from tkinter import \*

from tkinter import filedialog

from tkinter.messagebox import \*

from tkinter import ttk

import json

import csv

import os

oldx=0

oldy=0

iter=0

l=[]

class  application:

   def \_\_init\_\_(*self*):

*self*.window = Tk()

*self*.savefile={1:""}

*self*.file=None

*self*.list\_top=[]

*self*.dic\_color = {1: 'red', 2: 'blue', 3: 'yellow', 4: 'green', 5: 'grey', 6: 'pink' ,7: 'orange',8: 'purple', 9: 'cyan'}

   def personalise(*self*):

*self*.window.title("Application")

*self*.window.geometry("750x450")

   def nouveau(*self*):

*self*.fram = Toplevel(*self*.window)

*self*.list\_top.append(*self*.fram)

*self*.fram.protocol("WM\_DELETE\_WINDOW",*self*.fermer)

*self*.savefile[1]=""

*self*.fram.transient(*self*.window)

*self*.fram.focus\_set()

*self*.fram.geometry("550x300+200+250")

*self*.fram.title("Untitled.py")

*self*.canvas= Canvas(*self*.fram, *background*="blue")

*self*.canvas.pack(*side*=TOP, *fill*=BOTH, *expand*=1)

*self*.indice\_sommet=0

*self*.indice\_arete=0

*self*.list\_sommet=[]

*self*.list\_arete=[]

*self*.mod={}

*self*.mod['ls']=*self*.list\_sommet

*self*.mod['la']=*self*.list\_arete

   def get\_sommet(*self*):

      l=[]

      A = *self*.list\_sommet

      for i in A:

         l.append(int(i[0]))

      return l

   def get\_arete(*self*):

      l=[]

      B = *self*.list\_arete

      j=0

      for i in B:

         j+=1

         l.append(int(j))

      return l

   def affichagerdegre(*self*):

*self*.attrubie\_couleur()

      B=[]

      B=*self*.list\_sommet

      B.sort()

      B.reverse()

      i=100

      A=[]

      while i>=0:

         for j in B:

            if j[3]==i:

               A.append(j)

         i-=1

      B.sort()

      R=*self*.dic\_color

      popup = Toplevel(*self*.fram)

      popup.title("Matrice de degre oriente")

      popup.geometry("550x250+500+250")

      popup.transient(*self*.fram)

      mat=ttk.Treeview(popup)

      mat['columns']=("Liste des sommets","degre de chaque sommet","numero de couleur","nom de couleur")

      mat.column("#0",*width*=15,*minwidth*=10)

      mat.column("Liste des sommets",*width*=120)

      mat.column("degre de chaque sommet",*width*=160)

      mat.column("numero de couleur", *width*=120)

      mat.column("nom de couleur", *width*=120)

      mat.heading("#0",*text*="")

      mat.heading("Liste des sommets",*text*="Liste des sommets")

      mat.heading("degre de chaque sommet",*text*="degre de chaque sommet")

      mat.heading("numero de couleur", *text*="numero de couleur")

      mat.heading("nom de couleur", *text*="nom de couleur")

      ind=0

      for i in A:

         if i[4]==0:

            mat.insert(*parent*='', *index*='end', *iid*=i, *text*="", *values*=(str(i[0]), str(i[3]), "NULL", "pas de couleur"))

         else:

            mat.insert(*parent*='',*index*='end',*iid*=i,*text*="",*values*=(str(i[0]),str(i[3]),str(i[4]),R[i[4]]))

         ind+=1

      mat.pack(*pady*=20)

   def degree\_non\_oriente(*self*):

*self*.degree()

      C=*self*.list\_sommet

      popup = Toplevel(*self*.fram)

      popup.title("Matrice de degre non oriente")

      popup.geometry("350x250+200+250")

      popup.transient(*self*.fram)

      mat=ttk.Treeview(popup)

      mat['columns']=("Liste des sommets","degre de chaque sommet")

      mat.column("#0",*width*=15,*minwidth*=10)

      mat.column("Liste des sommets",*width*=120)

      mat.column("degre de chaque sommet",*width*=200)

      mat.heading("#0",*text*="")

      mat.heading("Liste des sommets",*text*="Liste des sommets")

      mat.heading("degre de chaque sommet",*text*="degre de chaque sommet")

      ind=0

      for i in C:

         mat.insert(*parent*='',*index*='end',*iid*=i,*text*="",*values*=(str(i[0]),str(i[3])))

         ind+=1

      mat.pack(*pady*=20)

   def voisin(*self*):

      B = *self*.list\_arete

      voisin = []

      for i in B:

         l =[i[1][0], i[2][0]]

         voisin.append(l)

      return voisin

   def creer\_matrice\_ad(*self*):

      A = *self*.list\_sommet

      C = *self*.voisin()

      matrice={}

      for i in A:

         adjacence= []

         for j in A:

            l1=[i[0], j[0]]

            l2=[j[0], i[0]]

            if (l1 in C) or (l2 in C):

               if i[1]==j[1] and i[2]==j[2]:

                  adja=2

               else:

                  adja=1

            else:

               adja=0

            adjacence.append(adja)

         matrice[int(i[0])]= tuple(adjacence)

      return matrice

   def matrice\_ad(*self*):

      popup = Toplevel(*self*.fram)

      if *self*.savefile[1]=="":

         popup.title("Matrice d'adjacence de votre nouveau fichier")

      else:

         popup.title("Matrice d'adjacence de "+(os.path.basename(*self*.savefile[1])))

      popup.geometry("290x250+200+250")

      popup.transient(*self*.fram)

      matrice = *self*.creer\_matrice\_ad()

      list\_sommet = *self*.get\_sommet()

      for i in range(len(list\_sommet)):

         list\_sommet[i] = str(list\_sommet[i])

      mat= ttk.Treeview(*master*=popup)

      mat['columns'] = tuple(list\_sommet)

      mat.column("#0",*width*=30,*minwidth*=20)

      for i in list\_sommet:

         mat.column(i,*width*=30,*minwidth*=20)

      mat.heading("#0",*text*="  ")

      for i in list\_sommet:

         mat.heading(i,*text*=str(i))

      list\_sommet = *self*.get\_sommet()

      for i in list\_sommet:

         mat.insert(*parent*="",*index*="end",*iid* = i,*text*=str(i)+" ",*values*=matrice[i])

      mat.pack(*expand*=1, *fill*=BOTH)

   def creer\_matrice\_in(*self*):

      A = *self*.list\_sommet

      B = *self*.list\_arete

      matrice={}

      for i in A:

         incidence= []

         for j in B:

            if i[0]==j[1][0] or i[0]==j[2][0]:

               if i[1]==j[1][1] and i[2]==j[2][2]:

                  incid=2

               else:

                  incid=1

            else:

               incid=0

            incidence.append(incid)

         matrice[int(i[0])]= tuple(incidence)

      return matrice

   def matrice\_in(*self*):

      popup = Toplevel(*self*.fram)

      if *self*.savefile[1]=="":

         popup.title("Matrice d'incidence de votre nouveau fichier")

      else:

         popup.title("Matrice d'incidence de "+(os.path.basename(*self*.savefile[1])))

      popup.geometry("400x250+500+250")

      popup.transient(*self*.fram)

      matrice = *self*.creer\_matrice\_in()

      list\_arete = *self*.get\_arete()

      list\_sommet = *self*.get\_sommet()

      for i in range(len(list\_sommet)):

         list\_sommet[i] = str(list\_sommet[i])

      mat= ttk.Treeview(*master*=popup)

      mat['columns'] = tuple(list\_arete)

      mat.column("#0",*width*=30,*minwidth*=20)

      for i in list\_arete:

         mat.column(i,*width*=30,*minwidth*=20)

      mat.heading("#0",*text*="  ")

      for i in list\_arete:

         mat.heading(i,*text*="e"+str(i))

      list\_sommet = *self*.get\_sommet()

      for i in list\_sommet:

         mat.insert(*parent*="",*index*="end",*iid* = i,*text*=str(i)+" ",*values*=matrice[i])

      mat.pack(*expand*=1, *fill*=BOTH)

   def vide(*event*):

      pass

   def ouvrir(*self*):

      file = filedialog.askopenfilename( *title*="Ouvrir un fichier",

*filetypes*=[("Python files","\*.py")])

      if file!="":

*self*.savefile[1] = file

*self*.fram = Toplevel(*self*.window)

*self*.fram.protocol("WM\_DELETE\_WINDOW",*self*.fermer)

*self*.fram.title(os.path.basename(file))

*self*.list\_top.append(*self*.fram)

*self*.fram.transient(*self*.window)

*self*.fram.focus\_set()

*self*.fram.geometry("550x300+200+250")

*self*.canvas= Canvas(*self*.fram, *background*="blue")

*self*.canvas.pack(*side*=TOP, *fill*=BOTH, *expand*=1)

*self*.indice\_sommet=0

*self*.indice\_arete=0

*self*.list\_sommet=[]

*self*.list\_arete=[]

*self*.mod={}

         with open(file, 'r') as f:

*self*.mod= json.load(f)

         dic = {}

         with open(file, 'r') as f:

            dic = json.load(f)

*self*.list\_sommet = dic['ls']

*self*.list\_arete = dic['la']

         for j in *self*.list\_arete:

            if j[1][1]==j[2][1] and j[1][2]==j[2][2]:

*self*.canvas.create\_oval(j[1][1], j[1][2],j[1][1]-30, j[1][2]-30)

               center\_x, center\_y = (j[1][1]-30, j[1][2]-30)

            else:

*self*.canvas.create\_line(j[1][1],j[1][2], j[2][1], j[2][2])

               center\_x, center\_y = ((j[1][1]) + (j[2][1])) / 2, ((j[1][2] + 15) + (j[2][2] + 15)) / 2

*self*.indice\_arete+=1

*self*.canvas.create\_text(center\_x, center\_y, *text*=j[0])

         for i in *self*.list\_sommet:

*self*.canvas.create\_oval(i[1]-10, i[2]-10, i[1]+10, i[2]+10, *fill*="red")

*self*.indice\_sommet+=1

*self*.canvas.create\_text(i[1],i[2], *text*=i[0], *fill*="white")

   def enregistrer(*self*):

      dic = {}

      dic['ls']=*self*.list\_sommet

      dic['la']=*self*.list\_arete

      if *self*.savefile[1] == "":

*self*.enregistrer\_sous()

      else:

         with open(*self*.savefile[1], 'w+') as f:

            json.dump(dic, f)

*self*.mod={}

         with open(*self*.savefile[1], 'r') as f:

*self*.mod= json.load(f)

   def enregistrer\_sous(*self*):

      file = filedialog.asksaveasfilename(*initialfile*='Untitled.py',

*defaultextension*="\*.py",

*filetypes*=[("Fichier python", "\*.py")])

      if file!="":

*self*.fram.title(os.path.basename(file))

         dic = {}

         dic['ls']=*self*.list\_sommet

         dic['la']=*self*.list\_arete

         with open(file, 'w+') as f:

                     json.dump(dic, f)

*self*.savefile[1]=file

*self*.mod={}

         with open(file, 'r') as f:

*self*.mod= json.load(f)

   def fermer(*self*):

      if len(*self*.list\_top)==0 :

*self*.window.quit()

      else:

         if (len(*self*.list\_sommet)!=0 or len(*self*.list\_arete)!=0) and *self*.savefile[1]=="":

            bns=askyesnocancel("Ferme","Voulez-vous enregistre la modification apportee à nouveau fichier  ?")

            if bns:

*self*.enregistrer()

               n=len(*self*.list\_top)

*self*.list\_top[n-1].destroy()

*self*.list\_top.remove(*self*.list\_top[n-1])

            elif bns!=None:

               n=len(*self*.list\_top)

*self*.list\_top[n-1].destroy()

*self*.list\_top.remove(*self*.list\_top[n-1])

         elif  *self*.list\_sommet!=*self*.mod['ls'] or *self*.list\_arete!=*self*.mod['la']:

            ans=askyesnocancel("Ferme","Voulez-vous enregistre la modification apportee à "+(os.path.basename(*self*.savefile[1]))+" ?")

            if ans:

*self*.enregistrer()

               n=len(*self*.list\_top)

*self*.list\_top[n-1].destroy()

*self*.list\_top.remove(*self*.list\_top[n-1])

            elif ans!=None:

               n=len(*self*.list\_top)

*self*.list\_top[n-1].destroy()

*self*.list\_top.remove(*self*.list\_top[n-1])

         else:

            n=len(*self*.list\_top)

*self*.list\_top[n-1].destroy()

*self*.list\_top.remove(*self*.list\_top[n-1])

   def ajoute\_point(*self*,*event*):

*self*.canvas.focus\_set()

      x = *event*.x

      y = *event*.y

      vide=True

      for i in range(len(*self*.list\_sommet)):

         sx=*self*.list\_sommet[i][1]

         sy=*self*.list\_sommet[i][2]

         if (x > sx-15) and (x < sx+15) and (y > sy-15)  and (y < sy+15):

            vide=False

      if vide:

*self*.canvas.create\_oval(x-10, y-10, x+10, y+10, *fill*='red')

*self*.indice\_sommet+=1

*self*.canvas.create\_text(x,y, *text*=str(*self*.indice\_sommet), *fill*='white')

         l=[str(*self*.indice\_sommet),x,y,0,0]

*self*.list\_sommet.append(l)

   def sommet(*self*):

*self*.canvas.bind("<Button-1>", *self*.ajoute\_point)

   def tracer(*self*, *event*):

      global iter, l

      global oldx,oldy

      if iter==0:

         oldx,oldy = *event*.x,*event*.y

         for i in range(len(*self*.list\_sommet)):

            osx=*self*.list\_sommet[i][1]

            osy=*self*.list\_sommet[i][2]

            if (oldx > osx-10) and (oldx < osx+10) and (oldy > osy-10) and (oldy < osy+10):

               oldx, oldy=*self*.list\_sommet[i][1], *self*.list\_sommet[i][2]

               l=*self*.list\_sommet[i]

               iter+=1

      else:

         x,y = *event*.x,*event*.y

         for i in range(len(*self*.list\_sommet)):

            sx=*self*.list\_sommet[i][1]

            sy=*self*.list\_sommet[i][2]

            if (x > sx-10) and (x < sx+10) and (y > sy-10) and (y < sy+10):

               x,y=*self*.list\_sommet[i][1], *self*.list\_sommet[i][2]

               if x==oldx and y==oldy:

*self*.canvas.create\_oval(x, y, x-30, y-30)

*self*.canvas.create\_oval(x-10, y-10, x+10, y+10, *fill*='red')

*self*.canvas.create\_text(x,y, *text*=*self*.list\_sommet[i][0], *fill*='white')

                  center\_x, center\_y = (x-30, y-30)

               else:

*self*.canvas.create\_line(oldx, oldy, x, y)

*self*.canvas.create\_oval(oldx-10, oldy-10, oldx+10, oldy+10, *fill*='red')

*self*.canvas.create\_text(oldx, oldy, *text*=l[0], *fill*='white')

*self*.canvas.create\_oval(x-10, y-10, x+10, y+10, *fill*='red')

*self*.canvas.create\_text(x,y, *text*=*self*.list\_sommet[i][0], *fill*='white')

                  center\_x, center\_y = ((oldx) + (x)) / 2, ((oldy + 15) + (y + 15)) / 2

*self*.indice\_arete+=1

*self*.canvas.create\_text(center\_x, center\_y, *text*="e"+str(*self*.indice\_arete))

               list=["e"+str(*self*.indice\_arete),l, *self*.list\_sommet[i] ]

*self*.list\_arete.append(list)

               iter=0

               oldx=0

               oldy=0

               l=[]

   def arete(*self*):

*self*.canvas.bind("<Button-1>", *self*.tracer)

   def degree(*self*):

      voisin=*self*.voisin()

      for i in range(len(*self*.list\_sommet)):

         degre=0

         for j in voisin:

            if *self*.list\_sommet[i][0] in j:

               degre+=1

*self*.list\_sommet[i][3]=degre

   def mat\_couleur(*self*):

      C = *self*.dic\_color

      popup = Toplevel(*self*.fram)

      popup.title("Matrice de couleur")

      popup.geometry("350x250+800+250")

      popup.transient(*self*.fram)

      mat = ttk.Treeview(popup)

      mat['columns'] = ("numero de couleur", "nom de couleur")

      mat.column("#0", *width*=15, *minwidth*=10)

      mat.column("numero de couleur", *width*=120)

      mat.column("nom de couleur", *width*=200)

      mat.heading("#0", *text*="")

      mat.heading("numero de couleur", *text*="numero de couleur")

      mat.heading("nom de couleur", *text*="nom de couleur")

      ind = 0

      for i in range(1,10):

         mat.insert(*parent*='', *index*='end', *iid*=i, *text*="", *values*=(i, C[i]))

         ind += 1

      mat.pack(*pady*=20)

   def attrubie\_couleur(*self*):

      for i in range(len(*self*.list\_sommet)):

*self*.list\_sommet[i][4] = 0

      dic\_voisin={}

      B = *self*.list\_arete

      voisin = []

*self*.degree()

      for i in B:

         l =[i[1], i[2]]

         voisin.append(l)

      for i in range(len(*self*.list\_sommet)):

         l=[]

         for j in voisin:

            if *self*.list\_sommet[i][0]==j[0][0]:

               l.append(j[1][0])

            elif *self*.list\_sommet[i][0]==j[1][0]:

               l.append(j[0][0])

         dic\_voisin[*self*.list\_sommet[i][0]]=l

      dic\_voisin=dict(sorted(dic\_voisin.items(),*reverse*=True))

      dic\_ordre={}

      i=10

      while i>=0:

         for k,v in dic\_voisin.items():

            if i==len(v):

               dic\_ordre[k]=v

         i-=1

      for key,lst in dic\_ordre.items():

         ls=[]

         for j in lst:

            for e in *self*.list\_sommet:

               if e[0]==j and key!=e[0]:

                  ls.append(e[4])

         k=1

         while k in ls:

            k+=1

         for i in range(len(*self*.list\_sommet)):

            if *self*.list\_sommet[i][0]==key:

*self*.list\_sommet[i][4]=k

   def coloration(*self*):

*self*.attrubie\_couleur()

      fram=*self*.fram

*self*.color = Toplevel(fram)

*self*.color.transient(*self*.fram)

*self*.color.geometry("550x300+300+350")

*self*.color.title("Coloration welsh and powell")

*self*.color\_can= Canvas(*self*.color, *background*="white")

*self*.color\_can.pack(*side*=TOP, *fill*=BOTH, *expand*=1)

      for j in *self*.list\_arete:

         if j[1][1]==j[2][1] and j[1][2]==j[2][2]:

*self*.color\_can.create\_oval(j[1][1], j[1][2],j[1][1]-30, j[1][2]-30)

            center\_x, center\_y = (j[1][1]-30, j[1][2]-30)

         else:

*self*.color\_can.create\_line(j[1][1],j[1][2], j[2][1], j[2][2])

            center\_x, center\_y = ((j[1][1]) + (j[2][1])) / 2, ((j[1][2] + 15) + (j[2][2] + 15)) / 2

*self*.color\_can.create\_text(center\_x, center\_y, *text*=j[0])

      for i in *self*.list\_sommet:

            if *self*.dic\_color[i[4]]=='yellow':

*self*.color\_can.create\_oval(i[1]-10, i[2]-10, i[1]+10, i[2]+10, *fill*=*self*.dic\_color[i[4]])

*self*.color\_can.create\_text(i[1],i[2], *text*=i[0], *fill*="black")

            else:

*self*.color\_can.create\_oval(i[1]-10, i[2]-10, i[1]+10, i[2]+10, *fill*=*self*.dic\_color[i[4]])

*self*.color\_can.create\_text(i[1],i[2], *text*=i[0], *fill*="white")

   def add\_menu(*self*):

      menu = Menu(*self*.window)

      fichier = Menu (menu, *tearoff*=0)

      fichier.add\_command(*label*="Nouveau", *command*=*self*.nouveau)

      fichier.add\_command(*label*="Ouvrir...", *command*=*self*.ouvrir)

      fichier.add\_command(*label*="Enregistrer", *command*=*self*.enregistrer)

      fichier.add\_command(*label*="Enregistrer sous...", *command*=*self*.enregistrer\_sous)

      fichier.add\_separator()                                        *#ligne de separation*

      fichier.add\_command(*label*="Fermer", *command*=*self*.fermer)

      fichier.add\_command(*label*="quitter",*command*=quit)

      creation = Menu (menu, *tearoff*=0)

      creation.add\_command(*label*="Sommet", *command*=*self*.sommet)

      creation.add\_command(*label*="Arete",*command*=*self*.arete)

      affichage = Menu (menu, *tearoff*=0)

      affichage.add\_command(*label*="Graphe", *command*=*self*.vide)

      affichage.add\_command(*label*="Chaines", *command*=*self*.vide)

      matrice = Menu(affichage, *tearoff*=0)

      matrice.add\_command(*label*="Matrice d'incidence", *command*=*self*.matrice\_in)

      matrice.add\_command(*label*="Matrice d'adjacence", *command*=*self*.matrice\_ad)

      affichage.add\_cascade(*label*="Matrice", *menu*=matrice)

      execution = Menu (menu, *tearoff*=0)

      execution.add\_command(*label*="Plus court chemin", *command*=*self*.vide)

      execution.add\_command(*label*="coloration welsh and powell", *command*=*self*.coloration)

      execution.add\_command(*label*="matrice de couleur", *command*=*self*.mat\_couleur)

      degre= Menu(execution, *tearoff*=0)

      degre.add\_command(*label*="degre non oriente", *command*=*self*.degree\_non\_oriente)

      degre.add\_command(*label*="degre oriente", *command*=*self*.affichagerdegre)

      execution.add\_cascade(*label*="Degre", *menu*=degre)

      edition = Menu(menu, *tearoff*=0)

      edition.add\_command(*label*="Graphe")

      menu.add\_cascade(*label*="Fichier", *menu*=fichier)

      menu.add\_cascade(*label*="Création", *menu*=creation)

      menu.add\_cascade(*label*="Affichage", *menu*=affichage)

      menu.add\_cascade(*label*="Exécution", *menu*=execution)

      menu.add\_cascade(*label*="Edition", *menu*=edition)

*self*.window.config(*menu* = menu)

   def affiche(*self*):

*self*.window.mainloop()

ap = application()

ap.personalise()

ap.add\_menu()

ap.affiche()