*Florida International University*

*School of Computing and Information Sciences*

Software Engineering Focus

Final Deliverable

Project Title: Interactive Paint

**Team Members:** Daniel Mederos, Jim Louro, Camilo Riviere

**Product Owner(s)**: Francisco Ortega

**Mentor(s)**: Francisco Ortega

**Instructor**: Masoud Sadjadi

The MIT License (MIT)

Copyright (c) *2016 Florida International University*

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

**Abstract**

Interactive Paint is a Windows only Desktop application as of now, but could work on macOS and Linux if QT is installed. It allows users to paint on a canvas at the same time using different tools. It's a cross platform application that can be run on systems that have QT installed and its base code is modular which allows for easier maintenance, adding and removing features more efficiently and allows for better testing.

**Table of Contents**

**Introduction** 4

[Current System](#_64964pmr4uid) 4

[Interactive Paint](#_d7i51wswuk1g) 4

[Purpose of New System](#_ew0c6ybj4f0) 4

[Interactive Paint](#_9816eeksp7uo)

The goal behind our system is to recreate Interactive Paint without an excessive amount of dependencies and have it run in a cross platform environment while allowing key paint features and multimodal features. 4

[**User Stories**](#_9fhbfg31oq) **5**

[Implemented User Stories](#_85zbqpoue7q9) 5

[Project Plan](#_eulz7du8g5k7) 8

[Sprint Plan](#_gn7qonmw75g3) 9

[Sprint 2](#_2j5jwp4310jk) 9

[Sprint 3](#_pc1id51hlryt) 9

[User Story #307 - Create an Ellipse Shape](#_o57q4uk5okm9) 9

[Acceptance Criteria:](#_n3wky3z3275x) 10

[Acceptance Criteria:](#_55a28xx404bc) 10

[Related Tasks:](#_b1q9mof3p7j1) 10

[Sprint 4](#_rxu513uhp9v6) 11

[Sprint 5](#_md7ydiuzhwhk) 12

[Sprint 6](#_orrv0k4b6w2m) 13

[Deployment](#_dj6j76yd5lzr) 14

[Design Patterns](#_a4m2lamu8ybb) 14

[**System Design**](#_6kxvhv7xecrc) **15**

[Deployment](#_cs78agqo12n4) 46

[Design Patterns](#_3on7t2yakimz) 47

[**System Validation**](#_jatmx6oxaldl) **48**

[**Glossary**](#_s36ek2w69y4q) **53**

[**Appendix**](#_82zowgeb3aev) **54**

[Appendix A - UML Diagrams](#_85k3c07yy3l3) 54

Appendix B - User Interface Design 68

[Appendix C - Sprint Review Reports](#_3axz8z8bxape) 75

**References 80**

# 

# 

**Introduction**

This document shows the design and architecture of Interactive Paint. This will cover our current system design and the purpose behind it. This also includes the user stories that we had to guide us throughout the implementation of this application. We cover the hardware and software that was used to make this application. Finally we show the test cases we used to validate our system.

## **Current System**

### **Interactive Paint**

Interactive Paint’s system is similar to the previous iteration but we had to start from scratch using a new modern framework and incorporating basic tools for drawing while also being modular and cross platform. This version allows for mouse and touch screen and up to 10 users at the same time. Future iterations will be able to add more input devices like the microsoft kinect, intel realSense and Motion Leap.

## **Purpose of New System**

The final purpose of our new system was to separate into individualized groups focusing on one of 4 main aspects: Painting Software, Modular code base, Research, and Multi Users, at the request of our product owner. Much of our functions are fully integrated, but we were not able to add any of the complex input devices.

### **Interactive Paint**

The goal behind our system is to be able to reuse the components that handle input devices. A future iteration of Interactive Paint should integrate Vulkan and virtual reality so we want the developers spending more time on building a fully fleshed out application than trying to make devices like the RealSense and Kinect work properly.

# **User Stories**

The following section provides the detailed user stories that were implemented in this iteration of the Interactive Paint project. These user stories were approved by Dr. Francisco Ortega and served as the basis for the implementation of the project’s features.

## **Implemented User Stories**

**User Story #268 - Learning C++**

As a developer, I would like to be able to code efficiently and effectively for this project in C++

**User Story #272 - Learning Mingle**

As a developer, I would like to learn how to effortlessly use Mingle to better be productive

**User Story #280 - Learning User Stories**

As a developer, I would like to create user stories for future sprints

**User Story #315 - Learning Observer Pattern**

As a developer, I would like to learn how to create Observer patterns in C++

**User Story #292 - Sample GUI**

As a developer I want to use the QT framework to design a basic GUI to test the C++ and QT integration

**User Story #298 - Cross Platform apps**

As a developer, I must create a window in Visual Studios that will serve as a .pro solution so we can compile the QT code in different operating systems

**User Story #303 - Test QT plug in on Visual Studios**

As a developer, I want to install and test the QT plug in so we can use all the QT tools in creating a multi model paint application

**User Story #304 - Create Main Window Source**

As a developer, I want to create the source code for our main window that will serve as the GUI

**User Story #305 - Create Main Window Header**

As a developer, I want to create the header code for our main window that will serve as the GUI

**User Story #306 - Create Rectangle Shape**

As a developer, I want to create a new shape (Rectangle) for the paint application

**User Story #307 - Create Ellipse Shape**

As a developer, I want to create a new shape (Ellipse) for the paint application

**User Story #316 - Implement Thread Safe Observer Pattern in Java**

As a developer, I want to create a basic example of the Thread Safe Observer Pattern in java

**User Story #332 - Create a video for sprint 3**

As a developer, I want to create a video to demonstrate all of the use cases that were done in Sprint 3, this video includes information from Jim and Daniels use cases as we worked together t o start the project from scratch

**User Story #319 - Implement a brush to draw any movement made by the user**

As a developer, I want to give the users the ability to scribble on the canvas and have the paintbrush record it

**User Story #320 - Implement a line brush to read in mouse movement**

As a developer I want to implement a class that allows the user to freehand draw on the painting application   
  
**User Story #326 - Implement an Eraser to eliminate the drawing on the canvas**

As a developer I want to give the user the option to erase any section of the drawing they originally made  
  
**User Story #327 - Implement a Fill**

As a developer, I want to fill the area within an enclosed drawing and change the color with a selected color.

**User Story #328 - Implement Observer Pattern in C++**

By implementing the Observer pattern in C++ we can gain an understand of

the Observer pattern in C++ code and eventually implement a custom thread safe Observer

pattern using C++.

**User Story #329 - Complete rough draft of research paper**

As a developer, I want to complete a rough draft of the research paper for the multi-modal framework

**User Story #347 - Create an Undo feature for paint**

As a developer, I want to create a button that allows the user remove the last action that was made

**User Story #347 - Create an Undo feature for paint**

As a developer, I want to create a button that allows the user remove the last action that was made

**User Story #348 - Create a Redo feature for paint**

As a developer, I want to create a button that allows the user to recreate the last shape that has been modified

**User Story #359 - Create MultiModal mode for multiple pen touches**

As a developer, I want to create a mode in the Paint application where we can accept multiple inputs from the user on the same canvas at the same time

**User Story #359 - Create MultiModal mode for multiple pen touches**

As a developer, I want to create a mode in the Paint application where we can accept multiple inputs from the user on the same canvas at the same time

**User Story #363 - Create Main Menu**

As a developer, I would like to be able to create a Main Menu to start the Single User canvas or Multi User canvas

**User Story #366 - Jim Louro’s Poster**

<https://docs.google.com/presentation/d/1Zi4aqoXipblbMms6ZNclsVbndJzva5a9GWG8MXdD7BY/edit#slide=id.p4>

**User Story #363 - Daniel Mederos’s Poster**

<https://docs.google.com/presentation/d/1ZypQO3ytsXc8MQxMEWH3HhjyjVmuTkTwFlyFg-Ada8g/edit>

**Project Plan**

This section describes the planning that went into the realization of this project. This project incorporated the agile development techniques and as such required the sprints to be planned. These sprint planning are detailed in the section. This section also describes the components, both software and hardware, chosen for this project.

## **Hardware and Software Resources**

The following is a list of all hardware and software resources that were used in this project:

**Hardware:**

Multi-touch Display - A screen that recognizes up to 10 simultaneous touches.

Multi core processor on a computer to handle the GUI and touch events

**Software:**

C++ - The application was built using C++17.

Visual Studio - Used as the environment for development.

Cinder - A library that abstracts OpenGL calls.

Qt any version from 4.8 to the latest

## 

## 

## 

## 

## 

## **Sprint Plan**

### **Sprint 2**

**User Story #268 - Learning C++**

As a developer, I would like to be able to code efficiently and effectively for this project in C++

**Acceptance Criteria:**

1. To be able to build and compile C++ programs

2. To be able to incorporate C++ threads into the programs

3. To be able to implement and design algorithms in C++

**User Story #271 - Learn STL and basic Data structures in C++**

As a developer, I would like to learn STL and basic Data structures in C++

**Acceptance Criteria:**

1. To be able to type efficient code using STL

2. To be able to design better Object Oriented designs

3. To be able to use data structures in the program

**User Story #284 - Review C++**

As a developer, I want to review C++ in order to make sure I know how to leverage the language in my favor.

**Acceptance Criteria:**

1. Feel comfortable with C++

2. Feel comfortable with the toolchain (on at least Linux or Windows)

3. Get a better understanding of the package manager

**User Story #282 - Test Software**

As a developer, I want the hook up the current system and test it to see if it compiles.

### **Sprint 3**

### **User Story #307 - Create an Ellipse Shape**

As a Developer I want to create a new shape for the paint application to give the user better functionality.

**Acceptance Criteria:**

1. Must use QT Widgets
2. Must be able to draw different size rectangles and take in input from mouse or touch screen or surface pro pen.

**User Story #304 - Create Main Window**

As a Developer I want to create the source code for our main window that will serve as the GUI

### **Acceptance Criteria:**

1. Must be cross platform using all the QT features
2. Must include options to Draw different shapes
3. Must include ability to save and reopen existing shapes

**User Story #306 - Create Rectangle Shape**

As a Developer I want to create a new shape for the paint application

### **Acceptance Criteria:**

1. Must use QT Widgets
2. Must be able to draw different size rectangles and take in input from mouse or touch screen or surface pro pen.

### **Related Tasks:**

User story #333

User story #311

User story #308

**User Story #298 – Cross Platform apps**

As a developer I must create a window in Visual Studios that will serve as a .pro solution so we can compile the QT code in different operating systems.

**Acceptance Criteria:**

1. It must have QT Widgets to make create a successful UI

2. It must include .pro extension on the main window.

3. Must use QT 5 libraries and plugin with Visual studios to have cross platform compatibility.

**Related Task:**

User story #299

User story #300

User story #301

User story #302

**User Story #292 – Sample GUI**

As a developer, I want to use the QT framework to design a basic GUI to test the C++ and QT integration.

**Acceptance Criteria:**

1. The Gui code must compile and run

2. The sample gui should work with different touch inputs

3. The GUI must include at least one QT Widget

**Related Task:**

User story #293

User story #294

User story #295

User story #296

User story #297

User story #334

**User Story #292 – Test Qt plug in on Visual Studios**

As a developer, I want to install and test the QT plug in so we can use all the QT tools in creating a multimodal paint application.

**Acceptance Criteria:**

1. The test code must compile using QT’s libraries and Widgets

2. The test code must be viewed in a window with sample GUI requirements

3. The test code should run cross platform

**Related Tasks:**

User story #309

User story #310

User story #340

User story #344

### **Sprint 4**

**User Story #319 – Implement a brush to draw any movement made by the user**

As a developer, I want to give the users the ability to scribble on the canvas and have the paintbrush record it.

**Acceptance Criteria:**

1. It should have the least amount of dependencies so it extends only the needed classes (QT)

**Related Tasks:**

User story # 346

User story # 341

User story # 331

User story # 323

User story # 322

User story # 321

**User Story #320 - Implement a line brush to read in mouse movement\**

As a developer, I want to implement a class that allows the user to free hand draw on the painting application.

**Acceptance Criteria:**

1. Must use Qt Widgets

2. Must not depend on other classes only Qt so it can be extended to host multiple input

**Related Task:**

User story # 335

User story # 325

User story # 324

**User Story #326 - Implement an Eraser to eliminate the drawing on the canvas**

As a developer, I want to give the user the option to erase any section of the drawing they originally made.

**Acceptance Criteria:**

1. Must remove only the specific pixels that the eraser has gone over

2. Must be able to work above any image under it.

**Related Tasks:**

User story #336

User story #337

**User Story #327 - Implement a Fill that measures the area within an enclosed drawing and change the color in that area**

As a developer, I want to fill the area in the shape created with a selected color.

**Acceptance Criteria:**

1. Must measure the area inside the shape

2. Fill should change the color only inside the lines of the shape

**Related Tasks:**

User story #345

User story #342

### **Sprint 5**

**User Story #347 – Create an Undo Feature for paint**

As a developer, I want to give the users the ability to remove the last shape that was made.

**Acceptance Criteria:**

1. Must be able to remove multiple shapes yet only 1 shape at a time (once per click)

2. Must keep track of all the shapes that were removed in order LIFO

**Related Tasks:**

User story # 358

User story # 356

User story # 349

**User Story #348 - Create an Redo Feature for paint**

As a developer, I want to give the users the ability to recreate the last shape that has been modified.

**Acceptance Criteria:**

1. Fully recreates the last modified shape and so on in the reverse order of how they were undone.

2. Must keep track of all the shapes undone not just the ones recreated.

**Related Task:**

User story # 347

User story # 354

User story # 353

User story # 352

### **Sprint 6**

**User Story #359 – Create Multimodal mode for multiple Pen Inputs**

As a developer, I want to create a mode in the Paint application where we can accept multiple inputs from the user on the same canvas at the same time.

**Acceptance Criteria:**

1. The application must read multiple inputs and treat them all the same way.

2. The mode must have the same file options as the single mode GUI (Save, & Open)

3. The mode must allow the user to clear the canvas and restart the painting

4. The mode must allow the user to begin different touch points at the same and at various times without causing a delay or an issue with the painting style.

**Related Tasks:**

User story # 362

User story # 361

User story # 360

**User Story #363 - Create a MainMenu**

As a developer, I want to create a Main Menu to start the Single user or Multi user Canvas.

## 

## 

## 

## 

## 

## **Deployment**

Currently each product deploys with a .Pro solution in Visual studios or Qt creator(Qt editor and compiler). The default is not automatically set for Visuo Studios so the Plug in for Qt is required in Visual Studios then you must select to open a Qt project ending with .Pro to build the code in Visual Studios

## **Design Patterns**

**Actor Model** - Components are independent actors in an application, the end user will chose to have a Single mode with 1 input and 1 actor or multi mode with up to 10 actors(assuming one per brush).

**Object Model -** In order to avoid a complex hierarchy we separated the models into two forms a multi and a single where the File features(Save & Open), resize, Exit and minimize all perform a basic operation in both forms. In Single mode the canvas has options to change the brush style by implementing a shape class that includes but is not limited to having a rectangle ellipse free dawr line and an eraser. Along with the option to fill within an image or section and to change the brushes size and color. Single also keeps track of the shapes created in a FIFO order to allow redo and undo features. While Multi has separate event handlers that are always listening to new touch events to create a new brush on the canvas for the drawing. Multi also gives you the options to clear the canvas of all drawings. Seeing as how multiple brushes will have full access to the canvas at every point in time it would be very complicated to implement various styles of pens efficiently in short time or to have a FIFO order at all with the design.

# **System Design**

This section contains information on the design decisions that went into Interactive Paint. The architecture patterns are outlined and explained. The entire system is shown in a package diagram and the subsystems are explained. Finally, the design patterns used in the project are discussed.

For all aspects of this project, we decided to implement a modular design, so that every component of the program could be easily implemented or removed, maintained and tested.

Below are all the modular components of the program.

**Main Class**

#include "mainwindow.h"

#include "mainwindow2.h"

#include <QPushButton>

#include <QHBoxLayout>

#include <QMessageBox>

#include <QApplication>

#include <QInputDialog>

#include <QString>

int main(int argc, char \*argv[])

{

QApplication app(argc, argv);

/\*

bool ok;

QString text = QInputDialog::getText(NULL, "Interactive Paint", "1 for Single | 2 for Multi", QLineEdit::Normal, QString::null, &ok);

if ( ok && !text.isEmpty() ) {

// user entered something and pressed OK

} else {

// user entered nothing or pressed Cancel

}

if(text == 1){

Paint::MainWindow w;

w.show();

}

if(text == 2){

MainWindow window;

window.show();

}

\*/

Paint::MainWindow w;

w.show();

return app.exec();

}

**Command.cpp**

#include "command.h"

#include "document.h"

namespace Paint {

/\* Needed for merging \*/

enum CommandId { ResizeCommandId = 1 };

ShapeCommand::ShapeCommand(QWidget \*doc, QImage \*image, std::unique\_ptr<Shape> &&s) : doc(doc), image(image), undoImage(image->copy(s->rect())), shape(std::move(s))

{

}

void ShapeCommand::undo()

{

const QRect rect = shape->rect();

QPainter painter(image);

painter.drawImage(rect, undoImage);

doc->update(rect);

}

void ShapeCommand::redo()

{

QPainter painter(image);

shape->draw(painter);

doc->update(shape->rect());

}

FlipCommand::FlipCommand(QWidget \*doc, QImage \*image, bool horizontal, bool vertical) : doc(doc), image(image), horizontal(horizontal), vertical(vertical)

{

}

void FlipCommand::undo()

{

redo();

}

void FlipCommand::redo()

{

\*image = image->mirrored(horizontal, vertical);

doc->update();

}

ResizeCommand::ResizeCommand(QWidget \*doc, QImage \*image, const QSize &size) : doc(doc), image(image), oldSize(image->size()), newSize(size)

{

}

int ResizeCommand::id() const

{

return ResizeCommandId;

}

bool ResizeCommand::mergeWith(const QUndoCommand \*command)

{

if (command->id() != id())

return false;

newSize = static\_cast<const ResizeCommand\*>(command)->newSize; //Update the size witht he latest resize command

return true;

}

void ResizeCommand::undo()

{

\*image = image->copy(0, 0, oldSize.width(), oldSize.height());

doc->update();

}

void ResizeCommand::redo()

{

if (image->size() == newSize) {

return;

}

QImage newImage(newSize, QImage::Format\_RGB32);

newImage.fill(Qt::white);

QPainter painter(&newImage);

painter.drawImage(QPoint(0, 0), \*image);

\*image = newImage;

doc->update();

}

RotateCommand::RotateCommand(QWidget \*doc, QImage \*image, qreal deg) : doc(doc), image(image), deg(deg)

{

}

void RotateCommand::undo()

{

rotate(-deg);

}

void RotateCommand::redo()

{

rotate(deg);

}

void RotateCommand::rotate(qreal deg)

{

QMatrix transf;

\*image = image->transformed(transf.rotate(deg));

doc->update();

}

} // namespace Paint

**Command.h**

#ifndef COMMAND\_H

#define COMMAND\_H

#include "shape.h"

#include <QImage>

#include <QWidget>

#include <QUndoCommand>

#include <memory>

#include <vector>

namespace Paint {

/\*\*

\* @brief The ShapeCommand class for creating/undoing shapes on the document.

\*/

class ShapeCommand : public QUndoCommand

{

public:

explicit ShapeCommand(QWidget \*doc, QImage \*image, std::unique\_ptr<Shape> &&s);

virtual void undo() override;

virtual void redo() override;

private:

QWidget \*doc;

QImage \*image;

QImage undoImage;

std::unique\_ptr<Shape> shape;

};

/\*\*

\* @brief The FlipCommand class for flipping the image (document). Can flip

\* both horizontally and vertically in any combination.

\*/

class FlipCommand : public QUndoCommand

{

public:

explicit FlipCommand(QWidget \*doc, QImage \*image, bool horizontal, bool vertical);

virtual void undo() override;

virtual void redo() override;

private:

QWidget \*doc;

QImage \*image;

bool horizontal, vertical;

};

/\*\*

\* @brief The ResizeCommand class for resizing the image (document). Only

\* increases the size. Able to be merged with other subsequent resize commands.

\*/

class ResizeCommand : public QUndoCommand

{

public:

explicit ResizeCommand(QWidget \*doc, QImage \*image, const QSize &size);

virtual int id() const override;

virtual bool mergeWith(const QUndoCommand \*command) override;

virtual void undo() override;

virtual void redo() override;

private:

QWidget \*doc;

QImage \*image;

QSize oldSize, newSize;

};

/\*\*

\* @brief The RotateCommand class for rotating the image (document).

\*/

class RotateCommand : public QUndoCommand

{

public:

explicit RotateCommand(QWidget \*doc, QImage \*image, qreal deg);

virtual void undo() override;

virtual void redo() override;

private:

void rotate(qreal deg);

QWidget \*doc;

QImage \*image;

qreal deg;

};

} // namespace Paint

#endif // COMMAND\_H

**Document.cpp**

#include "document.h"

#include "command.h"

#include "floodfill.h"

#include <qevent.h>

#include <QPainter>

#include <algorithm>

namespace Paint {

Document::Document(QUndoStack \*undoStack, QWidget \*parent) :

QWidget(parent),

undoStack(undoStack),

penWidth(1),

penColor(Qt::blue)

{

}

bool Document::isModified() const

{

return !undoStack->isClean(); //undo stack keeps the document's modification state

}

std::vector<QPoint> Document::floodFill(const QPoint &pos, const QRgb &color)

{

return Paint::floodFill(&image, pos, color);

}

void Document::flip(bool horiz, bool vert)

{

undoStack->push(new FlipCommand(this, &image, horiz, vert));

}

void Document::rotate(qreal deg)

{

undoStack->push(new RotateCommand(this, &image, deg));

}

bool Document::openImage(const QString &fileName)

{

QImage loadedImage;

if (!loadedImage.load(fileName)) {

return false;

}

const QSize newSize = loadedImage.size().expandedTo(size());

image = loadedImage;

ResizeCommand(this, &image, newSize).redo(); // reuse the command's functionality out of the undo stack

undoStack->clear();

return true;

}

bool Document::saveImage(const QString &fileName, const char \*fileFormat)

{

if (image.save(fileName, fileFormat)) {

undoStack->clear();

return true;

} else {

return false;

}

}

void Document::setPenColor(const QColor &newColor)

{

penColor = newColor;

}

void Document::setPenWidth(int newWidth)

{

penWidth = newWidth;

}

void Document::setShapeFactory(shape\_factory\_t f)

{

factory = f;

}

void Document::mousePressEvent(QMouseEvent \*event)

{

if (event->button() == Qt::LeftButton && factory) {

currentShape = factory(event->pos(), penWidth, penColor);

}

}

void Document::mouseMoveEvent(QMouseEvent \*event)

{

if ((event->buttons() & Qt::LeftButton) && currentShape) {

const QRect prevRect = currentShape->rect();

currentShape->update(event->pos());

update(currentShape->rect().united(prevRect));

}

}

void Document::mouseReleaseEvent(QMouseEvent \*event)

{

if (event->button() == Qt::LeftButton && currentShape) {

// Done with drawind the image. Now pass the buck to the undo stack.

undoStack->push(new ShapeCommand(this, &image, std::move(currentShape)));

}

}

void Document::paintEvent(QPaintEvent \*event)

{

const QRect paintRect = event->rect();

QPainter painter(this);

painter.setRenderHint(QPainter::Antialiasing, true);

painter.drawImage(paintRect, image, paintRect);

if (currentShape) {

currentShape->draw(painter);

}

}

void Document::resizeEvent(QResizeEvent \*event)

{

if (width() > image.width() || height() > image.height()) {

const int newWidth = qMax(width() + 128, image.width());

const int newHeight = qMax(height() + 128, image.height());

const QSize newSize(newWidth, newHeight);

if (isModified()) {

undoStack->push(new ResizeCommand(this, &image, newSize));

} else {

ResizeCommand(this, &image, newSize).redo();

}

}

QWidget::resizeEvent(event);

}

} // Paint

**Document.h**

#ifndef DOCUMENT\_H

#define DOCUMENT\_H

#include "shape.h"

#include <QUndoStack>

#include <QWidget>

#include <functional>

#include <memory>

namespace Paint {

/\*\*

\* @brief The Document class representing a document where we draw on.

\*/

class Document : public QWidget

{

Q\_OBJECT

public:

typedef std::function<std::unique\_ptr<Shape>(const QPoint&, int, const QColor&)> shape\_factory\_t;

explicit Document(QUndoStack \*undoStack, QWidget \*parent = 0);

bool isModified() const;

bool openImage(const QString &fileName);

bool saveImage(const QString &fileName, const char \*fileFormat);

void setPenColor(const QColor &newColor);

void setPenWidth(int newWidth);

QColor getPenColor() const { return penColor; }

int getPenWidth() const { return penWidth; }

void setShapeFactory(shape\_factory\_t f);

void flip(bool horiz, bool vert);

void rotate(qreal deg);

std::vector<QPoint> floodFill(const QPoint &pos, const QRgb &color);

protected:

void mousePressEvent(QMouseEvent \*event);

void mouseReleaseEvent(QMouseEvent \*event);

void mouseMoveEvent(QMouseEvent \*event);

void paintEvent(QPaintEvent \*event);

void resizeEvent(QResizeEvent \*event);

private:

QImage image;

QUndoStack \*undoStack;

int penWidth;

QColor penColor;

shape\_factory\_t factory;

std::unique\_ptr<Shape> currentShape;

};

} // namespace Paint

#endif // DOCUMENT\_H

**Ellipse.cpp**

#include "shape.h"

namespace Paint {

class Ellipse : public Shape

{

public:

explicit Ellipse(const QPoint &topLeft,

int penWidth,

const QColor& penColor);

protected:

virtual void doDraw(QPainter &painter) override;

virtual QRect doRect() const override;

virtual void doUpdate(const QPoint &toPoint) override;

private:

QRect r;

};

Ellipse::Ellipse(const QPoint &topLeft, int penWidth, const QColor &penColor) :

Shape(penWidth, penColor), r(topLeft, topLeft)

{

}

void Ellipse::doDraw(QPainter &painter)

{

if (!r.isNull()) {

painter.drawEllipse(r.normalized());

}

}

QRect Ellipse::doRect() const

{

return r.normalized();

}

void Ellipse::doUpdate(const QPoint &toPoint)

{

r.setBottomRight(toPoint);

}

std::unique\_ptr<Shape> createEllipse(const QPoint &topLeft,

int penWidth,

const QColor& penColor)

{

return std::unique\_ptr<Shape>(new Ellipse(topLeft, penWidth, penColor));

}

} // namespace Paint

**Fill.cpp**

#include "document.h"

#include "shape.h"

#include <QPoint>

#include <vector>

namespace Paint {

class Fill : public Shape

{

public:

explicit Fill(Document \*doc, const QPoint &topLeft, int penWidth, const QColor& penColor);

protected:

virtual void doDraw(QPainter &painter) override;

virtual QRect doRect() const override;

virtual void doUpdate(const QPoint &toPoint) override;

private:

std::vector<QPoint> points;

QRect r;

};

Fill::Fill(Document \*doc, const QPoint &topLeft, int penWidth, const QColor &penColor) : Shape(penWidth, penColor), points(doc->floodFill(topLeft, penColor.rgb())),

r(topLeft, topLeft)

{

// Find the containing rectangle

for (const QPoint &p : points) {

if (p.x() < r.left()) {

r.setLeft(p.x());

} else if (p.x() > r.right()) {

r.setRight(p.x());

}

if (p.y() < r.top()) {

r.setTop(p.y());

} else if (p.y() > r.bottom()) {

r.setBottom(p.y());

}

}

}

void Fill::doDraw(QPainter &painter)

{

painter.drawPoints(points.data(), points.size());

}

QRect Fill::doRect() const

{

return r;

}

void Fill::doUpdate(const QPoint &)

{

// Flood-fill at the point of button press. Ignore "dragging".

}

std::unique\_ptr<Shape> createFill(Document \*doc, const QPoint &topLeft, int penWidth, const QColor& penColor)

{

return std::unique\_ptr<Shape>(new Fill(doc, topLeft, penWidth, penColor));

}

} // namespace Paint

**Floodfill.cpp**

#include "floodfill.h"

#include <vector>

namespace Paint {

/\*\*

\* The whole flood-fill thing is extracted from gPaint's 'src/fill.c' source.

\* Left as close as possible to the original code with some required

\* modifications (eg. using `std::vector` as the stack rather than a

\* preallocated buffer).

\*/

struct fillpixelinfo

{

int y, xl, xr, dy;

};

#define PUSH(py, pxl, pxr, pdy) \

{ \

struct fillpixelinfo p;\

if (((py) + (pdy) >= 0) && ((py) + (pdy) < image.height()))\

{\

p.y = (py);\

p.xl = (pxl);\

p.xr = (pxr);\

p.dy = (pdy);\

stack.push\_back(p); \

}\

}

#define POP(py, pxl, pxr, pdy) \

{\

struct fillpixelinfo p = stack.back();\

stack.pop\_back();\

(py) = p.y + p.dy;\

(pxl) = p.xl;\

(pxr) = p.xr;\

(pdy) = p.dy;\

}

std::vector<QPoint>

floodFill(QImage \*img, const QPoint &pos, const QRgb &newColor)

{

QImage image = img->copy();

std::vector<QPoint> modified;

int x = pos.x(), y = pos.y();

const QRgb oldColor = image.pixel(x, y);

std::vector<fillpixelinfo> stack;

int l, x1, x2, dy;

if ((x >= 0) && (x < image.width()) && (y >= 0) && (y < image.height()))

{

if (oldColor == newColor)

{

return modified;

}

PUSH(y, x, x, 1);

PUSH(y + 1, x, x, -1);

while (!stack.empty())

{

POP(y, x1, x2, dy);

for (x = x1; (x >= 0) && image.pixel(x, y) == oldColor; x--)

{

image.setPixel(x, y, newColor);

modified.emplace\_back(x, y);

}

if (x >= x1)

{

goto skip;

}

l = x + 1;

if (l < x1)

{

PUSH(y, l, x1 - 1, -dy);

}

x = x1 + 1;

do

{

for (; (x < image.width()) && image.pixel(x, y) == oldColor; x++)

{

image.setPixel(x, y, newColor);

modified.emplace\_back(x, y);

}

PUSH(y, l, x - 1, dy);

if (x > x2 + 1)

{

PUSH(y, x2 + 1, x - 1, -dy);

}

skip:

for (x++; x <= x2 && image.pixel(x, y) != oldColor; x++)

{

/\* empty \*/ ;

}

l = x;

} while (x <= x2);

}

}

return modified;

}

} // namespace Paint

**Mainwindow.cpp**

#include "mainwindow.h"

#include "document.h"

#include "shape.h"

#include <QtWidgets>

#include <functional>

#include <utility>

namespace Paint {

MainWindow::MainWindow(QWidget \*parent) : QMainWindow(parent), brushActionGroup(this), doc(&undoStack)

{

createMenus();

setCentralWidget(&doc);

setWindowTitle(tr("Paint"));

resize(500, 500);

brushActionGroup.actions().first()->trigger();

}

MainWindow::~MainWindow()

{

}

void MainWindow::closeEvent(QCloseEvent \*event)

{

if (maybeSave()) {

event->accept();

} else {

event->ignore();

}

}

void MainWindow::open()

{

if (maybeSave()) {

const QString fileName =

QFileDialog::getOpenFileName(this, tr("Open File"), QDir::currentPath());

if (!fileName.isEmpty()) {

doc.openImage(fileName);

}

}

}

void MainWindow::save()

{

saveFile("bmp");

}

void MainWindow::penColor()

{

const QColor newColor = QColorDialog::getColor(doc.getPenColor());

if (newColor.isValid()) {

doc.setPenColor(newColor);

}

}

void MainWindow::penWidth()

{

bool ok;

const int newWidth = QInputDialog::getInt(this, tr("Scribble"), tr("Select pen width:"), doc.getPenWidth(), 1, 50, 1, &ok);

if (ok) {

doc.setPenWidth(newWidth);

}

}

void MainWindow::createMenus()

{

using namespace std::placeholders;

QMenu \*const fileMenu = menuBar()->addMenu(tr("&File"));

fileMenu->addAction(tr("&Open..."), this, SLOT(open()), QKeySequence::Open);

fileMenu->addAction("&Save As...", this, SLOT(save()));

fileMenu->addSeparator();

fileMenu->addAction(tr("E&xit"),this, SLOT(close()), QKeySequence::Quit);

QMenu \*const editMenu = menuBar()->addMenu(tr("&Edit"));

QAction \*const undoAct = undoStack.createUndoAction(editMenu, tr("&Undo"));

undoAct->setShortcut(QKeySequence::Undo);

editMenu->addAction(undoAct);

QAction \*const redoAct = undoStack.createRedoAction(editMenu, tr("&Redo"));

redoAct->setShortcut(QKeySequence::Redo);

editMenu->addAction(redoAct);

QMenu \*const brushMenu = menuBar()->addMenu(tr("&Brush"));

brushMenu->addAction(tr("&Pen Color..."), this, SLOT(penColor()));

brushMenu->addAction(tr("Pen &Width..."), this, SLOT(penWidth()));

brushMenu->addSeparator();

const std::pair<QString, Document::shape\_factory\_t> shapeActions[] = {

std::make\_pair(tr("Ellipse"), createEllipse),

std::make\_pair(tr("Eraser"), createEraser),

std::make\_pair(tr("Fill"), std::bind(createFill, &doc, \_1, \_2, \_3)),

std::make\_pair(tr("Rectangle"), createRectangle),

std::make\_pair(tr("Line"), createScribble)

};

for (const auto &actionDesc : shapeActions) {

QAction \*const act = brushMenu->addAction(actionDesc.first);

connect(act, &QAction::triggered, std::bind(&Document::setShapeFactory, &doc, actionDesc.second));

brushActionGroup.addAction(act)->setCheckable(true);

}

brushActionGroup.setExclusive(true);

}

bool MainWindow::maybeSave()

{

if (doc.isModified()) {

const QMessageBox::StandardButton ret = QMessageBox::warning(this, tr("Paint"), tr("The image has been modified.\n" "Do you want to save your changes?"), QMessageBox::Save | QMessageBox::Discard | QMessageBox::Cancel);

if (ret == QMessageBox::Save) {

return saveFile("bmp");

} else if (ret == QMessageBox::Cancel) {

return false;

}

}

return true;

}

bool MainWindow::saveFile(const QByteArray &fileFormat)

{

const QString initialPath = QDir::currentPath() + "/untitled." + fileFormat;

const QString fileName = QFileDialog::getSaveFileName(this, tr("Save As"), initialPath, tr("%1 Files (\*.%2);;All Files (\*)").arg(QString::fromLatin1(fileFormat.toUpper())).arg(QString::fromLatin1(fileFormat)));

return !fileName.isEmpty() && doc.saveImage(fileName, fileFormat.constData());

}

} // namespace Paint

**Mainwindow2.cpp**

#include <QtWidgets>

#include <functional>

#include <utility>

#include "mainwindow2.h"

#include "multiscribblearea.h"

MainWindow::MainWindow()

{

scribbleArea = new ScribbleArea;

setCentralWidget(scribbleArea);

createActions();

createMenus();

setWindowTitle(tr("Paint"));

resize(500, 500);

}

void MainWindow::closeEvent(QCloseEvent \*event)

{

if (maybeSave()) {

event->accept();

}

else {

event->ignore();

}

}

void MainWindow::open()

{

if (maybeSave()) {

QString fileName = QFileDialog::getOpenFileName(this,

tr("Open File"), QDir::currentPath());

if (!fileName.isEmpty())

scribbleArea->openImage(fileName);

}

}

void MainWindow::save()

{

QAction \*action = qobject\_cast<QAction \*>(sender());

QByteArray fileFormat = action->data().toByteArray();

saveFile(fileFormat);

}

//fix this its expecting a doc to run the fucntion to get the new pen color

void MainWindow::penColor()

{

const QColor newColor = QColorDialog::getColor(scribbleArea->getPenColor());

if (newColor.isValid()) {

scribbleArea->setPenColor(newColor);

}

}

void MainWindow::createActions()

{

openAct = new QAction(tr("&Open..."), this);

openAct->setShortcut(tr("Ctrl+O"));

connect(openAct, SIGNAL(triggered()), this, SLOT(open()));

foreach(QByteArray format, QImageWriter::supportedImageFormats()) {

QString text = tr("%1...").arg(QString(format).toUpper());

QAction \*action = new QAction(text, this);

action->setData(format);

connect(action, SIGNAL(triggered()), this, SLOT(save()));

saveAsActs.append(action);

}

printAct = new QAction(tr("&Print..."), this);

connect(printAct, SIGNAL(triggered()), scribbleArea, SLOT(print()));

exitAct = new QAction(tr("E&xit"), this);

exitAct->setShortcut(tr("Ctrl+Q"));

connect(exitAct, SIGNAL(triggered()), this, SLOT(close()));

clearScreenAct = new QAction(tr("&Clear Screen"), this);

clearScreenAct->setShortcut(tr("Ctrl+L"));

connect(clearScreenAct, SIGNAL(triggered()),

scribbleArea, SLOT(clearImage()));

selectPenColor = new QAction(tr("&Selet Color"), this);

connect(selectPenColor, SIGNAL(triggered()),

scribbleArea, SLOT(setPen()));

aboutAct = new QAction(tr("&About"), this);

connect(aboutAct, SIGNAL(triggered()), this, SLOT(about()));

aboutQtAct = new QAction(tr("About &Qt"), this);

connect(aboutQtAct, SIGNAL(triggered()), qApp, SLOT(aboutQt()));

}

void MainWindow::createMenus()

{

saveAsMenu = new QMenu(tr("&Save As"), this);

foreach(QAction \*action, saveAsActs)

saveAsMenu->addAction(action);

fileMenu = new QMenu(tr("&File"), this);

fileMenu->addAction(openAct);

fileMenu->addMenu(saveAsMenu);

fileMenu->addAction(printAct);

fileMenu->addSeparator();

fileMenu->addAction(exitAct);

optionMenu = new QMenu(tr("&Options"), this);

optionMenu->addAction(clearScreenAct);

optionMenu->addAction(selectPenColor);

helpMenu = new QMenu(tr("&Help"), this);

helpMenu->addAction(aboutAct);

helpMenu->addAction(aboutQtAct);

menuBar()->addMenu(fileMenu);

menuBar()->addMenu(optionMenu);

menuBar()->addMenu(helpMenu);

}

bool MainWindow::maybeSave()

{

if (scribbleArea->isModified()) {

QMessageBox::StandardButton ret;

ret = QMessageBox::warning(this, tr("Scribble"),

tr("The image has been modified.\n"

"Do you want to save your changes?"),

QMessageBox::Save | QMessageBox::Discard

| QMessageBox::Cancel);

if (ret == QMessageBox::Save) {

return saveFile("png");

}

else if (ret == QMessageBox::Cancel) {

return false;

}

}

return true;

}

bool MainWindow::saveFile(const QByteArray &fileFormat)

{

QString initialPath = QDir::currentPath() + "/untitled." + fileFormat;

QString fileName = QFileDialog::getSaveFileName(this, tr("Save As"),

initialPath,

tr("%1 Files (\*.%2);;All Files (\*)")

.arg(QString::fromLatin1(fileFormat.toUpper()))

.arg(QString::fromLatin1(fileFormat)));

if (fileName.isEmpty()) {

return false;

}

else {

return scribbleArea->saveImage(fileName, fileFormat.constData());

}

}

**Mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include "document.h"

#include <QActionGroup>

#include <QMainWindow>

#include <QUndoStack>

namespace Paint {

class MainWindow : public QMainWindow //The MainWindow class representing the main application's window

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = 0);

~MainWindow();

protected:

void closeEvent(QCloseEvent \*event);

private slots:

void open();

void save();

void penColor();

void penWidth();

private:

void createMenus();

bool maybeSave();

bool saveFile(const QByteArray &fileFormat);

QActionGroup brushActionGroup;

QUndoStack undoStack;

Document doc;

};

} // namespace Paint

#endif // MAINWINDOW\_H

**Mainwindow2.h**

#ifndef MAINWINDOW2\_H

#define MAINWINDOW2\_H

#include <QList>

#include <QMainWindow>

class ScribbleArea;

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow();

protected:

void closeEvent(QCloseEvent \*event) override;

private slots:

void open();

void save();

void penColor();

//void penWidth();

private:

void createActions();

void createMenus();

bool maybeSave();

bool saveFile(const QByteArray &fileFormat);

QWidget \*doc;

ScribbleArea \*scribbleArea;

QMenu \*saveAsMenu;

QMenu \*fileMenu;

QMenu \*optionMenu;

QMenu \*helpMenu;

QAction \*openAct;

QList<QAction \*> saveAsActs;

QAction \*exitAct;

QAction \*printAct;

QAction \*clearScreenAct;

QAction \*selectPenColor;

QAction \*aboutAct;

QAction \*aboutQtAct;

};

#endif

**Multiscribblearea.cpp**

#include <QtWidgets>

#ifndef QT\_NO\_PRINTER

#include <QPrinter>

#include <QPrintDialog>

#endif

#include "multiscribblearea.h"

#include "mainwindow2.h"

static const qreal MinimumDiameter = 3.0;

static const qreal MaximumDiameter = 5.0;

ScribbleArea::ScribbleArea(QWidget \*parent)

: QWidget(parent)

{

setAttribute(Qt::WA\_AcceptTouchEvents);

setAttribute(Qt::WA\_StaticContents);

modified = false;

void ScribbleArea::setPenColor(const QColor &newColor)

{

penColor = newColor;

}

bool ScribbleArea::openImage(const QString &fileName)

{

QImage loadedImage;

if (!loadedImage.load(fileName))

return false;

QSize newSize = loadedImage.size().expandedTo(size());

resizeImage(&loadedImage, newSize);

image = loadedImage;

modified = false;

update();

return true;

}

bool ScribbleArea::saveImage(const QString &fileName, const char \*fileFormat)

{

QImage visibleImage = image;

resizeImage(&visibleImage, size());

if (visibleImage.save(fileName, fileFormat)) {

modified = false;

return true;

}

else {

return false;

}

}

void ScribbleArea::clearImage()

{

image.fill(qRgb(255, 255, 255));

modified = true;

update();

}

void ScribbleArea::paintEvent(QPaintEvent \*event)

{

QPainter painter(this);

const QRect rect = event->rect();

painter.drawImage(rect.topLeft(), image, rect);

}

void ScribbleArea::resizeEvent(QResizeEvent \*event)

{

if (width() > image.width() || height() > image.height()) {

int newWidth = qMax(width() + 128, image.width());

int newHeight = qMax(height() + 128, image.height());

resizeImage(&image, QSize(newWidth, newHeight));

update();

}

QWidget::resizeEvent(event);

}

void ScribbleArea::resizeImage(QImage \*image, const QSize &newSize)

{

if (image->size() == newSize)

return;

QImage newImage(newSize, QImage::Format\_RGB32);

newImage.fill(qRgb(255, 255, 255));

QPainter painter(&newImage);

painter.drawImage(QPoint(0, 0), \*image);

\*image = newImage;

}

void ScribbleArea::print()

{

#if !defined(QT\_NO\_PRINTER) && !defined(QT\_NO\_PRINTDIALOG)

QPrinter printer(QPrinter::HighResolution);

QPrintDialog printDialog(&printer, this);

if (printDialog.exec() == QDialog::Accepted) {

QPainter painter(&printer);

QRect rect = painter.viewport();

QSize size = image.size();

size.scale(rect.size(), Qt::KeepAspectRatio);

painter.setViewport(rect.x(), rect.y(), size.width(), size.height());

painter.setWindow(image.rect());

painter.drawImage(0, 0, image);

}

#endif // QT\_NO\_PRINTER

}

bool ScribbleArea::event(QEvent \*event)

{

switch (event->type()) {

case QEvent::TouchBegin:

case QEvent::TouchUpdate:

case QEvent::TouchEnd:

{

QTouchEvent \*touch = static\_cast<QTouchEvent \*>(event);

QList<QTouchEvent::TouchPoint> touchPoints = static\_cast<QTouchEvent \*>(event)->touchPoints();

foreach(const QTouchEvent::TouchPoint &touchPoint, touchPoints) {

switch (touchPoint.state()) {

case Qt::TouchPointStationary:

case Qt::TouchPointReleased:

// don't do anything if this touch point hasn't moved or has been released

continue;

default:

{

QRectF rect = touchPoint.rect();

if (rect.isEmpty()) {

qreal diameter = MaximumDiameter;

if (touch->device()->capabilities() & QTouchDevice::Pressure)

diameter = MinimumDiameter + (MaximumDiameter - MinimumDiameter) \* touchPoint.pressure();

rect.setSize(QSizeF(diameter, diameter));

}

QPainter painter(&image);

painter.setPen(Qt::NoPen);

painter.setBrush(penColor);//pencolor.at(touchPoint.id()));

painter.drawEllipse(rect);

painter.end();

modified = true;

int rad = 2;

update(rect.toRect().adjusted(-rad, -rad, +rad, +rad));

}

break;

}

}

break;

}

default:

return QWidget::event(event);

}

return true;

}

**Multiscribblearea.h**

#ifndef MULTISCRIBBLEAREA\_H

#define MULTISCRIBBLEAREA\_H

#include <QColor>

#include <QImage>

#include <QPoint>

#include <QWidget>

class ScribbleArea : public QWidget

{

Q\_OBJECT

public:

ScribbleArea(QWidget \*parent = 0);

bool openImage(const QString &fileName);

bool saveImage(const QString &fileName, const char \*fileFormat);

bool isModified() const { return modified; }

void setPenColor(const QColor &newColor);

//void setPenWidth(int newWidth);

QColor getPenColor() const { return penColor; }

int getPenWidth() const { return penWidth; }

public slots:

void clearImage();

void print();

protected:

void paintEvent(QPaintEvent \*event) override;

void resizeEvent(QResizeEvent \*event) override;

bool event(QEvent \*event) override;

private:

void resizeImage(QImage \*image, const QSize &newSize);

QWidget \*doc;

bool modified;

int penWidth;

QColor penColor;

// QList<QColor> myPenColors;

QImage image;

};

#endif

**Rectangle.cpp**

#include "shape.h"

namespace Paint {

class Rectangle : public Shape

{

public:

explicit Rectangle(const QPoint &topLeft, int penWidth, const QColor& penColor);

protected:

virtual void doDraw(QPainter &painter) override;

virtual QRect doRect() const override;

virtual void doUpdate(const QPoint &toPoint) override;

private:

QRect r;

};

Rectangle::Rectangle(const QPoint &topLeft, int penWidth, const QColor &penColor) : Shape(penWidth, penColor), r(topLeft, topLeft)

{

}

void Rectangle::doDraw(QPainter &painter)

{

if (!r.isNull()) {

painter.drawRect(r.normalized());

}

}

QRect Rectangle::doRect() const

{

return r.normalized();

}

void Rectangle::doUpdate(const QPoint &toPoint)

{

r.setBottomRight(toPoint);

}

std::unique\_ptr<Shape> createRectangle(const QPoint &topLeft, int penWidth, const QColor& penColor)

{

return std::unique\_ptr<Shape>(new Rectangle(topLeft, penWidth, penColor));

}

} // namespace Paint

**Scribble.cpp**

#include "shape.h"

#include <QPolygon>

namespace Paint {

class Scribble : public Shape

{

public:

explicit Scribble(const QPoint &topLeft, int penWidth, const QColor& penColor);

protected:

virtual void doDraw(QPainter &painter) override;

virtual QRect doRect() const override;

virtual void doUpdate(const QPoint &toPoint) override;

private:

QPolygon poly;

};

Scribble::Scribble(const QPoint &topLeft, int penWidth, const QColor &penColor) : Shape(penWidth, penColor)

{

update(topLeft);

}

void Scribble::doDraw(QPainter &painter)

{

painter.drawPolyline(poly);

}

QRect Scribble::doRect() const

{

return poly.boundingRect();

}

void Scribble::doUpdate(const QPoint &toPoint)

{

poly << toPoint;

}

std::unique\_ptr<Shape> createEraser(const QPoint &topLeft, int penWidth, const QColor&)

{

return std::unique\_ptr<Shape>(new Scribble(topLeft, penWidth, Qt::white));

}

std::unique\_ptr<Shape> createScribble(const QPoint &topLeft, int penWidth, const QColor& penColor)

{

return std::unique\_ptr<Shape>(new Scribble(topLeft, penWidth, penColor));

}

} // namespace Paint

**Shape.cpp**

#include "shape.h"

namespace Paint {

Shape::Shape(int penWidth, const QColor& penColor) :

penWidth(penWidth), penColor(penColor)

{

}

Shape::~Shape()

{

}

void Shape::draw(QPainter &painter)

{

const QPen prevPen = painter.pen();

painter.setPen(QPen(penColor, penWidth, Qt::SolidLine, Qt::RoundCap,

Qt::RoundJoin));

doDraw(painter);

painter.setPen(prevPen);

}

QRect Shape::rect() const

{

// Consider pen's width and make sure the outline is included in the

// returned rectangle.

const int rad = penWidth / 2 + 2;

return doRect().adjusted(-rad, -rad, +rad, +rad);

}

void Shape::update(const QPoint &toPoint)

{

doUpdate(toPoint);

}

} // namespace Paint

**Shape.h**

#ifndef SHAPE\_H

#define SHAPE\_H

#include <QColor>

#include <QPainter>

#include <QPoint>

#include <QRect>

#include <memory>

namespace Paint {

class Document;

class Shape //The Shape base class representing all shapes drawn on the document

{

public:

Shape(int penWidth, const QColor& penColor);

virtual ~Shape();

void draw(QPainter &painter); //Draw the shape using a painter

QRect rect() const; //Get the rectangle containing the shape

void update(const QPoint &toPoint); //Update the shape while "stretching" (drawing) it

protected:

virtual void doDraw(QPainter &painter) = 0;

virtual QRect doRect() const = 0;

virtual void doUpdate(const QPoint &toPoint) = 0;

private:

int penWidth;

QColor penColor;

};

std::unique\_ptr<Shape> createEllipse(const QPoint &topLeft, int penWidth, const QColor& penColor);

std::unique\_ptr<Shape> createEraser(const QPoint &topLeft, int penWidth, const QColor&);

std::unique\_ptr<Shape> createFill(Document \*doc, const QPoint &topLeft, int penWidth, const QColor&);

std::unique\_ptr<Shape> createRectangle(const QPoint &topLeft, int penWidth, const QColor& penColor);

std::unique\_ptr<Shape> createScribble(const QPoint &topLeft, int penWidth, const QColor& penColor);

} // namespace Paint

#endif // SHAPE\_H

**Paint.pro**

QT += core gui

qtHaveModule(printsupport)

#greaterThan(QT\_MAJOR\_VERSION, 4): QT += printsupport

QT += printsupport

TARGET = Paint

TEMPLATE = app

INSTALLS += target

SOURCES += main.cpp\

mainwindow.cpp \

document.cpp \

shape.cpp \

ellipse.cpp \

rectangle.cpp \

scribble.cpp \

command.cpp \

floodfill.cpp \

fill.cpp \

multiscribblearea.cpp \

mainwindow2.cpp

HEADERS += mainwindow.h \

document.h \

shape.h \

command.h \

floodfill.h \

multiscribblearea.h \

mainwindow2.h

OTHER\_FILES += \

README.md \

UNLICENSE.txt \

LICENSE.GPL

**System Validation**

This section shows the unit tests and integration tests that were done throughout the course of developing Interactive Paint.

Test case: Sample GUI

Purpose: Test the GUI on 2 different devices to see if the structure holds.

Precondition: User must have QT and Visual Studios installed, then connect them.

Expected Result: The code compiles and runs.

Actual Result: Actual result was the Expected Result.

Status: PASSED

Test case: Sample GUI

Purpose: Test the buttons to see if the event handlers get fired.  
Precondition: User must have QT and Visual Studios installed, then connect them.  
Input: The user clicks the “File”.  
Expected Result: The drop down menu opens.

Actual Result: Actual result was found to be the expected result.  
Status: PASSED

Test case: Create Rectangle

Purpose: to test that the rectangle button appears when user clicks on brush button

Precondition The user must have QT plugin for visual studio and must run the program

Input: User clicks on brush button which then a drop down menu appears

Expected Result: Rectangle button appears on GUI Canvas from a drop down menu

Actual Result: Actual result was found to be the expected result.

Status: PASSED

Test case: Create Rectangle

Purpose: to test that the rectangle button properly lets the user draw an rectangle on the GUI

Canvas

Precondition The user must have QT plugin for visual studio and must run the program

Input: User clicks on brush button which then a drop down menu appears and must click rectangle

Expected Result: Rectangle appears on GUI Canvas and is finally cemented when user clicks on

another area of canvas

Actual Result: Actual result was found to be the expected result.

Status: PASSED

Test case: Create Rectangle

Purpose: to test that the rectangle button appears when user clicks on brush button

Precondition The user must have QT plugin for visual studio and must run the program

Input: User clicks on brush button which then a drop down menu appears

Expected Result: Rectangle button appears on GUI Canvas from a drop down menu

Actual Result: Actual result was found to be the expected result.

Status: PASSED

Test case: Test QT plugin on Visual Studio

Purpose: to Test that GUI properly runs and pops up on the screen

Precondition: The user must have QT plugin in order to run the GUI

Input: User clicks the run in visual studio

Expected Result: GUI pops up in the middle of the screen

Actual Result: Actual result was found to be the expected result.

Status: PASSED

Test case: Test QT plugin on Visual Studio

Purpose: to Test that GUI properly runs and pops up on the screen  
Precondition: The user must have QT plugin in order to run the GUI  
Input: User clicks the run in visual studio  
Expected Result: GUI pops up in the middle of the screen  
Actual Result: Actual result was found to be the expected result.  
Status: PASSED

Test case: Create Ellipse

Purpose: to test that the ellipse button appears when user clicks on brush button  
Precondition The user must have QT plugin for visual studio and must run the program  
Input: User clicks on brush button which then a drop down menu appears  
Expected Result: Ellipse button appears on GUI Canvas from a drop down menu  
Actual Result: Actual result was found to be the expected result.  
Status: PASSED

Test case: Create Ellipse  
Purpose: to test that the ellipse button properly lets the user draw an ellipse on the GUI Canvas  
Precondition The user must have QT plugin for visual studio and must run the program  
Input: User clicks on brush button which then a drop down menu appears and must click ellipse  
Expected Result: Ellipse appears on GUI Canvas and is finally cemented when user clicks on another area of canvas  
Actual Result: Actual result was found to be the expected result.  
Status: PASSED

Test case: Create Ellipse  
Purpose: to test that the ellipse button appears when user clicks on brush button  
Precondition The user must have QT plugin for visual studio and must run the program  
Input: User clicks on brush button which then a drop down menu appears  
Expected Result: Ellipse button appears on GUI Canvas from a drop down menu  
Actual Result: Actual result was found to be the expected result.  
Status: PASSED

Test case: Create a Brush

Purpose: to test that the line button appears when user clicks on brush button  
Precondition The user must have QT plugin for visual studio and must run the program  
Input: User clicks on brush button which then a drop down menu appears  
Expected Result: line button appears on GUI Canvas from a drop down menu  
Actual Result: Actual result was found to be the expected result.  
Status: PASSED  
  
  
Test case: Create a Brush  
Purpose: to test that the line button properly lets the user draw a line on the GUI Canvas  
Precondition The user must have QT plugin for visual studio and must run the program  
Input: User clicks on brush button which then a drop down menu appears and must click line  
Expected Result: line appears on GUI Canvas when dragged across canvas  
Actual Result: Actual result was found to be the expected result.  
Status: PASSED  
  
  
Test case: Create a Brush  
Purpose: to test that the line button appears when user clicks on brush button  
Precondition The user must have QT plugin for visual studio and must run the program  
Input: User clicks on brush button which then a drop down menu appears  
Expected Result: line button appears on GUI Canvas from a drop down menu  
Actual Result: Actual result was found to be the expected result.  
Status: PASSED

Test case: Create a brush in the GUI Canvas  
Purpose: to test that the line button appears when user clicks on brush button  
Precondition The user must have QT plugin for visual studio and must run the program  
Input: User clicks on brush button which then a drop down menu appears  
Expected Result: line button appears on GUI Canvas from a drop down menu  
Actual Result: Actual result was found to be the expected result.  
Status: PASSED  
  
Test case: Create a brush in the GUI Canvas  
Purpose: to test that the line button properly lets the user draw a line on the GUI Canvas  
Precondition The user must have QT plugin for visual studio and must run the program  
Input: User clicks on brush button which then a drop down menu appears and must click line Expected Result: line appears on GUI Canvas when dragged across canvas  
Actual Result: Actual result was found to be the expected result.  
Status: PASSED  
  
  
Test case: Create a brush in the GUI Canvas  
Purpose: to test that the line button appears when user clicks on brush button  
Precondition The user must have QT plugin for visual studio and must run the program  
Input: User clicks on brush button which then a drop down menu appears  
Expected Result: line button appears on GUI Canvas from a drop down menu  
Actual Result: Actual result was found to be the expected result.  
Status: PASSED

Test case: Implement an Eraser

Purpose: to test that the Eraser button appears when user clicks on brush button

Precondition The user must have QT plugin for visual studio and must run the program

Input: User clicks on brush button which then a drop down menu appears

Expected Result: Eraser button appears on GUI Canvas from a drop down menu

Actual Result: Actual result was found to be the expected result.

Status: PASSED

Test case: Implement an Eraser

Purpose: to test that the Eraser button properly lets the user draw a line on the GUI Canvas

Precondition The user must have QT plugin for visual studio and must run the program

Input: User clicks on brush button which then a drop down menu appears and must click Eraser

Expected Result: White line appears on GUI Canvas when dragged across canvas

Actual Result: Actual result was found to be the expected result.

Status: PASSED

Test case: Implement an Eraser

Purpose: to test that the Eraser button appears when user clicks on brush button

Precondition The user must have QT plugin for visual studio and must run the program

Input: User clicks on brush button which then a drop down menu appears

Expected Result: Eraser button appears on GUI Canvas from a drop down menu

Actual Result: Actual result was found to be the expected result.

Test case: Create Fill

Purpose: to test that the fill button appears when user clicks on fill button  
Precondition The user must have QT plugin for visual studio and must run the program  
Input: User clicks on brush button which then a drop down menu appears  
Expected Result: fill button appears on GUI Canvas from a drop down menu  
Actual Result: Actual result was found to be the expected result.  
Status: PASSED  
  
Test case: Create Fill  
Purpose: to test that the fill button properly lets the user fill a closed area on the GUI Canvas  
Precondition The user must have QT plugin for visual studio and must run the program  
Input: User clicks on brush button which then a drop down menu appears and must click fill  
Expected Result: filled area appears on GUI Canvas when clicked inside closed area  
Actual Result: Actual result was found to be the expected result.  
Status: PASSED  
  
  
Test case: Create Fill  
Purpose: to test that the fill button appears when user clicks on brush button  
Precondition The user must have QT plugin for visual studio and must run the program  
Input: User clicks on fill button which then a drop down menu appears  
Expected Result: fill button appears on GUI Canvas from a drop down menu  
Actual Result: Actual result was found to be the expected result.  
Status: PASSED

Test case: Create Undo

Purpose: to test that the undo button appears when user clicks on undo button

Precondition The user must have QT plugin for visual studio and must run the program

Input: User clicks on undo button which then a drop down menu appears

Expected Result: undo button appears on GUI Canvas from a drop down menu

Actual Result: Actual result was found to be the expected result.

Status: PASSED

Test case: Create Undo

Purpose: to test that the undo button properly lets the user return the canvas to the previous state

Precondition The user must have QT plugin for visual studio and must run the program

Input: User clicks on undo button which then a drop down menu appears and must click undo

Expected Result: returns the canvas to the previous state

Actual Result: Actual result was found to be the expected result.

Status: PASSED

Test case: Create Main Menu

Purpose: to test that the SingleTouch Button launches Single Touch application

Precondition The user must have QT plugin for visual studio and must run the program

Input: User clicks on SingleTouch Button

Expected Result: launches SingleTouch Application

Actual Result: Does not launch SingleTouch Application

Status: FAILED

Test case: Create Main Menu

Purpose: to test that the MultiTouch Button launches Multi Touch application

Precondition The user must have QT plugin for visual studio and must run the program

Input: User clicks on MultiTouch Button

Expected Result: launches MultiTouch Application

Actual Result: Does not launch MultiTouch Application

Status: FAILED

**Integration Tests**

The painting application built by Daniel and Jim was tested manually by allowing multiple users to use the application for several minutes at a time. Each trying out all the different features that the application has.

As most of the work done was split into individualized groups, there is little room for integration, or integration tests. At the request of our product owner, the concept of integration was to be reserved for future iterations of the project.

# 

# **Glossary**

**QT:** A cross-platform application framework that can be run on various software and hardware platforms with little or no change in the underlying codebase, while still being a native application with native capabilities and speed.

**C++:** is a [general-purpose programming language](https://en.wikipedia.org/wiki/General-purpose_programming_language). It has [imperative](https://en.wikipedia.org/wiki/Imperative_programming), [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) and [generic](https://en.wikipedia.org/wiki/Generic_programming) programming features, while also providing facilities for [low-level](https://en.wikipedia.org/wiki/Low-level_programming) [memory](https://en.wikipedia.org/wiki/Memory_(computing)) manipulation. It was designed with a bias toward [system programming](https://en.wikipedia.org/wiki/System_programming) and [embedded](https://en.wikipedia.org/wiki/Embedded_software), resource-constrained and large systems, with [performance](https://en.wikipedia.org/wiki/Performance_(software)), efficiency and flexibility of use as its design highlights.

**Touchscreen:** is an [input](https://en.wikipedia.org/wiki/Input_device) and [output device](https://en.wikipedia.org/wiki/Output_device) normally layered on the top of an [electronic visual display](https://en.wikipedia.org/wiki/Electronic_visual_display) of an [information processing system](https://en.wikipedia.org/wiki/Information_processing_system). A user can give input or control the [information processing system](https://en.wikipedia.org/wiki/Information_processing_system) through simple or [multi-touch gestures](https://en.wikipedia.org/wiki/Multi-touch) by touching the screen with a special [stylus](https://en.wikipedia.org/wiki/Stylus_(computing)) and/or one or more fingers.

# 

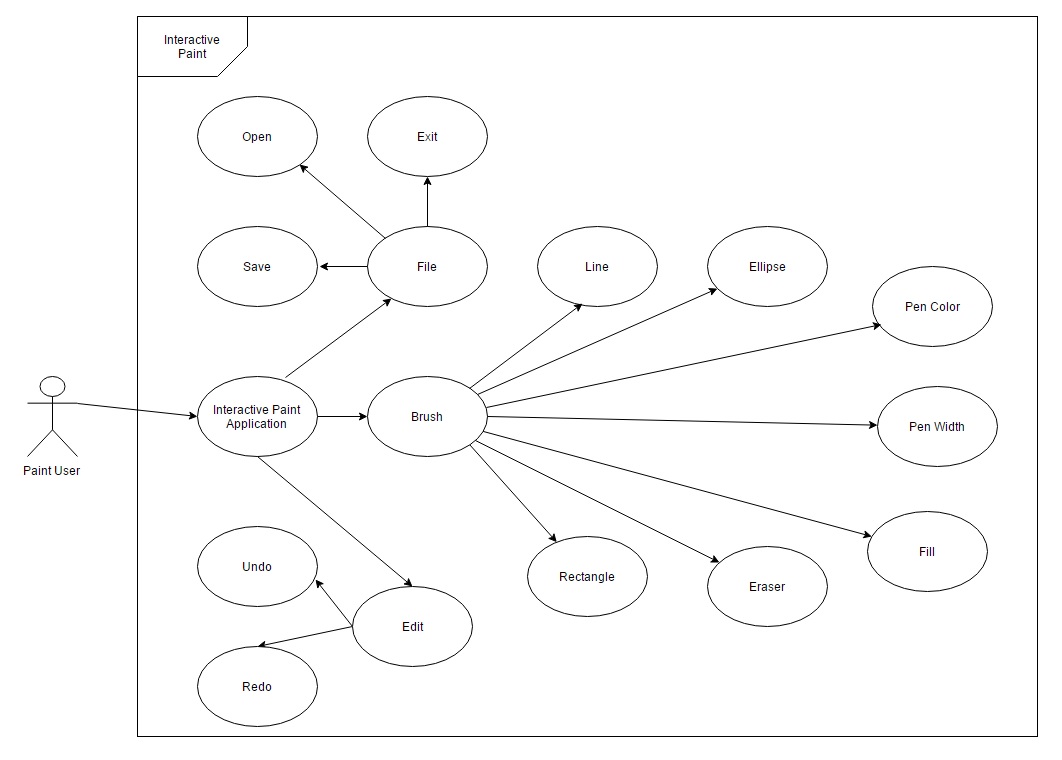
# 

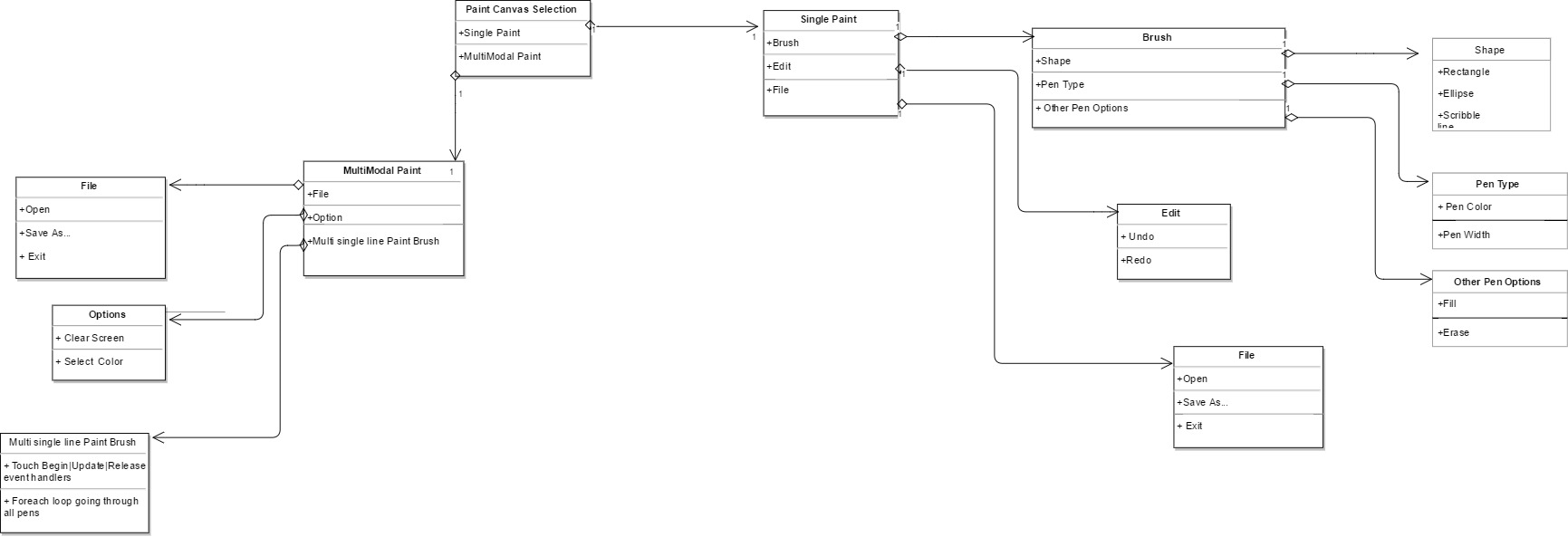
# 

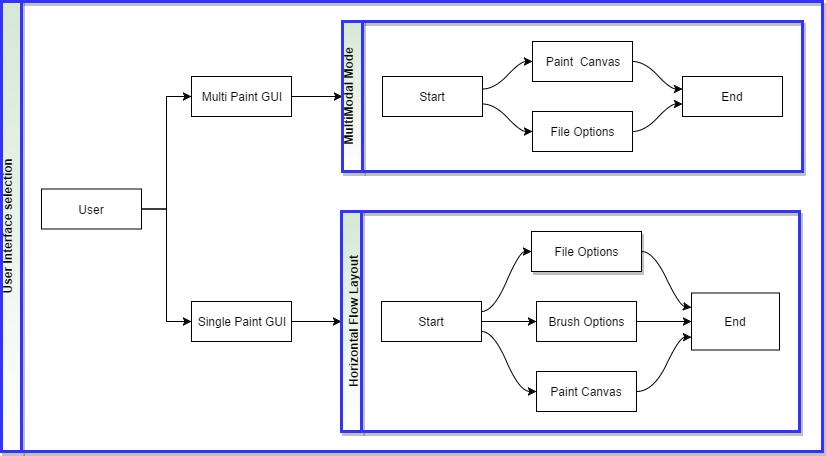
# 

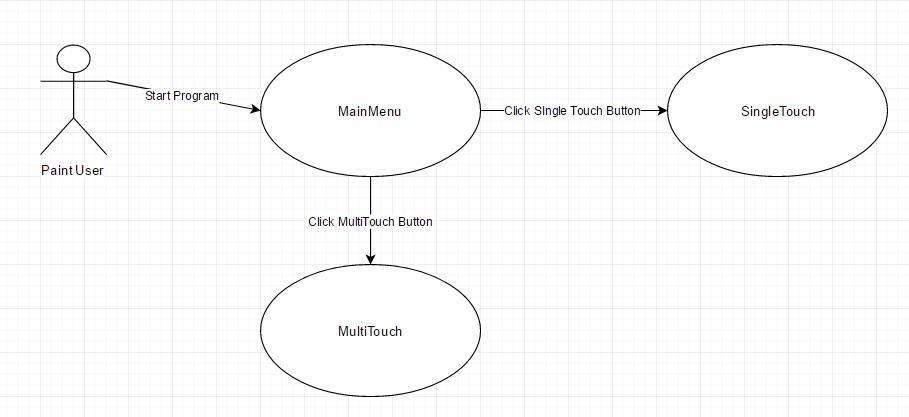
# **Appendix**

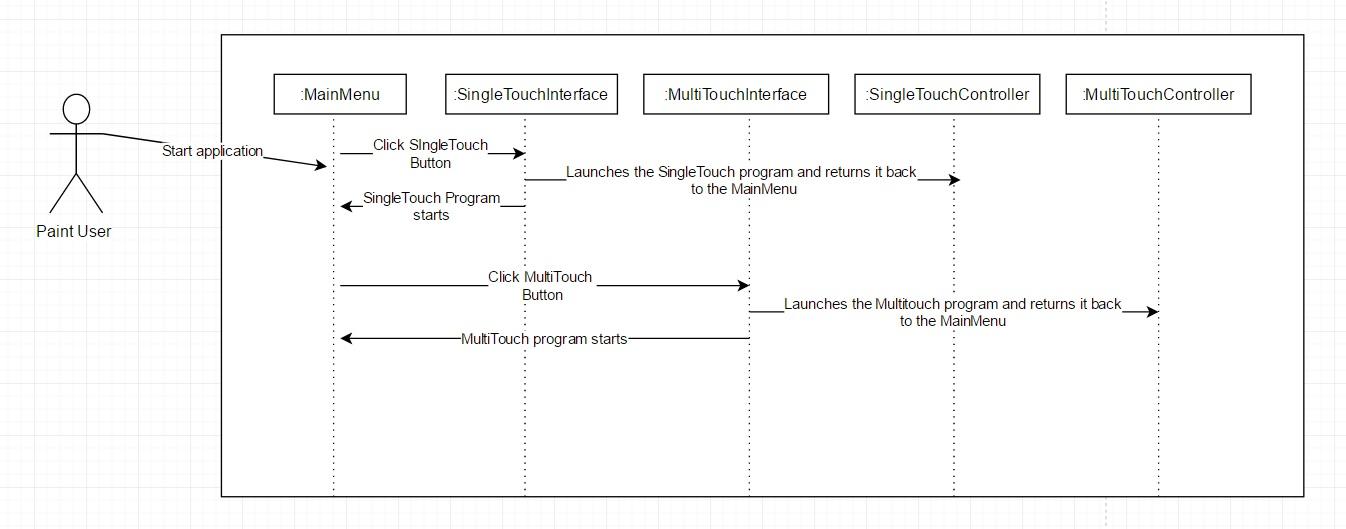
## **Appendix A - UML Diagrams**



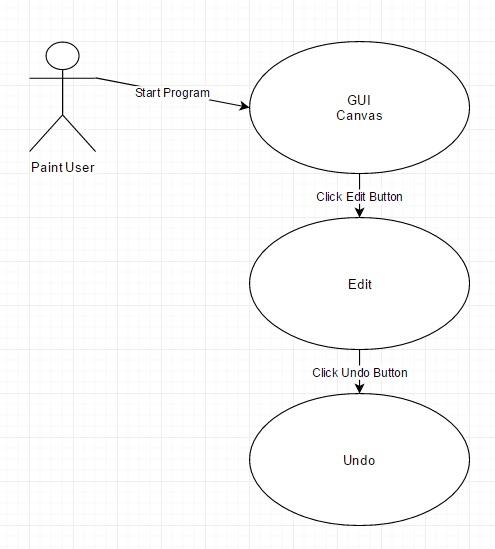


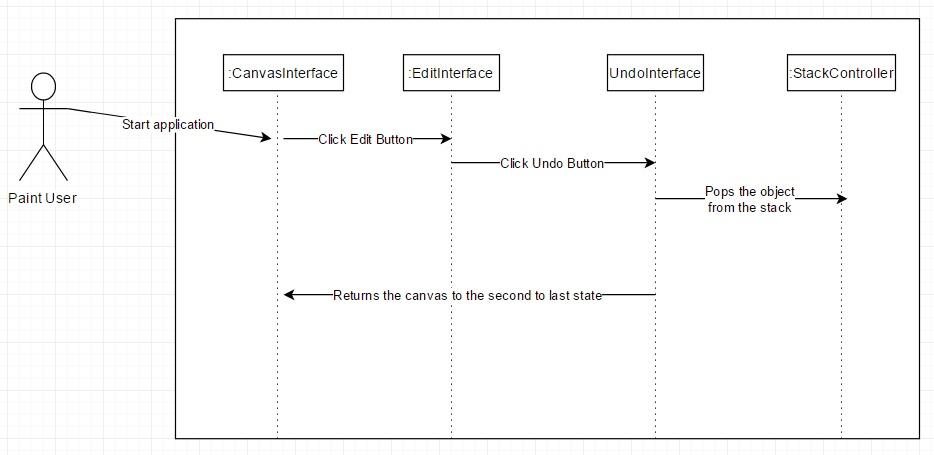


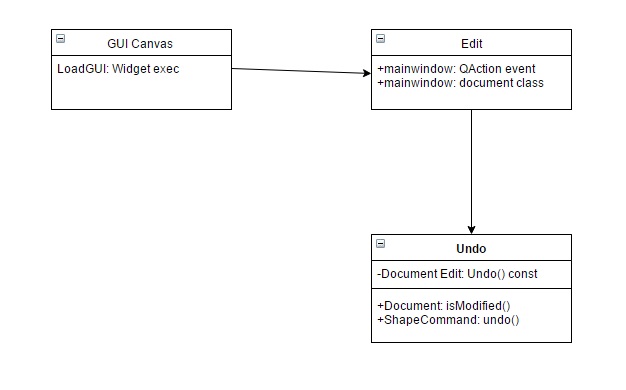




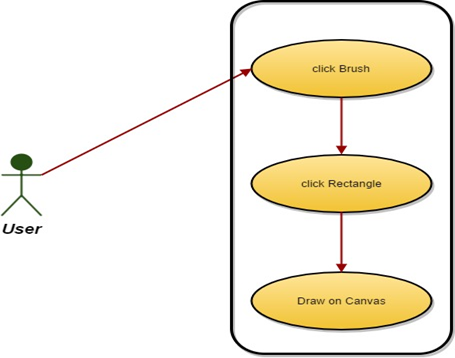
**Use case 327**

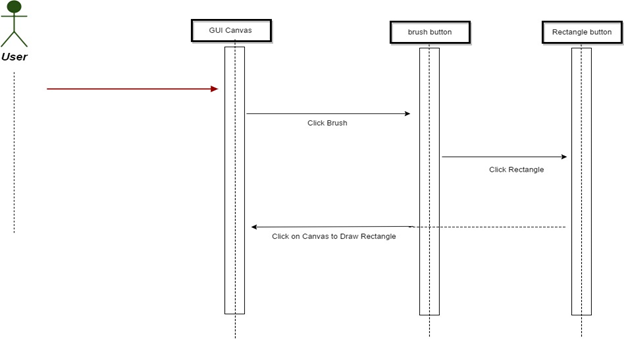


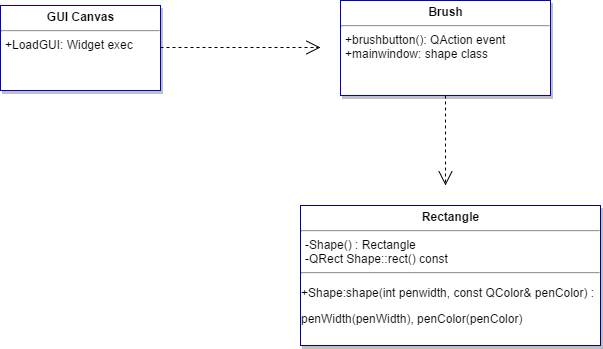




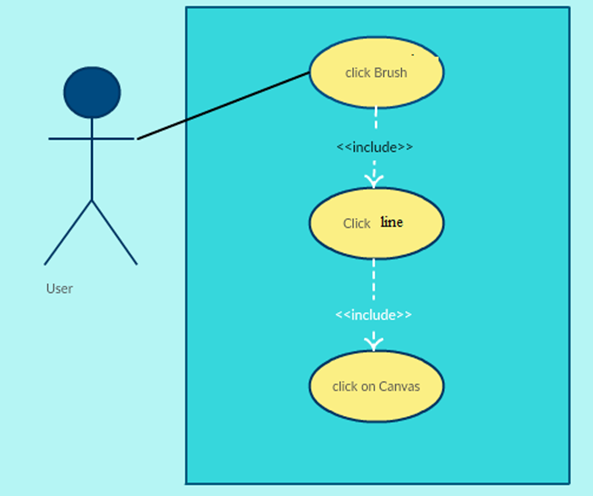
**Use case 306**

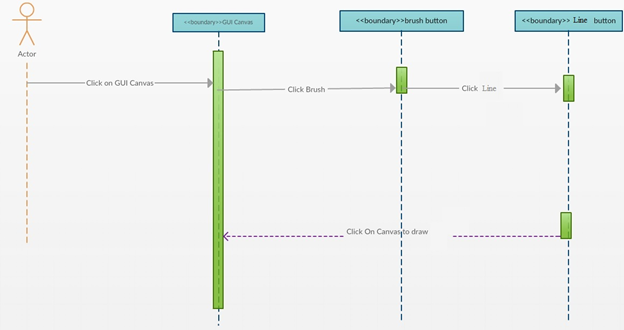


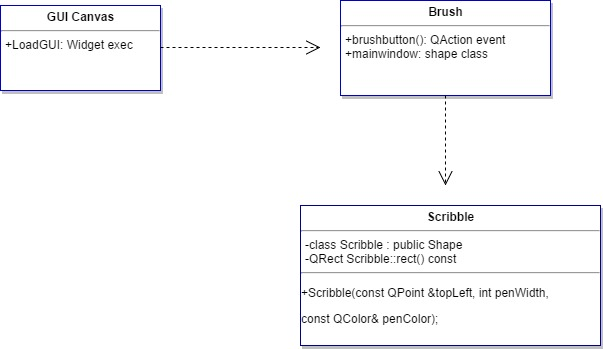




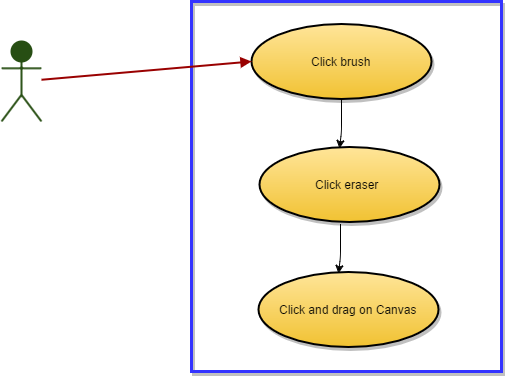
**Use case 320**

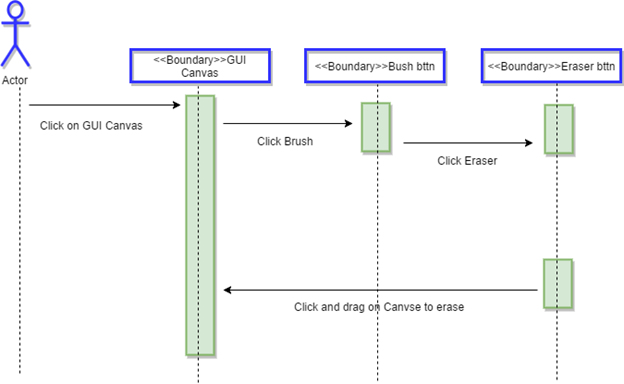


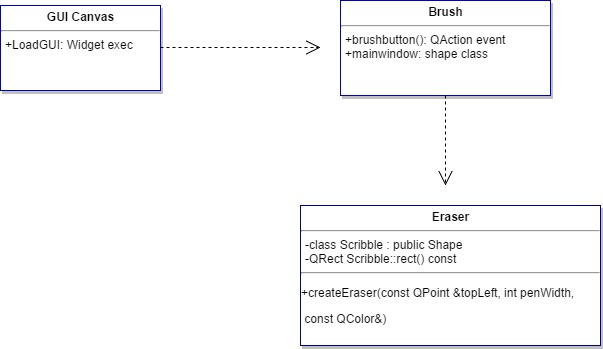




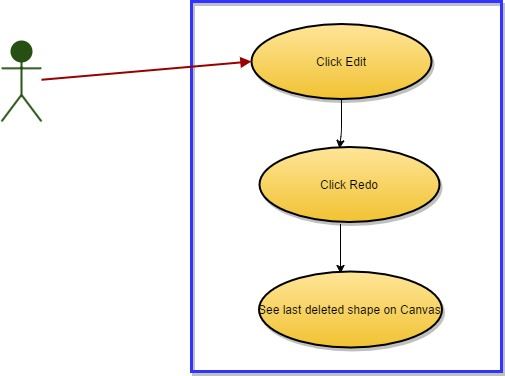
**Use case 326**

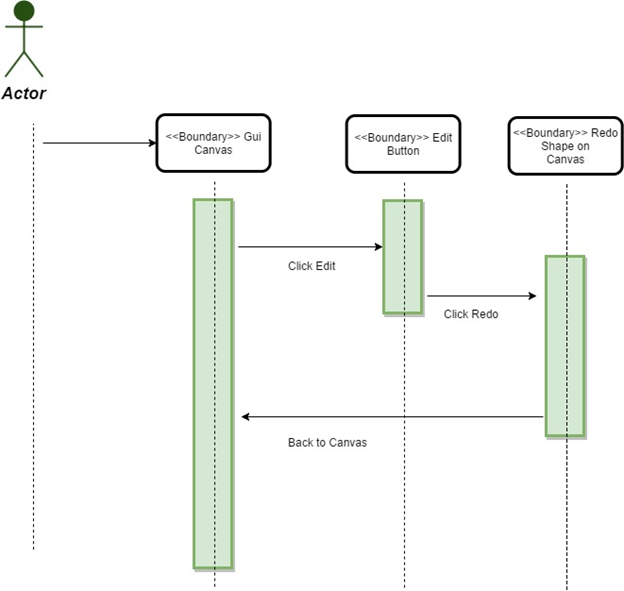


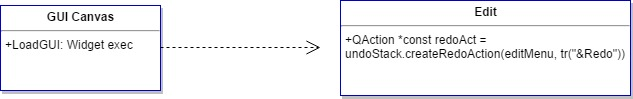




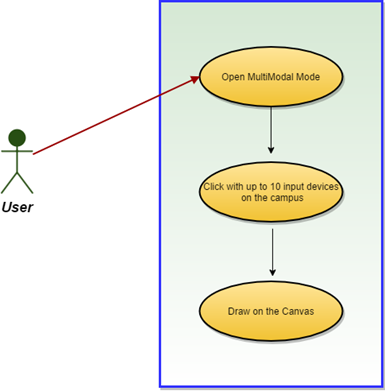
**Use case 348**

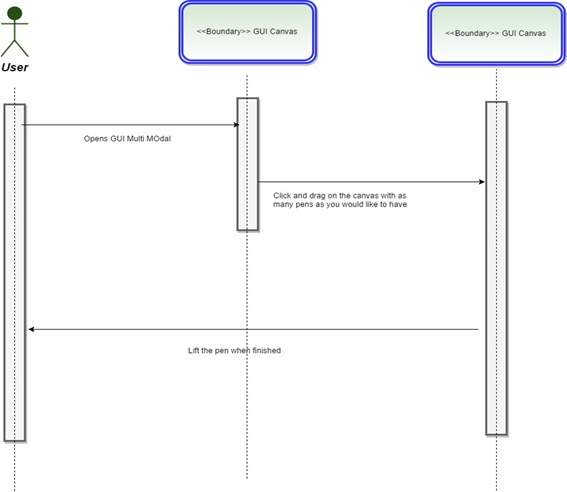


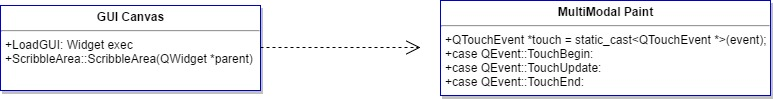




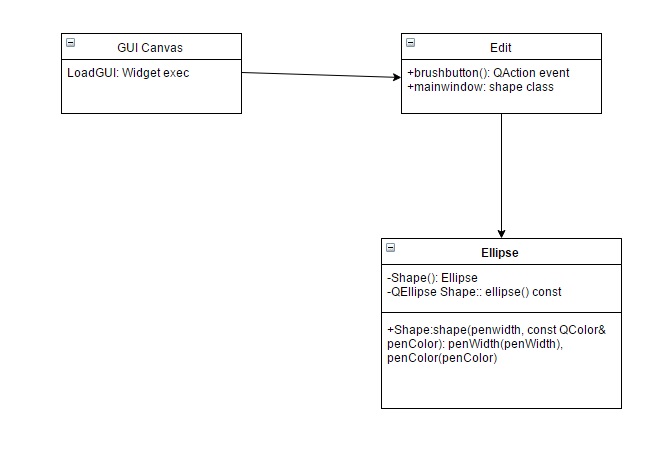
**Use case 359**



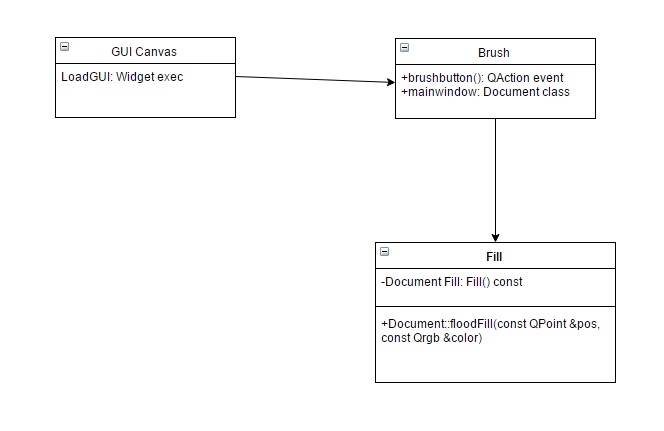




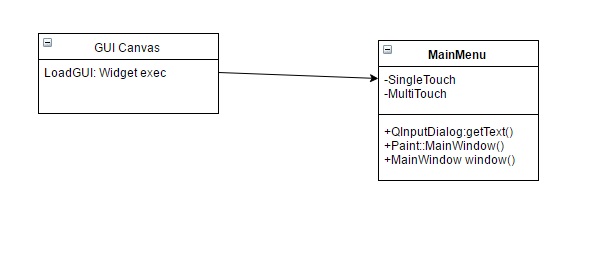
**Use case 307**



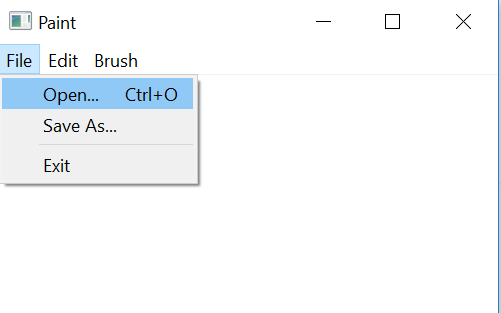
**Use case 348**

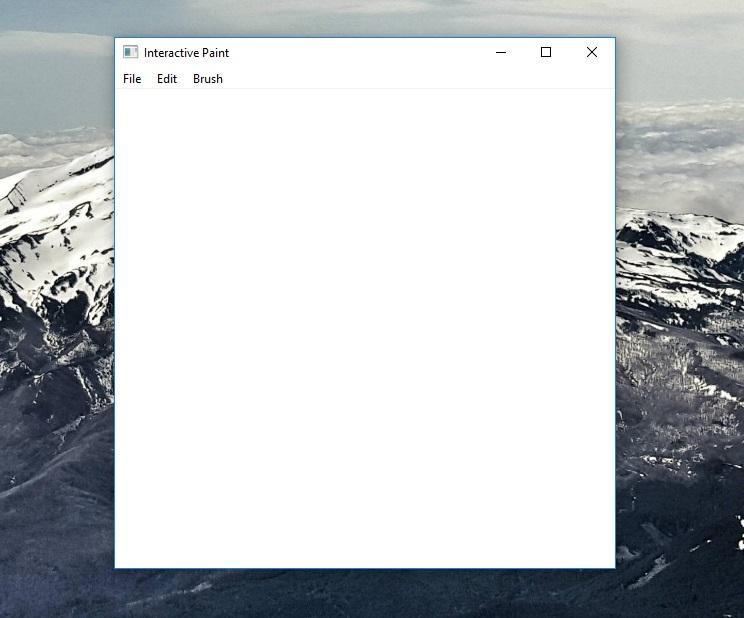


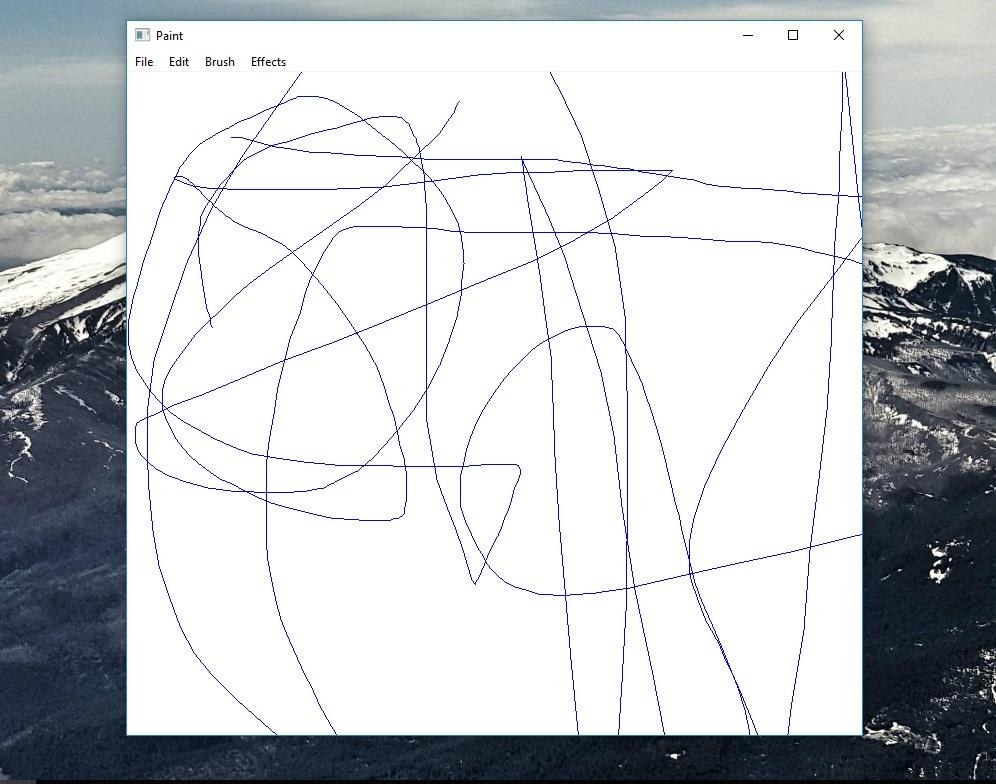
**Use case 357**

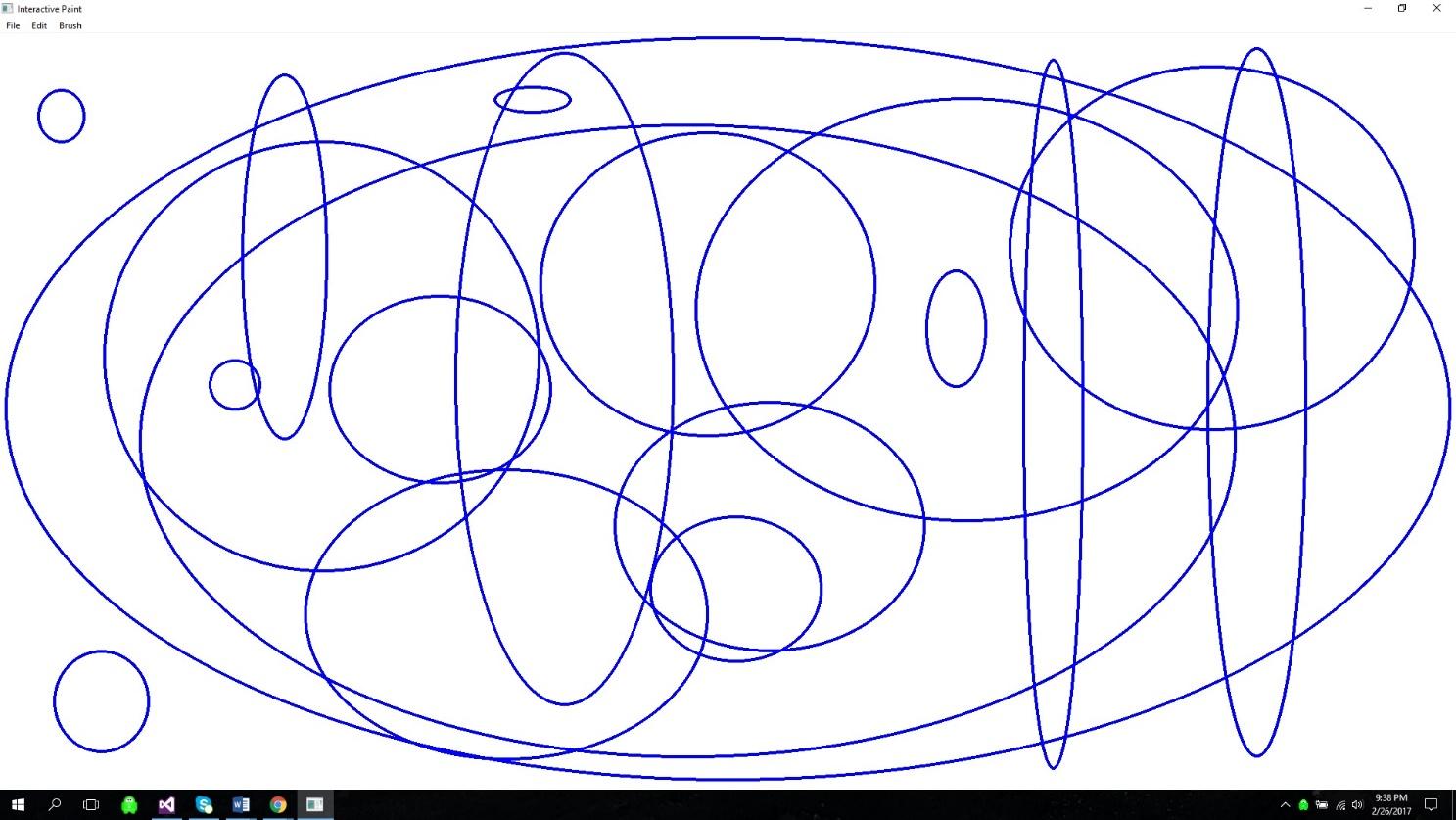


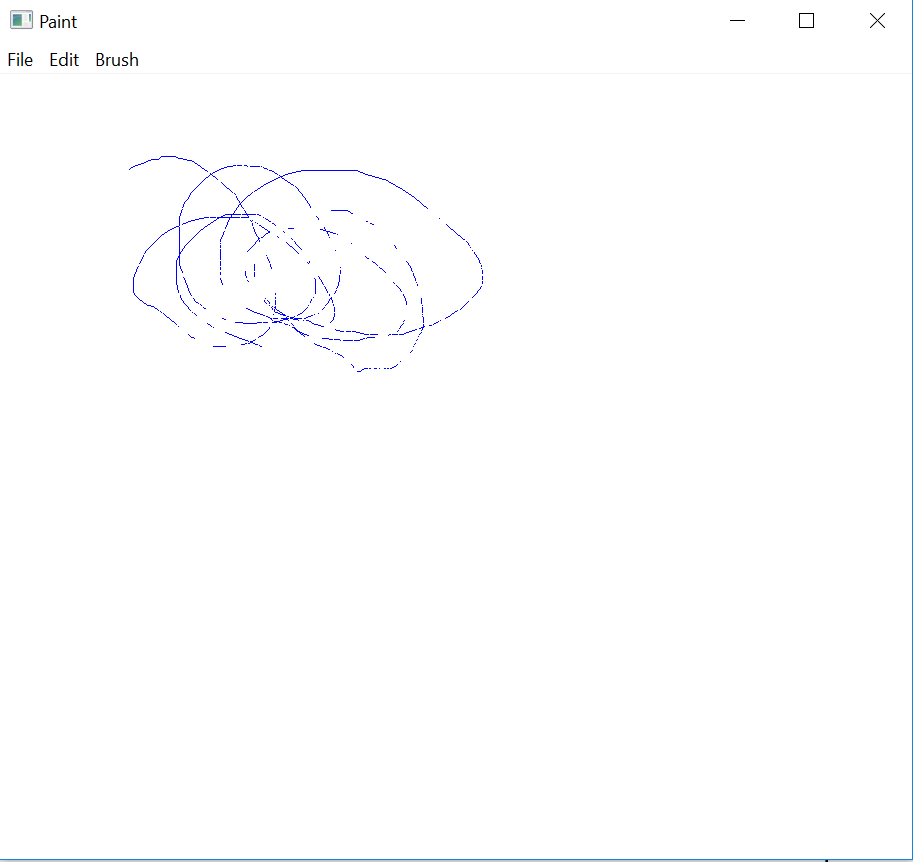
**Appendix B - User Interface Design**

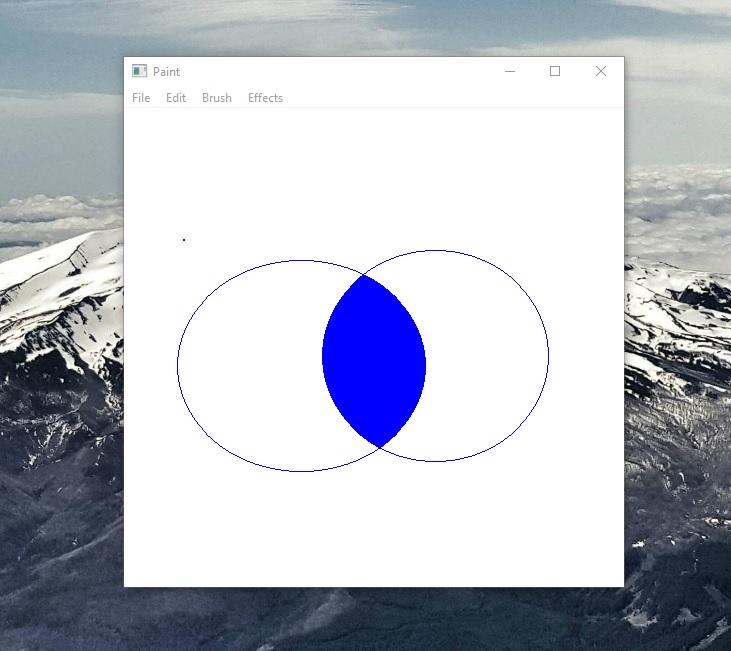


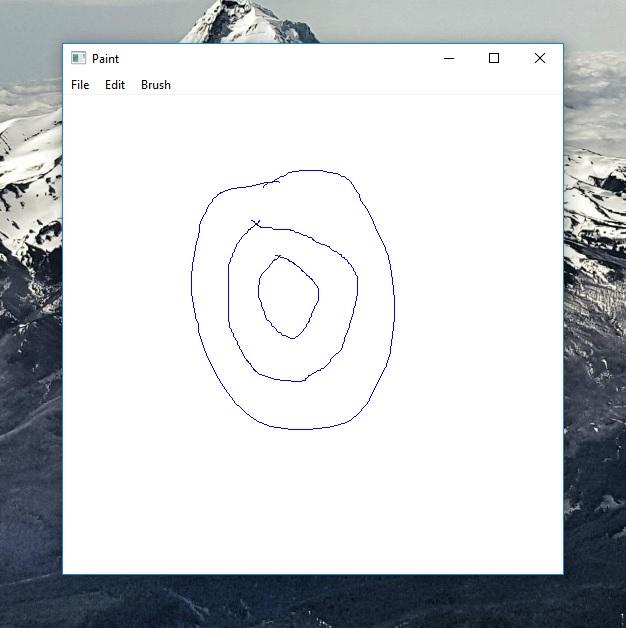


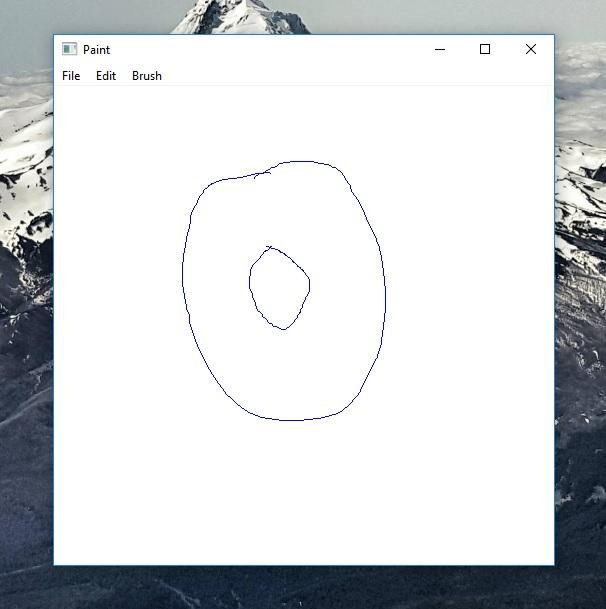


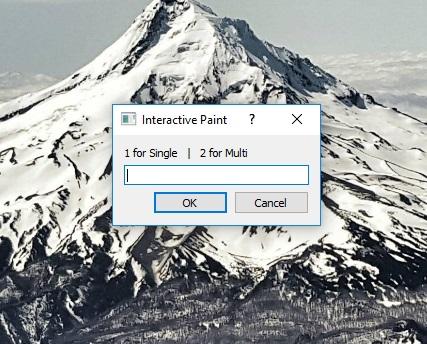


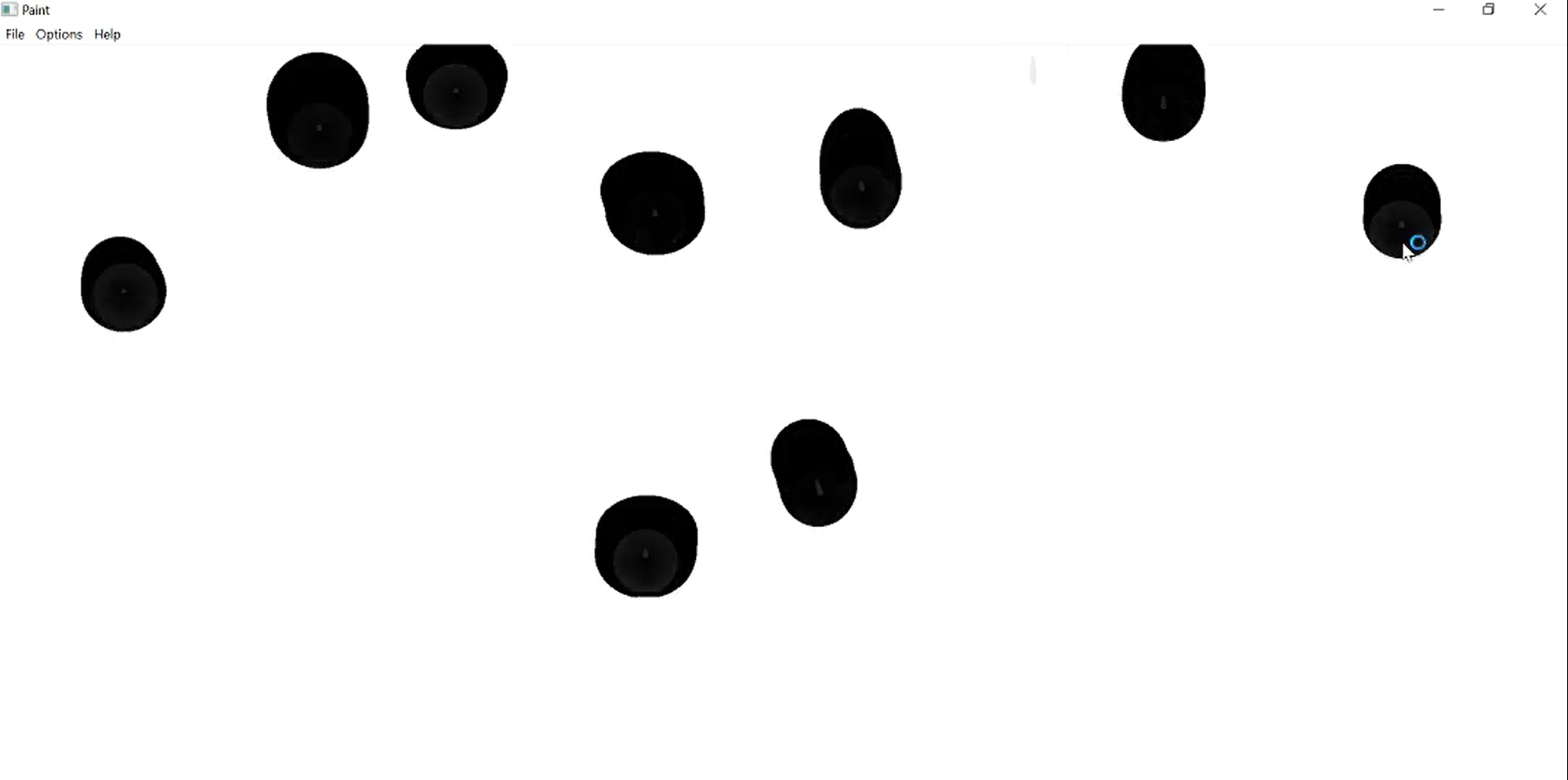


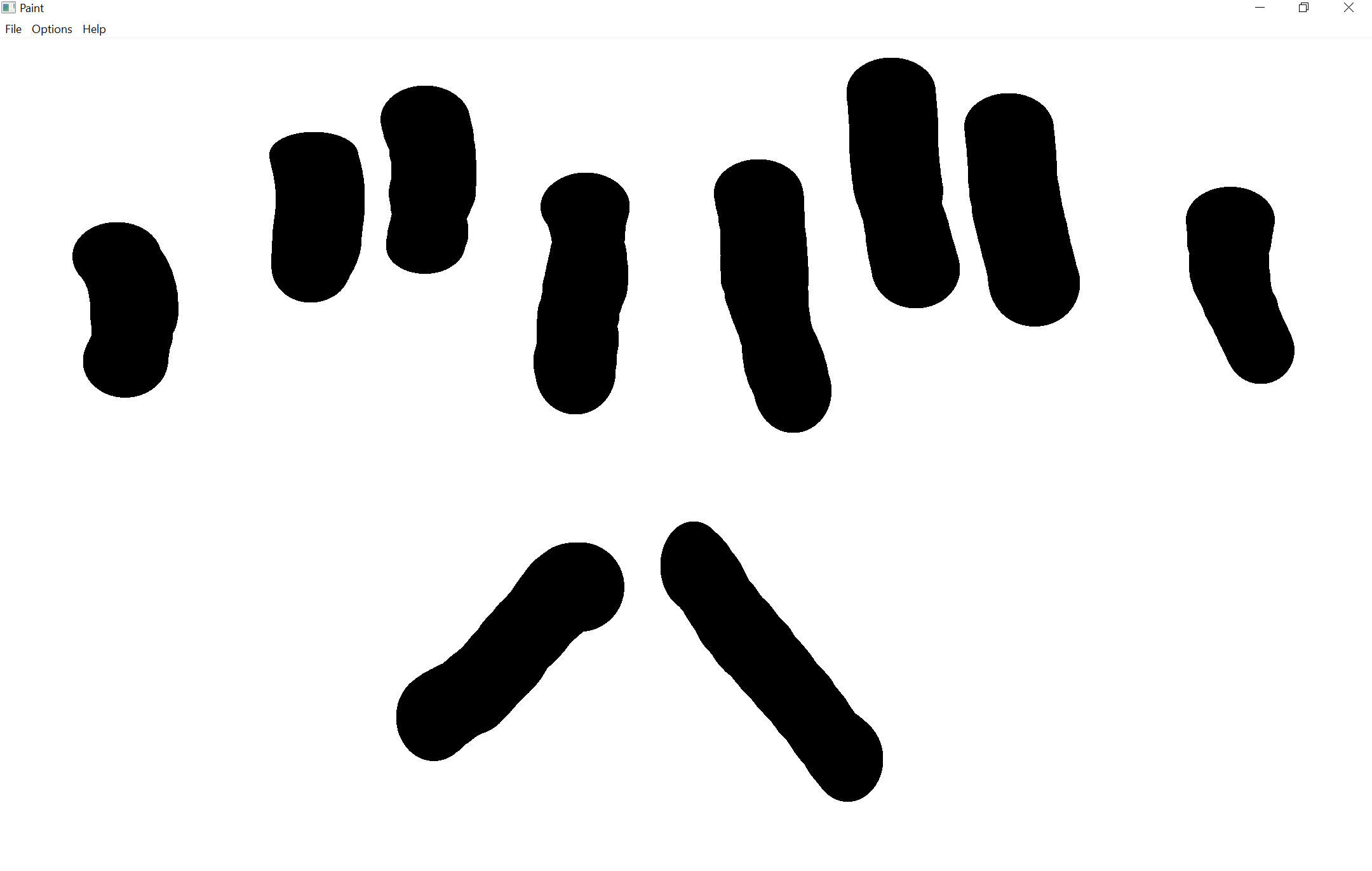


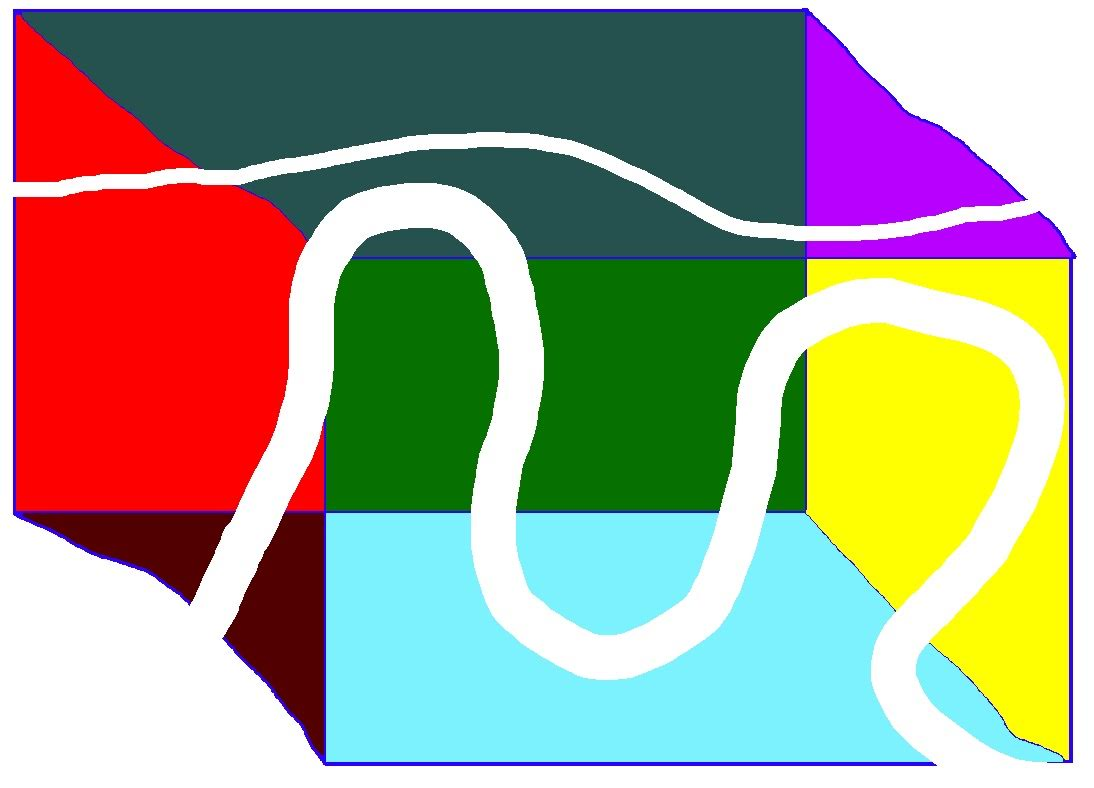












## **Appendix C - Sprint Review Reports**

**Sprint 1**

Attendees: Daniel Mederos, Jim Louro

Start time: 12:30 PM

End time: 1:00 PM

What went wrong?

● Did we do a good job estimating our team's velocity?

**No**, most of us ended with an incorrect number of hours

● Did we do a good job estimating the points (time required) for each user story?

**Yes**, but not exact

● Did each team member work as scheduled?

**Yes**.

What went right?

● Nothing, previous project could not be compiled

How to address the issues in the next sprint?

● How to improve the process?

Be more exact on sprint meeting times.

● How to improve the product?

Keep trying harder to find all the missing files

**Sprint 2**

Attendees: Daniel Mederos, Jim Louro

Start time: 12:30 PM

End time: 1:00 PM

What went wrong?

● Did we do a good job estimating our team's velocity?

**No**

● Did we do a good job estimating the points (time required) for each user story?

**Yes**

● Did each team member work as scheduled?

**Yes**.

What went right?

● Nothing, previous project could still not be compiled

How to address the issues in the next sprint?

● How to improve the process?

Talk to the product owner for advice

● How to improve the product?

Talk to the product owner for advice

**Sprint 3**

Attendees: Daniel Mederos, Jim Louro

Start time: 12:30 PM

End time: 1:00 PM

What went wrong?

● Did we do a good job estimating our team's velocity?

**Yes**

● Did we do a good job estimating the points (time required) for each user story?

**Yes**

● Did each team member work as scheduled?

**Yes**.

What went right?

● We started the project from scratch, we are trying to learn a new framework

How to address the issues in the next sprint?

● How to improve the process?

Learn the QT framework in C++ which is very extensive

● How to improve the product?

Set up a basic program and play around with some basic examples

**Sprint 4**

Attendees: Daniel Mederos, Jim Louro

Start time: 12:30 PM

End time: 1:00 PM

What went wrong?

● Did we do a good job estimating our team's velocity?

**Yes**

● Did we do a good job estimating the points (time required) for each user story?

**Yes**

● Did each team member work as scheduled?

**Yes**.

What went right?

●

How to address the issues in the next sprint?

● How to improve the process?

Learn the QT framework in C++ which is very extensive

● How to improve the product?

Set up a basic program and play around with some basic examples

**Sprint 5**

Attendees: Daniel Mederos, Jim Louro

Start time: 12:30 PM

End time: 1:00 PM

What went wrong?

● Did we do a good job estimating our team's velocity?

**Yes, even though we took Spring break to apply for jobs and prepare for interviews we each implemented a use case**

● Did we do a good job estimating the points (time required) for each user story?

**Yes**

● Did each team member work as scheduled?

**Yes**.

What went right?

● The ability to work on the project in two different environments add functionality and research more on Qt

How to address the issues in the next sprint?

● How to improve the process?

Learn the QT framework in C++, there is more documentation on older versions of Qt and the use of methods in Qt are confusing. So more research needs to be done with the framework

● How to improve the product?

Set up a way to use the Qt widgets more effectively so later allow multimodal interaction and add devices.

**Sprint 6**

Attendees: Daniel Mederos, Jim Louro

Start time: 12:30 PM

End time: 1:00 PM

What went wrong?

● Did we do a good job estimating our team's velocity?

**Not really as using QAction and Qthreads in Qt did not aid in the multimodal process because it brought more issues than benefits, so more work was needed to implement the touch event handlers.**

● Did we do a good job estimating the points (time required) for each user story?

**Not really as we put in more hours overnight in this sprint to assure that we fixed all the bugs created with the multimodal canvas**

● Did each team member work as scheduled?

**Yes**.

What went right?

● The Multimodal canvas finally got up and running

How to address the issues in the next sprint?

● Test the overall project in multiple environments to assure expected results before demo

How to improve the process?

● Learn the QT framework in C++ much earlier to start of using most appropriate methods and functions so no extra work will be done by the developers and to use all the tools Qt has to offer correctly.

How to improve the product?

● With the Single and Multi canvas both up and running now it would be best to introduce new input devices to the program.

**References**

1. C++ reference

http://en.cppreference.com/w/

2. QT

<https://wiki.qt.io/Qt_5.8_Release>

<https://www.qt.io/developers/>

<http://doc.qt.io/qt-5/qtouchevent.html>

<https://www.quora.com/How-do-I-get-started-with-GUI-programming-in-C++-using-Qt-framework>