# Qt Lab #2

Xiang WEI - IGD IP Paris

## Introduction

In this work, I completely finished a drawing application based on Qt toolkit by following all the instructions in Lab 2. The application has an interactive interface which allows users to draw multiple shapes in different colors, line styles and thickness, by clicking buttons or dragging sliders. Moreover, by clicking bottom buttons users can switch to editing part which allows users to select an existing shape and move, resize or change attributes of it. From the top toolbar, users can read or save the images and are required to double-check before quitting.

## 1. Create a new project

In this part, I created a class name "canvas" inherits from *QWidget* and used *setMinimumSize( )* method to enforce an adequate size. Then in the MainWindow class, I created pointer in canvas class and used *setCentralWidget(canvas)* to show it on the windows.

## 2. Draw a persistent line

In this part, I set two *QPoint* variables for saving the coordinates of start and end points from the mouse click. I override the *paintEvent( )* and 3 mouse related functions. In order to get a persistent line, I called *update( )* function in the 3 mouse functions.

## 3. Specify graphical attributes

In this part, I created 9 *QAction* and distributed them into 3 *QActionGroup*. Then I connected each group into their functions, where I used *setXXX( )* function to change the variables in canvas class. In the end I called *setPen( )* function in *paintEvent* based on those variables.

## 4. Draw other geometric shapes

In this part, I added another QActionGroup for shapes and used the return results in *paintEvent* to call different draw functions from the painter.

## 5. Draw multiple shapes

In this part, I created an abstract class for shape which has a virtual function called *"draw"* and several functions for setting start and end points and other attributes. Then I made different shapes as subclass and override the *draw* function. Later, in canvas class, I created a *QList<AbstractShape *>.* In *mousePressedEvent* a new object in different subclasses is created regarding the selected shape and was appended into the list. In the end, I went through the *list* and called the *draw* function for each element in the paintEvent function.

## 6. Edit shapes

In this part, I added another *QActionGroup* for editing existed shapes. I created another QList in QPainterPath type for detecting the selection. In the drawing mode, I kept the original code and also maintained the QPainterPath list.
Then for selecting part, I created a 4*4 rectangle around my mouse click and called the intersects function to get the index of the selected shape.
For moving, I get the distance between my mouse click and release and change the start and end point of the selected shape based on the index I got from the select part.
For scaling, I get the coordinate of the selected shape and modified them by a number that I'll get from my MainWindow / ui.

For changing attributes, I simply called *setPen* function which I declared in the abstract shape class.

## 7. Use QtDesigner

In this part, I dragged some layouts and widgets directly on the panel. And in MainWindow class, I added my canvas class to the *MainLayout* and active the ui. Also, I created several *QButtonGroup* and classified different radio buttons into groups. Later, I added two sliders to leave the users more freedom.

## 8. Ask for confirmation before exiting the program

In this part, I used the code from my first lab work based on *QMessageBox*.

## 9. (optional): Read and save shapes

In this part, I acquired the loading/saving path by using the same function in my first lab work based on *QFileDialog*. For reading, I used *QImage.load( )* function and sent the image to the canvas. Then call the *update( )* to show the image automatically. For saving, I called *QImage.save( )* function and linked it to the MainWindow.

## Conclusion

By following all the steps, I successfully created a simple drawing application by using qt designer. Although there are still some flaws ( for example: the reading image part, it will show the image with a white background ), I truly learnt a lot not only by reading the instruction files but also by debugging my own code. Thanks for giving us such a chance to do the lab and I'll try to perfect it later.