gui_unifiée/functions_main.py

```
import os
import sys

from PyQt5.QtWidgets import QMessageBox, QFileDialog

# Source : https://www.geeksforgeeks.org/understanding-python-pickling-example/
import pickle # Permet de sauvegarder et charger des variables
# On n'utilise plus shelve, car il n'est pas cross-platform

from platform import system as systemPlatform

from class_Parametres import Parametres

from class_Fenetre import Fenetre

if systemPlatform() == "Linux" :
    sys.path.append("../extraction")
    from traitement_3D import traitement_3D_main


"""
Fonction de traitement d'image (Appel tout le travail de traitement d'image)
@param fichier : Fichier TIFF choisi par l'utilisateur
"""
def traitementImage( fichier, supprimerLesBords =  False ) :
    dossierActuel = os.path.dirname(__file__)
    dossierDuTraitement = os.path.abspath( os.path.join(dossierActuel.replace("\\", "/"), "../extraction") )
    os.chdir( dossierDuTraitement )
    print( "[Info Main] Nouveau WD : " + dossierDuTraitement )
    print( "[Info Main] LANCEMENT DU TRAITEMENT !" )
    fichierExporte = traitement_3D_main( fichier, supprimerBords = supprimerLesBords )
    print( "[Info Main] FIN DU TRAITEMENT !" )
    os.chdir( dossierActuel )
    print( "[Info Main] Retour au WD : " + dossierActuel )
    return fichierExporte


"""
Importer un fichier exporté par le système de traitement
@param fichier : Fichier d'exportation du traitement
@return False si ça a merdé, ou si les données ont des incohérences
@author Amaury
"""
def importerTraitement( fichier, objParams ) :
    print( "[Info Main] Importation du traitement : " + fichier )
    try :
        fichierExporté = open(fichier, 'rb')
    except Exception :
        print( "[Erreur Main] Fichier invalide !" )
        return False

    sys.path.append(fichierExporté)
    try :
        bdd = pickle.load(fichierExporté)
```

```
    except Exception :
        print( "[Erreur Main] Fichier corrompu !" )
        return False

    try :
        objParams.NB_IMGS = bdd[ "NB_IMGS" ]
    except KeyError :
        print( "[Erreur Main] Le fichier ne contient pas la variables
NB_IMGS !" )
        return False
    try :
        objParams.INTERVALLE_XY = bdd[ "INTERVALLE_XY" ]
    except KeyError :
        print( "[Erreur Main] Le fichier ne contient pas la variables
INTERVALLE_XY !" )
        return False
    try :
        objParams.INTERVALLE_XZ = bdd[ "INTERVALLE_XZ" ]
    except KeyError :
        print( "[Erreur Main] Le fichier ne contient pas la variables
INTERVALLE_XZ !" )
        return False
    try :
        objParams.INTERVALLE_YZ = bdd[ "INTERVALLE_YZ" ]
    except KeyError :
        print( "[Erreur Main] Le fichier ne contient pas la variables
INTERVALLE_YZ !" )
        return False
    try :
        objParams.URL_PGM = bdd[ "URL_PGM" ]
    except KeyError :
        print( "[Erreur Main] Le fichier ne contient pas la variables
URL_PGM !" )
        return False
    try :
        objParams.URL_VTK = bdd[ "URL_VTK" ]
    except KeyError :
        print( "[Erreur Main] Le fichier ne contient pas la variables
URL_VTK !" )
        return False
    try :
        objParams.URL_GRAPHIQUE_3D = bdd[ "URL_GRAPHIQUE_3D" ]
    except KeyError :
        print( "[Erreur Main] Le fichier ne contient pas la variables
URL_GRAPHIQUE_3D !" )
        return False
    try :
        objParams.URL_VITESSE_MOY_GRAINS = bdd[ "URL_VITESSE_MOY_GRAINS" ]
    except KeyError :
        print( "[Erreur Main] Le fichier ne contient pas la variables
URL_VITESSE_MOY_GRAINS !" )
        return False

    objParams.contientVariablesImportees = True

    # Sauvegarde du répertoire absolu du répertorie du fichier
d'exportation
    # Sert à localiser à partir des URL relatives qu'il contient
    objParams.CHEMIN_ABSOLU_FICHIER_IMPORTE =
os.path.dirname(os.path.abspath(fichier))
```

```
    fichierExporté.close()

    return objParams.verifierParams()


"""
Demande à l'utilisateur un fichier pour lancer un traitement ou ouvrir un
fichier de traitement
@param lancer : True pour lancer un traitement, False pour ouvrir un
traitement
@author Amaury
"""
def lancerOuOuvrirTraitement( lancer, application ) :
    print( "[Info Main] Actuel WD : " + os.path.dirname(__file__) )

    fileDialog = QFileDialog() # Crée un objet de type QFileDialog (Fenêtre
pour choisir un fichier)
    if lancer : fileDialog.setWindowTitle("Veuillez choisir le fichier à
traiter") # Définit le nom de la fenêtre
    else : fileDialog.setWindowTitle("Veuillez choisir le fichier à
importer")
    fichierDemande = fileDialog.getOpenFileName()[0] # Permet aussi
d'attendre qu'il y ait un fichier demandé
    print( "[Info Main] Fichier demandé : " + fichierDemande )
    fileDialog.close() # Fermer la fenêtre

    creationObjParams = Parametres() # Permet de passer aux onglets les
paramètres chargés

    if lancer :
        msgBox = QMessageBox()
        msgBox.setText("Voulez-vous couper aux bords ?\nAttention, les
premiéres coupes seront vides.")
        msgBox.setWindowTitle("Information")
        msgBox.setStandardButtons(QMessageBox.Yes | QMessageBox.No)
        returnValue = msgBox.exec()

        if returnValue == QMessageBox.Yes :
            fichierExporte = traitementImage( fichierDemande,
supprimerLesBords = True )
        else :
            fichierExporte = traitementImage( fichierDemande,
supprimerLesBords = False )

        autorisationDeLancer = importerTraitement( fichierExporte,
creationObjParams )
    else :
        autorisationDeLancer = importerTraitement( fichierDemande,
creationObjParams )

    if autorisationDeLancer :
        fenetre = Fenetre( objParams = creationObjParams ) # Crée un objet
de type Fenetre
        fenetre.setWindowTitle("WoopWoop 3D") # Définit le nom de la
fenêtre
        fenetre.show() # Affiche la fenêtre
        application.exec_() # Attendre que tout ce qui est en cours soit
exécuté
    else :
        QMessageBox.about(None, "Information", "Fichier inutilisable !")
```