gui_unifiée/class_TabAffichageCoupes.py

```
import os
import sys

#from PyQt5.QtCore import *
#from PyQt5.QtGui import *
#from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
from PyQt5.QtGui import QPixmap
from PyQt5.QtWidgets import QApplication, QWidget, QGridLayout, QLabel,
QScrollBar, QHBoxLayout, QVBoxLayout, QGroupBox,QRadioButton, QPushButton,
QProgressBar
import functools
from math import floor

from class_Parametres import Parametres # Ne sert que si est exécuté
séparemment
from class_TabGraphique3D import Graphique3D
from class_AfficheLienVTK import AfficheLienVTK

import numpy as np

from platform import system as systemPlatform


######## Il faut que je modifie le code dans tracking 3D pour faire
######## ../extraction pout pouvoir le lancer depuis mon code
if systemPlatform() == "Linux" :
    sys.path.append("../extraction")
    from tracking_3D import retrouve_grain


"""
PARAMETRES
"""
IMAGE_AXES = "image_AxesXYZ.png"


"""
Classe TabAffichageCoupes, hérite de la classe QGridLayout, c'est donc une
grille
Cette classe représente le contenu d'une fenêtre PyQt
Elle peut donc aussi être utilisée comme un onglet dans une fenêtre
@author Maylis et Alexandre
"""
class TabAffichageCoupes(QGridLayout) :
    """
    Constructeur, crée le contenu de l'onglet
    """
    def __init__(self, objParams, parent=None) :
        super(TabAffichageCoupes, self).__init__(parent) # Appel du
constructeur de QGridLayout

        self.objParams = objParams


        """
        Création des barres de Scroll
        """
```

```
        # Défilement de la couche X
        self.barreScrollAxeX = QScrollBar()
        self.barreScrollAxeX.setMaximum(
self.objParams.nombreImagesPlanYZ() )
        self.barreScrollAxeX.valueChanged.connect( self.changeImages )

#self.barreScrollAxeX.setValue(round(self.objParams.nombreImagesPlanYZ()/2)
)

        # Défilement de la couche Y
        self.barreScrollAxeY = QScrollBar()
        self.barreScrollAxeY.setMaximum(
self.objParams.nombreImagesPlanXZ() )
        self.barreScrollAxeY.valueChanged.connect( self.changeImages )

#self.barreScrollAxeY.setValue(round(self.objParams.nombreImagesPlanXZ()/2)
)

        # Défilement de la couche Z
        self.barreScrollAxeZ = QScrollBar()
        self.barreScrollAxeZ.setMaximum(
self.objParams.nombreImagesPlanXY() )
        self.barreScrollAxeZ.valueChanged.connect( self.changeImages )

#self.barreScrollAxeZ.setValue(round(self.objParams.nombreImagesPlanXY()/2)
)

        # Défilement temporel
        self.barreScrollTemps = QScrollBar(Qt.Horizontal)
        self.barreScrollTemps.setMaximum(
self.objParams.nombreInstantsTemporels() )
        self.barreScrollTemps.valueChanged.connect( self.changeImages )

        # Ajout des barres de scroll
        self.addWidget(self.barreScrollTemps, 2, 1)
        self.addWidget(self.barreScrollAxeX, 1, 2)
        self.addWidget(self.barreScrollAxeY, 1, 3)
        self.addWidget(self.barreScrollAxeZ, 1, 4)




        """
        Création de toute la partie images
        """
        # Création d'un contenant pour les images
        contenant_widget = QWidget()
        contenant_grille = QGridLayout()

        # Création des labels allant contenir les images
        self.label_image_xy = QLabel()
        self.label_image_yz = QLabel()
        self.label_image_zx = QLabel()

        # Rendre les labels cliquables pour retrouver le grain

self.label_image_xy.mousePressEvent=functools.partial(self.get_pixel,
source_object=self.label_image_xy)

self.label_image_yz.mousePressEvent=functools.partial(self.get_pixel,
source_object=self.label_image_yz)
```

```
self.label_image_zx.mousePressEvent=functools.partial(self.get_pixel,
source_object=self.label_image_zx)

        # Epêcher le redimentionnement des images
        self.label_image_xy.setFixedSize(240,240)
        self.label_image_yz.setFixedSize(240,500)
        self.label_image_zx.setFixedSize(240,500)

        # Ajout des images dans le contenant à images
        contenant_grille.addWidget(self.label_image_xy, 5, 2)
        contenant_grille.addWidget(self.label_image_yz, 2, 3)
        contenant_grille.addWidget(self.label_image_zx, 2, 1)
        contenant_widget.setLayout(contenant_grille)

        # Création des textes correspondant aux images (au dessus)
        texte_xy = QLabel("Image (X, Y)")
        texte_yz = QLabel("Image (Y, Z)")
        texte_zx = QLabel("Image (X, Z)")

        # Ajout dans la grille des textes au dessus des images
        contenant_grille.addWidget(texte_xy, 3, 2)
        contenant_grille.addWidget(texte_yz, 1, 3)
        contenant_grille.addWidget(texte_zx, 1, 1)

        # Rendre le layout avec les images plus gros que les autres
        self.setColumnStretch(1,2)

        # Image centrale avec les axes
        label_image_axes = QLabel()
        label_image_axes.setPixmap(QPixmap(IMAGE_AXES))
        contenant_grille.addWidget(label_image_axes,2,2)
        if not os.path.isfile( IMAGE_AXES ) : # Si le chemin d'accès à
l'image n'existe pas
            print( "[Erreur TabAffichageCoupes] " + IMAGE_AXES + " n'existe
pas !" )



        """
        Positions courantes de X, Y, Z et du temps
        """
        # Création d'un contenant pour les valeurs courantes
        group_box=QGroupBox("Positions courantes des barres de scroll")
        horizontal_layout = QHBoxLayout()
        group_box.setLayout(horizontal_layout)

        # Création des labels qui affichent les valeurs courantes
        self.valeur_temps = QLabel("Temps : 0")
        self.valeur_X = QLabel("X : 0")
        self.valeur_Y = QLabel("Y : 0")
        self.valeur_Z = QLabel("Z : 0")

        # Ajout des labels dans le contenant
        horizontal_layout.addWidget(self.valeur_temps)
        horizontal_layout.addWidget(self.valeur_X)
        horizontal_layout.addWidget(self.valeur_Y)
        horizontal_layout.addWidget(self.valeur_Z)
```

```
        # Ajout dans un layout vertical contenant les images et les
positions courantes
        vertical_layout=QVBoxLayout()
        vertical_layout.addWidget(group_box)
        vertical_layout.addWidget(contenant_widget,stretch=2)

        # Ajout dans l'onglet
        self.addLayout(vertical_layout,1,1)



        """
        Création d'un RadioButton pour choisir le traitement à afficher
        """
        # Création d'un groupe de RadioButton et d'un layout
        group_box1=QGroupBox("Images utilisées")
        vl_boutons1=QVBoxLayout()
        group_box1.setLayout(vl_boutons1)

        # Création des différentes options à choisir
        self.bouton1=QRadioButton("Originales sans contours")
        self.bouton2=QRadioButton("Originales contours blancs")
        self.bouton3=QRadioButton("Originales contours colorés")
        self.bouton4=QRadioButton("Seuillées et grains séparés")
        self.bouton5=QRadioButton("Carte de distance")

        # Connecter à la fonction qui change l'image
        self.bouton1.clicked.connect(self.changeImages)
        self.bouton2.clicked.connect(self.changeImages)
        self.bouton3.clicked.connect(self.changeImages)
        self.bouton4.clicked.connect(self.changeImages)
        self.bouton5.clicked.connect(self.changeImages)

        # Initialisation du bouton correspondant à l'image initiale
        self.bouton1.setChecked(True)

        # Ajout des boutons au layout
        vl_boutons1.addWidget(self.bouton1)
        vl_boutons1.addWidget(self.bouton2)
        vl_boutons1.addWidget(self.bouton3)
        vl_boutons1.addWidget(self.bouton4)
        vl_boutons1.addWidget(self.bouton5)




        """
        Informations sur le grain cliqué et renvoi vers la fenêtre de
trajectoire
        """

        ## Création d'un groupe contenant les informatins du grain cliqué
et d'un layout
        group_infos=QGroupBox("Informations sur le grain cliqué")
        vl_grain=QVBoxLayout()
        group_infos.setLayout(vl_grain)

        # Récupération des valeurs d'accélération et vitesse moyenne
        try :
```

```
            moyenne = np.load( self.objParams.genererURLInfos() )
        except FileNotFoundError :
            print( "[Erreur Bienvenue] Fichier introuvable : " +
self.objParams.genererURLInfos() )
            moyenne = [0, 0, 0, 0]

        # Création de labels qui affichent les valeurs du grain cliqué
        self.label_grain_X=QLabel("X : ")
        self.label_grain_Y=QLabel("Y : ")
        self.label_grain_Z=QLabel("Z : ")
        self.label_grain_Temps=QLabel("Temps : ")
################
        self.label_grain_volume = QLabel("Volume : ")
        self.label_grain_vitesse = QLabel("Vitesse grain : ")
        self.label_grain_vitessemoy = QLabel("Vitesse moyenne : " +
str(round(moyenne[0],2)) + " px/t")#moyenne[0]))
        self.label_grain_accel = QLabel("Accélération grain : ")
        self.label_grain_accelmoy = QLabel("Accélération moyenne : " +
str(round(moyenne[1],2)) + " px/t²")#moyenne[1]))


        # Ajout des labels dans le layout
        vl_grain.addWidget(self.label_grain_X)
        vl_grain.addWidget(self.label_grain_Y)
        vl_grain.addWidget(self.label_grain_Z)
        vl_grain.addWidget(self.label_grain_Temps)
################
        vl_grain.addWidget(self.label_grain_volume)
        vl_grain.addWidget(self.label_grain_vitesse)
        vl_grain.addWidget(self.label_grain_vitessemoy)
        vl_grain.addWidget(self.label_grain_accel)
        vl_grain.addWidget(self.label_grain_accelmoy)

        # Création d'un contenant pour les Radiobutton, la modification des
        # images et les informations du grain cliqué ; Ajout dans l'onglet
        contenant=QVBoxLayout()
        self.addLayout(contenant,1,5)

        # Ajout des widgets dans le contenant
        contenant.addWidget(group_box1)
        contenant.addSpacing(50)
        contenant.addWidget(group_infos)




        """
        Création de deux fenêtres qu'on affiche en cliquant sur un grain
        """
        # Création de la fenêtre et d'un layout
        self.fenetre_graph = QWidget()
        self.fenetre_graph.setWindowTitle("Trajectoire du grain cliqué")
        self.layout_traj_sepa=QGridLayout()
        self.fenetre_graph.setLayout(self.layout_traj_sepa)

        # Création de la figure 3D et insertion dans le Layout
        self.graphique3D = Graphique3D()
        self.layout_traj_sepa.addWidget(self.graphique3D)

        self.fenetre_vtk = QWidget()
        self.fenetre_vtk.setWindowTitle("Grain cliqué en 3D")
```

```
        self.afficheLienVTK=AfficheLienVTK()
        self.fenetre_vtk.setLayout(self.afficheLienVTK)



        """
        Appel de la fonction qui gère les barres de scroll et les images
        """
        self.changeImages(0)



    """
    Obtenir la position du clic
    """
    def get_pixel(self, event, source_object=None):

        # rapports : x=80   y=80    z=250
        #    image xy :   240 x 240      Diviser par 3  ; Diviser par 3
        #    image yz :   240 x 500      Diviser par 3  ; Diviser par 2
        #    image xz :   240 x 500      Diviser par 3  ; Diviser par 2

        # Récupère les coordonnées du point cliqué
        if (source_object==self.label_image_xy):
            x=floor(event.pos().x()/3)
            y=floor(event.pos().y()/3)
            z=self.barreScrollAxeZ.value()
        elif (source_object==self.label_image_yz):
            y=80-floor(event.pos().x()/3)
            z=floor(event.pos().y()/2)
            x=self.barreScrollAxeX.value()
        elif (source_object==self.label_image_zx):
            x=floor(event.pos().x()/3)
            z=floor(event.pos().y()/2)
            y=self.barreScrollAxeY.value()
        temps=self.barreScrollTemps.value()

        # Change l'affichage des coordonnées du point cliqué
        self.label_grain_X.setText("X : " + str(x))
        self.label_grain_Y.setText("Y : " + str(y))
        self.label_grain_Z.setText("Z : " + str(z))
        self.label_grain_Temps.setText("Temps : " + str(temps))

        # Appeler retrouve_grain pour obtenir la liste des positions au
cours du temps
        # et le volume du grain cliqué (seulement si on est sous Linux)
        ######## retour[0] = volume du grain
        ######## retour[1] = liste dont on a besoin
        ######## retour[2] = vitesse
        ######## retour[3] = accélération
        ######## retour[4] = vitesse moyenne
        ######## retour[5] = acceleration moyenne
        ######## retour[6] = lien VTK grain
        if systemPlatform() == "Linux" :
            retour = retrouve_grain(x,y,z,temps)
        else :
            retour = 0
```

```
        # Afficher et actualiser le graphique de la trajectoire
        self.fenetre_graph.show()


        if (retour==0):
            volume_grain = 0
            vitesse_grain = 0
            acceleration_grain = 0
            if self.objParams.tabGraphique3D != None :
                self.graphique3D.dessinerGraphique3D(
np.array([[[],[],[]]]), 1, 0,
limites=self.objParams.tabGraphique3D.graphique3D.getLimitesGraphe() ) #
Affiche un graphe vide
            else :
                self.graphique3D.dessinerGraphique3D(
np.array([[[],[],[]]]), 1, 0, conserverLimites = False )
        else :
            self.fenetre_vtk.show()
            self.afficheLienVTK.AfficherNouveauVTK(lienVTK=retour[6])
            volume_grain = retour[0]
            vitesse_grain = retour[2]
            acceleration_grain = retour[3]
            if self.objParams.tabGraphique3D != None :
                self.graphique3D.dessinerGraphique3D( [retour[1]], 1, 0,
limites=self.objParams.tabGraphique3D.graphique3D.getLimitesGraphe() ) #
Affiche la trajectoire du grain sélectionné
            else :
                self.graphique3D.dessinerGraphique3D( [retour[1]], 1, 0,
conserverLimites = False )

        # Changement de la valeur des labels relatives au grain
        self.label_grain_volume.setText("Volume :
"+str(round(volume_grain,2))+ " px³")
        self.label_grain_vitesse.setText("Vitesse grain :
"+str(round(vitesse_grain,2)) +" px/t")
        self.label_grain_accel.setText("Accélération grain :
"+str(round(acceleration_grain,2))+ " px/t²")



    """
    Gère l'affichage et son actualisation par les barres de scroll et le
choix
    de l'image à afficher
    """
    def changeImages(self, value) :
        # Image plan (X, Y)
        if (self.bouton1.isChecked()):
            image_xy = self.objParams.genererURLdesPGM3D( 'XY',
self.barreScrollTemps.value(), self.barreScrollAxeZ.value(),
typeDeTraitement = "originales")
        elif (self.bouton2.isChecked()):
            image_xy = self.objParams.genererURLdesPGM3D( 'XY',
self.barreScrollTemps.value(), self.barreScrollAxeZ.value(),
typeDeTraitement = "contours_blancs" )
        elif (self.bouton3.isChecked()):
            image_xy = self.objParams.genererURLdesPGM3D( 'XY',
self.barreScrollTemps.value(), self.barreScrollAxeZ.value(),
typeDeTraitement = "contours_rouges" )
        elif (self.bouton4.isChecked()):
```

```
            image_xy = self.objParams.genererURLdesPGM3D( 'XY',
self.barreScrollTemps.value(), self.barreScrollAxeZ.value(),
typeDeTraitement = "water" )
        elif (self.bouton5.isChecked()):
            image_xy = self.objParams.genererURLdesPGM3D( 'XY',
self.barreScrollTemps.value(), self.barreScrollAxeZ.value(),
typeDeTraitement = "carte_dist" )
        if os.path.isfile( image_xy ) : # Si le chemin d'accès à l'image
existe
            width=self.label_image_xy.width()
            height=self.label_image_xy.height()

self.label_image_xy.setPixmap(QPixmap(image_xy).scaled(width,height,Qt.Keep
AspectRatio))
        else :
            print( "[Erreur TabAffichageCoupes] " + image_xy + " n'existe
pas !" )


        # Image plan (Y, Z)
        if (self.bouton1.isChecked()):
            image_yz = self.objParams.genererURLdesPGM3D( 'YZ',
self.barreScrollTemps.value(), self.barreScrollAxeX.value(),
typeDeTraitement = "originales")
        elif (self.bouton2.isChecked()):
            image_yz = self.objParams.genererURLdesPGM3D( 'YZ',
self.barreScrollTemps.value(), self.barreScrollAxeX.value(),
typeDeTraitement = "contours_blancs" )
        elif (self.bouton3.isChecked()):
            image_yz = self.objParams.genererURLdesPGM3D( 'YZ',
self.barreScrollTemps.value(), self.barreScrollAxeX.value(),
typeDeTraitement = "contours_rouges" )
        elif (self.bouton4.isChecked()):
            image_yz = self.objParams.genererURLdesPGM3D( 'YZ',
self.barreScrollTemps.value(), self.barreScrollAxeX.value(),
typeDeTraitement = "water" )
        elif (self.bouton5.isChecked()):
            image_yz = self.objParams.genererURLdesPGM3D( 'YZ',
self.barreScrollTemps.value(), self.barreScrollAxeX.value(),
typeDeTraitement = "carte_dist" )
        if os.path.isfile( image_yz ) : # Si le chemin d'accès à l'image
existe
            width=self.label_image_yz.width()
            height=self.label_image_yz.height()

self.label_image_yz.setPixmap(QPixmap(image_yz).scaled(width,height,Qt.Keep
AspectRatio))
        else :
            print( "[Erreur TabAffichageCoupes] " + image_yz + " n'existe
pas !" )


        # Image plan (X, Z)
        if (self.bouton1.isChecked()):
            image_zx = self.objParams.genererURLdesPGM3D( 'XZ',
self.barreScrollTemps.value(), self.barreScrollAxeY.value(),
typeDeTraitement = "originales")
        elif (self.bouton2.isChecked()):
            image_zx = self.objParams.genererURLdesPGM3D( 'XZ',
self.barreScrollTemps.value(), self.barreScrollAxeY.value(),
typeDeTraitement = "contours_blancs" )
```

```
        elif (self.bouton3.isChecked()):
            image_zx = self.objParams.genererURLdesPGM3D( 'XZ',
self.barreScrollTemps.value(), self.barreScrollAxeY.value(),
typeDeTraitement = "contours_rouges" )
        elif (self.bouton4.isChecked()):
            image_zx = self.objParams.genererURLdesPGM3D( 'XZ',
self.barreScrollTemps.value(), self.barreScrollAxeY.value(),
typeDeTraitement = "water" )
        elif (self.bouton5.isChecked()):
            image_zx = self.objParams.genererURLdesPGM3D( 'XZ',
self.barreScrollTemps.value(), self.barreScrollAxeY.value(),
typeDeTraitement = "carte_dist" )

        if os.path.isfile( image_zx ) : # Si le chemin d'accès à l'image
existe
            width=self.label_image_zx.width()
            height=self.label_image_zx.height()

self.label_image_zx.setPixmap(QPixmap(image_zx).scaled(width,height,Qt.Keep
AspectRatio))
        else :
            print( "[Erreur TabAffichageCoupes] " + image_zx + " n'existe
pas !" )


        # Changement de la valeur des labels des axes
        self.valeur_temps.setText("Temps : " +
str(self.barreScrollTemps.value()))
        self.valeur_X.setText("X : " + str(self.barreScrollAxeX.value()))
        self.valeur_Y.setText("Y : " + str(self.barreScrollAxeY.value()))
        self.valeur_Z.setText("Z : " + str(self.barreScrollAxeZ.value()))



"""
Code principal pour démonstration
"""
# Si on est le script principal
# Cela permet de ne pas exécuter ce bloc de codes lorsque ce script est
importé par un autre
# Source : https://stackoverflow.com/questions/419163/what-does-if-name-
main-do
if __name__ == '__main__' :
    application = QApplication(sys.argv) # Crée un objet de type
QApplication (Doit être fait avant la fenêtre)
    fenetre = QWidget() # Crée un objet de type QWidget
    fenetre.setWindowTitle("MODE DÉMONSTRATION") # Définit le nom de la
fenêtre
    fenetre.setLayout( TabAffichageCoupes( Parametres() ) )
    fenetre.show() # Affiche la fenêtre
    application.exec_() # Attendre que tout ce qui est en cours soit
exécuté
```