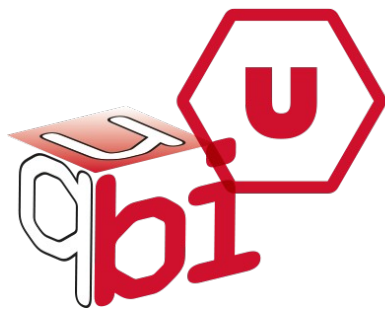


Internship Report Part I - Paweł Batóg

Research group - Description



QuBi

Quantitative Biolmaging

The Quantitative Biolmaging (QuBi) lab was formed in 2017 by Dr. Carlo Manzo (Ramón y Cajal Fellow) at the Faculty of Science and Technology of the UVic-UCC. It has been recognized as an emerging research group by the Government of Catalonia (2017SGR940)

The research group in Quantitative Biolmaging is focused to study molecular mechanisms underlying complex biological processes through advanced microscopy and image analysis techniques. The group aims at applying these technologies to characterize at the molecular level cellular signaling processes in health and disease, with the final objective of providing guidelines for the development of new approaches for nanodiagnostics and personalized medicine.

The group activity is carried out along three main lines of research, focused to the development of:

- i) localization microscopy techniques, including single particle tracking and super-resolution imaging
- ii) quantitative methods for the analysis and the treatment of super-resolution images and single-molecule data
- iii) stochastic approaches for the modeling of protein motion and organization

The research has a highly interdisciplinary character, bridging elements of physics, engineering and data science to tackle problems in biomedicine and cell biology.

The goal of apprenticeships

The goal of apprenticeship is to create software that will be able to simulate the behavior of proteins on previously selected conditions in a clear, simple and concise manner. This process requires not only knowledge about application programming, but also knowledge about the functioning of algorithms that simulate protein behavior and the selection of appropriate libraries that will satisfactorily meet their performance in future simulations.

Collecting the necessary knowledge

First step in preparing the application sketch was to familiarize with the algorithms provided by Professor Carlo Manzo. These algorithms were written in python and presented protein simulations.

A complete understanding of each function that occurs in its code is extremely important, because the application will be based on the these modified algorithms.

Into the code

The program contains two fundamental functions that will be the most important elements (from the point of protein simulation) of software development.

1) **Generate Molecules Function** - Function that is responsible for creating points (proteins)

```
def generate_molecules(pixels,n,min_d):
    xn=np.random.uniform(0,(pixels-1),n)
    yn=np.random.uniform(0,(pixels-1),n)

    z=np.stack((xn,yn), axis=-1)
    D = squareform(pdist(z)) + 1000*min_d*np.identity(n)

    while D.min()<min_d:
        ind1, ind2 = np.where(D < min_d)
        unique = (ind1 < ind2)
        ind1 = ind1[unique]

        xn[ind1]=xn[ind1]+np.random.normal(0,min_d,ind1.shape[-1])
        yn[ind1]=yn[ind1]+np.random.normal(0,min_d,ind1.shape[-1])
        z=np.stack((xn,yn), axis=-1)
        D = squareform(pdist(z)) + 1000*min_d*np.identity(n)

    return [xn,yn]
```

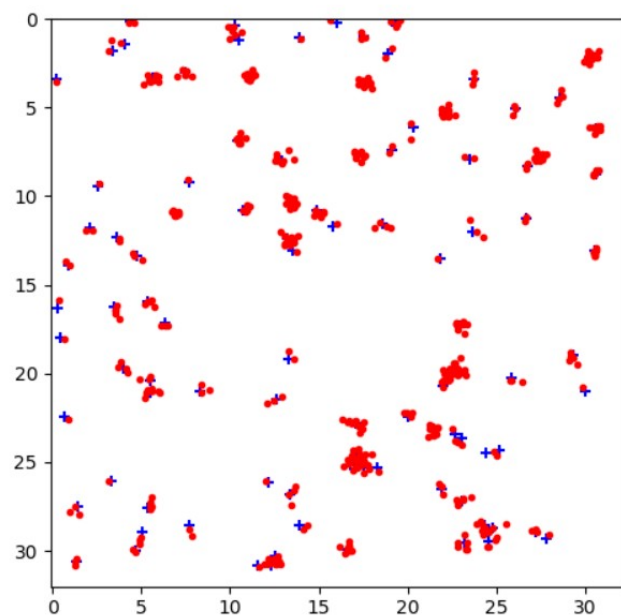
2) **Frame Simulation Function** - Function that plot the graph based on previously created points

```
def frame_simulation(pixels,xn,yn,stdev, disp_locs):  
  
    n=xn.shape[-1]  
    x=[]  
    y=[]  
  
    localization=np.random.geometric(p=0.2, size=n)  
  
    for i in range(0,n):  
        Xij = np.random.normal(loc=0,scale=stdev, size=int(localization[i]))  
        Yij = np.random.normal(loc=0,scale=stdev, size=int(localization[i]))  
        x.append(xn[i] + Xij)  
        y.append(yn[i] + Yij)  
  
    x_unchain= np.asarray(list(chain.from_iterable(x)))  
    y_unchain= np.asarray(list(chain.from_iterable(y)))  
  
    if disp_locs==1:  
        fig = plt.figure(1, figsize=(5.5,5.5))  
        plt.scatter(xn,yn,c='b',marker='+')  
        plt.scatter(x_unchain,y_unchain,c='r',marker='.')  
        plt.axis('equal')  
        plt.xlim((0,pixels))  
        plt.ylim((0,pixels))  
        plt.gca().invert_yaxis()  
        plt.show()  
  
    return [x_unchain,y_unchain]
```

The final result generated by these functions should look like below:

Blue crosses indicate the actual positions of molecules.

Red dots represent the place where the molecule is revealed.



Selection of library and programming language for building applications

The second step was to choose a programming language that will be suitable for creating the interface and that will also allow to implement the fundamental version of the source code.

Since the python programming language performs both in the role of calculations and in the creation of applications based on a graphical interface, it will work great with that issue.



When searching for the appropriate library for building the interface, I decided to use the **Python QT** library.



This library has not only extensive documentation, but also **QT - Designer** application, which allows for more efficient writing of the code responsible for the application interface.

Installation of appropriate libraries - Python QT

There are many ways to install the **Python - QT** library. In the case of the Windows 10 operating system on which I had the opportunity to create software, the installation is not complicated.

Assuming that the user who wants to install the **Python - QT** library already has basic **Python 3.x** library and **PIP** software that allows the installation of additional libraries all what is needed to do is to install two **PyQT** libraries using pip install function by Command Prompt Commands (CMD).

```
C:\Users\user>pip install PyQt5
```

- Installs the Python QT library

```
C:\Users\user>pip install PyQt5-tools
```

- Installs the Python QT tools (designer application)

After installing the libraries using **PIP**, **Python QT** should work without the need for additional library configuration. From now on, to be able to use the **Python QT** library in the application, all that is needed to do is to import the library in the header of the .py file

```
from PyQt5 import QtCore, QtGui, QtWidgets
```

The **designer.exe** application should be in a folder containing all Python libraries. In the case of Windows 10 this is:

```
C:\Python37\Lib\site-packages\pyqt5_tools
```

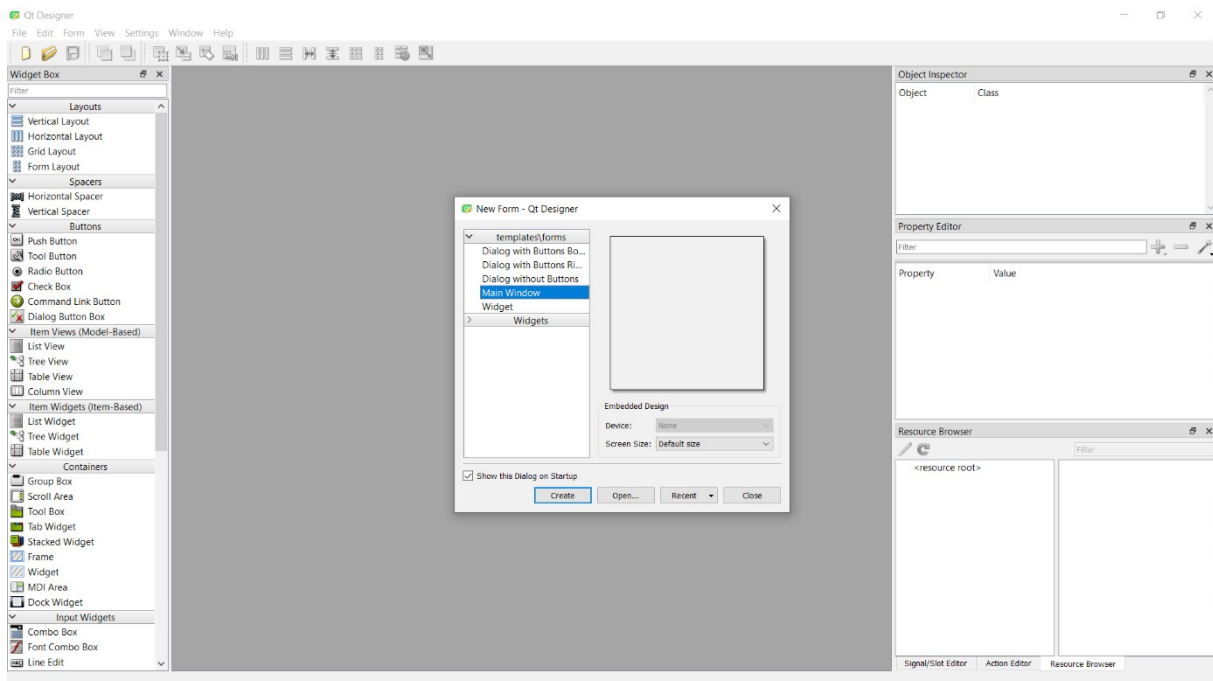
For more information on installing the library, look at the **Python QT** documentation, which contains a detailed description of software installation in various environments. Full **Python QT** documentation is included on the site:

<https://www.riverbankcomputing.com/static/Docs/PyQt5/>

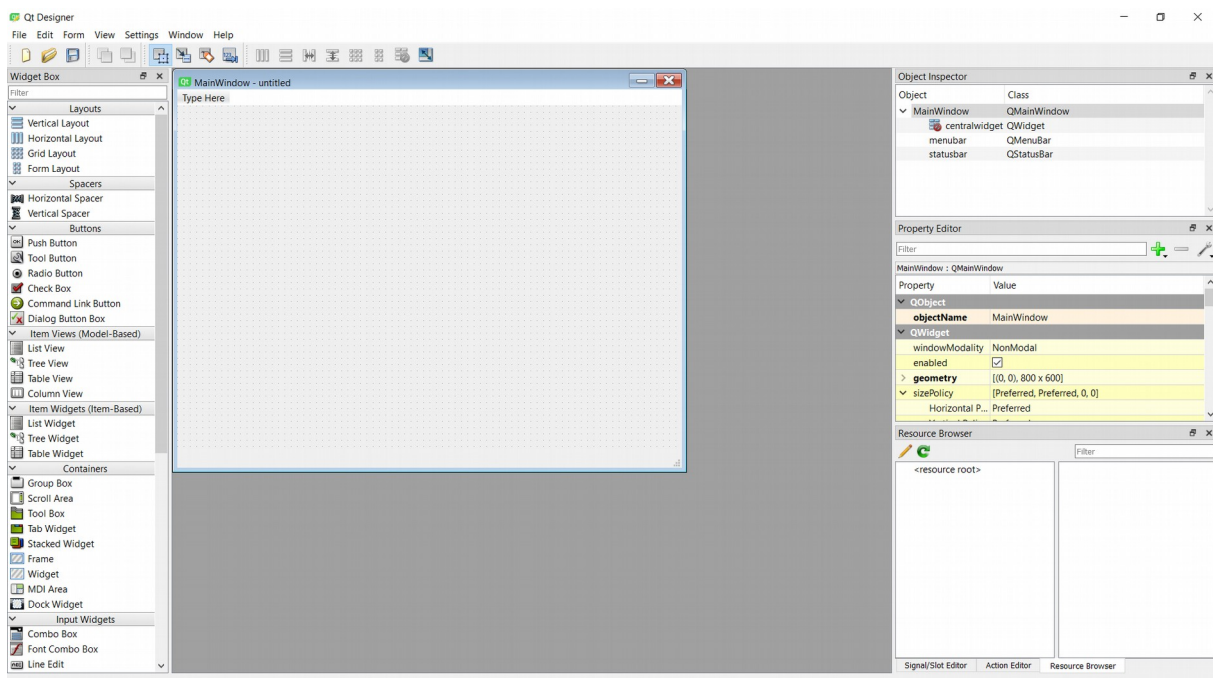
- Introduction
 - License
 - PyQt5 Components
- Contributing to this Documentation
 - Repository Structure
 - Description Files
 - Contributing Patches
- Platform Specific Issues
 - macOS
- Deprecated Features and Behaviours
- Incompatibilities with Earlier Versions
 - PyQt v5.12
 - PyQt v5.11
 - PyQt v5.6
 - PyQt v5.5
 - PyQt v5.3
- Installing PyQt5
 - Understanding the Correct Version to Install
 - Installing from Wheels
 - Building and Installing from Source
 - Installing PyQt3D, PyQtChart, PyQtDataVisualization and PyQtPurchasing
- Differences Between PyQt4 and PyQt5
 - Supported Python Versions
 - Deprecated Features
 - Multiple APIs
 - Old-style Signals and Slots
 - New-style Signals and Slots
 - QtDeclarative, QtScript and QtScriptTools Modules

Python QT - Environment description

After opening the program, there will be a window in which user should choose whether to create a main window or widget. In the case of an application containing more than one window, it has to be remembered that any other window than the main window will be the widget of our



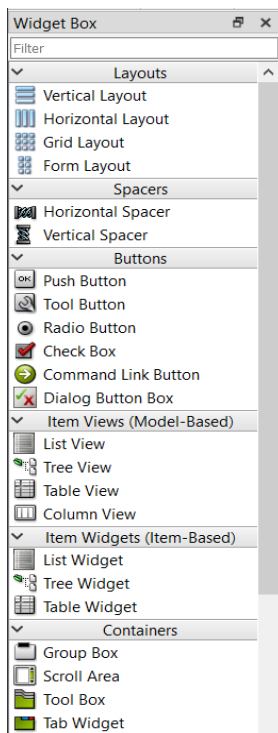
application.



Widget Box

The "Designer" application provides a large set of functions to help you create a graphical interface. The most important element are in the window located on the left side of the program - "Widget Box" in which there are all available widgets such as layouts, spacers, buttons, containers and much more...

In the creation of our application mainly used widgets are:



Grid Layout - Layout that is responsible for the correct placement of widgets in the application,

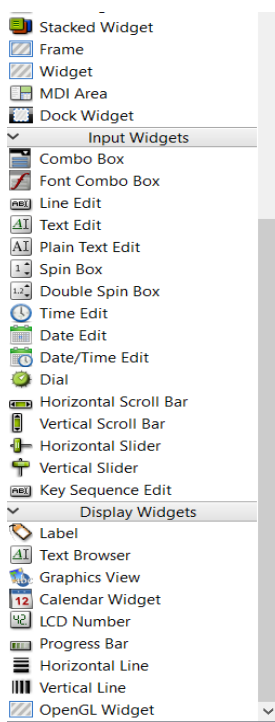
Horizontal / Vertical Spacers - A widget that allows to create and maintain a fixed or dynamic distance between application widgets,

Buttons -> Push Buttons - Buttons that allow the application user to interact with it,

Containers -> Frames - Widget that allows to create an individual window in the main window. This can be understood as a new submain window in the main window,

Input Widgets -> Text Edit - The field in which the application user can enter input data such as numbers or words,

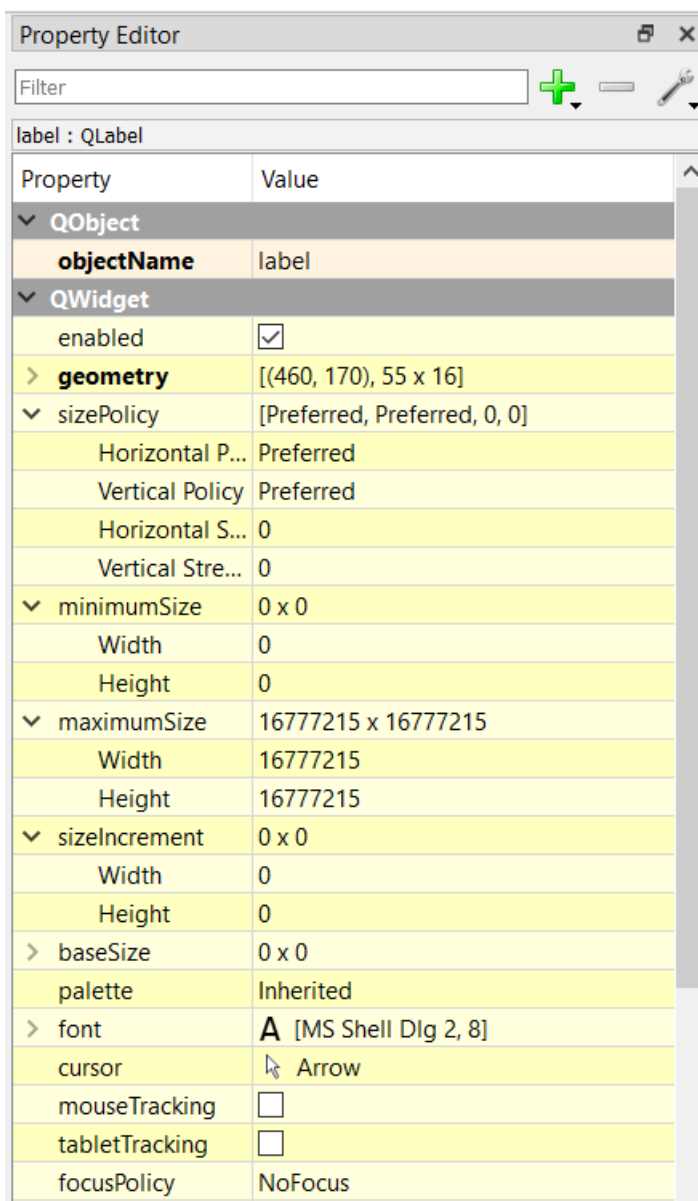
Input Widgets -> Horizontal Slider - Widget allowing selection of the right position on the slider, usually these items are represented by numbers,



Display Widgets -> Label - One of the most important widgets. It allows displaying output data. In the case of the application being created, it can be both information in the form of text and photos.

Property Editor

Another important function of the application is the property editor. It allows to manage widget properties such as height or width. In addition, it is possible to configure the speed of application expansion when increasing the dimensions of the window.



Property	Value
QObject	
objectName	label
QWidget	
enabled	<input checked="" type="checkbox"/>
geometry	[(460, 170), 55 x 16]
sizePolicy	[Preferred, Preferred, 0, 0]
Horizontal P...	Preferred
Vertical Policy	Preferred
Horizontal S...	0
Vertical Stre...	0
minimumSize	0 x 0
Width	0
Height	0
maximumSize	16777215 x 16777215
Width	16777215
Height	16777215
sizeIncrement	0 x 0
Width	0
Height	0
baseSize	0 x 0
palette	Inherited
font	A [MS Shell Dlg 2, 8]
cursor	Arrow
mouseTracking	<input type="checkbox"/>
tabletTracking	<input type="checkbox"/>
focusPolicy	NoFocus

In addition to setting basic window properties, for specific windows containing text, it is possible to set the font, font size and text formatting.

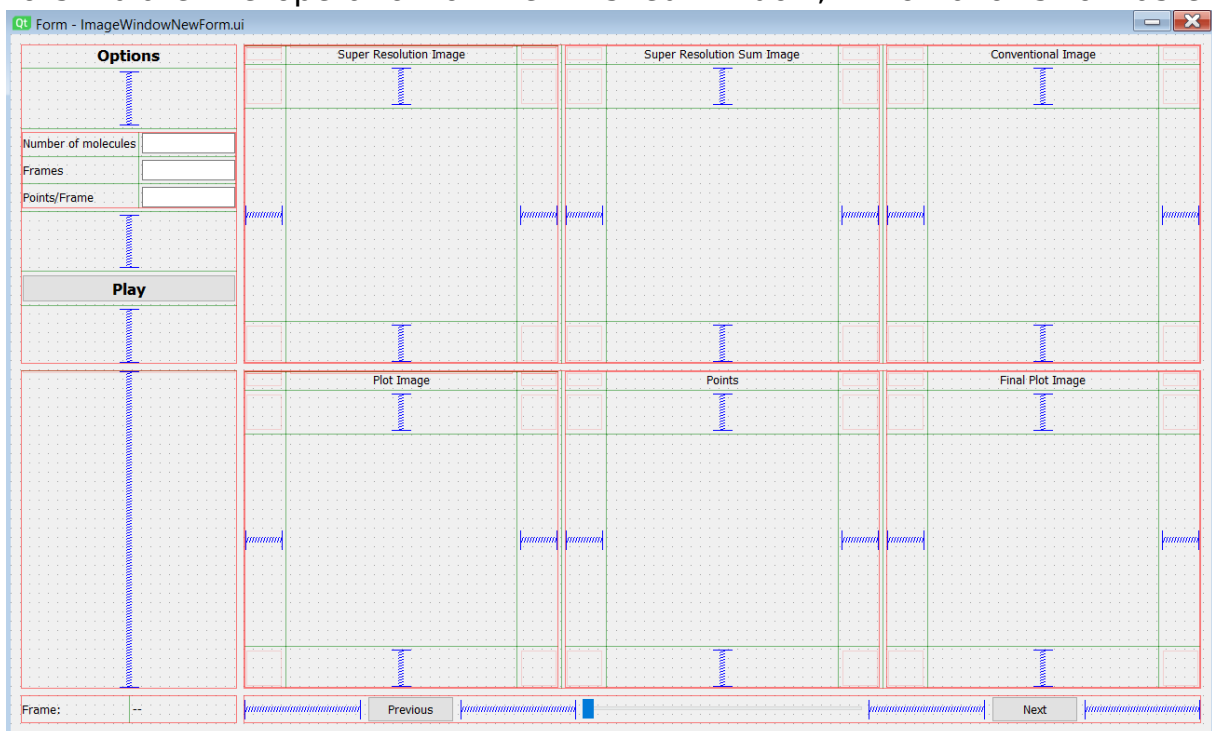
In the case of widgets that allow input of data, it is possible to limit what information will be accepted. The editor allows to set properties such as the type of data entered and the number of characters entered.

It is worth noting that this application is only used to create the graphical interface of the program. It is impossible to create relations between individual

elements (unless they are geometric relations). It is impossible to program widgets from the window of the "designer" application.

Creating an interface - Python QT

Creating a graphical interface in Python QT is a job requiring constant testing whether individual widgets interact in a correct manner. The hardest part in preparing the interface is not the layout of the widgets, but the creation of geometric relations that will make all the widgets look, not only correctly but also aesthetically when the application window is enlarged. This is what "spacers" are used, to ensure the distances between individual interface elements. The Python QT application allows to simulate the operation of the finished window, which allows for faster



work and tests, and whether the application works correctly.

This is the final version of the created application. The interface is built on a grid layout. Each grid is visible as a red window, and the widgets placed inside the red window are its elements.

Blue lines (spacers) allow to keep a specific grid in the chosen form. For example, if the window does not contain spacers and does not contain any other widgets, it will collapse into itself, making it invisible. This is especially useful in the case of a grid on the left, which does not

contain any widget but must maintain its form so that other grids can form with it.

The other elements of the window are labels that allows to display data, buttons that are used as a bridge between user and application, text windows in which user can enter input data and one horizontal slider in the middle.

Labels are blank squares in the central and right part of the window. The generated images will be displayed in them.

The buttons with the slider are used to select the appropriate frame of the generated photos. One of the "Play" buttons is used to initiate the frame creation algorithm.

In the text window, the user will be able to place parameters according to which the frames will be generated. Only numbers can be entered in the text window.

Creating a python file - Python QT

The window created should be saved in the application by default to .ui format. This is no the format that **Python** is going to be able to read.



The next step in creating the application is to convert the .ui file to a file with the extension .py. PyQt provides a special converter that allows to convert the file without any problems. Once again in case of having Windows 10 it is needed to use Command Prompt Commands (CMD) with

```
C:\Users\user>pyuic5 -o Application_Name_Python.py Application_Name_PyQT.ui
```

exact command.

“Application_Name_Python” refers to name of the .py file

“Application_Name_PyQT” refers to name of the .ui file

After the conversion, a new file will be created:

 Application_Name_Python.py	01.05.2019 12:02	Plik PY	21 KB
--	------------------	---------	-------

Code of the Interface - Python QT

There will be many lines of code in the .py file. When creating and implementing further algorithms, it will be necessary to create relations between individual elements. Unfortunately, PythonQT does not provide any text formatting, so most of the elements have to be found after the identical (as in the designer application) name.

```
class Ui_ImageWindow(object):
    def setupUi(self, ImageWindow):
        ImageWindow.setObjectName("ImageWindow")
        ImageWindow.resize(1286, 736)
        self.gridLayout_9 = QtWidgets.QGridLayout(ImageWindow)
        self.gridLayout_9.setObjectName("gridLayout_9")
        self.gridLayout_2 = QtWidgets.QGridLayout()
        self.gridLayout_2.setObjectName("gridLayout_2")
        self.gridLayout = QtWidgets.QGridLayout()
        self.gridLayout.setObjectName("gridLayout")
        self.label_6 = QtWidgets.QLabel(ImageWindow)
        self.label_6.setAlignment(QtCore.Qt.AlignCenter)
        self.label_6.setObjectName("label_6")
        self.gridLayout.addWidget(self.label_6, 2, 1, 1, 1)
        self.label_7 = QtWidgets.QLabel(ImageWindow)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Expanding)
        sizePolicy.setHorizontalStretch(50)
        sizePolicy.setVerticalStretch(50)
        sizePolicy.setHeightForWidth(self.label_7.sizePolicy().hasHeightForWidth())
        self.label_7.setSizePolicy(sizePolicy)
        self.label_7.setMinimumSize(QtCore.QSize(200, 200))
        self.label_7.setMaximumSize(QtCore.QSize(400, 400))
        self.label_7.setSizeIncrement(QtCore.QSize(0, 0))
        self.label_7.setText("")
        self.label_7.setObjectName("label_7")
        self.gridLayout.addWidget(self.label_7, 4, 1, 1, 1)
        spacerItem = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Expanding)
        self.gridLayout.addItem(spacerItem, 5, 1, 1, 1)
        spacerItem1 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)
        self.gridLayout.addItem(spacerItem1, 4, 2, 1, 1)
        spacerItem2 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Expanding)
        self.gridLayout.addItem(spacerItem2, 3, 1, 1, 1)
        spacerItem3 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)
        self.gridLayout.addItem(spacerItem3, 4, 0, 1, 1)
        self.gridLayout_2.addLayout(self.gridLayout, 0, 0, 1, 1)
        self.gridLayout_3 = QtWidgets.QGridLayout()
        self.gridLayout_3.setObjectName("gridLayout_3")
        spacerItem4 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)
        self.gridLayout_3.addItem(spacerItem4, 2, 2, 1, 1)
        self.label_9 = QtWidgets.QLabel(ImageWindow)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Expanding)
        sizePolicy.setHorizontalStretch(50)
        sizePolicy.setVerticalStretch(50)
        sizePolicy.setHeightForWidth(self.label_9.sizePolicy().hasHeightForWidth())
        self.label_9.setSizePolicy(sizePolicy)
        self.label_9.setMinimumSize(QtCore.QSize(200, 200))
        self.label_9.setMaximumSize(QtCore.QSize(400, 400))
        self.label_9.setText("")
        self.label_9.setObjectName("label_9")
        self.gridLayout_3.addWidget(self.label_9, 2, 1, 1, 1)
        self.label_8 = QtWidgets.QLabel(ImageWindow)
        self.label_8.setAlignment(QtCore.Qt.AlignCenter)
        self.label_8.setObjectName("label_8")
```

Coding algorithms

The first step in the implementation of the algorithms that generate the behavior of molecules was to adapt the algorithms of Professor Carlo Manzo to the environment in which Python QT works.

The main difference between the functions found in the PyQt environment and normal python environment is that each of them must have relationships with itself. This means that one of the parameters given when calling a function is itself.

Definition of the function:

```
def PreviousFrame(self):  
    frame = self.horizontalSlider.value()  
    frame = frame - 1  
    self.horizontalSlider.setValue(frame)
```

Calling the function:

```
self.pushButton_2.clicked.connect(self.PreviousFrame)
```

Knowing that it is possible to start adding new lines of code that will not only create new functions of the algorithms but will also create relations between the buttons and the actions that should be made after pressing them.

The application is divided into 4 parts due to the type of function:

- i) Functions that create elements of the application. These are all those lines that are responsible for the placement and creation of individual widgets.
- ii) Functions that are strongly associated with the user interface. These are functions that will be executed directly after the user's action (eg pressing a button).
- iii) Functions that are indirectly related to the user interface. That is, they can create folders or generate images, but they do not have a direct effect on the graphical interface.

- iv) Functions that are used to perform calculations. They do not work directly on the interface. They are triggered by other functions. Eg. Generating molecules

The most important lines of the code are commented along with explanations of what the individual functions are for. All functions are described in detail in the second part of the report, which also serves as documentation so that it is possible to reproduce or upgrade the work done on the application.

Application tests

The application has been tested in the Windows 10 operating system environment. In particular, individual cases of bad software usage were tested:

- i) Wrong Input Data,
- ii) Closing the application at the wrong time (ie generating photos, displaying photos, making calculations),
- iii) Killing the application from the administrator panel and restarting it,
- iv) Application operation on a different sized main window,
- v) Multiple images generation.

The application is protected against incorrect input from the user, ie words, negative numbers or special characters. In the case of entering parameters for which the generation of frames function should not take place, the program will not allow to run the function and will ask to change the parameters.

The program checks whether the requirements for running the application are met. It means whether there are folders in which photos will be generated. The program gets rid of the generated folders at closing.

The elements of the program window automatically adjust to the new window dimensions. The program is resistant to clogging memory. It allows for continuous generation of even large quantities of frames.

Conclusion

Software development is not an easy process. It requires not only programming knowledge but also a strategic and systematic approach to the discussed issue. Creating an application requires creative thinking and the right approach in the same measure as coding.

It is not only important to know the functions that are going to be implemented in the software being developed. It is equally important to think about a convenient interface that will ensure comfortable use of the application, the time at which the application performs its calculations and tests that check whether the software is reliable even in uncontrolled conditions.

What surprises the most is the fact how much work on the application prevails around it. This is, for example, finding the right environment, writing concise and meaningful documentation or commenting on the most important lines of code.

A well-written application will reliably serve its users. The created application will allow to be an irreplaceable tool in situations in which it is necessary to simulate and show the behavior of molecules. In addition, it works well at the moment when there is no access to advanced equipment that will allow to see protein behavior live or in situations where person wants to show images known in the laboratory but for a larger audience which laboratory would not fit.

Software development is a path that should not be overcome as soon as possible. It is necessary to approach the topic of creating applications slowly and gradually. Only this action will allow to create a program that will be both well-written and well-structured. In addition, this approach will make such software reliable even in the heaviest environment.

Bibliography

- 1) <https://www.python.org>
- 2) <https://www.riverbankcomputing.com/static/Docs/PyQt5/>
- 3) <https://www.qt.io>
- 4) <https://stackoverflow.com>
- 5) <https://docs.scipy.org/doc/>
- 6) <https://pillow.readthedocs.io/en/stable/>