C++ - BsCV:

Project Report - CandyXel

By: Corsin Romain

Professor: Yohan FOUGEROLLE

May 23, 2017

Contents

I. DEFINITION OF THE PROJECT

A. What is the project ?

This project main goal was to create a special application to create some pixel art using candies and if possible return the price of the amount of candies used to do the pixel art. The software could be used paired to a robot to draw at the end the candy image. Actually my project can manage any jpg image, from the first folder or any wanted folder. I tried to create an interface user friendly.

B. Definition of a pixel art

A pixel art is a technique of numeric drawing, it come from the 80's and 90's when the screen screen resolution and the number pixel color were low, which make graphic designer draw pixel by pixel.

Here the principle on my project is to convert an normal image to an other image where the pixels will be bigger. The principle of the project is to take an image, define some groups of pixels and change it by an image which will represent the set of pixels inside the group, with a good algorithm the representation of the group of image doesn't modify the image a lot when we are watching at it not very close. The precision of the image depend of the quality of the algorithm and the number of the images.

II. GENERAL REVIEW

A. Different class

First I will explain you my 4 different classes used inside the project.

1. MainWindow

The MainWindow class is the most useful class in my project and probably in every project, this class permit the interaction between the program and the user, it permit to use the function of slots and every widgets inside on the graphic user interface.

Inside this class we will find the greatest part of my function.

2. Pixel

The Pixel class will be use to contain the value of every pixel of the entered image, it almost a QColor but I needed a boolean, so I had to create this class.

3. Candy Image

This class will contain every information needed for my images :

- A pointer to the image
- The mean color of the image
- The price when I will find a solution to implement it

4. Dico image triple

This one will contain the image library, I need to keep this one inside a separated class to avoid any problems of memories.

III. GRAPHIC USER INTERFACE (GUI)

A. Principle of the graphic interface

A graphic interface is a way to make the software more user friendly, almost everything can be coded in code line only, asking value or data to the user, but a graphic user interface permit to get an easy interaction between the code and the user, it permit to display some information like images, obtain some graphic data like the position of the mouse and permit to structure the program using some slots.

For each widget we are using, we can create some subprogram depending of the event link to a widget, for example, with a button we can select to create many different functions depending if the button is pressed, released of clicked. Some other widgets are using mostly in a passive way, they are not used to collect informations but to display or give some like label or graphic view. The last type of widget is the "design" widget, they are just used to make the interface cleaner. We can create our own widget but this one will be used as one of the three type I give before.

```
void on_XPixel_new_editingFinished();
void on_XPixel_new_valueChanged();
```

To get a example from my project I will use two different events of spinbox : if the value is changed by hand (keyboard enter) or modified with scrolling arrow. This two events look similar but if we delete one, one event will not be taking in account, the "editingFinished" signal is sent when the spinbox is modified with the keyboard while the "valueChanged" signal is sent when the spinbox value is modified using arrows.

The type of signal managed by the program is different depending of the widget, to see every signals of a widget or to "open" the function inside our code, we just need to do a right click on the wanted widget click on "go to the slot", after this Qt will display every type of signals managed by the widget, when we will choose one this will open a private slot inside our header file linked to the user interface and open the function inside the cpp file.

```
    private slots:
        void on_button_pixelise_clicked();
        //inside the header file

        void MainWindow::on_button_pixelise_clicked(){}
        //inside the cpp file
```

B. View and explanation of my GUI

Now I will present my own user interface, it is composed of 4 buttons, one to load the image we want to be pixelised, one to pixelise with the setup we want, one to save the image and the last is to load every images inside a folder, this images will be used to replace pixels.

We got 3 radio buttons, the specificity of radio buttons is that we can got only one radio button checked at the same time, when we check another one, the one checked before set itself its boolean to false. I use them to choose the type of pixelisation.

We also have a tabWidget, this widget permit to sort things, here I use one to let the user decide to modify the type of selection, he can choose the number of big pixels or the size of big pixels. Each method is inside a different tab. We can create as many tabs as we want inside a tabWidget, we can give as example the software photoshop where you get a tab for the drawing of poligons, one for selection or colors.

We have 4 spinbox inside the tabWidget, a spinbox is a widget containing a value, this value can be change using two arrows to increment or decrement the value of the spinbox, we can modify the value entering the value with the keyboard. On my code this spinbox permit to change the number or the size of pixels on coordinate X or Y.

The last widget I used is the label, a label can contain many things, an image, some text, here I display two images inside two labels and some text.

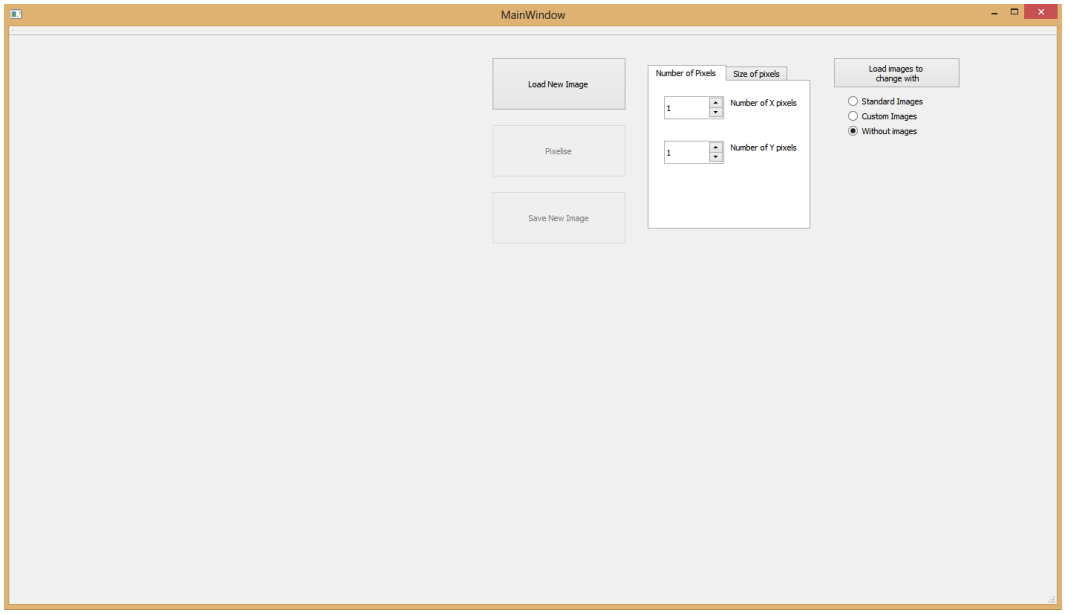

FIG. 1: User interface just opened

On the image above we can see the graphic user interface exactly as it is when we compile the code. I opted for a clean interface, every visible widget are on the top right of the screen. Two labels are hidden because they need the user to enter an image to be visible, as we can see on the image below, on this image the label on the top left contain the original image given by the user, on the bottom right we can see a second image with a different set of pixels, a pixel on the bottom right image contain 5x5 = 25 pixels of the original image.

The bottom left image is not inside a label, this is draw by the paintevent function of the mainWindow class. The images on the bottom left and right get the exact same property of the pixels for the size, so we can say that here the left one is the representation of the right one using a set of 60 images.

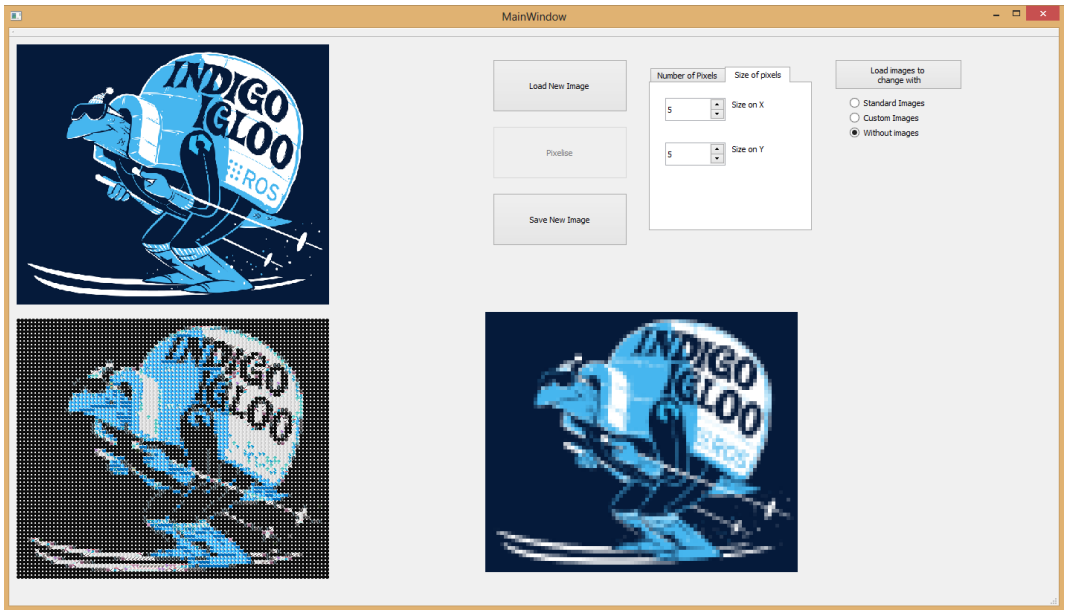

FIG. 2: User interface with an image, the pixelised image and the pixel art

Here the original image is the logo of ros indigo, on this set of images we can see that we got some trouble with the dark blue, replaced by black. I will show some image done with the program with the same set of images.

FIG. 3: Pixel art vs original image

Here I took some good result images.

IV. CODE EXPLANATION

A. Created class

Here I will define and explain every functions I'm using inside my code. We will not see accessors and mutators because this is just a cleanest way to access or modify some data.

1. Pixel

This class is used to contain a QColor and a boolean, the color will be used to replace the data of the original image, the boolean will permit to compute only once each pixels of the matrix.

The constructor is basic, we are setting the boolean to false and it's over, this will be a passive class.

```
class pixel
{
public:
    pixel();
    ~pixel() {}
    QColor color;
    bool done;
};
```

```
pixel::pixel()
{
    done = false;
}
```

2. *Candy Image*

This class will manage my images, I collect the path with a pointer used when I load the image folder. When I create an occurrence of this class, we use a function to setup the mean color of the image, to do this I go through the image, add each component of each pixel to a total of red, green and blue. After do the total of each component we divide the total by the number of pixels used, then we obtain the mean color of the image. The last thing which for the moment is useless is a double which represent the price for a candy, for the moment I don't know how to take the value with a user friendly manner.

```
class Candy_Image
{
public:
    Candy_Image();
    Candy_Image(QImage* );
    ~Candy_Image() { }

    QImage *candy;
    QColor mean_color_image;
    double price;

    void setupColor();
};
```

About the algorithm of the mean color of the image I think it could be optimize for the lines of codes and instructions updating the mean each time instead of doing the mean at the end like this :

```
mean = ( mean * (ex number of occurences) + new value ) / new
   number of occurences
```

This algorithm permit to decrease the number of instructions but due to the calculus I think the one I'm using is better for processing time.

3. *Dico image triple*

Actually this class could be useless, I could use a standart vector and put the function inside the MainWindow, but for me it's clearer like this. This class represent the image library i'm using for the pixel art.

The vector i'm using contains pointers to images, the function will be used to find the best image to replace a pixel. This function will take as argument a color and return the pointer to the corresponding image.

```
class Dico_image_triple
{
public:
    Dico_image_triple();
    ~Dico_image_triple() {  }

    std::vector<Candy_Image> Image_Array;

    QImage* find_nearest_image(QColor);
};
```

Here I will explain the function to find the best match image. I'm using two integers to compare the matching chance between two images, I'm going through the images vector while I'm comparing every image with the previous best match, if the match is better, we change the matching image. To compare two images we compare the color to the mean color of the image, we calculate the difference using the trichannel optimization of the RGB component. The principle is simple, we calculate the difference for each component and add them together to obtain an integer which will be compared to the best previous difference match. At the end we return the best image using the iterator we change during the for loop.

```
QImage* Dico_image_triple::find_nearest_image(QColor transfo)
{
    int totalDiff;
    int diff = 255*3;

    int it = 0;


    for(int i = 0 ; i < Image_Array.size() ; i++)
    {
        totalDiff = abs(transfo.red() - Image_Array[i].
           mean_color_image.red())
                    + abs(transfo.green() - Image_Array[i].
                       mean_color_image.green())
                    + abs(transfo.blue() - Image_Array[i].
                       mean_color_image.blue());

        if(totalDiff
        {
            it = i;
            diff = totalDiff;
        }
    }
    return Image_Array[it].candy;
}
```

B. MainWindow

The MainWindow class is the biggest one so we will partition it on sections.

1. header file

I will explain succinctly every functions and variables. For this project I need to draw inside the mainwindow, the find color function permit to find the mean of color of each big pixel. The transform pixel function will change the representation of the pixelise image or the pixel art. the loading function permit to load every png from a folder. I need a bidimensionnal matrix containing some pixel to calculate the colors of big pixels and I need the boolean to not compute many times each pixels. A QDir permit to keep a directory to a folder or an image, here it will be used to point to the folder of images. We need the image library, a boolean to know when should we repaint the window, two QImages to contain the original image and the one we will change.

```
explicit MainWindow(QWidget *parent = 0);
~MainWindow();

void paintEvent(QPaintEvent *); //the function which will
   paint on the window
void find_color();
void transform_pixel();
bool LoadAllPixmaps ( const QDir& = QDir("Image_Library"),
   bool verbose = true);

std::vector < std::vector < pixel > > pixel_matrix;

QDir image_user;
Dico_image_triple Dico_image;
bool paint;
QImage origin_image;
QImage new_image;
```

2. find color

The function find color permit to find the mean color of each big pixels, first we calculate the size of every big pixels. On this purpose we go through the image and we create the big pixel groups we create a first loop to compute the mean after this we set the color of every pixels of the group to the color of the mean. We change at the same time the value of pixels of the new image.

To avoid any trouble we also create a sieve to avoid to try a cell of the vector outside its length, because of this we are not using the size of big pixels to compute the mean, we create an iterator which will be incremented for each iterations of the loop.

At the end of this function we are calling the following one, which will represent every pixels by an image are show the new pixelised image.

3. transform pixel

To represent the image, we need first to know the type of pixelisation we want, we need to check the radiobutton, because we are using three radiobuttons we prefer check the only one non linked to images, if the one without images is checked, we just show the pixmap of the new image inside the label.

If it is not checked we need to repaint the window, to do this we set the boolean of repainting to true and let the paint event do the job.

```
void MainWindow::transform_pixel()  //repaint the window
{

    if(ui->radioButton_noImage->isChecked() == true)    //we
       are checking the radio button used
    {
        paint = false;
        ui -> label_image_pixel -> setPixmap(QPixmap::fromImage
           (new_image));
    }
    else
    {
        paint = true;
        repaint();
    }
}
```

4. Paintevent

The paint event is a function from the paintevent library, it allow us to draw on the mainwindow. The library is provided with a lot of functions, we need first to create a tool used to draw, if we wanted to draw lines or geometrical shape, we could set the color.

To draw an image as we want, we need to define the target rectangle and the image. We first recover the the color inside the pixel vector, then we search the best image corresponding to the color and then we display it on the window, the target rectangle is define thanks to the loops.

When everything is drawn, we set the boolean value to false.

```
painter.drawImage(target rectangle, image, space of the image);
```

To create the target we use a QRectF, a variable which can contain four coordinates.

We use four different type of slot used on three different widget, two for the spinbox, if the value had been changed by hand or using arrows, one for pushbuttons when the button is clicked and the last if the radiobutton is clicked.

```
        void on_YPixel_size_editingFinished();
        void on_YPixel_size_valueChanged();

        void on_Button_load_images_clicked();
        void on_radioButton_standard_clicked();
```

The spinbox event are always linked to same events we modify the value of the size or the number of big pixels, they are linked together.

Action behind pushbuttons are more complicated, for the image load button we first search the image with a dialog box, then we display it inside a label. We modify the setup of spinbox maximum value, because we can't get a size of a number of big pixel greater than the size of the image.

For the pushbutton which load images folder, we got two different possibilities, the first is if the user want a standard library, if this is the case, the standard folder path is in the script. For the other case we open a dialog box to obtain the wanted folder.

We also get a button to save the pixel art, using again a dialog box to choose the folder and the name of the image. We take a screen shot of a part of the window, the part of the window where is the pixel art.

The last one will pixelise the image, during the pixelisation we resize the matrix and set color of the pixel matrix.

V. TO GO FURTHER

I keep some things unfinished, the price of candies, for me the thing I really wanted to achieve is a smart way to display the image, more than a drawImage, I think with a GraphicScene it could be better.

Another thing I wanted to work more on is the time of processing, with a better algorithm I think it could be faster, and to correct the principle of paintevent, dialog boxes reset the window draw, so this erase the pixel art after the save.

I wanted to create a mobile application but on this project a mobile application is not compelling.

VI. CONCLUSION

During this project I tried to see things with a new approach, I tried to read codes ,understand them by myself and with this I understand that I have some trouble to understand an approach which is not mine. I'm probably too much focused on my point of view.

During the holidays I hope to finish this project and make the one I wanted to do before, try to create a mobile application on the principle of magic drawing.

github repository : https://github.com/RCorsin/QTProject