UNIVERSITY OF BURGUNDY

COMPUTER SCIENCE

PROJECT REPORT

A SIMPLE PAINT APPLICATION

Author:
Oluwafemi Paul OLALEYE

Supervisors:
Prof. Yohan FOUGEROLLE
Prof. Cansen JIANG

May 14, 2015

Contents

1 Introduction

The main goal of this project was to design a simple paint application in C++ using Qt. This report explains the main parts of the project listed below:

1. The Structure of the Project
    - The classes that were created
    - The connection and communication between classes
    - The internal structure and properties of the classes
2. The List and Properties of the most important functions
3. Introduction to the Apps' GUI

2 Project Structure

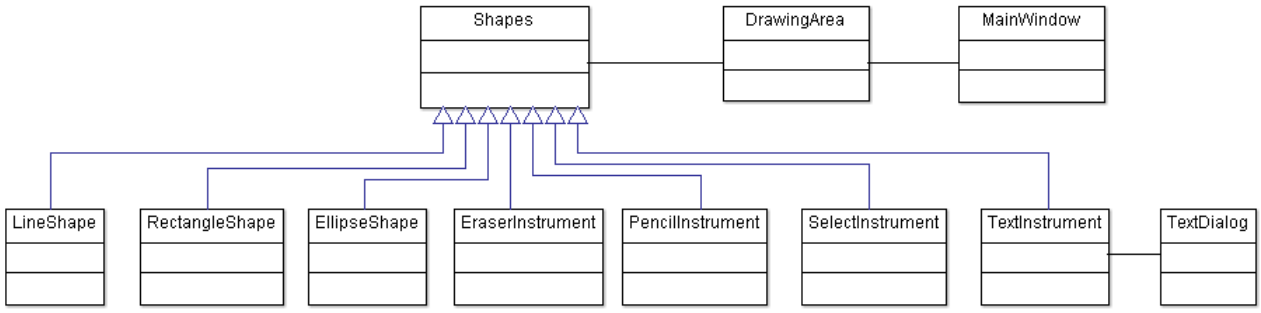

Figure 1: Class Diagram

The UML Class diagram above shows the relationship between the classes in the project. The Classes `lineShape`, `rectangleShape`, `ellipseShape`, `eraserInstrument`, `textInstrument`, `pencilInstrument`, `selectInstrument` are all child classes of the `Shapes` class.

NOTE: In this report, `Tools` is used to refer to a combination of `Shapes` and `Instruments` in general. The words `Shapes`, `Instruments` will be used for specific cases.

2.1 The `Shapes` Class

The `Shapes` class is the parent class for all Tools (Shapes and Instruments). The other classes are child classes of the `Shapes` class.

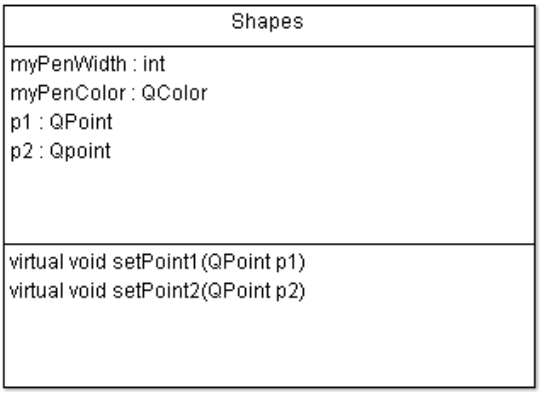

Figure 2: Shapes Class Diagram

2.2 The `DrawingArea` Class

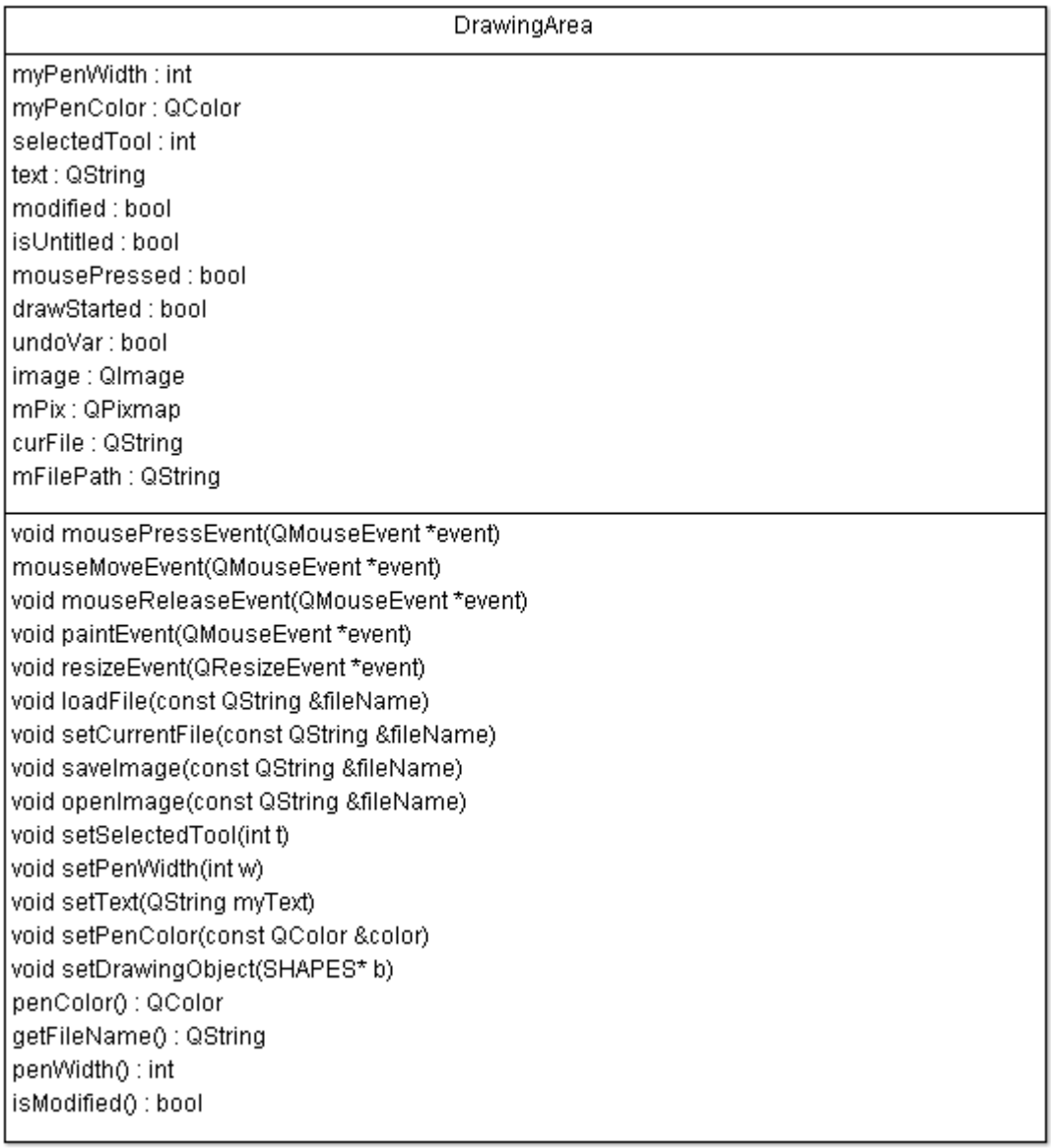

Figure 3: DrawingArea Class Diagram

The `DrawingArea` Class contains the `DrawingArea` Widget where painting is done. To use the `Tools` on the widget:

1. A Standard Library Template (STL) container `vector` was used to create a container to store shapes.

   ```
   vector<SHAPES*> myShapeVector;
   ```

   The container is identified by the name `myShapeVector`. A pointer to `Shapes` is passed as the `type` for the elements of the vector. Therefore, every element in the container is a child of the `Shapes` class.

2. A Pointer to `Shapes` identified by `myShape` was created which stores the specific shape to be drawn. For example to draw a Line, a pointer to `lineShape` is created and stored in `myShape`.

   ```
   SHAPES *myShape;
   LINESHAPE *line = new LINESHAPE;
   myShape = line;
   ```

This is done for every `Tool` whenever its about to be used.

2.3 The `MainWindow`

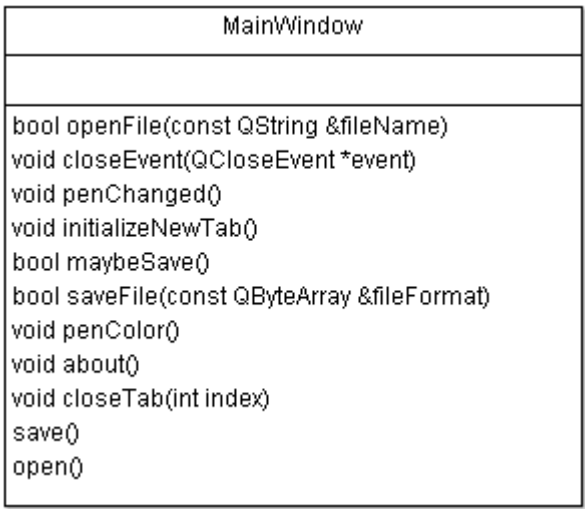

Figure 4: MainWindow Class Diagram

The main window provides a framework for building the application's user interface. It contains the menu, the buttons and the widget for drawing.

In its constructor, a pointer to the `DrawingArea` class is created to be used in different functions in the class. In the next section, we'll go into more details describing the most important functions in the `MainWindow` and `DrawingArea` classes.

3 List & Properties of Functions

3.1 Functions in DrawingArea Class

3.1.1 The mousePressEvent Function

The `mousePressEvent` is called when a mouse button is pressed while the mouse cursor is inside the widget, or when the widget has grabbed the mouse using `grabMouse()`. Pressing the mouse without releasing it is effectively the same as calling `grabMouse()`.

- The boolean variable `mousePressed` is set to `true`.
- The pointer `myShape` (from the parent class `Shapes`) is cast/converted and made a pointer to the child class of the Tool to be used. Hence, `myShape` now points to the child class and is created dynamically. This is done in the renowned `if ... else if` conditional statement to ensure that just one Tool is pointed to and used at every instance.
- The positions are set for the Tool to be used.

```
void DrawingArea::mousePressEvent(QMouseEvent* event)
{
    //Mouse is pressed for the first time
    mousePressed = true;

    /*Checking if the MyShape object is the MyLineShape
    type then create a new object of that type*/
    if(dynamic_cast<LINESHAPE*>(myShape))
    {
        myShape = new LINESHAPE;
```

```
    }
    else if(dynamic_cast<RECTANGLESHAPE*>(myShape))
    {
        myShape = new RECTANGLESHAPE;
    }
    else if(dynamic_cast<ELLIPSESHAPE*>(myShape))
    {
        myShape = new ELLIPSESHAPE;
    }
    else if(dynamic_cast<PENCILINSTRUMENT*>(myShape))
    {
        myShape = new PENCILINSTRUMENT;
    }
    else if(dynamic_cast<ERASERINSTRUMENT*>(myShape))
    {
        myShape = new ERASERINSTRUMENT;
    }
    else if(dynamic_cast<TEXTINSTRUMENT*>(myShape))
    {
        myShape = new TEXTINSTRUMENT;
    }

    //depending on Object type setPoint methods of
    //Different object is called
    myShape->setPoint1(event->pos());
    myShape->setPoint2(event->pos());

}
```

mousePressEvent.cpp

3.1.2 The mouseMoveEvent Function

The `mouseMoveEvent` Function is called whenever the mouse moves while a mouse button is held down. This is used for drag and drop operations.

- Used to set and reset the new position of the mouse to the second point again and again.
- Event is updated whenever a new position is set by moving the mouse.

```
void DrawingArea::mouseMoveEvent(QMouseEvent *event)
{

    //As mouse is moving set the second point again and again
    // and update continuously
    if(event->type() == QEvent::MouseMove){
        myShape->setPoint2(event->pos());
    }

    //it calls the paintEvent() function continuously
    update();

 }
```

mouseMoveEvent.cpp

3.1.3 The mouseReleaseEvent Function

The `mouseReleaseEvent` Function is called when a mouse button is released. The widget receives mouse release events when it has received the corresponding mouse press event. This means that if the mouse is pressed and dragged to another position inside the widget before release, the widget receives the release event.

- The boolean variable `mousePressed` is set to `false`.

- Updates one more time when mouse is released.

```
void DrawingArea::mouseReleaseEvent(QMouseEvent *event)
{
    //When mouse is released update for the one last time
    mousePressed = false;
    update();

}
```

mouseReleaseEvent.cpp

3.1.4 The paintEvent Function

A paint event is a request to repaint all or part of a widget.

- An object of `QPainter` named `painter` is created.
- `painter` sets the properties of the Pen.
- When mouse is pressed and moved, a reference to the `pixmap` is taken again and again and drawing/painting is done with the current tool in use.
- If `drawStarted` is `true` and `undoVar` is `false`, a new object of `QPainter` named `tempPainter` is created by taking a reference to the `QPixmap` object created. earlier. Then drawing/painting is done with the current tool in use and the `pixmap` is updated.

```
void DrawingArea::paintEvent(QPaintEvent *event)
{

    painter.begin(this);
    painter.setPen(QPen(myPenColor, myPenWidth, Qt::SolidLine, Qt::RoundCap, Qt::
    RoundJoin));


    //When the mouse is pressed
    if(mousePressed){
        // we are taking QPixmap reference again and again
        //on mouse move and drawing a line again and again
        //hence the painter view has a feeling of dynamic drawing

        painter.drawPixmap(0,0,mPix);

        /*Depending on myShape type object from those shapes are called
            for painting on the painter using appropriate method,
            benefit is we don't need to know Shape type beforehand */
        if(dynamic_cast<LINESHAPE*>(myShape)){
            painter.drawLine(dynamic_cast<LINESHAPE*>(myShape)->qline);
        }
        else if(dynamic_cast<RECTANGLESHAPE*>(myShape))
        {
            painter.drawRect(dynamic_cast<RECTANGLESHAPE*>(myShape)->qRect);
        }
        else if(dynamic_cast<TEXTINSTRUMENT*>(myShape))
        {
            painter.setPen(QPen(myPenColor, myPenWidth, Qt::DashLine, Qt::RoundCap,
    Qt::RoundJoin));
            painter.setBackgroundMode(Qt::TransparentMode);
            //painter.drawRect(dynamic_cast<TEXTINSTRUMENT*>(myShape)->qRect);
            //painter.drawText(dynamic_cast<TEXTINSTRUMENT*>(myShape)->qRect, Qt::
    AlignCenter, "Great");
            //painter.drawText(dynamic_cast<TEXTINSTRUMENT*>(myShape)->qRect, Qt::
    AlignCenter, text);
```

```
            painter.drawText(dynamic_cast<TEXTINSTRUMENT*>(myShape)->qRect, Qt::
    AlignCenter, text);
        }
        else if(dynamic_cast<PENCILINSTRUMENT*>(myShape)){
            QPainter tPainter(&mPix);
            tPainter.setPen(QPen(myPenColor, myPenWidth, Qt::SolidLine, Qt::RoundCap,
     Qt::RoundJoin));
            tPainter.drawLine(dynamic_cast<PENCILINSTRUMENT*>(myShape)->qline.p1(),
    dynamic_cast<PENCILINSTRUMENT*>(myShape)->qline.p2());

            dynamic_cast<PENCILINSTRUMENT*>(myShape)->setPoint1(dynamic_cast<
    PENCILINSTRUMENT*>(myShape)->qline.p2());
        }
        else if(dynamic_cast<ELLIPSESHAPE*>(myShape))
        {
            painter.drawEllipse(dynamic_cast<ELLIPSESHAPE*>(myShape)->qEllipse);
        }
        else if(dynamic_cast<ERASERINSTRUMENT*>(myShape))
        {

            QPainter tPainter(&mPix);
            tPainter.setPen(QPen(Qt::white, myPenWidth, Qt::SolidLine, Qt::RoundCap,
    Qt::RoundJoin));
            tPainter.drawLine(dynamic_cast<ERASERINSTRUMENT*>(myShape)->qEraser.p1(),
     dynamic_cast<ERASERINSTRUMENT*>(myShape)->qEraser.p2());

            dynamic_cast<ERASERINSTRUMENT*>(myShape)->setPoint1(dynamic_cast<
    ERASERINSTRUMENT*>(myShape)->qEraser.p2());

        }
        else if(dynamic_cast<SELECTINSTRUMENT*>(myShape))
        {
            painter.setPen(QPen(Qt::blue, 1, Qt::DashLine, Qt::RoundCap, Qt::
    RoundJoin));
            painter.drawRect(dynamic_cast<SELECTINSTRUMENT*>(myShape)->qRect);
        }

        drawStarted = true;
        undoVar = false;
    }
    else if (drawStarted && !undoVar){
        // It created a QPainter object by taking  a reference
        // to the QPixmap object created earlier, then draws a line
        // using that object, then sets the earlier painter object
        // with the newly modified QPixmap object
        QPainter tempPainter(&mPix);
        tempPainter.setPen(QPen(myPenColor, myPenWidth, Qt::SolidLine, Qt::RoundCap,
     Qt::RoundJoin));

        if(dynamic_cast<RECTANGLESHAPE*>(myShape))
        {
            tempPainter.drawRect(dynamic_cast<RECTANGLESHAPE*>(myShape)->qRect);
        }
        else if(dynamic_cast<LINESHAPE*>(myShape))
        {
            tempPainter.drawLine(dynamic_cast<LINESHAPE*>(myShape)->qline);
        }
        else if(dynamic_cast<TEXTINSTRUMENT*>(myShape))
        {
            tempPainter.setPen(QPen(myPenColor, myPenWidth, Qt::DashLine, Qt::
    RoundCap, Qt::RoundJoin));
            //tempPainter.drawRect(dynamic_cast<TEXTINSTRUMENT*>(myShape)->qRect);
```

```
            //tempPainter.drawText(dynamic_cast<TEXTINSTRUMENT*>(myShape)->qRect, Qt
::AlignCenter, "Great");
            tempPainter.drawText(dynamic_cast<TEXTINSTRUMENT*>(myShape)->qRect, Qt::
AlignCenter, text);
        }
        else if(dynamic_cast<PENCILINSTRUMENT*>(myShape)){
            tempPainter.drawLine(dynamic_cast<PENCILINSTRUMENT*>(myShape)->qline);
        }
        else if(dynamic_cast<ELLIPSESHAPE*>(myShape)){
            tempPainter.drawEllipse(dynamic_cast<ELLIPSESHAPE*>(myShape)->qEllipse);
        }
        else if(dynamic_cast<SELECTINSTRUMENT*>(myShape))
        {
            tempPainter.setPen(QPen(Qt::blue, 1, Qt::DashLine, Qt::RoundCap, Qt::
RoundJoin));
            tempPainter.drawRect(dynamic_cast<SELECTINSTRUMENT*>(myShape)->qRect);
        }
        else if(dynamic_cast<ERASERINSTRUMENT*>(myShape)){
            tempPainter.setPen(QPen(Qt::white, myPenWidth, Qt::SolidLine, Qt::
RoundCap, Qt::RoundJoin));
            tempPainter.drawLine(dynamic_cast<ERASERINSTRUMENT*>(myShape)->qEraser);
        }

        //saving shapes in a vector
        myShapeVector.push_back((myShape));

        painter.drawPixmap(0,0,mPix);
        undoVar = false;
    }

    else if (undoVar){

        if(!myShapeVector.empty())
        {
        myShapeVector.pop_back();

        mPix.fill(Qt::white);
        QPainter tempPainter(&mPix);
        tempPainter.setPen(QPen(myPenColor, myPenWidth, Qt::SolidLine, Qt::RoundCap,
 Qt::RoundJoin));

        unsigned int size = myShapeVector.size();
        qDebug() << size;
        for (unsigned int i = 0; i < size; i++)
        {
            if(dynamic_cast<RECTANGLESHAPE*>(myShapeVector[i]))
            {
                tempPainter.drawRect(dynamic_cast<RECTANGLESHAPE*>(myShapeVector[i])
->qRect);
            }
            else if(dynamic_cast<LINESHAPE*>(myShapeVector[i]))
            {
                tempPainter.drawLine(dynamic_cast<LINESHAPE*>(myShapeVector[i])->
qline);
            }
            else if(dynamic_cast<ELLIPSESHAPE*>(myShapeVector[i])){
                tempPainter.drawEllipse(dynamic_cast<ELLIPSESHAPE*>(myShapeVector[i
])->qEllipse);
            }
            else if(dynamic_cast<TEXTINSTRUMENT*>(myShapeVector[i]))
            {
                tempPainter.setPen(QPen(myPenColor, myPenWidth, Qt::DashLine, Qt::
RoundCap, Qt::RoundJoin));
```

```
                //tempPainter.drawRect(dynamic_cast<TEXTINSTRUMENT*>(myShape)->qRect
    );
                //tempPainter.drawText(dynamic_cast<TEXTINSTRUMENT*>(myShapeVector[i
    ])->qRect, Qt::AlignCenter, "Great");
                tempPainter.drawText(dynamic_cast<TEXTINSTRUMENT*>(myShapeVector[i])
    ->qRect, Qt::AlignCenter, text);
            }
            else if(dynamic_cast<PENCILINSTRUMENT*>(myShapeVector[i]))
            {
                tempPainter.drawLine(dynamic_cast<PENCILINSTRUMENT*>(myShapeVector[i
    ])->qline) ; //.p1(), dynamic_cast<MyPenShape*>(myShapeVector[i])->qline.p2());

            }
            else if(dynamic_cast<SELECTINSTRUMENT*>(myShapeVector[i]))
            {
                tempPainter.setPen(QPen(Qt::blue, 1, Qt::DashLine, Qt::RoundCap, Qt
    ::RoundJoin));
                tempPainter.drawRect(dynamic_cast<SELECTINSTRUMENT*>(myShapeVector[i
    ])->qRect);
            }
            else if(dynamic_cast<ERASERINSTRUMENT*>(myShapeVector[i]))
            {
                tempPainter.setPen(QPen(Qt::white, myPenWidth, Qt::SolidLine, Qt::
    RoundCap, Qt::RoundJoin));
                tempPainter.drawLine(dynamic_cast<ERASERINSTRUMENT*>(myShapeVector[i
    ])->qEraser);
            }
        }
        painter.drawPixmap(0,0,mPix);
        }
    }

    painter.end();

}
```

paintEvent.cpp

3.1.5 Other Functions

The `setSelectedTool()`, `setPenWidth()`, `setPenColor()`, `setText()`, `setCurrentFile()`, `setDrawingObject()`, `loadFile()`, `saveImage()` and `openImage()` Functions

The operations of these functions are exactly as their names imply.

- `setSelectedTool()` sets the Tool (`selectedTool`) in use.
- `setPenWidth()` sets the width of the Pen (`penWidth`) based on the value in the `PenWidthSpinBox`.
- `setPenColor()` sets the color of the Pen (`penColor`) based on the color selected in the color palette.
- `setText()` sets the text entered in the `textDialog` to a variable `text` which is painted on the widget with `drawText` function.
- `setCurrentFile()` sets the current working widget.
- `setDrawingObject()` sets the current object to be painted on the widget.
- `loadFile()` loads a specific file to be opened in the widget.
- `saveImage()` saves an Image.
- `openImage()` opens an Image.

4 Brief Explanation of GUI

The GUI is very easy to understand because its user friendly. For every button clicked or action triggered, the cursor changes and you just click and drag the mouse on any part of the `DrawingArea` and the widget will be painted.

For the Text button, when clicked, a text dialog pops up where you can enter your text. When through entering text, click the `Ok` button. It appears like nothing happens but the text has been saved into a String variable which will be painted on the widget when you click and drag. So, after clicking `Ok`, move the text dialog away from the widget and the click and drag the mouse on the widget. The text you entered is painted on the widget.

5 Conclusion

The language used for the project development was C++ and the software used was Qt.

C++ was chosen because it was the main objective of the module Computer Science (whose requirement is the completion of this project).
The following were emphasized during the course lectures of the module:

- Understanding the principles of object oriented programming.
- Ability to design and implement graphical apps using Qt.
- Being Comfortable with the basics of Qt/C++ as a foundation to implementing more complicated apps in the nearest future.

Qt was chosen because:

- It uses standard C++ with extensions including signals and slots that simplifies handling of events.
- It supports many compilers, including the GCC C++ compiler and the Visual Studio suite.

More details about Qt can be found on the Qt website `http://www.qt.io` and the Qt documentation webpage `http://www.doc.qt.io`.

In conclusion, this project has really helped me to learn the basic concepts of OOP in C++, understand the basic relationship between classes in C++ using Qt, design and implement an App with good GUI.
The project was challenging at first because its was my first project in C++ and my first experience with Qt. Understanding the basics took some time and my progress on the project was slow but with the help of the instructors and my colleagues, it became a reality.
I can say now that I have the basic understanding I need for more complex and advanced assignments and the goal of the module was achieved.

6 References

The references listed below were the major references used for this project.

- Qt website `http://www.qt.io` and documentation webpage `http://www.doc.qt.io`.
- Stack Overflow `http://www.stackoverflow.com`.
- Qt Forum `https://forum.qt.io/`
- `EasyPaint` App `https://github.com/Gr1N/EasyPaint`
- Qt `BasicDrawing` App `http://doc.qt.io/qt-5/qtwidgets-painting-basicdrawing-example.html` and of course
- Prof Yohan Fougerolle, IUT Le Creusot, France
- Prof Cansen Jiang, IUT Le Creusot, France

7 Screenshots of the paintApp

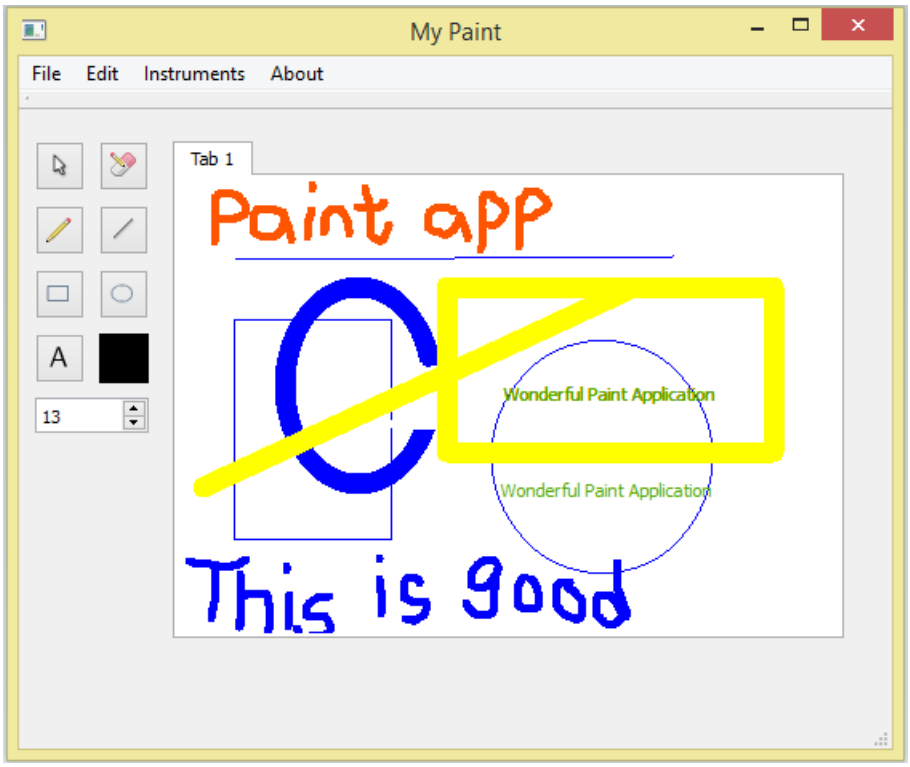

Figure 5: Screenshot

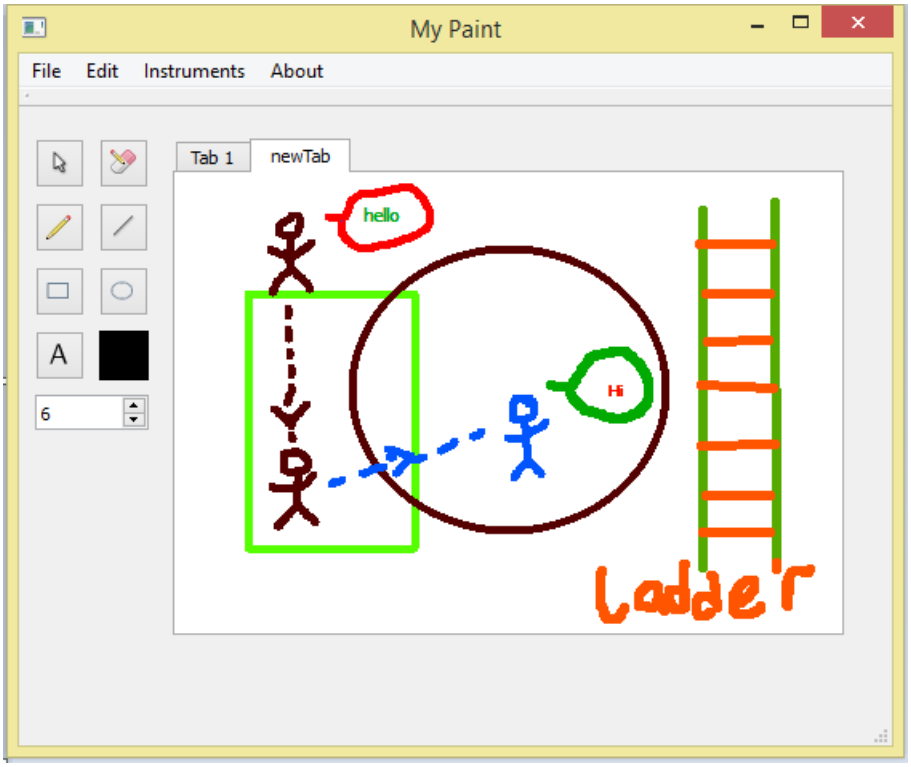

Figure 6: Screenshot 2

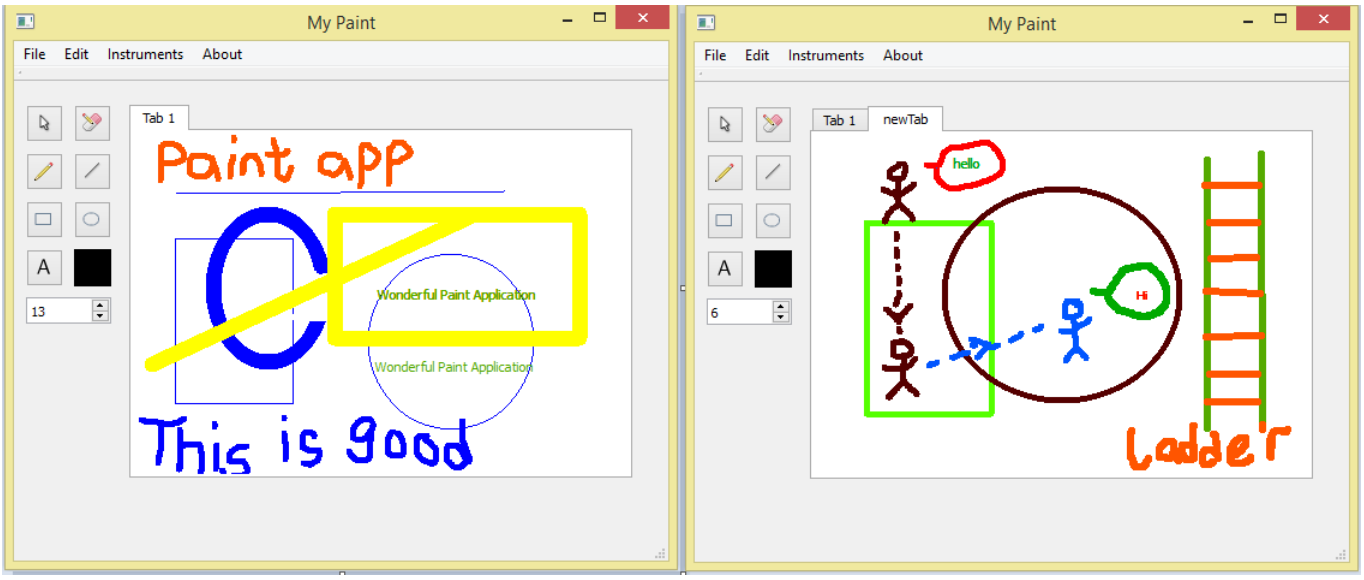

Figure 7: Screenshot 3