Computer Science - QT and C++

Pixel art project report

Pixel art project report

Contents

1 Project purpose

The main purpose of this project is:

- Ability to develop graphical applications using Qt, C++ and good oriented object practices.
- Focus on OOP practices in particular constructors, destructors, access rights, inheritances, templates and classes.
- Focus on simplicity of the interface.
- Focus on your ability to use numerous Qt widgets and objects and be illustrative or your ability to construct rich GUI applications with menus, drawing, short cuts, tabs, proper signal and slot connections, etc.
- Be fully commented and documented for clarity and conciseness.

2 Introduction

Pixel art is a form of digital art wherein images are created and edited at the pixel level using a graphics editing software. What defines pixel art is its unique visual style, where individual pixels serve as the building blocks that make up the image. The effect is a visual style very similar to that of mosaic art, cross-stitch and other types of embroidery techniques.

3 Project structure - part 1

- Design and draw the interface using the Qt design.
  - Create widget to display image.
- Include all directives needed to load and display image.
  - In the header file, include all the required libraries needed.
  - Define all the private attributes of the class Mainwindow.
  - Create and define a class in a node to host the pixelisation and the pixel art process.
  - First important step : Load and Display an image.

3.0.1 Load and display an image

3.0.2 Understanding

In this section the goal is to include in the header file the required Qt libraries suitable for all actions that need to be taken to process and display an image.

3.0.3 Standard Libraries

In the C++ programming language, the C++ Standard Library is a collection of classes and functions, pre-built which can be use depending on the need of the desire program.

- The QVBoxLayout class lines up widgets vertically.
- The QScrollArea class provides a scrolling view onto another widget.
- The QFileDialog class provides a dialog that allow users to select files or directories.
- The QLabel class provides a text or image display.
- The QMenu class provides a menu widget for use in menu bars.
- The QAction class provides an abstract user interface action that can be inserted into widgets.
- The QTabBar class The QTabBar class provides a tab bar, e.g. for use in tabbed dialogs.

3.0.4 Algorithm

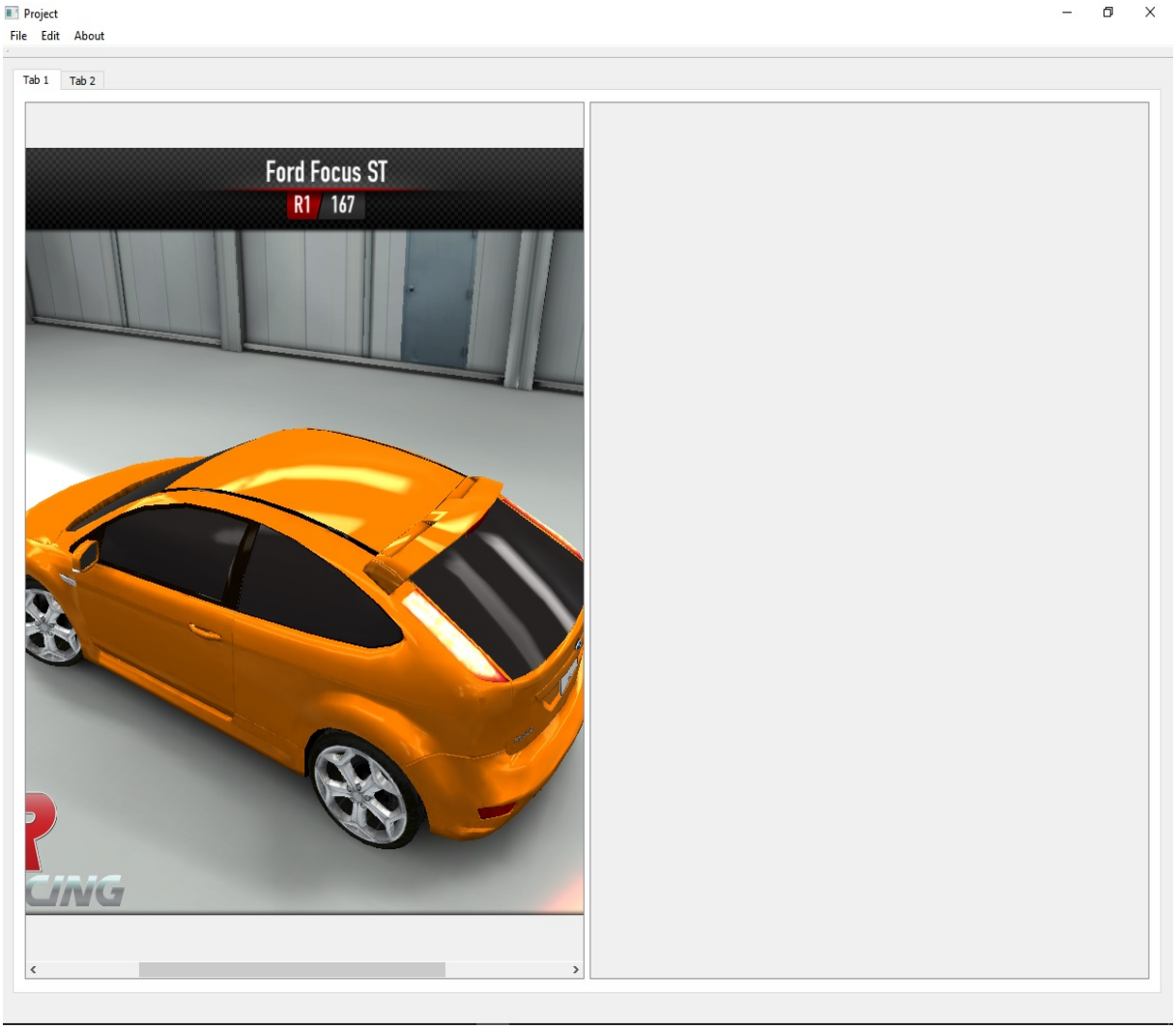
Header File mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

// Include required standard library

#include <QMainWindow>
#include <QVBoxLayout>
#include <QScrollArea>
#include <QFileDialog>
#include <QLabel>
#include <QMenu>
// #include <QSpinBox>
#include <QTabBar>
#include <QMessageBox>
#include <QAction>
#include <iostream>

namespace Ui {
class MainWindow;
}

// The Qmainwindow is the class which Organise and understand my widgets
class MainWindow : public QMainWindow
{

// QObject is the base class for many of the important classes that will be used in
    the Qt library,
// such as QEvent, QApplication, QLayout, and QWidget.
Q_OBJECT

public:

// Construtor
explicit MainWindow(QWidget *parent = 0);
// Destructor
// of my widgets ends.
~MainWindow();

public slots:
// Function to load image
void load_image();

private:

// Define pointers for every action to be taken later.

Ui::MainWindow *ui;
QScrollArea *scroll_original, *scroll_final;
QLabel *label_original, *label_final;

QMenuBar *menuBar;
QMenu *menuFile;
QAction *actionOpen, *actionSave, *actionExit;

};

#endif // MAINWINDOW_H
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
QMainWindow(parent),

//Constructor
ui(new Ui::MainWindow)
{
ui->setupUi(this);

//Constructing labels
label_original = new QLabel();
label_final = new QLabel();

//Constructing scrollarea
scroll_original = new QScrollArea();
scroll_original->setWidget(label_original);

//Constructing Menus
menuBar = new QMenuBar();
// Containing a new menu file
menuFile = new QMenu();
// which has a new action
actionOpen = new QAction();
// which must be added to my menu
menuFile->addAction(actionOpen);

// Add Menu functionalities
connect(ui->actionOpen, SIGNAL(triggered()), this, SLOT(load_image()));

menuBar->addMenu(menuFile);
}
// Destructor of the mainwindow
MainWindow::~MainWindow()
{
delete ui;
}
// Behaviour of the function to load the image
void MainWindow::load_image() {
// Here is where we declare the behaviour of the function load_image

// The QFileDialog class provides a dialog that allow users to select files on
    directories.

QString filename = QFileDialog::getOpenFileName(this, tr("Open"),
QDir::currentPath());
QImage image (filename);

// The QPixmap class is used for painting the Image in the predefinded display
    widget .
ui->label_original->setPixmap(QPixmap::fromImage(image));
ui->label_original->adjustSize();
```

Interface output



4 Project structure - part 2

- Second Important step : Define in the class " improcess " a function to pixelised an image.
- Process image pixelation.
- Paint image in display widget.

4.1 Average method

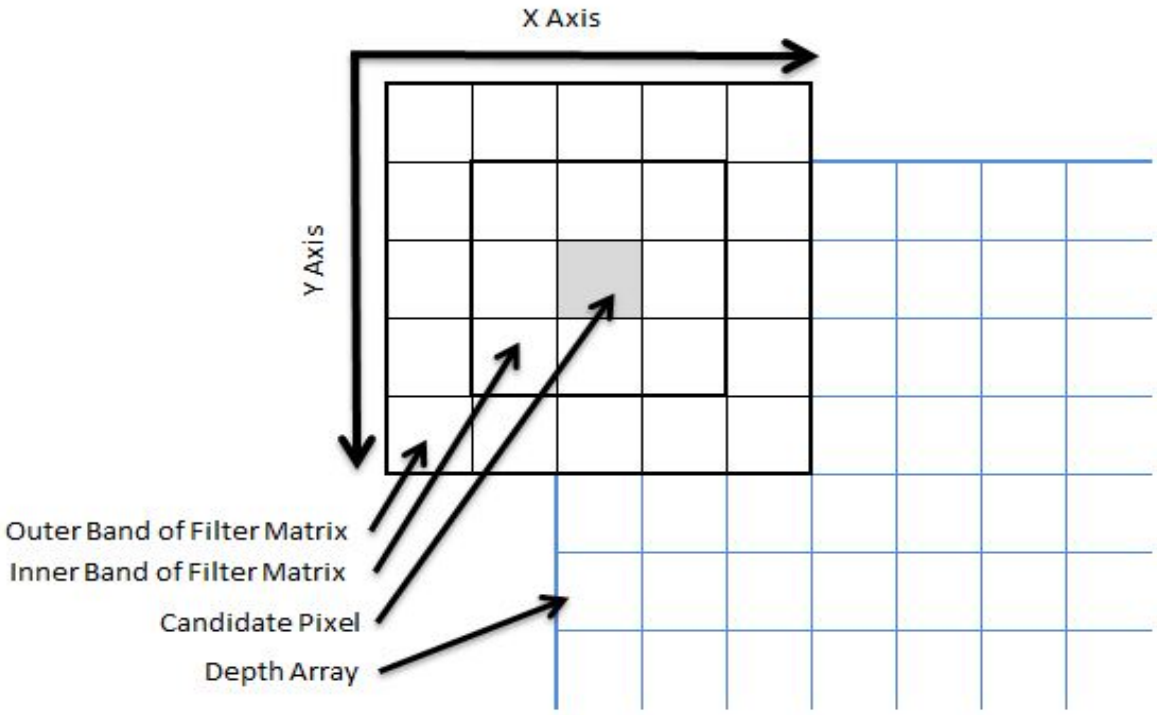
4.1.1 Definition

Pixelating an image consist of divide an image into pixel so that it is break up into visible pixels.

The Average method consist of parsing the uploaded image and collecting information of this image, information which will later be used to compute the average of each pixel in the Image. Pixel in the image will then be replaced by their average pixel value to obtain a new pixelised image.

4.1.2 Understanding

How the mask move over a image



The mask goes over each pixel in an image and collect information.

Average Method Technique

The average method, consist of retrieving all the pixels values of the neighbourhood pixels, calculate the average value and replace the main pixel by its average.

4.1.3 Algorithm

Header File

```
#ifndef IMGPROCESS_H
#define IMGPROCESS_H
#include <QImage>


// Defining a class improcess for the pixelisation process
class imgprocess
{
public:
imgprocess();

public slots:
QImage pixeliseFunction (QImage img, int window);
};

#endif // IMGPROCESS_H
```

Source file

```
#include "imgprocess.h"

imgprocess::imgprocess()
{
}

// Function to pixelised my image
QImage imgprocess::pixeliseFunction (QImage img, int window) {

//Retrieving my upload image caracteristics
QImage imgFinal(img.width(), img.height(), img.format());

// Walking on my Image to get the RGB colour and the total pixel in my image
for (int i = 0; i < img.width(); i+=window-1) {
for (int j = 0; j < img.height(); j+=window-1) {
// Defining an array name " sum   "to save each RGB total intensity value
// the array basically take 3 main value The Red pixel value, the Green and the blue
// all initialised at 0
int sum[3] = {0,0,0};
// Defining a variable of type integer initialised to 0 to save the total pixel
    value
int count = 0;

// this loop statement will go over the hole image
for (int x = i - window/2; x <= i + window/2; x++) {
for (int y = j - window/2; y <= j + window/2; y++) {

// Each time that my loop statement encounter a pixel
// in my image he increament the total pixel value of the variable "count(type:
    integer)"
if (x >= 0 && x < img.width() && y >= 0 && y < img.height()) {
count++;
// and also for each type of pixel value my loop will enconter it will
// check its type weather it is Red, Green or Bleu chanel and increement it value in
     the array to have the Sum of each pixel value
// =+ is the Addition assignment in cpp

// sum of the pixel chanel red will be at the first position of my array
sum[0] += img.pixelColor(x,y).red();
```

```
// sum of the pixel chanel green will be at the second position of my array
sum[1] += img.pixelColor(x,y).green();
// sum of the pixel chanel blue will be at the third position of my array
sum[2] += img.pixelColor(x,y).blue();
}
}
}
// Compute pixel average value
// Replace pixel by average pixel value
for (int x = i - window/2; x <= i + window/2; x++) {
for (int y = j - window/2; y <= j + window/2; y++) {
if (x >= 0 && x < img.width() && y >= 0 && y < img.height()) {
imgFinal.setPixelColor(x,y,QColor(sum[0]/count, sum[1]/count, sum[2]/count));
}
}
}
}
}
// Save image
imgFinal.save(QString("teste.png"));
// My function " pixeliseFunction " will return the pixelised Image
return imgFinal;
}
```

mainwindow.cpp file

```


// Display my pixelised image
// The QPixmap class is use for painting the Image in the predefinded display widget
    .
ui->label_original->setPixmap(QPixmap::fromImage(image));
ui->label_original->adjustSize();

// Processing the Image pixelation and directing the image to its display
// widget to appear

// Processing image pixelisation
QImage img = imgProcess->pixeliseFunction(image, 20);
ui->label_final->setPixmap(QPixmap::fromImage(img));

// Ajusting image in my Display widjet to fit
ui->label_final->adjustSize();
```

Interface output

5 Project structure part3

- Create and describe the behaviour of the pixel art function.
- Define the process for finding the best match image in folder.
- Define a function for choosing the pixel cube size.
- Display and save the pixel art image.

5.1 Pixel Art rendering

5.1.1 Understanding

Pixel art process description

The pixelation process is defined as follow:
Inside the a function, a statement will be define to go over each pixel block in an image and replace them by their best match image in a set of images using the absolute difference summation method.

Summation method

Sum of Absolute Difference can be computed via obtaining the absolute difference among every pixel within the Source image(original block) and therefore the subsequent pixel within the block that is meant for the aim of comparison. It could be used as a measure to determine the similarity among image blocks.

5.1.2 Algorithm

Best match function

```
QImage imgprocess::bestMatch (std::vector<QImage> vecImages, std::vector<double>
    distance,
int window, QImage img_original) {

//Max summation of all the pixel into a square (window,window)
double finalDist = pow(window,2)*255*3;

//Block where will save the final square that has the best matches
QImage block (window, window, img_original.format());

for (int i = 0; i < vecImages.size(); i++) {
double distEuclidian = 0;
std::vector<int> rgb {0,0,0};

//For one square, calculates the summation of all the pixels
for (int x = 0; x < vecImages[i].width(); x++) {
for (int y = 0; y < vecImages[i].height(); y++) {
rgb[0] += vecImages[i].pixelColor(x,y).red();
rgb[1] += vecImages[i].pixelColor(x,y).green();
rgb[2] += vecImages[i].pixelColor(x,y).blue();
}
}
for (int z = 0; z < 3; z++) {
distEuclidian += fabs(rgb[z] - distance[z]);
}

if (distEuclidian < finalDist) {
finalDist = distEuclidian;
block = vecImages[i];
}
}
return block;
}
```

Pixelt art function

```
QImage imgprocess::pixelArt (QImage img_original, int window, std::vector<QImage>
    vecImages) {
QImage img_final = img_original.copy();

for (int i = 0; i < img_original.width(); i+=window) {
for (int j = 0; j < img_original.height(); j+=window) {

std::vector<double> rgb {0,0,0};

//Run into a square block on the original image with size (window, window)
for (int x = i; x < i + window; x++) {
for (int y = j; y < j + window; y++) {

//Verify if there's no pixel out the image
if (x >= 0 && x < img_original.width() &&
y >= 0 && y < img_original.height()) {

rgb[0] += img_original.pixelColor(x,y).red();
rgb[1] += img_original.pixelColor(x,y).green();
rgb[2] += img_original.pixelColor(x,y).blue();
}
}
}


QImage block = bestMatch(vecImages, rgb, window, img_original);

//Print into the new image (img_final)
for (int x = i; x < i + window; x++) {
for (int y = j; y < j + window; y++) {

if (x >= 0 && x < img_original.width() &&
y >= 0 && y < img_original.height()) {
img_final.setPixelColor(x, y,
QColor(block.pixelColor(x-i,y-j).red(),
block.pixelColor(x-i,y-j).green(),
block.pixelColor(x-i,y-j).blue()));
}
}
}
}
}

std::cout << "Finish" << std::endl;
return img_final;
}
```

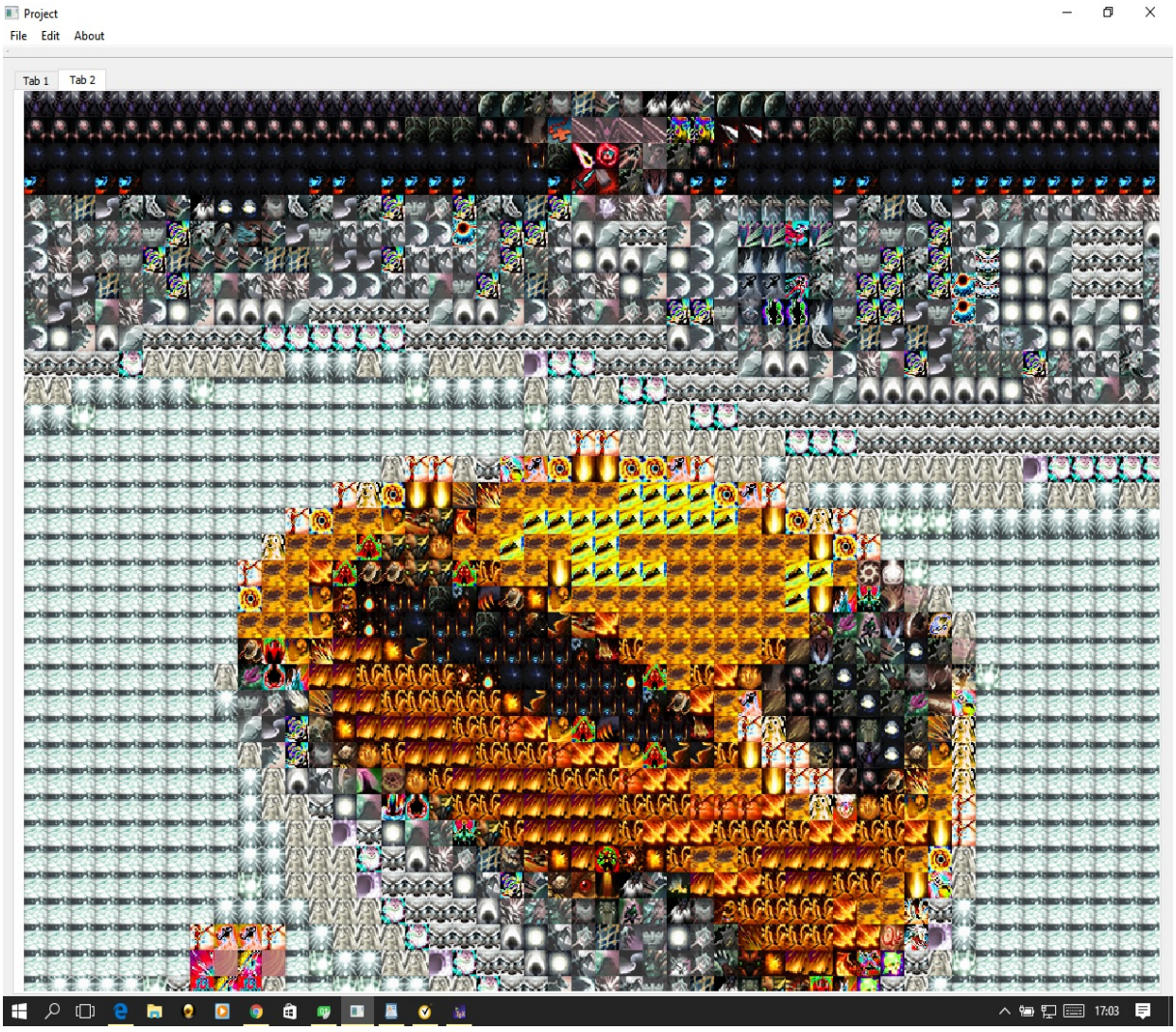
- Extract the mean colour of the sample image.
- Calculate the distance of the mean colour of the sample image and the mean colour of the original image.

Knowing the summation value we can compare and get the best match sample image.

mainwindow.cpp

```
QStringList imgPath = QFileDialog::getOpenFileNames(this, "Open");
std::vector<QImage> vecImages;
for (int i = 0; i < imgPath.size(); i++) {
vecImages.push_back(QImage(imgPath[i]));
}

img2 = imgProcess->pixelArt(img1, 20, vecImages);
ui->finaLabel->setPixmap(QPixmap::fromImage(img2).scaled(QSize(ui->finaLabel->width
    (),ui->finaLabel->height()),Qt::KeepAspectRatio));
ui->finaLabel->adjustSize();
```

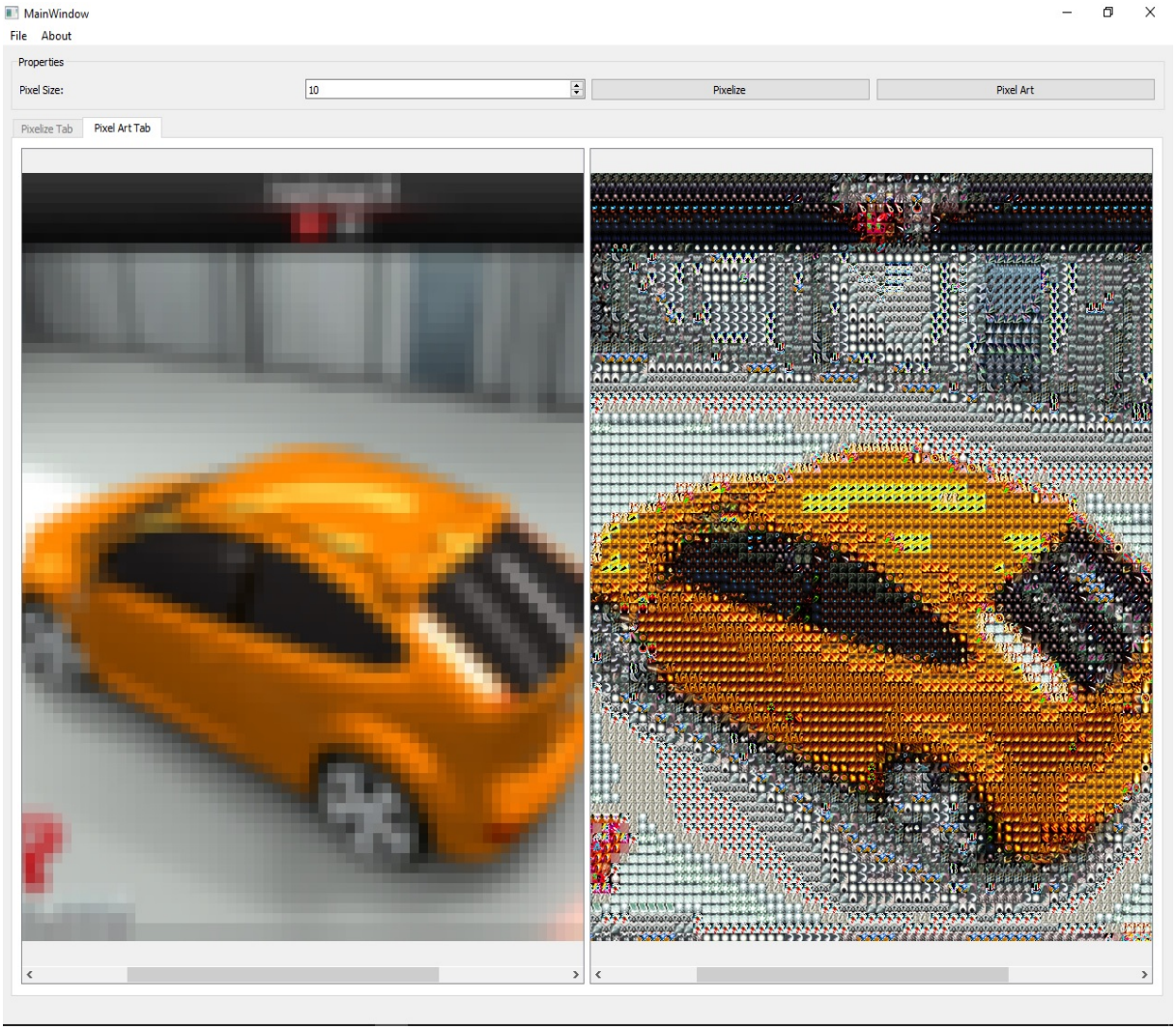
Interface output



6 Final program

After the understanding of the pixel art process. Few changes has been added such as:

- A spinBox widget to input pixel blocks size.
- A pixelisation button.
- A Pixel Art button.



Few changes

Apart from using the "Go to slot function " and redefining my pixelisation and my pixel art functions, the main change here is to add the spinBox to define the pixel block size and pass this value to my pixelisation function to set pixel cube size.

7 Conclusion

This project promotes the development of two pixel manipulation approaches: pixelize and pixel art. We presented information about how to pixelize an image and to create a pixel art. This program has not been fully optimize, the program takes sometime to run. An optimize version may be send later.

8 References

- https://github.com/agboliver
- //github.com/rafaelasouza/Pixelation
- http://en.cppreference.com/w/cpp/memory/c/malloc
- http://www.ijcaonline.org/archives/volume153/number10/swaroop-2016-ijca-912165.pdf
- http://stackoverflow.com/questions/8099591/variable-block-size-sum-of-absolute-difference-calculation-in-c
- https://www.codeproject.com/KB/audio-video/317974/filter.JPG
- Doctor Yohan Fougerolle/ Centre Universitaire Condorcet/ Yohan.Fougerolle@u-bourgogne.fr