*The code can be found in the folder scr. This is a non-exhaustive and summarised explanation of what happens on the main scripts. The goal of this explanation is to have a general view of what can be found on each part of the code, not to explain with detail what each function or script does. To understand more about how the program works, please refer to the manuscript.*

It all starts by running the script \_\_main\_\_.py. This script calls viewer.py, which is the first script to be used. It has 2 important classes: **PALMS and Viewer**. PALMS is the real main class of the program. All the global variables are defined here in the constructor and init functions.

In the init function we also initialise the data from the file to be used. This can be done by getting an existing file (this just happens when we open a file that we had already opened before). Or if we want to load a new file, the function request\_user\_input\_database\_and\_file is called. This opens the QDialog defined in the **MainImport** class, which is responsible for letting the user choose the data. On top of this dialog we find a menu with the 4 import options. But there are really just 3 important options, as the last one just lets the user split and existing file, but not load it.

The management of all these options is done in the function changeContent. We can see that the 3 import options call the class ImportData. This class manages the components that we will see in the QDialog by calling at 3 other classes. We have Options, which has different labels, spaces and combo boxes with the idea of letting the user choose the specifics they want to get from the file. The second class is DataPreview, where the specifications of the user are collected and applied to the given file to return the actual data that the user is asking for. And the last of these classes is SignalPreview, with the same idea as DataPreview but in this case showing the plot instead of just the tabular data. In these 3 functions we can see different implementations for different data formats, but at the end we are just reading files with the specifications and showing what we are reading.

Once all the specifications are set and the import button clicked, the MainImport class will get the final data and send it to PALMS. For this we have different functions and computations done depending on the file format we have, if we are loading a previous analysis or a new file, loading RR or ECG data, …

The fourth option in the QDialog calls the function SplitFiles instead of ImportData. We can see this class is quite similar to the others but with small changes. It also contains the components Options, DataPreview and SinglePreview.

Once the data is loaded, the interface itself is created with PALMS and Viewer. During **PALMS** construction many other classes were initialized and these are called now. In particular we are calling the database, Modes and the components of the interface. At the end of the after\_input function we see the loop that defines the screen will be visible until the exit\_code is found.

From this initialization, it is important to explain what the Databases, operation modes, annotations and peak detection are doing. The idea is that the Database stores the important annotation data. It creates the start and end indexes and calls the function set\_annotation\_data. This is the function where the peaks are detected and the annotations saved by using the function \_set\_annotation\_from\_idx.

On the other hand, the operation modes (script operation\_mode.py) represent the state in which the graphs in the interface will be. Depending on the state they will react differently to the action done on the graph, depending on whether the mode is annotation (for outliers), partition (for samples) or noise partition (for noise).

The class **Viewer** acts as the front-end of PALMS. It is really where the components are placed. We find the menus with a function for each of the options and the function guiAddPanel. In this function the ECG and RR graphs are created by calling the functions createNewButtonPanel (this creates the options we see on top to edit peaks and noise), createNewOutliersPanel (this creates the outlier selector under the previous edit options) and createNewPanel (this creates the ECG/RR graph by calling functions from DisplayPanel and setting the layouts and sliders).

Apart from the classes that define the signal visualization (DisplayPanel mainly with ButtonPanel and OutliersPanel), Viewer also calls the LeftOptionsPanel and ResultsPanel. These 5 panel classes follow similar implementations. Each is defined inside a Frame class that lets the Panel (which is just a widget) be placed in the main screen as the layout of the Frame.

The classes **ButtonPanel and OutlierPanel** are quite simple. They just have the buttons with the basic functionalities calling functions from other classes. The first just activates the proper Mode, which will define what will happen when touching the graph (peak, noise or segment editing). The second class is connected to LeftOptionsPanel and initializes or updates the visual representations of the outliers in these panels.

But the actual changes are implemented in the function **LeftOptionsPanel**. This class has some of the data management functionalities of the program. The actual values are saved in PALMS so they are more accessible also for other classes. But the work done with them happens in this class. Mainly the idea is to manage the 3 annotated variables: noise, outliers and segments. But other things this class does are computing and exporting results, and saving or loading the analysis.

For outliers, the main function is outlier\_decision\_central. This does the outlier detection and correction and then updates the values were necessary. This means updating the text and annotations in the DisplayPanel, modifying all the necessary PALMS variables and calling the PALMS functions that update the graphs, as it is necessary to change the RR plot and reset the annotations. For the outlier detection and correction itself, the functions outlier\_decision\_algorithm and outlier\_threshold might be used depending on whether the user has chosen them. Also, the input to these functions is determined by the user.

For samples and noise, we have the management options here. Although both things are objects of another class. These classes, located in the folder logic/operation\_mode is in the scripts partitioning.py, noise\_partitioning.py and rr\_noise\_partitioning.py. They 3 follow mainly the same idea of having a list of LinearRegionItems that are shown in the DisplayPanel and represent noise or segments. Each has functions to manage individual items by adding, moving or deleting. But there are also collective functions for things like adding or deleting many, getting all the names and starting points, or finding a specific item in the list.

In the **LeftOptionsPanel**, we mainly have the connection between the interface and these scripts. We show the buttons to add, remove or edit noise and samples and we call the specific functions based on what was chosen. Then we display the new things when necessary. For noise in this case we also have a detection function that works as an algorithm. It works similarly to the outlier detection functions, doing the detection and then modifying the DisplayPanel and PALMS variables.

To work with noise we just have the main button in LeftOptionsPanel and the selector in OutlierPanel. But choosing samples is a bit more complex, as we need to define time, repetitions and some other specifications. For this we have a particular Widget that we add to the LeftOptionsPanel block which is called SampleOptions. This follow a similar idea of the Options in the data import. We have some labels, spaces and drowdown where the user can set the specifications. Like there can be different sample blocks, there are also options to create many and selectors to move from one to the other. We see in this class functions to add samples based on start or point and to check whether the sample is valid or too noisy and should be skipped.

Almost at the end of this LeftOptionsPanel we also see the exportResults function. This just calls the functions to compute the results from results\_panel.py with the specific requirements (specified values, outliers, noise and samples) and fills and output csv file with the proper format. There is also an option to append the results in a previous file, which works similarly to the other one.

And at the very end of the script, we find the functions to save the analysis and load it again. This basically collects all the important information from the analysis (settings, outliers, noise, samples and some other variables) and saves all in an h5 file. The loading function simply gets this same data from the existing file and reloads PALMS to start again with the new data.

Next to the LeftOptionPanel we find the **DisplayPanel**. Here the ECG and RR graphs are created and displayed. The idea is to call the class **PlotArea**, which is a GraphicsView. We can see implemented a custom axis and some util functions that let us plot vertical lines or specific points. This happens because each graph we have is synchronised with a ViewBox (from PyQtGraph), which is interactive. Depending on the mode we are in and whether we do left or right-click, we will call a function to add a peak, delete it, create a sample, or any of the other options we have.

Under the DisplayPanel we have the **ResultsPanel**. This has few buttons that determine what should be shown on this frame. Depending on what button is active, one of the results functions will be called and its results will be displayed.

From the results section it is also important to note that we include a modified pyHRV in the modified\_dependencies folder. This has mainly the same code as the original pyHRV implementation, but we decided to change some details in the frequency variables computations to be more aligned to what is normally used in HRV analysis. The original implementation of pyHRV could also be used with no problem.