```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from tkinter import *
from tkinter import ttk
from tkinter.ttk import *
from tkinter.messagebox import *
from tkinter.filedialog import *
from tkinter.colorchooser import *
from confr import *
from progress import *
from tooltip import *

class Configurator:
    colors = {lg('black'): 'black', lg('white'): 'white', lg('blue'): 'blue', lg('green'): '
green', lg('yellow'): 'yellow', lg('red'): 'red', lg('pink'): 'pink', lg('orange'): 'orange'
, lg('grey'): 'grey', }
    colors_name = [v for v, _ in colors.items()]
    font_lst = ['Courier', 'Calibri', 'Arial']
    #lgs = ['an', 'fr', 'al', 'es', 'it', 'ch']
    langs = {lg('anglais') : 'an',
             lg('francais') : 'fr',
             lg('allemand') : 'al',
             lg('espagnol') : 'es',
             lg('italien') : 'it',
             lg('chinois') : 'ch',}

    codages = ['UTF-8', 'UTF-16', 'UTF-4', 'ASCII']
    browsers = ['firefox']
    languages = ['Python', 'C++', 'C', 'Fortran', 'BASIC', 'Brain F', 'Cobol', 'Assembly']
    nom_bts = {'copy':      lg('copy'),
               'cut':       lg('cut'),
               'past':      lg('past'),
               'cstyle':    lg('cstyle'),
               'news':      lg('news'),
               'new':       lg('new'),
               'open':      lg('open'),
               'exit':      lg('exit'),
               'print':     lg('print'),
               'save':      lg('save'),
               'saveas':    lg('saveas'),
               'undo':      lg('undo'),
               'redo':      lg('redo'),
               'search':    lg('search'),
               'word':      lg('word'),
                'pdf':        lg('pdf'),
                'about':      lg('about'),
                'struct':     lg('struct'),
                'close':      lg('close'),
                'savecopyas':lg('savecopyas'),
                'replace':    lg('replace'),
                'gotol':      lg('gotol'),
                'tasks':      lg('tasks'),
                'puces':      lg('puces'),
                'research':   lg('research'),}

    def cancel(self):
        self.tk.destroy()
        self.dialoging = False

    def info(self, _):
        showinfo(self.title, lg('MWSNS'))

    def IHM(self):
        if self.dialoging:
            return

        self.dialoging = True
        self.tk = Toplevel(self.master)
        self.tk.iconbitmap(self.ico['config'])
        self.tk.transient(self.master)
        self.tk.title(lg('Configurator'))
        self.tk.resizable(width=False, height=False)
        self.tk.protocol('WM_DELETE_WINDOW', self.cancel)

        self.note = ttk.Notebook(self.tk)
        self.note.grid(row = 0, column = 0)

        self.root = Frame(self.note)
        self.note.add(text = lg('Settings'), child = self.root)

        ## Liste des cadres

        g = LabelFrame(self.root, text=lg('Global'))
        g.grid(row=0, column=0, sticky='w')

        m = LabelFrame(self.root, text=lg('Menu'))
        m.grid(row=1, column=0, sticky='e')
        s = LabelFrame(self.root, text=lg('Security'))
        s.grid(row=0, column=1, sticky='w')

        a = Frame(self.root)
        a.grid(row=1, column=1, sticky='w')

        d = Frame(a)
        d.grid(row=0, column=0)

        c = LabelFrame(d, text=lg('Communication'))
        c.grid(row=0, column=0, sticky='w')

        e = Frame(d)
        e.grid(row=0, column=1)

        l = LabelFrame(e, text=lg('perso'))
        l.grid(row=0, column=0)

        k = LabelFrame(e, text=lg('as'))
        k.grid(row=1, column=0)

        t = LabelFrame(a, text=lg('Text'))
        t.grid(row=1, column=0, sticky='w')

        v = LabelFrame(a, text=lg('View'))
        v.grid(row = 2, column = 0, sticky = 'w')

        ## Cadre g pour les variables globales

        self.mode_dark_ = IntVar(master = self.master)
        self.mode_dark = Checkbutton(g, text=lg('Dark_Mode'), variable=self.mode_dark_, onva
lue=1, offvalue=0)
        self.mode_dark.grid(row=0, column=0, sticky='w')
        if read('global', 'mode_dark') == '1':self.mode_dark_.set(1)
        self.line_number_ = IntVar(master = self.master)
        self.line_number = Checkbutton(g, text=lg('Line_Number'), variable=self.line_number_
, onvalue=1, offvalue=0)
        self.line_number.grid(row=1, column=0, sticky='w')
        if read('global', 'line_number') == '1':self.line_number_.set(1)
        self.enc_ = IntVar(master = self.master)
        self.enc = Checkbutton(g, text=lg('Encrypting'), variable=self.enc_, onvalue=1, offv
alue=0)
        self.enc.grid(row=2, column=0, sticky='w')
        if read('global', 'encrypt') == '1':self.enc_.set(1)
        self.puc_ = IntVar(master = self.master)
        self.puc = Checkbutton(g, text=lg('Puces'), variable=self.puc_, onvalue=1, offvalue=
0)
        self.puc.grid(row=3, column=0, sticky='w')
        if read('text', 'puces') == '1':self.puc_.set(1)
        self.update_ = IntVar(master = self.master)
        self.update = Checkbutton(g, text=lg('Update'), variable=self.update_, onvalue=1, of
fvalue=0)
        self.update.grid(row=4, column=0, sticky='w')
        if read('global', 'look_update') == '1':self.update_.set(1)
        self.notifs_ = IntVar(master = self.master)
        self.notifs = Checkbutton(g, text=lg('Notifs'), variable=self.notifs_, onvalue=1, of
fvalue=0)
        self.notifs.grid(row=5, column=0, sticky='w')
        if read('global', 'notifs') == '1':self.notifs_.set(1)

        minic = Frame(g)
        minic.grid(row = 6, column = 0, sticky = 'e')

        Label(minic, text = lg('Codage')).grid(row = 0, column = 0, sticky = 'w')
        self.coda = Combobox(minic, value=self.codages, width = 6)
        self.coda.grid(row = 0, column = 1, sticky = 'w')
        self.coda.current(self.get_codes_pos(read('crypt', 'code')))
        self.coda.bind('<<ComboboxSelected>>', self.info)
        self.coda.config(stat = 'disabled')

        Label(minic, text = lg('Langage')).grid(row=1, column = 0, sticky = 'w')
        self.lang = Combobox(minic, value = self.languages, width = 6)
        self.lang.grid(row = 1, column = 1, sticky = 'w')
        self.lang.current(self.get_lang_pos(read('global', 'lang')))

        ## Cadre m pour les menus

        self.menufile_ = IntVar(master = self.master)
        self.menufile = Checkbutton(m, text=lg('File'), variable=self.menufile_, onvalue=1,
offvalue=0)
        self.menufile.grid(row=0, column=0, sticky='w')
        if read('menu', 'file') == '1':self.menufile_.set(1)
        self.menuedit_ = IntVar(master = self.master)
        self.menuedit = Checkbutton(m, text=lg('Edit'), variable=self.menuedit_, onvalue=1,
offvalue=0)
        self.menuedit.grid(row=1, column=0, sticky='w')
        if read('menu', 'edit') == '1':self.menuedit_.set(1)
        self.menustyle_ = IntVar(master = self.master)
        self.menustyle = Checkbutton(m, text=lg('Style'), variable=self.menustyle_, onvalue=
1, offvalue=0)
        self.menustyle.grid(row=2, column=0, sticky='w')
        if read('menu', 'style') == '1':self.menustyle_.set(1)
        self.menufor_ = IntVar(master = self.master)
        self.menufor = Checkbutton(m, text=lg('Format'), variable=self.menufor_, onvalue=1,
offvalue=0)
        self.menufor.grid(row=3, column=0, sticky='w')
        if read('menu', 'format') == '1':self.menufor_.set(1)
        self.menurun_ = IntVar(master = self.master)
        self.menurun = Checkbutton(m, text=lg('Run'), variable=self.menurun_, onvalue=1, off
value=0)
        self.menurun.grid(row=4, column=0, sticky='w')
        if read('menu', 'run') == '1':self.menurun_.set(1)
        self.menucrypt_ = IntVar(master = self.master)
        self.menucrypt = Checkbutton(m, text=lg('Crypt'), variable=self.menucrypt_, onvalue=
1, offvalue=0)
        self.menucrypt.grid(row=5, column=0, sticky='w')
        if read('menu', 'crypt') == '1':self.menucrypt_.set(1)
        self.menuexp_ = IntVar(master = self.master)
        self.menuexp = Checkbutton(m, text=lg('Export'), variable=self.menuexp_, onvalue=1,
offvalue=0)
        self.menuexp.grid(row=6, column=0, sticky='w')
        if read('menu', 'export') == '1':self.menuexp_.set(1)
        self.menuarch_ = IntVar(master = self.master)
        self.menuarch = Checkbutton(m, text=lg('Archive'), variable=self.menuarch_, onvalue=
1, offvalue=0)
        self.menuarch.grid(row=7, column=0, sticky='w')
        if read('menu', 'arch') == '1':self.menuarch_.set(1)
        self.menumin_ = IntVar(master = self.master)
        self.menumin = Checkbutton(m, text=lg('Minitel'), variable=self.menumin_, onvalue=1,
 offvalue=0)
        self.menumin.grid(row=8, column=0, sticky='w')
        if read('menu', 'minitel') == '1':self.menumin_.set(1)
        self.menuupd_ = IntVar(master = self.master)
        self.menuupd = Checkbutton(m, text=lg('Update'), variable=self.menuupd_, onvalue=1,
offvalue=0)
        self.menuupd.grid(row=9, column=0, sticky='w')
        if read('menu', 'update') == '1':self.menuupd_.set(1)
        self.menuex_ = IntVar(master = self.master)
        self.menuex = Checkbutton(m, text=lg('Extension'), variable=self.menuex_, onvalue=1,
 offvalue=0)
        self.menuex.grid(row=10, column=0, sticky='w')
        if read('menu', 'extension') == '1':self.menuex_.set(1)
        self.menuopt_ = IntVar(master = self.master)
        self.menuopt = Checkbutton(m, text=lg('Options'), variable=self.menuopt_, onvalue=1,
 offvalue=0, stat = 'disabled')
        self.menuopt.grid(row=11, column=0, sticky='w')
        if read('menu', 'opt') == '1':self.menuopt_.set(1)
        self.menuhlp_ = IntVar(master = self.master)
        self.menuhlp = Checkbutton(m, text=lg('Help'), variable=self.menuhlp_, onvalue=1, of
fvalue=0)
        self.menuhlp.grid(row=12, column=0, sticky='w')
        if read('menu', 'help') == '1':self.menuhlp_.set(1)
        self.menuvie_ = IntVar(master = self.master)
        self.menuvie = Checkbutton(m, text=lg('View'), variable=self.menuvie_, onvalue=1, of
fvalue=0)
        self.menuvie.grid(row=13, column=0, sticky='w')
        if read('menu', 'view') == '1':self.menuvie_.set(1)

        ## Cadre s pour la sécurité

        self.conn_ = IntVar(master = self.master)
        self.conn = Checkbutton(s, text=lg('Connexion'), variable=self.conn_, onvalue=1, off
value=0)
        self.conn.grid(row=0, column=0, sticky='w')
        if read('global', 'conn') == '1':self.conn_.set(1)
        Label(s, text=lg('Username')).grid(row=1, column=0, sticky='e')
        Label(s, text=lg('Password')).grid(row=2, column=0, sticky='e')
        Label(s, text=lg('Key')).grid(row=3, column=0, sticky='e')
        self.usn_ = StringVar(master = self.master)
        self.usn = Entry(s, textvariable=self.usn_, width=20)
        self.usn.grid(row=1, column=1, sticky='w')
        self.usn.delete('0', END)
        self.usn.insert(END, read('security', 'username'))
        self.pwd_ = StringVar(master = self.master)
        self.pwd = Entry(s, textvariable=self.pwd_, show='*', width=20)
        self.pwd.grid(row=2, column=1, sticky='w')
        self.pwd.delete('0', END)
        self.pwd.insert(END, read('security', 'password'))
        self.key_ = StringVar(master = self.master)
        self.key = Entry(s, textvariable=self.key_, show='*', width=5)
        self.key.grid(row=3, column=1, sticky='w')
        self.key.delete('0', END)
        self.key.insert(END, read('crypt', 'key'))
        self.err_ = IntVar(master = self.master)
        self.err = Checkbutton(s, text=lg('Errors'), variable=self.err_, onvalue=1, offvalue
=0)
```

```
        self.err.grid(row=4, column=0, sticky='w')
        if read('global', 'errors') == '1':self.err_.set(1)
        self.ac_ = IntVar(master = self.master)
        self.ac = Checkbutton(s, text=lg('AskC'), variable=self.ac_, onvalue=1, offvalue=0)
        self.ac.grid(row=5, column=0, sticky='w')
        if read('global', 'askclose') == '1':self.ac_.set(1)

        ## Cadre c pour les communication minitel

        Label(c, text='Dev : ').grid(row=0, column=0, sticky='e')
        Label(c, text=lg('Speed')).grid(row=1, column=0, sticky='e')
        Label(c, text='Bytesize : ').grid(row=2, column=0, sticky='e')
        Label(c, text='Timeout : ').grid(row=3, column=0, sticky='e')

        Label(c, text='/dev/ttyACM0', relief = 'flat', bd = 2).grid(row=0, column=1, sticky=
'w')
        Label(c, text='4800').grid(row=1, column=1, sticky='w')
        Label(c, text='7').grid(row=2, column=1, sticky='w')
        Label(c, text='2').grid(row=3, column=1, sticky='w')
        self.min_al_ = IntVar(master = self.master)
        self.min_al = Checkbutton(c, text=lg('alertemin'), variable=self.min_al_, onvalue=1,
 offvalue=0)
        self.min_al.grid(row=4, column=1, sticky='w')
        if read('minitel', 'alerte') == '1':self.min_al_.set(1)

        ## Cadre t pour l'apparence du texte

        Label(t, text=lg('Light_Background_Color')).grid(row=0, column=0, sticky='e')
        Label(t, text=lg('Light_Foreground_Color')).grid(row=1, column=0, sticky='e')
        Label(t, text=lg('Dark_Background_Color')).grid(row=2, column=0, sticky='e')
        Label(t, text=lg('Dark_Foreground_Color')).grid(row=3, column=0, sticky='e')
        Label(t, text=lg('Font')).grid(row=4, column=0, sticky='e')
        Label(t, text=lg('FS')).grid(row=5, column=0, sticky='e')
        Label(t, text=lg('tab')).grid(row=6, column=0, sticky='e')

        self.bgl = Combobox(t, value=self.colors_name)
        self.bgl.grid(row=0, column=1, sticky='w')
        vt = self.get_color_pos(read('text', 'bgl'))
        if isinstance(vt, int):
            self.bgl.current(vt)
            val0 = self.colors_name[vt]
        else:
            self.colors_name.append(vt)
            self.bgl['value'] = self.colors_name
            self.bgl.current(END)
            val0 = vt
        Button(t, text=lg('...'), command = lambda : self.askcolor('lbc', val0)).grid(row=0,
column = 2, sticky = 'w')

        self.fgl = Combobox(t, value=self.colors_name)
        self.fgl.grid(row=1, column=1, sticky='w')
        vt = self.get_color_pos(read('text', 'fgl'))
        if isinstance(vt, int):
            self.fgl.current(vt)
            val1 = self.colors_name[vt]
        else:
            self.colors_name.append(vt)
            self.fgl['value'] = self.colors_name
            self.fgl.current(END)
            val1 = vt
        Button(t, text=lg('...'), command = lambda : self.askcolor('lfc', val1)).grid(row=1,
column = 2, sticky = 'w')

        self.bgd = Combobox(t, value=self.colors_name)
        self.bgd.grid(row=2, column=1, sticky='w')
        vt = self.get_color_pos(read('text', 'bgd'))
        if isinstance(vt, int):
            self.bgd.current(vt)
            val2 = self.colors_name[vt]
        else:
            self.colors_name.append(vt)
            self.bgd['value'] = self.colors_name
            self.bgd.current(END)
            val2 = vt
        Button(t, text=lg('...'), command = lambda : self.askcolor('dbc', val2)).grid(row=2,
column = 2, sticky = 'w')

        self.fgd = Combobox(t, value=self.colors_name)
        self.fgd.grid(row=3, column=1, sticky='w')
        vt = self.get_color_pos(read('text', 'fgd'))
        if isinstance(vt, int):
            self.fgd.current(vt)
            val3 = self.colors_name[vt]
        else:
            self.colors_name.append(vt)
            self.fgd['value'] = self.colors_name
            self.fgd.current(END)
            val3 = vt
        Button(t, text=lg('...'), command = lambda : self.askcolor('dfc', val3)).grid(row=3,
column = 2, sticky = 'w')

        self.font = Combobox(t, value=self.font_lst)
        self.font.grid(row=4, column=1, sticky='w')
        self.font.current(self.get_font_pos(read('text', 'font')))
        self.size = Combobox(t, value=[i for i in range(6, 73)])
        self.size.grid(row=5, column=1, sticky='w')
        self.size.current(int(read('text', 'size'))-6)

        self.tabs = Spinbox(t, value = int(read('text', 'tab')), from_ = 2, to = 16)
        self.tabs.grid(row = 6, column = 1, sticky = 'w')

        ## Cadre l pour la personnalisation

        Label(l, text = lg('langage')).grid(row = 0, column = 0)
        self.lg = Combobox(l, value=list(self.langs.keys()))
        self.lg.grid(row=0, column=1, sticky='w')
        self.lg.current(self.get_lg_pos(sel_lg()))
        self.lg.bind('<<ComboboxSelected>>', self.info)

        Label(l, text = lg('navig')).grid(row = 1, column = 0)
        self.bro = Combobox(l, value=self.browsers)
        self.bro.grid(row = 1, column = 1, sticky = 'w')
        ind = self.get_bro_pos(read('global', 'browser'))
        if isinstance(ind, int):
            self.bro.current(ind)
        else:
            self.bro.set(ind)

        self.bro.bind('<<ComboboxSelected>>', self.info)

        ## Cadre k pour l'enregistrement

        Label(k, text=lg('delay')).grid(row=0, column=0, sticky='e')
        Label(k, text=lg('path')).grid(row=1, column=0, sticky='e')
        self.spn = Combobox(k, value=[i for i in range(1, 60)])
        self.spn.grid(row=0, column=1, sticky='w')
        self.spn.bind('<<ComboboxSelected>>', self.info)
        self.spn.current(int((int(read('auto_save', 'delay')) / 60) - 1))
        ToolTip(self.spn, text = lg('time_autosave'))
        mic = Frame(k)
        mic.grid(row=1, column=1)
        self.path_ = StringVar(master = self.master)
        self.path_ihm = Entry(mic, textvariable=self.path_, width=18)
        self.path_ihm.grid(row=0, column=0, sticky='w')
        self.path_ihm.delete('0', END)
        self.path_ihm.insert(END, read('auto_save', 'path'))
        Button(mic, text='...', width=3, command=self.in_asp).grid(row=0, column=1)

        ## Cadre v pour l'affichage des barres

        self.vbt_ = IntVar(master = self.master)
        self.vbt = Checkbutton(v, text=lg('buttonbar'), variable=self.vbt_, onvalue=1, offva
lue=0)
        self.vbt.grid(row=0, column=0, sticky='w')
        if read('view', 'bar_buttons') == '1':self.vbt_.set(1)
        self.vinf_ = IntVar(master = self.master)
        self.vinf = Checkbutton(v, text=lg('infobar'), variable=self.vinf_, onvalue=1, offva
lue=0)
        self.vinf.grid(row=1, column=0, sticky='w')
        if read('view', 'bar_info') == '1':self.vinf_.set(1)

        ## Boutons en bas de la fenêtre

        ca = Frame(self.tk)
        ca.grid(row=1, column=0)

        Button(ca, text=lg('Cancel'), command=self.cancel,          width = 15).grid(row=0, c
olumn=0, sticky='w')
        Button(ca, text=lg('Apply'),  command=self.apply,           width = 23).grid(row=0, c
olumn=1)
        Button(ca, text=lg('OK'),    command=self.validate_choice, width = 23).grid(row=0, c
olumn=2)

        ###########################################
        ## ZONE n° 2 : Les raccourcis claviers ! ##
        ###########################################

        zak = Frame(self.note)
        self.note.add(text = lg('racc'), child = zak)

        self.tree = ttk.Treeview(zak, show = 'headings', columns = (1, 2, 3), height = 24)
        scroll = ttk.Scrollbar(zak, orient = 'vertical', command = self.tree.yview)
        self.tree.place(x = 0, y = 0)
        self.tree.config(yscrollcommand = scroll.set)
        self.tree.heading(1, text = lg('event'))
        self.tree.heading(2, text = lg('key_t'))
        self.tree.heading(3, text = lg('action'))
        self.tree.column(1, width = 150)
        self.tree.column(2, width = 150)
        self.tree.column(3, width = 180)
        scroll.place(x = self.tree.winfo_reqwidth(), y = 0, height = self.tree.winfo_reqheig
ht(), width = 20)
        self.tree.bind('<Double-Button-1>', self.change_linkkey)

        self.insert_keys()

        ##########################################
        ## ZONE n°3 : Le menu du clique droit ! ##
        ##########################################

        tk = Frame(self.note)
        self.note.add(text = lg('menuclkr'), child = tk)

        self.lst_bt = Listbox(tk, height = 25, font = ('Courier', 14), width = 42)
        ToolTip(tk, lg('PPKTA'))
        scroll2 = ttk.Scrollbar(tk, orient = 'vertical', command = self.lst_bt.yview)
        self.lst_bt.place(x = 0, y = 0)
        self.lst_bt.config(yscrollcommand = scroll2.set)
        scroll2.place(x = self.lst_bt.winfo_reqwidth(), y = 0, height = self.lst_bt.winfo_re
qheight(), width = 20)
        f = open(self.path_prog + '/menus.m', 'r')
        r = f.read()
        f.close()
        mod = False
        for line in r.split('\n'):
            if line == '':
                continue

            if line == '#clk':
                mod = True
                continue
            elif line[0] == '#':
                mod = False
                continue

            if mod:
                ln = line.split(',')
                if ln[4] == '1':
                    self.lst_bt.insert('end', lg('Separateur'))
                elif ln[2] == '1':
```

```
                self.lst_bt.insert('end', lg('Puces'))
            elif ln[3] == '1':
                self.lst_bt.insert('end', lg('search'))
            else:
                self.lst_bt.insert('end', self.nom_bts[ln[0]])

    def append11(evt):
        a = Toplevel()
        a.transient(tk)
        a.title(lg('configurator'))
        a.resizable(False, False)
        Label(a, text = lg('add')).place(x = 5, y = 5)
        lst = []
        for k, v in self.nom_bts.items():
            lst.append(v)
        c = ttk.Combobox(a, values = lst)
        c.place(x = 5, y = 35)
        def append12():
            pass

        b = Button(a, text = lg('add'), command = append12, stat = 'disabled')
        b.place(x = 5, y = 65)
        ToolTip(b, lg('notimp'))
        a.geometry('150x95')

    self.lst_bt.bind('+', append11)

    ##################################################
    ## ZONE n°4 : Le menu de la barre des boutons ! ##
    ##################################################

    tk2 = Frame(self.note)
    self.note.add(text = lg('menubts'), child = tk2)

    self.lst_bt2 = Listbox(tk2, height = 25, font = ('Courier', 14), width = 42)
    ToolTip(tk2, lg('PPKTA'))
    scroll3 = ttk.Scrollbar(tk2, orient = 'vertical', command = self.lst_bt2.yview)
    self.lst_bt2.place(x = 0, y = 0)
    self.lst_bt2.config(yscrollcommand = scroll3.set)
    scroll3.place(x = self.lst_bt2.winfo_reqwidth(), y = 0, height = self.lst_bt2.winfo_
reqheight(), width = 20)
    f = open(self.path_prog + '/menus.m', 'r')
    r = f.read()
    f.close()
```

```
        mod = False
        for line in r.split('\n'):
            if line == '':
                continue

            if line == '#bts':
                mod = True
                continue
            elif line[0] == '#':
                mod = False
                continue

            if mod:
                ln = line.split(',')
                if ln[2] == '1':
                    self.lst_bt2.insert('end', lg('Separateur'))
                else:
                    self.lst_bt2.insert('end', self.nom_bts[ln[1]])

        def append21(evt):
            a = Toplevel()
            a.transient(tk2)
            a.title(lg('configurator'))
            a.resizable(False, False)
            Label(a, text = lg('add')).place(x = 5, y = 5)
            lst = []
            for k, v in self.nom_bts.items():
                lst.append(v)
            c = ttk.Combobox(a, values = lst)
            c.place(x = 5, y = 35)
            def append():
                pass

            b = Button(a, text = lg('add'), command = append22, stat = 'disabled')
            b.place(x = 5, y = 65)
            ToolTip(b, lg('notimp'))
            a.geometry('150x95')

        self.lst_bt2.bind('+', append21)

    def change_linkkey(self, evt):
        self.selected = self.tree.item(self.tree.selection())['values']
        self.root = Toplevel(self.tk)
        self.root.transient(self.tk)
        self.root.title(lg('configurator'))
        Label(self.root, text = self.selected[2], font = ('Consolas', 12), wraplength = 175)
.place(x = 10, y = 10)
        Label(self.root, text = lg('newrac'), font = ('Consolas', 13, 'bold')).place(x = 10,
 y = 50)
        e = StringVar(master = self.master)
        self.e = Entry(self.root, textvariable = e, font = ('Consolas', 13, 'italic'), width
 = 17)
        self.e.place(x = 10, y = 90)
        self.e.insert('end', self.selected[1])
        self.list_keys = self.selected[1].split(' + ')
        self.fin_key = []
        shift = False
        for i in self.list_keys:
            if i == 'Ctrl':
                self.fin_key.append('<Control')
            elif i == 'Alt':
                self.fin_key.append('Alt')
            elif i == 'Shift':
                shift = True
            else:
                self.fin_key.append(i.lower() if not shift else i.upper())
        self.e.bind('<Key>', self.keypress_link)

        Button(self.root, command = self.valide_linkkey, text = lg('OK')).place(x = 10, y =
130)
        Button(self.root, command = self.root.destroy, text = lg('cancel')).place(x = 110, y
 = 130)
        Button(self.root, command = lambda : self.valide_linkkey(True), text = lg('retirer')
).place(x = 10, y = 160)
        self.root.geometry('200x200')

    def keypress_link(self, evt):
        if len(evt.keysym) == 1:
            if 96 < ord(evt.keysym) < 96 + 26 or 64 < ord(evt.keysym) < 64 + 26:
                self.list_keys.append(evt.keysym.upper())
                self.fin_key.append(evt.keysym + '>')
        elif evt.keysym in ('Control_L', 'Control_R'):
            self.list_keys = ['Ctrl']
            self.fin_key = ['<Control']
        elif evt.keysym in ('Shift_L', 'Shift_R') and not 'Shift' in self.list_keys:
            self.list_keys.append('Shift')
        elif evt.keysym in ('Alt_L', 'Alt_R') and not 'Alt' in self.list_keys:
            self.list_keys.append('Alt')
            self.fin_key.append('Alt')
        elif evt.keysym == 'ISO_Level3_Shift':
            self.list_keys = ['Ctrl']
            self.fin_key = ['<Control']
            self.list_keys.append('Alt')
            self.fin_key.append('Alt')
        elif evt.keysym[0] == 'F':
            self.list_keys = [evt.keysym]
            self.fin_key = ['<' + evt.keysym + '>']

        self.e.delete('0', 'end')
        self.e.insert('end', ' + '.join(self.list_keys))

    def valide_linkkey(self, delete = False):
        f = open(self.path_prog + '/keys.k', 'r', encoding = get_encode())
        r = f.read()
        f.close()

        res = ''
        for line in r.split('\n'):
            if not line:
                continue

            name, event = line.split(' = ')
            if name == self.selected[0]:
                if delete:
                    line = name + ' = '
                else:
                    line = name + ' = ' + '-'.join(self.fin_key)
            else:
                line = name + ' = ' + event
            res += line + '\n'

        f = open('keys.k', 'w', encoding = get_encode())
        f.write(res)
        f.close()
        self.root.destroy()
        self.insert_keys()

    def clear_tree(self):
        for x in self.tree.get_children():
            self.tree.delete(x)

    def insert_keys(self):
        self.clear_tree()
        try:
            self.__keyb__()
        except Exception:
            return

        f = open('keys.k', 'r', encoding = get_encode())
        r = f.read()
        f.close()
        for line in r.split('\n'):
            if line == '':
                continue

            name = line.split(' = ')[0]
            event = self.get_accelerator(name)
            self.tree.insert('', 'end', values = (name, event, ''))

    def askcolor(self, type, oldcolor = None):
        title = ''
        if type == 'lbc':
            title = lg('Light_Background_Color')
        elif type == 'lfc':
            title = lg('Light_Foreground_Color')
        elif type == 'dbc':
            title = lg('Dark_Background_Color')
        elif type == 'dfc':
            title = lg('Dark_Foreground_Color')

        color = askcolor(color = oldcolor, title = title)
        if color[0] != None:
            self.colors_name.append(color[1])
            if type == 'lbc':
                self.bgl['value'] = self.colors_name
                self.bgl.current(END)
            elif type == 'lfc':
                self.fgl['value'] = self.colors_name
                self.fgl.current(END)
            elif type == 'dbc':
                self.bgd['value'] = self.colors_name
                self.bgd.current(END)
            elif type == 'dfc':
                self.fgd['value'] = self.colors_name
                self.fgd.current(END)
        self.info(None)


    def in_asp(self):
        n = asksaveasfilename(title=lg('open') + ' ' + lg('bu'), initialdir='.', filetypes=[
(lg('bu'), '.bu')])
        if n:
            self.path_ihm.delete('0', END)
            self.path_ihm.insert(END, n)

    def get_color_pos(self, data):
        try:
            return self.colors.index(data)
        except:
            return data

    def get_font_pos(self, data):
        try:
            return self.font_lst.index(data)
        except:
            pass

    def get_bro_pos(self, data):
        try:
            return self.browsers.index(data)
        except:
            return data

    def get_codes_pos(self, data):
        try:
            return self.codages.index(data)
        except:
            pass

    def get_lang_pos(self, value):
        try:
            return self.languages.index(value)
        except:
            pass

    def get_lg_pos(self, data):
        try:
            return list(self.langs.values()).index(data)
        except:
            pass

    def validate_choice(self):
        p = Progress(self.root, title = lg('Configurator'), maximum = 40, decimals = 0, onco
lor = 'blue')
        self.tree.unbind('<Double-Button-1>')
        log = open(self.path_prog + '/log.txt', 'a')
        try:
            p.step('mode_dark')
            write('global', 'mode_dark', self.mode_dark_.get())
        except Exception as e:
            log.write(str(e) + '\n')

        try:
            p.step('line_number')
            write('global', 'line_number', self.line_number_.get())
        except Exception as e:
            log.write(str(e) + '\n')

        try:
            p.step('encrypt')
            write('global', 'encrypt', self.enc_.get())
        except Exception as e:
            log.write(str(e) + '\n')

        try:
            p.step('puces')
            write('text', 'puces', self.puc_.get())
        except Exception as e:
            log.write(str(e) + '\n')

        try:
            p.step('language')
            write('global', 'lang', self.lang.get())
        except Exception as e:
            log.write(str(e) + '\n')

        try:
            p.step('notifications')
            write('global', 'notifs', self.notifs_.get())
        except Exception as e:
            log.write(str(e) + '\n')
```

```
try:
    p.step('file')
    write('menu', 'file', self.menufile_.get())
except Exception as e:
    log.write(str(e) + '\n')

try:
    p.step('edit')
    write('menu', 'edit', self.menuedit_.get())
except Exception as e:
    log.write(str(e) + '\n')

try:
    p.step('format')
    write('menu', 'format', self.menufor_.get())
except Exception as e:
    log.write(str(e) + '\n')

try:
    p.step('run')
    write('menu', 'run', self.menurun_.get())
except Exception as e:
    log.write(str(e) + '\n')

try:
    p.step('crypt')
    write('menu', 'crypt', self.menucrypt_.get())
except Exception as e:
    log.write(str(e) + '\n')

try:
    p.step('export')
    write('menu', 'export', self.menuexp_.get())
except Exception as e:
    log.write(str(e) + '\n')

try:
    p.step('arch')
    write('menu', 'arch', self.menuarch_.get())
except Exception as e:
    log.write(str(e) + '\n')

try:
    p.step('minitel')
```

```
        write('menu', 'minitel', self.menumin_.get())
    except Exception as e:
        log.write(str(e) + '\n')

    try:
        p.step('opt')
        write('menu', 'opt', self.menuopt_.get())
    except Exception as e:
        log.write(str(e) + '\n')

    try:
        p.step('help')
        write('menu', 'help', self.menuhlp_.get())
    except Exception as e:
        log.write(str(e) + '\n')

    try:
        p.step('update')
        write('menu', 'update', self.menuupd_.get())
    except Exception as e:
        log.write(str(e) + '\n')

    try:
        p.step('extension')
        write('menu', 'extension', self.menuex_.get())
    except Exception as e:
        log.write(str(e) + '\n')

    try:
        p.step('style')
        write('menu', 'style', self.menustyle_.get())
    except Exception as e:
        log.write(str(e) + '\n')

    try:
        p.step('view')
        write('menu', 'view', self.menuvie_.get())
    except Exception as e:
        log.write(str(e) + '\n')


    try:
        p.step('conn')
        write('global', 'conn', self.conn_.get())
    except Exception as e:
        log.write(str(e) + '\n')

    try:
        p.step('username')
        write('security', 'username', self.usn_.get())
    except Exception as e:
        log.write(str(e) + '\n')

    try:
        p.step('password')
        write('security', 'password', self.pwd_.get())
    except Exception as e:
        log.write(str(e) + '\n')

    try:
        p.step('errors')
        write('global', 'errors', self.err_.get())
    except Exception as e:
        log.write(str(e) + '\n')

    try:
        p.step('askclose')
        write('global', 'askclose', self.ac_.get())
    except Exception as e:
        log.write(str(e) + '\n')

    try:
        p.step('browser')
        write('global', 'browser', self.bro.get())
    except Exception as e:
        log.write(str(e) + '\n')


    try:
        p.step('bgd')
        write('text', 'bgd', self.bgd.get())
    except Exception as e:
        log.write(str(e) + '\n')

    try:
        p.step('fgd')
        write('text', 'fgd', self.fgd.get())
    except Exception as e:
            log.write(str(e) + '\n')

        try:
            p.step('bgl')
            write('text', 'bgl', self.bgl.get())
        except Exception as e:
            log.write(str(e) + '\n')

        try:
            p.step('fgl')
            write('text', 'fgl', self.fgl.get())
        except Exception as e:
            log.write(str(e) + '\n')

        try:
            p.step('font')
            write('text', 'font', self.font.get())
        except Exception as e:
            log.write(str(e) + '\n')

        try:
            p.step('tabs')
            write('text', 'tab', str(self.tabs.get()))
        except Exception as e:
            log.write(str(e) + '\n')


        try:
            p.step('check updates')
            write('global', 'look_update', str(self.update_.get()))
        except Exception as e:
            log.write(str(e) + '\n')

        try:
            p.step('Info bar')
            write('view', 'bar_info', self.vinf_.get())
        except Exception as e:
            log.write(str(e) + '\n')

        try:
            p.step('Button bar')
            write('view', 'bar_buttons', self.vbt_.get())
        except Exception as e:
            log.write(str(e) + '\n')
```

```
try:
    p.step('size', 'CANCELED')
    write('text', 'size', self.size.get())
except Exception as e:
    log.write(str(e) + '\n')

try:
    p.step('codage', 'CANCELED')
    #write('crypt', 'code', self.coda.get())

except Exception as e:
    log.write(str(e) + '\n')

try:
    ## Chemin minitel ! (minitel / [dev, speed = 4800, bytesize = 7, timeout = 2]
    p.step('minitel\'s alertes')
    write('minitel', 'alerte', self.min_al_.get())
except Exception as e:
    log.write(str(e) + '\n')


try:
    p.step('delay')
    write('auto_save', 'delay', str(int(self.spn.get())*60))
except Exception as e:
    log.write(str(e) + '\n')

try:
    p.step('path')
    write('auto_save', 'path', self.path_.get())
except Exception as e:
    log.write(str(e) + '\n')

log.close()

p.step('Saving choosed langage')
set_n_lg(self.langs[self.lg.get()])

print('Restarting...')
self.cancel()
self.master.destroy()
self.__start__()
```

```
    def apply(self):
        self.configurating = True
        self.validate_choice()


if __name__ == '__main__':
    from __init__ import *
```