maintk.py

```
import tkinter as tk
import json
from tkinter.messagebox import showinfo
from PIL import ImageTk, Image

TRANSCOLOUR = ''

class  Application(tk.Frame):
    """ Classe principale de l'application"""
    def __init__(self, master=None):
        super().__init__(master)
        self.niveau = "débutant"
        screen_width = root.winfo_screenwidth()-60
        screen_height = root.winfo_screenheight()-100
        self.root = root
        self.root.attributes('-fullscreen', True)
        self.root.title("GPS Courchevel")
        self.image_path = "data/plan-pistes.jpg"
        self.image = Image.open(self.image_path)
        self.photo = ImageTk.PhotoImage(self.image)
        self.canvas = tk.Canvas(self.root, width=screen_width, height=screen_height,
bg="black",
                                scrollregion=(0, 0, self.image.width, self.image.height))
        self.canvas.grid(row=1, column=0, sticky=tk.N+tk.S+tk.E+tk.W)
        self.canvas.create_image(0, 0, anchor=tk.NW, image=self.photo)
        self.time_trajet = 0
        self.result = tk.Label(self.root, text="", bg="white", fg="black", font=("Helvetica",
16))

        tk.Button(self.root, text="Reset", command=self.reset).grid(row=0, column=1)

        self.x_scrollbar = tk.Scrollbar(self.root, orient=tk.HORIZONTAL,
                                        command=self.canvas.xview, width= 40)
        self.x_scrollbar.grid(row=2, column=0, sticky=tk.E+tk.W)
        self.y_scrollbar = tk.Scrollbar(self.root, orient=tk.VERTICAL,
                                        command=self.canvas.yview, width= 40)
        self.y_scrollbar.grid(row=1, column=1, sticky=tk.N+tk.S)
        self.canvas.config(xscrollcommand=self.x_scrollbar.set,
yscrollcommand=self.y_scrollbar.set)
        bar = tk.Menu(self.root)
        self.root.config(menu=bar)

        niveau = tk.Menu(bar, tearoff=0)
        niveau.add_command(label="Débutant", command=self.debutant)
        niveau.add_command(label="Confirmé", command=self.confimé)
        bar.add_cascade(label="Niveau", menu=niveau)
        tools = tk.Menu(bar, tearoff=0)
        tools.add_command(label="Exit", command=self.root.quit)
        tools.add_command(label="Montrer les pistes", command=self.show_piste)
        bar.add_cascade(label="Outils", menu=tools)



        self.root.columnconfigure(0, weight=1)
        self.root.rowconfigure(0, weight=1)



        self.noeuds = []
```

```
        self.pistes = []
        self.chemins = []

        # Ajouter les noeuds sur le canvas
        for noeud in liste_noeuds:
            x, y = noeud.coord
            noeud_obj = self.canvas.create_oval(x-10, y-10, x+10, y+10, fill="orange",
                                                outline="black", width=2, tags="noeud")
            self.noeuds.append(noeud)
            noeud.point = noeud_obj

        # Ajouter les pistes sur le canvas
        for piste in liste_pistes:
            piste_obj = []
            for i in range(len(piste.coords)-1):
                xi, yi = piste.coords[i]
                xi1, yi1 = piste.coords[i+1]
                piste_obj.append(self.canvas.create_line(xi, yi, xi1, yi1,
                                                         fill =TRANSCOLOUR, width=5, tags =piste.nom))
            self.pistes.append(piste)
            piste.segment = piste_obj

        # Ajouter les chemins sur le canvas
        self.canvas.bind("<Button-1>", self.clic_droit)



    def confimé(self):
        """ change le niveau de difficulté des pistes et des noeuds"""
        data.niveau = "confirmé"
        vitesse = {"green": 55, "blue": 60, "red": 80, "black": 150, "grey" : 80}
        for piste in self.pistes:
            piste.dure = round(piste.longueur / (vitesse[piste.couleur]/piste.cat),2)

    def debutant(self):
        """ change le niveau de difficulté des pistes et des noeuds"""
        data.niveau = "débutant"
        vitesse = {"green": 50, "blue": 45, "red": 40, "black": 35 , "grey" : 80}
        for piste in self.pistes:
            piste.dure = round(piste.longueur / (vitesse[piste.couleur]/piste.cat),2)

    def find_noeud(self, x, y):
        """ retourne le noeud sur lequel on a cliqué, ou None si on n'a pas cliqué sur un
noeud"""
        for noeud in self.noeuds:
            x1,y1,x2,y2= self.canvas.coords(noeud.point)
            if x1  < x <  x2 and y1  < y < y2:
                return noeud
        return None


    def clic_droit(self,event):
        """ gère le clic droit de la souris: si on a cliqué sur un noeud,
        on l'ajoute à la liste des chemins, sinon on affiche un message d'erreur"""
        if len(self.chemins) % 2 == 1 or len(self.chemins) == 0:
            x, y =(event.x + self.image.width*self.x_scrollbar.get()[0],
                   event.y + self.image.height*self.y_scrollbar.get()[0])
            noeud = self.find_noeud(x, y)
            if noeud is not None:
                self.chemins.append(noeud)
```

```
            else:
                pass
        if len(self.chemins) % 2 == 0 and len(self.chemins) != 0:
            self.trajet()


    def trajet(self,):
        """ gère l'affichage du trajet entre deux noeuds"""

        debut = self.chemins[0]
        fin = self.chemins[1]
        self.dijkstra(debut,fin)


    def dijkstra(self, depart, arrivee):
        """Calcule le plus court chemin entre deux noeuds avec l'algorithme de Dijkstra"""

        if depart == arrivee:
            showinfo("attention", "Vous êtes déjà sur place")
            self.reset()
            return

        # Initialisation

        for noeud in self.noeuds:
            noeud.distance = float("inf")
            noeud.precedent = None

        for vois in depart.voisins:
            longueur = data.get_piste(depart, vois).dure
            vois.distance = longueur
            vois.precedent = depart

        depart.distance = 0
        file = []

        # Boucle principale
        while True:
            self.noeuds.sort(key = lambda x: x.distance)
            noeud = self.noeuds.pop(0)
            file.append(noeud)
            if noeud == arrivee:
                break
            for vois in noeud.voisins:

                longueur = data.get_piste(noeud, vois).dure

                if noeud.distance + longueur < vois.distance:
                    vois.distance = noeud.distance + longueur
                    vois.precedent = noeud
        # reinitialisation des noeuds
        for noeud in file:
            self.noeuds.append(noeud)

        # Affichage du chemin
        noeud = arrivee
        if noeud.precedent is None:
            showinfo("attention", "Vous ne pouvez pas atteindre cette destination")
            self.reset()
            return
```

```
        while noeud.precedent is not None:
            self.canvas.itemconfig(noeud.point, fill="yellow", outline="yellow", width=2)
            piste = data.get_piste(noeud.precedent, noeud)
            for trait in piste.segment:
                self.canvas.itemconfig(trait, fill=piste.couleur, width=4 )
            noeud = noeud.precedent
        self.canvas.itemconfig(noeud.point, fill="yellow", outline="yellow", width=2)

        self.chemins = []
        self.time_trajet += round(arrivee.distance,2)
        self.result.config(text="Durée du trajet: " + str(self.time_trajet) + " min")
        self.result.grid(row=0, column=0, sticky="nsew")

    def reset(self):
        """Remet le canvas à son état initial"""
        for noeud in self.noeuds:
            self.canvas.itemconfig(noeud.point, fill="orange", outline="black", width=1)
        for piste in self.pistes:
            for trait in piste.segment:
                self.canvas.itemconfig(trait, fill=TRANSCOLOUR, width=5)
        self.chemins = []
        self.time_trajet = 0
        self.result.grid_forget()

    def show_piste(self):
        for piste in self.pistes:
            for trait in piste.segment:
                self.canvas.itemconfig(trait, fill=piste.couleur, width=5)



fichier = json.load(open("data/data.json","r"))


class Noeuds ():
    """Classe qui représente un noeud du graphe"""
    def __init__(self, nom,coord, point = None ) -> None:
        self.nom = nom
        self.voisins = list()
        self.coord = coord
        self.point = point

    def __repr__(self) -> str:
        return self.nom

class Pistes():
    """Classe qui représente une piste de ski"""
    def __init__(self,nom, couleur, noeud_d, noeud_f, longueur, coords,cat, segment = None ) -
> None:
        vitesse = {"green": 50, "blue": 45, "red": 40, "black": 35 , "grey" : 80}
        self.nom = nom
        self.couleur = couleur
        self.cat = int(cat)
        self.depart = noeud_d
        self.fin = noeud_f
        self.longueur = longueur
        self.dure = round(longueur / (vitesse[self.couleur]/self.cat), 2)
        self.coords = coords
        self.segment = segment
```

```
    def __repr__(self) -> str:
        return self.nom


class Data():
    """Classe qui représente l'ensemble des données du graphe"""
    def __init__(self, noeuds, pistes) -> None:
        self.noeuds = list(noeuds)
        self.pistes = list(pistes)
        self.niveau = "débutant"
        for piste in self.pistes:
            piste.depart = self.get_noeud(piste.depart)
            piste.fin = self.get_noeud(piste.fin)
        for noeud in self.noeuds:
            self.voisin(noeud)


    def get_noeud(self, nom):
        """Renvoie le noeud correspondant au nom donné"""
        for noeud in self.noeuds:
            if noeud.nom == nom:
                return noeud
        return None


    def get_piste(self, debut, fin):
        """Renvoie la piste la plus rapide entre les deux noeuds donnés"""
        pistes = []
        for piste in self.pistes:
            if piste.depart == debut and piste.fin == fin:
                pistes.append(piste)
        pistes.sort(key = lambda x: x.longueur)
        return pistes[0]


    def voisin(self, noeud):
        """Ajoute les voisins d'un noeud"""
        for piste in self.pistes:
            if piste.depart == noeud:
                noeud.voisins.append(piste.fin)


liste_noeuds = []
for elem in fichier["noeuds"]:
    liste_noeuds.append(Noeuds(elem["name"],(elem["x"],elem["y"])))

liste_pistes = []
for elem in fichier["pistes"]:
    liste_pistes.append(Pistes(elem["name"], elem["couleur"], elem["noeud_depart"],
                               elem["noeud_fin"], elem["longueur"], coords = elem["coords"],
cat = elem["remonte"]))

data = Data(liste_noeuds, liste_pistes)
root = tk.Tk()


app = Application(master=root)
app.mainloop()
```