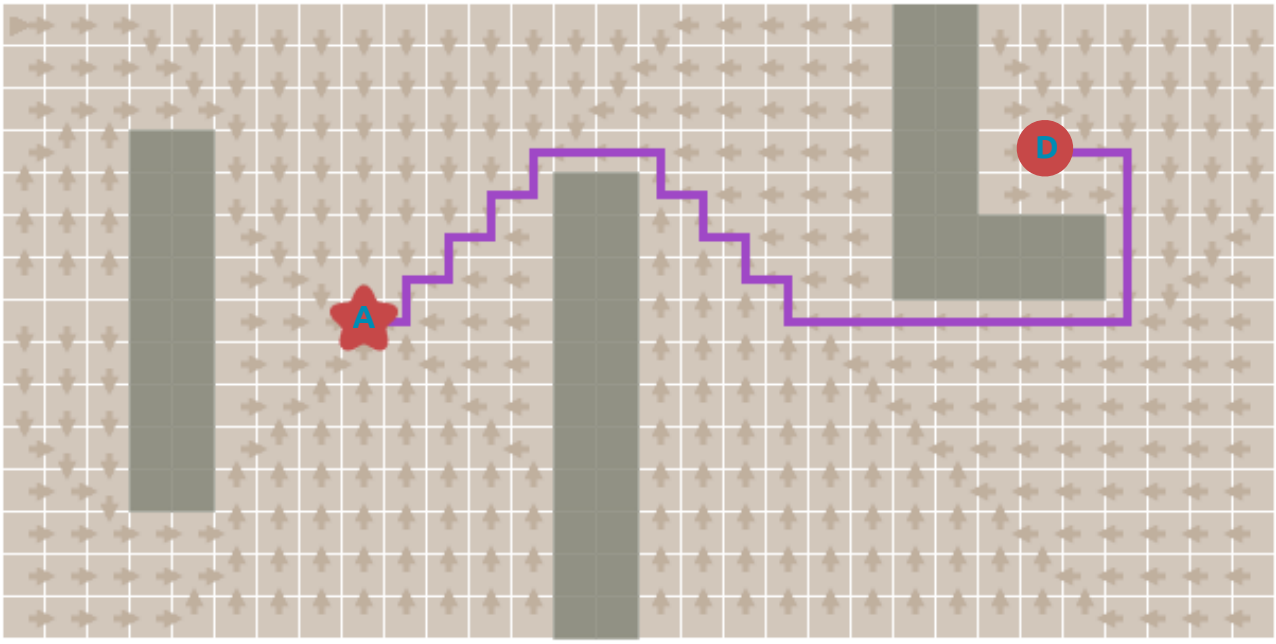
- Le mode « Nœuds » ajouter une nouvelle salle avec le nom indiqué dans le champ prévu à cet effet en faisant un clic gauche sur le plan qui s'affiche à gauche. On peut également choisir d'ajouter des nœuds « intermédiaires ». Ils sont plus petits et n'ont pas de nom, ils servent à représenter les trajets dans les couloirs. Plus on en met, meilleure sera la précision. Il suffit de faire un clic droit sur un nœud pour le supprimer.

- Le mode « Liaisons » sélectionne un nœud en cliquant dessus (le nœud devient vert), et le lie avec un autre nœud en cliquant également dessus. Un trait noir apparaît alors, représentant l'arête du graphe entre les deux nœuds sélectionnés. En faisant un clic droit sur une liaison, elle disparait.

Le graphe peut ensuite être sauvegardé. Il est convertit en JSON, un format qui convertit le dictionnaire en texte, puis enregistré sous un fichier d'extension « .graphe » qui pourra ensuite être chargé dans le programme de recherche du plus court chemin.

B. Recherche du plus court chemin

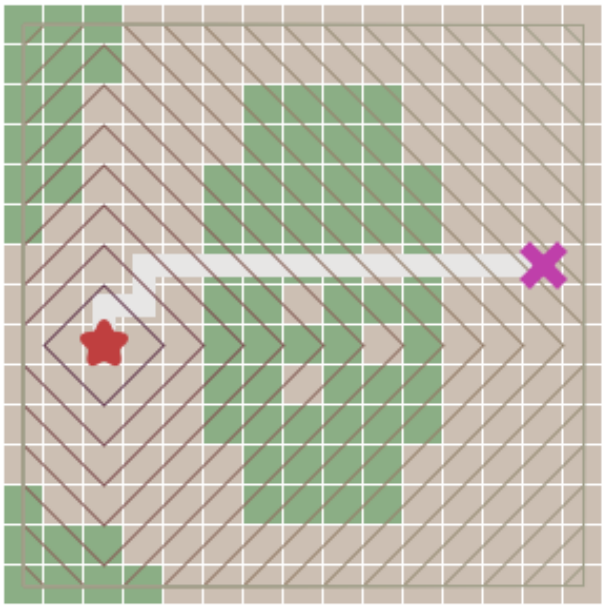
L'algorithme de recherche du chemin le plus court entre deux nœuds se nomme « Dijkstra ». Il se base sur l'« Algorithme de parcours en largeur ». Ce dernier se charge de parcourir tous les nœuds du graphe : à partir d'un nœud d'arrivée (A), il va parcourir tous ses nœuds voisins jusqu'à ce qu'il ait visité tous les nœuds du graphe, en enregistrant à chaque fois le nœud d'où il provient.



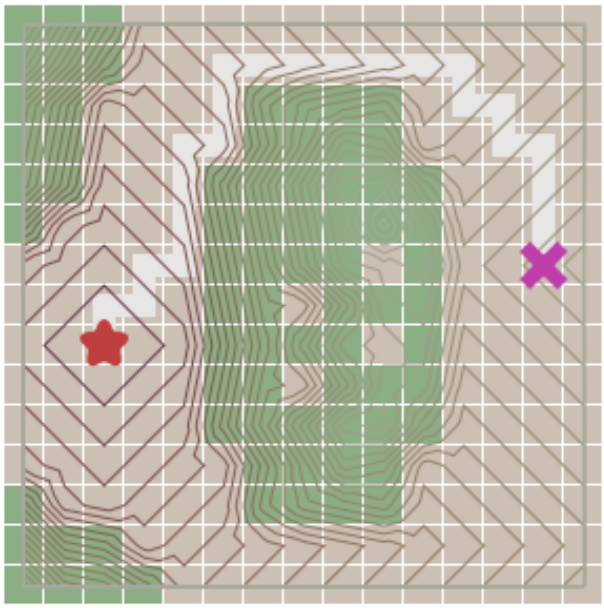
Représentation des chemins trouvés par l'algorithme de parcours en largeur sur une grille. Une grille est essentiellement un graphe dont les cases sont les nœuds.

Une fois le processus terminé, il suffit de prendre un nœud d'arrivée (B) sur le graphe et de parcourir le chemin à l'envers pour connaître le chemin le plus court entre les deux points (sur l'image ci-dessus, il suffit de suivre les flèches depuis n'importe quel point).

Algorithme de parcours en largeur

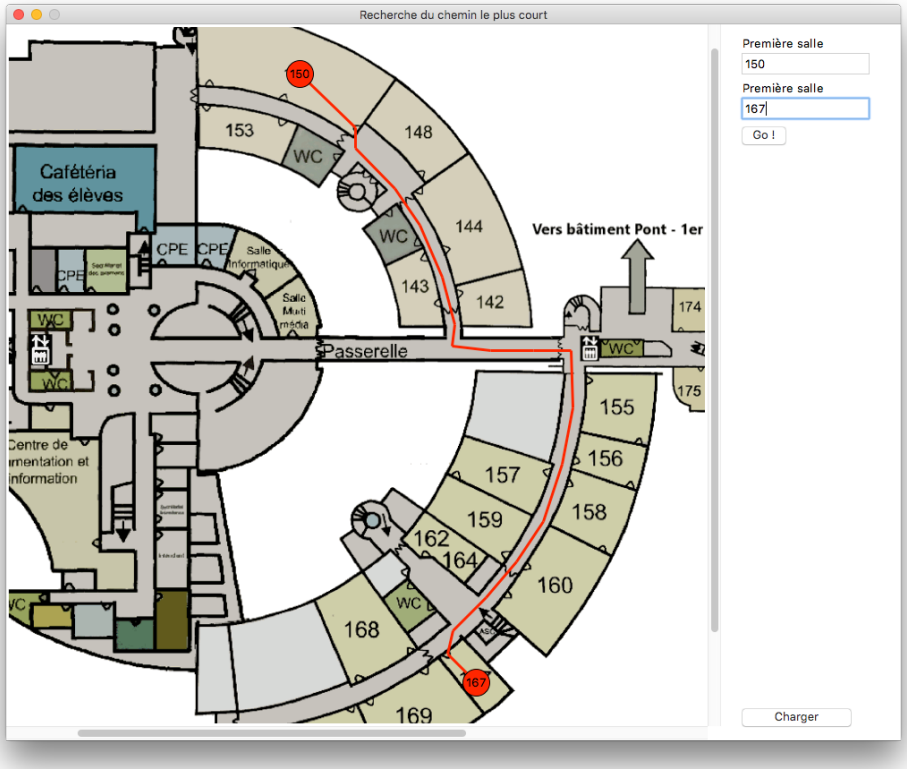


Algorithme de Dijkstra



Les lignes représentent la façon dont chaque algorithme parcourt les nœuds.
Les cases vertes ont un coût plus élevé, il est plus rapide de les contourner.

Cet algorithme parcours le graphe ne tient pas compte du poids des arêtes. Sur l'image, le passage d'une case à une autre (d'un nœud à un autre) est égal. L'algorithme de Dijkstra corrige ce problème.



Dans le prototype (image ci-dessus), deux champs sont disponibles pour entrer le nom de deux salles du lycée, et un clic sur le bouton « Go ! » trace directement le chemin le plus court à prendre pour atteindre cette salle.

C. *Fonctions communes*

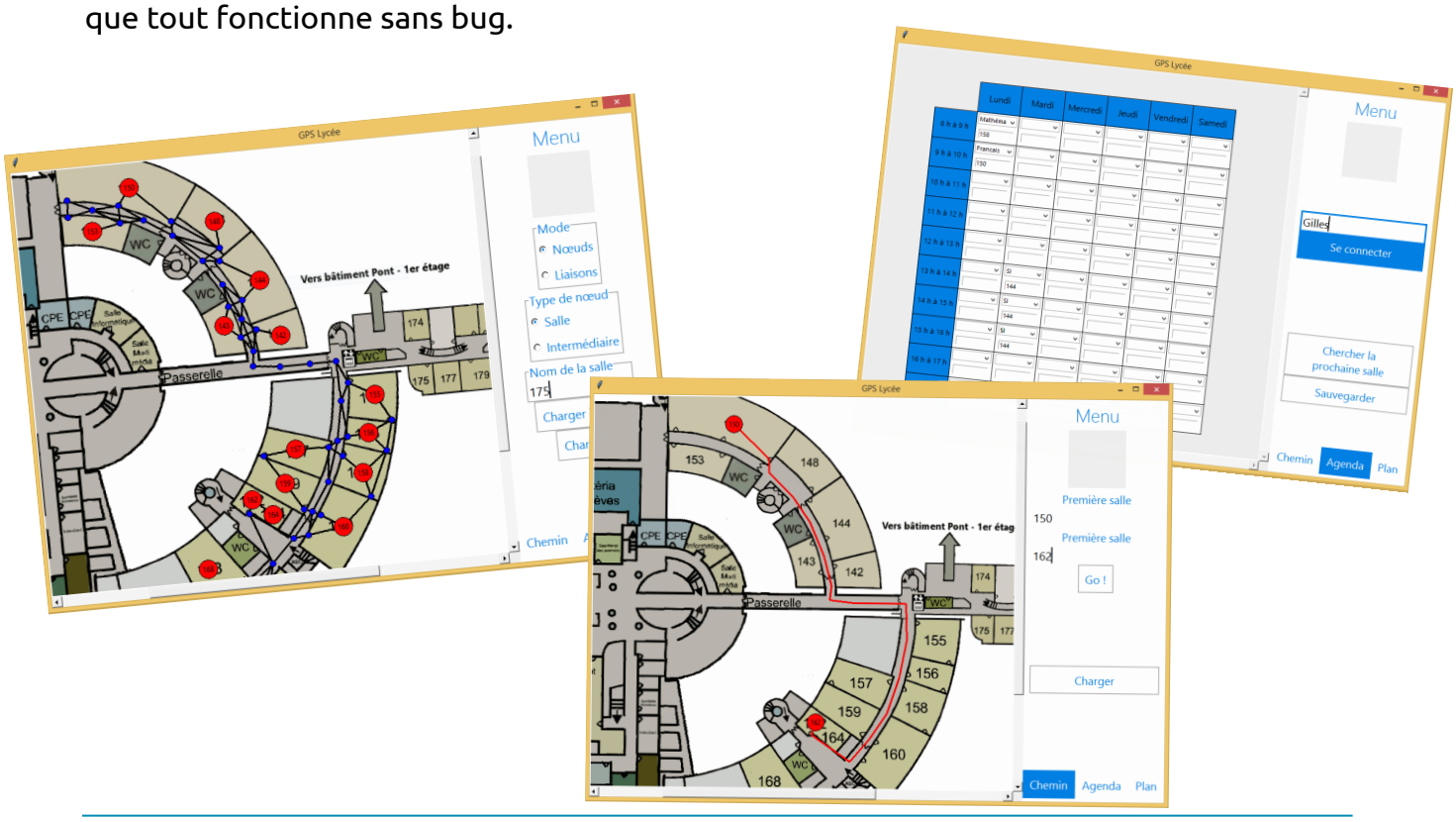
Pour réaliser les deux parties précédentes, il a fallu programmer des fonctions communes aux deux programmes. Pour éviter les répétitions et rendre le code modulaire, je les ai placées dans un fichier séparé nommé « utilitaires.py ». Ces fonctions servent à dessiner les nœuds (`dessinerNoeud()`) et les liaisons (`dessinerLiaison()`) ainsi qu'à charger (`chargerGraphe()`) ou sauvegarder (`sauvegarderGraphe()`) un graphe ou simplement charger une image (`chargerImageSurCanvas()`).

5. Intégration

La partie qui nous a posé problème est incontestablement l'intégration globale de nos parties dans une seule et même application finale. Nous avons chacun commencé à travailler sur nos parties respectives, en faisant nos applications « prototype » chacun de notre côté, en aidant les uns les autres et en imaginant l'intégration finale assez simple.

Au final, l'utilisation de variables globales pour nos programmes est une solution simple, claire et efficace mais c'était probablement une erreur car il est difficile, sans programmation orientée objet, de réaliser une application graphique avec Tkinter de cette façon.

Pour avoir une solution finale fonctionnelle, nous avons donc rassemblé tous nos codes dans un seul et même fichier (à part le fichier « utilitaires.py » qui ne contient que des fonctions et qui par conséquent s'intègre facilement) et avons modifié le code pour que tout fonctionne sans bug.



6. Conclusion

Nous avons pensé à plusieurs améliorations éventuelles pour notre application :

- Une refactorisation complète du code, en séparant chaque partie dans son propre fichier, éventuellement dans sa propre classe.

- Une connexion au service de vie scolaire en ligne du lycée, pour adapter l'emploi du temps de chaque élève en fonction des absences des professeurs.

- Un système de sauvegarde amélioré.

- La fenêtre principale redimensionnable.

- Une recherche automatique des toilettes les plus proches.

Au final, ce projet a été une expérience très intéressante pour tout notre groupe. Je programme assez régulièrement pendant mon temps libre, mais jamais avec d'autres personnes. Ce projet a été l'occasion pour nous tous de développer un esprit d'équipe et de collaboration important et nous a confortés dans notre choix d'orientation vers des métiers d'ingénieur, dans le domaine informatique pour ma part.

Nous avons pu voir les choix à faire lors du travail à plusieurs, et les erreurs à ne pas commettre. L'organisation et la rigueur sont les clés de la réussite.

De plus, une programmation orientée objet nous aurait aidé à encapsuler chaque partie et aurait facilité l'intégration globale.

Nous avons également acquis de nouvelles connaissances en programmation et en réalisation d'application complète avec interface graphique, chose que nous n'avions jamais réalisée auparavant.

7. Annexes

1. Prototype

Programme de modélisation du lycée « CreationGraphe.py » :

```
from tkinter import *
from tkinter.filedialog import *
import math
import os
from utilitaires import *

graphe = {}
listeNoeuds = {}        # Stocke les graphismes des nœuds
listeLiaisons = {}      # Stocke les graphismes des liaisons
noeudSelectionne = {}   # Stocke le nom et les graphismes du nœud sélectionné
idNoeudIntermedaire = 0 # Nombre de nœuds intermédiaires

# ----- Fenêtre ----- #

win = Tk()
win.geometry("1000x800")
win.resizable(0, 0) # Empêche le redimensionnement de la fenêtre
win.title("Modélisation")

# Canvas contenant le plan
frameCanvas = Frame(win)

scrollbarCanvasHorizontal = Scrollbar(frameCanvas, orient = HORIZONTAL)
scrollbarCanvasHorizontal.grid(row = 1, column = 0, sticky = E + W)
scrollbarCanvasVertical = Scrollbar(frameCanvas)
scrollbarCanvasVertical.grid(row = 0, column = 1, sticky = N + S)

canvas = Canvas(frameCanvas,
                width = 778,
                height = 778,
                bg = "#FFFFFF",
                xscrollcommand = scrollbarCanvasHorizontal.set,
                yscrollcommand = scrollbarCanvasVertical.set,
                scrollregion = (0, 0, 0, 0))
canvas.grid(row = 0, column = 0, sticky = N + S + E + W)

scrollbarCanvasVertical.config(command = canvas.yview)
scrollbarCanvasHorizontal.config(command = canvas.xview)

frameCanvas.place(x = 0, y = 0)


# Instructions
messageInstructions = Message(win,
                              width = 170,
                              text = "Clic gauche pour ajouter\nClic droit pour supprimer")
messageInstructions.place(x = 810, y = 10)

# Identifiants des boutons de souris
ID_CLIC_GAUCHE = "<Button-1>"
ID_CLIC_DROIT = "<Button-2>" if os.sys.platform == "darwin" else "<Button-3>" # Compatibilité
OSX


# ----- Chargement du plan ----- #

imageAffichee = None
```

```
def chargerImage(cheminImage):
    global imageAffichee
    if imageAffichee is not None:
        canvas.delete(imageAffichee)

    imageAffichee = chargerImageSurCanvas(canvas, cheminImage)


boutonChargementImage = Button(win,
                               text = "Charger image",
                               command = lambda: chargerImage(None))
boutonChargementImage.place(x = 820, y = 700)


# ----- Gestion des nœuds ----- #

def noeudClique(souris): # Retourne le nœud qui a été cliqué
    for noeud in graphe:
        if noeud[0] == "_":
            rayon = RAYON_NOEUD_INTERMEDIAIRE
        else:
            rayon = RAYON_NOEUD

        if (graphe[noeud]["x"] >= souris.x - rayon and
            graphe[noeud]["x"] <= souris.x + rayon and
            graphe[noeud]["y"] >= souris.y - rayon and
            graphe[noeud]["y"] <= souris.y + rayon):
            return noeud
            break

def selectionnerNoeud(nomNoeud): # Un noeud sélectionné s'affiche en vert
    if nomNoeud == "":
        return

    deselectionnerNoeud()

    noeudSelectionne["nom"] = nomNoeud
    noeudSelectionne["dessin"] = dessinerNoeud(
        canvas,
        graphe,
        nomNoeud,
        "green",
    )

def deselectionnerNoeud():
    global noeudSelectionne
    if noeudSelectionne:
        if noeudSelectionne["nom"][0] == "_": # Si le nœud est un nœud intermédiaire
            canvas.delete(noeudSelectionne["dessin"]) # Efface le rond
        else:
            canvas.delete(noeudSelectionne["dessin"][0]) # Efface le rond
            canvas.delete(noeudSelectionne["dessin"][1]) # Efface le nom

        noeudSelectionne = {}

def ajouterNoeud(nomNoeud, x, y):
    if nomNoeud in graphe:
        # Faire clignoter en rouge le champ « Nom de la salle »
        entryNomNoeud.configure(bg = "red")
        entryNomNoeud.after(500,
                            lambda: entryNomNoeud.configure(bg = ENTRY_NOM_COULEUR_PAR_DEFAUT))
        return

    graphe[nomNoeud] = {}
    graphe[nomNoeud]["x"] = x
    graphe[nomNoeud]["y"] = y

    listeNoeuds[nomNoeud] = dessinerNoeud(canvas, graphe, nomNoeud)

def supprimerNoeud(nomNoeud):
```

```
    if nomNoeud[0] == "_": # Si le nœud est un nœud intermédiaire
        canvas.delete(listeNoeuds[nomNoeud]) # Efface le rond
    else:
        canvas.delete(listeNoeuds[nomNoeud][0]) # Efface le rond
        canvas.delete(listeNoeuds[nomNoeud][1]) # Efface le nom

    # Nécessaire sinon graphe[nomNoeud] change pendant l'itération
    noeuds = tuple(graphe[nomNoeud].keys())

    for noeud in noeuds: # Supprimer toutes les liaisons de ce nœud
        if noeud != "x" and noeud != "y":
            delier(nomNoeud, noeud)
    del listeNoeuds[nomNoeud]
    del graphe[nomNoeud]

labelframeframeNomNoeud = LabelFrame(win, text = "Nom de la salle")
labelframeframeNomNoeud.place(x = 820, y = 100)
entryNomNoeud = Entry(labelframeframeNomNoeud, width = 15)
entryNomNoeud.pack()
entryNomNoeud.focus_set()
ENTRY_NOM_COULEUR_PAR_DEFAUT = entryNomNoeud.cget("bg")


# ----- Gestion des liaisons ----- #

def clicSurLiaison(event):
    event.y = canvas.canvasy(event.y) # Position relative au canvas
    event.x = canvas.canvasx(event.x)

    liaisonCliquee = event.widget.find_closest(event.x, event.y)[0]
    for liaison in listeLiaisons.keys():
        if listeLiaisons[liaison] == liaisonCliquee:
            delier(liaison[0], liaison[1])
            return

def lier(noeud1, noeud2):
    if noeud1 == noeud2:
        return

    idLiaison = [noeud1, noeud2]
    idLiaison.sort()
    idLiaison = tuple(idLiaison)
    if idLiaison in listeLiaisons:
        return

    distanceEntreNoeuds = round(math.sqrt(
        pow(graphe[noeud1]["x"] - graphe[noeud2]["x"], 2) +
        pow(graphe[noeud1]["y"] - graphe[noeud2]["y"], 2))
    )

    graphe[noeud1][noeud2] = graphe[noeud2][noeud1] = distanceEntreNoeuds

    listeLiaisons[idLiaison] = dessinerLiaison(canvas,
                                               graphe,
                                               noeud1,
                                               noeud2,
                                               "black",
                                               3)

    # Permet de redessiner les ronds au dessus de la liaison
    if noeud1[0] == "_": # Si le nœud est un nœud intermédiaire
        canvas.tag_raise(listeNoeuds[noeud1]) # Remonte le rond
    else:
        canvas.tag_raise(listeNoeuds[noeud1][0]) # Remonte le rond
        canvas.tag_raise(listeNoeuds[noeud1][1]) # Remonte le nom

    if noeud2[0] == "_": # Si le nœud est un nœud intermédiaire
        canvas.tag_raise(listeNoeuds[noeud2]) # Remonte le rond
    else:
```

```
        canvas.tag_raise(listeNoeuds[noeud2][0]) # Remonte le rond
        canvas.tag_raise(listeNoeuds[noeud2][1]) # Remonte le nom

    canvas.tag_bind(listeLiaisons[idLiaison], ID_CLIC_DROIT, clicSurLiaison)

def delier(noeud1, noeud2):
    del graphe[noeud1][noeud2]
    del graphe[noeud2][noeud1]

    idLiaison = [noeud1, noeud2]
    idLiaison.sort()
    idLiaison = tuple(idLiaison)

    canvas.delete(listeLiaisons[idLiaison])
    del listeLiaisons[idLiaison]


# ----- Actions ----- #

def clicGauche(souris):
    global idNoeudIntermedaire

    souris.y = canvas.canvasy(souris.y) # Position sur le canvas
    souris.x = canvas.canvasx(souris.x)

    if modeSelectionne.get() == "noeuds":
        nomNoeud = entryNomNoeud.get()

        if typeNoeudSelectionne.get() == "salle":
            if nomNoeud:
                ajouterNoeud(nomNoeud, souris.x, souris.y)
                entryNomNoeud.delete(0, len(entryNomNoeud.get()))
        else:
            ajouterNoeud("_" + str(idNoeudIntermedaire), souris.x, souris.y)
            idNoeudIntermedaire += 1
    else:
        nomNoeud = noeudClique(souris)
        if not noeudSelectionne:
            if nomNoeud:
                selectionnerNoeud(nomNoeud)
        else:
            if nomNoeud:
                lier(noeudSelectionne["nom"], nomNoeud)
            deselectionnerNoeud()

def clicDroit(souris):
    souris.y = canvas.canvasy(souris.y) # Position sur le canvas
    souris.x = canvas.canvasx(souris.x)

    if modeSelectionne.get() == "noeuds":
        nomNoeud = noeudClique(souris)
        if nomNoeud:
            supprimerNoeud(nomNoeud)


canvas.bind(ID_CLIC_GAUCHE, clicGauche)
canvas.bind(ID_CLIC_DROIT, clicDroit)

# ----- Sélection du mode d'édition ----- #

def changementDeMode():
    if modeSelectionne.get() == "noeuds":
        deselectionnerNoeud()
        radiobuttonTypeNoeudSalle.config(state = NORMAL)
        radiobuttonTypeNoeudIntermediaire.config(state = NORMAL)
    else:
        radiobuttonTypeNoeudSalle.config(state = DISABLED)
        radiobuttonTypeNoeudIntermediaire.config(state = DISABLED)
```

```
modeSelectionne = StringVar()
labelframeMode = LabelFrame(win, text = "Mode")
labelframeMode.place(x = 820, y = 200)

radiobuttonModeNoeuds = Radiobutton(labelframeMode,
                                    text = "Nœuds",
                                    variable = modeSelectionne,
                                    value = "noeuds",
                                    command = changementDeMode)
radiobuttonModeNoeuds.select()
radiobuttonModeNoeuds.pack(anchor = W)

radiobuttonModeLiaisons = Radiobutton(labelframeMode,
                                      text = "Liaisons",
                                      variable = modeSelectionne,
                                      value = "liaisons",
                                      command = changementDeMode)
radiobuttonModeLiaisons.pack(anchor = W)


# ----- Sélection du type de nœud ----- #

typeNoeudSelectionne = StringVar()
labelframeTypeNoeud = LabelFrame(win, text = "Type de nœud")
labelframeTypeNoeud.place(x = 820, y = 300)

radiobuttonTypeNoeudSalle = Radiobutton(labelframeTypeNoeud,
                                        text = "Salle",
                                        variable = typeNoeudSelectionne,
                                        value = "salle")
radiobuttonTypeNoeudSalle.select()
radiobuttonTypeNoeudSalle.pack(anchor = W)

radiobuttonTypeNoeudIntermediaire = Radiobutton(labelframeTypeNoeud,
                                                text = "Intermédiaire",
                                                variable = typeNoeudSelectionne,
                                                value = "intermediaire")
radiobuttonTypeNoeudIntermediaire.pack(anchor = W)


# ----- Sauvegarde & Chargement ----- #

def sauvegarder():
    sauvegarderGraphe(graphe, canvas.cheminImage)

def nouveauGraphe():
    global graphe, listeNoeuds, listeLiaisons

    for noeud in listeNoeuds:
        if noeud[0] == "_":
            canvas.delete(listeNoeuds[noeud])
        else:
            canvas.delete(listeNoeuds[noeud][0])
            canvas.delete(listeNoeuds[noeud][1])

    for liaison in listeLiaisons.keys():
        delier(liaison[0], liaison[1])

    graphe = listeNoeuds = listeLiaisons = {}

def charger():
    global graphe, listeNoeuds, listeLiaisons, idNoeudIntermedaire

    fichier = chargerGraphe()

    if not fichier:
        return

    nouveauGraphe()
```

```
    graphe = fichier["graphe"]
    chargerImage(fichier["image"])

    for noeud in graphe:
        for voisin in graphe[noeud]:
            if voisin == "x" or voisin == "y":
                continue
            idLiaison = [noeud, voisin]
            idLiaison.sort()
            idLiaison = tuple(idLiaison)
            if idLiaison not in listeLiaisons:
                listeLiaisons[idLiaison] = dessinerLiaison(canvas,
                                                           graphe,
                                                           noeud,
                                                           voisin,
                                                           "black",
                                                           3)
                canvas.tag_bind(listeLiaisons[idLiaison], ID_CLIC_DROIT, clicSurLiaison)

    for noeud in graphe: # Pour dessiner les ronds au dessus des liaisons
        listeNoeuds[noeud] = dessinerNoeud(canvas, graphe, noeud)

    if noeud[0] == "_": # On cherche l'ID du dernier nœud intermédiaire
        idNoeud = int(noeud[1:])
        if idNoeud > idNoeudIntermedaire:
            idNoeudIntermedaire = idNoeud


boutonSauvegarde = Button(win, text = "Sauvegarder", command = sauvegarder)
boutonSauvegarde.place(x = 820, y = 730)
boutonChargement = Button(win, text = "Charger", command = charger)
boutonChargement.place(x = 820, y = 760)


win.mainloop()
```

Programme de recherche du plus court chemin « RechercheChemin.py » :

```
from tkinter import *
from tkinter.filedialog import *
from utilitaires import *

graphe = {}
listeLiaisons = []
noeudDepart = {}
noeudArrive = {}

def cheminPlusCourt(depart, arrivee):
    # Visite de tous les chemins
    frontiere = []
    frontiere.append(depart)
    depuis = {}
    coutDeplacement = {}
    depuis[depart] = None
    coutDeplacement[depart] = 0

    while len(frontiere) > 0:
        actuel = frontiere.pop()

        for suivant in graphe[actuel]:
            if suivant == "y" or suivant == "x" or suivant == "rayon":
                continue

            nouveauCoutDeplacement = coutDeplacement[actuel] + graphe[actuel][suivant]
            if suivant not in coutDeplacement or nouveauCoutDeplacement <
coutDeplacement[suivant]:
                coutDeplacement[suivant] = nouveauCoutDeplacement
```

```
                priorite = nouveauCoutDeplacement
                frontiere.insert(priorite, suivant)
                depuis[suivant] = actuel

    # Détermination du chemin le plus court
    actuel = arrivee
    chemin = [actuel]
    while actuel != depart:
        current = actuel = depuis[actuel]
        chemin.append(current)
    chemin.reverse()

    return chemin


# ----- Fenêtre ----- #

win = Tk()
win.geometry("1000x800")
win.resizable(0, 0)
win.title("Recherche du chemin le plus court")

# Canvas du plan
frameCanvas = Frame(win)

scrollbarCanvasHorizontal = Scrollbar(frameCanvas, orient = HORIZONTAL)
scrollbarCanvasHorizontal.grid(row = 1, column = 0, sticky = E + W)
scrollbarCanvasVertical = Scrollbar(frameCanvas)
scrollbarCanvasVertical.grid(row = 0, column = 1, sticky = N + S)

canvas = Canvas(frameCanvas, width = 778, height = 778, bg = "#FFFFFF", xscrollcommand =
scrollbarCanvasHorizontal.set, yscrollcommand = scrollbarCanvasVertical.set, scrollregion = (0,
0, 0, 0))
canvas.grid(row = 0, column = 0, sticky = N + S + E + W)

scrollbarCanvasVertical.config(command = canvas.yview)
scrollbarCanvasHorizontal.config(command = canvas.xview)

frameCanvas.place(x = 0, y = 0)


# ----- Liste des noeuds ----- #

def dessinerCheminPlusCourt():
    global noeudDepart, noeudArrive, listeLiaisons

    if noeudDepart:
        canvas.delete(noeudDepart[0])
        canvas.delete(noeudDepart[1])
        canvas.delete(noeudArrive[0])
        canvas.delete(noeudArrive[1])
        for liaison in listeLiaisons:
            canvas.delete(liaison)

    listeLiaisons = []

    noeud1 = entryNoeudDepart.get()
    noeud2 = entryNoeudArrive.get()

    chemin = cheminPlusCourt(noeud1, noeud2)

    for idNoeud in range(len(chemin) - 1):
        listeLiaisons.append(dessinerLiaison(canvas, graphe, chemin[idNoeud], chemin[idNoeud +
1], "red", 3))

    noeudDepart = dessinerNoeud(canvas, graphe, noeud1)
    noeudArrive = dessinerNoeud(canvas, graphe, noeud2)


labelNoeudDepart = Label(win, text = "Première salle")
```

```
labelNoeudDepart.place(x = 820, y = 10)
entryNoeudDepart = Entry(win, width = 15)
entryNoeudDepart.place(x = 820, y = 30)
labelNoeudDepart = Label(win, text = "Première salle")
labelNoeudDepart.place(x = 820, y = 60)
entryNoeudArrive = Entry(win, width = 15)
entryNoeudArrive.place(x = 820, y = 80)
boutonGo = Button(win, text = "Go !", command = dessinerCheminPlusCourt)
boutonGo.place(x = 820, y = 110)


# ----- Chargement du plan ----- #

imageAffichee = PhotoImage()

def chargerImage(cheminImage):
    global imageAffichee
    if imageAffichee is not None:
        canvas.delete(imageAffichee)

    imageAffichee = chargerImageSurCanvas(canvas, cheminImage)


# ----- Sauvegarde et chargement ----- #

def charger():
    global graphe
    fichier = chargerGraphe()

    graphe = fichier["graphe"]
    chargerImage(fichier["image"])


boutonChargement = Button(win, text = "Charger", width = 11, command = charger)
boutonChargement.place(x = 820, y = 760)


win.mainloop()
```

Programme de fonctions utilitaires « utilitaires.py » :

```
from tkinter import *
from tkinter.filedialog import *
import json

RAYON_NOEUD = 15
RAYON_NOEUD_INTERMEDIAIRE = 5

# ----- Sauvegarde & Chargement ----- #

def chargerImageSurCanvas(canvas, cheminImage):
    if cheminImage is None:
        cheminImage = askopenfilename(filetypes = [("GIF", "*.gif")])
        if cheminImage is None:
            return

    image = PhotoImage(file = cheminImage)

    canvas.cheminImage = cheminImage # À garder pour sauvegarder
    canvas.image = image
    canvas.config(scrollregion = (0, 0, image.width(), image.height()))
    return canvas.create_image(0, 0, image = image, anchor = NW)

def sauvegarderGraphe(graphe, cheminImage):
    fichier = {}
    fichier["graphe"] = graphe
    fichier["image"] = cheminImage
```

```
    f = asksaveasfile(mode = "w", defaultextension = ".graphe", filetypes = [("Graphe",
"*.graphe")])
    json.dump(fichier, f)
    f.close()

def chargerGraphe():
    f = askopenfile(mode="r", filetypes = [("Graphe", "*.graphe")])
    if f is None:
        return
    fichier = json.load(f)
    f.close()
    return fichier


# ----- Utilitaires de dessin ----- #

def dessinerNoeud(parent, graphe, nomNoeud, couleur = "red"):
    if nomNoeud[0] == "_":
        rayon = RAYON_NOEUD_INTERMEDIAIRE
        if couleur == "red": # Si aucune couleur n'a été passée en argument
            couleur = "blue"
    else:
        rayon = RAYON_NOEUD

    rond = parent.create_oval(
        graphe[nomNoeud]["x"] - rayon,
        graphe[nomNoeud]["y"] - rayon,
        graphe[nomNoeud]["x"] + rayon,
        graphe[nomNoeud]["y"] + rayon,
        fill = couleur
    )

    if nomNoeud[0] != "_":
        texte = parent.create_text(
            graphe[nomNoeud]["x"],
            graphe[nomNoeud]["y"],
            text = nomNoeud
        )
        return (rond, texte)

    return rond

def dessinerLiaison(parent, graphe, noeud1, noeud2, couleur, epaisseur):
    return parent.create_line(
        graphe[noeud1]["x"],
        graphe[noeud1]["y"],
        graphe[noeud2]["x"],
        graphe[noeud2]["y"],
        fill = couleur,
        width = epaisseur
    )
```

2. Application finale

Programme de l'application « GPSLycee.py » :

```
# ----- Imports ----- #

from tkinter import *
from tkinter.filedialog import *
from utilitaires import *
from tkinter import ttk
from time import strftime
import math
```

```
import os


# ----- Initialisation de l'interface ----- #

# Couleurs
accentDark  = "#0080E5"
accentLight = "#0177D7"
gris        = "#EEEEEE"
backColor   = "white"

# Polices
title    = ("Segoe UI Light", 24)
subtitle = ("Segoe UI", 20)
body     = ("Segoe UI", 15)

# Fenêtre
mainWindow = Tk()
mainWindow.title("GPS Lycée")
mainWindow.geometry("1000x700")
mainWindow.configure(bg = backColor)

# Menu
menu = Frame(mainWindow, bg = backColor)
menu.pack(side = RIGHT, fill = Y)

labelMenu = Label(menu, text = "Menu", font = title, fg = accentDark, bg = backColor)
labelMenu.pack(fill = X)

icone = Canvas(menu, width = 100, height = 100, highlightbackground = backColor)
icone.pack()

# Cavenas
container = Frame(mainWindow)
container.pack(fill = BOTH, expand = 1)

graphe = {}

# ----- Onglets ----- #

tabList = ["Chemin", "Agenda", "Plan"]
selectedTab = 0

tabContainer = Frame(menu, bg = backColor, pady = 6)
tabContainer.pack(side = BOTTOM)

# Création de tableaux d'objets pour chaque onglet

ongletBoutons= []
ongletFrames = []
ongletDessin = []
ongletCanvas = []
ongletScrollbarHorizontal = []
ongletScrollbarVertical = []

for n, nom in enumerate(tabList):
    ongletBoutons.append(Button(tabContainer, text = nom, font = body, fg = accentDark, bg =
backColor, activebackground = accentDark, activeforeground = backColor, relief = FLAT,
overrelief = GROOVE, padx = 5, command = lambda onglet = n: Afficher(onglet)))
    ongletBoutons[n].grid(row=0,column=n)
    ongletFrames.append(Frame(menu, bg = backColor, padx = 10))
    ongletDessin.append(Frame(container))
    ongletCanvas.append(Canvas(ongletDessin[n], background = gris, highlightbackground =
backColor))
    ongletScrollbarHorizontal.append(Scrollbar(ongletDessin[n], orient = HORIZONTAL, command =
ongletCanvas[n].xview))
    ongletScrollbarVertical.append(Scrollbar(ongletDessin[n], orient = VERTICAL, command =
ongletCanvas[n].yview))
    ongletCanvas[n].configure(yscrollcommand = ongletScrollbarVertical[n].set, xscrollcommand =
ongletScrollbarHorizontal[n].set)
```

```
    ongletScrollbarHorizontal[n].grid(row = 1, column = 0, sticky = E + W)
    ongletScrollbarVertical[n].grid(row = 0, column = 1, sticky = N + S)
    ongletCanvas[n].grid(row = 0, column = 0, sticky = N+S+E+W)
    Grid.rowconfigure(ongletDessin[n], 0, weight = 1)
    Grid.columnconfigure(ongletDessin[n], 0, weight = 1)
    # icone = PhotoImage("Images/Icones/Icone" + nom)

def Afficher(onglet):
    global selectedTab
    ongletDessin[selectedTab].pack_forget()
    ongletDessin[onglet].pack(fill = BOTH, expand = 1)
    ongletBoutons[selectedTab].config(fg = accentDark, bg = backColor)
    ongletFrames[selectedTab].pack_forget()
    ongletBoutons[onglet].config(fg = backColor, bg = accentDark)
    ongletFrames[onglet].pack(fill = BOTH, expand = 1)
    selectedTab = onglet
    # imageAffichee = canvas.create_image(400, 400, image = icone)
    # canvas.icone = icone


# ----- Styles ----- #

boutonStyle = {"font" : body, "fg" : accentDark, "bg" : backColor, "activeforeground" :
backColor, "activebackground" : accentDark, "relief" : GROOVE, "padx" : 5}
labelStyle = {"font" : body, "fg" : accentDark, "bg" : backColor}
entryStyle = {"font" : body, "relief" : FLAT, "bg" : backColor}
radioButtonStyle = {"font" : body, "bg" : backColor, "fg" : accentDark}


# ----- Widgets Chemin ----- #

Label(ongletFrames[0], text = "Première salle", **labelStyle).pack()
entryNoeudDepart = Entry(ongletFrames[0], **entryStyle)
entryNoeudDepart.pack()
Label(ongletFrames[0], text = "Première salle", **labelStyle).pack()
entryNoeudArrive = Entry(ongletFrames[0], **entryStyle)
entryNoeudArrive.pack()
boutonGo = Button(ongletFrames[0], text = "Go !", **boutonStyle)
boutonGo.pack()
boutonChargement1 = Button(ongletFrames[0], text = "Charger", **boutonStyle)
boutonChargement1.pack(fill = X, expand = 1)


# ----- Widgets Agenda ----- #

part1 = Frame(ongletFrames[1], bg = backColor, pady = 10)
part1.pack(fill = BOTH, expand = 1)
part2 = Frame(ongletFrames[1], bg = backColor, pady = 10)
part2.pack(fill = BOTH, expand = 1)

frameConnection = Frame(part1, bg = accentDark, padx = 3, pady = 3)
frameConnection.pack(fill = X, expand = 1)
entryConnecter = Entry(frameConnection, **entryStyle)
entryConnecter.pack(fill = X)
boutonConnecter = Button(frameConnection, text = "Se connecter", **boutonStyle)
boutonConnecter.config(fg = backColor, bg = accentDark, relief = FLAT)
boutonConnecter.pack(fill = X)

frameButtons = Frame(part2, bg = backColor)
frameButtons.pack(fill = X, expand = 1)
boutonChercher = Button(frameButtons, text = "Chercher la prochaine salle", wraplength = 160,
**boutonStyle)
boutonChercher.pack(fill = X)
boutonSauvegarder = Button(frameButtons, text = "Sauvegarder", ** boutonStyle)
boutonSauvegarder.pack(fill = X)



# ----- Widgets Plan ----- #
```

```
modeSelectionne = StringVar()
labelframeMode = LabelFrame(ongletFrames[2], text = "Mode", **labelStyle)
labelframeMode.pack()
radiobuttonModeNoeuds = Radiobutton(labelframeMode, text = "Nœuds", variable = modeSelectionne,
value = "noeuds", **radioButtonStyle)
radiobuttonModeNoeuds.select()
radiobuttonModeNoeuds.pack(anchor = W)
radiobuttonModeLiaisons = Radiobutton(labelframeMode, text = "Liaisons", variable =
modeSelectionne, value = "liaisons", **radioButtonStyle)
radiobuttonModeLiaisons.pack(anchor = W)


typeNoeudSelectionne = StringVar()
labelframeTypeNoeud = LabelFrame(ongletFrames[2], text = "Type de nœud", **labelStyle)
labelframeTypeNoeud.pack()
radiobuttonTypeNoeudSalle = Radiobutton(labelframeTypeNoeud, text = "Salle", variable =
typeNoeudSelectionne, value = "salle", **radioButtonStyle)
radiobuttonTypeNoeudSalle.select()
radiobuttonTypeNoeudSalle.pack(anchor = W)
radiobuttonTypeNoeudIntermediaire = Radiobutton(labelframeTypeNoeud, text = "Intermédiaire",
variable = typeNoeudSelectionne, value = "intermediaire", **radioButtonStyle)
radiobuttonTypeNoeudIntermediaire.pack(anchor = W)

labelframeframeNomNoeud = LabelFrame(ongletFrames[2], text = "Nom de la salle", **labelStyle)
labelframeframeNomNoeud.pack()
entryNomNoeud = Entry(labelframeframeNomNoeud, width = 15, **entryStyle)
entryNomNoeud.pack()
entryNomNoeud.focus_set()
entryNomNoeudCouleurParDefaut = entryNomNoeud.cget("bg")

boutonChargementImage = Button(ongletFrames[2], text = "Charger image", **boutonStyle)
boutonChargementImage.pack()
boutonChargement2 = Button(ongletFrames[2], text = "Charger", **boutonStyle)
boutonChargement2.pack()


# ----- ONGLET CHEMIN ----- #

listeLiaisonsCheminChemin = []
noeudDepart = {}
noeudArrive = {}


# ----- Calcul de l'itinéraire ----- #

def cheminPlusCourt(depart, arrivee):
    # Visite de tous les chemins
    frontiere = []
    frontiere.append(depart)
    depuis = {}
    coutDeplacement = {}
    depuis[depart] = None
    coutDeplacement[depart] = 0

    while len(frontiere) > 0:
        actuel = frontiere.pop()

       # if actuel == arrivee: # Si on est arrivé, on stoppe la recherche
        #    break

        for suivant in graphe[actuel]:
            if suivant == "y" or suivant == "x" or suivant == "rayon":
                continue

            nouveauCoutDeplacement = coutDeplacement[actuel] + graphe[actuel][suivant]
            if suivant not in coutDeplacement or nouveauCoutDeplacement <
coutDeplacement[suivant]:
                coutDeplacement[suivant] = nouveauCoutDeplacement
                priorite = nouveauCoutDeplacement
                frontiere.insert(priorite, suivant)
```

```
                depuis[suivant] = actuel

    # Détermination du chemin le plus court
    actuel = arrivee
    chemin = [actuel]
    while actuel != depart:
        current = actuel = depuis[actuel]
        chemin.append(current)
    chemin.reverse()

    return chemin


# ----- Liste des noeuds ----- #

def dessinerCheminPlusCourt(noeud1, noeud2):
    global noeudDepart, noeudArrive, listeLiaisonsChemin

    if noeudDepart:
        ongletCanvas[0].delete(noeudDepart[0])
        ongletCanvas[0].delete(noeudDepart[1])
        ongletCanvas[0].delete(noeudArrive[0])
        ongletCanvas[0].delete(noeudArrive[1])
        for liaison in listeLiaisonsChemin:
            ongletCanvas[0].delete(liaison)

    listeLiaisonsChemin = []

    chemin = cheminPlusCourt(noeud1, noeud2)

    for idNoeud in range(len(chemin) - 1):
        listeLiaisonsChemin.append(dessinerLiaison(ongletCanvas[0], graphe, chemin[idNoeud],
chemin[idNoeud + 1], "red", 3))

    noeudDepart = dessinerNoeud(ongletCanvas[0], graphe, noeud1)
    noeudArrive = dessinerNoeud(ongletCanvas[0], graphe, noeud2)


# ----- Sauvegarde et chargement ----- #

def chargerGrapheChemin():
    global graphe
    fichier = chargerGraphe()

    graphe = fichier["graphe"]
    chargerImageSurCanvas(ongletCanvas[0], fichier["image"])


# ----- ONGLET AGENDA ----- #

# ----- Listes -----#

salles=["" for i in range(10*6)]                                        #Cette liste
contiendra la liste de toute les salles de l'emploi du temps
matieres = ["" for i in range(10*6)]                                    #Cette liste
contiendra la liste de toute les matieres de l'EDT

combo=[]                                                                #Cette liste
contiendra la liste de toute les combobox du tableau
entry = []                                                              #Cette liste
contiendra la liste de tout les Entry du tableau

user = ""                                                               #Utilisateur,
donc initalement vide


# ----- Création du tableau ----- #

subjects = ("Mathématiques","Physique-chimie","SVT","SI","H-G-
EDC","SES","Francais","Philo",'Espagnol',         #Les matières que l'on pourra sélectionner
```

```
            "Allemand","Anglais","Sport","AP","SPE","An. Euro")

Jours = ["Lundi","Mardi", "Mercredi", "Jeudi", "Vendredi", "Samedi",            #Les jours en
francais pour les afficher dans le tableau
"Monday","Tuesday","Wednesday","Thursday","Friday","Saturday", "Sunday"]        #En anglais
pour les utiliser et obtenir l'Indice du jour dans la semaine


for j in range(11):
#Une boucle pour chaque ligne, j est l'indice vertical

    for i in range(7):
#Une boucle pour chaque colonne, i est l'indice horizontal

        if i == 0 and j == 0:
#En haut à gauche, on laisse une case vide

            pass

        elif (j ==0 and i>0)  :
#On place les jours sur la première ligne :

            ongletCanvas[1].create_rectangle( (i+1)*(700/9), (j+1)*(700/13), (i+2)*(700/9), (j
+2)*(700/13),fill = accentDark)                  #On créé les cases
            Jours_heures = Label(ongletCanvas[1], text = Jours[i-1], bg = accentDark, font =
("Segoe UI",12))                                 #On y entre les jours
            Jours_heures.place(x = (i+1.5)*(700/9), y =(j+1.5)*(700/13), anchor = "center")

        elif i == 0:
#On place les heures sur la première colonne :

            ongletCanvas[1].create_rectangle( (i+1)*(700/9), (j+1)*(700/13), (i+2)*(700/9), (j
+2)*(700/13),fill = accentDark)                  #On créé les cases
            Jours_heures = Label(ongletCanvas[1], text = str(j+7)+" h à "+str(j+8)+" h", bg =
accentDark, font = ("Segoe UI",10))             #On y entre les jours
            Jours_heures.place(x = (i+1.5)*(700/9), y =(j+1.5)*(700/13), anchor = "center")

        else:
#Puis on créé la grille :

            ongletCanvas[1].create_rectangle((i+1)*(700/9), (j+1)*(700/13), (i+2)*(700/9), (j
+2)*(700/13),fill = "white")                    #On créé les cases
            mat_wid = ttk.Combobox(ongletCanvas[1], values=subjects , font = ("Segoe UI",8),
width =8)                                       #Puis le combobox pour rentrer les matières
            combo.append(mat_wid)
#Que l'on ajoute à la liste des combobox
            mat_wid.place(x = (i+1.5)*(700/9), y =(j+1.5)*(700/13)-15, anchor = "center")
            sal_wid = Entry(ongletCanvas[1], width = 10, font = ("Segoe UI",8))
#Et enfin les entry pour rentrer les salles
            entry.append(sal_wid)
#Que l'on ajoute à la liste des entry
            sal_wid.place(x = (i+1.5)*(700/9), y =(j+1.5)*(700/13)+12, anchor = "center")


# ----- Sauvegarde ----- #

def Sauvegarde():

    """Cette fonction sert à mettre à jour les listes qui contiennent les matière et les salles
de l'emploi du temps.
    Dans un second temps si un utilisateur est connecté alors le fichier lié à l'utilisateur
est mis à jour (voir créé)"""

    valeur = 0

    for j in range(60):
#Une boucle pour chaque case de l'emploi du temps

        salles[valeur] = (entry[valeur].get())
#On récupère chaque valeurs des entry et des combobox pour les stocker dans une liste
```

```
        matieres[valeur] = (combo[valeur].get())
        valeur += 1

    if user.strip() == "":
# 1 - aucun utilisateur connecté =>

        warn = Label(mainWindow, text = "Attention vous n'êtes pas\n connecté", bg ="white",
font = ("Segoe UI", 15))
        warn.place(x = 875, y  = 375, anchor = "center")
        warn.after(2000, lambda : warn.place_forget())

    else :
# 2 - utilisateur connecté =>

        with open(user + ".txt","w") as fichier:

            fichier.write(",".join(salles)+"..."+",".join(matieres))
#On met à jour les salles et les matières de l'emploi du temps dans celui-ci
            fichier.close()


# ----- Connexion ----- #

def Valider():

    """ Cette fonction sert à valider son nom d'utilisateur. Si l'utilisateur possède déjà un
emploi du temps,
    l'emploi du temps actuellement à l'écran est remplacé par celui de l'uti_lisateur."""

    global user, matieres, salles
#On récupère les variables globales pour les modifier
    user = entryConnecter.get()
#Récupération de l'identifiant de l'utilisateur

    if os.path.isfile((user + ".txt")) is True :        #A condition que l'utilsateur possède
un emploi du temps =>

        with open(user + ".txt","r") as Fichier_user:   #On ouvre le fichier lié à
l'utilisateur

            text= str(Fichier_user.read()).split("...")
#On récupère le texte dans le fichier lié à l'utilisateur.
            Fichier_user.close()

        salles = text[0].split(",")
#On récupère les salles inscrite dans le fichier
        matieres = text[1].split(",")
#Idem pour les matières

        for i in range(60):

            combo[i].delete(0, END)
#On enlève chaque case de l'emploi du temps une par une ...
            combo[i].insert(0,matieres[i])
#...Pour les remplacer par l'emlpoi du temps de l'utilisateur.
            entry[i].delete(0, END)
            entry[i].insert(0,salles[i])

    return user , matieres, salles
#On retourne les variables globales pour que leur modification ne soit pas temporaire


# ----- Chemin actuel ----- #

def Chercher_chemin():

    """Cette fonction permet à l'utilisateur de trouver le chemin le plus court pour aller à sa
prochaine salle de cours."""
```

```
    Sauvegarde()                                                    #Permet de mettre à
jour les données (salles, matières)
    Indice_heure = (int(strftime("%H")) -7)                         #Un indice variant
en fonction de l'heure
    Indice_jour = Jours.index(strftime("%A"))-6                     #Un indice variant
en fonction du jour

    if Indice_heure < 0 or  Indice_heure > 10 or Indice_jour > 5:   # 1 - Equivaut à i
l'utilisateur n'est pas en cours =>

        if Indice_heure < 0 :                                       #Ici, il est entre 0
H et 8H

            Indice_heure = 0                                        #Le prochain cours
est donc le jour même à 8H

        elif  Indice_heure > 10 :                                   #Ici, il est entre
18H et 24H

            Indice_heure = 0                                        #Le prochain cours
est donc le jour suivant à 8H
            Indice_jour+=1
        if Indice_jour > 5:                                         #ici, on est
dimanche.
            Indice_heure = 0                                        #Le prochain cours
est donc le lundi 8H
            Indice_jour = 0
        Indice_final = 6*Indice_heure+Indice_jour                   #Cette variable
correspond à l'indice de l'entry de la prochaine heure de cours
        salle = ["HALL", salles[Indice_final]]                  #Si l'élève n'a pas cours,
alors il part du hall jusqu'à sa salle

    else :                                                          # 2 - L'élève est en
cours
        Indice_final = 6*Indice_heure+Indice_jour
        salle = [salles[Indice_final-6], salles[Indice_final]]      #Il doit donc aller
de sa salle actuelle à la prochaine salle.

    dessinerCheminPlusCourt(salle[0], salle[1])

    Afficher(0)


# ----- ONGLET PLAN ----- #

graphe = {}
listeNoeuds = {}        # Stocke les graphismes des nœuds
listeLiaisons = {}      # Stocke les graphismes des liaisons
noeudSelectionne = {}   # Stocke le nom et les graphismes du nœud sélectionné
idNoeudIntermedaire = 0 # Nombre de nœuds intermédiaires

# Identifiants des boutons de souris
ID_CLIC_GAUCHE = "<Button-1>"
ID_CLIC_DROIT = "<Button-2>" if os.sys.platform == "darwin" else "<Button-3>" # Compatibilité
OSX


# ----- Chargement du plan ----- #

imageAffichee = PhotoImage()

def chargerImage(canvas, cheminImage):
    global imageAffichee
    if imageAffichee is not None:
        canvas.delete(imageAffichee)

    nouveauGraphe()
    imageAffichee = chargerImageSurCanvas(canvas, cheminImage)
```

```
# ----- Gestion des nœuds ----- #

def noeudClique(souris):
    for noeud in graphe:
        if noeud[0] == "_":
            rayon = RAYON_NOEUD_INTERMEDIAIRE
        else:
            rayon = RAYON_NOEUD

        if (graphe[noeud]["x"] >= souris.x - rayon and
        graphe[noeud]["x"] <= souris.x + rayon and
        graphe[noeud]["y"] >= souris.y - rayon and
        graphe[noeud]["y"] <= souris.y + rayon):
            return noeud
            break


def selectionnerNoeud(nomNoeud): # Clic gauche
    if nomNoeud == "":
        return

    deselectionnerNoeud()

    noeudSelectionne["nom"] = nomNoeud
    noeudSelectionne["dessin"] = dessinerNoeud(
        ongletCanvas[2],
        graphe,
        nomNoeud,
        "green",
    )


def deselectionnerNoeud():
    global noeudSelectionne
    if noeudSelectionne:
        if noeudSelectionne["nom"][0] == "_":
            ongletCanvas[2].delete(noeudSelectionne["dessin"])
        else:
            ongletCanvas[2].delete(noeudSelectionne["dessin"][0])
            ongletCanvas[2].delete(noeudSelectionne["dessin"][1])

        noeudSelectionne = {}


def ajouterNoeud(nomNoeud, x, y): # Clic droit
    if nomNoeud in graphe:
        entryNomNoeud.configure(bg = "red")
        entryNomNoeud.after(500, lambda: entryNomNoeud.configure(bg =
entryNomNoeudCouleurParDefaut))
        return

    graphe[nomNoeud] = {}
    graphe[nomNoeud]["x"] = x
    graphe[nomNoeud]["y"] = y

    listeNoeuds[nomNoeud] = dessinerNoeud(ongletCanvas[2], graphe, nomNoeud)


def supprimerNoeud(nomNoeud):

    if nomNoeud[0] == "_":
        ongletCanvas[2].delete(listeNoeuds[nomNoeud])
    else:
        ongletCanvas[2].delete(listeNoeuds[nomNoeud][0])
        ongletCanvas[2].delete(listeNoeuds[nomNoeud][1])

    noeuds = tuple(graphe[nomNoeud].keys()) # Nécessaire sinon graphe[nomNoeud] change pendant
l'itération
    for noeud in noeuds:
        if noeud != "x" and noeud != "y":
```

```
            delier(nomNoeud, noeud)
    del listeNoeuds[nomNoeud]
    del graphe[nomNoeud]


# ----- Gestion des liaisons ----- #

def clicSurLiaison(event):
    event.y = ongletCanvas[2].canvasy(event.y) # Position sur le canvas
    event.x = ongletCanvas[2].canvasx(event.x)

    foo = event.widget.find_closest(event.x, event.y)[0]
    for liaison in listeLiaisons.keys():
        if listeLiaisons[liaison] == foo:
            delier(liaison[0], liaison[1])
            return


def lier(noeud1, noeud2):
    if noeud1 == noeud2:
        return

    idLiaison = [noeud1, noeud2]
    idLiaison.sort()
    idLiaison = tuple(idLiaison)
    if idLiaison in listeLiaisons:
        return

    distanceEntreNoeuds = round(math.sqrt(
        pow(graphe[noeud1]["x"] - graphe[noeud2]["x"], 2) +
        pow(graphe[noeud1]["y"] - graphe[noeud2]["y"], 2))
    )

    graphe[noeud1][noeud2] = graphe[noeud2][noeud1] = distanceEntreNoeuds

    listeLiaisons[idLiaison] = dessinerLiaison(ongletCanvas[2], graphe, noeud1, noeud2,
"black", 3)

    # Permet de redessiner les ronds au dessus de la liaison
    if noeud1[0] == "_":
        ongletCanvas[2].tag_raise(listeNoeuds[noeud1])
    else:
        ongletCanvas[2].tag_raise(listeNoeuds[noeud1][0])
        ongletCanvas[2].tag_raise(listeNoeuds[noeud1][1])

    if noeud2[0] == "_":
        ongletCanvas[2].tag_raise(listeNoeuds[noeud2])
    else:
        ongletCanvas[2].tag_raise(listeNoeuds[noeud2][0])
        ongletCanvas[2].tag_raise(listeNoeuds[noeud2][1])

    if noeudSelectionne["nom"][0] == "_":
        ongletCanvas[2].tag_raise(noeudSelectionne["dessin"])
    else:
        ongletCanvas[2].tag_raise(noeudSelectionne["dessin"][0])
        ongletCanvas[2].tag_raise(noeudSelectionne["dessin"][1])

    ongletCanvas[2].tag_bind(listeLiaisons[idLiaison], ID_CLIC_DROIT, clicSurLiaison)


def delier(noeud1, noeud2):
    del graphe[noeud1][noeud2]
    del graphe[noeud2][noeud1]

    idLiaison = [noeud1, noeud2]
    idLiaison.sort()
    idLiaison = tuple(idLiaison)

    ongletCanvas[2].delete(listeLiaisons[idLiaison])
    del listeLiaisons[idLiaison]
```

```
# ----- Actions ----- #

def clicGauche(souris):
    global idNoeudIntermedaire

    souris.y = ongletCanvas[2].canvasy(souris.y) # Position sur le canvas
    souris.x = ongletCanvas[2].canvasx(souris.x)

    if modeSelectionne.get() == "noeuds":
        nomNoeud = entryNomNoeud.get()

        if typeNoeudSelectionne.get() == "salle":
            if nomNoeud:
                ajouterNoeud(nomNoeud, souris.x, souris.y)
                entryNomNoeud.delete(0, len(entryNomNoeud.get()))
        else:
            ajouterNoeud("_" + str(idNoeudIntermedaire), souris.x, souris.y)
            idNoeudIntermedaire += 1
    else:
        nomNoeud = noeudClique(souris)
        if not noeudSelectionne:
            if nomNoeud:
                selectionnerNoeud(nomNoeud)
        else:
            if nomNoeud:
                lier(noeudSelectionne["nom"], nomNoeud)
            deselectionnerNoeud()


def clicDroit(souris):
    souris.y = ongletCanvas[2].canvasy(souris.y) # Position sur le canvas
    souris.x = ongletCanvas[2].canvasx(souris.x)

    if modeSelectionne.get() == "noeuds":
        nomNoeud = noeudClique(souris)
        if nomNoeud:
            supprimerNoeud(nomNoeud)


ongletCanvas[2].bind(ID_CLIC_GAUCHE, clicGauche)
ongletCanvas[2].bind(ID_CLIC_DROIT, clicDroit)


# ----- Sélection du mode d'édition ----- #

def changementDeMode():
    if modeSelectionne.get() == "noeuds":
        deselectionnerNoeud()
        radiobuttonTypeNoeudSalle.config(state = NORMAL)
        radiobuttonTypeNoeudIntermediaire.config(state = NORMAL)
    else:
        radiobuttonTypeNoeudSalle.config(state = DISABLED)
        radiobuttonTypeNoeudIntermediaire.config(state = DISABLED)


# ----- Sauvegarde & Chargement ----- #

def sauvegarder():
    sauvegarderGraphe(graphe, ongletCanvas[2].cheminImage)


def nouveauGraphe():
    global graphe, listeNoeuds, listeLiaisons

    for noeud in listeNoeuds:
        if noeud[0] == "_":
            ongletCanvas[2].delete(listeNoeuds[noeud])
        else:
```

```
            ongletCanvas[2].delete(listeNoeuds[noeud][0])
            ongletCanvas[2].delete(listeNoeuds[noeud][1])

    for liaison in listeLiaisons.keys():
        delier(liaison[0], liaison[1])

    graphe = listeNoeuds = listeLiaisons = {}


def charger():
    global graphe, listeNoeuds, listeLiaisons, idNoeudIntermedaire

    fichier = chargerGraphe()

    if not fichier:
        return

    chargerImage(ongletCanvas[2], fichier["image"])
    chargerImage(ongletCanvas[0], fichier["image"])
    graphe = fichier["graphe"]

    for noeud in graphe:
        for voisin in graphe[noeud]:
            if voisin == "x" or voisin == "y":
                continue
            idLiaison = [noeud, voisin]
            idLiaison.sort()
            idLiaison = tuple(idLiaison)
            if idLiaison not in listeLiaisons:
                listeLiaisons[idLiaison] = dessinerLiaison(ongletCanvas[2], graphe, noeud,
voisin, "black", 3)
                ongletCanvas[2].tag_bind(listeLiaisons[idLiaison], ID_CLIC_DROIT,
clicSurLiaison)

    for noeud in graphe: # Pour dessiner les ronds au dessus des liaisons
        listeNoeuds[noeud] = dessinerNoeud(ongletCanvas[2], graphe, noeud)

    if noeud[0] == "_":
        idNoeud = int(noeud[1:])
        if idNoeud > idNoeudIntermedaire:
            idNoeudIntermedaire = idNoeud


# ----- Commandes boutons ----- #

boutonGo.configure(command = lambda: dessinerCheminPlusCourt(entryNoeudDepart.get(),
entryNoeudArrive.get()))
boutonChargement1.configure(command = charger)
boutonConnecter.configure(command = Valider)
boutonChercher.configure(command = Chercher_chemin)
boutonSauvegarder.configure(command = Sauvegarde)
boutonChargementImage.configure(command = lambda: chargerImage(ongletCanvas[2], None))
boutonChargement2.configure(command = charger)

radiobuttonModeNoeuds.configure(command = changementDeMode)
radiobuttonModeNoeuds.configure(command = changementDeMode)

# ----- Lancement de l'application ----- #
Afficher(selectedTab)

mainWindow.mainloop()
```