Documentation of 461 Drawing Project

Longwood Spring 2021 CS Senior Seminar Students

April 26, 2021

Contents

0.1 Preface

This compilation of documentation exists as a reference for students in the CMSC 461 Senior Seminar course at Longwood University. The Following code has been created by all the students of the class, and an attempt has been made to make a cross platform application that is useful and simple, and document it in a detailed manner.

System Dependencies

1.1 For Linux:

To get QT:

- sudo apt get update
- sudo apt get upgrade
- sudo apt install gt5-default

To get pdflatex

In order to use the LaTeX tool in the app you must have the pdflatex tool downloaded, this can be retrieved from https://miktex.org/download and click over to MacOS and click download

1.2 For Windows:

To get QT:

- Go to www.qt.io
- Click "Download. Try."
- Scroll down and click "Go open source"
- Scroll down to the bottom and click "Download the Qt Online installer"
- Scroll down and click "download"
- Open the downloaded program
- Follow installers instructions
- Make sure to click "Default Qt 5.15 desktop installation"

1.2.1 To get pdflatex

In order to use the LaTeX tool in the app you must have the pdflatex tool downloaded, this can be retrieved from https://miktex.org/download and clicking download.

1.3 For MacOS:

To get QT:

- Go to https://www.qt.io click on "Download. Try."
- Scroll down and click on "Go open source"
- Scroll down and click on "Download the Qt Online Installer"
- Click "Download"

To get pdflatex

In order to use the LaTeX tool in the app you must have the pdflatex tool downloaded, this can be retrieved from https://miktex.org/download and click over to MacOS and click download.

JSON Protocol

Defined here is what a JSON Object should look like for sending and receiving shapes.

2.1 Rectangles

For Rectangles:

```
{
  "data": {
    "bid": "<board ID>",
    "fill_color": "<a color in HEX>",
    "outline_color": "<a color in HEX>",
    "end": {
      "x": "<x2; end x; second corner>",
      "y": "<y2; end y; second corner>"
    },
    "sid": <shape ID>,
    "start": {
      "x": "<x1; start x; first corner>",
      "y": "<y2; start y; first corner>"
    }
  },
  "shape": "rect"
}
```

2.2 Ellipses

For Ellipses:

```
{
  "data": {
    "bid": "<board ID>",
    "fill_color": "<a color in HEX>",
    "outline_color": "<a color in HEX>",
    "end": {
      "x": "<x2; end x; second corner>",
      "y": "<y2; end y; second corner>"
    },
    "sid": <shape ID>,
    "start": {
      "x": "<x1; start x; first corner>",
```

```
      "y": "<y2; start y; first corner>"
    }
  },
  "shape": "ellipse"
}
```

2.3 LaTeX

For LaTeX:

```
{
    "data": {
        "bid": "<board ID>",
        "color": "<a color in HEX>",
        "start": {
            "x": "<x1; start x; first corner>",
            "y": "<y2; start y; first corner>"
        },
        "sid": "<Shape ID>",
        "text": "$<LaTeX math markup>$"

    },
    "shape": "text"
}
```

2.4 Lines

For Lines:

```
{
  "data": {
    "bid": "<board ID>",
    "color": "<a color in HEX>",
    "end": {
      "x": "<x2; end x>",
      "y": "<y2; end y>"
    },
    "sid": <shape ID>,
    "start": {
      "x": "<x1; start x>",
      "y": "<y1; start y>"
    }
  },
  "shape": "line"
}
```

2.5 Text

For Text:

```
{
    "data": {
        "bid": "<board ID>",
        "color": "<a color in HEX>",
        "start": {
            "x": "<x1; start x; first corner>",
            "y": "<y2; start y; first corner>"
        },
        "sid": "<shape ID>",
        "text": "<the text of the string>"
    },
    "shape": "text"
}
```

Client

The following is an explanation of the client side of the 461 Drawing Project.

3.1 ItemStats

3.1.1 .h

itemStats is a struct which contains all of the values which a shape/object contains. This is used in other documents and functions to help parse out and separate data.

```
struct itemStats
{
        //Lines get tracked the same as everything else.
        //Set the width/height variables to x2 and y2, respectively.
        std::string board_id;
    std::string type;
    std::string text;
    int id;
    double x;
    double y;
    double height;
    double width;

    double scenex;
    double sceney;

    QColor outline;
    QColor fill;


        //Functions to turn the thing into a QJsonObject
        //and a QByteArray that can be sent directly over a socket.
        QJsonObject toJson();
        QByteArray byteData();

        //Several different constructors for a couple of different
        //uses. You can pass in nothing, a QGraphicsItem* for the
        //client end, or a bunch of variables that it will simply
        //insert.
        itemStats();
        itemStats(std::string board_id, QGraphicsItem* item);
        itemStats(std::string board_id, std::string type, int id, double x, double y, double height, double width, double s
        itemStats(std::string board_id, std::string type, int id, double x, double y, double height, double width, double s
        itemStats(std::string board_id, std::string type, int id, double x, double y, std::string text, QColor outline); //
```

```
        ~itemStats();
};
```

3.1.2 .cpp

```
itemStats::itemStats(std::string nboard_id, std::string ntype, int nid, double nx, double ny, \
double nheight, double nwidth, QColor nrgb)
  //UPDATE THIS IF YOU ADD THINGS TO THE PLACED ITEMS

itemStats::itemStats(std::string nboard_id, QGraphicsItem* item)
  /*UPDATE THIS IF YOU ADD THINGS TO THE PLACED ITEMS
  Parses the QGraphicsItem and throws the variables into their appropriate places.*/

itemStats::~itemStats()
  //destructor for itemStats

QJsonObject itemStats::toJson()
  /*UPDATE THIS IF YOU ADD THINGS TO THE PLACED ITEMS
  Turns the item into a JSON object, formatted as it would be
  in the jsonExamples folder. This doesn't actually differ at
  all from different shape types.

  For the 'end' parameter, it uses the width and height as x and y coordinates.
  If the thing's a line, the end is just what it says on the tin - the ending coordinates of the line. */

QByteArray itemStats::byteData()
  /*Turns it first into a JSON object, then passes that object
        into a QByteArray. This array can be sent straight to the server.*/
```

3.2 ProjectScene

3.2.1 .h

projectScene is a class which contains functions that relate to the QGraphicsScene, that is, the 'behinds the scenes' of the graphical window.

```
class ProjectScene : public QGraphicsScene
{
        Q_OBJECT
        private:
                QTcpSocket* m_socket;
                std::vector<itemStats> m_tracked_items;
                std::string m_board_id;

        public slots:
                void sceneChanged(const QList<QRectF> &region);
                void readSocket();
                void disconnect();
        signals:
        public:
                int trackItem(QGraphicsItem* item);
                ProjectScene();
                ~ProjectScene();
};
```

3.2.2 .cpp

```
ProjectScene::ProjectScene()
  /*Sets up the socket that will be connected to the server. Connects signals from the socket to functions on the client,

ProjectScene::~ProjectScene()
  //A destructor for the ProjectScene class

ProjectScene::readSocket()
  //Reads in QByteArray data that is received from the server into a JSON object

ProjectScene::disconnect()
  /*Deletes the socket from the scene, effectively removing the connection from
    the server.*/

ProjectScene::updateCanvas()
  //Checks what has changed between different canvas' and updates them all if different

ProjectScene::trackItem()
  /*Pushes back all the data from a received graphics item into an internal list
    of graphics items.*/

ProjectScene::checkPos()
  //no comments in function

ProjectScene::sceneChanged()
  //checks to see if items on the canvas have changed in any way and passes the change to the server

ProjectScene::sceneChanged()
  /*When something in the scene changes, this event is triggered. What we do is
   get a list of items that have changed (compare between the last update and
   our internal list) and then tells the server about it.*/
```

3.3 ProjectView

3.3.1 .h

projectView is a class which contains the functions which relate to the QGraphicsView, that is, the canvas and buttons that we see on the graphical window.

```
class ProjectView : public QGraphicsView
{
        Q_OBJECT
        private:
                int m_tool;
                int m_color_r;
                int m_color_g;
                int m_color_b;
                //QBrush m_fill;

        QPointF firstClick;
        private slots:

        public:
        int get_m_tool();
```

```
        int get_m_color_r();
        int get_m_color_g();
        int get_m_color_b();

        void change_tool(int tool);
        void change_color(int r, int g, int b);
        void fill();
        void text_tool(qreal x, qreal y);
        void latex_tool(qreal x, qreal y);
        void arrow_tool(qreal x, qreal y, qreal x2, qreal y2);
        void circle_tool(qreal x, qreal y,qreal x2, qreal y2);
        void line_tool(qreal x, qreal y, qreal x2, qreal y2);
        void bezier_tool(qreal x, qreal y, qreal x2, qreal y2);
        void rect_tool(qreal x, qreal y, qreal x2, qreal y2);
        void mousePressEvent(QMouseEvent* event);
        void mouseReleaseEvent(QMouseEvent* event);
        ProjectView();
        ~ProjectView();
};
```

3.3.2 .cpp

```
ProjectView::ProjectView()
  //A constructor for the ProjectView class

ProjectView::~ProjectView()
  //A destructor for the ProjectView class

ProjectView::change_tool(int tool)
  //Change the current tool based on an integer parameter

ProjectView::change_color(int r, int g, int b)
  //Change the current tool's color based on 3 integer parameters, r, g, and b.

ProjectView::get_m_tool()
  //no comments in file

ProjectView::get_m_color_r()
  //no comments in file

ProjectView::get_m_color_g()
  //no comments in file

ProjectView::get_m_color_b()
  //no comments in file

ProjectView::fill()
  //no useful comments in file

ProjectView::text_tool()
  //no useful comments in file

ProjectView::latex_tool()
  //could use better comments i dont want to guess

ProjectView::arrow_tool()
```

```
  //no comments in file



ProjectView::circle_tool(qreal x, qreal y, qreal x2, qreal y2)
  /*Draw a circle on the canvas using two points, (x,y) and (x2,y2). This allows
    us to draw ellipses easily as well as circeles, similarly to how we draw
    rectangles.
    (x,y) is the first location the user clicked, and (x2,y2) is the second.*/

ProjectView::line_tool(qreal x, qreal y, qreal x2, qreal y2)
    /*Draw a line on the canvas using two points, (x,y) and (x2,y2).
    (x,y) is the first location the user clicked, and (x2,y2) is the second.*/

ProjectView::bezier_tool()
  //no useful comments

ProjectView::rect_tool(qreal x, qreal y, qreal x2, qreal y2)
    /*Draw a rectangle on the canvas using two points, (x,y) and (x2,y2).
    (x,y) is the first location the user clicked, and (x2,y2) is the second.*/

ProjectView::mousePressEvent()
     /*This event is triggered when the user clicks down on the mouse. Technically,
     it stores this point and doesn't do anything with it until a mouseReleaseEvent.*/

ProjectView::mouseReleaseEvent()
    /*This event is triggered when the user releases the click on the mouse. This
    function takes the second point, figures out which action to take based on
    the selected tool, and calls the appropriate function.*/
```

3.4 ToolBar

3.4.1 .h

toolbar is a class which contains the functions to change the different tools in the graphical window.

```
class ToolBar : public QWidget
{
        Q_OBJECT
        private:
                QPushButton* m_circle;
                QPushButton* m_line;
                QPushButton* m_rect;
                QPushButton* m_default;
                QPushButton* m_black;
                QPushButton* m_red;
                QPushButton* m_green;
                QPushButton* m_yellow;
                QPushButton* m_blue;
                QPushButton* m_color_picker;
                QVBoxLayout* m_layout;
                QPushButton* m_fill;
                QPushButton* m_text;
                QPushButton* m_latex;
                QPushButton* m_arrow;
                QPushButton* m_bezier;
```

```
        ProjectView* m_view;
    public slots:
        // set default
    void set_default();
    // shapes
    void place_rectangle();
    void set_line();
    void set_circle();
    // colors
    void set_color_black();
    void set_color_red();
    void set_color_green();
    void set_color_yellow();
    void set_color_blue();
    void set_color_custom();
    void fill();
    void set_text();
    void set_latex();
    void set_arrow();
    void set_bezier();
    public:
        ToolBar();
        ~ToolBar();
        void set_view(ProjectView* view);

};
```

3.4.2 .cpp

```
struct itemStats
{
    //Lines get tracked the same as everything else.
    //Set the width/height variables to x2 and y2, respectively.
    std::string board_id;
  std::string type;
  std::string text;
  int id;
  double x;
  double y;
  double height;
  double width;

  double scenex;
  double sceney;

  QColor outline;
  QColor fill;


    //Functions to turn the thing into a QJsonObject
    //and a QByteArray that can be sent directly over a socket.
    QJsonObject toJson();
    QByteArray byteData();

    //Several different constructors for a couple of different
    //uses. You can pass in nothing, a QGraphicsItem* for the
```

```
        //client end, or a bunch of variables that it will simply
        //insert.
        itemStats();
        itemStats(std::string board_id, QGraphicsItem* item);
        itemStats(std::string board_id, std::string type, int id, double x, double y, double height, double width, double s
        itemStats(std::string board_id, std::string type, int id, double x, double y, double height, double width, double s
        itemStats(std::string board_id, std::string type, int id, double x, double y, std::string text, QColor outline); //
        ~itemStats();
};
```

3.5 Window

3.5.1 .h

window is essentially the 'main' of the graphical window, though there is a 'main' that spawns it.

```
class Window : public QMainWindow //Extension on the base QMainWindow class
{
        Q_OBJECT //Must be included for qmake to recognize this

        private: //Menus and features of the window
                ToolBar* m_bar;
                QDockWidget* m_tool_dock;
                ProjectScene* m_scene;
                ProjectView* m_view;

        // shape tools

        private slots: //Where functions attached to buttons go
        // make a popup window
                void popup();
        public:
        ToolBar* get_m_bar();
        QDockWidget* get_m_tool_dock();
        ProjectScene* get_m_scene();
        ProjectView* get_m_view();

                Window();
                ~Window();
};
```

3.5.2 .cpp

```
ToolBar::ToolBar()
  //A constructor for the ToolBar class

ToolBar::~ToolBar()
  //A destructor for the ToolBar class

ToolBar::set_view(ProjectView* view)
  //Sets the tool's m_view to view

ToolBar::set_default()
  //Sets the default tool to the select tool
```

```
ToolBar::set_line()
  //Sets the tool to the line tool

ToolBar::set_circle()
  //Sets the tool to the circle tool

ToolBar::set_rect()
  //Sets the tool to the rect tool

ToolBar::set_color_black()
  //Sets the tool's color to be black

ToolBar::set_color_red()
  //Sets the tool's color to be red

ToolBar::set_color_green()
  //Sets the tool's color to be green

ToolBar::set_color_yellow()
  //Sets the tool's color to be yellow

ToolBar::set_color_blue()
  //Sets the tool's color to be blue

ToolBar::set_color_custom()
  /*Brings up a color picker menu for the user to set their own color using a
    graphical interface, rgb, hexadeicimal, or cmyk.*/

ToolBar::place_rectangle()
  //no comments

ToolBar::fill()
  //no comments

ToolBar::set_text()
  //no comments

ToolBar::set_latex()
  //no comments

ToolBar::set_arrow()
  //no comments

ToolBar::set_bezier()
  //no comments
```

3.6 Main

3.6.1 .cpp

main.cpp is a small file which contains one function: int main. This is the start of the entire client, where a new window is created in memory and executed. From there, upon returning, the event loop begins.

```
int main(int argc, char* argv[])
  /*This is the start of the entire client, where a new window is created in memory and executed.*/
```

Server

The server is run off of a Qt framework using Qt sockets. Clients can connect to the server via these Qt sockets, and are assigned unique identifiers using the socketDescriptor value of the connecting socket. By maintaining a constant connection, the server is able to receive and send the most current information about databases and shapes.

Running the server

To run the server use the command `./Server` followed by the port number you want it to run on. EX: `./Server 5000`

4.1 Server

4.1.1 .h

Server.h includes some tools to set up an item based on its data and turn it into a byte array. Also turn it into a QJsonObject.

```
/*This file includes tools to turn an item into a byte array based on its data, and also turn it into a QJsonObject.
struct ownedDB{
    int id;
    QSqlDatabase db;
};

class Server : public QMainWindow
{
        Q_OBJECT
        private:
                QTcpServer* m_server;
                QSet<QTcpSocket*> connected;
                std::string m_board_id;
        QVector<ownedDB> databases;

        std::vector<QJsonObject> m_shapes;

        public slots:
                void newConnection();
                void readSocket();
                void disconnect();
        public:
                Server();
                ~Server();
                void appendSocket(QTcpSocket* sock);
        void createBoard(QTcpSocket* socket);
                void saveDB(QTcpSocket* socket);
```

```
        void deleteDB(QTcpSocket* socket);
        void fullUpdate(QString databaseName, QTcpSocket* socket);
};
```

4.1.2 .cpp

```
Server::Server() : QMainWindow()
  /*Sets up the server itself. The server will give
  out a 'signal' when it receives a new connection.
  This connects that signal to the function 'newConnection()'.*/

Server::~Server(){
  //Destructor for the Server

void Server::newConnection()
  /*Checks for pending connections, then calls the appendSocket() that actually adds the socket to the list of sockets we ha

void Server::appendSocket(QTcpSocket* socket)
  //adds a new socket to all current clients are stored in the vector.

void Server::disconnect()
  //Determines which socket that needs to be disconnected.

void Server::readSocket()
  /*There is a QByteArray array of bytes that the socket has just sent to the server.
  The array is formatted as a JSON object. The client sends out this JSON object in
  the exact format that is specified in the jsonExamples folder. The data gets received and isvready to be parsed and turned

  QJsonDocument doc = QJsonDocument::fromJson(buf);
        QJsonObject obj = doc.object();

void Server::fullUpdate(QString databasename, QTcpSocket* socket){
  /*This function keeps track of all shapes across all clients concurrently via a QByteArray of JSON objects. In order to ac

void Server::createBoard(QTcpSocket* socket)
      /*Create a QSqlDatabase, either connect to the database named CMSC461.db or
      create it if it doesn't exist. Create and QSqlQuery pointer to be used to execute commands*/

void Server::deleteDB(QTcpSocket* socket)

Server::saveDB(QTcpSocket* socket)
/*Saves a massive JSON array of every shape in the database into a JSON object and sends it to a client who is about to disc
```

4.2 Main

4.2.1 .cpp

```
#include <QApplication>
#include "Server.h"
#include <iostream>

int main(int argc, char* argv[])
{
        QApplication new_window(argc, argv);
```

```
        Server w;
        return new_window.exec();
}
```

User Manual

This chapter provides an explanation of the canvas and how to manipulate the tools provided. Below is what a blank canvas will look like:

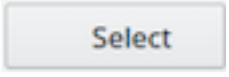
Figure 5.1: Blank Canvas

5.1 Joining/Creating a session

To join or create a session, the user will be prompted for an IP address, Port number, and Board ID. With that information a canvas will be created or joined if a matching one exists.

5.2 Manipulating Items

5.2.1 Select



The way to manipulate items drawn on the canvas is to use the "select" tool. With the select tool equipped click, hold, and drag an item to the desired location, then release the item and it will be relocated to its new position.

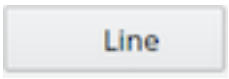
5.2.2 Deleting Items

The way to delete items drawn on the canvas is to also use the "select" tool. With the select tool equipped click the desired item, then click the backspace/delete key on your keyboard to remove it.

5.3 Drawing

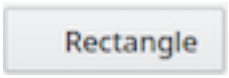
Below is an explanation on how to use each drawing tool

5.3.1 Lines



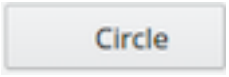
Select the line button, then on the canvas, click two separate points you want to draw a line between.

5.3.2 Rectangles



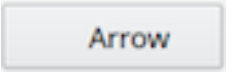
Select the rectangle button, then on the canvas, click one point to be the top right corner and click the second point to be the bottom left corner and a rectangle will be drawn in between.

5.3.3 Circle



Select the Circle button, then on the canvas, click two separate points, the first is the center point and the second is the radius. A circle will be drawn using those two reference points.

5.3.4 Arrow

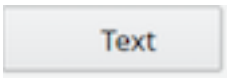


Select the Arrow button, then click two separate points on the canvas, the first is the tail of the arrow and the second is the head. An arrow will be drawn using those two reference points.

5.3.5 LaTeX

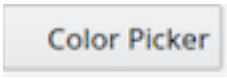
Select the Math button, then in the text box type in your desired LaTeX code and click the enter button.

5.3.6 Text



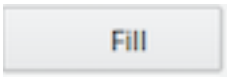
Select the text button, then in the text box type in your desired sentence and click the enter button.

5.3.7 Changing Colors



There are two ways to change color of the shapes, the first is to select one the colors on the side bar, the second is to click 'Color Picker' and choose a color based on RGB value.

5.3.8 Fill



The fill tool is used to fill a shape with a desired color. To use the tool first click the "fill" button then click the desired fill color, and finally click within the boundaries of the shape you want to fill.