Compte rendu des modifications apportées

uHDR v7

Informations sur le document

Auteurs

Laurane Mouronval
Gauthier Corion

Historique des versions

Auteur	Date	Description	Version
Laurane Mouronval	11/06/2024	Création du document	1.0
Laurane Mouronval Gauthier Corion	14/06/2024	Relecture du document	1.1

But du document

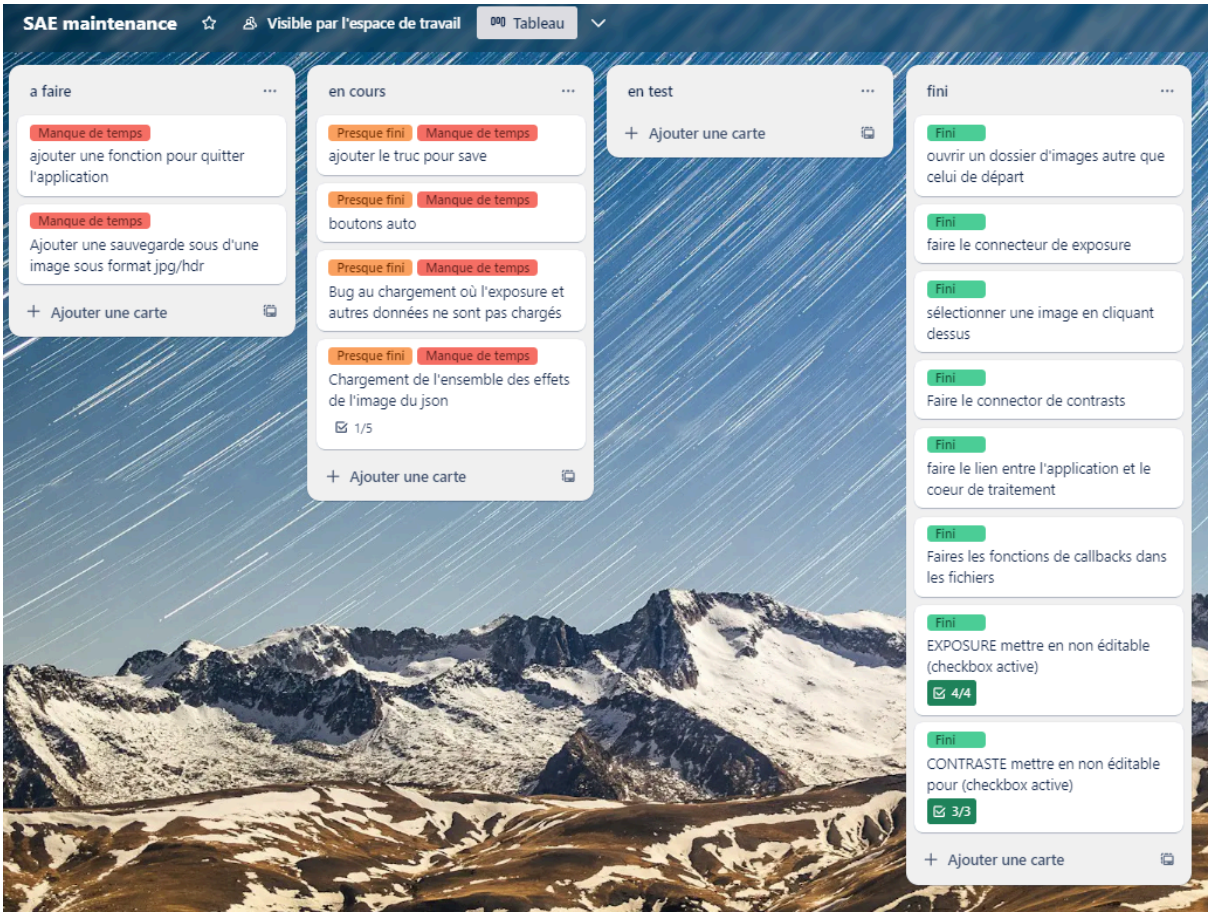
Ce document a pour but de répertorier toutes les modifications apportées à la version 7 de l'application uHDR ainsi que toutes les fonctionnalités manquantes.

Lien vers le projet

https://github.com/MrBaguette07/UHDR

Répartition du travail

Vous pouvez retrouver sur le planning en temps réel ce que chacun a fait pendant ce projet. Nous avons utilisé un trello pour s'organiser dans les tâches restantes à faire. Nous avons utilisé Github pour le versionning. Et enfin nous avons utilisé liveshare pour pouvoir modifier à deux le code en même temps pour gagner en temps et en efficacité.



Modifications réalisées

Nous allons présenter les principales modifications dans un ordre plutôt chronologique.

Correction d'erreur typographique

"selction" au lieu de "Selection" dans les onglets color

Ajout de commentaires de documentation des méthodes en anglais

Nous avons commenté toutes les fonctions et les modifications que nous avons apportées à l'application.
Exemple : voir fichiers **/core/**image.py, **/app/**app.y ImageFiles.py, **/guiQt/**mainwindow.py contrast.py curveWidget.py editor.py editorBlock.py IchSelector.py LightBlock.py,

LightBlockScroll.py colorEditor.py colorEditorBlock.py advanceSlider.py AdvanceSliderLine.py ChannelSelector.py

Exemple de commentaires en anglais dans **/app/ImageFiles.py** :

```python
class ImageFiles(QObject):

    def saveProcesspipe(self, namefile: str, dico: dict) -> bool | None:
        """
        Get the 'processpipe' list in the given JSON file
        and change the values to then write it back to the file to save the changes

        Args:
            namefile (str, Required)
            dico (dict, Required)

        Returns:
            (bool, Required): bool
        """

        path, name, ext = filenamesplit(namefile)
        if ext != 'json':
            namefile = f"{name}.json"

        filepath = os.path.join(self.imagePath, namefile)

        if not os.path.isfile(filepath):
            print(f"File not found: {filepath}")
            return None
        try:
            with open(filepath, 'r') as json_file:
                file = json.load(json_file)
            file['processpipe'] = dico
            with open(filepath, 'w') as json_file:
                json.dump(file, json_file)

        except Exception as e:
            print(f"Error reading {namefile}: {e}")
            return False
```

Modification du fichier image.py

Pour l'importation de dossier d'image hdr, il était nécessaire de modifier le fichier image.py, en ajoutant une condition pour traiter l'ouverture et l'écriture des fichier dont l'extension est "hdr".

```python
def write(self: Image, fileName: str):
    """write image to system."""

    if self.hdr:
        colour.write_image(self.cData, fileName, bit_depth='float32', method='Imageio')
    else:
        colour.write_image((self.cData * 255.0).astype(np.uint8), fileName, bit_depth='uint8', method='Imageio')

    # Debugging: Output image min/max values
    print(f"Image written to {fileName} with min/max values: {np.min(self.cData)}, {np.max(self.cData)}")
```

```python
def read(fileName : str) -> Image:
    """read image from system."""
    img : Image
    path, name, ext = filenamesplit(fileName)
    if os.path.exists(fileName):
        if ext == "jpg":
            imgData :  np.ndarray = colour.read_image(fileName, bit_depth='float32', method= 'Imageio')
            img = Image(imgData, ColorSpace.sRGB, False, True, name)
        if ext == "hdr":
            imgData :  np.ndarray = colour.read_image(fileName, bit_depth='float32', method= 'Imageio')
            img = Image(imgData, ColorSpace.sRGB, True, True, name)
    else:
        img = Image(np.ones((600,800,3))*0.50, ColorSpace.sRGB, False, True, name)
    return img
```

Ajout des Connecteurs

Étape 1 : Création des signaux

Fichiers concernés dans le dossier GuiQt :

- Contrast
- advanceSlider
- advanceSliderLine
- lchSelector
- ChannelSelector

Exemple avec la class Contrast :
Nous commençons par déclarer les signaux qui seront appliqués dans tous les fichiers utilisés pour transférer les signaux sauf app.py.

```python
class Contrast(QFrame):
    # Declaration of signals
    scalingChanged = pyqtSignal(float)
    offsetChanged = pyqtSignal(float)
    lightnessRangeChanged = pyqtSignal(tuple)
    activeContrastChanged = pyqtSignal(bool)

    loadJsonChanged: pyqtSignal = pyqtSignal(list)

    def __init__(self: Self) -> None:
```

1. Changevalue(value): permet de Set le slider ou autres parammètres venant de app.py pour notamment charger les informations déjà présentes dans les fichiers JSON

2. onActive…Changed(self: self) : Permet d'envoyer et de gérer le système d'activation et de désactivation des blocs (checkbox "active")

```python
class Contrast(QFrame):
        self.checkBoxActive.stateChanged.connect(self.onActiveContrastChanged)

        # self.loadJsonChanged.connect(self.changeValue)

    def changeValue(self, value: list):
        """
        Manage the activation/desactivation system of the element contrast

        Args :
            value (list, Required)
        """
        self.scalingSlider.setValue(value[1]['contrast']['contrast'])

    def onScalingChanged(self, str: str, value: float):
        value = float(value)
        self.scaling = value
        self.scalingChanged.emit(value)

    def onOffsetChanged(self, str: str, value: float):
        value = float(value)
        self.offset = value
        self.offsetChanged.emit(value)

    def CBlightnessselectionChanged(self: Self) -> None:
        self.LightnessRange = self.lightnessSelector.getValues()
        self.lightnessRangeChanged.emit(self.LightnessRange)
        self.updateView()

    def onActiveContrastChanged(self: Self) -> None:
        """
        Manage the activation/desactivation system of the element contrast
        """
        if self.active == True:
            self.active = False
        else:
            self.active = True

        self.scalingSlider.slider.setEnabled(self.active)
        self.activeContrastChanged.emit(self.active)
```

Étape 2 : Création des fonction de callback

Fichiers concernés dans le dossier guiQt:

- lightblock
- editorblock
- coloreditorblock

Création des callbacks des évènements, curve, contrast, exposure, etc…

```
# Connect signals from LightBlock to LightBlockScroll
self.light.exposure.valueChanged.connect(self.onExposureChanged)
self.light.contrast.scalingChanged.connect(self.onContrastScalingChanged)
self.light.contrast.activeContrastChanged.connect(self.onActiveContrastChanged)
self.light.activeExposureChanged.connect(self.onActiveExposureChanged)
# self.light.autoClickedExposure.connect(self.autoClickedExposure)
# self.light.contrast.loadJsonChanged.emit(self.onLoadJsonChanged)

self.light.contrast.offsetChanged.connect(self.onContrastOffsetChanged)
self.light.contrast.lightnessRangeChanged.connect(self.onLightnessRangeChanged)
self.light.curve.highlightsChanged.connect(self.onHighlightsChanged)
self.light.curve.shadowsChanged.connect(self.onShadowsChanged)
self.light.curve.whitesChanged.connect(self.onWhitesChanged)
self.light.curve.blacksChanged.connect(self.onBlacksChanged)
self.light.curve.mediumsChanged.connect(self.onMediumsChanged)
self.light.curve.activeLightnessChanged.connect(self.onActiveLightnessChanged)
```

Création des fonctions de callback pour envoyer les différents signaux :

```
class LightBlockScroll(QScrollArea):
    def __init__(self : Self) -> None:
        self.light.curve.activeLightnessChanged.connect(self.onActiveLightnessChanged)

    def onActiveLightnessChanged(self, value: bool):
        """
        Returns the signal of the activation state of lightness

        Args :
            value (bool, Required)
        """
        self.activeLightnessChanged.emit(value)

    def onLoadJsonChanged(self, value: list):
        """
        Returning the signal after the loading of a json file

        Args :
            value (list, Required)
        """
        self.loadJsonChanged.emit(value)

    def onExposureChanged(self, value: float):
        """
        Returns the signal when the slider of Exposure change

        Args :
            value (float, Required)
        """
        self.exposureChanged.emit(value)
```

Étape 3 : Ajout des callbacks dans l'application

Fichier concerné dans le dossier guiQt :

- MainWindow

Ajouts de tous les callbacks de l'application pour les transférer dans app.py

```python
class MainWindow(QMainWindow):
    # Declaration of signals
    dirSelected = pyqtSignal(str)
    requestImages = pyqtSignal(int, int)
    imageSelected = pyqtSignal(int)
    tagChanged = pyqtSignal(tuple, bool)
    scoreChanged = pyqtSignal(int)
    scoreSelectionChanged = pyqtSignal(list)

    exposureChanged = pyqtSignal(float)
    contrastScalingChanged = pyqtSignal(float)
    contrastOffsetChanged = pyqtSignal(float)
    lightnessRangeChanged = pyqtSignal(tuple)
    hueShiftChanged = pyqtSignal(float, int)
    saturationChanged = pyqtSignal(float, int)
    colorExposureChanged = pyqtSignal(float, int)
    colorContrastChanged = pyqtSignal(float, int)
    highlightsChanged = pyqtSignal(float)
    shadowsChanged = pyqtSignal(float)
    whitesChanged = pyqtSignal(float)
    blacksChanged = pyqtSignal(float)
    mediumsChanged = pyqtSignal(float)

    hueRangeChanged = pyqtSignal(tuple, int)
    chromaRangeChanged = pyqtSignal(tuple, int)
    lightness2RangeChanged = pyqtSignal(tuple, int)

    activeContrastChanged = pyqtSignal(bool)
    activeExposureChanged = pyqtSignal(bool)
    activeLightnessChanged = pyqtSignal(bool)
    activeColorsChanged = pyqtSignal(bool, int)

    loadJsonChanged: pyqtSignal = pyqtSignal(list)
```

```python
class MainWindow(QMainWindow):
    def __init__(self: MainWindow, nbImages: int = 0, tags : dict[Tuple[str,str], bool] = {}) -> None:
        self.setCentralWidget(self.imageGallery)

        ## menu
        self.buildFileMenu()

        ## callbacks
        ### from AdvanceImageGallery
        self.imageGallery.requestImages.connect(self.CBrequestImages)
        self.imageGallery.imageSelected.connect(self.CBimageSelected)
        self.metaBlock.tagChanged.connect(self.CBtagChanged)
        self.metaBlock.scoreChanged.connect(self.CBscoreChanged)
        self.metaBlock.scoreSelectionChanged.connect(self.CBscoreSelectionChanged)

        ### from EditorBlock
        self.editBlock.exposureChanged.connect(self.exposureChanged)
        self.editBlock.contrastScalingChanged.connect(self.contrastScalingChanged)
        self.editBlock.contrastOffsetChanged.connect(self.contrastOffsetChanged)
        self.editBlock.lightnessRangeChanged.connect(self.lightnessRangeChanged)
        self.editBlock.hueShiftChanged.connect(self.hueShiftChanged)
        self.editBlock.saturationChanged.connect(self.saturationChanged)
        self.editBlock.colorExposureChanged.connect(self.colorExposureChanged)
        self.editBlock.colorContrastChanged.connect(self.colorContrastChanged)
        self.editBlock.highlightsChanged.connect(self.highlightsChanged)
        self.editBlock.shadowsChanged.connect(self.shadowsChanged)
        self.editBlock.whitesChanged.connect(self.whitesChanged)
        self.editBlock.blacksChanged.connect(self.blacksChanged)
        self.editBlock.mediumsChanged.connect(self.mediumsChanged)

        self.editBlock.hueRangeChanged.connect(self.hueRangeChanged)
        self.editBlock.chromaRangeChanged.connect(self.chromaRangeChanged)
        self.editBlock.lightness2RangeChanged.connect(self.lightness2RangeChanged)

        self.editBlock.activeContrastChanged.connect(self.activeContrastChanged)
        self.editBlock.activeExposureChanged.connect(self.activeExposureChanged)
        self.editBlock.activeLightnessChanged.connect(self.activeLightnessChanged)
        self.editBlock.activeColorsChanged.connect(self.activeColorsChanged)
```

Étape 4 : Ajout du coeur de l'application

Fichier concerné dans le dossier hdrCore:

- processing

Le cœur de l'application a été intégralement repris de la v6 et adapté pour la v7.

D'abord, on transforme tous les "colorspace et colordata" en cSpace et cData, car dans le fichier image.py, ce sont les noms des attributs. Notamment parce que nous passons l'image en paramètres dans les fonctions de processing.

```
215             if (img.type == image.imageType.HDR):
216
217                 # encode
218 -               imgRGBprime =
    colour.cctf_encoding(res.colorData,function=function)
219
220                 # update attributes
221 -               res.colorData       = imgRGBprime
222                 res.type            = image.imageType.SDR
223                 res.linear          = False
224                 res.scalingFactor   = 1.0
225 -               res.colorSpace      =
    colour.models.RGB_COLOURSPACES[function].copy()
226
227             return res
228     # ----------------------------------------------------------------
```

```
216             if (img.type == image.imageType.HDR):
217
218                 # encode
219 +               imgRGBprime =
    colour.cctf_encoding(res.cData,function=function)
220
221                 # update attributes
222 +               res.cData       = imgRGBprime
223                 res.type            = image.imageType.SDR
224                 res.linear          = False
225                 res.scalingFactor   = 1.0
226 +               res.cSpace      =
    colour.models.RGB_COLOURSPACES[function].copy()
227
228             return res
229     # ----------------------------------------------------------------
```

Puis, on enlève tous les “cumba”, “numba”, car nous n’en avons plus besoin pour la V7

Enfin, on enlève toutes les intégrations de prefs

```
374             if contrastValue != defaultContrast:
375                 # contrast scaling is computed in prime colorspace
376                 if img.linear:
377 -                   if pref.computation == 'python':
378 -                       start = timer()
379 -                       res.colorData = colour.cctf_encoding(res.colorData, function='sRGB') # encode to prime
380 -                       res.linear =        False
381 -
382 -                   elif pref.computation == 'numba':
383 -                       start = timer()
384 -                       res.colorData = numbafun.numba_cctf_sRGB_encoding(res.colorData) # encode to prime
385 -                       res.linear =        False
386 -
387 -                   elif pref.computation == 'cuda':
388 -                       start = timer()
389 -                       res.colorData = numbafun.cuda_cctf_sRGB_encoding(res.colorData) # encode to prime
390 -                       res.linear =        False
```

```
364             if contrastValue != defaultContrast:
365                 # contrast scaling is computed in prime colorspace
366                 if img.linear:
367 +                   start = timer()
368 +                   res.cData =     colour.cctf_encoding(res.cData, function='sRGB') # encode to prime
369 +                   res.linear =        False
```

```
@@ -1249,28 +1206,28 @@ def getImage(self,toneMap=True):
1249                     self.__outputImage.colorData = colour.cctf_encoding(self.__outputImage.colorData, function='sRGB')
1250                     self.__outputImage.linear =  False
1251
1252 -                   if pref.verbose: print(" [PROCESS] >> ProcessPipe.getImage(",self.__outputImage.name,", toneMap:",toneMap,"): encode to sRGB !")
1253
1254                 elif self.__outputImage.isHDR() and self.__outputImage.linear and toneMap:
1255                     self.__outputImage.colorData = colour.cctf_encoding(self.__outputImage.colorData, function='sRGB')
1256                     self.__outputImage.linear =  False
1257
1258 -                   if pref.verbose: print(" [PROCESS] >> ProcessPipe.getImage(",self.__outputImage.name,", ,toneMap:",toneMap,"): tone map using cctf encoding !")
1259
1260                 elif self.__outputImage.isHDR() and (not self.__outputImage.linear) and (not toneMap):
1261                     self.__outputImage.colorData =
```

```
1206                     self.__outputImage.colorData = colour.cctf_encoding(self.__outputImage.colorData, function='sRGB')
1207                     self.__outputImage.linear =  False
1208
1209 +                   # print(" [PROCESS] >> ProcessPipe.getImage(",self.__outputImage.name,", toneMap:",toneMap,"): encode to sRGB !")
1210
1211                 elif self.__outputImage.isHDR() and self.__outputImage.linear and toneMap:
1212                     self.__outputImage.colorData = colour.cctf_encoding(self.__outputImage.colorData, function='sRGB')
1213                     self.__outputImage.linear =  False
1214
1215 +                   # print(" [PROCESS] >> ProcessPipe.getImage(",self.__outputImage.name,", ,toneMap:",toneMap,"): tone map using cctf encoding !")
1216
1217                 elif self.__outputImage.isHDR() and (not self.__outputImage.linear) and (not toneMap):
1218                     self.__outputImage.colorData =
```

Étape 5 : Créations des fonctions de callbacks dans le controller principal de l’application

Fichier concerné dans le dossier app:

- app

L’ensemble des callbacks reçu pour les changements dans l’interface :

```
class App:
    def __init__(self: App) -> None:
        ## ----------------------------------------------------

        self.mainWindow : MainWindow = MainWindow(nbImages, self.tags.toGUI())
        self.mainWindow.showMaximized()
        self.mainWindow.show()

        ## callbacks
        self.mainWindow.dirSelected.connect(self.CBdirSelected)
        self.mainWindow.requestImages.connect(self.CBrequestImages)
        self.mainWindow.imageSelected.connect(self.CBimageSelected)

        self.mainWindow.tagChanged.connect(self.CBtagChanged)
        self.mainWindow.scoreChanged.connect(self.CBscoreChanged)
        self.mainWindow.scoreSelectionChanged.connect(self.CBscoreSelectionChanged)

        self.mainWindow.exposureChanged.connect(self.onExposureChanged)
        self.mainWindow.contrastScalingChanged.connect(self.onContrastScalingChanged)

        # Offset is not used for the moment, as the function in the core is not defined yet
        # self.mainWindow.contrastOffsetChanged.connect(self.onContrastOffsetChanged)

        self.mainWindow.lightnessRangeChanged.connect(self.onLightnessRangeChanged)
        self.mainWindow.hueShiftChanged.connect(self.onHueShiftChanged)
        self.mainWindow.saturationChanged.connect(self.onSaturationChanged)
        self.mainWindow.colorExposureChanged.connect(self.onColorExposureChanged)
        self.mainWindow.colorContrastChanged.connect(self.onColorContrastChanged)
        self.mainWindow.highlightsChanged.connect(self.onHighlightsChanged)
        self.mainWindow.shadowsChanged.connect(self.onShadowsChanged)
        self.mainWindow.whitesChanged.connect(self.onWhitesChanged)
        self.mainWindow.blacksChanged.connect(self.onBlacksChanged)
        self.mainWindow.mediumsChanged.connect(self.onMediumsChanged)

        self.mainWindow.hueRangeChanged.connect(self.onHueRangeChanged)
        self.mainWindow.chromaRangeChanged.connect(self.onChromaRangeChanged)
        self.mainWindow.lightness2RangeChanged.connect(self.onLightness2RangeChanged)

        self.mainWindow.activeContrastChanged.connect(self.onActiveContrastChanged)
        self.mainWindow.activeExposureChanged.connect(self.onActiveExposureChanged)
        self.mainWindow.activeLightnessChanged.connect(self.onActiveLightnessChanged)
        self.mainWindow.activeColorsChanged.connect(self.onActiveColorsChanged)
```

Modification de la fonction CBimageSelected

1. Changement des informations de l’image actuelle :

metaImage : Les metadata de processpipe actuellement appliqué sur l’image

originalMeta : Les metadata avant les modifications actuels (pour les boutons active notamment)

saveMeta : Pour le sauvegarde des metadata avant l’activation/désactivation

2. Envoi des signaux pour changer la valeur au chargement. Tous les signaux n’ont pas été effectué par manque de temps, mais la base est là, il faut “simplement” l’appliqué à toute l’interface.

3. La liste des composants actuellement désactivé / activé pour l’image actuellement sélectionnée

```python
class App:
    self.originalImages[filename].setMetadata(self.imagesManagement.getProcesspipe(filename))

    #### image selected
    #### ------------------------------------------------------------------
    def CBimageSelected(self: App, index):


        self.selectedImageIdx = index # index in selection

        gIdx : int | None= self.selectionMap.selectedlIndexToGlobalIndex(index)# global index

        if (gIdx != None):

            image : ndarray = self.imagesManagement.getImage(self.imagesManagement.getImagesFilesnames()[gIdx])
            tags : Tags = self.imagesManagement.getImageTags(self.imagesManagement.getImagesFilesnames()[gIdx])
            exif : dict[str,str] = self.imagesManagement.getImageExif(self.imagesManagement.getImagesFilesnames()[gIdx])
            score : int = self.imagesManagement.getImageScore(self.imagesManagement.getImagesFilesnames()[gIdx])

            self.mainWindow.setEditorImage(image)

            # update image info
            imageFilename : str =  self.imagesManagement.getImagesFilesnames()[gIdx]
            imagePath : str =  self.imagesManagement.imagePath
            self.processPipe = self.buildProcessPipe()
            #### if debug : print(f'App.CBimageSelected({index}) > path:{imagePath}')
            self.metaImage = self.imagesManagement.getProcesspipe(imageFilename)
            self.originalMeta = self.imagesManagement.getProcesspipe(imageFilename)
            self.saveMeta = self.imagesManagement.getProcesspipe(imageFilename)

            # self.mainWindow.loadJsonChanged.emit(self.metaImage)
            if self.metaImage:
                self.mainWindow.editBlock.edit.lightEdit.light.contrast.changeValue(self.metaImage)
                self.mainWindow.editBlock.edit.lightEdit.light.exposure.changeValue(self.metaImage)

            self.disabledContent = [{'exposure': True},{'contrast': True},{'lightness': True},[{'0': True},{'1': True},{'2': True},{'3': True},{'4': True}]]


            self.mainWindow.setInfo(imageFilename, imagePath, *Jexif.toTuple(exif))

            self.mainWindow.setScore(score)

            # update tags info
            self.mainWindow.resetTags()
            if tags:
                self.mainWindow.setTagsImage(tags.toGUI())
```

UpdateImage : Permet de modifier l'image dans la gallery et dans le selector. De plus il met à jour les metadata de processpipe, et sauvegarde l'image

```python
class App:
    def updateImage(self, imageName: str, new_image: Image) -> None:
        """ xx
        Update the image in ImageFiles and refresh the GUI.

        Args:
            imageName (str, required)
            new_image (Image, required)
        """
        imageIdx = self.selectionMap.imageNameToSelectedIndex(imageName)
        if imageIdx is not None:
            self.mainWindow.setGalleryImage(imageIdx, new_image.cData)
            if self.selectedImageIdx == imageIdx:
                self.mainWindow.setEditorImage(new_image.cData)
        self.metaImage = new_image.metadata
        self.originalImages[imageName].setMetadata(self.metaImage)
        self.imagesManagement.saveProcesspipe(imageName,self.metaImage)
```

applyProcessing : Permet de générer l'image avec le fichier dll fournit.

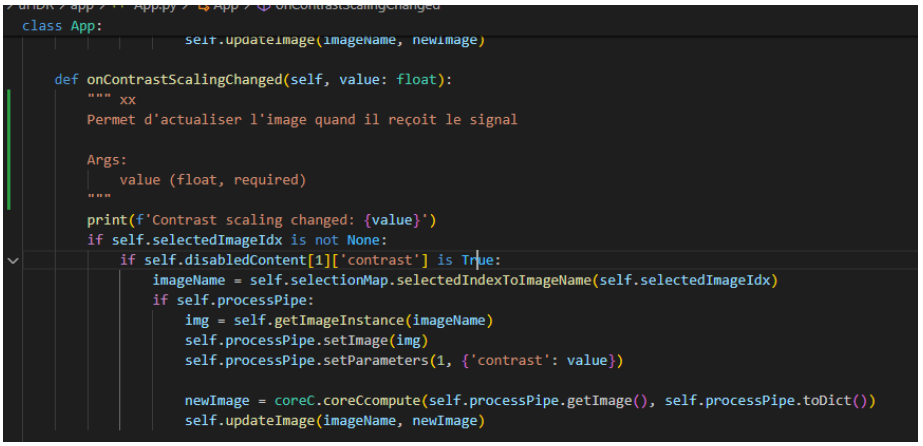
```python
class App:
    def applyProcessing(self, img: Image, processPipe: dict) -> Image:
        Get an Image instance from image name.

        Args:
            imageName (str, required)
        """
        return coreC.coreCcompute(img, processPipe)

    def onExposureChanged(self, value: float):
```

Par manque de temps la qualité de code n'est pas optimale pour les fonction changed : il y a une fonction par événement, il aurait été préférable de faire une fonction globale ou deux suivant les types de signaux (changed d'une valeur, activation/désactivation ou show selection)

on__signalName__Changed : Permet de mettre à jour l'image en fonction du signal reçu. Si c'est un changement de "contrast" (voir image) il vient d'abord vérifier si l'image est désactivée ou non, sinon, on vient appliquer l'image actuelle dans le processpipe et modifier la valeur "contrast" par la valeur du signal. Et ensuite on vient appliquer les modifications.



```
class App:
                    self.updateImage(imageName, newImage)

    def onContrastScalingChanged(self, value: float):
        """ xx
        Permet d'actualiser l'image quand il reçoit le signal

        Args:
            value (float, required)
        """
        print(f'Contrast scaling changed: {value}')
        if self.selectedImageIdx is not None:
            if self.disabledContent[1]['contrast'] is True:
                imageName = self.selectionMap.selectedIndexToImageName(self.selectedImageIdx)
                if self.processPipe:
                    img = self.getImageInstance(imageName)
                    self.processPipe.setImage(img)
                    self.processPipe.setParameters(1, {'contrast': value})

                    newImage = coreC.coreCcompute(self.processPipe.getImage(), self.processPipe.toDict())
                    self.updateImage(imageName, newImage)
```

De même que **on__signalName__Changed**, le mieux aurait été de faire une fonction globale, mais nous n'avons pas eu le temps.

Cette fonction, comme toutes les fonctions **onActive__signalName__Changed**, permet d'activer un bloc ou non, s'il est activé et qu'il devient désactivé, alors on sauvegarde les metadata data dans **self.savemeta** et on vient appliquer les paramètres par défaut. Et, à l'inverse, si on réactive le bloc, on applique les metadata sauvegardés

```
class App:
    def onActiveContrastChanged(self, value: bool):
        """ xx
        Permet d'actualiser l'image quand il reçoit le signal d'activation ou désactivation

        Args:
            value (bool, required)
        """
        print(f'Active contrast changed: {value}')
        if self.originalMeta is not None:
            imageName = self.selectionMap.selectedIndexToImageName(self.selectedImageIdx)
            if self.saveMeta is None:
                tempSave = self.imagesManagement.getProcesspipe(imageName)
                self.saveMeta[1]['contrast']['contrast'] = tempSave[1]['contrast']['contrast']

            img = self.getImageInstance(imageName)
            self.processPipe.setImage(img)
            if value == True:
                self.processPipe.setParameters(1, {'contrast': self.saveMeta[1]['contrast']['contrast']})
            if value == False:
                self.saveMeta[1]['contrast']['contrast'] = self.metaImage[1]['contrast']['contrast']
                self.processPipe.setParameters(1, {'contrast': self.originalMeta[1]['contrast']['contrast']})

            self.disabledContent[1]['contrast'] = value
            newImage = coreC.coreCcompute(self.processPipe.getImage(), self.processPipe.toDict())
            self.updateImage(imageName, newImage)
```

buildProcessPipe: nous l'avons ajouté afin de générer les processing par défaut venant de la première version de l'application uHDR V6

```python
@staticmethod
def buildProcessPipe():
    """
    WARNING:
        here the process-pipe is built
        initial pipe does not have input image
        initial pipe has processes node according to EditImageView
    """
    processPipe = processing.ProcessPipe()

    # exposure ------------------------------------------------------------------------------------------------------------ 0
    defaultParameterEV = {'EV': 0}
    idExposureProcessNode = processPipe.append(processing.exposure(), paramDict=None,name="exposure")
    processPipe.setParameters(idExposureProcessNode, defaultParameterEV)

    # contrast ------------------------------------------------------------------------------------------------------------ 1
    defaultParameterContrast = {'contrast': 0}
    idContrastProcessNode = processPipe.append(processing.contrast(), paramDict=None,  name="contrast")
    processPipe.setParameters(idContrastProcessNode, defaultParameterContrast)

    #tonecurve ----------------------------------------------------------------------------------------------------------- 2
    defaultParameterYcurve = {'start':[0,0],
                              'shadows': [10,10],
                              'blacks': [30,30],
                              'mediums': [50,50],
                              'whites': [70,70],
                              'highlights': [90,90],
                              'end': [100,100]}
    idYcurveProcessNode = processPipe.append(processing.Ycurve(), paramDict=None,name="tonecurve")
    processPipe.setParameters(idYcurveProcessNode, defaultParameterYcurve)

    # masklightness ------------------------------------------------------------------------------------------------------- 3
    defaultMask = { 'shadows': False,
                    'blacks': False,
                    'mediums': False,
                    'whites': False,
                    'highlights': False}
    idLightnessMaskProcessNode = processPipe.append(processing.lightnessMask(), paramDict=None, name="lightnessmask")
    processPipe.setParameters(idLightnessMaskProcessNode, defaultMask)

    # saturation ---------------------------------------------------------------------------------------------------------- 4
    defaultValue = {'saturation': 0.0,  'method': 'gamma'}
    idSaturationProcessNode = processPipe.append(processing.saturation(), paramDict=None, name="saturation")
    processPipe.setParameters(idSaturationProcessNode, defaultValue)

    # colorEditor0 -------------------------------------------------------------------------------------------------------- 5
    defaultParameterColorEditor0= {'selection': {'lightness': (0,100),'chroma': (0,100),'hue':(0,360)},
                                   'edit': {'hue': 0.0, 'exposure':0.0, 'contrast':0.0,'saturation':0.0},
                                   'mask': False}
    idColorEditor0ProcessNode = processPipe.append(processing.colorEditor(), paramDict=None, name="colorEditor0")
    processPipe.setParameters(idColorEditor0ProcessNode, defaultParameterColorEditor0)

    # colorEditor1 -------------------------------------------------------------------------------------------------------- 6
    defaultParameterColorEditor1= {'selection': {'lightness': (0,100),'chroma': (0,100),'hue':(0,360)},
                                   'edit': {'hue': 0.0, 'exposure':0.0, 'contrast':0.0,'saturation':0.0},
                                   'mask': False}
    idColorEditor1ProcessNode = processPipe.append(processing.colorEditor(), paramDict=None, name="colorEditor1")
    processPipe.setParameters(idColorEditor1ProcessNode, defaultParameterColorEditor1)

    # colorEditor2 -------------------------------------------------------------------------------------------------------- 7
    defaultParameterColorEditor2= {'selection': {'lightness': (0,100),'chroma': (0,100),'hue':(0,360)},
                                   'edit': {'hue': 0.0, 'exposure':0.0, 'contrast':0.0,'saturation':0.0},
```

Fonctionnalités manquantes

Ce sont les différences de fonctionnalités manquante dans la v7 par rapport à la v6

- Lecture d'images raw
- Paramètre offset dans la partie Contraste
- La partie avec les checkbox "mask lightness"
- La courbe bleue sur le graphe
- Le bouton "auto color selection"
- Les information complémentaires sur l'image (exposure time, f-number, ISO, camera, software, lens, focal length)
- Les informations sur Aesthetics (color palette, process output, number of colors)
- Les options Display HDR (Display HDR image, Compare raw and edited HDR image, reset HDR display), Export HDR Image et Preferences
- La partie avec cropping adj. et rotation
- La partie avec hdr preview

Modifications restantes à réaliser

Ce sont les éléments que nous aurions aimé terminer dans la v7 par rapport à ce qui était demandé.

- refactoring du code pour améliorer sa qualité

Voir le trello pour plus de détails.

Autre remarque :

- Lorsqu'un utilisateur utilise l'application, la modification du json en simultanée n'est pas prise en compte et elle est écrasée.

En résumé, nous n'avons pas eu le temps de faire tous les changements mais nous avons compris que la documentation des classes n'était pas suffisante car nous travaillons sur une application complexe.