progam_V1alpha.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# * program_V1alpha.py * #
# Application gammique évolutive
# Version 1 : Calculer les gammes
#
from tkinter import *


class Gammique(Tk):
    """ Ramification Gammique """

    def __init__(self):
        Tk.__init__(self)
        "Tableau de bord"
        self.title('Entité Gammique :')

        # Fenêtre écran_résultat
        self.can = Canvas(self, bg='white', height=500, width=750)
        self.can.pack(side=RIGHT)
        self.cad = Frame(self, width=200, height=600)
        self.cad.pack(side=LEFT)
        self.can.delete(ALL)

        # Tracé d'encadrement
        # Données de l'encadré : Axes(x,y)=365(x),220(y)
        self.can.create_line(10, 0, 10, 450, fill='black')
        self.can.create_line(740, 10, 0, 10, fill='blue')
        self.can.create_line(740, 450, 740, 10, fill='black')
        self.can.create_line(10, 450, 740, 450, fill='blue')
        # Bouton gamme_audio
        # winsound.Beep(frequency, duration)
        Button(self.cad, text='Radio_inactive', width=15, bg='light
yellow').pack()
        # Bouton choix chromatique
        Button(self.cad, text='Chrome_inactif', width=15, bg='light
yellow').pack()
        # Bouton tableaux instruments
        Button(self.cad, text='Tabla_inactif', width=15, bg='light
yellow').pack()
        # Bouton accords1357
        Button(self.cad, text='A1357_inactif', width=15, bg='light
yellow').pack()

        # Les notes cursives scalpha : Graduations gérées.
        self.sca = [
            Scale(
                self,
                length=250,
                orient=HORIZONTAL,
                label=label,
                troughcolor=color,
                sliderlength=20,
                showvalue=1,
                from_=f,
                to=t,
                tickinterval=1,
                command=command,
            ) for (label, color, f, t, command) in (
                ("C", "black", 0, 5, self.scanote1),
                ("D", "green", -1, 4, self.scanote2),
                ("E", "blue", -2, 3, self.scanote3),
                ("F", "grey", -2, 3, self.scanote4),
                ("G", "red", -3, 2, self.scanote5),
                ("A", "orange", -4, 1, self.scanote6),
                ("B", "yellow", -5, 0, self.scanote7),
                ("CDEFGAB", "ivory", -5, 5, self.scanote8),
            )
        ]
        for x in self.sca:
            x.pack()

        # Bouton gamme_naturelle
        Button(self, text='Zéro', width=25, command=self.zero).pack()
        # Bouton gamme_calculée
        self.btgama = Button(self, text='gamme', width=25,
command=self.gama)
        self.btgama.pack()

    # Définition des curseurs
    def scanote1(self, xc):
        do = int(xc)
        xsi = self.sca[6].get()
        xre = self.sca[1].get()
        # Initialise sca[7](from_)
        fromhu = -6 - do
        self.sca[7].configure(from_=fromhu)
        if do < xsi: self.sca[6].set(do)
        if do > xre + 1: self.sca[1].set(do - 1)

    def scanote2(self, xd):
        re = int(xd)
        xdo = self.sca[0].get()
        xmi = self.sca[2].get()
        if re < xdo - 1: self.sca[0].set(re + 1)
        if re > xmi + 1: self.sca[2].set(re - 1)

    def scanote3(self, xe):
        mi = int(xe)
        xre = self.sca[1].get()
        xfa = self.sca[3].get()
        if mi < xre - 1: self.sca[1].set(mi + 1)
        if mi > xfa: self.sca[3].set(mi)

    def scanote4(self, xf):
        fa = int(xf)
        xmi = self.sca[2].get()
        xsol = self.sca[4].get()
        if fa < xmi: self.sca[2].set(fa)
        if fa > xsol + 1: self.sca[4].set(fa - 1)

    def scanote5(self, xg):
        sol = int(xg)
        xfa = self.sca[3].get()
        xla = self.sca[5].get()
        if sol < xfa - 1: self.sca[3].set(sol + 1)
        if sol > xla + 1: self.sca[5].set(sol - 1)

    def scanote6(self, xa):
        la = int(xa)
        xsol = self.sca[4].get()
        xsi = self.sca[6].get()
        if la < xsol - 1: self.sca[4].set(la + 1)
        if la > xsi + 1: self.sca[6].set(la - 1)

    def scanote7(self, xb):
        si = int(xb)
        xla = self.sca[5].get()
        xdo = self.sca[0].get()
        # Initialise sca[7](to)
        tohu = 6 - si
        self.sca[7].configure(to=tohu)
        if si < xla - 1: self.sca[5].set(si + 1)
        if si > xdo: self.sca[0].set(si)

    def scanote8(self, xh):
        sch = int(xh)
        f_t = 0
        xsi = self.sca[6].get()
        tosi = t_si = self.sca[6].cget("to")
        if (xsi + sch > t_si): f_t = -1
        xdo = self.sca[0].get()
        fromdo = f_do = self.sca[0].cget("from")
        todo = t_do = self.sca[0].cget("to")
        if (xdo + sch < f_do) or (f_t == -1):
            fromdo = xdo + sch
            todo = t_do + sch
            f_t = -1
        xre = self.sca[1].get()
        fromre = f_re = self.sca[1].cget("from")
        tore = t_re = self.sca[1].cget("to")
        if f_t == -1:
            fromre = f_re + sch
            tore = t_re + sch
        xmi = self.sca[2].get()
        frommi = f_mi = self.sca[2].cget("from")
        tomi = t_mi = self.sca[2].cget("to")
        if f_t == -1:
            frommi = f_mi + sch
            tomi = t_mi + sch
        xfa = self.sca[3].get()
        fromfa = f_fa = self.sca[3].cget("from")
        tofa = t_fa = self.sca[3].cget("to")
        if f_t == -1:
            fromfa = f_fa + sch
            tofa = t_fa + sch
        xsol = self.sca[4].get()
        fromsol = f_sol = self.sca[4].cget("from")
        tosol = t_sol = self.sca[4].cget("to")
        if f_t == -1:
            fromsol = f_sol + sch
            tosol = t_sol + sch
        xla = self.sca[5].get()
        fromla = f_la = self.sca[5].cget("from")
        tola = t_la = self.sca[5].cget("to")
        if f_t == -1:
            fromla = f_la + sch
            tola = t_la + sch
        xsi = self.sca[6].get()
        fromsi = f_si = self.sca[6].cget("from")
        tosi = t_si = self.sca[6].cget("to")
        if (xsi + sch > t_si) or (f_t == -1):
            fromsi = f_si + sch
            tosi = t_si + sch
            f_t = -1
        self.sca[0].configure(from_=fromdo, to=todo)
        self.sca[0].set(xdo + sch)
        self.sca[1].configure(from_=fromre, to=tore)
        self.sca[1].set(xre + sch)
        self.sca[2].configure(from_=frommi, to=tomi)
        self.sca[2].set(xmi + sch)
        self.sca[3].configure(from_=fromfa, to=tofa)
        self.sca[3].set(xfa + sch)
        self.sca[4].configure(from_=fromsol, to=tosol)
        self.sca[4].set(xsol + sch)
        self.sca[5].configure(from_=fromla, to=tola)
        self.sca[5].set(xla + sch)
        self.sca[6].configure(from_=fromsi, to=tosi)
        self.sca[6].set(xsi + sch)
        self.btgama.invoke()

    def zero(self):
        self.can.delete(ALL)
        # Tracé d'encadrement
        # Données de l'encadré : Axes(x,y)=365(x),220(y)
        self.can.create_line(10, 0, 10, 450, fill='black')
        self.can.create_line(740, 10, 0, 10, fill='blue')
        self.can.create_line(740, 450, 740, 10, fill='black')
        self.can.create_line(10, 450, 740, 450, fill='blue')
        self.can.create_line(360, 450, 360, 10, fill='green')
        self.can.create_line(10, 220, 740, 220, fill='green')
        fnotes = [0, -1, -2, -2, -3, -4, -5]
        tnotes = [+5, +4, +3, +3, +2, +1, 0]
        self.sca[0].configure(from_=fnotes[0], to=tnotes[0])
        self.sca[0].set(0)
        self.can.create_oval(300 - 5, 220 - 5, 300 + 5, 220 + 5,
fill='black')
        self.sca[1].configure(from_=fnotes[1], to=tnotes[1])
        self.sca[1].set(0)
        self.can.create_oval(320 - 5, 220 - 5, 320 + 5, 220 + 5,
fill='green')
        self.sca[2].configure(from_=fnotes[2], to=tnotes[2])
        self.sca[2].set(0)
        self.can.create_oval(340 - 5, 220 - 5, 340 + 5, 220 + 5,
fill='blue')
        self.sca[3].configure(from_=fnotes[3], to=tnotes[3])
        self.sca[3].set(0)
        self.can.create_oval(350 - 5, 220 - 5, 350 + 5, 220 + 5,
fill='grey')
        self.sca[4].configure(from_=fnotes[4], to=tnotes[4])
        self.sca[4].set(0)
        self.can.create_oval(370 - 5, 220 - 5, 370 + 5, 220 + 5,
fill='red')
        self.sca[5].configure(from_=fnotes[5], to=tnotes[5])
        self.sca[5].set(0)
        self.can.create_oval(390 - 5, 220 - 5, 390 + 5, 220 + 5,
fill='orange')
        self.sca[6].configure(from_=fnotes[6], to=tnotes[6])
        self.sca[6].set(0)
        self.can.create_oval(410 - 5, 220 - 5, 410 + 5, 220 + 5,
fill='yellow')
        self.sca[7].set(0)
        self.btgama.invoke()

    def gama(self):
        self.can.delete(ALL)
        # Tracé d'encadrement
        # Données de l'encadré : Axes(x,y)=365(x),220(y)
        self.can.create_line(10, 0, 10, 450, fill='black')
        self.can.create_line(740, 10, 0, 10, fill='blue')
        self.can.create_line(740, 450, 740, 10, fill='black')
        self.can.create_line(10, 450, 740, 450, fill='blue')
        self.can.create_line(460, 450, 460, 110, fill='green')
        self.can.create_line(220, 220, 740, 220, fill='green')
        # De la table gammique aux tables diatoniques surnommées
        gammes = [[1, 1, 0, 1, 1, 1, 0], [0, 2, 0, 1, 1, 1, 0], [2, 0, 0,
1, 1, 1, 0],
                  [4, 0, 0, 0, 0, 1, 0], [1, 0, 1, 1, 1, 1, 0], [0, 1, 1,
1, 1, 1, 0],
                  [1, 0, 3, 0, 0, 1, 0], [1, 2, 1, 0, 0, 1, 0], [2, 2, 0,
0, 0, 1, 0],
                  [0, 0, 1, 2, 1, 1, 0], [1, 3, 0, 0, 0, 1, 0], [0, 0, 2,
1, 1, 1, 0],
                  [1, 2, 2, 0, 0, 0, 0], [0, 0, 4, 0, 0, 1, 0], [1, 4, 0,
0, 0, 0, 0],
                  [1, 0, 0, 2, 1, 1, 0], [0, 1, 0, 2, 1, 1, 0], [1, 1, 3,
0, 0, 0, 0],
                  [0, 0, 0, 3, 1, 1, 0], [1, 1, 0, 0, 2, 1, 0], [0, 2, 0,
0, 2, 1, 0],
                  [0, 2, 0, 2, 0, 1, 0], [2, 0, 0, 0, 2, 1, 0], [1, 0, 1,
0, 2, 1, 0],
                  [1, 0, 1, 2, 0, 1, 0], [1, 1, 1, 2, 0, 0, 0], [2, 0, 0,
3, 0, 0, 0],
                  [0, 0, 2, 0, 2, 1, 0], [1, 2, 0, 2, 0, 0, 0], [1, 0, 0,
3, 0, 1, 0],
                  [1, 0, 0, 1, 2, 1, 0], [1, 1, 0, 3, 0, 0, 0], [1, 1, 2,
1, 0, 0, 0],
                  [0, 1, 0, 0, 3, 1, 0], [0, 0, 1, 0, 3, 1, 0], [0, 0, 0,
1, 3, 1, 0],
                  [0, 0, 0, 2, 2, 1, 0], [1, 0, 0, 0, 3, 1, 0], [0, 0, 2,
2, 0, 1, 0],
                  [0, 0, 0, 0, 4, 1, 0], [0, 0, 2, 3, 0, 0, 0], [1, 0, 0,
4, 0, 0, 0],
                  [0, 0, 0, 5, 0, 0, 0], [1, 1, 0, 1, 0, 2, 0], [1, 1, 0,
1, 2, 0, 0],
                  [0, 2, 0, 1, 0, 2, 0], [0, 2, 0, 1, 2, 0, 0], [2, 0, 0,
1, 0, 2, 0],
                  [2, 0, 0, 1, 2, 0, 0], [1, 0, 1, 1, 0, 2, 0], [1, 0, 1,
1, 2, 0, 0],
                  [1, 1, 0, 0, 1, 2, 0], [1, 1, 0, 0, 3, 0, 0], [1, 1, 0,
2, 1, 0, 0],
                  [1, 1, 2, 0, 1, 0, 0], [0, 2, 0, 0, 0, 3, 0], [1, 0, 0,
2, 2, 0, 0],
                  [1, 0, 0, 1, 0, 3, 0], [1, 3, 0, 0, 1, 0, 0], [1, 0, 0,
0, 1, 3, 0],
                  [0, 0, 0, 3, 0, 2, 0], [0, 0, 2, 1, 2, 0, 0], [1, 0, 0,
0, 0, 4, 0],
                  [0, 0, 0, 3, 2, 0, 0], [1, 1, 0, 0, 0, 3, 0], [3, 0, 0,
0, 0, 2, 0]]
        gamnoms = ['0', '-2', '+2', '^2', '-3', '-23', '-34x', '+34',
'+23x', '-34',
                   'x3', '°3', '+34x', '°34x', '^3', '-4', '-24', '^4',
'°4', '-5',
                   '-25', '-25+', '+25-', '-35', '-35+', '+45x', '+25x',
'°35-', '+35x',
                   '-45+', '-45', 'x5', 'x45+', '-25°', '-35°', '-45°',
'°45-', '°5',
                   '°35+', '*5', '°35x', '-45x', '°45x', '-6', '+6', '-26',
'-26+', '+26-',
                   '+26', '-36', '-36+', '-56', '-56+', '+56', 'x46+', '-
26°', '-46+',
                   '-46°', 'x36+', '-56°', '°46-', '°36+', '*6', '°46+',
'°6', 'x26-']

        # Récupération des notes cursives
        ydo = self.sca[0].get()
        xcpos_ = 300 + 100
        ycpos_ = 220
        xc_ = xcpos_ + (ydo * 10)
        yc_ = ycpos_ - (ydo * 10)
        rc_ = 5
        self.can.create_oval(xc_ - rc_, yc_ - rc_, xc_ + rc_, yc_ + rc_,
fill='black')
        yre = self.sca[1].get()
        xcpos_ = 320 + 100
        ycpos_ = 220
        xd_ = xcpos_ + (yre * 10)
        yd_ = ycpos_ - (yre * 10)
        rd_ = 5
        self.can.create_oval(xd_ - rd_, yd_ - rd_, xd_ + rd_, yd_ + rd_,
fill='green')
        ymi = self.sca[2].get()
        xcpos_ = 340 + 100
        ycpos_ = 220
        xe_ = xcpos_ + (ymi * 10)
        ye_ = ycpos_ - (ymi * 10)
        re_ = 5
        self.can.create_oval(xe_ - re_, ye_ - re_, xe_ + re_, ye_ + re_,
fill='blue')
        yfa = self.sca[3].get()
        xcpos_ = 350 + 100
        ycpos_ = 220
        xf_ = xcpos_ + (yfa * 10)
        yf_ = ycpos_ - (yfa * 10)
        rf_ = 5
        self.can.create_oval(xf_ - rf_, yf_ - rf_, xf_ + rf_, yf_ + rf_,
fill='grey')
        ysol = self.sca[4].get()
        xcpos_ = 370 + 100
        ycpos_ = 220
        xg_ = xcpos_ + (ysol * 10)
        yg_ = ycpos_ - (ysol * 10)
        rg_ = 5
        self.can.create_oval(xg_ - rg_, yg_ - rg_, xg_ + rg_, yg_ + rg_,
fill='red')
        yla = self.sca[5].get()
        xcpos_ = 390 + 100
        ycpos_ = 220
        xa_ = xcpos_ + (yla * 10)
        ya_ = ycpos_ - (yla * 10)
        ra_ = 5
        self.can.create_oval(xa_ - ra_, ya_ - ra_, xa_ + ra_, ya_ + ra_,
fill='orange')
        ysi = self.sca[6].get()
        xcpos_ = 410 + 100
        ycpos_ = 220
        xb_ = xcpos_ + (ysi * 10)
        yb_ = ycpos_ - (ysi * 10)
        rb_ = 5
        self.can.create_oval(xb_ - rb_, yb_ - rb_, xb_ + rb_, yb_ + rb_,
fill='yellow')

        # Mesure de l'intervalle tempéré
        c1 = (yre + 1) - ydo
        d2 = (ymi + 1) - yre
        e3 = yfa - ymi
        f4 = (ysol + 1) - yfa
        g5 = (yla + 1) - ysol
        a6 = (ysi + 1) - yla
        b7 = i = cum_diat = ok = x = 0
        diata = [c1, d2, e3, f4, g5, a6, b7]
        while i < 6:
            cum_diat += diata[i]
            i += 1
        diata[i] = 5 - cum_diat

        # Recherche diatonique par l'itération
        cc1 = dd2 = ee3 = ff4 = gg5 = aa6 = bb7 = 0
        diata2 = [cc1, dd2, ee3, ff4, gg5, aa6, bb7]
        while x < 7:
            m = x
            y = 0
            while y < 7:
                diata2[y] = diata[m]
                y += 1
                m += 1
                if m > 6: m = 0
            myx = myx2 = 0
            for my in gammes:
                if diata2 == my:
                    degre = x
                    myx2 = myx
                    x = 7
                myx += 1
            x += 1
        # Ici : diata(original cursif).degré(tonique).my(gamme)
        # Définition diatonique
        # GMAJ= gammes[0]
        gmaj = [1, 1, 0, 1, 1, 1, 0]  # Forme majeure simplifiée
        # GNAT= Ordre cursif comme diata[]
        gnat = ['C', 'D', 'E', 'F', 'G', 'A', 'B']  # Forme alphabétique
        cnat = ['', '', '', '', '', '', '']
        # Niveaux d'altérations
        nordiese = ['', '+', 'x', '^', '+^', 'x^', '^^', '+^^', 'x^^',
'^^^', '+^^^', 'x^^^', '^^^^']
        subemol = ['', '****', '°***', '-***', '***', '°**', '-**', '**',
'°*', '-*', '*', '°', '-']
        # Configuration modale
        gdeg = ['I', 'II', 'III', 'IV', 'V', 'VI', 'VII']
        # Définition des notes cursives
        cursifs = [ydo, yre, ymi, yfa, ysol, yla, ysi]
        ynat = ymod = 0
        for ycurs in cursifs:
            if ycurs > 0:
                ymod = nordiese[ycurs]
```

```
    if ycurs < 0:
        ymod = subemol[ycurs]
    if ycurs == 0:
        ymod = subemol[ycurs]
    cnat[ynat] = ymod
    ynat += 1

# Une tournée produit une tonalité modale de 7 notes
nat2 = degre
deg = nom = 0
ynote = xgdeg = 30
ytone = 50
while deg < 7:
    nat = deg  # Degré tonal en question
    cri = gimj = gmod = maj = 0
    xdeg = 60
    text0 = gdeg[deg]
    self.can.create_text(xgdeg, ynote + 10, text=text0,
                         font='bold', fill='black')
    while maj < 7:  # Tonalité modale du degré
        gmj = gmaj[maj]  # Forme majeure (1101110)
        imaj = diata2[nat]  # Forme modale (DIATA[DEGRE])
        ynt = cnat[nat2]  # Forme altérative des notes
        gnt = gnat[nat2]  # Forme tonale (CDEFGAB)
        ideg = gdeg[deg]
        cri = cri + gimj  # Tonalité cumulée
        gimj = imaj - gmj  # Calcul tonal PAS/PAS
        cmod = gmod = cri
        if gmod > 0:  # Forme altérative des tonalités
            imod = nordiese[cmod]
        if gmod < 0:
            imod = subemol[cmod]
        if gmod == 0:
            imod = subemol[cmod]
        gmod = gmod + cri  # Transition tonale
        # Construction du nom de la gamme
        if nom == 0:
            ynom = ynt
            gnom = gnt
            tnom = ynom, gnom, gamnoms[myx2]
            self.can.create_text(xdeg + 250, ynote, text=tnom,
                                 font='bold', fill='black')
        nat += 1
        nat2 += 1
        if nat > 6:
            nat = 0
        if nat2 > 6:
            nat2 = 0
        maj = maj + 1
        text1 = [ynt, gnt]
        text2 = [imod, maj]
        self.can.create_text(xdeg, ynote, text=text1)
        self.can.create_text(xdeg, ytone, text=text2, fill='blue')
        xdeg += 30
        nom = 1
    ynote += 60
    ytone += 60
    nat2 += 1
    if nat2 > 6:
        nat2 = 0
    deg = deg + 1
```

```
Gammique().mainloop()
```