```
#!/usr/bin/env python2.7
# encoding: utf-8
#*************************************************************
#*
#* MODULE:
#*
#* AUTHOR(S): yann Pottier
#*
#* PURPOSE: en entrant les coordonnées de la digue, de la source de l'inondation, et la fourchette d'altitudes de l'inondation
#*          on obtient la visualisation de la digue, avec sa longueur et son orientation, ainsi que l'etendu d'eau retenue
#*
#*************************************************************

#%module
#% description: cree une digue et simule l inondation
#% keywords: raster
#%end
#%option
#% key: input
#% type: string
#% gisprompt: oldraster,raster
#% description: nom du raster MNT dans grass
#% required : yes
#%end
#%option
#% key: coordigue
#% type: string
#% description: coordonnees de depart de la digue
#% required : yes
#%end
#%option
#% key: reg
#% type: string
#% gisprompt: oldvector,vector
#% description: nom du vecteur servant a delimiter la region
#% required : no
#%end
#%option
#% key: fond
#% type: string
#% gisprompt: oldraster,raster
#% description: nom du raster qui servira d image de fond
#% answer: MNT
#% required : no
#%end
#%option
#% key: reso
#% type: string
#% description: resolution d'affichage
#% answer: originale
#% required : no
#%end
#%option
#%key:couleur
#%type: string
#%key_desc: style
#%options:
aspect,aspectcolr,bcyr,bgyr,byg,byr,celsius,curvature,differences,elevation,etopo2,evi,grey,grey1.0,grey255,gyr,ndvi,population,precipitation,rainbow,ramp,ryb,r
yg,sepia,slope,srtm,terrain,wave,random
#%description: choisir couleur de la carte de fond
#%answer:elevation
#%required:no
#%end
#%option
#% key:niveaudeaumin
#% type: string
#% description: altmin surface de l'inondation
#% required : yes
#%end
#%option
#% key: niveaudeaumax
#% type: string
#% description: altmax surface de l'inondation
#% required : yes
#%end
#%option
#% key: pasduniveaudeau
#% type: string
#% description: saut d altitude
#% required : yes
#%end
#%option
#% key: ratio
#% type: string
#% description: taille d affichage
#% answer: 100
#% required : no
#%end
#%option
#% key: coordremp
#% type: string
#% description: coordonnees de remplissage
#% required : yes
#%end
#%option
#% key: outputlac
#% type: string
#% description: nom des inondations crees
#% answer:lac
#% required : no
```

```
#%end
#%option
#% key: outputrast
#% type: string
#% description: nom rasters mnt avec digue incrustee
#% answer:blabla
#% required : no
#%end
#%option
#% key: postnom
#% type: string
#% description: nom ajoute au debut du nom de l image quand on veux l enregister
#% answer:A
#% required : no
#%end
#%option
#% key: TailleMaxIle
#% type: string
#% description: nbr pixels d alt sup que la digue peut sauter
#% answer:0
#% required : no
#%end
#%flag
#% key: o
#% description: Override projection check
#%end


from Tkinter import *
import Image, ImageTk
from math import *
import wx
import tkFileDialog
import os, os.path
try:
    from osgeo import gdal
    from osgeo import osr
except ImportError:
    import gdal
from os import chdir

import grass.script as grass

#objets Globaux
ID_PLUS = 100#objets servant pour les evenements des boutons
ID_MOINS = 101
ID_MONTER = 102
ID_BASE = 103
ID_BAISSER = 104
ID_AIDE=105
ID_MANUEL=106
ID_PRINT=107

im="" #objet servant à contenir les images a afficher
altlacmax="" #stockera input altitude max
altlacmin=""#stockera input altitude min
altlacpas=""#stockera input saut d'altitude entre les images
outlac=''
outlac1=''
entree="" #stockera le fichier MNT de grass
imbase=""#stockera le fichier MNT sous forme png pour l'affichage
outmap=""
outdiguepng=""
nomlac=""
mask=""#région de travail et d'affichage, vecteur en input
res=""#resolution d'affiche et de travail quand le programme ne necessite pas la resolution d'origine
dp=""#longueur de la digue en pixels
dpg=""#longueur de la digue en mères
Tx=""#taille des pixels en x
Ty=""#taille des pixels en y
nu=float()##stock l'altitude de travail, sert a ecrire le nom de l image a afficher
#car j'ai utilise "nomdelacarte" + nu+.png pour pourvoir apeller les cartes a afficher.
#nu = altitude min du lac + multipe du saut d'altitude entre les images = altlacmin+ x*altlacpas
listrastinG=[] #ici seront listées les rasters créées dans GRASS, sert principalement pour pouvoir les effacers quand on ferme l'applicaton
listpng=[] #ici seront listées les rasters crées en png dans le dossier où se trouve le fichier du script , sert principalement pour pouvoir les effacers
quand on ferme l'application.
listLacinG=[]#liste specifique pour les inondations dans grass, sert quand on veut les sauvegarder
dtd={} #créé un dictionnaire, je l'utilise pour retrouver (lors de l'affichage) les longueurs calculées des digues en fonction de l altitude de l inondation.
fen=""#fenetre principal
fen1=""#fenetre de fermeture (sauver effacer images et cartes)
filename=""#serra le MNT exporté hors grass (tiff) pour pouvoir utiliser Gdal (valeur des pixels etc.)pour le calcule de la digue (cela ne fonctionnait pas
dans grass donc je l'exporte)
cartedigue=""
tailledigue=""
rep=""
fic=""
fic2=None
lstphoto=[]



class main(wx.Frame):
    def __init__(self, titre):
        global fen,outdiguepng,dp,dpg, TRX,TRY,im,res,mask,nomlac,im,altlacpas,imbase,outmap,listrastinG,listpng,listLacinG,entree,filename, outmap, outlac,
        outlac1,nu, altlacmax,altlacmin,cartedigue,dtd

        ############ gestion des inputs#############################
        #pour les commentaires voir le manuel
        entree = options['input']#raster elevation deja dans grass
        mask = options['reg'] #vecteur delimitant la region de travail
```

```
        #je ne devrait pas l'apeller "mask" pour eviter les confusion car c'est pour definir la region
        #mais c'est dans le programme, l'utilisateur le verra sous le nom "reg" et " nom du vecteur servant a delimiter la region"
        altlacmin = options['niveaudeaumin']
        altlacmax = options['niveaudeaumax']
        altlacpas = options['pasduniveaudeau']
        fond = options['fond']
        rat = options['ratio']
        res = options['reso']
        rat=int(rat)
        TMI= options['TailleMaxIle']
        TMI=int(TMI)

        gisbase = os.getenv('GISBASE')
        os.chdir(os.path.join(gisbase, 'etc', 'gui','scripts'))#selection le dossier de travail, le programme a toujour bien fonctionné sans cette ligne
        #car ce dois être le dossier de travail predefini mais je prefere le confirmer car peut être sur d'autre ordi ou d'autre version de grass ce ne sera
        pas le cas

        # verification que la "couleur" choisie est disponible
        couleur  = options['couleur']
        if couleur:
                couleur_opts = os.listdir(os.path.join(gisbase, 'etc', 'colors'))#liste les couleurs disponisbles
                if couleur not in couleur_opts:
                    grass.fatal(_("n'est pas dans la table des couleurs") % couleur +
                                _("les couleur disponibles sont: %s") % ' '.join(couleur_opts))


        coordig = options['coordigue'] #coordonnées du point de depart pour le calcule de la digue

        #ici je separe la latitude de la longitude, pour la digue
        n=len(coordig)
        i=0
        a=0
        lat=""
        lg=""
        for i in range(n):
            if a==0 and coordig[i]!=",":
                lat= lat+coordig[i]
            elif coordig[i]==",":
                i=i+1
                a=1
            else:
                lg=lg+coordig[i]
            i=i+1


        no = float(altlacmin)
        filename = options['outputrast'] #nom donnée a l'export (hors grass) de l'input "entree"
                                    # je n'ai pas reussi a faire fonctionner les fonction Gdal avec le
                                    # fichier raster deja importe dans grass donc je l'export pour travailler avec Gdal

        coordr = options['coordremp'] #coordonnées source du remplissage dans r.lake
        nomlac = options['outputlac'] #base du nom donné aux raster d'inondation créés
        drap = flags['o']
        nu=float(altlacmin)
        #############fin de gestion des input#######################################

        wx.Frame.__init__(self, None, 1, title = titre, size = (800, 700))#création du cadre de la fenetre d'affichage


        ####ici je pose les bases pour les trois images à afficher et leurs positionnements
        self.imlORIG = None
        self.imlORIX = 0
        self.imlORIY = 0
        self.bmpLAC = None

        self.imbORIG = None
        self.imbORIX = 0
        self.imbORIY = 0
        self.bmpBASE = None

        self.imdORIG = None
        self.imdORIX = 0
        self.imdORIY = 0
        self.bmdDIG = None

        ###
        self.ratio = rat #création de l'objet servant à definir de la taille d'affichage de l'image
        self.inc = 5 #self.inc sera additionné ou soustrait à "self.ratio" pour modifier la taille d'affichage de l'image (zoom in,out)


        #############creation de la barre de menu#######################
        menuFichier = wx.Menu(style = wx.MENU_TEAROFF)
        menuFichier.Append(wx.ID_EXIT, "&Quitter\tCTRL+q", "Quitter l'application")
        menuFichier.Append(wx.ID_PRINT, "&Imprimer\tCTRL+i", "Imprimer vue")

        menuAfficher = wx.Menu(style = wx.MENU_TEAROFF)
        menuAfficher.Append(wx.ID_UNDO,"&Taille d'origine\tCTRL+t", "taille d origine")
        menuAfficher.Append(ID_PLUS, "&Agrandir\tCTRL+i", "Agrandir l'image")
        menuAfficher.Append(ID_MOINS, "&Diminuer\tCTRL+o", "Diminuer l'image")
        menuAfficher.Append(ID_MONTER, "Monter&\tCTRL+u", "monter le niveau d'eau")
        menuAfficher.Append(ID_BAISSER, "Baisser&\tCTRL+d", "baisser le niveau d'eau")
        menuAfficher.Append(ID_BASE, "Basse&\tCTRL+c", "afficher le MNT seul")

        menuAide=wx.Menu(style = wx.MENU_TEAROFF)
        menuAide.Append(ID_AIDE, "&aide\tCTRL+a", "aide")
        menuAide.Append(ID_MANUEL,"&manuel\tCTRL+m","manuel")

        menuBarre = wx.MenuBar()
        menuBarre.Append(menuFichier, "&Fichier")
        menuBarre.Append(menuAfficher, "&Afficher")
```

```
        menuBarre.Append(menuAide, "&Aide")
        self.SetMenuBar(menuBarre)

        self.barre = wx.StatusBar(self, -1)
        self.SetStatusBar(self.barre)
        #########fin de la creation de la barre de menu#################

        #########creation de la barre d outil######################
        #les images (.png) utilisés pour les boutons doivent être dans le même dossier que le fichier du programme (.py)
        outils = wx.ToolBar(self, 1,style = wx.TB_HORIZONTAL | wx.NO_BORDER)
        outils.AddSimpleTool(wx.ID_UNDO,
        wx.Bitmap("Origine.png", wx.BITMAP_TYPE_PNG),shortHelpString = "Taille originale",longHelpString = "Taille originale")
        outils.AddSimpleTool(ID_PLUS, wx.Bitmap("zoomplus.png", wx.BITMAP_TYPE_PNG),
                             shortHelpString = "Agrandir",longHelpString = "Agrandir l'image")
        outils.AddSimpleTool(ID_MOINS, wx.Bitmap("zoommoins.png", wx.BITMAP_TYPE_PNG),
                             shortHelpString = "Diminuer",longHelpString = "Diminuer l'image")
        outils.AddSeparator()
        outils.AddSimpleTool(wx.ID_EXIT,wx.Bitmap("Exit.png", wx.BITMAP_TYPE_PNG),
                             shortHelpString = "Quitter",longHelpString = "Quitter l'application")
        outils.AddSimpleTool(ID_MONTER, wx.Bitmap("monter.png", wx.BITMAP_TYPE_PNG),
                             shortHelpString = "monter",longHelpString = "monter niveau d eau")
        outils.AddSimpleTool(ID_BAISSER, wx.Bitmap("baisser.png", wx.BITMAP_TYPE_PNG),
                             shortHelpString = "baisser",longHelpString = "baisser niveau d eau")
        outils.AddSimpleTool(ID_BASE, wx.Bitmap("cartebase.png", wx.BITMAP_TYPE_PNG),
                             shortHelpString = "cartebase",longHelpString = "afficher carte base")
        outils.AddSimpleTool(ID_PRINT, wx.Bitmap("print.png", wx.BITMAP_TYPE_PNG),
                             shortHelpString = "imprimer",longHelpString = "imprimer la vue")
        outils.Realize()
        self.SetToolBar(outils)
        ###############fin de creation des outils#################################

        ###creation de la fenetre d'affichage
        sizer = wx.BoxSizer()

        self.panneau = wx.ScrolledWindow(self)
        self.panneau.EnableScrolling(True, True)
        self.panneau.SetScrollRate(20,20)

        self.panneau.SetScrollbars(2, 2,self.GetVirtualSize()[0], self.GetVirtualSize()[1], noRefresh=False)

        sizer.Add(self.panneau, 1, wx.EXPAND|wx.ALL, 0)

        self.SetSizer(sizer)
        ###fin creation de la fenetre d'affichage

        ###gestion des evenements   menu et barre d'outil
        wx.EVT_MENU(self, wx.ID_EXIT, self.OnExit)
        wx.EVT_MENU(self, wx.ID_UNDO, self.Retour)
        wx.EVT_MENU(self, ID_PLUS, self.Plus)
        wx.EVT_MENU(self, ID_MOINS, self.Moins)
        wx.EVT_MENU(self, ID_MONTER, self.monter)
        wx.EVT_MENU(self, ID_BAISSER, self.baisser)
        wx.EVT_MENU(self, ID_BASE, self.base)
        wx.EVT_MENU(self, ID_AIDE, self.aide)
        wx.EVT_MENU(self, ID_MANUEL, self.manuel)
        wx.EVT_MENU(self, ID_PRINT, self.imprimer)



        ################################### DEBUT DU TRAVAIL AVEC GDAL ##########################################################################

        ###construction de la digue##

        grass.run_command('g.region',rast=entree)#(re)defini la region en fonction du raster sur lequel on travail
                                            # securite au cas où ce n'est pas le cas pour divers raisons
        ##extraire le fichier de grass (car cela ne fonctionnait pas dans grass)
        grass.run_command('r.out.gdal', input=entree, output=filename, nodata=0)
        couche = gdal.Open(filename)
        #projection=couche.GetProjection()
        gt =couche.GetGeoTransform()#
        bandes = couche.RasterCount
        TRX=couche.RasterXSize #donne la taille du raster en pixel, utile pour eviter que la digue qui sorte de la carte
        TRY=couche.RasterYSize
        Tx=float(gt[1])#taille du pixel en x
        Ty=float(gt[5])#taille du pixel en y

        #coordonnées géoréférencées
        y = float(lg)
        x  = float(lat)

        #transformation des coord géoréférencées en coordonnées pixels
        rasterx = int((x - gt[0]) / (gt[1]))    #coordonnées x en pixel(gt[0]:coord x coin haut gauche, gt[1]:taille du pixel en x(ouest-est))
        rastery = int((y - gt[3]) / (gt[5]))    #coordonnées y en pixel(gt[3]:coord y coin haut gauche, gt[5]:taille du pixel en y(sud-nord))
        bande1 = couche.GetRasterBand(1)

        #si raster multibandes:
        #bande2 = couche.GetRasterBand(2)
        #bande3 = couche.GetRasterBand(3)

        while no <= (float(altlacmax)+0.001):#boucle pour chaque altitude à simuler

                if os.path.exists('digue'+str(no)+'.png') is True:#si les fichiés existent déjà on les listent pour pouvour les effacers et afficher le noms
                dans l'application
                    outdiguepng='digue'+str(no)+'.png'              #et si ils existent déjà le programme ne les recalcule pas
                    listpng=listpng+[str(outdiguepng)]
                    listpng=listpng+['tailledig'+str(no)]
                    outmap = 'digue'+str(no)
                    listrastinG=listrastinG+[outmap]

                    cartedigue="cartedigue"+str(no)
```

```
            listrastinG=listrastinG+[cartedigue]

            inlac=nomlac+str(no)
            listrastinG=listrastinG+[inlac]
            outlacpng=inlac +".png"
            listpng=listpng+[str(outlacpng)]

            listLacinG=listLacinG+[inlac]

            f = open('tailledig'+str(no), 'r')
            dpg= f.readline()
            dpg=float(dpg)

            dtd["g"+str(no)]=str(int(dpg))

            no+=float(altlacpas)
            #im=nomlac+str(nu)+".png"
            #dig="digue"+str(nu)+".png"

        elif os.path.exists("digue"+str(no)+".png") is False:
            #definir la region , tres important qu'elle soit caller sur et à la resolution du raster utiliser
            grass.run_command('g.region',rast=entree, nsres=-float(gt[5]),ewres=float(gt[1]))

            #dans cette partie je fais chercher les coordonnées de la droite la plus courte
            #dont les extremitées atteignent la valeur d'altitudes choisies pour l'inondation
            #passant par le point de coordonnées de depart de la digue

            #pour l'essaie du programme je travail seulement sur 4 orientations
            #nordest/sudouest, nord/sud, nordouest/sudest, est/ouest
            #c'est resté car Philippe trouve cela suffisant
            #on part dans 8 directions à partir du points de départ
            #comme on veut des digues doites cela donne 4 directions passant par le point de depard


            #création des objets utilisés
            ryN3=ryN1=ryN2=rastery#definition des coords pixel de depard pour les differentes directions testées
            rxE3=rxE1=rxE2=rasterx
            rxO1=rxO2=rxO3=rasterx
            ryS1=ryS2=ryS3=rastery
            valN=0#objet valeur des pixels
            valS=0
            valNO=0
            valSO=0
            valNE=0
            valSE=0
            valO=0
            valE=0
            i1=i2=i3=i4=i5=i6=i7=i8=0#servira a calculer la distance en pixel
            stop=0#nombre de pixel depassant l'altitude desirée (voi manuel)
            ryN3L=ryN1L=ryN2L=None#coordonnées si on depasse une île dont la valeur est superieur à l'altitude désirée
            rxE3L=rxE1L=rxE2L=None
            rxO1L=rxO2L=rxO3L=None
            ryS1L=ryS2L=ryS3L=None


            ##NORD##
            #ici je fais avancer pixels par pixel
            #jusqu au moment ou la valeur du pixel depasse la valeur d altitude demandee
            while valN <= float(no) and ryN1 in range(int(TRY))  :#range TRY: pour que l'on reste dans la carte sinon créé des erreurs
                valN=bande1.ReadAsArray(rasterx,ryN1, 1, 1)       #lit la valeur du pixel de coordonnées (rasterx, ryN1)
                ryN1=ryN1-1   #-1 pour aller vers le nord
                i1=i1+1
            ryN1c=ryN1       #je continu avec un nouveau nom car si finalement il n'y a besoin de continuer je peux conservé les coordonées
            i1c=i1

            #pour eviter que la digue s'arrete sur une ile où l'altitude est superieur
            #je fais continuer tant qu'il n'y a pas plus de X pixel d'affillés qui depasse la valeur d'altitude demandee
            #si il y a plus de X pixel avec la valeur on concidere que ce n'est pas une ile

            while stop < TMI and ryN1c in range(int(TRY)):#continu tant que l on depasse le nombre de pixel depassant l'altitude desirée
                ryN1c=ryN1c-1
                i1c=i1c+1
                valN=bande1.ReadAsArray(rasterx,ryN1c, 1, 1)
                if valN > float(no):
                    stop=stop+1         #quand stop atteint TMI on ne considere plus que c'est une ile où l'altitude est superieur
                else:
                    stop=0              #on remet stop à 0 si on atteint une altitude inferieur à l'altitude desirée
                    ryN1L=ryN1c   #ici je garde la derniere valeur où l'on etait en dessous de la valeur d'altitude demandee
                    i1L=i1c
            if ryN1L != None:    #si il y avait effectivement une ile je change la valeur de coordonnées où s'arrete la digue
                  ryN1 = ryN1L-1
                  i1=i1L


            ##NORD OUEST##
            stop=0
            while valNO <= float(no)and ryN2 in range(int(TRY))and rxO2 in range(int(TRX)):
                valNO=bande1.ReadAsArray(rxO2,ryN2, 1, 1)
                ryN2=ryN2-1
                rxO2=rxO2+1
                i2=i2+1

            ryN2c=ryN2
            rxO2c=rxO2
            i2c=i2

            while stop < TMI and ryN2c in range(int(TRY))and rxO2c in range(int(TRX)):
                ryN2c=ryN2c-1
                rxO2c=rxO2c+1
                i2c=i2c+1
```

```
        valNO=bande1.ReadAsArray(rxO2c,ryN2c, 1, 1)
        if valNO > float(no):
            stop=stop+1
        else:
            stop=0
            ryN2L=ryN2c
            rxO2L= rxO2c
            i2L=i2c
    if ryN2L != None:
          ryN2 = ryN2L-1
          rxO2 = rxO2L+1
          i2=i2L


    ##NORD EST##
    stop=0
    while valNE <= float(no)and ryN3 in range(int(TRY))and rxE2 in range(int(TRX)):
        valNE=bande1.ReadAsArray(rxE2,ryN3, 1, 1)
        ryN3=ryN3-1
        rxE2=rxE2-1
        i3=i3+1

    ryN3c=ryN3
    rxE2c=rxE2
    i3c=i3

    while stop < TMI and ryN3c in range(int(TRY))and rxE2c in range(int(TRX)):
        ryN3c=ryN3c-1
        rxE2c=rxE2c-1
        i3c=i3c+1
        valNE=bande1.ReadAsArray(rxE2c,ryN3c, 1, 1)
        if valNE> float(no):
            stop=stop+1
        else:
            stop=0
            ryN3L=ryN3c
            rxE2L=rxE2c
            i3L=i3c
    if ryN3L != None:
        ryN3=ryN3L-1
        rxE2=rxE2L-1
        i3=i3L


    ##SUD##
    stop=0
    while valS <= float(no)and ryS1 in range(int(TRY)):
        valS=bande1.ReadAsArray(rasterx,ryS1, 1, 1)
        ryS1=ryS1+1
        i4=i4+1

    ryS1c=ryS1
    i4c=i4

    while stop < TMI and ryS1c in range(int(TRY)):
        ryS1c=ryS1c+1
        i4c=i4c+1
        valS=bande1.ReadAsArray(rasterx,ryS1c, 1, 1)
        if valS> float(no):
            stop=stop+1
        else:
            stop=0
            ryS1L=ryS1c
            i4L=i4c

    if ryS1L !=None:
        ryS1=ryS1L+1
        i4=i4L


    ##SUD OUEST##

    stop=0
    while valSO <= float(no)and ryS2 in range(int(TRY))and rxO3 in range(int(TRX)):
        valSO=bande1.ReadAsArray(rxO3,ryS2, 1, 1)
        ryS2=ryS2+1
        rxO3=rxO3+1
        i5=i5+1

    ryS2c=ryS2
    rxO3c=rxO3
    i5c=i5

    while stop < TMI and ryS2c in range(int(TRY))and rxO3c in range(int(TRX)):
        ryS2c=ryS2c+1
        rxO3c=rxO3c+1
        i5c=i5c+1
        valSO=bande1.ReadAsArray(rxO3c,ryS2c, 1, 1)
        if valSO > float(no):
            stop=stop+1
        else:
            stop=0
            ryS2L = ryS2c
            rxO3L = rxO3c
            i5L=i5c
    if ryS2L != None:
        ryS2=ryS2L+1
        rxO3=rxO3L+1
        i5=i5L
```

```
#SUD EST##
stop=0
while valSE <= float(no)and ryS3 in range(int(TRY))and rxE3 in range(int(TRX)):
    valSE=bande1.ReadAsArray(rxE3,ryS3, 1, 1)
    ryS3=ryS3+1
    rxE3=rxE3-1
    i6=i6+1

ryS3c=ryS3
rxE3c=rxE3
i6c=i6

while stop < TMI and ryS3c in range(int(TRY))and rxE3c in range(int(TRX)):
    ryS3c=ryS3c+1
    rxE3c=rxE3c-1
    i6c=i6c+1
    valSE=bande1.ReadAsArray(rxE3c,ryS3c, 1, 1)

    if valSE > float(no):
        stop=stop+1
    else:
        stop=0
        ryS3L=ryS3c
        rxE3L=rxE3c
        i6L=i6c
if ryS3L != None:
    ryS3=ryS3L
    rxE3=rxE3L
    i6=i6L


##EST##
stop=0
while valE <= float(no)and rxE1 in range(int(TRX)):
    valE=bande1.ReadAsArray(rxE1,rastery, 1, 1)
    rxE1=rxE1-1
    i7=i7+1

rxE1c=rxE1
i7c=i7

while stop < TMI and rxE1c in range(int(TRX)):
    rxE1c=rxE1c-1
    i7c=i7c+1
    valE=bande1.ReadAsArray(rxE1c,rastery, 1, 1)
    if valE > float(no):
        stop=stop+1
    else:
        stop=0
        rxE1L=rxE1c
        i7L=i7c

if rxE1L != None:
    rxE1=rxE1L
    i7=i7L

##OUEST##
stop=0
while valO <= float(no)and rxO1 in range(int(TRX)):
    valO=bande1.ReadAsArray(rxO1,rastery, 1, 1)
    rxO1=rxO1+1
    i8=i8+1

rxO1c=rxO1
i8c=i8

while stop < TMI and rxO1 in range(int(TRX)):
    rxO1c=rxO1c+1
    i8c=i8c+1
    valO=bande1.ReadAsArray(rxO1c,rastery, 1, 1)
    if valO > float(no):
        stop=stop+1
    else:
        stop=0
        rxO1L=rxO1c
        i8L=i8c

if rxO1L != None:
    rxO1=rxO1L
    i8=i8L



#ici je fais choisir la plus courte distances entre les deux points
    #parmis les 4 directions
if (i1+i4)<=(i2+i6)and (i1+i4)<=(i3+i5)and (i1+i4)<=(i7+i8):
    x1=rasterx
    y1=ryN1
    x2=rasterx
    y2=ryS1
    val1=valN
    val2=valS

elif (i2+i6)<(i1+i4)and(i2+i6)<=(i3+i5)and(i2+i6)<=(i7+i8):
    x1=rxO2
    y1=ryN2
    x2=rxE3
    y2=ryS3
    val1=valNO
```

```
            val2=valSE

        elif (i3+i5)<(i1+i4)and(i3+i5)<=(i7+i8):
            x1=rxE2
            y1=ryN3
            x2=rxO3
            y2=ryS2
            val1=valNE
            val2=valSO
        elif (i7+i8)<(i3+i5):
            x1=rxE1
            y1=rastery
            x2=rxO1
            y2=rastery
            val1=valO
            val2=valE

        #ici je convertie les coord pixel en coord georeferenceees
        gx1 = float((x1*gt[1]) +gt[0])
        gx2 = float((x2*gt[1]) +gt[0])
        gy1 = float(y1*(gt[5])+gt[3])
        gy2 = float(y2*(gt[5])+gt[3])

        #calcule de longueur de la digue
        dp=(sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2))) #longueur en pixel
        dpg=(sqrt((((x1-x2)*Tx)*((x1-x2)*Tx))+((y1-y2)*Ty)*((y1-y2)*Ty)))#longueur en metres
        f = open('tailledig'+str(no),'w')
        f.write(str(dpg))
        f.close()
        listpng=listpng+['tailledig'+str(no)]

        dtd["p"+str(no)]=str(int(dp))
        dtd["g"+str(no)]=str(int(dpg))#lie dans un catalogue l'altitude et la longueur de la digue

        #ici je crée un fichier vecteur ligne en ascii dont les extremités sont les deux points dont les coordonnées ont été définies ci dessus
        f = open('coordline.asc','w')
        f.write("VERTI:\n"+"L 2\n"+str(gx1)+" "+str(gy1)+"\n"+str(gx2)+" "+str(gy2))
        f.close()#tres important si le fichier n est pas ferme il n'est pas pris en compte quand on s'en sert pour creer le vecteur

################################### FIN DU TRAVAIL AVEC GDAL ##########################################################################


################################### DEBUT DU TRAVAIL DANS GRASS ##########################################################################


        outputvect="outputvect"
        grass.run_command('v.in.ascii', input='coordline.asc', output=outputvect,format="standard")#j'import dans grass le vecteur ligne de la
        digue
        vectcat=outputvect+"cat"
        grass.run_command('v.category', input=outputvect, output=vectcat)#création de la colonne utilisée pour la convertion en raster
        rastdigue=filename+"digue"
        grass.run_command('v.to.rast', input=vectcat, output=rastdigue, use="cat")#la digue est convertie en raster pour etre intégrée au MNT


        #ici j'epaissi la digue car si elle est en diagonale avec un seul pixel d'epaisseur les pixels de part et d'autre sont en contact, la
        digue serait comme percée
        if res =="originale":
            digue2=rastdigue+"2"
            grass.run_command('r.grow', input=rastdigue, output=digue2)

        else:
            dres=int(float(res)/float(Tx))#l'epaissiment de la digue dependra directement du ratio  entre la resolution original et la ressolution
            final
            digue2=rastdigue+"2"
            grass.run_command('r.grow', input=rastdigue, output=digue2,radius=dres)

        #redefinition de la region pour n'afficher que la zone desirée et diminuer le temps de calcule (optionnel)
        if res =="originale":
            Tyn=-Ty
            grass.run_command('g.region', vect=mask,nsres=Tyn, ewres=Tx)
        else:
            grass.run_command('g.region',vect=mask, res=str(res) )

        #création la 3eme dimension (alt) de la digue en fonction de l'altitude le l'inondation desirée
        inmap = digue2
        outmap = "digue"+str(no)
        grass.mapcalc("$outmap = int($inmap + $value)", inmap = inmap, outmap = outmap, value = altlacmax)
        outdiguepng=str(outmap)+".png"
        grass.run_command('r.out.png',input=outmap, output=outdiguepng, flags='t')#exportation en png pour affichage
                                            #"flags='t' important: rend transparent les pixels de valeur NULL permet de la superposer au MNT
        listpng=listpng+[str(outdiguepng)]#ajout du nom du fichier créé à la liste (reutilisation et effacage)
        grass.run_command('r.null', map=outmap, null=0)#null=0 servira à l'ajout de la digue sur le MNT avec mapcalc, j'utilise la fonction
        "if(x,a,b,c)"
                                                #si x>0 =>a, si x=0 =>b, si x<0 =>c    ( si digue >0 =>digue, si digue =0 =>MNT )
        listrastinG=listrastinG+[outmap]


        #efface les fichiers intermediaire, important car sinon le programme les reutilises et ne recalcule pas quand les parametres changent
        grass.run_command('g.remove', rast=rastdigue, vect=outputvect)
        grass.run_command('g.remove', rast=digue2, vect=vectcat)

        #construction de la carte utilisée pour simuler l'inondation, MNT sur lequel on ajoute la digue
        inmap=entree
        inmap2=outmap
        outmap="cartedigue"+str(no)
        grass.mapcalc("$outmap = float(if($inmap2,$inmap2,$inmap,$inmap))", inmap = inmap, inmap2=inmap2, outmap = outmap)
        cartedigue=outmap
        listrastinG=listrastinG+[cartedigue]
```

```
                #CREATION DES INONDATIONS
                inlac=nomlac+str(no)

                grass.run_command('r.lake', dem=cartedigue, wl=no, xy=(coordr),lake=inlac, flags='n')
                listrastinG=listrastinG+[inlac]
                listLacinG=listLacinG+[inlac]
                outlacpng=inlac +".png"
                grass.run_command('r.out.png',input=inlac, output=outlacpng, flags='t')
                listpng=listpng+[str(outlacpng)]
                no+=float(altlacpas)


    #création de l'image de fond (exportation du MNT en .png)
    if res =="originale":
        Tyn=-Ty
        grass.run_command('g.region', vect=mask,nsres=Tyn, ewres=Tx)
    else:
        grass.run_command('g.region',vect=mask, res=str(res) )
    if fond=='MNT':
        pbase=str(entree)+".png"
        grass.run_command('r.colors', map=entree,flags='e', color=couleur)
        grass.run_command('r.out.png',input=entree, output=pbase)
    else:
        pbase=str(fond)+".png"
        grass.run_command('r.colors', map=fond,flags='e', color=couleur)
        grass.run_command('r.out.png',input=fond, output=pbase)
    imbase=pbase
    listpng=listpng+[str(imbase)]


    ################################## FIN DU TRAVAIL DANS GRASS ###########################################################################

    ##preparation de l'image affichée à l'ouverture de la fenetre d'affichage
    self.imbORIG = wx.Image(imbase, wx.BITMAP_TYPE_ANY)
    self.imbORIX = self.imbORIG.GetWidth()
    self.imbORIY = self.imbORIG.GetHeight()
    largeurb = (self.imbORIX * self.ratio)/100
    hauteurb = (self.imbORIY * self.ratio)/100
    self.bmpBASE = self.imbORIG.Scale(largeurb, hauteurb).ConvertToBitmap()
    self.bmpLAC= None#me sert ensuite de condition (if self.bmpLAC != None: ...)
                    #pour distinguer les cas où il y a juste la carte de fond de quand il y a aussi l'inondation
    self.panneau.Bind(wx.EVT_PAINT, self.OnPaint)#lie les evenenments "la fenetre doit etre redessinnée": (deplacement , changement de taille, defilement)
    self.Show()                                  #avec la partie du programme ecrite plus bas: "def On.paint"

def imprimer (self,evt):
    #enregistre ce qui est dans la fenetre d'affichage
    #ima = wx.Image(imbase, wx.BITMAP_TYPE_ANY)
    imaX = (self.panneau.GetSize()[0])
    imaY = (self.panneau.GetSize()[1])
    dc = wx.ClientDC(self.panneau)
    bmp = wx.EmptyBitmap(imaX, imaY)
    memDC = wx.MemoryDC()
    memDC.SelectObject(bmp)
    memDC.Blit(0,0,imaX,imaY,dc,0,0)
    memDC.SelectObject(wx.NullBitmap)
    img = bmp.ConvertToImage()
    nomdufichier1 = "monImage.png"
    img.SaveFile(nomdufichier1, wx.BITMAP_TYPE_PNG)
    self.ouvrir()
    if fic2 == None:
        os.remove(nomdufichier1)

    else:
        repfic2=os.path.join(rep,fic2+".png")
        os.rename(nomdufichier1,repfic2)


def imprimerencour (self):
    #enregistre ce qui est dans la fenetre d'affichage pour chaque altitude d'inondation
    #puis les stocke dans une liste pour qu'elles puissent être enregistrées ou effacées
    global lstphoto
    #ima = wx.Image(imbase, wx.BITMAP_TYPE_ANY)
    imaX = (self.panneau.GetSize()[0])
    imaY = (self.panneau.GetSize()[1])
    dc = wx.ClientDC(self.panneau)
    bmp = wx.EmptyBitmap(imaX, imaY)
    memDC = wx.MemoryDC()
    memDC.SelectObject(bmp)
    memDC.Blit(0,0,imaX,imaY,dc,0,0)
    memDC.SelectObject(wx.NullBitmap)
    img = bmp.ConvertToImage()
    nomdufichier ="photo"+str(nu)
    nomdufichierpng="photo"+str(nu)+".png"
    img.SaveFile(nomdufichierpng, wx.BITMAP_TYPE_PNG)
    lstphoto=lstphoto+[str(nomdufichier)]

def base(self,evt):#(bouton)reaffiche la carte de fond (MNT)
    global fen

    self.imbORIG = wx.Image(imbase, wx.BITMAP_TYPE_ANY)
    self.imbORIX = self.imbORIG.GetWidth()
    self.imbORIY = self.imbORIG.GetHeight()
    largeurb = (self.imbORIX * self.ratio)/100
    hauteurb = (self.imbORIY * self.ratio)/100
    self.bmpBASE = self.imbORIG.Scale(largeurb, hauteurb).ConvertToBitmap()


    dc = wx.ClientDC(self.panneau)#créer une zone où l'on peux dessiner, afficher image ...
    self.panneau.PrepareDC(dc)#tres important !!! prepare les coordonées virtuelles necessaire lors du defilement (droite/gauche , monter/descendre l'image)
    dc.DrawBitmap(self.bmpBASE,0,0)
```

```
        self.bmpLAC= None

        self.SetTitle("Visualiseur d'images [%s]"% self.bmpBASE)
        self.Show(True)



    def Retour(self, evt):#(bouton) affiche la carte a sa taille originale
        self.ratio = 100
        if self.bmpBASE != None and self.bmpLAC!= None:#si on affiche l'inondation


            largeurd = (self.imdORIX * self.ratio)/100
            hauteurd = (self.imdORIY * self.ratio)/100
            self.bmpDIG = self.imdORIG.Scale(largeurd, hauteurd).ConvertToBitmap()

            largeurb = (self.imbORIX * self.ratio)/100
            hauteurb = (self.imbORIY * self.ratio)/100
            self.bmpBASE = self.imbORIG.Scale(largeurb, hauteurb).ConvertToBitmap()

            largeurl = (self.imlORIX * self.ratio)/100
            hauteurl = (self.imlORIY * self.ratio)/100
            self.bmpLAC = self.imlORIG.Scale(largeurl, hauteurl).ConvertToBitmap()

            dc = wx.ClientDC(self.panneau)
            mask = wx.Mask(self.bmpLAC, wx.BLACK)#me sert a pouvoir affiche les trois image l'une sur l'autre
            self.panneau.PrepareDC(dc)
            dc.DrawBitmap(self.bmpBASE,0,0,False)
            dc.DrawBitmap(self.bmpLAC,0,0,True)
            dc.DrawBitmap(self.bmpDIG,0,0,True)

            self.SetTitle("Visualiseur d'inondation [%s]"% im+"  longueur digue = "+str(dtd["g"+str(nu)])+" metres")#utilisation du catalogue "dtd" pour
            retrouver la longueur calculée de la digue
            self.Show(True)                                                                    #en fonction de l'altidude de l'inondation
            affichée

        else:#si on affiche seulement la carte de base
            largeurb = (self.imbORIX * self.ratio)/100
            hauteurb = (self.imbORIY * self.ratio)/100
            self.bmpBASE = self.imbORIG.Scale(largeurb, hauteurb).ConvertToBitmap()

            dc = wx.ClientDC(self.panneau)
            dc.DrawBitmap(self.bmpBASE,0,0)
            self.SetTitle("Visualiseur de carte [%s]"% im)
            self.Show(True)




    def Plus(self, evt):#(bouton)agrandi la taille d affichage de la carte
        self.ratio = self.ratio + self.inc #augment le ratio d'affichage(self.ratio) de +"self.inc"
        if self.bmpLAC!= None:

            #pour la digue
            largeurd = (self.imdORIX * self.ratio)/100
            hauteurd = (self.imdORIY * self.ratio)/100
            self.bmpDIG = self.imdORIG.Scale(largeurd, hauteurd).ConvertToBitmap()

            #pour le MNT
            largeurb = (self.imbORIX * self.ratio)/100
            hauteurb = (self.imbORIY * self.ratio)/100
            self.bmpBASE = self.imbORIG.Scale(largeurb, hauteurb).ConvertToBitmap()

            #pour l'inondation
            largeurl = (self.imlORIX * self.ratio)/100
            hauteurl = (self.imlORIY * self.ratio)/100
            self.bmpLAC = self.imlORIG.Scale(largeurl, hauteurl).ConvertToBitmap()

            dc = wx.ClientDC(self.panneau)
            mask = wx.Mask(self.bmpLAC, wx.BLACK)
            self.panneau.PrepareDC(dc)
            dc.DrawBitmap(self.bmpBASE,0,0,False)
            dc.DrawBitmap(self.bmpLAC,0,0,True)
            dc.DrawBitmap(self.bmpDIG,0,0,True)

            self.SetTitle("Visualiseur de carte [%s]"% im)
            self.Show(True)

        else:

            largeurb = (self.imbORIX * self.ratio)/100
            hauteurb = (self.imbORIY * self.ratio)/100
            self.bmpBASE = self.imbORIG.Scale(largeurb, hauteurb).ConvertToBitmap()

            dc = wx.ClientDC(self.panneau)
            self.panneau.PrepareDC(dc)
            dc.DrawBitmap(self.bmpBASE,0,0)
            self.SetTitle("Visualiseur d'inondation [%s]"% im+"  longueur digue = "+str(dtd["g"+str(nu)])+" metres")
            self.Show(True)


    def Moins(self, evt): #(bouton)diminu la taille d affichage de la carte
        self.ratio = self.ratio - self.inc
        if self.bmpLAC!= None:
            self.ratio = self.ratio - self.inc

            largeurd = (self.imdORIX * self.ratio)/100
            hauteurd = (self.imdORIY * self.ratio)/100
            self.bmpDIG = self.imdORIG.Scale(largeurd, hauteurd).ConvertToBitmap()
```

```
            largeurb = (self.imbORIX * self.ratio)/100
            hauteurb = (self.imbORIY * self.ratio)/100
            self.bmpBASE = self.imbORIG.Scale(largeurb, hauteurb).ConvertToBitmap()

            largeurl = (self.imlORIX * self.ratio)/100
            hauteurl = (self.imlORIY * self.ratio)/100
            self.bmpLAC = self.imlORIG.Scale(largeurl, hauteurl).ConvertToBitmap()



            dc = wx.ClientDC(self.panneau)
            mask = wx.Mask(self.bmpLAC, wx.BLACK)
            self.panneau.PrepareDC(dc)
            dc.DrawBitmap(self.bmpBASE,0,0,False)
            dc.DrawBitmap(self.bmpLAC,0,0,True)
            dc.DrawBitmap(self.bmpDIG,0,0,True)

            self.SetTitle("Visualiseur de carte [%s]"% im)
            self.Show(True)

        else:

            largeurb = (self.imbORIX * self.ratio)/100
            hauteurb = (self.imbORIY * self.ratio)/100
            self.bmpBASE = self.imbORIG.Scale(largeurb, hauteurb).ConvertToBitmap()

            dc = wx.ClientDC(self.panneau)
            self.panneau.PrepareDC(dc)
            dc.DrawBitmap(self.bmpBASE,0,0)
            self.SetTitle("Visualiseur d'inondation [%s]"% im+"  longueur digue = "+str(dtd["g"+str(nu)])+" metres")
            self.Show(True)

    def monter(self,evt):#(bouton)monte le niveau d eau
        global im, nu, nomlac
        #pour chaque "cession" il n'y a qu'une inondation par altitude donc
        #la référence pour jongler entre les objets est leur atitude (qui a été mise dans leur nom) ici "nu"
        if nu <= (float(altlacmax)-float(altlacpas)+0.001):#pour ne pas aller plus loin que l'altitude la plus haute
            nu+=float(altlacpas)
            im=nomlac+str(nu)+".png"
            dig="digue"+str(nu)+".png"


            self.imdORIG = wx.Image(dig, wx.BITMAP_TYPE_ANY)
            self.imdORIX = self.imdORIG.GetWidth()
            self.imdORIY = self.imdORIG.GetHeight()
            largeurd = (self.imdORIX * self.ratio)/100
            hauteurd = (self.imdORIY * self.ratio)/100
            self.bmpDIG = self.imdORIG.Scale(largeurd, hauteurd).ConvertToBitmap()

            self.imlORIG = wx.Image(im, wx.BITMAP_TYPE_ANY)
            self.imlORIX = self.imlORIG.GetWidth()
            self.imlORIY = self.imlORIG.GetHeight()
            largeurl = (self.imlORIX * self.ratio)/100
            hauteurl = (self.imlORIY * self.ratio)/100
            self.bmpLAC = self.imlORIG.Scale(largeurl, hauteurl).ConvertToBitmap()

            self.imbORIG = wx.Image(imbase, wx.BITMAP_TYPE_ANY)
            self.imbORIX = self.imbORIG.GetWidth()
            self.imbORIY = self.imbORIG.GetHeight()
            largeurb = (self.imbORIX * self.ratio)/100
            hauteurb = (self.imbORIY * self.ratio)/100
            self.bmpBASE = self.imbORIG.Scale(largeurb, hauteurb).ConvertToBitmap()

            dc = wx.ClientDC(self.panneau)
            mask = wx.Mask(self.bmpLAC, wx.BLACK)
            self.panneau.PrepareDC(dc)
            dc.DrawBitmap(self.bmpBASE,0,0,False)
            dc.DrawBitmap(self.bmpLAC,0,0,True)
            dc.DrawBitmap(self.bmpDIG,0,0,True)

            self.SetTitle("Visualiseur d'inondation [%s]"% im+"  longueur digue = "+str(dtd["g"+str(nu)])+" metres")
            self.Show(True)
            if os.path.exists("photo"+str(nu)+".png") is False:
                self.imprimerencour()


    def baisser(self,evt):#(bouton)baisse le niveau d eau
        global im, nu, nomlac, dp, dpg, Tx,Ty


        if nu > float(altlacmin):
            nu-=float(altlacpas)

            im=nomlac+str(nu)+".png"
            dig="digue"+str(nu)+".png"


            self.imdORIG = wx.Image(dig, wx.BITMAP_TYPE_ANY)
            self.imdORIX = self.imdORIG.GetWidth()
            self.imdORIY = self.imdORIG.GetHeight()
            largeurd = (self.imdORIX * self.ratio)/100
            hauteurd = (self.imdORIY * self.ratio)/100
            self.bmpDIG = self.imdORIG.Scale(largeurd, hauteurd).ConvertToBitmap()

            self.imlORIG = wx.Image(im, wx.BITMAP_TYPE_ANY)
            self.imlORIX = self.imlORIG.GetWidth()
            self.imlORIY = self.imlORIG.GetHeight()
            largeurl = (self.imlORIX * self.ratio)/100
            hauteurl = (self.imlORIY * self.ratio)/100
            self.bmpLAC = self.imlORIG.Scale(largeurl, hauteurl).ConvertToBitmap()
```

```
            self.imbORIG = wx.Image(imbase, wx.BITMAP_TYPE_ANY)
            self.imbORIX = self.imbORIG.GetWidth()
            self.imbORIY = self.imbORIG.GetHeight()
            largeurb = (self.imbORIX * self.ratio)/100
            hauteurb = (self.imbORIY * self.ratio)/100
            self.bmpBASE = self.imbORIG.Scale(largeurb, hauteurb).ConvertToBitmap()

            dc = wx.ClientDC(self.panneau)
            mask = wx.Mask(self.bmpLAC, wx.BLACK)
            self.panneau.PrepareDC(dc)
            dc.DrawBitmap(self.bmpBASE,0,0,False)
            dc.DrawBitmap(self.bmpLAC,0,0,True)
            dc.DrawBitmap(self.bmpDIG,0,0,True)

            self.SetTitle("Visualiseur d'inondation [%s]"% im+"  longueur digue = "+str(dtd["g"+str(nu)])+" metres")
            self.Show(True)
            if os.path.exists("photo"+str(nu)+".png") is False:
                self.imprimerencour()

    def aide (self,evt):
        fen2= Tk()
        tex1 = Label(fen2, text='si vous avez un probleme n\'hesitez pas a me contacter', fg='red')
        tex1.pack()
        text2 = Label(fen2, text='yann.pottier@yahoo.com', fg='black')
        text2.pack()
        fen2.mainloop()

    def manuel(self,evt):
        os.startfile("manuel d utilisation.pdf")


    def OnPaint(self, evt):#redessine les images quand on les bouge ou redimensionne la fenetre
        if self.bmpLAC!= None:
            dc = wx.ClientDC(self.panneau)
            mask = wx.Mask(self.bmpLAC, wx.BLACK)
            self.panneau.PrepareDC(dc)
            dc.DrawBitmap(self.bmpBASE,0,0,False)
            dc.DrawBitmap(self.bmpLAC,0,0,True)
            dc.DrawBitmap(self.bmpDIG,0,0,True)

        else:

            dc = wx.PaintDC(self.panneau)
            self.panneau.PrepareDC(dc)
            dc.DrawBitmap(self.bmpBASE,0,0)

        evt.Skip()



    #les def qui suivent servent a gerer les cartes et images crees pour l application quand on la ferme
    #"if os.path.exists" est utilisé pour laisser le programme faire la suite quand il ne trouve pas un des fichiers à effacer
    def ouvrir(self):
        global rep,fic,fic2
        root = Tk()
        repfic = tkFileDialog.asksaveasfilename(title="sauver le fichier:", initialdir=rep, \
                        initialfile=fic, filetypes = [("All", "*"),("Fichiers PNG","*.png")])
        if len(repfic) > 0:
            rep=os.path.dirname(repfic) #chemin d'acces selectionné enregistrer sous forme d'objet
            fic2=os.path.basename(repfic)#nom choisi pour fichier enregistré
        else:

            fic2=None
        root.destroy()#root=Tk() et root.destroy sont là pour fermé une fernetre Tk qui apparait
                        #avec tkFileDialog et qui sinon reste et fini par bloquer le programme

    def ouvrir2(self):
        global rep,fic
        root = Tk()
        nom="rien a ecrire, vous avez deja choisi le nom dans les options"
        repfic = tkFileDialog.asksaveasfilename(title="choisir dossier", initialdir=rep, \
                        initialfile=nom, filetypes = [("All", "*"),("Fichiers PNG","*.png")])

        rep=os.path.dirname(repfic) #chemin d'acces selectionné enregistrer sous forme d'objet
        fic=os.path.basename(repfic)

        root.destroy()#root=Tk() et root.destroy sont là pour fermé une fernetre Tk qui apparait
                        #avec tkFileDialog et qui sinon reste et fini par bloquer le programme

    def garderpng(self):#efface les cartes crées dans Grass
        global fen1, nu,rep
        "efface les fichiers créés dans Grass pour l'application"
        postnom= options['postnom']


        self.ouvrir2()
        if rep != "":#si un repertoire a été choisie

            nu=float(altlacmin)
            while nu <float(altlacmax)+0.0001:#boucle pour enregistrer les images d'inondation pour chaque altitude


                fic2=str(postnom)+"photo"+str(nu)+'D'+str(dtd["g"+str(nu)])+'.png'
                repfic2=os.path.join(rep,fic2)
                if os.path.exists("photo"+str(nu)+'.png') is True:
                    os.rename("photo"+str(nu)+'.png',repfic2)
                nu=nu+float(altlacpas)
```

```
            grass.run_command('g.remove', rast=listrastinG)#efface les raster créés dans Grass
            if os.path.exists(filename) is True:
                os.remove(filename)#efface MNT exporter pourGdal

            if os.path.exists(filename+'.aux.xml') is True:
                os.remove(filename+'.aux.xml')

                os.remove('coordline.asc')#efface fichier vecteur ligne ascii
            a=0
            n=len(listpng)
            while a <n:
                try:
                    os.remove(listpng[a])#efface les fichiers qui ont servi  à l'affichage
                    a=a+1
                except:
                    a=a+1
            self.Destroy()
            fen1.destroy()
        else:#si à la place de choisir un repertoire on a cliqué sur annuler
            fen1.destroy()

    def effacertous(self):#effaces toutes les cartes dans Grass et images png crees pour l application
        global fen1, nu
        "efface tous les fichiers créés pour l'application"
        grass.run_command('g.remove', rast=listrastinG)
        a=0
        n=len(listpng)
        for a in range(n):
            if os.path.exists(listpng[a]) is True:
                os.remove(listpng[a])
        b=0
        m=len(lstphoto)
        for b in range(m):
            if os.path.exists(str(lstphoto[b])+'.png') is True:
                os.remove(str(lstphoto[b])+'.png')

        if os.path.exists(filename) is True:
            os.remove(filename)#efface MNT exporter pourGdal
        if os.path.exists(filename+'.aux.xml') is True:
            os.remove(filename+'.aux.xml')
        if os.path.exists('coordline.asc') is True:
            os.remove('coordline.asc')#efface fichier vecteur ligne ascii

        self.Destroy()
        fen1.destroy()

    def annuler(self):
        global fen1
        "annule la fermeture de l'application"
        fen1.destroy()


    def gardergrass (self):
        global fen1,nu
        "renomme les fichiers dans grass et efface les images créés pour l application"
        postnom= options['postnom']
        b=0
        m=len(listLacinG)

        while b < m:
            nu=(float(b)*float(altlacpas))+float(altlacmin)
            grass.run_command('g.region', rast=listLacinG[b])
            inmap=listLacinG[b]
            outmap=str(postnom)+str(listLacinG[b])+'D'+str(dtd["g"+str(nu)])
            grass.mapcalc("$outmap = float($inmap)", inmap = inmap, outmap = outmap)
            grass.run_command('r.colors', map=str(outmap),flags='e', raster=str(listLacinG[b]))
            b=b+1
        grass.run_command('g.region', rast=entree)
        grass.run_command('g.remove', rast=listrastinG)
        a=0
        n=len(listpng)
        for a in range(n):
            if os.path.exists(listpng[a]) is True:
                os.remove(listpng[a])
        c=0
        m=len(lstphoto)
        for c in range(m):
            if os.path.exists(str(lstphoto[c])+'.png') is True:
                os.remove(str(lstphoto[c])+'.png')
        if os.path.exists('coordline.asc') is True:
            os.remove('coordline.asc')#efface fichier vecteur ligne ascii
        if os.path.exists(filename) is True:
            os.remove(filename)#efface MNT exporter pourGdal
        if os.path.exists(filename+'.aux.xml') is True:
            os.remove(filename+'.aux.xml')
        self.Destroy()
        fen1.destroy()

    def gardertous(self):
        global fen1, nu,lstphoto
        "garde les fichiers créés dans Grass pour l'application"
        postnom= options['postnom']


        self.ouvrir2()
        if rep != "":

            nu=float(altlacmin)
            while nu <float(altlacmax)+0.0001:
```

```
                fic2=str(postnom)+"photo"+str(nu)+'D'+str(dtd["g"+str(nu)])+'.png'
                repfic2=os.path.join(rep,fic2)
                if os.path.exists("photo"+str(nu)+'.png') is True:
                    os.rename("photo"+str(nu)+'.png',repfic2)
                nu=nu+float(altlacpas)

            c=0
            l=len(listLacinG)

            while c < l:
                nu=(float(c)*float(altlacpas))+float(altlacmin)
                grass.run_command('g.region', rast=listLacinG[c])
                inmap=listLacinG[c]
                outmap=str(postnom)+str(listLacinG[c])+'D'+str(dtd["g"+str(nu)])
                grass.mapcalc("$outmap = float($inmap)", inmap = inmap, outmap = outmap)
                grass.run_command('r.colors', map=str(outmap),flags='e', raster=str(listLacinG[c]))
                c=c+1

            grass.run_command('g.remove', rast=listrastinG)



            if os.path.exists(filename) is True:
                os.remove(filename)#efface MNT exporter pourGdal
            if os.path.exists(filename+'.aux.xml') is True:
                os.remove(filename+'.aux.xml')
            if os.path.exists('coordline.asc') is True:
                os.remove('coordline.asc')#efface fichier vecteur ligne ascii
            a=0
            n=len(listpng)
            while a <n:
                try:
                    if os.path.exists(listpng[a]) is True:
                        os.remove(listpng[a])
                    a=a+1
                except:
                    a=a+1
            self.Destroy()
            fen1.destroy()

        else:

            fen1.destroy()



    def OnExit(self, evt):#ouvre une fenetre pour gerer les cartes et image crees pour l application, avant de la fermer

        "fenetre qui apparait quand on quitte, demande si on efface les fichiés crées pour l'application"
        global listrastinG, imbase, fen1
        grass.run_command('g.region', rast=entree) #redefini la région du secteur grass (modifié par le programme) en ce basant sur le raster utilisé

        fen1= Tk()
        bou5=Button(fen1,text='sauvegarder les images en .png', command = self.garderpng).grid(row =3, column =1)
        bou10=Button(fen1,text='sauvegarder les fichiers inondation dans GRASS', command = self.gardergrass).grid(row =5, column =1)
        bou6=Button(fen1,text='sauvegarder les images en .png et les raster inondations dans GRASS', command = self.gardertous).grid(row =7, column =1)
        bou7=Button(fen1,text='effacer tous les fichiers et images créées', command = self.effacertous).grid(row =1, column =1)
        bou9=Button(fen1, text='annuler', command = self.annuler).grid(row =11, column =1)

        fen1.mainloop()


class MonApp(wx.App):
    def OnInit(self):
        wx.InitAllImageHandlers()
        fen = main("Visualiseur de carte")
        fen.Show(True)
        self.SetTopWindow(fen)
        return True


if __name__ == "__main__":
    options, flags = grass.parser()#permet la prise en compte des options et flags de grass
    app = MonApp()
    app.MainLoop()#lance l application (enfin !!)
```