CODE DOCUMENTATION FOR DEVELOPERS

The application has been developed with Python 3.8 and PyQT5 for the GUI development.

We used PyQt Widgets instead of QtQuick and QML. QtDesigner application was used for designing the UI.

The development was done in a virtual environment with PyCharm. It has been tested on Windows 10 and Ubuntu 20.04. It works with WSL2 when an X-server application, like Vcxsrv is used for GUI forwarding.

External Libraries and Packages used:

This is all there in the requirements.txt file.

But just to reiterate, if you want to download different versions of the package directly without using the same versions we did.

1. Numpy

This was for manipulating the image. The image was converted to a numpy array to introduce the noise in it.

2. PyQT5

This is what we MAINLY used for the Graphical User Interface (GUI) Application Development

3. Pyqt5-tools

This is the tool that gives access to the QtDesigner application which we used for creating the GUI for this Visual Snow Syndrome Diagnostic application.

4. OpenCV-python

This is again for manipulating the image. Blur, Brightness, resizing the image, all of this is possible because of OpenCV.

File Organization Structure:

Python Files :

Python files are not stored in any folder. They are directly in the project directory.

1. VisualSnowSyndrome_Diagnostic.py

This is the main application python file. Contents of this file discussed below in the Code Organization Structure section.

2. resources.py

This file uses the Qt Resource system. The icons, images and stylesheets associated with the application were loaded into a .qrc file and this was converted into a python file.

This file is imported by VisualSnowSyndrome_Diagnostic.py.

This is used so that hardcoded file paths to an image or icon is not necessary. Especially since this application has been developed for Linux, and the files are separated by '\' unlike Windows, this Qt Resource system proves really useful.

Other files :

1. README.md

This contains the installation instructions and general stuff about the application.

2. LICENSE

MIT License is used which is pretty flexible.

3. resources.qrc

This file is an XML file. All of the icons, images, and the stylesheets' hardcoded path is specified here. The alias which is set in this file is what is used in the main VisualSnowSyndrome_Diagnostic.py when calling the icons, images, and stylesheet. This file was converted into resources.py using a script so that it can be imported into the main application.

4. requirements.txt

This contains all of the package dependencies and libraries and modules used for the development of this application. For developers, according to the OS you are on, you can just 'pip install requirements.txt' to install all the dependencies of the application and start developing.

Folders :

1. images

This folder contains 2 images. The eye chart which can be used and a test image too. However, any image as long as it is a .jpg or .png image, can be selected by the user.

- eye_chart.png
- test_image.jpg

2. icons

This folder contains all of the icons used in the application. It also contains the application's main icon. These are all loaded into the Qt Resource system however.

Fun Fact: We made the icon ourselves but to add the Visual Snow Syndrome and blurry effect to the icon, we used our own application!

3. UI

This folder contains everything related to the UI of the application.

- Darkeum.qss

  This is the QSS stylesheet used to make the application theme blue and black. This was selected from the templates available in:

  https://qss-stock.devsecstudio.com/

- MainWindow.ui

  This is the file which is written in XML. This file is automatically generated when the configuration in QtDesigner is saved. This same file can be opened with QtDesigner to make any changes.

- vertical_blur.py

  The MainWindow.ui file was converted into a python file using a command. This is the file which was copy pasted into the main application python file 'VisualSnowSyndrome_Diagnostic.py' and edited.

- horizontal.py

  This has a horizontal layout for the sliders. This is not currently used in the application, we had just tested it out.

4. tests

This folder is currently empty. However, we will be adding the files after unittesting and static testing the application.

5. documentation

This folder contains 4 files.

- The user manual shows the application and how to use it for new users. It is definitely recommended to go through this before installing the application so users know what to expect.
- Developer_Installation.md contains the instructions for installing on your respective OS and also some ideas of what we have to contribute.
- VSS Progress contains what we initially did. It's pretty nice to see how far we've come!
- And, the code documentation, i.e. this file.

6. .idea

This basically comes with PyCharm. And, there's a .gitignore which just ignores whatever we want it to when pushing to GitHub.

Code Organization Structure:

There are comments made to the code wherever possible so that contributors can understand why a particular line was coded and its importance. However to describe the overall functionality and the importance of each function in a class, the below documentation is provided.

VisualSnowSyndrome_Diagnostic.py

We implemented the Model View Controller (MVC) architecture to organize the code of our application.

There are 2 classes in the Python file.

1. Class Ui_MainWindow

This contains most of the content from MainWindow.ui. Basically this class contains everything designed by QtDesigner and it describes the UI of the file.

In this class we added code related to:

- Signals and slots connections:

  Some are connected to built in functions and most of the functions are coded. The code for these functions is in the next class. Basically every widget/item which requires a user input uses a signals and slots mechanism. All of these connections are defined in this class.
- Icons
- Declaring global variables
- Setting the stylesheet

2. Class MainWindow

This uses the previous class.

The functions described here grouped:

Category 1: Open, Set, Save, and Remove an Image.

- loadImage

  Basically just opens a filedialog for you to select an image.
  - conditions

    This describes all the edge cases for each action a user can do so that the application doesn't crash.

- setPhoto

  Takes an image and sets it as a Pixmap. Every time the image is manipulated, (either blur, noise, etc.) the image has to be set. Also this function resizes the image so that it will fit any application.

  Both of these functions (setPhoto and eventFilter) are connected to the widget_2 (i.e. the image widget's) size.

- eventFilter

  This is the function called by setPhoto function. This function is for the case that whenever an image is resized, i.e. reducing the size of the application window, or going fullscreen.

- savePhoto

  This function is for opening a file dialog for saving the image.

  - file_save

    I don't remember what this does, I guess we can remove it.
  - test_cases

    The edge cases so that the application doesn't crash no matter what the user does.
  - save_conditions

    This function is for saving the text file which contains the configuration of the image.

- removePhoto

  This function removes the Photo when the user makes a mistake and selects a wrong button contradictory to what they had originally clicked. This function is ONLY connected to the error popups.

Category 2: Error Popups

- popup_critical

  A critical MessageBox is shown.
- popup_warning

  A warning MessageBox is shown.

Category 3: Fullscreen

- switchFullScreen

  Basically hiding all of the other Widgets other than the image widget.

  Since the image widget is set as expanding in the UI class, the image widget will expand to how much ever and thus the user ends up seeing a bigger image with all of the sliders and buttons hidden.

- mouseDoubleClickEvent

  If you double click on the image, it becomes fullscreen. Double clicking again escapes full screen.

- exit_full

  This function is basically for the menubar.

- full_view

  This function is basically for the menubar.

Category 4: Image manipulation

- update function

  Invokes the setPhoto function every time after an image is manipulated i.e. when any of the following functions changeBlur, changeBrightness, or salt_pepper_noise is called.

- *Subcategory 1:* Blur
  - blur_value

    This is for the signals and slots connections to the slider. Takes in the value from the slider.

  - changeBlur

    This is what uses OpenCV built in function for blurring the image according to the value obtained from the slider.

- *Subcategory 2:* Brightness
  - brightness_value

    This is for the signals and slots connections to the slider. Takes in the value from the slider.

  - changeBrightness

    This is what uses OpenCV functions to simulate brightness. This also takes in the value obtained from the slider.

- *Subcategory 3:* Noise
  - noise_value

    This is for the signals and slots connections to the slider. Takes in the value from the slider.

  - salt_pepper_noise

    Numpy is used to add black and white spots randomly to the image. There is a separate variable for black and a separate variable for white so anything, salt or pepper can be removed in the code.

Category 5: Close Application

- closeEvent

This is a built in function that is customized it to our requirements. This is so that if the user accidentally clicks the exit button, a popup shows us to confirm their action.

Category 6: Menubar

- web_link

  This function is connected to the signals and slots of the user manual and information tab in the Help section of the menubar. It redirects the user to the Github repository.