Photo Editor

‘PhotoX’

Oliwia Mlonek

Computer Programming

Semester Project

Teacher: mgr inż. Wojciech Dudzik

Wydział Automatyki, Elektroniki i Informatyki

Politechnika Śląska

12.06.2020

1 Topic

The main goal of the project was to get familiar with using object oriented features and mechanisms of the C++ programming language. My choice was to create a simply Photo Editor that would allow editing and manipulation of the uploaded photo.

The user has the option of:
- photo color changes
- applying an effect to the image
- rotation and flipping
- face detection
- color space changes
- lightening and darkening the shadows
- basic operations related to the histogram

2 Analysis and Development

The first decision in the design process was to choose the library that would provide interfaces for displaying a window and rendering objects to that window. I decided the program will use wxWidgets library, since it had very clear and detailed documentation, which was easy to use.

Another library was to provide functions that would allow image manipulation. I chose OpenCV because it is a very well documented, popular, open source computer vision library that allowed me to perform all operations listed above.

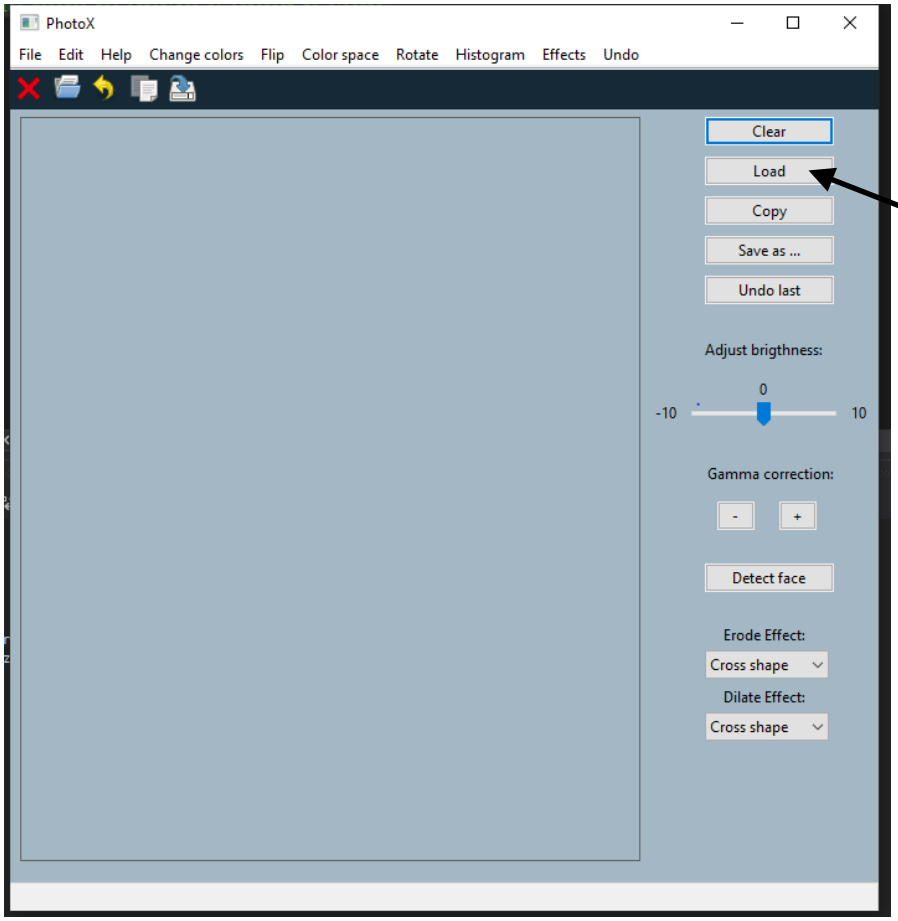
Most major operations were to be specified for smaller actions, such as:

- *changing photo color* would allow to perform three changes: to black and white color, to grey color and to sepia color.
- *applying an effect* would allow to perform two changes: apply dilate effect or apply erode effect.
- *rotation* would allow to perform two changes: rotate to left or rotate to right
- *flipping* would allow to perform three changes: flip vertically or horizontally or both at once.
- *color space changes* would allow to perform five changes: convert the color space to RGB or HLS or HSV or YCrCb or to BGR.
- *lightening and darkening the shadows* would allow to perform three changes: modify the actual image brightness working directly in pixels or through gamma corrections.
- *basic operations related to the histogram* would allow to perform two operations: display a histogram or make histogram equalization.

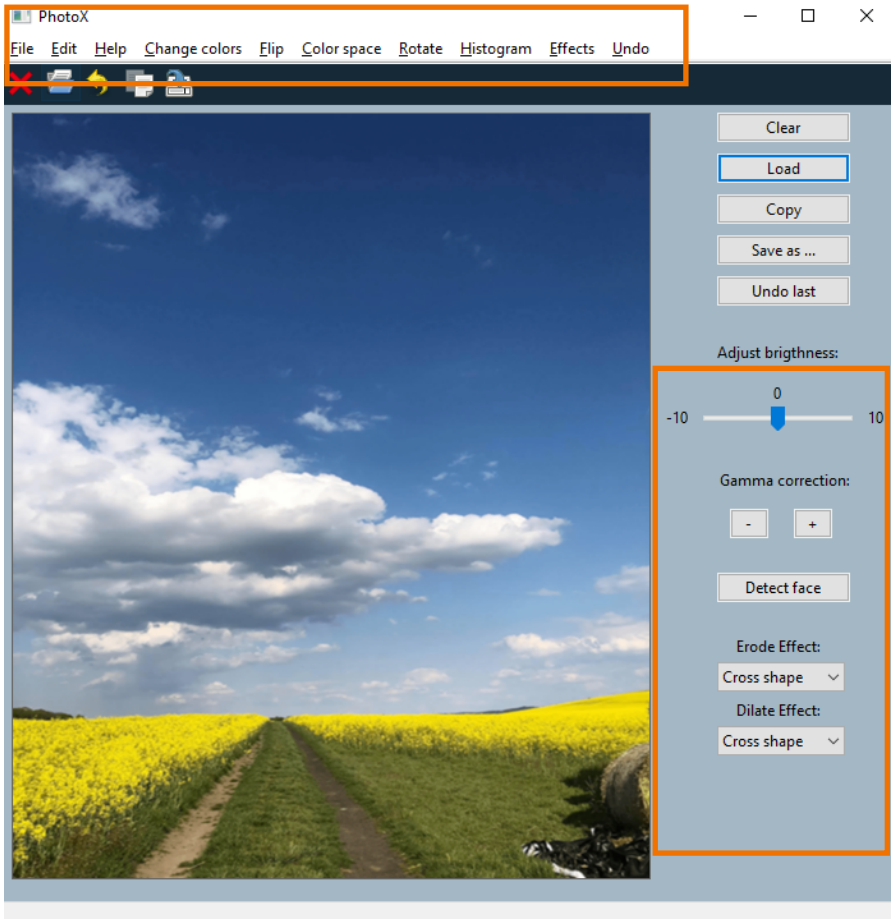
This structure prompted the idea of arranging classes for the project. We could define a family of algorithms, encapsulate each one, and make them interchangeable instead of implementing a single algorithm directly. So I decided that the structure of project class connections will be based on the Strategy Pattern Design. In addition to the class responsible for the GUI and the class managing the properties of the loaded image, the class Editor will be created which will be responsible for maintaining a reference to one of the concrete strategies and communicates with this object only via the strategy interface.

3 External specification

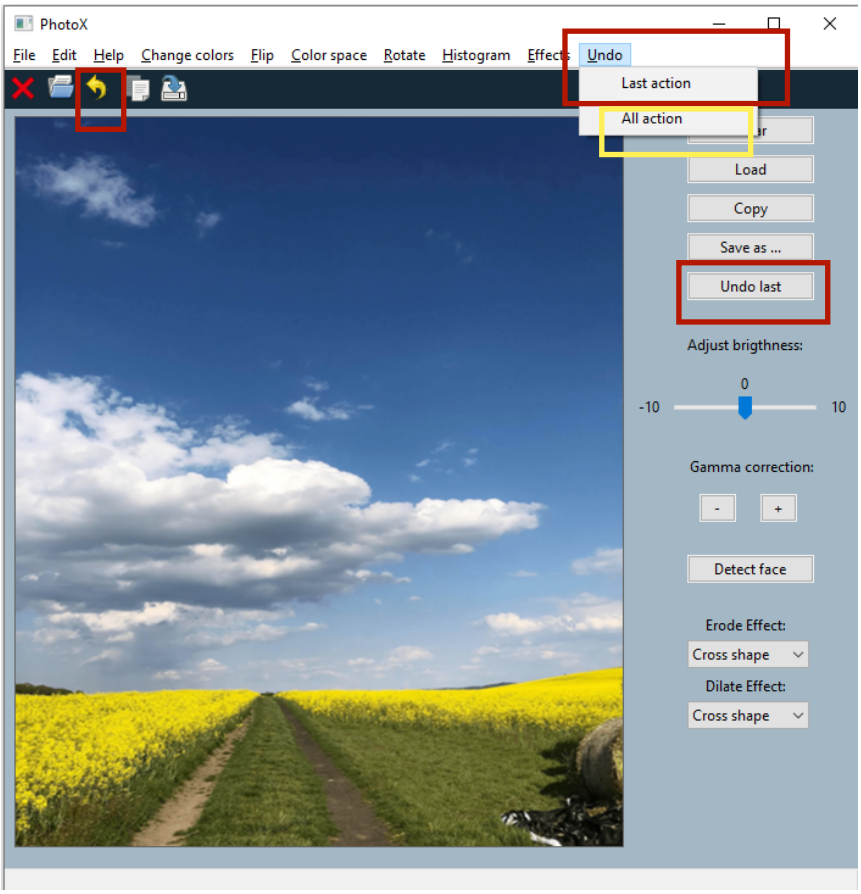
After launching the app user sees the GUI of the app, as shown on the screenshoot below:



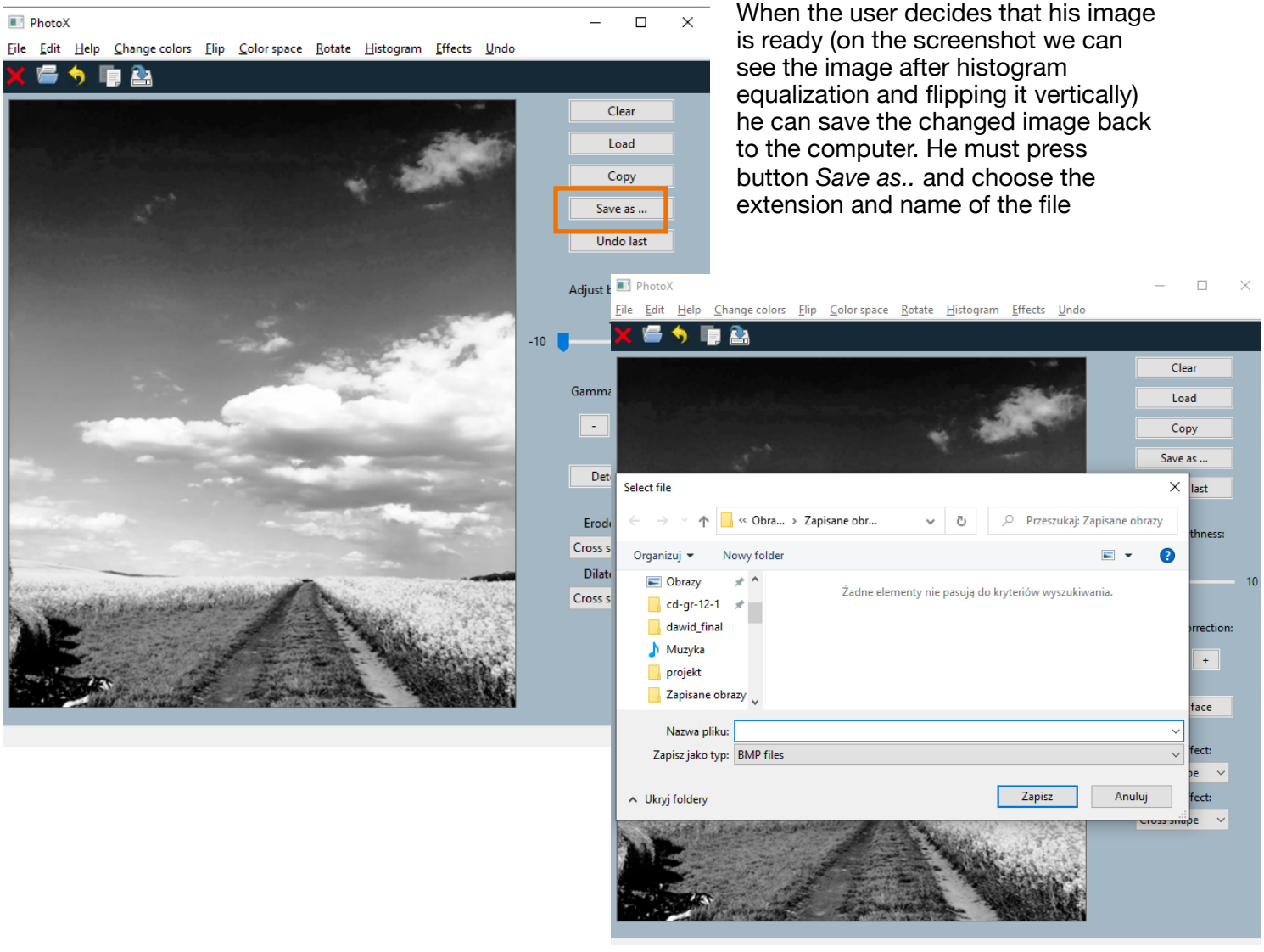
First thing to do should be loading an image in a proper extension (BMP or PNG) by pressing the *Load* button and choosing the proper file.



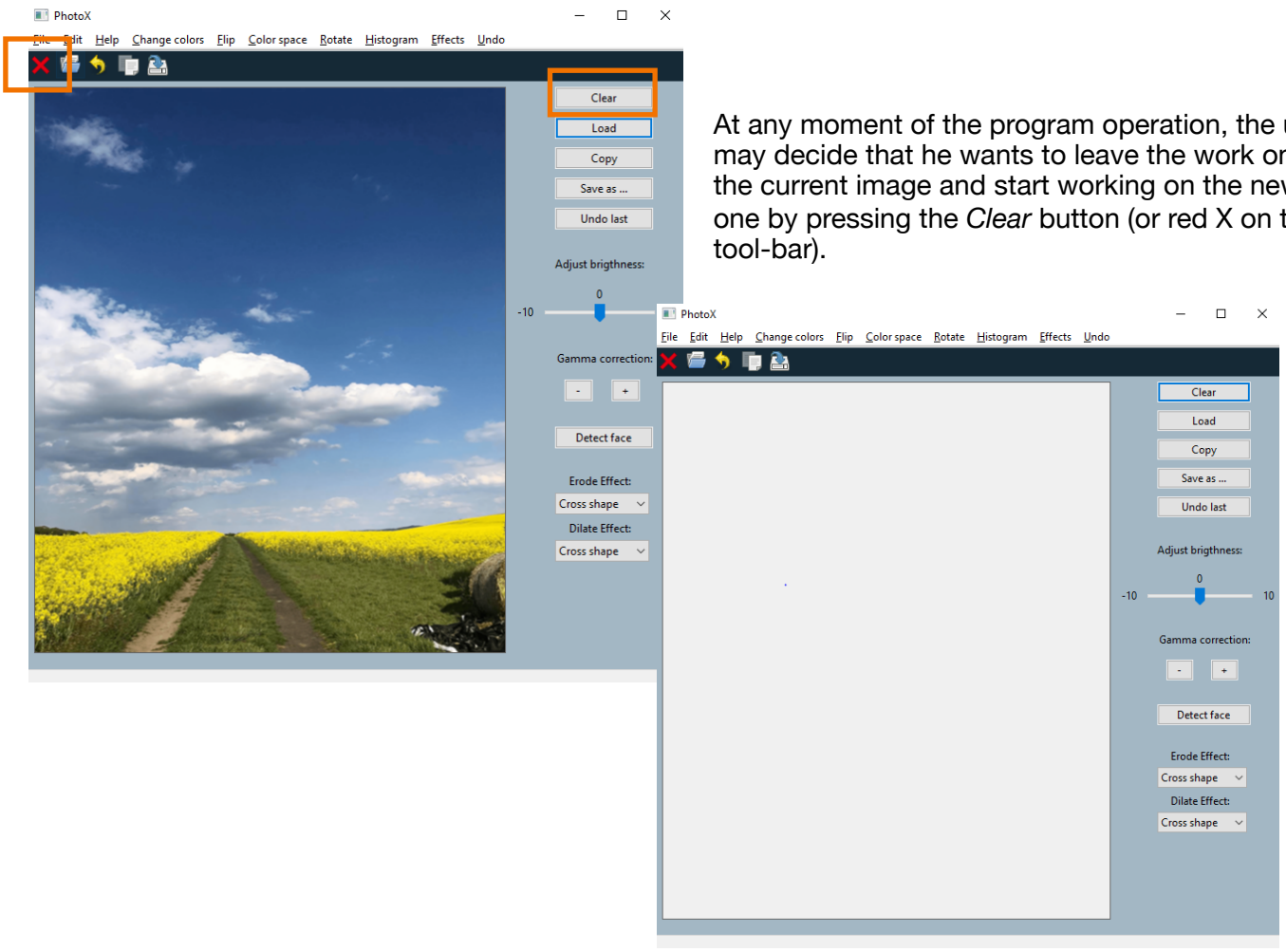
After loading user can select any option do modify an image. We can see a menubar with options, different buttons and slider, all for making changes in the chosen photo.



However, the user can reverse any operation by running the on of two UNDO function. He/she can cancel the last action using button, menubar or tool-menu
or reverse all changes made to date using menubar.



When the user decides that his image is ready (on the screenshot we can see the image after histogram equalization and flipping it vertically) he can save the changed image back to the computer. He must press button *Save as..* and choose the extension and name of the file



At any moment of the program operation, the user may decide that he wants to leave the work on the current image and start working on the new one by pressing the *Clear* button (or red X on the tool-bar).

4 Internal specification

- The data structures (C++ STL Containers) used in the project are as follows:

std::list<std::shared_ptr<Mat>> ImagesList; *holds the pointers to the images represented as Mats.*
std::vector<int> brightness; *stores information about current and previous values of light intensity in the picture.*
std::vector<int> PreviousSpaces; *stores information about current and previously selected color spaces of the photo.*
std::vector<int> mapping; *helper vector, necessary to convert Mat to wxImage.*

I chose the list instead of other containers because it allowed easy access to its beginning and end, as well as easy removal of elements. I didn't need random access so choosing the list was the best option. The list is also more effective in terms of time complexity.

- In the program, wherever it was possible and made sense, smart pointers were used. They can be found in MainWindow.h file, where point to the objects of different classes, in Photo.h where the list of pointers can be found and in every concrete strategy, as we must push the new image after any change to the beginning of a list so this new image can be displayed.

Two examples of using smart pointers:

```
11
12  void Erode::change_effect(std::list<std::shared_ptr<Mat>>& lista, int morph)
13  {
14      Mat img = lista.front()->clone();
15      Mat result;
16      int erosion_type = 0; if (morph == 0) { erosion_type = MORPH_RECT; }
17      else if (morph == 1) { erosion_type = MORPH_CROSS; }
18      else if (morph == 2) { erosion_type = MORPH_ELLIPSE; }
19      Mat element = getStructuringElement(erosion_type, Size(2 * 1 + 1, 2 * 1 + 1), Point(1, 1));
20      erode(img, result, element);
21
22      lista.push_front(std::make_shared<Mat>(result));
23  }
24
```

```
private:
    std::shared_ptr<Photo> photo;
    std::shared_ptr<Editor> edytor;
    wxString Path;
    string FileName;
    Mat toConvert;
```

However, there are some cases where I must have used raw pointers. It was due to the specificity of the libraries I selected which at some moments did not work with smart pointers.

- Many parts of the program use iterators. They allow free iteration through the containers and removal of unnecessary content and give access to selected elements.

Two examples of using iterators:

```
void UndoAll::change_undo(std::list<std::shared_ptr<Mat>>& lista)
{

    auto i = lista.begin();
    while (i != std::prev(lista.end(), 1))
    {
        lista.erase(i++);

    }
```

```
    auto it = std::next(picture->GetImageList().begin(), 1);

    if (CheckMatEquality(*picture->GetImageList().front(), (**it)))
        throw(NoFaceFound());
}
```

- In the project, there are some actions that may cause throwing an exception. I created my own three exception classes (Exceptions.h file). They can be thrown if:

* the list is empty, user did not upload any photo and try to make some actions related to photo editing;
* list has only one element, so it contains only originally loaded photo from directory and user try to undo something;
* face detection can not found any face.

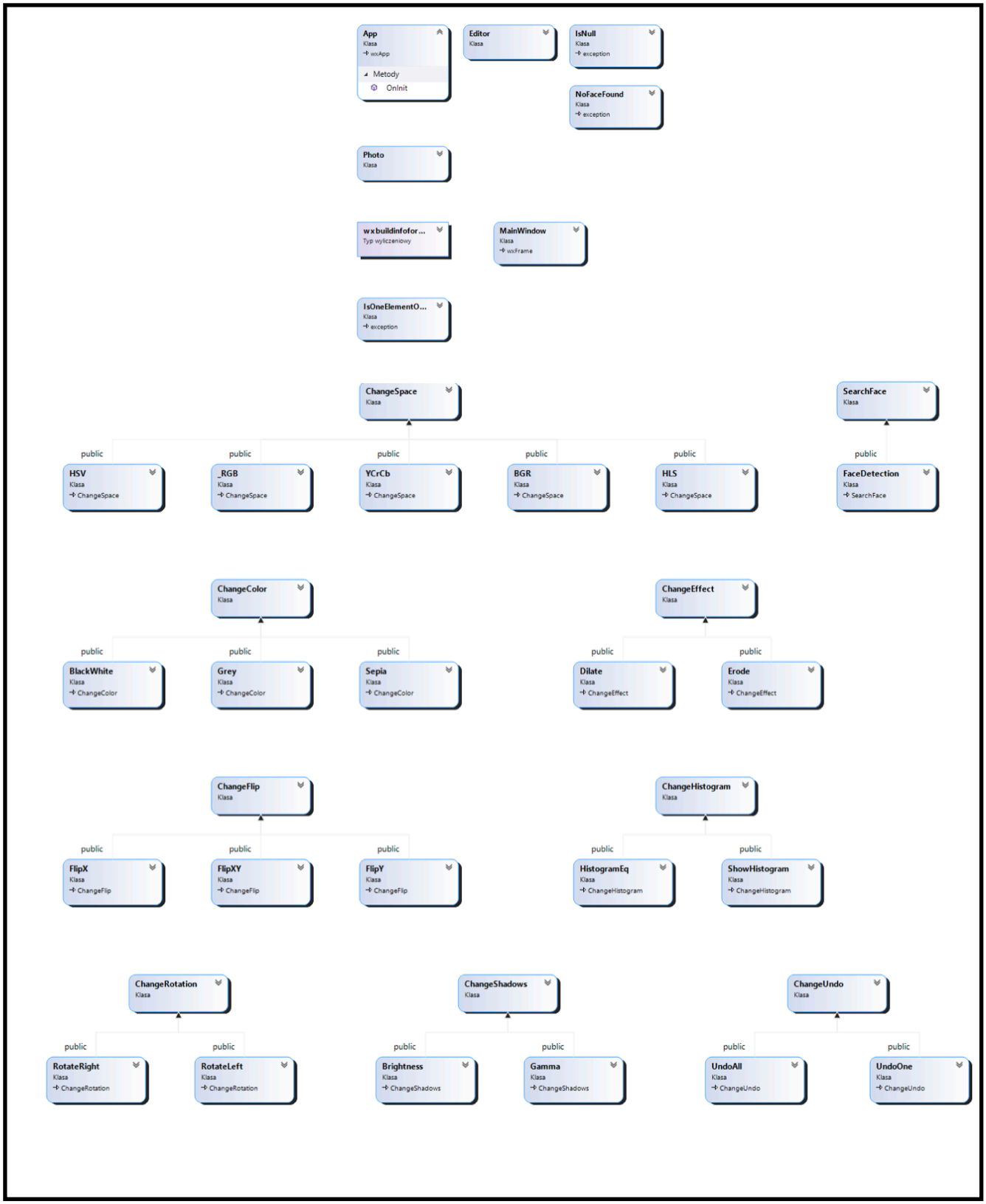
Example of using exceptions:

```
void MainWindow::UndoOneF(wxMouseEvent& event)
{

    try {
        edytor->which_undo(1);
    }
    catch (const IsNull& e) {
        wxLogMessage(e.what());
        return;
    }
    catch (const IsOneElementOnly& e) {
        wxLogMessage(e.what());
        return;
    }
    catch (const std::exception& e) {
        wxLogMessage(e.what());
        return;
    }
```

```
void Editor::which_undo(int which)
{
    if (picture->GetImageList().empty())
        throw(IsNull());
    else if (picture->GetImageList().size() == 1)
        throw(IsOneElementOnly());
```

- Basic class diagram:



Above class diagram is quite simple and it shows only basic relationships, i.e. inheritance. Expanded, UML class diagram is in the attached 'Class Diagram' file.

- Algorithms for Adjusting Brightness

- First used algorithm is based on the equation:

$$g(i,j) = \alpha f(i,j) + \beta$$

where i and j indicates that the pixel is located in the i-th row and j-th column. The parameters α>0 and β are called the gain and bias parameters, they control contrast and brightness respectively. To increase the brightness we need to increase the intensity of each pixel by a constant and similarly to darken the image we need to decrease the intensity of every pixel of the image.

To perform the operation we access to each pixel in image. Since we are operating with BGR images, we have three values per pixel (B, G and R), so we will also access them separately. Here is the code:

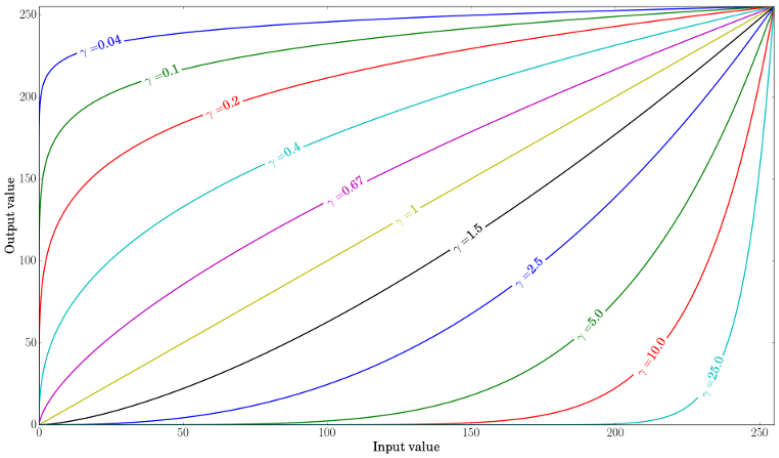
```
for (int y = 0; y < final->rows; y++) {
      for (int x = 0; x < final->cols; x++) {
         for (int c = 0; c < final->channels(); c++) {
            uchar piksel = final->at<Vec3b>(y, x)[c];
            uchar default_piksel = saturate_cast<uchar>(piksel - beta_poprzednia);
            new_image.at<Vec3b>(y, x)[c] =
               saturate_cast<uchar>(1 * default_piksel + beta);
                  }
         }
}
```

- Second algorithm - Gamma correction - can be used to correct the brightness of an image by using a non linear transformation between the input values and the mapped output values:

$$O = (\frac{I}{255})^{\gamma} * 255$$

As this relation is non linear, the effect will not be the same for all the pixels and will depend to their original value.

When γ<1, the original dark regions will be brighter and the histogram will be shifted to the right whereas it will be the opposite with γ>1.



Here is the code:

```
double invGamma = 1 / FromSlider;
   Mat lookUpTable(1, 256, CV_8U);
   uchar* p = lookUpTable.ptr();
   for (int i = 0; i < 256; ++i)
      p[i] = saturate_cast<uchar>(pow(i / 255.0, invGamma) * 255.0);
   LUT(*src, lookUpTable, *dst);
```

- Algorithm for Face Detection using Haar Cascade

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

Here is how to algorithm is loaded and activated:

```
CascadeClassifier cascade;
        cascade.load("E:/opencv/sources/data/haarcascades/
haarcascade_frontalface_alt2.xml");  //the proper cascade with needed algorithm is
loaded


        vector<Rect> faces;
        Mat gray;
        cvtColor(img, gray, COLOR_BGR2GRAY);
        cascade.detectMultiScale(gray, faces, 1.1, 2, 0 | CASCADE_SCALE_IMAGE,
Size(30, 30));// algorithms  is called, faces are detected
                for (size_t i = 0; i < faces.size(); i++)//draw  squares around the found
//faces
        {
                Rect r = faces[i];
                Scalar color = Scalar(255, 0, 0);
                rectangle(img, Point(cvRound(r.x * scale), cvRound(r.y * scale)),
Point(cvRound((r.x +
                r.width - 1) * scale), cvRound((r.y + r.height - 1) * scale)), color, 3, 8,
0);
        }
```

- General scheme of the program operation

After launching the application and displaying the GUI, the program waits for user move. Pressing any button in the displayed GUI calls the appropriate methods and functions necessary to perform the task. If the image is already loaded and user chooses some method of image editing, his choice is passed to the object of the class Editor, where in properly grouped switches his choice determines and evokes concrete strategies. After finishing the modification, the newest image is pushed at the begging of the list. Afterwards, refresh function is called to display the new image with changes. This

function takes always the first element of the list and displays it. However, if the undo function is called some elements od the list are erases instead of pushing any new image to it. First element of the list points to one of the older images, so after refreshing, the image without unwanted changes is displayed.

Rest of the technical details of the program can be found in the doxygen documentation (‘Documentation’) file attached to the project.

5 Testing and debugging

The main problem I found during writing this project was lack of conversion between variables responsible for storing an image from OpenCv library to wxWidgets. I decided that I want wxWidgets library to be in charge of displaying the image, so after editing a photo (using variables and functions from OpenCV) I was not able to display it properly. After some research I wrote a separate function which would convert those two types. After implementing it, the images were displayed correctly.

I also had unexpected ‘assertions failed’ messages that crushed the app. It was related to functions that iterate through lists and vectors. However, they were very easy to fix

I tested the program manually, over the weeks, by clicking many options in different combinations on different types of images. I also asked for tests of people not involved in the project, who did not know how exactly the application works.

The program was also tested for memory leaks and and none were found.