

CSE 1325: Object-Oriented Programming

Lectures 12 and 13 – Chapter 16

(Using gtkmm)

More GUI and Drawing via CSE1325 Paint

Mr. George F. Rice

george.rice@uta.edu

Based on material by Bjarne Stroustrup
www.stroustrup.com/Programming

ERB 402
Office Hours:
Tuesday Thursday 11 - 12
Or by appointment

Quick Review

- A **class library** is a collection of prewritten classes, often designed as templates, that work together to facilitate writing applications.
 - Name some popular ones for C++. .NET and FCL, iOS Frameworks, Oracle Class Libraries, gtkmm / Gtk+, Qt, WxWidgets, FLTK, Boost
- **True** or False: Multiple namespace blocks with the same name are permitted.
- A **typedef** defines an alias for an (often complex) type, simplifying code and increasing readability.
- The **Adapter Pattern** implements a bridge between two classes with incompatible interfaces.
- The set of implementation elements and associated integration mechanisms necessary to meet the system requirements is called the system's **architecture**. The engineering role primarily responsible for it is the **architect**.
- **Generalization** is extracting shared characteristics from two or more classes, combining them into a generalized base class. The opposite, decomposing a general base class into specialized derived classes, is **specialization**.

Quick Review

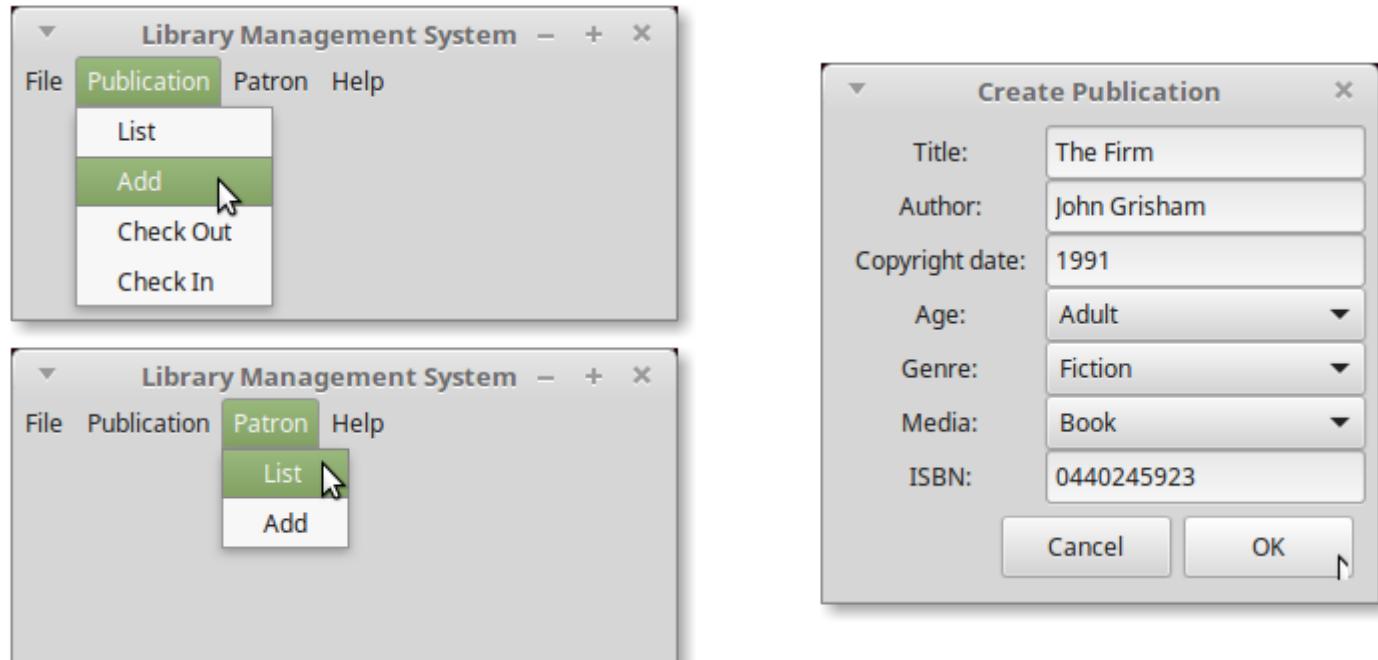
- An anonymous function object is called a **lambda**. They have a **capture clause** to determine which outer scope variables are visible, an optional parameter list, and a body.
- **True** or False: The parameter list of a lambda function, including the parentheses, are optional.
- True or **False**: By default, a lambda function can access all variables from the enclosing scope. **None are visible by default**.
- **True** or False: By default, a captured variable is copied. Prepend an & to capture the variable by reference.
- Describe how can all variables in the surrounding scope can be captured, first by reference, and then by access (copied). **[&], [=]**
- While the return type of a lambda function is usually inferred by the compiler, describe how to explicitly define the return type **[] -> bool { };**

Quick Review

- True or **False**: A friend function is a member of the class to which it is friend.
- Friend functions have access to its friend's (1) public members, (2) protected members, (3) private members, (4) 1 & 2, **(5) All of the above**
- The + operator may be defined for a class by specifying the **operator+** method or function.
- (1) All **(2) Most** (3) Some (4) Few C++ operators may be overloaded for C++ classes.
- **True** or False: An abstract class may be used as a base type and as a variable type, but may not be instanced.

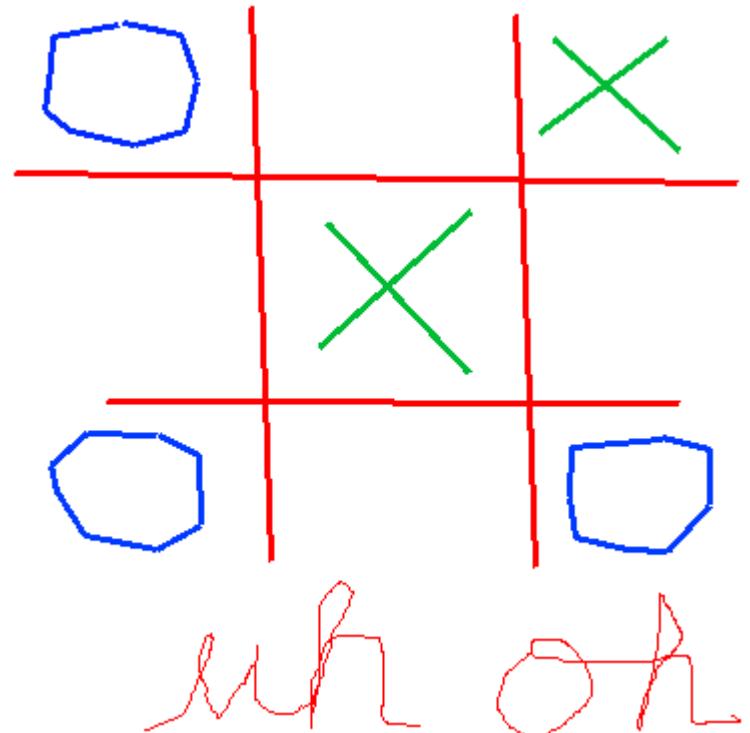
Homework #6 Questions?

- Homework #6 is due **Thursday, October 12 at 8 am.**
 - You have much leeway in your GUI design – simply meet the requirements
 - Doing the bonus is strongly recommended



Overview: GUI Widgets and Drawing

- A Simple GUI App
 - Exit with unsaved data warning
- Drawing Lines
 - Segments
 - Changing color and width
- Adding a Toolbar
- Adding File I/O
 - Save / Save As
 - Open / New
- Adding Draw Modes
 - Segments
 - Contiguous
 - Freehand
 - Dashed Lines
- Undo
- About



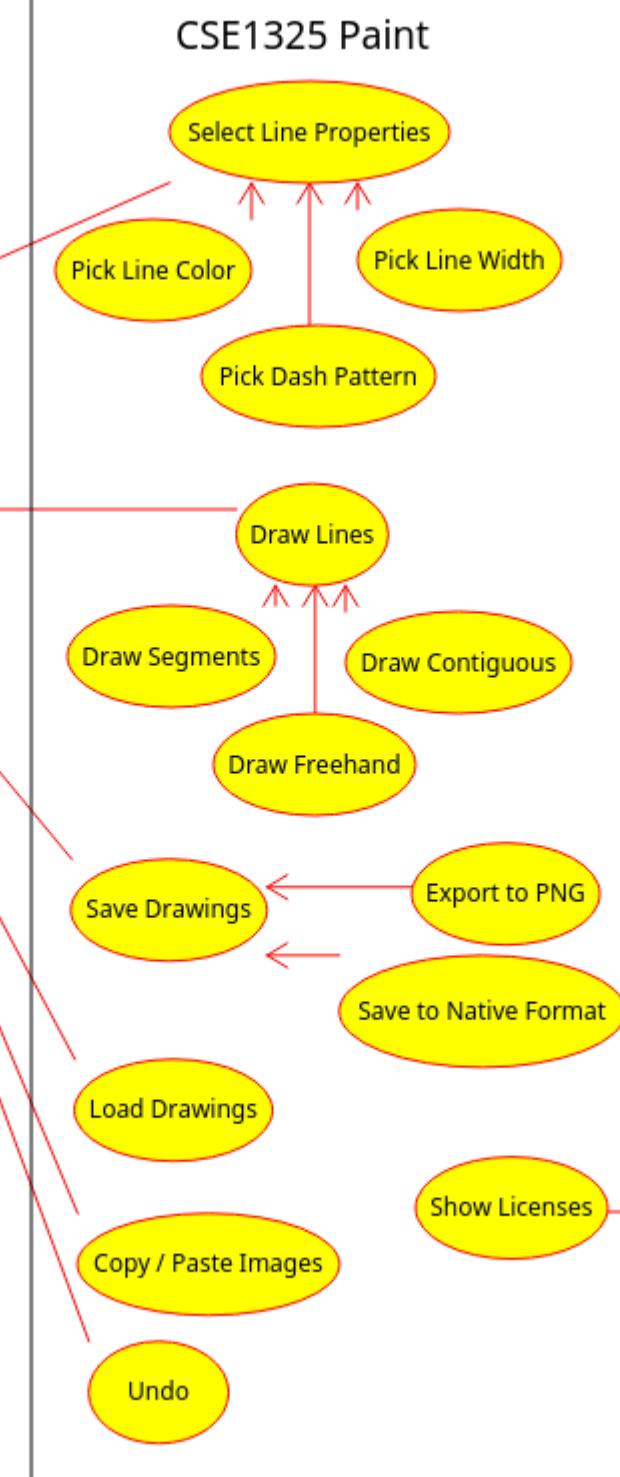
Building a Paint App with gtkmm

- To illustrate how widgets interact, let's build a very simple Paint application with C++ and gtkmm
 - We'll follow our Baby Steps philosophy
 - We'll analyze and design in the UML
 - We'll use Scrum for process management
 - We'll use git for version control
- So what's our first step?

Basic Use Case Diagram

Other Possible Use Cases

- Add Curved Lines
- Add Text
 - Select Font, Face, Size, Color
- Add Shapes
 - Circles / Ellipses, Squares / Rectangles, Triangles,
 - Fill with Color, Pattern, Image
 - Control Border as per Line
- Add Image
 - Bitmap, Vector
- Scripting
 - Turtle Graphics
 - Python
 - Public API
- [YOUR FAVORITE GOES HERE]



SCRUM Product Backlog

Status	As an... I want to...	So that I can...
Artist	Exit via x – check for “dirty” bit and prompt to save given new data	Close the window
Artist	Exit via File > Exit – creating a File > Exit menu item	Close the window independent of the OS
Artist	Draw discrete lines – click for (x1, y1), drag to (x2, y2)	Create line art
Artist	Change pen color – use a color selection dialog	Create colorful drawings
Artist	Change pen width – use a text input dialog and atoi	Provide more texture and movement
Artist	Select pen properties from a toolbar	Switch pen properties more quickly
Artist	Save – write the lines vector to a default filename	Reload, reuse, and share drawings
Artist	New canvas (vector) – toss drawing (prompt to save) and start new	Start a new drawing without closing the program
Artist	Open – load the lines vector for the default file	Reload, reuse, and share drawings
Artist	Save as – write the lines vector to a user-specified file	Manage my drawing files with less work
Artist	Open – load the lines vector from a user-selected file	Reload, reuse, and share drawings
Artist	Draw contiguous straight lines – sequence of clicks	Create polygons
Artist	Draw freehand – hold down the mouse and scribble	Create complex shapes
Artist	Draw dashed lines in all modes	Make my drawings more interesting and attractive
Artist	Undo	Make mistakes without having to start over
Artist	Run Turtle Graphics file	Create reusable complex shapes
Artist	Copy Image	Place my drawings into e.g., Office documents
Artist	Paste Image	Insert drawings from e.g., Gnome Screenshot
Artist	Export to PNG format	Save my drawings in an industry standard format
Lawyer	Acknowledge license obligations	Avoid litigation

Basic Menu / Toolbar Design

File

New

Open...

Save

Save As...

Export

Exit

Edit

Undo

Copy

Paste

Pen

Segment

Contiguous

Freehand

Line Color

Line Width

Dash Pattern

Turtles

Select Turtle

Help

About

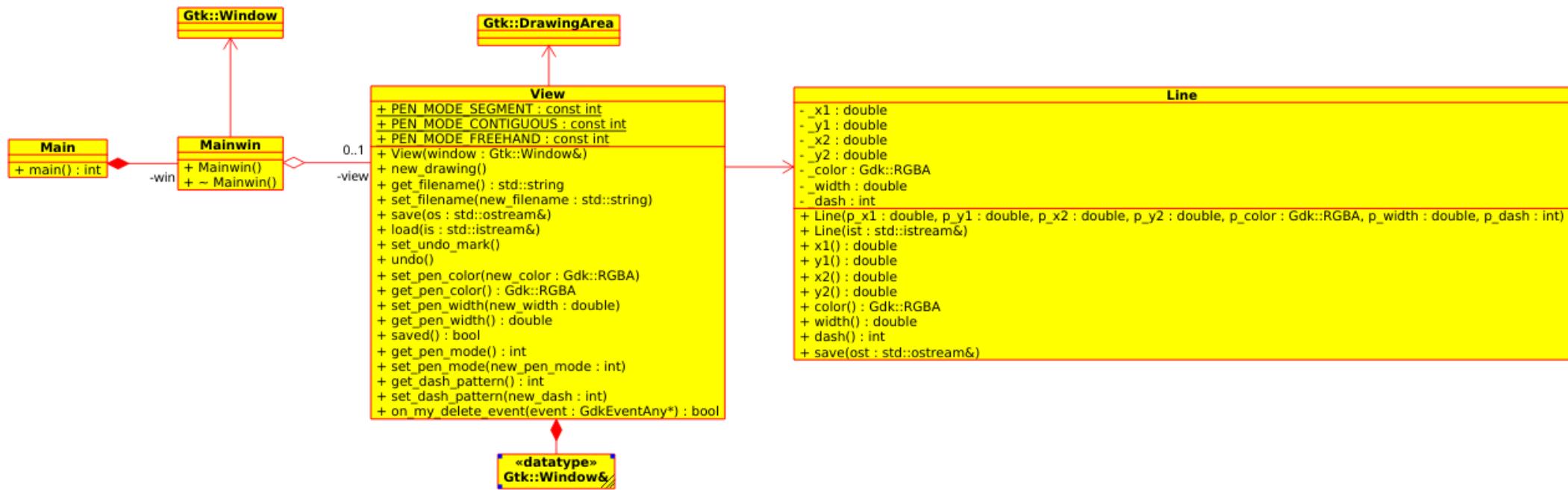
File

Pen

Line

New Open Save Undo | Segment Contiguous Freehand Dash | Color Width

Our Class Diagram



As usual, Main is not a class but a file with the main() function.

We'll baseline the code from our Turtle Graphics demo, which included a **Line** class and basic drawing mechanisms, and implement our sprints from there.

(A **baseline** is a reference point in our version control system, usually indicating completion and formal approval of a product release, and sometimes used as a starting point for a branch or new product. I'm using the word somewhat loosely here.)

Our Sprint #1 Backlog

(1 of 2)

Feature ID	Description	Status
XX	Observe signal_delete_event, implementing a dirty <u>bool</u> in View	
XFE	Derive a Mainwin class from <u>Gtk::Window</u> for customization	
XFE	Create File menu with Quit entry	
XFE	Initiate signal_delete_event when File > Quit is selected	
DDL	Add a <u>DrawingArea</u> to View	
DDL	Observe mouse clicks in the <u>DrawingArea</u> and create Line objects	
DDL	Override <u>DrawingArea::on_draw</u> and draw the line objects	
PC	Add set_pen_color method to View	
PC	Add Pen > Color menu item, display color chooser, and set in View	
PC	Add color attribute to Line	
PW	Add set_pen_width method to View	
PW	Add Pen > Width menu item	
PW	Create custom Set Pen Width dialog, set in View	
TB	Enable the <u>toolbar</u>	
TB	Add pen color button and connect signal	
TB	Add pen width button and connect signal	
SAV	Add Line::save to stream	
SAV	Add View::save to stream	
SAV	Add File > Save to menu	
SAV	Add save button to tool bar	
NEW	Add View::new	
NEW	Add File > New to menu	
NEW	Add new button to tool bar	
OP1	Add Line::Line(input stream)	
OP1	Add View::Open(filename)	
OP1	Add File > Open, passing default filename	
OP1	Add open button to tool bar	
SVA	Add <u>Mainwin::on_save_as_click</u> with file chooser dialog	
SVA	Add File > Save As to menu	
SVA	Save selected filename to view	
OPN	Add file chooser dialog to open	

Our Sprint #1 Backlog

(2 of 2)

DCL	Add View::set_draw_mode with constants
DCL	Add contiguous line drawing to mainwin
DCL	Add Pen > Contiguous and Pen > Segment to menu
DCL	Add contiguous and segment to tool bar
DFH	Add motion notify event handler to View
DFH	Add Pen > Freehand
DFH	Add freehand to tool bar
DSH	Add dash patterns, dash index, and getters / setters to View
DSH	Add dash index to Line
DSH	Update View::on_draw to use Line's dash setting
DSH	Add Pen > Dash with custom dialog to select dash pattern
UNDO	Add View::set_undo_mark and undo methods
UNDO	Set undo marks when a line segment is drawn
UNDO	Set undo marks when entering freehand mode
UNDO	Add Edit > Undo
UNDO	Add undo button to tool bar
TURT	Add Turtle base class from which programs will be derived
TURT	Add several common turtle graphics programs
TURT	Add View::new_turtle method to add lines to vector
TURT	Add Pen > Turtle
TURT	Add turtle button to toolbar
CP	Convert DrawingArea (View) to pixmap, then copy to buffer
CP	Add Edit > Copy
CP	Add copy to tool bar
PAS	Add Imager:Line type
PAS	Accept paste of pixmap
PAS	Convert pixmap to Imager object and add to vector
PAS	Add Edit > Paste
PAS	add paste to tool bar
EXP	Export pixmap (reused from CP) to PNG file
EXP	Add File > Export
ABT	Add About dialog
ABT	Add Help > About

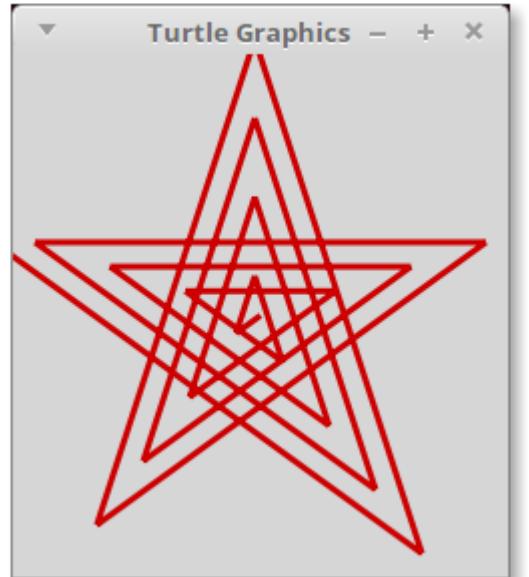
The Accompanying Files

- As usual, this slide deck includes an attached ZIP archive of supporting files and source code
 - The actual Scrum spreadsheet used for planning and managing this code development is attached
 - The “dev” directory contains the git repository activity
 - Numbered directories are code corresponding to the Step numbers (not feature numbers!) in the slides that follow
- Each source code directory contains a Makefile
 - Type “make” to compile
 - “make rebuild” or “make clean ; make” rebuilds from scratch
 - “make debug” rebuilds from scratch for the ddd debugger
 - Type “./paint” to run
 - Type “ddd paint” to debug

Step 000 – Baseline

- This is the source code from our Turtle Graphics demo
 - The executable here is named ‘stars’
- We’re primarily interested in reusing the Line and View classes
 - We’ll rip out a little and add a LOT for our Paint!
- Add the baseline to git!

`ee35027 Baseline from turtle graphics demo`



Step 001 – Refactor

- We need to rip out a little
 - Remove Turtle Graphics class and code in main()
 - Add View::dirty and saved() getter (both bool)

```
#ifndef _VIEW_H
#define _VIEW_H

#include "line.h"
#include <vector>
#include <gtkmm/drawingarea.h>

using namespace std;

class View : public Gtk::DrawingArea {
public:
    View(vector<Line> L);
    bool saved();
    bool on_draw(const Cairo::RefPtr<Cairo::Context>& cr) override;
private:
    vector<Line> lines;
    bool dirty = false; // True if any changes since last save
};

#endif
```

Step 001 – Refactor

```
// Draw the lines any time gtkmm needs to refresh the widget
bool View::on_draw(const Cairo::RefPtr<Cairo::Context>& cr) {
    // If nothing to draw, we're done
    if (lines.size() == 0) return true;

    // Create a Cairomm context to manage our drawing
    Gtk::Allocation allocation = get_allocation();

    // Center the drawing in the window
    const double width = allocation.get_width();
    const double height = allocation.get_height();
    cr->translate(width / 2, height / 2);

    // Use a 3 pixel wide red line
    cr->set_line_width(3.0);
    cr->set_source_rgb(0.8, 0.0, 0.0);

    // Draw the lines
    for(Line l : lines) {
        cr->move_to(l.x1(), l.y1());
        cr->line_to(l.x2(), l.y2());
    }
    // Apply the colors to the window
    cr->stroke();

    // Drawing was successful
    return true;
}
```

```
#include "view.h"
#include "line.h"
#include <vector>
#include <gtkmm/drawingarea.h>
#include <cairomm/context.h>

using namespace std;

View::View(vector<Line> L) : lines{L} { }

// True if all data has been written to disk
bool View::saved() {
    return !dirty;
}
```

view.cpp

Step 001 – Refactor

```
#include "view.h"
#include <gtkmm/application.h>
#include <gtkmm/window.h>

int main(int argc, char** argv)
{
    // Create the application
    auto app = Gtk::Application::create(argc, argv, "edu.uta.cse1325.turtle1");

    // Instance a window
    Gtk::Window win;
    win.set_title("CSE1325 Paint");

    // Instance a drawing surface
    // containing the lines and
    // add to the window
    View view{vector<Line>{}};
    win.add(view);
    view.show();

    return app->run(win);
}
```

main.cpp

```
#ifndef _LINE_H
#define _LINE_H

class Line {    // Define a line from (x1,
y1) to (x2, y2)
    double _x1, _y1, _x2, _y2;
public:
    Line(double p_x1, double p_y1,double
p_x2, double p_y2);
    double x1();
    double y1();
    double x2();
    double y2();
};

#endif
```

line.h

Step 001 – Refactor

```
CXXFLAGS = -std=c++11
```

Makefile

```
all: main
```

```
debug: CXXFLAGS += -g  
debug: rebuild
```

```
rebuild: clean main
```

```
main: main.o line.o view.o *.h  
      $(CXX) $(CXXFLAGS) -o paint main.o line.o view.o  
                           `/usr/bin/pkg-config gtkmm-3.0 --cflags --libs`
```

```
main.o: main.cpp *.h  
      $(CXX) $(CXXFLAGS) -c main.cpp `/usr/bin/pkg-config gtkmm-3.0 --cflags --libs`
```

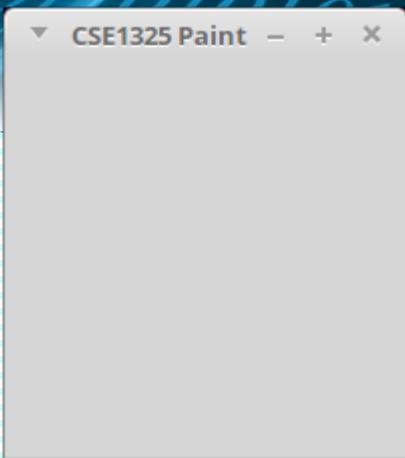
```
line.o: line.cpp *.h  
      $(CXX) $(CXXFLAGS) -c line.cpp
```

```
view.o: view.cpp *.h  
      $(CXX) $(CXXFLAGS) -c view.cpp `/usr/bin/pkg-config gtkmm-3.0 --cflags --libs`
```

```
clean:  
      -rm -f *.o paint
```

afffb70 Refactor, rm turtle.*

ee35027 Baseline from turtle graphics demo



Feature 1

Step 002 – Dialog on “Dirty”

- Before erasing the user's data (e.g., via exit), we should warn the user and give options
 - Discard the data
 - Save the data
 - Cancel the exit and continue editing
- This is often done with a “dirty bit”
 - True if unsaved data exists in the program
 - Set true when data is changed
 - Set false when data is saved
- Gtkmm exits when ‘x’ is clicked by default
 - We must intercept this to show our dialog using a signal

Product Backlog

Feature	Priority	Status	As an... I want to...	So that I can...
XX	1		Artist Exit via x – check for “dirty” bit and prompt to save given new data	Close the window

Sprint Backlog

Feature ID	Description	Status
XX	Observe signal_delete_event, implementing a dirty <u>bool</u> in View	

Feature 1

Step 002 – Dialog on “Dirty”

```
include "view.h"
#include <gtkmm/application.h>
#include <gtkmm/window.h>

int main(int argc, char** argv)
{
    // Create the application
    auto app = Gtk::Application::create(argc, argv, "edu.uta.cse1325.turtle1");

    // Instance a window
    Gtk::Window win;
    win.set_title("CSE1325 Paint");

    // Instance a drawing surface containing the lines and add to the window
    View view{win};
    win.add(view);
    view.show();

    // Allow view to determine if closing is permitted
    win.signal_delete_event().connect(sigc::mem_fun(view, &View::on_my_delete_event));

    // Return gtkmm's result code to the OS
    return app->run(win);
}
```

main.cpp

“When the main window signals a delete event,
call View’s on_my_delete_event method.”

Feature 1

Step 002 – Dialog on “Dirty”

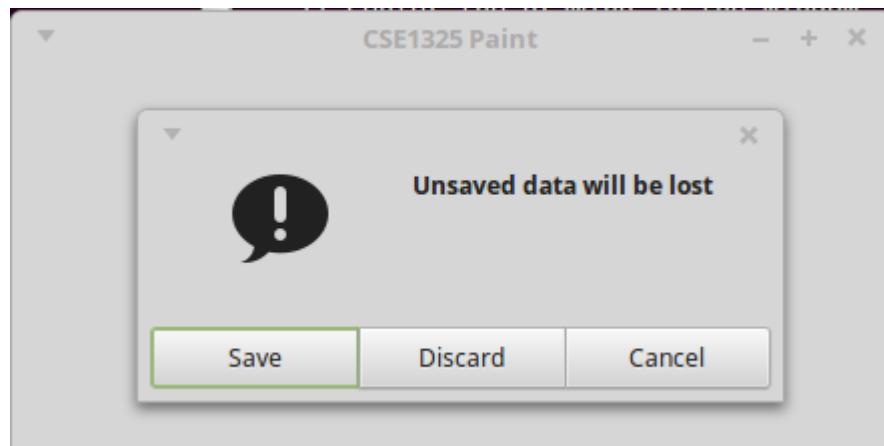
```
bool View::on_my_delete_event(GdkEventAny* event) {
    if (saved()) return false;
    Gtk::MessageDialog d{win, "Unsaved data will be lost", false,
                         Gtk::MESSAGE_WARNING, Gtk::BUTTONS_NONE};
    d.add_button("Save", 1);
    d.add_button("Discard", 2);
    d.add_button("Cancel", 3);
    int response = d.run();
    if (response == 1) {
        cout << "Saving data, then exit" << endl;
        return false;
    } else if (response == 2) {
        cout << "Discarding data and exiting" << endl;
        return false;
    } else {
        return true;
    }
}
```

view.cpp

“return true” means “the signal has been handled”.
Thus, no further handlers will be called and the application
will NOT exit.

“return false” means “the signal hasn’t been handled”.
Thus, additional handlers may be called, and the
application WILL exit.

Demonstrating Feature 1



```
2e10779 Exit via x, checking dirty bit
afffb70 Refactor, rm turtle.*
ee35027 Baseline from turtle graphics demo
```

SCRUM Product Backlog and Next Sprint Backlog Tasks

Status	As an... I want to...	So that I can...
Finished in Sprint 1	<p>Artist Exit via x – check for “dirty” bit and prompt to save given new data</p> <p>Artist Exit via File > Exit – creating a File > Exit menu item</p> <p>Artist Draw discrete lines – click for (x1, y1), drag to (x2, y2)</p> <p>Artist Change pen color – use a color selection dialog</p> <p>Artist Change pen width – use a text input dialog and atoi</p> <p>Artist Select pen properties from a toolbar</p> <p>Artist Save – write the lines vector to a default filename</p> <p>Artist New canvas (vector) – toss drawing (prompt to save) and start new</p> <p>Artist Open – load the lines vector for the default file</p> <p>Artist Save as – write the lines vector to a user-specified file</p> <p>Artist Open – load the lines vector from a user-selected file</p> <p>Artist Draw contiguous straight lines – sequence of clicks</p> <p>Artist Draw freehand – hold down the mouse and scribble</p> <p>Artist Draw dashed lines in all modes</p> <p>Artist Undo</p> <p>Artist Run Turtle Graphics file</p> <p>Artist Copy Image</p> <p>Artist Paste Image</p> <p>Artist Export to PNG format</p> <p>Lawyer Acknowledge license obligations</p>	<p>Close the window</p> <p>Close the window independent of the OS</p> <p>Create line art</p> <p>Create colorful drawings</p> <p>Provide more texture and movement</p> <p>Switch pen properties more quickly</p> <p>Reload, reuse, and share drawings</p> <p>Start a new drawing without closing the program</p> <p>Reload, reuse, and share drawings</p> <p>Manage my drawing files with less work</p> <p>Reload, reuse, and share drawings</p> <p>Create polygons</p> <p>Create complex shapes</p> <p>Make my drawings more interesting and attractive</p> <p>Make mistakes without having to start over</p> <p>Create reusable complex shapes</p> <p>Place my drawings into e.g., Office documents</p> <p>Insert drawings from e.g., Gnome Screenshot</p> <p>Save my drawings in an industry standard format</p> <p>Avoid litigation</p>

Feature ID	Description	Status
XFE	Derive a Mainwin class from Gtk::Window for customization	
XFE	Create File menu with Quit entry	
XFE	Initiate signal_delete_event when File > Quit is selected	

Feature 2

Step 003 – Main Menu + Exit

```
#ifndef MAIN_WINDOW_H  
#define MAIN_WINDOW_H  
  
#include <gtkmm.h>  
  
class Mainwin : public Gtk::Window {  
public:  
    Mainwin();  
    virtual ~Mainwin();  
protected:  
    // Exit the program  
    void on_quit_click();  
};  
#endif
```

mainwin.h

“on_quit_click” is called by gtkmm when File > Quit is clicked.

We use close() instead of hide(), because the former generates the delete event, triggering our “dirty bit” warning dialog.

```
#include "mainwin.h"  
  
Mainwin::~Mainwin() { }  
  
void Mainwin::on_quit_click() {  
    close(); // This is equivalent to clicking the 'x' in the window manager  
            // (Otherwise, we would use hide() instead.)  
}
```

mainwin.cpp
(1 of 2)

Feature 2

Step 003 – Main Menu + Exit

```
Mainwin::Mainwin() {  
    set_default_size(640, 480);
```

mainwin.h

```
    // Put a vertical box container as the Window contents  
    Gtk::Box *vbox = Gtk::manage(new Gtk::Box{Gtk::ORIENTATION_VERTICAL, 0});  
    add(*vbox);
```

This code creates our main window and adds a File > Quit menu entry.
We'll continue to build our menu (and eventually, our toolbar) here.

```
// /////  
// M E N U
```

```
    // Add a menu bar as the top item in the vertical box  
    Gtk::MenuBar *menubar = Gtk::manage(new Gtk::MenuBar{});  
    vbox->pack_start(*menubar, Gtk::PACK_SHRINK, 0);
```

```
//      F I L E
```

```
    // Create a File menu and add to the menu bar  
    Gtk::MenuItem *menuitem_file = Gtk::manage(new Gtk::MenuItem{"_File", true});  
    menubar->append(*menuitem_file);  
    Gtk::Menu *filemenu = Gtk::manage(new Gtk::Menu{});  
    menuitem_file->set_submenu(*filemenu);
```

```
//      Q U I T
```

```
    // Append Quit to the File menu  
    Gtk::MenuItem *menuitem_quit = Gtk::manage(new Gtk::MenuItem{"_Quit", true});  
    menuitem_quit->signal_activate().connect(sigc::mem_fun(*this,  
        &Mainwin::on_quit_click));  
    filemenu->append(*menuitem_quit);
```

```
vbox->show_all();
```

```
}
```

Feature 2

Step 003 – Main Menu + Exit

```
rebuild: clean all
```

Makefile

```
debug: CXXFLAGS += -g  
debug: rebuild
```

```
main: main.o mainwin.o line.o view.o *.h  
      $(CXX) $(CXXFLAGS) -o paint main.o mainwin.o line.o view.o `/usr/bin/pkg-config  
gtkmm-3.0 --cflags --libs`  
main.o: main.cpp *.h  
      $(CXX) $(CXXFLAGS) -c main.cpp `/usr/bin/pkg-config gtkmm-3.0 --cflags --libs`  
mainwin.o: mainwin.cpp *.h  
      $(CXX) $(CXXFLAGS) -c mainwin.cpp `/usr/bin/pkg-config gtkmm-3.0 --cflags --libs`  
view.o: view.cpp *.h  
      $(CXX) $(CXXFLAGS) -c view.cpp `/usr/bin/pkg-config gtkmm-3.0 --cflags --libs`  
line.o: line.cpp *.h  
      $(CXX) $(CXXFLAGS) -c line.cpp `/usr/bin/pkg-config gtkmm-3.0 --cflags --libs`
```

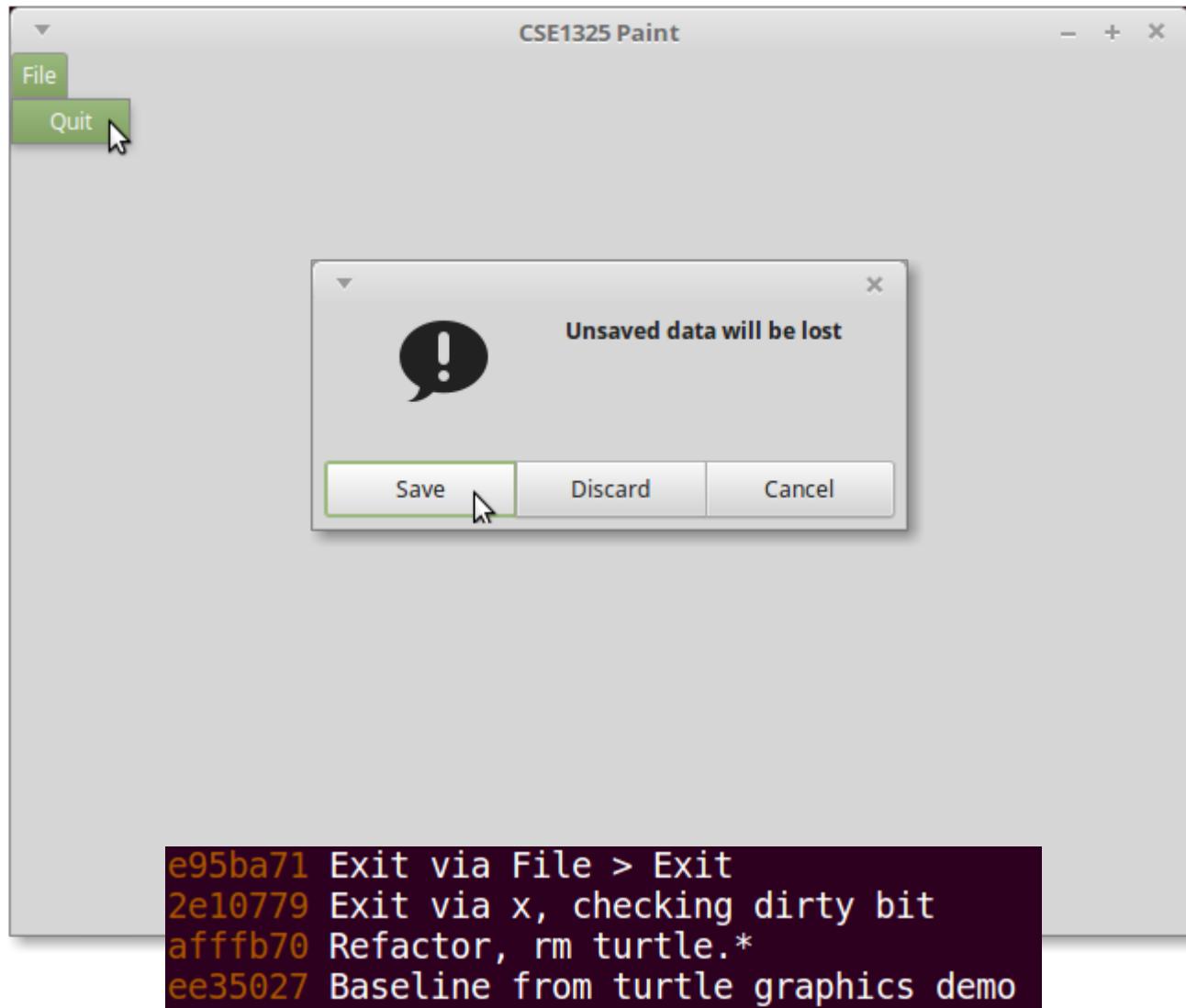
```
clean:
```

```
    -rm -f *.o paint
```

```
div:
```

```
@echo  
@echo 'X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-'  
@echo '-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X'  
@echo 'X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-'  
@echo '-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X'
```

Demonstrating Feature 2



SCRUM Product Backlog and Next Sprint Backlog Tasks

Status	As an... I want to...	So that I can...
Finished in Sprint 1	Artist Exit via x – check for "dirty" bit and prompt to save given new data	Close the window
Finished in Sprint 1	Artist Exit via File > Exit – creating a File > Exit menu item	Close the window independent of the OS
	Artist Draw discrete lines – click for (x1, y1), drag to (x2, y2)	Create line art
	Artist Change pen color – use a color selection dialog	Create colorful drawings
	Artist Change pen width – use a text input dialog and atoi	Provide more texture and movement
	Artist Select pen properties from a toolbar	Switch pen properties more quickly
	Artist Save – write the lines vector to a default filename	Reload, reuse, and share drawings
	Artist New canvas (vector) – toss drawing (prompt to save) and start new	Start a new drawing without closing the program
	Artist Open – load the lines vector for the default file	Reload, reuse, and share drawings
	Artist Save as – write the lines vector to a user-specified file	Manage my drawing files with less work
	Artist Open – load the lines vector from a user-selected file	Reload, reuse, and share drawings
	Artist Draw contiguous straight lines – sequence of clicks	Create polygons
	Artist Draw freehand – hold down the mouse and scribble	Create complex shapes
	Artist Draw dashed lines in all modes	Make my drawings more interesting and attractive
	Artist Undo	Make mistakes without having to start over
	Artist Run Turtle Graphics file	Create reusable complex shapes
	Artist Copy Image	Place my drawings into e.g., Office documents
	Artist Paste Image	Insert drawings from e.g., Gnome Screenshot
	Artist Export to PNG format	Save my drawings in an industry standard format
Lawyer	Acknowledge license obligations	Avoid litigation

Feature ID	Description	Status	Notes
DDL	Add a DrawingArea to View	Inherited from the baseline	
DDL	Observe mouse clicks in the DrawingArea and create Line objects		
DDL	Override DrawingArea::on_draw and draw the line objects		Inherited from the baseline

Feature 3

Step 004 – Draw Discrete Lines

- We move the View instance and delete _event signal connection from main to Mainwin
 - Because it's a more natural fit
- In View's constructor, we enable the button_press_mask event so we can detect clicks
- We add a View::new_line method to add lines to our vector of Line objects representing our drawing
- We override the on_button_press_event method (which View inherited from DrawingArea) to collect data and call View::new_line as needed

Feature 3

Step 004 – Draw Discrete Lines

```
#ifndef _VIEW_H
#define _VIEW_H

#include "line.h"
#include <vector>
#include <gtkmm/drawingarea.h>

using namespace std;

class View : public Gtk::DrawingArea {
public:
    View(Gtk::Window& window);
    bool saved();
    bool on_draw(const Cairo::RefPtr<Cairo::Context>& cr) override; // Redraw screen
    bool on_my_delete_event(GdkEventAny* event); // catch click on 'x' in title bar

protected:
    // Response to a button press
    bool on_button_press_event(GdkEventButton * event) override;

private:
    Gtk::Window& win;
    int x1, y1;
    vector<Line> lines;
    void new_line(int _x1, int _y1, int _x2, int _y2);
    bool dirty = false;
    bool click_in_progress = false; // distinguish 1st click from 2nd in line drawing
};

#endif
```

view.h

Feature 3

Step 004 – Draw Discrete Lines

```
View::View(Gtk::Window& window) : win{window} {  
    add_events(Gdk::BUTTON_PRESS_MASK);  
}
```

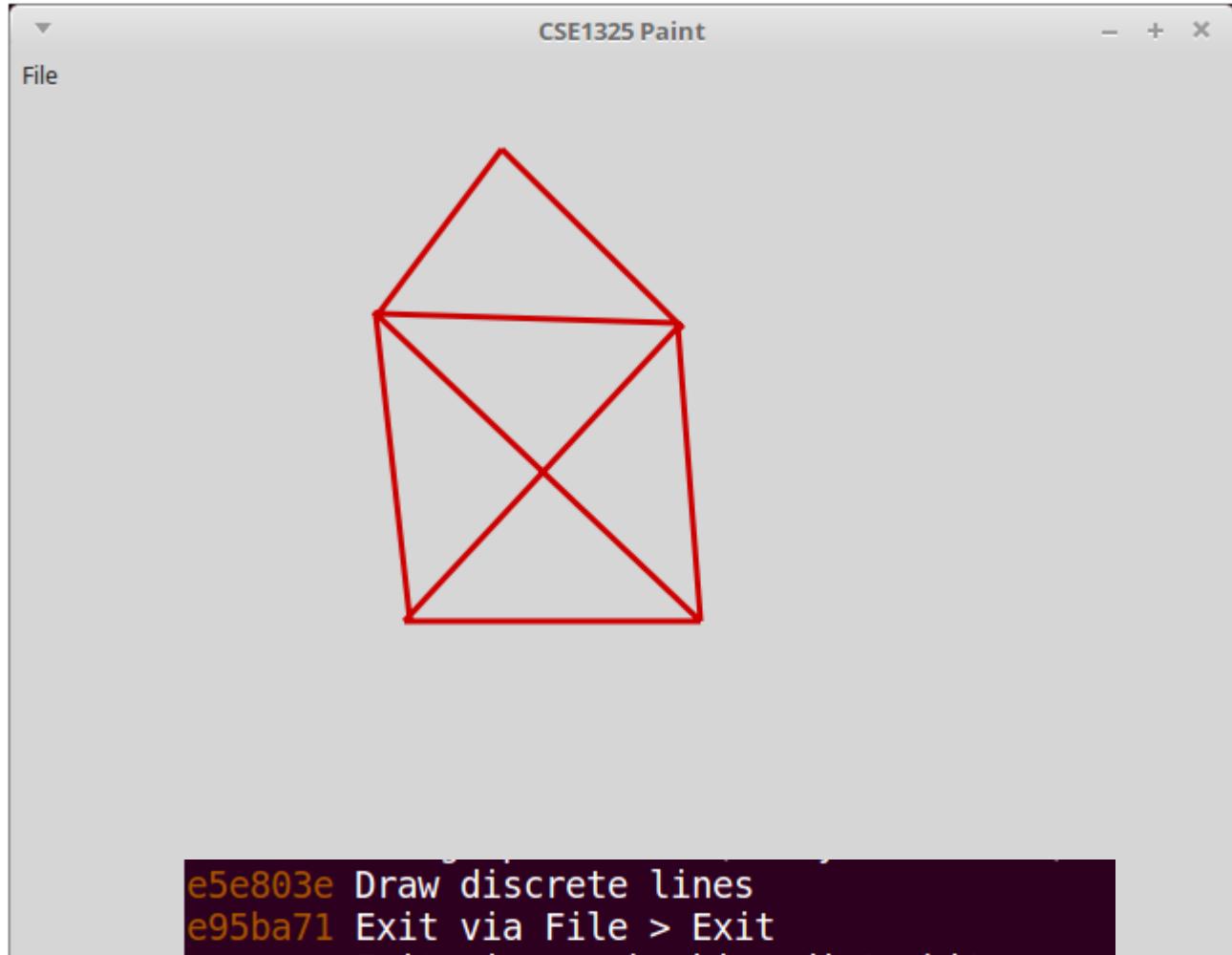
view.cpp

```
// Add a line to the vector  
void View::new_line(int _x1, int _y1, int _x2, int _y2) {  
    lines.push_back(Line(_x1, _y1, _x2, _y2));  
    dirty = true;  
}  
  
bool View::on_button_press_event(GdkEventButton * event) {  
    // If this is a left-mouse button press, we're interested!  
    if( (event->type == GDK_BUTTON_PRESS) && (event->button == 1)) {  
  
        // Just remember the coordinates on the first click, but  
        // add a new line on the second click  
        if (!click_in_progress) {  
            x1 = event->x;  
            y1 = event->y;  
            click_in_progress = true;  
        } else {  
            new_line(x1, y1, event->x, event->y);  
            click_in_progress = false;  
            queue_draw(); // initiate screen refresh  
        }  
        return true; // We handled this event  
    }  
    return false; // We did NOT handle this event  
}
```

If button 1 (left) is pressed,
we alternate between remembering
(x1, y1) on first click, then adding a
(x1,y1)-(x2,y2) line on second click.

DrawingArea::queue_draw requests
that the drawing area be redrawn.

Demonstrating Feature 3



SCRUM Product Backlog and Next Sprint Backlog Tasks

Status	As an... I want to...	So that I can...
Finished in Sprint 1	Artist Exit via x – check for “dirty” bit and prompt to save given new data	Close the window
Finished in Sprint 1	Artist Exit via File > Exit – creating a File > Exit menu item	Close the window independent of the OS
Finished in Sprint 1	Artist Draw discrete lines – click for (x1, y1), drag to (x2, y2)	Create line art
	Artist Change pen color – use a color selection dialog	Create colorful drawings
	Artist Change pen width – use a text input dialog and <code>atoi</code>	Provide more texture and movement
	Artist Select pen properties from a <u>toolbar</u>	Switch pen properties more quickly
	Artist Save – write the lines vector to a default filename	Reload, reuse, and share drawings
	Artist New canvas (vector) – toss drawing (prompt to save) and start new	Start a new drawing without closing the program
	Artist Open – load the lines vector for the default file	Reload, reuse, and share drawings
	Artist Save as – write the lines vector to a user-specified <u>file</u>	Manage my drawing files with less work
	Artist Open – load the lines vector from a user-selected file	Reload, reuse, and share drawings
	Artist Draw contiguous straight lines – sequence of clicks	Create polygons
	Artist Draw freehand – hold down the mouse and scribble	Create complex shapes
	Artist Draw dashed lines in all modes	Make my drawings more interesting and attractive
	Artist Undo	Make mistakes without having to start over
	Artist Run Turtle Graphics file	Create reusable complex shapes
	Artist Copy Image	Place my drawings into e.g., Office documents
	Artist Paste Image	Insert drawings from e.g., Gnome Screenshot
	Artist Export to PNG format	Save my drawings in an industry standard format
Lawyer	Acknowledge license obligations	Avoid litigation

Feature ID	Description	Status
PC	Add set_pen_color method to View	
PC	Add Pen > Color menu item, display color chooser, and set in View	
PC	Add color attribute to Line	

Feature 4

Step 005 – Change Pen Color

- Gtkmm has a color selection dialog for this
- We'll need to store the currently selected color with each Line stored in our lines vector

```
#ifndef _LINE_H
#define _LINE_H
#include <gdkmm/rgba.h>

class Line { // Define a line from (x1, y1) to (x2, y2)
    double _x1, _y1, _x2, _y2;
    Gdk::RGBA _color;
public:
    Line(double p_x1, double p_y1, double p_x2, double p_y2, Gdk::RGBA p_color);
    double x1();
    double y1();
    double x2();
    double y2();
    Gdk::RGBA color();
};

#endif
```

line.h

Some GUI class libraries (ahem, FLTK) store a color as a simple int. But gtkmm uses a class – RGBA (for Red, Green, Blue, Alpha).

Alpha sets a level of transparency.

Feature 4

Step 005 – Change Pen Color

- Use Line's pen color to update the screen

```
// Draw the lines any time gtkmm needs to refresh the widget
bool View::on_draw(const Cairo::RefPtr<Cairo::Context>& cr) {
    // If nothing to draw, we're done
    if (lines.size() == 0) return true;

    // Create a Cairomm context to manage our drawing
    Gtk::Allocation allocation = get_allocation();

    // Use a 3 pixel wide line of the user-selected color
    cr->set_line_width(3.0);

    // Draw the lines
    for(Line l : lines) {
        Gdk::Cairo::set_source_rgba(cr, l.color());
        cr->move_to(l.x1(), l.y1());
        cr->line_to(l.x2(), l.y2());
        cr->stroke();
    }

    // Drawing was successful
    return true;
}
```

view.cpp

Feature 4

Step 005 – Change Pen Color

- Add Pen > Color to the menu after File > Quit

```
//      P E N
// Create a Pen menu and add to the menu bar
Gtk::MenuItem *menuitem_pen = Gtk::manage(new Gtk::MenuItem("_Pen", true));
menubar->append(*menuitem_pen);
Gtk::Menu *penmenu = Gtk::manage(new Gtk::Menu());
menuitem_pen->set_submenu(*penmenu);

//          C O L O R
// Append Color to the Pen menu
Gtk::MenuItem *menuitem_color = Gtk::manage(new Gtk::MenuItem("_Color", true));
menuitem_color->signal_activate().connect(sigc::mem_fun(*this,
    &Mainwin::on_color_click));
penmenu->append(*menuitem_color);

void Mainwin::on_color_click() {
    Gtk::ColorChooserDialog dialog("Please choose a color");
    dialog.set_transient_for(*this); // Avoid the "discouraging" warning

    dialog.set_rgba(view->get_pen_color()); // Set the current color as default

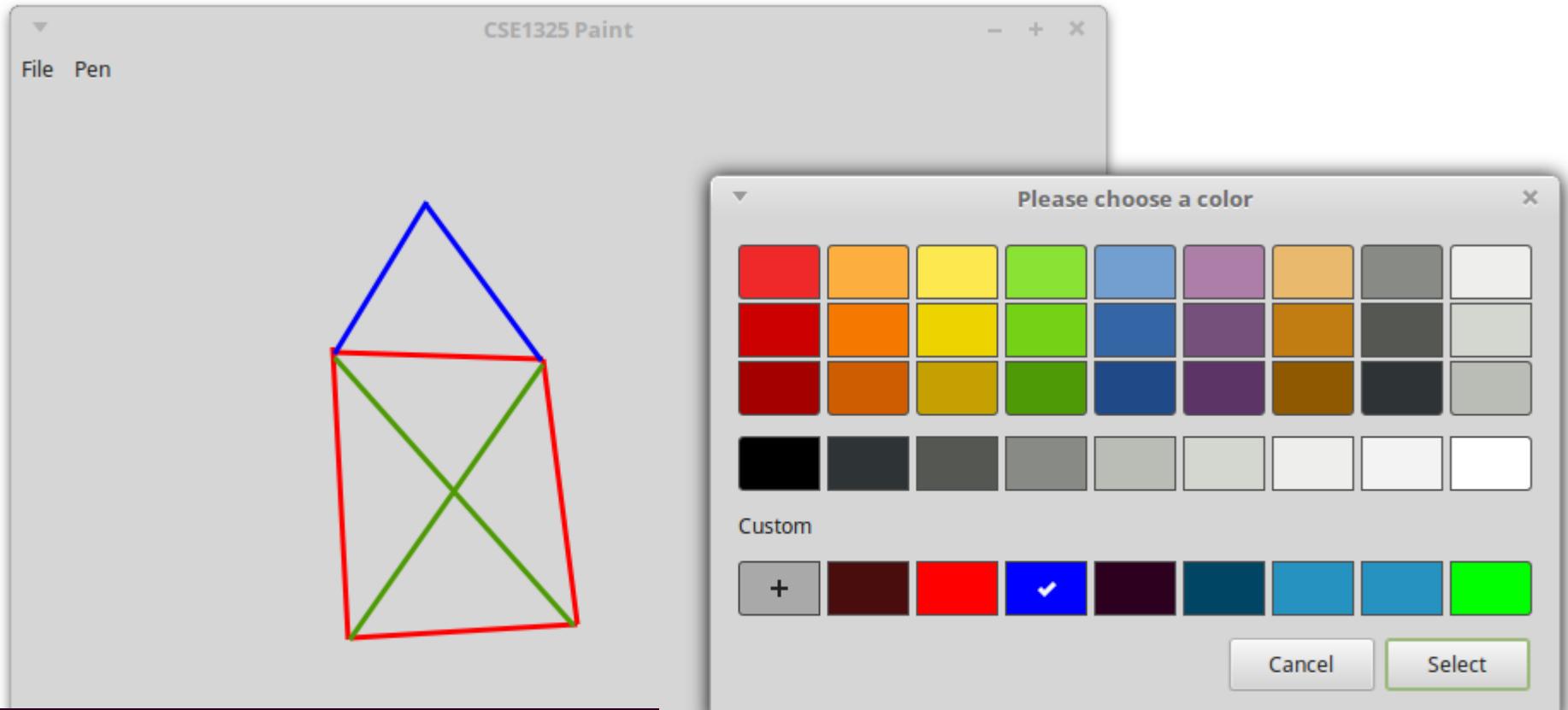
    int result = dialog.run();

    if (result == Gtk::RESPONSE_OK)
        view->set_pen_color(dialog.get_rgba());
}
```

mainwin.cpp

Only added code is shown.
Code may not be contiguous in file!

Demonstrating Feature 4



```
033a45d Change pen color (done!)
5699aec Change pen color (not yet in Line)
e5e803e Draw discrete lines
e95ba71 Exit via File > Exit
2e10779 Exit via x, checking dirty bit
afffb70 Refactor, rm turtle.*
ee35027 Baseline from turtle graphics demo
```

SCRUM Product Backlog and Next Sprint Backlog Tasks

Status	As an... I want to...	So that I can...
Finished in Sprint 1	Artist Exit via x – check for "dirty" bit and prompt to save given new data	Close the window
Finished in Sprint 1	Artist Exit via File > Exit – creating a File > Exit menu item	Close the window independent of the OS
Finished in Sprint 1	Artist Draw discrete lines – click for (x1, y1), drag to (x2, y2)	Create line art
Finished in Sprint 1	Artist Change pen color – use a color selection dialog	Create colorful drawings
	Artist Change pen width – use a text input dialog and <u>atoi</u>	Provide more texture and movement
	Artist Select pen properties from a <u>toolbar</u>	Switch pen properties more quickly
	Artist Save – write the lines vector to a default filename	Reload, reuse, and share drawings
	Artist New canvas (vector) – toss drawing (prompt to save) and start new	Start a new drawing without closing the program
	Artist Open – load the lines vector for the default file	Reload, reuse, and share drawings
	Artist Save as – write the lines vector to a user-specified file	Manage my drawing files with less work
	Artist Open – load the lines vector from a user-selected file	Reload, reuse, and share drawings
	Artist Draw contiguous straight lines – sequence of clicks	Create polygons
	Artist Draw freehand – hold down the mouse and scribble	Create complex shapes
	Artist Draw dashed lines in all modes	Make my drawings more interesting and attractive
	Artist Undo	Make mistakes without having to start over
	Artist Run Turtle Graphics file	Create reusable complex shapes
	Artist Copy Image	Place my drawings into e.g., Office documents
	Artist Paste Image	Insert drawings from e.g., Gnome <u>Screenshot</u>
	Artist Export to PNG format	Save my drawings in an industry standard format
Lawyer	Acknowledge license obligations	Avoid litigation

Feature ID	Description	Status
PW	Add set_pen_width method to View	
PW	Add Pen > Width menu item	
PW	Create custom Set Pen Width dialog, set in View	

Feature 5

Step 006 – Change Pen Width

- Gtkmm has no pre-defined dialog for this – had planned to use text, but why not a slider?
- We'll need to store the currently selected color with each Line stored in our lines vector

```
#ifndef _LINE_H
#define _LINE_H
#include <gdkmm/rgba.h>

class Line {    // Define a line from (x1, y1) to (x2, y2)
    double _x1, _y1, _x2, _y2;
    Gdk::RGBA _color;
    double _width;
public:
    Line(double p_x1, double p_y1, double p_x2, double p_y2,
          Gdk::RGBA p_color, double p_width);
    double x1();
    double y1();
    double x2();
    double y2();
    Gdk::RGBA color();
    double width();
};

#endif
```

line.h

Feature 5

Step 006 – Change Pen Width

```
//      W I D T H                                     mainwin.cpp
// Append Width to the Pen menu
Gtk::MenuItem *menuitem_width = Gtk::manage(new Gtk::MenuItem{"_Width", true});
menuitem_width->signal_activate().connect(sigc::mem_fun(*this,
    &Mainwin::on_width_click));
penmenu->append(*menuitem_width);

void Mainwin::on_width_click() {
    // Create a custom dialog for setting line width
    Gtk::Dialog dialog{"Select Pen Width", *this};

    // The lower widget is a slider (Gtk "scale"), so the user can mouse the new width
    Gtk::Scale scale{};
    scale.set_range(1.0,10.0);
    scale.set_value(view->get_pen_width()); // Preset scale to current width
    scale.show();
    dialog.get_vbox()->pack_start(scale);

    // OK and Cancel buttons
    dialog.add_button("Cancel", 0);
    dialog.add_button("OK", 1);
    dialog.set_default_response(1);

    // Show the dialog, get the button pressed
    dialog.show_all();
    int result = dialog.run();
    if (result == 1) view->set_pen_width(scale.get_value());
    dialog.close();
}
```

A text box in the upper area would be nice, so the user can type in a width as well. It's in the next version! (Old programmers' joke)

Feature 4

Step 005 – Change Pen Color

- Use Line's pen color to update the screen

```
// Draw the lines any time gtkmm needs to refresh the widget
bool View::on_draw(const Cairo::RefPtr<Cairo::Context>& cr) {
    // If nothing to draw, we're done
    if (lines.size() == 0) return true;

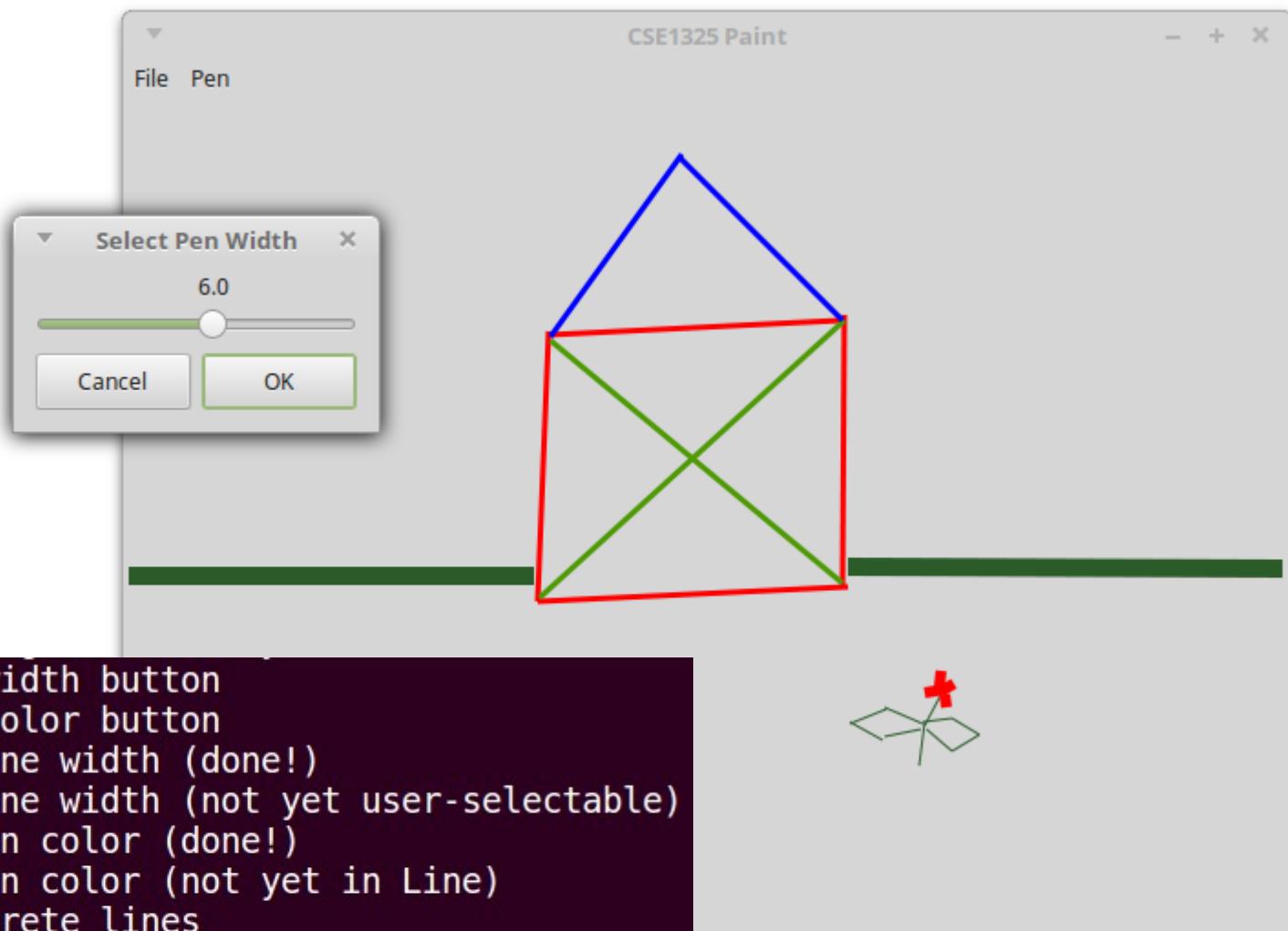
    // Create a Cairomm context to manage our drawing
    Gtk::Allocation allocation = get_allocation();

    // Draw the lines
    for(Line l : lines) {
        Gdk::Cairo::set_source_rgba(cr, l.color());
        cr->set_line_width(l.width());
        cr->move_to(l.x1(), l.y1());
        cr->line_to(l.x2(), l.y2());
        cr->stroke();
    }

    // Drawing was successful
    return true;
}
```

view.cpp

Demonstrating Feature 5



```
2a390f4 Add pen width button
4be73a9 Add pen color button
38ba644 Change line width (done!)
cd6bdffb Change line width (not yet user-selectable)
033a45d Change pen color (done!)
5699aec Change pen color (not yet in Line)
e5e803e Draw discrete lines
e95ba71 Exit via File > Exit
2e10779 Exit via x, checking dirty bit
afffb70 Refactor, rm turtle.*
ee35027 Baseline from turtle graphics demo
```

SCRUM Product Backlog and Next Sprint Backlog Tasks

Status	As an... I want to...	So that I can...
Finished in Sprint 1	Artist Exit via x – check for “dirty” bit and prompt to save given new data	Close the window
Finished in Sprint 1	Artist Exit via File > Exit – creating a File > Exit menu item	Close the window independent of the OS
Finished in Sprint 1	Artist Draw discrete lines – click for (x1, y1), drag to (x2, y2)	Create line art
Finished in Sprint 1	Artist Change pen color – use a color selection dialog	Create colorful drawings
Finished in Sprint 1	Artist Change pen width – use a text input dialog and atoi	Provide more texture and movement
	Artist Select pen properties from a toolbar	Switch pen properties more quickly
	Artist Save – write the lines vector to a default filename	Reload, reuse, and share drawings
	Artist New canvas (vector) – toss drawing (prompt to save) and start new	Start a new drawing without closing the program
	Artist Open – load the lines vector for the default file	Reload, reuse, and share drawings
	Artist Save as – write the lines vector to a user-specified file	Manage my drawing files with less work
	Artist Open – load the lines vector from a user-selected file	Reload, reuse, and share drawings
	Artist Draw contiguous straight lines – sequence of clicks	Create polygons
	Artist Draw freehand – hold down the mouse and scribble	Create complex shapes
	Artist Draw dashed lines in all modes	Make my drawings more interesting and attractive
	Artist Undo	Make mistakes without having to start over
	Artist Run Turtle Graphics file	Create reusable complex shapes
	Artist Copy Image	Place my drawings into e.g., Office documents
	Artist Paste Image	Insert drawings from e.g., Gnome Screenshot
	Artist Export to PNG format	Save my drawings in an industry standard format
Lawyer	Acknowledge license obligations	Avoid litigation

Feature ID	Description	Status	Notes
TB	Enable the toolbar		Inherited from baseline
TB	Add pen color button and connect signal		
TB	Add pen width button and connect signal		

Feature 6

Step 007 – Add Toolbar

- Just as we did with the Nim example
- We need icons: Pen color  and width 
though I considered this: 

Feature 6

Step 007 – Add Toolbar

```
// ///////////
// TOOLBAR -- Add a toolbar to the vertical box below the menu
//
Gtk::Toolbar *toolbar = Gtk::manage(new Gtk::Toolbar);
vbox->add(*toolbar);
```

mainwin.cpp

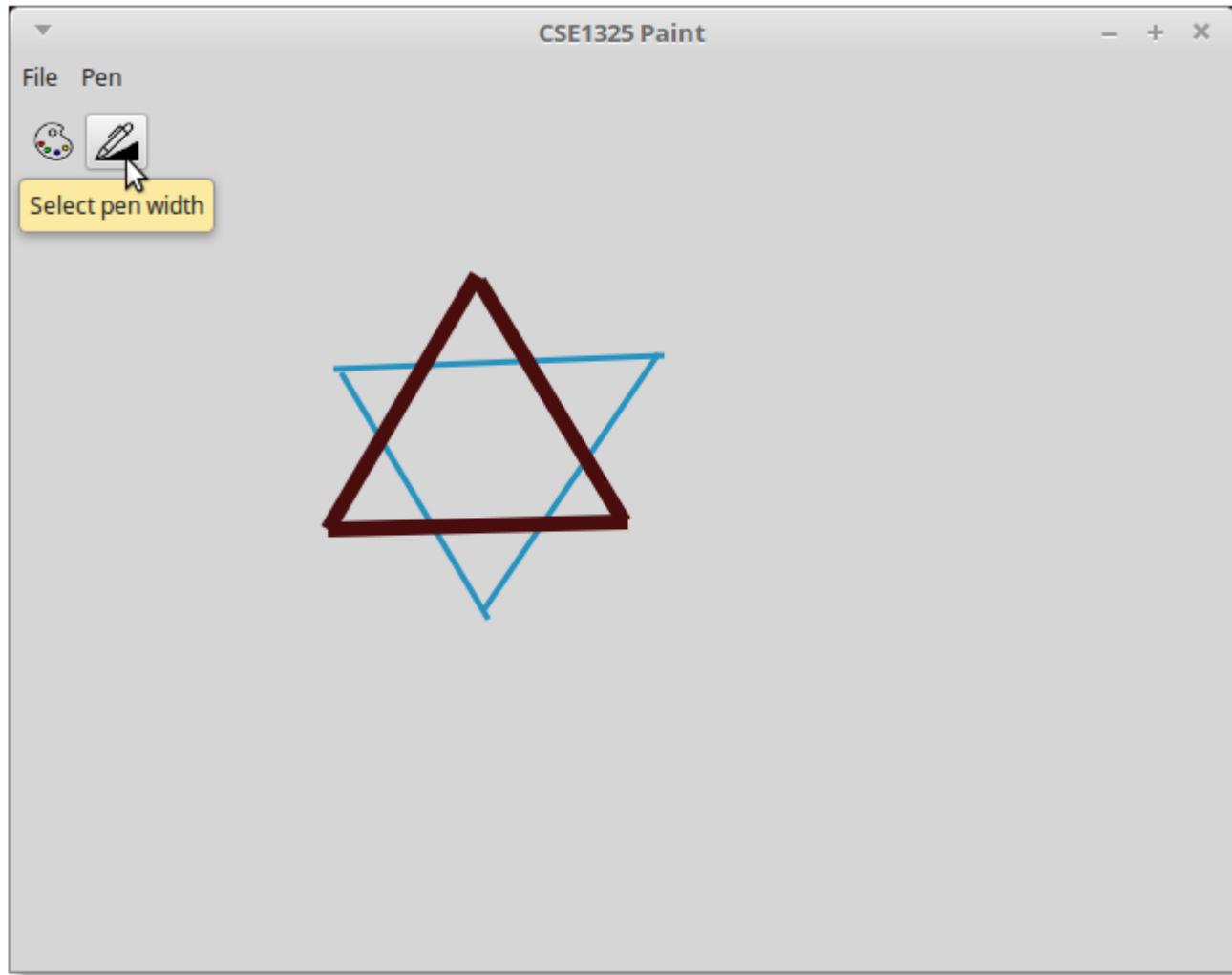
```
//      P E N      C O L O R
// Add a new pen color icon
Gtk::Image *button_pen_color_image =
    Gtk::manage(new Gtk::Image("select_pen_color.png"));
Gtk::ToolButton *button_pen_color =
    Gtk::manage(new Gtk::ToolButton(*button_pen_color_image));
button_pen_color->set_tooltip_markup("Select pen color");
button_pen_color->signal_clicked().connect(
    sigc::mem_fun(*this, &Mainwin::on_color_click));
toolbar->append(*button_pen_color);
```

Right below the menu code.

```
//      P E N      W I D T H
// Add a set pen width icon
Gtk::Image *button_pen_width_image =
    Gtk::manage(new Gtk::Image("select_pen_width.png"));
Gtk::ToolButton *button_pen_width =
    Gtk::manage(new Gtk::ToolButton(*button_pen_width_image));
button_pen_width->set_tooltip_markup("Select pen width");
button_pen_width->signal_clicked().connect(
    sigc::mem_fun(*this, &Mainwin::on_width_click));
toolbar->append(*button_pen_width);
```

We use exactly the same handlers for buttons
as for menu selections.

Demonstrating Feature 6



And I forgot to commit this version to git... *sigh*

SCRUM Product Backlog and Next Sprint Backlog Tasks

Status	As an... I want to...	So that I can...
Finished in Sprint 1	Artist Exit via x – check for "dirty" bit and prompt to save given new data	Close the window
Finished in Sprint 1	Artist Exit via File > Exit – creating a File > Exit menu item	Close the window independent of the OS
Finished in Sprint 1	Artist Draw discrete lines – click for (x1, y1), drag to (x2, y2)	Create line art
Finished in Sprint 1	Artist Change pen color – use a color selection dialog	Create colorful drawings
Finished in Sprint 1	Artist Change pen width – use a text input dialog and atoi	Provide more texture and movement
Finished in Sprint 1	Artist Select pen properties from a toolbar	Switch pen properties more quickly
	Artist Save – write the lines vector to a default filename	Reload, reuse, and share drawings
	Artist New canvas (vector) – toss drawing (prompt to save) and start new	Start a new drawing without closing the program
	Artist Open – load the lines vector for the default file	Reload, reuse, and share drawings
	Artist Save as – write the lines vector to a user-specified file	Manage my drawing files with less work
	Artist Open – load the lines vector from a user-selected file	Reload, reuse, and share drawings
	Artist Draw contiguous straight lines – sequence of clicks	Create polygons
	Artist Draw freehand – hold down the mouse and scribble	Create complex shapes
	Artist Draw dashed lines in all modes	Make my drawings more interesting and attractive
	Artist Undo	Make mistakes without having to start over
	Artist Run Turtle Graphics file	Create reusable complex shapes
	Artist Copy Image	Place my drawings into e.g., Office documents
	Artist Paste Image	Insert drawings from e.g., Gnome Screenshot
	Artist Export to PNG format	Save my drawings in an industry standard format
Lawyer	Acknowledge license obligations	Avoid litigation
And now for something completely new...		

Feature ID	Description	Status
SAV	Add Line::save to stream	
SAV	Add View::save to stream	
SAV	Add File > Save to menu	
SAV	Add save button to tool bar	

Reading and Writing Files

- Here's the 20,000 foot (not on the exam) view of file I/O
- For our purposes, reading and writing files is *exactly like reading and writing cin and cout*
 - Streams are streams
- We create a new output stream (aka cout) by instancing **ofstream**, with the filename as parameter
 - ostream is its base class, and is used for referencing an ofstream
- We create a new input stream (aka cin) by instancing **ifstream**, with the filename as parameter
 - istream is its base class, and is used for referencing an ifstream
- I/O is complicated, but for now, that's all you need to know – we'll just pretend everything will work error free



We Need a Data Format

- Each Line contains 6 objects
 - Given the 6 objects, we can recreate each line and push it back on the lines vector
- The question mark is `_color` – how to convert `Gdk::RGBA` objects to and from a string?
 - `RGBA` does NOT implement operator<< or operator>>
 - But it DOES offer `to_string` (e.g., `_color.to_string`)
 - This results in the string “`rgb(255,0,0)`” for red
 - And it DOES include a reverse constructor, e.g., `Gdk::RBGA{“rgb(255,0,0)”}`

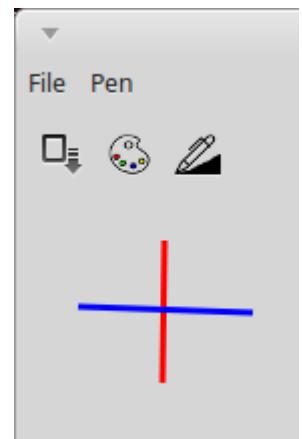
```
class Line {  
    // ...  
private:  
    double _x1, _y1, _x2, _y2;  
    Gdk::RGBA _color;  
    double _width;  
};
```

We Need a Data Format

- `cin` specializes in reading whitespace-separated values from a stream
 - No parsing required
 - Highly reliable *for known data*
- Simplest is often best – just stream the 6 space-separated strings out to a file
 - We can then just stream them back as batches of 6, creating each `Line` object on the vector, until we reach the end of the file

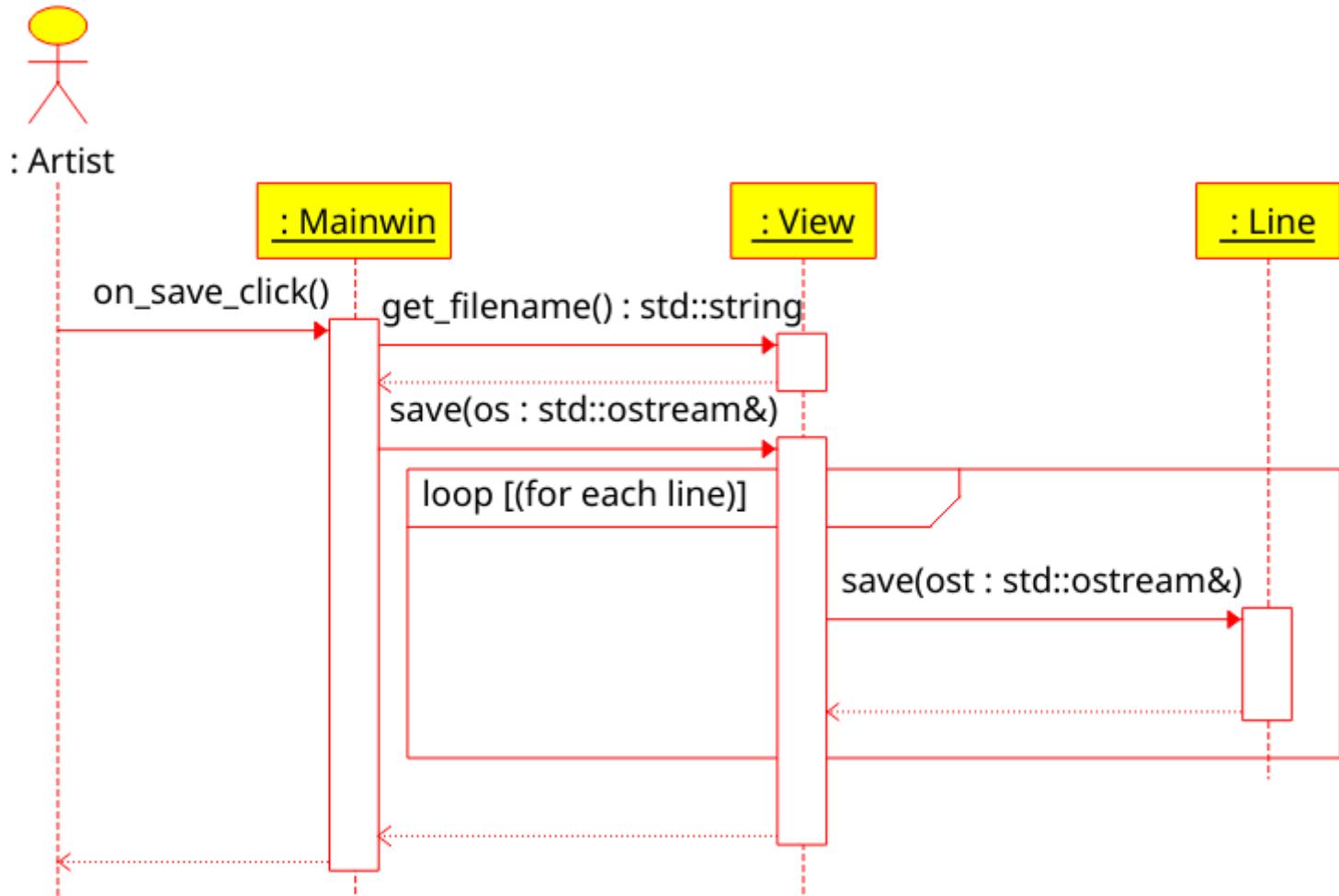
```
class Line {  
    // ...  
private:  
    double _x1, _y1, _x2, _y2;  
    Gdk::RGBA _color;  
    double _width;  
};
```

(X1,Y1) (X2,Y2) W Color (X1,Y1) (X2,Y2) W Color →
74 20 73 91 3 rgb(255,0,0) 31 53 118 56 3 rgb(0,0,255)



Feature 6 – Saving to a File

Sequence Diagram



We'll implement this design from right to left

Feature 7

Step 008 – Save

```
#ifndef _LINE_H
#define _LINE_H
#include <gdkmm/rgba.h>
#include <iostream>

class Line { // Define a line from (x1, y1) to (x2, y2)
public:
    Line(double p_x1, double p_y1, double p_x2, double p_y2,
          Gdk::RGBA p_color, double p_width);
    double x1();
    double y1();
    double x2();
    double y2();
    Gdk::RGBA color();
    double width();
    void save(std::ostream& ost);
private:
    double _x1, _y1, _x2, _y2;
    Gdk::RGBA _color;
    double _width;
};

#endif
```

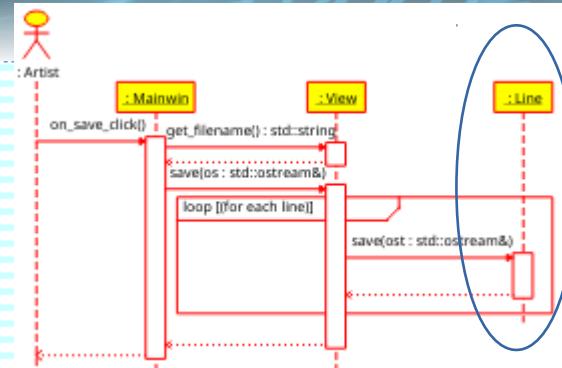
line.h

```
void Line::save(std::ostream& ost) {
    ost << _x1 << ' ' << _y1 << ' '
        << _x2 << ' ' << _y2 << ' '
        << _width << ' '
        << _color.to_string() << ' ';
}
```

line.cpp

We add a “save” method that sends the line as space-separated strings to a stream.
We’ll reconstruct the line later using a constructor:

```
Line::Line(std::istream& ist) {
    std::string rgba_color;
    ist >> _x1 >> _y1 >> _x2 >> _y2
        >> _width >> _dash >> rgba_color;
    _color = Gdk::RGBA{rgba_color};
}
```



Feature 7

Step 008 – Save

```
private:
```

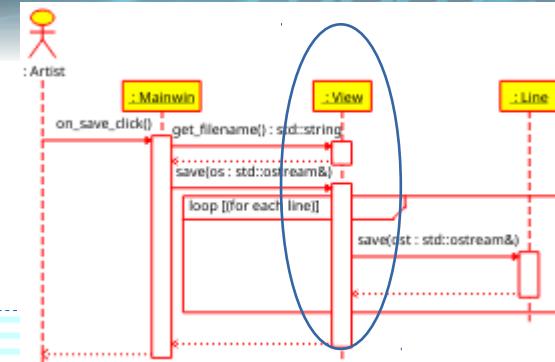
view.h

```
// Filename of the current drawing  
string filename = "untitled.ctp";
```

```
void View::save(ostream& os) {  
    for(Line l : lines)  
        l.save(os);  
    dirty = false;  
}
```

We also add a “save” method to View that calls the save method of every line in our vector.
We’ll reconstruct the vector later using this:

```
// Add a line to the vector  
void View::new_line(std::istream& is) {  
    lines.push_back(Line(is));  
}  
  
void View::load(std::istream& is) {  
    dirty = (lines.size() > 0);  
    while(is) new_line(is);  
}
```



view.cpp

Feature 7

Step 008 – Save

```
//      S A V E
// Append Save to the File menu
Gtk::MenuItem *menuitem_save =
    Gtk::manage(new Gtk::MenuItem("_Save", true));
menuitem_save->signal_activate().connect(
    sigc::mem_fun(*this, &Mainwin::on_save_click));
filemenu->append(*menuitem_save);
```

mainwin.cpp

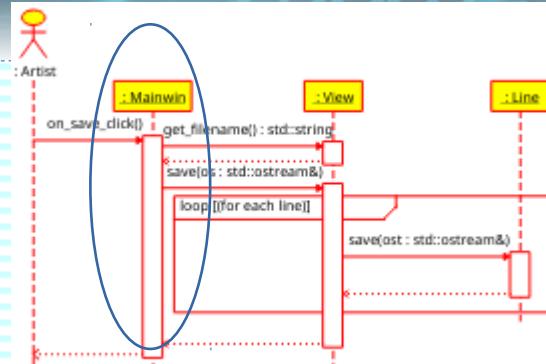
```
//      S A V E   D R A W I N G
// Save a drawing to the default filename
Gtk::ToolButton *save_button =
    Gtk::manage(new Gtk::ToolButton(Gtk::Stock::SAVE));
save_button->set_tooltip_markup("Save drawing to the default filename");
save_button->signal_clicked().connect(
    sigc::mem_fun(*this, &Mainwin::on_save_click));
toolbar->append(*save_button);
```

```
// The user wants to save the current drawing to the default filename
void Mainwin::on_save_click() {
    ofstream ofs{view->get_filename(), std::ofstream::out};
    view->save(ofs);
}
```

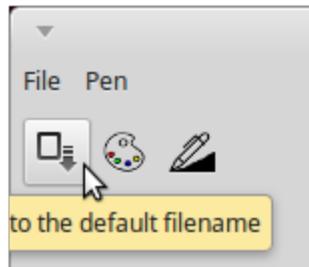
In C++, we open a file for writing by instancing a new output file stream (class ofstream).

- Parameter #1 is the filename.
- Parameter #2 is the mode, in this case, output.

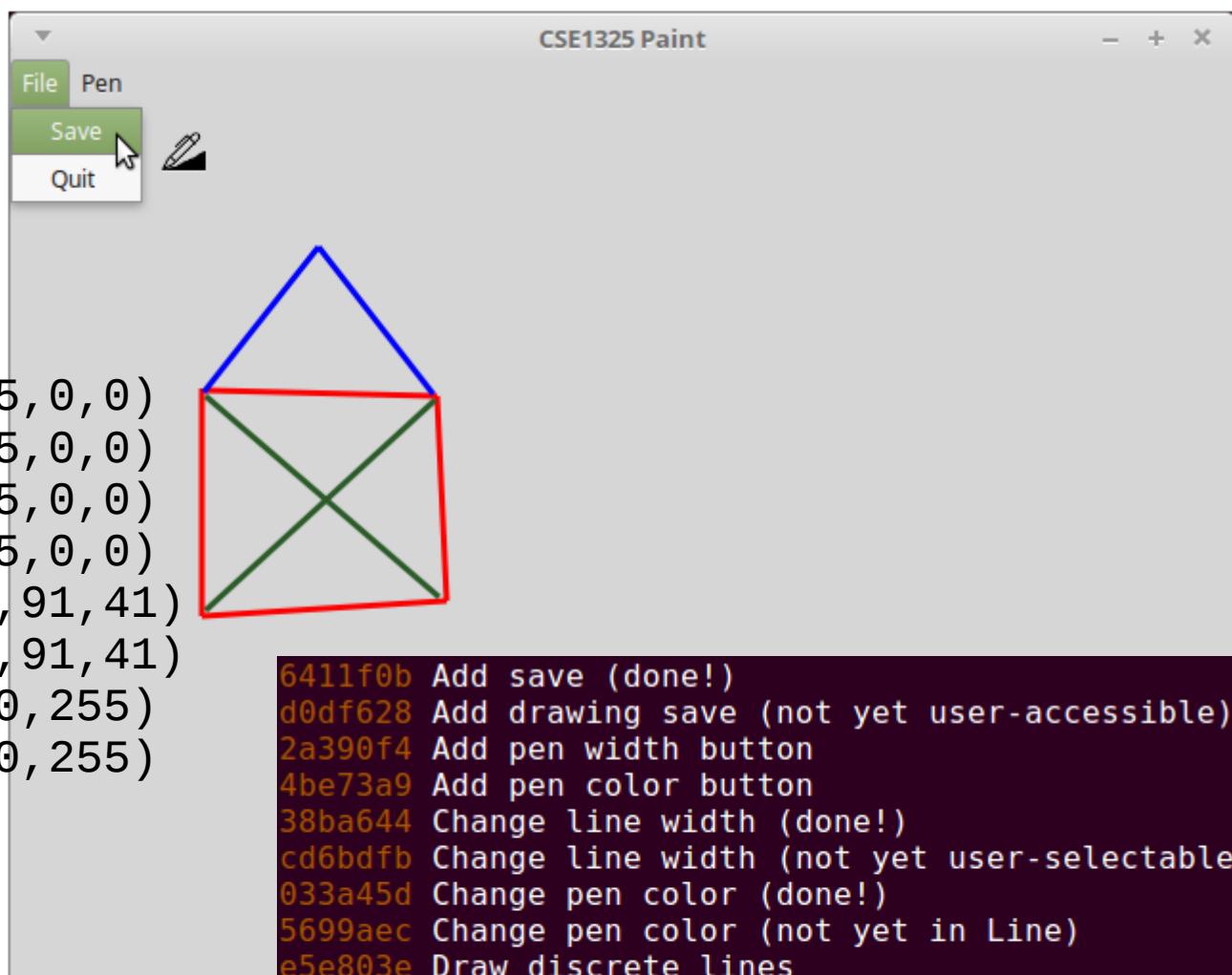
Then we use the object just as if it were cout.



Feature 6 – Saving a File



```
100 101 100 220 3 rgb(255,0,0)
100 220 228 212 3 rgb(255,0,0)
228 212 223 105 3 rgb(255,0,0)
223 105 102 102 3 rgb(255,0,0)
102 105 224 210 3 rgb(43,91,41)
222 107 102 217 3 rgb(43,91,41)
101 103 161 27 3 rgb(0,0,255)
161 27 221 104 3 rgb(0,0,255)
```



```
6411f0b Add save (done!)
d0df628 Add drawing save (not yet user-accessible)
2a390f4 Add pen width button
4be73a9 Add pen color button
38ba644 Change line width (done!)
cd6bd9b Change line width (not yet user-selectable)
033a45d Change pen color (done!)
5699aec Change pen color (not yet in Line)
e5e803e Draw discrete lines
e95ba71 Exit via File > Exit
2e10779 Exit via x, checking dirty bit
afffb70 Refactor, rm turtle.*
ee35027 Baseline from turtle graphics demo
```

(Note, newlines added for clarity.)

The actual data file is one long string of characters - though I added newlines in cleanup because it makes it much more legible!)

SCRUM Product Backlog and Next Sprint Backlog Tasks

Status	As an... I want to...	So that I can...
Finished in Sprint 1	Artist Exit via x – check for "dirty" bit and prompt to save given new data	Close the window
Finished in Sprint 1	Artist Exit via File > Exit – creating a File > Exit menu item	Close the window independent of the OS
Finished in Sprint 1	Artist Draw discrete lines – click for (x1, y1), drag to (x2, y2)	Create line art
Finished in Sprint 1	Artist Change pen color – use a color selection dialog	Create colorful drawings
Finished in Sprint 1	Artist Change pen width – use a text input dialog and <u>atoi</u>	Provide more texture and movement
Finished in Sprint 1	Artist Select pen properties from a <u>toolbar</u>	Switch pen properties more quickly
Finished in Sprint 1	Artist Save – write the lines vector to a default filename	Reload, reuse, and share drawings
	Artist New canvas (vector) – toss drawing (prompt to save) and start new	Start a new drawing without closing the program
	Artist Open – load the lines vector for the default file	Reload, reuse, and share drawings
	Artist Save as – write the lines vector to a user-specified file	Manage my drawing files with less work
	Artist Open – load the lines vector from a user-selected file	Reload, reuse, and share drawings
	Artist Draw contiguous straight lines – sequence of clicks	Create polygons
	Artist Draw freehand – hold down the mouse and scribble	Create complex shapes
	Artist Draw dashed lines in all modes	Make my drawings more interesting and attractive
	Artist Undo	Make mistakes without having to start over
	Artist Run Turtle Graphics file	Create reusable complex shapes
	Artist Copy Image	Place my drawings into e.g., Office documents
	Artist Paste Image	Insert drawings from e.g., Gnome <u>Screenshot</u>
	Artist Export to PNG format	Save my drawings in an industry standard format
Lawyer	Acknowledge license obligations	Avoid litigation

Feature ID	Description	Status
NEW	Add View::new	
NEW	Add File > New to menu	
NEW	Add new button to tool bar	

Feature 8

Step 009 – New

```
bool View::on_my_delete_event(GdkEventAny* event) {
    return drawing_is_unsaved();
}

bool View::drawing_is_unsaved() {
    if (saved()) return false;
    Gtk::MessageDialog d{win, "Unsaved data will be lost", false,
                         Gtk::MESSAGE_WARNING, Gtk::BUTTONS_NONE};
    d.add_button("Save", 1);
    d.add_button("Discard", 2);
    d.add_button("Cancel", 3);
    int response = d.run();
    if (response == 1) {
        ofstream ofs{filename, std::ofstream::out};
        save(ofs);
        return false;
    } else if (response == 2) {
        return false;
    } else {
        return true;
    }
}

void View::new_drawing() {
    if (drawing_is_unsaved()) return;
    lines.clear();
    dirty = false;
}
```

view.cpp

“New” is another risk of losing data.
Unlike File > Quit, no delete event is generated.

Therefore, we refactor on_my_delete_event to
create a private drawing_is_unsaved method for
use by new_drawing and on_my_delete_event.

Feature 8

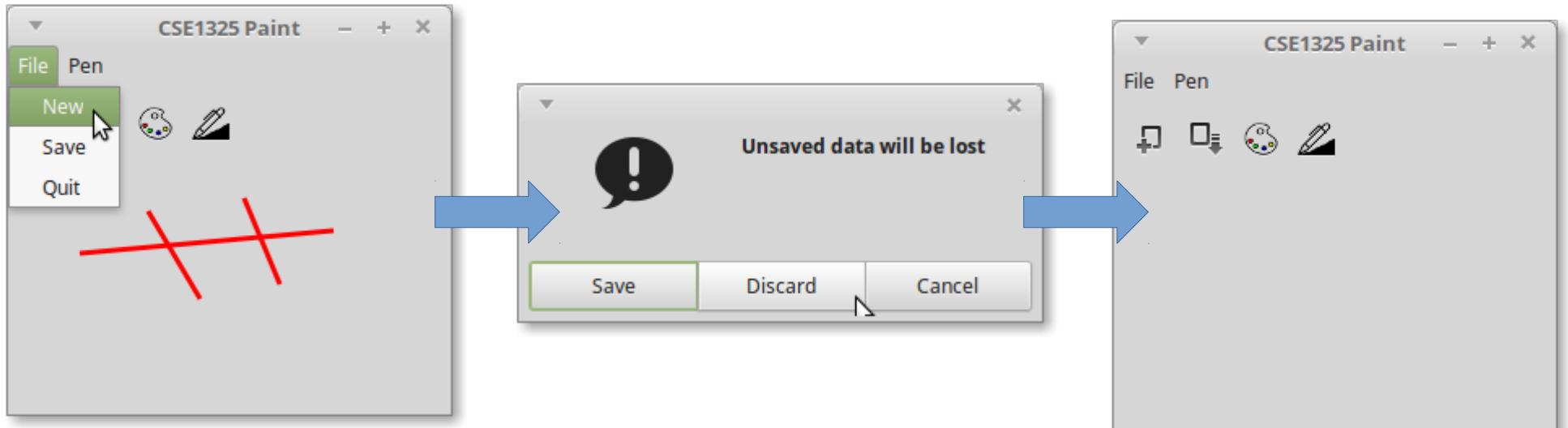
Step 009 – New

```
//      N E W                                     mainwin.cpp
// Append New to the File menu
Gtk::MenuItem *menuitem_new = Gtk::manage(new Gtk::MenuItem{"_New", true});
menuitem_new->signal_activate().connect(sigc::mem_fun(*this,
    &Mainwin::on_new_click));
filemenu->append(*menuitem_new);

//      N E W      D R A W I N G
// Start a new drawing
Gtk::ToolButton *new_button = Gtk::manage(
    new Gtk::ToolButton(Gtk::Stock::NEW));
new_button->set_tooltip_markup("Start a new drawing");
new_button->signal_clicked().connect(
    sigc::mem_fun(*this, &Mainwin::on_new_click));
toolbar->append(*new_button);

//
// The user wants a new, empty drawing
//
void Mainwin::on_new_click() {
    view->new_drawing();
}
```

Feature 8 – New



```
02c64d0 Add new (not yet user-accessible)
6411f0b Add save (done!)
d0df628 Add drawing save (not yet user-accessible)
2a390f4 Add pen width button
4be73a9 Add pen color button
38ba644 Change line width (done!)
cd6bdbfb Change line width (not yet user-selectable)
033a45d Change pen color (done!)
5699aec Change pen color (not yet in Line)
e5e803e Draw discrete lines
e95ba71 Exit via File > Exit
2e10779 Exit via x, checking dirty bit
afffb70 Refactor, rm turtle.*
ee35027 Baseline from turtle graphics demo
```

SCRUM Product Backlog and Next Sprint Backlog Tasks

Status	As an... I want to...	So that I can...
Finished in Sprint 1	Artist Exit via x – check for “dirty” bit and prompt to save given new data	Close the window
Finished in Sprint 1	Artist Exit via File > Exit – creating a File > Exit menu item	Close the window independent of the OS
Finished in Sprint 1	Artist Draw discrete lines – click for (x1, y1), drag to (x2, y2)	Create line art
Finished in Sprint 1	Artist Change pen color – use a color selection dialog	Create colorful drawings
Finished in Sprint 1	Artist Change pen width – use a text input dialog and atoi	Provide more texture and movement
Finished in Sprint 1	Artist Select pen properties from a toolbar	Switch pen properties more quickly
Finished in Sprint 1	Artist Save – write the lines vector to a default filename	Reload, reuse, and share drawings
Finished in Sprint 1	Artist New canvas (vector) – toss drawing (prompt to save) and start new	Start a new drawing without closing the program
	Artist Open – load the lines vector for the default file	Reload, reuse, and share drawings
	Artist Save as – write the lines vector to a user-specified file	Manage my drawing files with less work
	Artist Open – load the lines vector from a user-selected file	Reload, reuse, and share drawings
	Artist Draw contiguous straight lines – sequence of clicks	Create polygons
	Artist Draw freehand – hold down the mouse and scribble	Create complex shapes
	Artist Draw dashed lines in all modes	Make my drawings more interesting and attractive
	Artist Undo	Make mistakes without having to start over
	Artist Run Turtle Graphics file	Create reusable complex shapes
	Artist Copy Image	Place my drawings into e.g., Office documents
	Artist Paste Image	Insert drawings from e.g., Gnome Screenshot
	Artist Export to PNG format	Save my drawings in an industry standard format
Lawyer	Acknowledge license obligations	Avoid litigation

Feature ID	Description	Status
OP1	Add Line::Line(inputStream)	
OP1	Add View::Open(filename)	
OP1	Add File > Open, passing default filename	
OP1	Add open button to tool bar	

Feature 9

Step 010 – Open Default

```
Line::Line(std::istream& ist) {  
    std::string rgba_color;  
    ist >> _x1 >> _y1 >> _x2 >> _y2 >> _width >> rgba_color;  
    _color = Gdk::RGBA{rgba_color};  
}
```

line.cpp

```
void View::load(std::istream& is) {  
    dirty = (lines.size() > 0);  
    while(is) new_line(is);  
}
```

view.cpp

As promised...

Feature 9

Step 010 – Open Default

```
//      O P E N
// Append Open to the File menu
Gtk::MenuItem *menuitem_open =
    Gtk::manage(new Gtk::MenuItem("_Open", true));
menuitem_open->signal_activate().connect(
    sigc::mem_fun(*this, &Mainwin::on_open_click));
filemenu->append(*menuitem_open);
```

mainwin.cpp

```
//      O P E N   D R A W I N G
// Open a drawing from disk
Gtk::ToolButton *open_button =
    Gtk::manage(new Gtk::ToolButton(Gtk::Stock::OPEN));
open_button->set_tooltip_markup("Open a drawing");
open_button->signal_clicked().connect(
    sigc::mem_fun(*this, &Mainwin::on_open_click));
toolbar->append(*open_button);
```

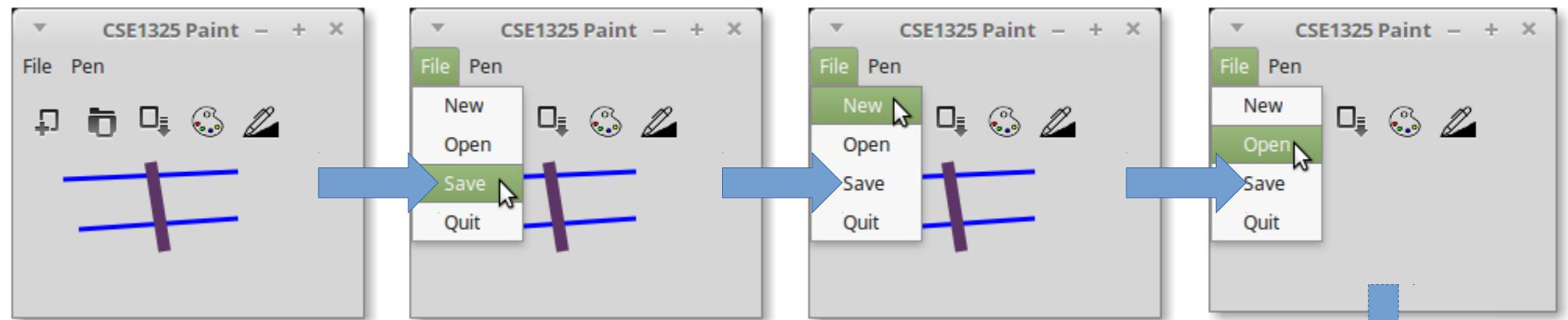
```
//
// The user wants to open a drawing from the filesystem
//
```

```
void Mainwin::on_open_click() {
    std::ifstream ifs{view->get_filename(), std::ifstream::in};
    view->load(ifs);
    view->queue_draw(); // refresh the DrawingArea with loaded drawing
}
```

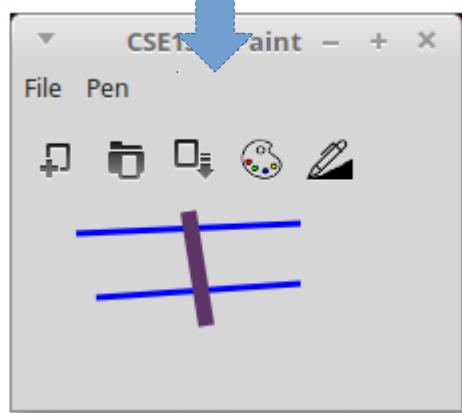
As with output streams,
in C++ instance ifstream
with filename and mode.
The resulting object
works very much like cin.

Don't forget to tell the system to redraw the window!

Feature 9 – Open Default



```
8e0d17c Add load from default (done!)\nc74196c Add load (not yet user-accessible)\n02c64d0 Add new (not yet user-accessible)\n6411f0b Add save (done!)\nd0df628 Add drawing save (not yet user-accessible)\n2a390f4 Add pen width button\n4be73a9 Add pen color button\n38ba644 Change line width (done!)\ncd6bd9b Change line width (not yet user-selectable)\n033a45d Change pen color (done!)\n5699aec Change pen color (not yet in Line)\ne5e803e Draw discrete lines\ne95ba71 Exit via File > Exit\n2e10779 Exit via x, checking dirty bit\nafffb70 Refactor, rm turtle.*\nee35027 Baseline from turtle graphics demo
```



SCRUM Product Backlog and Next Sprint Backlog Tasks

Status	As an...	I want to...	So that I can...
Finished in Sprint 1	Artist	Exit via x – check for "dirty" bit and prompt to save given new data	Close the window
Finished in Sprint 1	Artist	Exit via File > Exit – creating a File > Exit menu item	Close the window independent of the OS
Finished in Sprint 1	Artist	Draw discrete lines – click for (x1, y1), drag to (x2, y2)	Create line art
Finished in Sprint 1	Artist	Change pen color – use a color selection dialog	Create colorful drawings
Finished in Sprint 1	Artist	Change pen width – use a text input dialog and <code>atoi</code>	Provide more texture and movement
Finished in Sprint 1	Artist	Select pen properties from a <u>toolbar</u>	Switch pen properties more quickly
Finished in Sprint 1	Artist	Save – write the lines vector to a default filename	Reload, reuse, and share drawings
Finished in Sprint 1	Artist	New canvas (vector) – toss drawing (prompt to save) and start new	Start a new drawing without closing the program
Finished in Sprint 1	Artist	Open – load the lines vector for the default file	Reload, reuse, and share drawings
	Artist	Save as – write the lines vector to a user-specified file	Manage my drawing files with less work
	Artist	Open – load the lines vector from a user-selected file	Reload, reuse, and share drawings
	Artist	Draw contiguous straight lines – sequence of clicks	Create polygons
	Artist	Draw freehand – hold down the mouse and scribble	Create complex shapes
	Artist	Draw dashed lines in all modes	Make my drawings more interesting and attractive
	Artist	Undo	Make mistakes without having to start over
	Artist	Run Turtle Graphics file	Create reusable complex shapes
	Artist	Copy Image	Place my drawings into e.g., Office documents
	Artist	Paste Image	Insert drawings from e.g., Gnome Screenshot
	Artist	Export to PNG format	Save my drawings in an industry standard format
	Lawyer	Acknowledge license obligations	Avoid litigation

Feature ID	Description	Status
SVA	Add Mainwin::on_save_as_click with file chooser dialog	
SVA	Add File > Save As to menu	
SVA	Save selected filename to view	

Feature 10

Step 011 – Save As

```
//      S A V E   A S
// Append Save As to the File menu
Gtk::MenuItem *menuitem_save_as =
    Gtk::manage(new Gtk::MenuItem{"Save _As", true});
menuitem_save_as->signal_activate().connect(
    sigc::mem_fun(*this, &Mainwin::on_save_as_click));
filemenu->append(*menuitem_save_as);

//
// The user wants to save the current drawing to a new filename
//
void Mainwin::on_save_as_click() {
    Gtk::FileChooserDialog dialog("Please choose a file",
        Gtk::FileChooserAction::FILE_CHOOSER_ACTION_SAVE);
dialog.set_transient_for(*this);

//Add response buttons the the dialog:
dialog.add_button("_Cancel", 0);
dialog.add_button("_Save", 1);

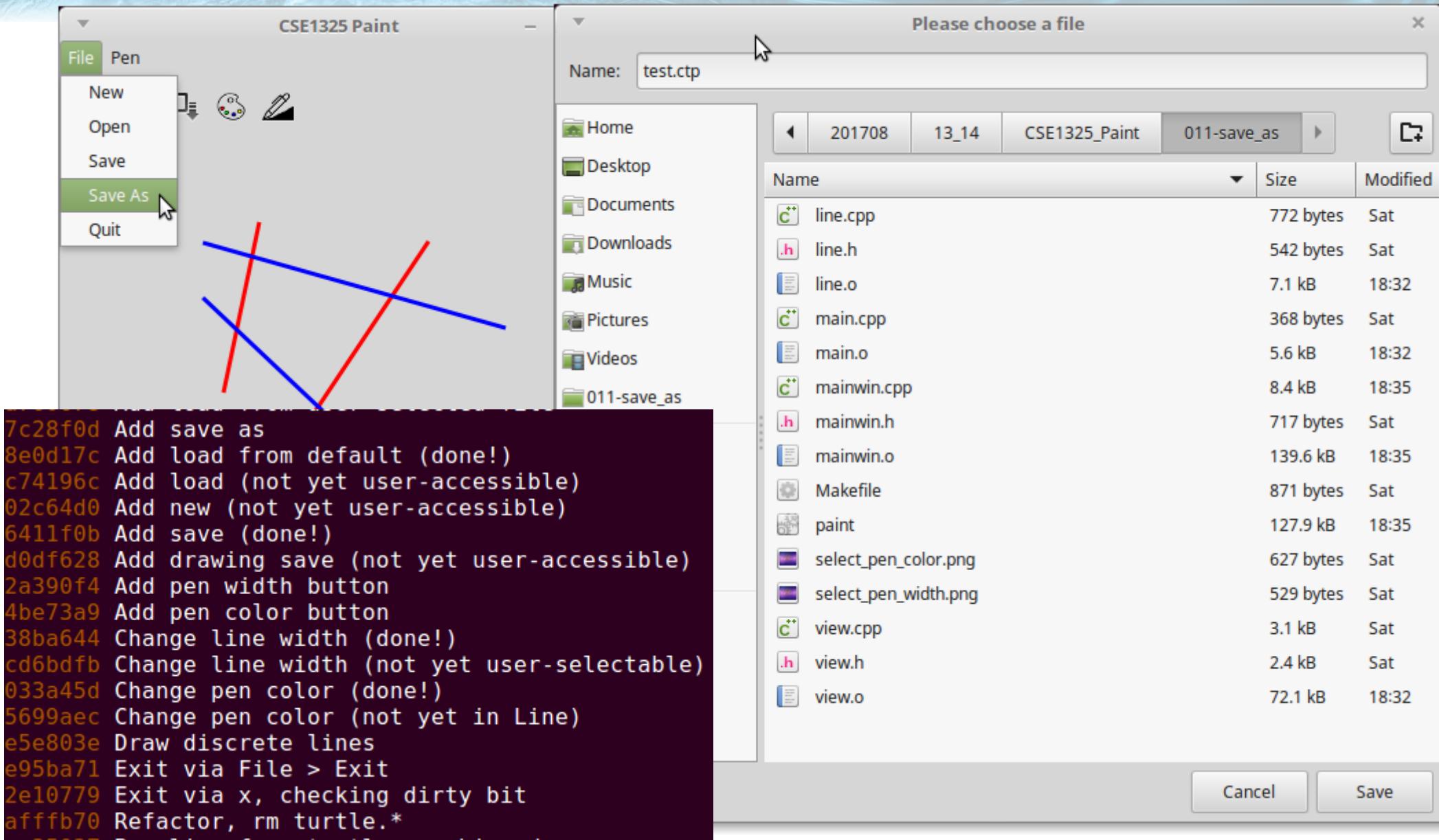
int result = dialog.run();

//Handle the response:
if (result == 1) {
    std::ofstream ofs{dialog.get_filename(), std::ofstream::out};
    view->save(ofs);
}
}
```

mainwin.cpp

gtkmm provides a pre-defined file chooser dialog to select a new filename for saving.

Feature 10 – Save As



SCRUM Product Backlog and Next Sprint Backlog Tasks

Status	As an... I want to...	So that I can...
Finished in Sprint 1	Artist Exit via x – check for "dirty" bit and prompt to save given new data	Close the window
Finished in Sprint 1	Artist Exit via File > Exit – creating a File > Exit menu item	Close the window independent of the OS
Finished in Sprint 1	Artist Draw discrete lines – click for (x1, y1), drag to (x2, y2)	Create line art
Finished in Sprint 1	Artist Change pen color – use a color selection dialog	Create colorful drawings
Finished in Sprint 1	Artist Change pen width – use a text input dialog and <u>atoi</u>	Provide more texture and movement
Finished in Sprint 1	Artist Select pen properties from a <u>toolbar</u>	Switch pen properties more quickly
Finished in Sprint 1	Artist Save – write the lines vector to a default filename	Reload, reuse, and share drawings
Finished in Sprint 1	Artist New canvas (vector) – toss drawing (prompt to save) and start new	Start a new drawing without closing the program
Finished in Sprint 1	Artist Open – load the lines vector for the default file	Reload, reuse, and share drawings
Finished in Sprint 1	Artist Save as – write the lines vector to a user-specified file	Manage my drawing files with less work
	Artist Open – load the lines vector from a user-selected file	Reload, reuse, and share drawings
	Artist Draw contiguous straight lines – sequence of clicks	Create polygons
	Artist Draw freehand – hold down the mouse and scribble	Create complex shapes
	Artist Draw dashed lines in all modes	Make my drawings more interesting and attractive
	Artist Undo	Make mistakes without having to start over
	Artist Run Turtle Graphics file	Create reusable complex shapes
	Artist Copy Image	Place my drawings into e.g., Office documents
	Artist Paste Image	Insert drawings from e.g., Gnome <u>Screenshot</u>
	Artist Export to PNG format	Save my drawings in an industry standard format
Lawyer	Acknowledge license obligations	Avoid litigation

Feature ID	Description	Status
<u>OPN</u>	Add file chooser dialog to open	

Feature 11

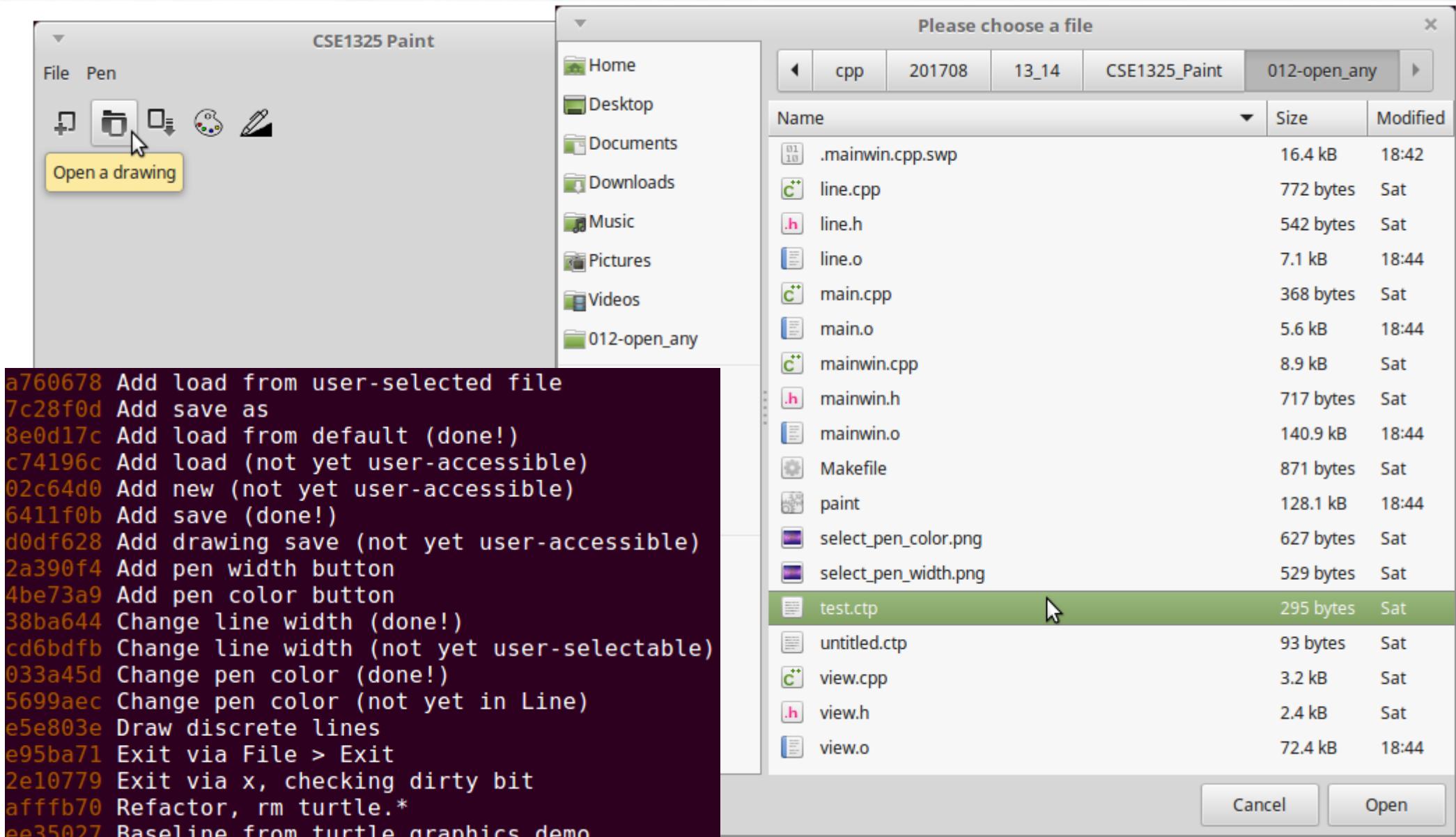
Step 012 – Open

```
//  
// The user wants to open a drawing from the filesystem  
  
void Mainwin::on_open_click() {  
    Gtk::FileChooserDialog dialog("Please choose a file",  
        Gtk::FileChooserAction::FILE_CHOOSER_ACTION_OPEN);  
    dialog.set_transient_for(*this);  
  
    //Add response buttons to the dialog:  
    dialog.add_button("_Cancel", 0);  
    dialog.add_button("_Open", 1);  
  
    int result = dialog.run();  
  
    //Handle the response:  
    if (result == 1) {  
        std::ifstream ifs{dialog.get_filename(), std::ifstream::in};  
        view->load(ifs);  
        view->queue_draw(); // refresh the DrawingArea with loaded drawing  
        view->set_filename(dialog.get_filename());  
    }  
}
```

mainwin.cpp

The same file chooser dialog may be used to select an existing filename for opening.

Feature 12 – Open



SCRUM Product Backlog and Next Sprint Backlog Tasks

Status	As an... I want to...	So that I can...
Finished in Sprint 1	Artist Exit via x – check for "dirty" bit and prompt to save given new data	Close the window
Finished in Sprint 1	Artist Exit via File > Exit – creating a File > Exit menu item	Close the window independent of the OS
Finished in Sprint 1	Artist Draw discrete lines – click for (x1, y1), drag to (x2, y2)	Create line art
Finished in Sprint 1	Artist Change pen color – use a color selection dialog	Create colorful drawings
Finished in Sprint 1	Artist Change pen width – use a text input dialog and atoi	Provide more texture and movement
Finished in Sprint 1	Artist Select pen properties from a toolbar	Switch pen properties more quickly
Finished in Sprint 1	Artist Save – write the lines vector to a default filename	Reload, reuse, and share drawings
Finished in Sprint 1	Artist New canvas (vector) – toss drawing (prompt to save) and start new	Start a new drawing without closing the program
Finished in Sprint 1	Artist Open – load the lines vector for the default file	Reload, reuse, and share drawings
Finished in Sprint 1	Artist Save as – write the lines vector to a user-specified file	Manage my drawing files with less work
	Artist Open – load the lines vector from a user-selected file	Reload, reuse, and share drawings
	Artist Draw contiguous straight lines – sequence of clicks	Create polygons
	Artist Draw freehand – hold down the mouse and scribble	Create complex shapes
	Artist Draw dashed lines in all modes	Make my drawings more interesting and attractive
	Artist Undo	Make mistakes without having to start over
	Artist Run Turtle Graphics file	Create reusable complex shapes
	Artist Copy Image	Place my drawings into e.g., Office documents
	Artist Paste Image	Insert drawings from e.g., Gnome Screenshot
	Artist Export to PNG format	Save my drawings in an industry standard format
Lawyer	Acknowledge license obligations	Avoid litigation

Feature ID	Description	Status
DCL	Add View::set_draw_mode with constants	
DCL	Add contiguous line drawing to mainwin	
DCL	Add Pen > Contiguous and Pen > Segment to menu	
DCL	Add contiguous and segment to tool bar	

Feature 12

Step 013 – Contiguous Mode

- Adding contiguous mode (a new line every click from the previous point of click) means we also need a way to select segment mode (two clicks per line segment)
- We definitely want tool bar buttons for these!
- Then we need to adjust our click handler to manage contiguous line creation

Feature 12

Step 013 – Contiguous Mode

```
//      P E N   M O D E : S E G M E N T           mainwin.cpp
// Append Segment to the Pen menu
Gtk::MenuItem *menuitem_pen_mode_segment =
    Gtk::manage(new Gtk::MenuItem{"Line _Segments", true});
menuitem_pen_mode_segment->signal_activate().connect(
    sigc::mem_fun(*this, &Mainwin::on_pen_mode_segment_click));
penmenu->append(*menuitem_pen_mode_segment);

//      P E N   M O D E : C O N T I G U O U S
// Append Contiguous to the Pen menu
Gtk::MenuItem *menuitem_pen_mode_contiguous =
    Gtk::manage(new Gtk::MenuItem{"_Contiguous Lines", true});
menuitem_pen_mode_contiguous->signal_activate().connect(
    sigc::mem_fun(*this,&Mainwin::on_pen_mode_contiguous_click));
penmenu->append(*menuitem_pen_mode_contiguous);

// Set for segment mode initially
on_pen_mode_segment_click();
}

void Mainwin::on_pen_mode_segment_click() {
    view->set_pen_mode(View::PEN_MODE_SEGMENT);
}

void Mainwin::on_pen_mode_contiguous_click() {
    view->set_pen_mode(View::PEN_MODE_CONTIGUOUS);
}
```

Feature 12

Step 013 – Contiguous Mode

```
// P E N   M O D E : S E G M E N T                                mainwin.cpp
// Add a toggle button to enable segment pen mode
Gtk::Image *segment_image = Gtk::manage(new Gtk::Image("segment.png"));
button_segment = Gtk::manage(new Gtk::RadioToolButton(*segment_image));
button_segment->set_tooltip_markup("Draw line segments");
auto group = button_segment->get_group();
button_segment->signal_clicked().connect(
    sigc::mem_fun(*this, &Mainwin::on_pen_mode_segment_click));
toolbar->append(*button_segment);

// P E N   M O D E : C O N T I G U O U S
// Add a toggle button to enable contiguous pen mode
Gtk::Image *contiguous_image =
    Gtk::manage(new Gtk::Image("contiguous.png"));
button_contiguous =
    Gtk::manage(new Gtk::RadioToolButton(*contiguous_image));
button_contiguous->set_tooltip_markup("Draw contiguous lines");
button_contiguous->set_group(group);
button_contiguous->signal_clicked().connect(
    sigc::mem_fun(*this, &Mainwin::on_pen_mode_contiguous_click));
toolbar->append(*button_contiguous);
```

A RadioToolButton is click-on, click-off.

Only one button may be on within a group.

This provides a good visual indication of which pen mode is active.

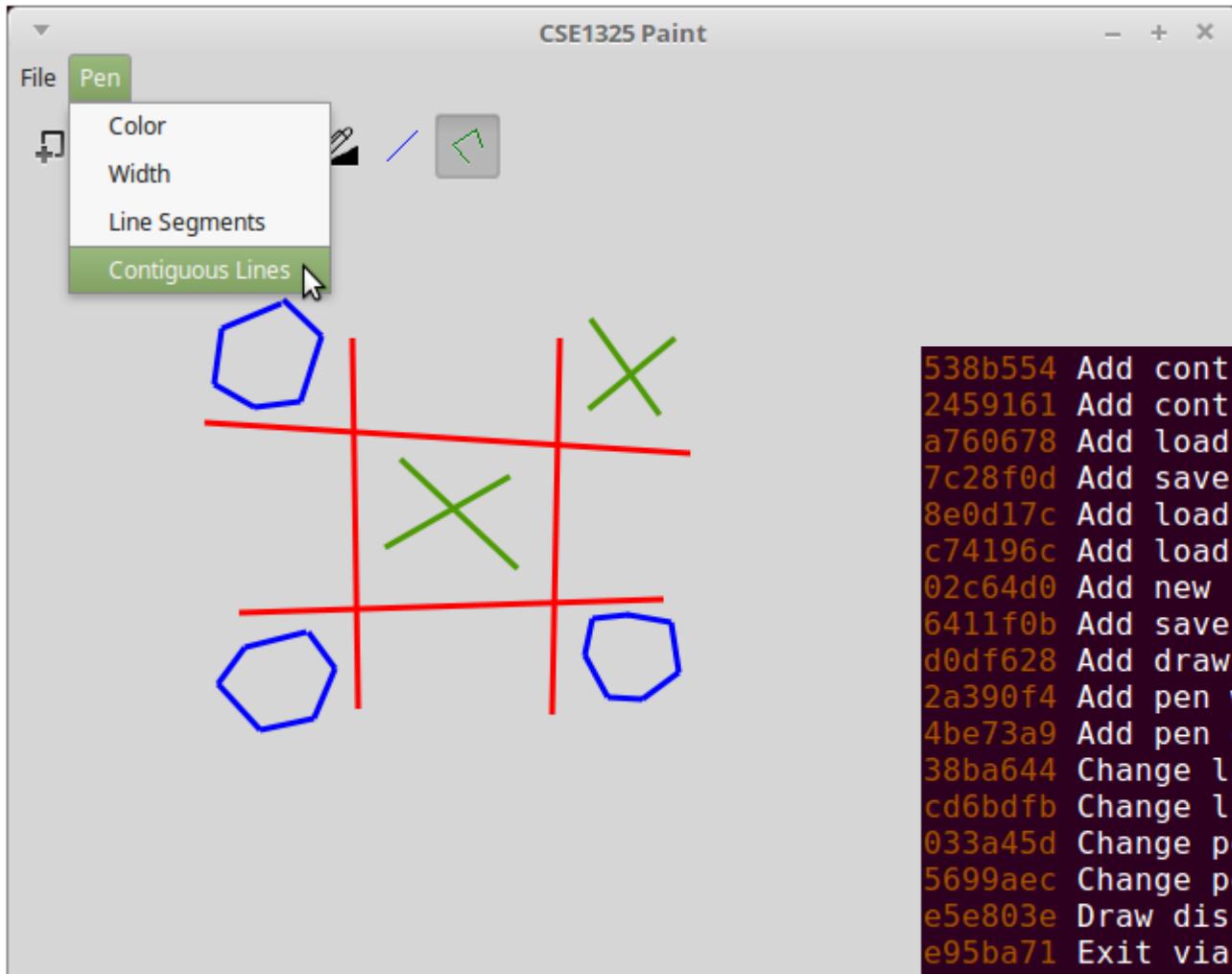
Feature 12

Step 013 – Contiguous Mode

```
bool View::on_button_press_event(GdkEventButton * event) {  
    static int previous_pen_mode;  
    if( (event->type == GDK_BUTTON_PRESS) && (event->button == 1)) {  
  
        // If the pen mode changed, clear any clicks in progress  
        if (pen_mode != previous_pen_mode) click_in_progress = false;  
  
        // Just remember the coordinates on the first click, but add a new line  
        // on the second click (and subsequent clicks in contiguous mode)  
  
        if (!click_in_progress) {  
            x1 = event->x;  
            y1 = event->y;  
            click_in_progress = true;  
        } else {  
            int x2 = event->x;  
            int y2 = event->y;  
            new_line(x1, y1, x2, y2, color, width);  
            queue_draw(); // initiate screen refresh  
            x1 = x2; // to support contiguous mode  
            y1 = y2;  
            if (pen_mode == PEN_MODE_SEGMENT) click_in_progress = false;  
        }  
        previous_pen_mode = pen_mode;  
        return true; // We handled this event  
    }  
    return false; // We did NOT handle this event  
}
```

view.cpp

Feature 13 – Draw Contiguous



```
538b554 Add contiguous mode (done!)
2459161 Add contiguous mode (sans buttons)
a760678 Add load from user-selected file
7c28f0d Add save as
8e0d17c Add load from default (done!)
c74196c Add load (not yet user-accessible)
02c64d0 Add new (not yet user-accessible)
6411f0b Add save (done!)
d0df628 Add drawing save (not yet user-accessible)
2a390f4 Add pen width button
4be73a9 Add pen color button
38ba644 Change line width (done!)
cd6bdffb Change line width (not yet user-selectable)
033a45d Change pen color (done!)
5699aec Change pen color (not yet in Line)
e5e803e Draw discrete lines
e95ba71 Exit via File > Exit
2e10779 Exit via x, checking dirty bit
afffb70 Refactor, rm turtle.*
ee35027 Baseline from turtle graphics demo
```

SCRUM Product Backlog and Next Sprint Backlog Tasks

Status	As an... I want to...	So that I can...
Finished in Sprint 1	Artist Exit via x – check for "dirty" bit and prompt to save given new data	Close the window
Finished in Sprint 1	Artist Exit via File > Exit – creating a File > Exit menu item	Close the window independent of the OS
Finished in Sprint 1	Artist Draw discrete lines – click for (x1, y1), drag to (x2, y2)	Create line art
Finished in Sprint 1	Artist Change pen color – use a color selection dialog	Create colorful drawings
Finished in Sprint 1	Artist Change pen width – use a text input dialog and atoi	Provide more texture and movement
Finished in Sprint 1	Artist Select pen properties from a toolbar	Switch pen properties more quickly
Finished in Sprint 1	Artist Save – write the lines vector to a default filename	Reload, reuse, and share drawings
Finished in Sprint 1	Artist New canvas (vector) – toss drawing (prompt to save) and start new	Start a new drawing without closing the program
Finished in Sprint 1	Artist Open – load the lines vector for the default file	Reload, reuse, and share drawings
Finished in Sprint 1	Artist Save as – write the lines vector to a user-specified file	Manage my drawing files with less work
Finished in Sprint 1	Artist Open – load the lines vector from a user-selected file	Reload, reuse, and share drawings
Finished in Sprint 1	Artist Draw contiguous straight lines – sequence of clicks	Create polygons
	Artist Draw freehand – hold down the mouse and scribble	Create complex shapes
	Artist Draw dashed lines in all modes	Make my drawings more interesting and attractive
	Artist Undo	Make mistakes without having to start over
	Artist Run Turtle Graphics file	Create reusable complex shapes
	Artist Copy Image	Place my drawings into e.g., Office documents
	Artist Paste Image	Insert drawings from e.g., Gnome Screenshot
	Artist Export to PNG format	Save my drawings in an industry standard format
	Lawyer Acknowledge license obligations	Avoid litigation

Feature ID	Description	Status
DFH	Add motion notify event handler to View	
DFH	Add Pen > Freehand	
DFH	Add freehand to tool bar	

Feature 13

Step 014 – Freehand Mode

- Freehand (or “scribble”) mode requires a new event handler
 - Button Press only responds to *buttons*, not *movement*
 - Pointer Motion is just the ticket!
- When the mouse moves with the button down, we’ll create a stream of line segments, just as if the artist is rapidly clicking the mouse button as she goes
- We also need to add a 3rd radio button to our group (and a menu item) to select Freehand mode

Feature 13

Step 014 – Freehand Mode

```
View::View(Gtk::Window& window) : win{window} {
    add_events(Gdk::BUTTON_PRESS_MASK);
    add_events(Gdk::POINTER_MOTION_MASK);
}

// Drag mouse event handler (for scribbling)
bool View::on_motion_notify_event(GdkEventMotion * event) {

    if (pen_mode != PEN_MODE_FREEHAND) return false;

    int x2, y2;
    guint state;

    x2 = (int)event->x;
    y2 = (int)event->y;
    state = event->state;

    if (state & Gdk::BUTTON1_MASK) {
        new_line(x1, y1, x2, y2, color, width);
        queue_draw(); // initiate screen refresh
        x1 = x2;
        y1 = y2;
    }

    click_in_progress = false;
    return true;
}
```

view.cpp

Although we could allow freehand drawing any time, we choose to only allow it when in Freehand mode to avoid confusion.

Feature 13

Step 014 – Freehand Mode

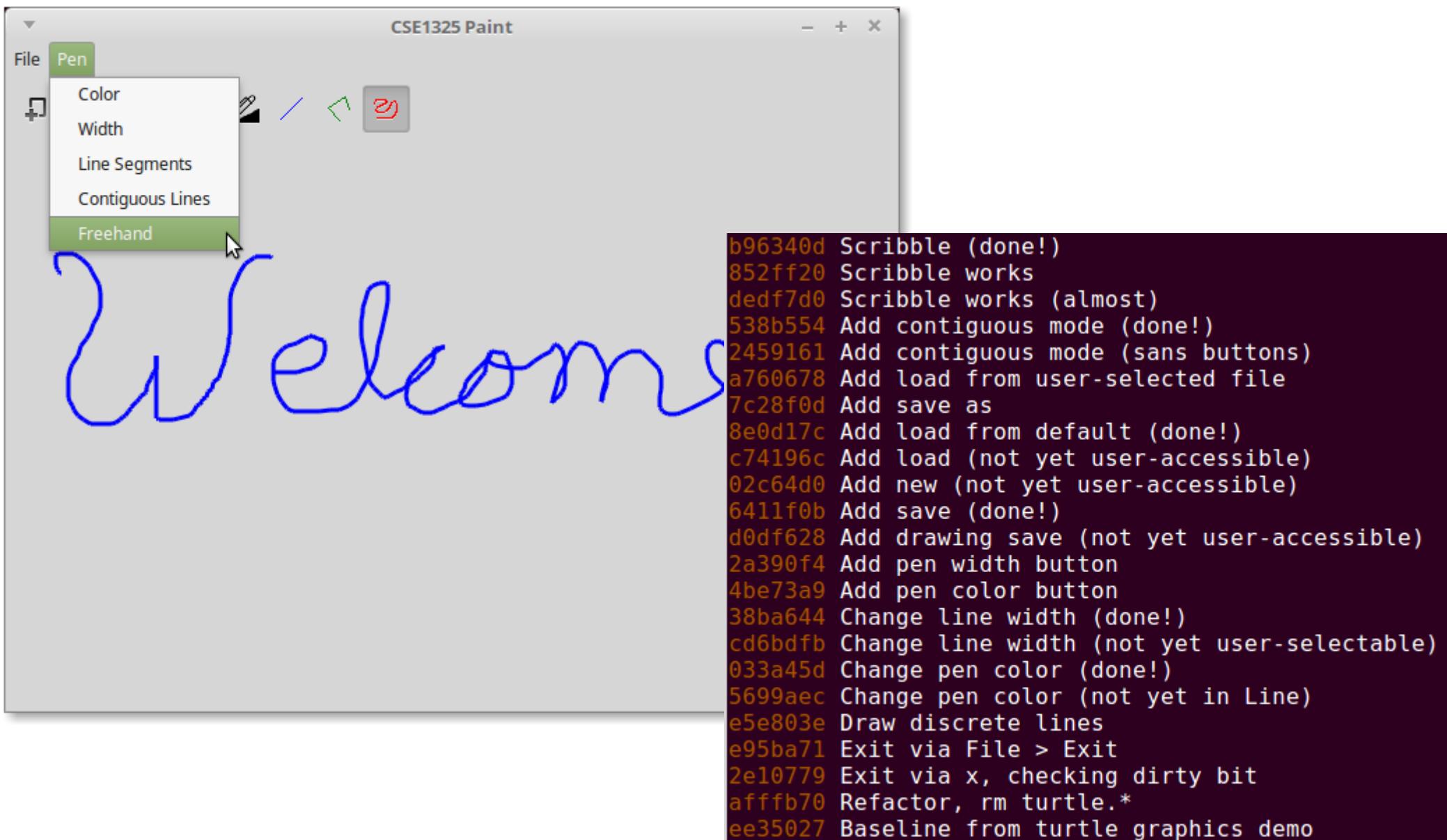
```
//      P E N   M O D E : F R E E H A N D
// Append Freehand to the Pen menu
Gtk::MenuItem *menuitem_pen_mode_freehand =
    Gtk::manage(new Gtk::MenuItem("_Freehand", true));
menuitem_pen_mode_freehand->signal_activate().connect(
    sigc::mem_fun(*this, &Mainwin::on_pen_mode_freehand_click));
penmenu->append(*menuitem_pen_mode_freehand);

//      P E N   M O D E : F R E E H A N D
// Add a toggle button to enable freehand pen mode
Gtk::Image *freehand_image = Gtk::manage(new Gtk::Image("freehand.png"));
Gtk::RadioToolButton *button_freehand =
    Gtk::manage(new Gtk::RadioToolButton(*freehand_image));
button_freehand->set_tooltip_markup("Scribble");
button_freehand->set_group(group);
button_freehand->signal_clicked().connect(
    sigc::mem_fun(*this, &Mainwin::on_pen_mode_freehand_click));
toolbar->append(*button_freehand);

void Mainwin::on_pen_mode_freehand_click() {
    view->set_pen_mode(View::PEN_MODE_FREEHAND);
}
```

mainwin.cpp

Feature 14 – Draw Freehand



SCRUM Product Backlog and Next Sprint Backlog Tasks

Status	As an... I want to...	So that I can...
Finished in Sprint 1	Artist Exit via x – check for “dirty” bit and prompt to save given new data	Close the window
Finished in Sprint 1	Artist Exit via File > Exit – creating a File > Exit menu item	Close the window independent of the OS
Finished in Sprint 1	Artist Draw discrete lines – click for (x1, y1), drag to (x2, y2)	Create line art
Finished in Sprint 1	Artist Change pen color – use a color selection dialog	Create colorful drawings
Finished in Sprint 1	Artist Change pen width – use a text input dialog and atoi	Provide more texture and movement
Finished in Sprint 1	Artist Select pen properties from a toolbar	Switch pen properties more quickly
Finished in Sprint 1	Artist Save – write the lines vector to a default filename	Reload, reuse, and share drawings
Finished in Sprint 1	Artist New canvas (vector) – toss drawing (prompt to save) and start new	Start a new drawing without closing the program
Finished in Sprint 1	Artist Open – load the lines vector for the default file	Reload, reuse, and share drawings
Finished in Sprint 1	Artist Save as – write the lines vector to a user-specified file	Manage my drawing files with less work
Finished in Sprint 1	Artist Open – load the lines vector from a user-selected file	Reload, reuse, and share drawings
Finished in Sprint 1	Artist Draw contiguous straight lines – sequence of clicks	Create polygons
Finished in Sprint 1	Artist Draw freehand – hold down the mouse and scribble	Create complex shapes
Artist	Draw dashed lines in all modes	Make my drawings more interesting and attractive
Artist	Undo	Make mistakes without having to start over
Artist	Run Turtle Graphics file	Create reusable complex shapes
Artist	Copy Image	Place my drawings into e.g., Office documents
Artist	Paste Image	Insert drawings from e.g., Gnome Screenshot
Artist	Export to PNG format	Save my drawings in an industry standard format
Lawyer	Acknowledge license obligations	Avoid litigation

DSH
DSH
DSH
DSH

Add dash patterns, dash index, and getters / setters to View
Add dash index to Line
Update View::on_draw to use Line's dash setting
Add Pen > Dash with custom dialog to select dash pattern

Feature 14

Step 015 – Dashed Lines

```
#include "line.h"
#include <gtkmm.h>

Line::Line(double p_x1, double p_y1, double p_x2, double p_y2,
           Gdk::RGBA p_color, double p_width, int p_dash)
: _x1{p_x1}, _y1{p_y1}, _x2{p_x2}, _y2{p_y2},
  _color{p_color}, _width{p_width}, _dash{p_dash} { }

Line::Line(std::istream& ist) {
    std::string rgba_color;
    ist >> _x1 >> _y1 >> _x2 >> _y2 >> _width >> _dash >> rgba_color;
    _color = Gdk::RGBA{rgba_color};
}

double Line::x1() {return _x1;}
double Line::y1() {return _y1;}
double Line::x2() {return _x2;}
double Line::y2() {return _y2;}
Gdk::RGBA Line::color() {return _color;}
double Line::width() {return _width;}
int Line::dash() {return _dash;}
void Line::save(std::ostream& ost) {
    ost << _x1 << ' ' << _y1 << ' ' << _x2 << ' ' << _y2 << ' '
        << _width << ' ' << _dash << ' ' << _color.to_string() << ' ';
}
```

line.cpp

A Word about File Compatibility

- ctp files written by earlier program iterations will fail for this iteration
 - And files written by this iteration will fail with earlier iterations as well
- How can we maximize file compatibility?

A Word about File Compatibility

- Linux uses heuristics to identify file types
 - See ‘man file’ for details
- Some standard formats, e.g., XML, are inherently extensible
- Executable & Linkable Format (ELF) uses a “magic number”
 - The first 32-bit word is 07F, ‘E’, ‘L’, ‘F’
 - Subsequent byte is e_version of file format
- By prepending a text identifier (e.g, “CTPF”) and version (e.g., “1.0.0”), additional code could provide backward compatibility (e.g., set dash to ‘0’)
 - Forward compatibility is a bit more challenging

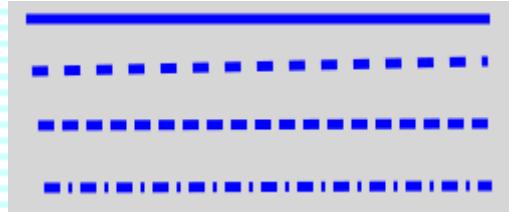
Feature 14

Step 015 – Dashed Lines

```
// Current dash pattern (a vector index)
int dash = 0;

// Supported dashed lines
typedef std::vector<double> dash_pattern;
std::vector<dash_pattern> dashes = {
    {8.0, 0.0},
    {8.0, 8.0},
    {8.0, 4.0},
    {8.0, 4.0, 2.0, 4.0}
}
```

view.h



- A dash pattern is a vector of doubles
 - Even indexes are number of “on” pixels
 - Odd indexes are number of “off” pixels
- For initial release, just predefined a few and name them
 - Long term, we’d want to enable user to create their own and select based on a dynamically generated visual representation

Feature 14

Step 015 – Dashed Lines

```
// Draw the lines any time gtkmm needs to refresh the widget
bool View::on_draw(const Cairo::RefPtr<Cairo::Context>& cr) {
    // If nothing to draw, we're done
    if (lines.size() == 0) return true;

    // Create a Cairomm context to manage our drawing
    Gtk::Allocation allocation = get_allocation();

    // Draw the lines
    for(Line l : lines) {
        Gdk::Cairo::set_source_rgba(cr, l.color());
        cr->set_line_width(l.width());
        cr->set_dash(dashes[l.dash()], 0.0);
        cr->move_to(l.x1(), l.y1());
        cr->line_to(l.x2(), l.y2());
        cr->stroke();
    }

    // Drawing was successful
    return true;
}
```

view.cpp

Feature 14

Step 015 – Dashed Lines

```
//      D A S H   P A T T E R N S
// Append dash patterns to the Pen menu
Gtk::MenuItem *menuitem_dash_pattern =
    Gtk::manage(new Gtk::MenuItem{"_Dash Pattern", true});
menuitem_dash_pattern->signal_activate().connect(
    sigc::mem_fun(*this, &Mainwin::on_dash_pattern_click));
penmenu->append(*menuitem_dash_pattern);

void Mainwin::on_dash_pattern_click() {
    Gtk::Dialog dialog{"Select Dash Pattern", *this};

    Gtk::ComboBoxText c_dash;
    c_dash.set_size_request(160);
    c_dash.append("Solid Line");
    c_dash.append("Alternating");
    c_dash.append("Mostly Dash");
    c_dash.append("Dash Dot");
    dialog.get_vbox()->pack_start(c_dash, Gtk::PACK_SHRINK);

    // Show dialog
    dialog.add_button("Cancel", 0);
    dialog.add_button("OK", 1);
    dialog.show_all();
    int result = dialog.run();

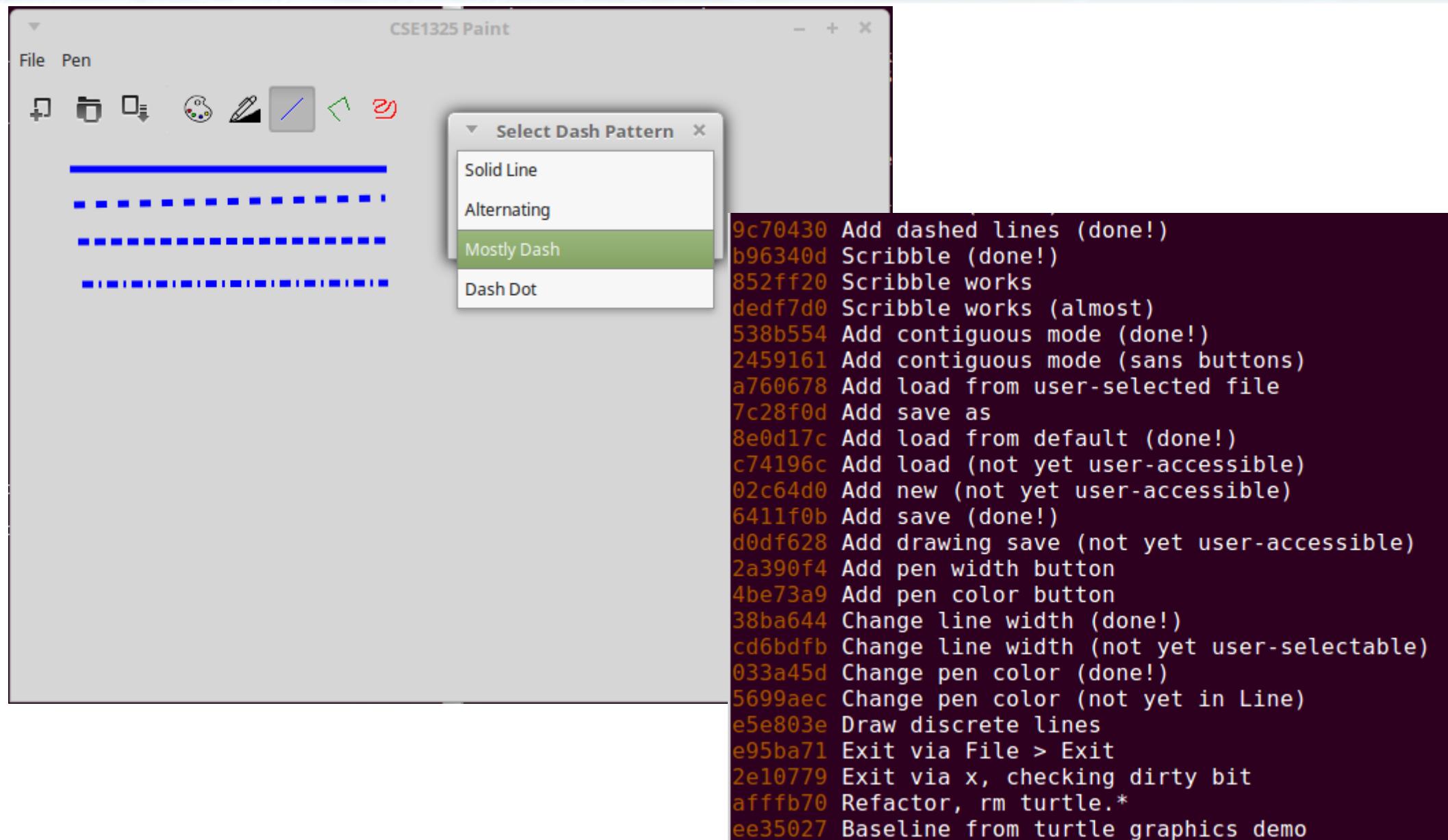
    if (result == 1) view->set_dash_pattern(c_dash.get_active_row_number());
    dialog.close();
}
```

mainwin.cpp

The lower widget is a drop-down list of patterns.
The patterns are actually defined as
View::dashes (in the header file). This list MUST
match that vector exactly!

A dynamically constructed list of pattern images
would be better, given time.
Definitely next release! :-D

Feature 14 – Dashed Lines



SCRUM Product Backlog and Next Sprint Backlog Tasks

Status	As an... I want to...	So that I can...
Finished in Sprint 1	Artist Exit via x – check for “dirty” bit and prompt to save given new data	Close the window
Finished in Sprint 1	Artist Exit via File > Exit – creating a File > Exit menu item	Close the window independent of the OS
Finished in Sprint 1	Artist Draw discrete lines – click for (x1, y1), drag to (x2, y2)	Create line art
Finished in Sprint 1	Artist Change pen color – use a color selection dialog	Create colorful drawings
Finished in Sprint 1	Artist Change pen width – use a text input dialog and atoi	Provide more texture and movement
Finished in Sprint 1	Artist Select pen properties from a toolbar	Switch pen properties more quickly
Finished in Sprint 1	Artist Save – write the lines vector to a default filename	Reload, reuse, and share drawings
Finished in Sprint 1	Artist New canvas (vector) – toss drawing (prompt to save) and start new	Start a new drawing without closing the program
Finished in Sprint 1	Artist Open – load the lines vector for the default file	Reload, reuse, and share drawings
Finished in Sprint 1	Artist Save as – write the lines vector to a user-specified file	Manage my drawing files with less work
Finished in Sprint 1	Artist Open – load the lines vector from a user-selected file	Reload, reuse, and share drawings
Finished in Sprint 1	Artist Draw contiguous straight lines – sequence of clicks	Create polygons
Finished in Sprint 1	Artist Draw freehand – hold down the mouse and scribble	Create complex shapes
Finished in Sprint 1	Artist Draw dashed lines in all modes	Make my drawings more interesting and attractive
	Artist Undo	Make mistakes without having to start over
	Artist Run Turtle Graphics file	Create reusable complex shapes
	Artist Copy Image	Place my drawings into e.g., Office documents
	Artist Paste Image	Insert drawings from e.g., Gnome Screenshot
	Artist Export to PNG format	Save my drawings in an industry standard format
Lawyer	Acknowledge license obligations	Avoid litigation

Feature ID	Description	Status
UNDO	Add View::set_undo_mark and undo methods	
UNDO	Set undo marks when a line segment is drawn	
UNDO	Set undo marks when entering freehand mode	
UNDO	Add Edit > Undo	
UNDO	Add undo button to tool bar	



Feature 15

Step 016 – Undo

- The user will expect to “undo” the last “thing” they drew
 - How would you implement this feature?



Feature 15

Step 016 – Undo

- Our drawing is a *time-ordered* list of vectors
 - Removing the last Line object from the lines vector is effectively an undo
- But what about Freehand (“scribble”) mode
 - This generates perhaps hundreds of vector entries
 - The user will expect an undo to delete the entire scribble, not one pixel of it at a time
 - So we implement to the user's expectations
(what is this called?)
- Keep a second vector of “undo marks”
 - Before an undoable drawing operation, set a mark – just push_back the size of the lines array onto the undo_marks vector
 - To undo, just delete lines from the end of the vector back to what is indicated in the last undo marks vector entry (back()), then delete the last undo marks entry

Feature 15

Step 016 – Undo

```
// Mark the current line as the last before a new undo event  
void set_undo_mark();  
  
// Undo drawings back to the most recent undo mark  
void undo();  
  
// List of indices into lines for undo implementation  
std::vector<int> undo_marks = {0};
```

view.h

```
void View::set_undo_mark() {  
    if (undo_marks.size() == 0 || undo_marks.back() != lines.size()) view.cpp  
        undo_marks.push_back(lines.size());  
}  
  
void View::undo() {  
    while (lines.size() > undo_marks.back()) {  
        lines.pop_back();  
    }  
    if (undo_marks.size() > 1) undo_marks.pop_back();  
}
```

Feature 15

Step 016 – Undo

```
// Normal button press event handler
bool View::on_button_press_event(GdkEventButton * event) {                                view.cpp
    static int previous_pen_mode;

    // If this is a left-mouse button press, we're interested!
    if( (event->type == GDK_BUTTON_PRESS) && (event->button == 1)) {

        // If the pen mode changed, clear any clicks in progress
        if (pen_mode != previous_pen_mode) click_in_progress = false;

        if (!click_in_progress) {
            x1 = event->x;
            y1 = event->y;
            click_in_progress = true;
        } else {
            set_undo_mark();
            int x2 = event->x;
            int y2 = event->y;
            new_line(x1, y1, x2, y2, color, width, dash);
            queue_draw(); // initiate screen refresh
            x1 = x2;      // to support contiguous mode
            y1 = y2;
            if (pen_mode == PEN_MODE_SEGMENT) click_in_progress = false;
        }
        previous_pen_mode = pen_mode;
        return true; // We handled this event
    }
    return false; // We did NOT handle this event
}
```

Feature 15

Step 016 – Undo

```
//      E D I T
// Create an Edit menu and add to the menu bar
Gtk::MenuItem *menuitem_edit =
    Gtk::manage(new Gtk::MenuItem{"_Edit", true});
menubar->append(*menuitem_edit);
Gtk::Menu *editmenu = Gtk::manage(new Gtk::Menu{});
menuitem_edit->set_submenu(*editmenu);

//          U N D O
// Append Undo to the Edit menu
Gtk::MenuItem *menuitem_undo = Gtk::manage(new Gtk::MenuItem{"_Undo", true});
menuitem_undo->signal_activate().connect(
    sigc::mem_fun(*this, &Mainwin::on_undo_click));
editmenu->append(*menuitem_undo);

//
// The user wants to undo the last operation
//

void Mainwin::on_undo_click() {
    view->undo();
    view->queue_draw();
}
```

mainwin.cpp

SCRUM Product Backlog and Next Sprint Backlog Tasks

Status	As an... I want to...	So that I can...
Finished in Sprint 1	Artist Exit via x – check for "dirty" bit and prompt to save given new data	Close the window
Finished in Sprint 1	Artist Exit via File > Exit – creating a File > Exit menu item	Close the window independent of the OS
Finished in Sprint 1	Artist Draw discrete lines – click for (x1, y1), drag to (x2, y2)	Create line art
Finished in Sprint 1	Artist Change pen color – use a color selection dialog	Create colorful drawings
Finished in Sprint 1	Artist Change pen width – use a text input dialog and <u>atoi</u>	Provide more texture and movement
Finished in Sprint 1	Artist Select pen properties from a <u>toolbar</u>	Switch pen properties more quickly
Finished in Sprint 1	Artist Save – write the lines vector to a default filename	Reload, reuse, and share drawings
Finished in Sprint 1	Artist New canvas (vector) – toss drawing (prompt to save) and start new	Start a new drawing without closing the program
Finished in Sprint 1	Artist Open – load the lines vector for the default file	Reload, reuse, and share drawings
Finished in Sprint 1	Artist Save as – write the lines vector to a user-specified file	Manage my drawing files with less work
Finished in Sprint 1	Artist Open – load the lines vector from a user-selected file	Reload, reuse, and share drawings
Finished in Sprint 1	Artist Draw contiguous straight lines – sequence of clicks	Create polygons
Finished in Sprint 1	Artist Draw freehand – hold down the mouse and scribble	Create complex shapes
Finished in Sprint 1	Artist Draw dashed lines in all modes	Make my drawings more interesting and attractive
Finished in Sprint 1	Artist Undo	Make mistakes without having to start over
Artist	Run Turtle Graphics file	Create reusable complex shapes
Artist	Copy Image	Place my drawings into e.g., Office documents
Artist	Paste Image	Insert drawings from e.g., Gnome <u>Screenshot</u>
Artist	Export to PNG format	Save my drawings in an industry standard format
Lawyer	Acknowledge license obligations	Avoid litigation

With Sprint 1 almost over, the lawyers manage to bump their feature enough to fit it into the current plan. Need to give credit where due, anyway.

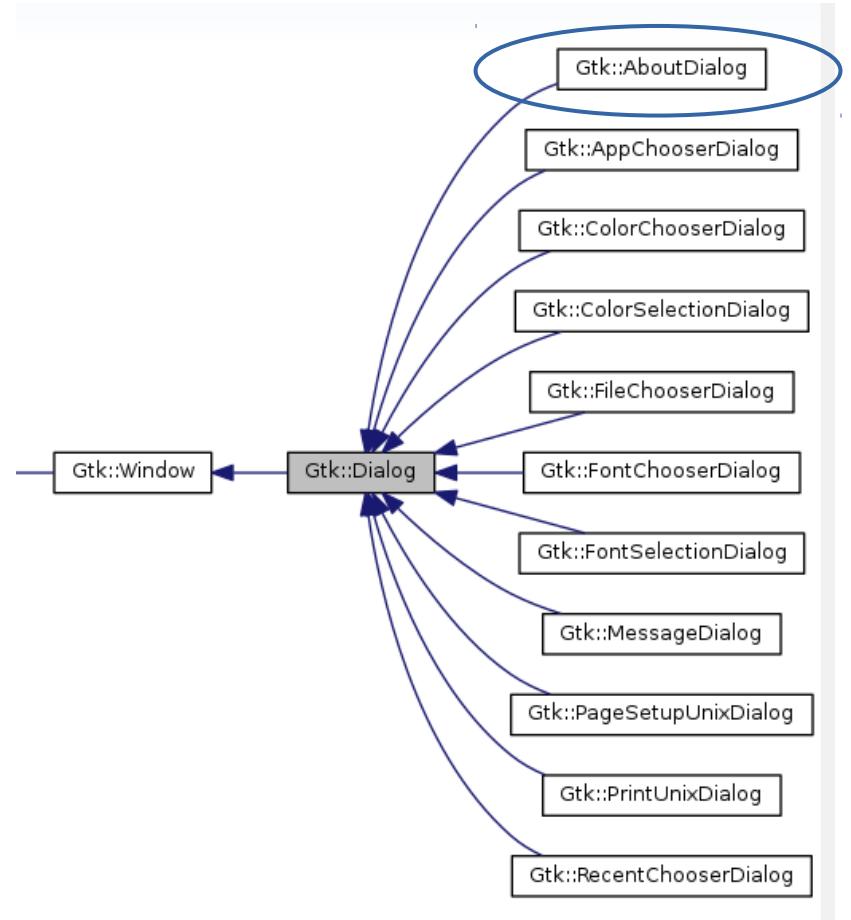
ABT
ABT

Add About dialog
Add Help > About

Feature 20

Step 017 – About

- For Nim, we used a `MessageDialog` with Pango enabled for our About dialog
- But gtkmm has a predefined class, `AboutMessage`, specifically for this purpose



Feature 20

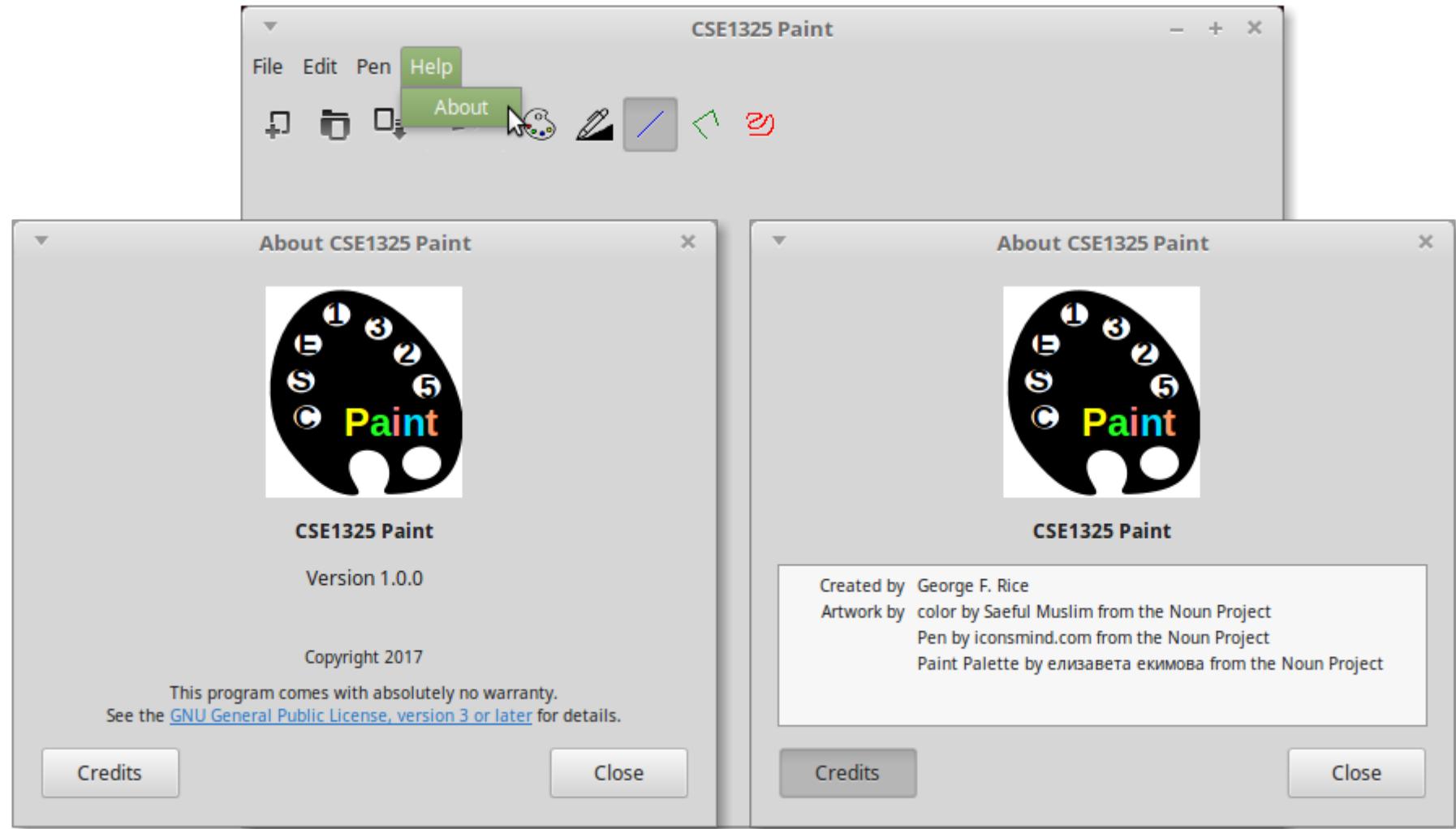
Step 017 – About

```
//      H E L P
// Create a Help menu and add to the menu bar
Gtk::MenuItem *menuitem_help =
    Gtk::manage(new Gtk::MenuItem("_Help", true));
menubar->append(*menuitem_help);
Gtk::Menu *helpmenu = Gtk::manage(new Gtk::Menu());
menuitem_help->set_submenu(*helpmenu);
//          A B O U T
// Append About to the Help menu
Gtk::MenuItem *menuitem_about = Gtk::manage(new Gtk::MenuItem("About", true));
menuitem_about->signal_activate().connect(
    sigc::mem_fun(*this, &Mainwin::on_about_click));
helpmenu->append(*menuitem_about);

void Mainwin::on_about_click() {
    Gtk::AboutDialog dialog{};
    dialog.set_transient_for(*this);
    dialog.set_program_name("CSE1325 Paint");
    auto logo = Gdk::Pixbuf::create_from_file("logo.png");
    dialog.set_logo(logo);
    dialog.set_version("Version 1.0.0");
    dialog.set_copyright("Copyright 2017");
    dialog.set_license_type(Gtk::License::LICENSE_GPL_3_0);
    std::vector< Glib::ustring > authors = {"George F. Rice"};
    dialog.set_authors(authors);
    std::vector< Glib::ustring > artists = {...}; // vector redacted for space
    dialog.set_artists(artists);
    dialog.run();
}
```

mainwin.cpp

Feature 20 – About



Feature 20 – About

```
ricegef@pluto:~/dev/cpp/201708/13_14/CSE1325_Paint/dev$ gitl
* 6618ef2 Final tweaks for release (4 seconds ago)
* 5dfdacb Official About dialog (23 hours ago)
* 8ca5534 Basic About dialog (23 hours ago)
* e5d43b8 Add undo (done!) (24 hours ago)
* 9c70430 Add dashed lines (done!) (24 hours ago)
* b96340d Scribble (done!) (25 hours ago)
* 852ff20 Scribble works (26 hours ago)
* dedf7d0 Scribble works (almost) (27 hours ago)
* 538b554 Add contiguous mode (done!) (2 days ago)
* 2459161 Add contiguous mode (sans buttons) (2 days ago)
* a760678 Add load from user-selected file (2 days ago)
* 7c28f0d Add save as (2 days ago)
* 8e0d17c Add load from default (done!) (2 days ago)
* c74196c Add load (not yet user-accessible) (2 days ago)
* 02c64d0 Add new (not yet user-accessible) (2 days ago)
* 6411f0b Add save (done!) (2 days ago)
* d0df628 Add drawing save (not yet user-accessible) (2 days ago)
* 2a390f4 Add pen width button (2 days ago)
* 4be73a9 Add pen color button (2 days ago)
* 38ba644 Change line width (done!) (2 days ago)
* cd6bdfb Change line width (not yet user-selectable) (3 days ago)
* 033a45d Change pen color (done!) (10 days ago)
* 5699aec Change pen color (not yet in Line) (10 days ago)
* e5e803e Draw discrete lines (10 days ago)
* e95ba71 Exit via File > Exit (10 days ago)
* 2e10779 Exit via x, checking dirty bit (10 days ago)
* afffb70 Refactor, rm turtle.* (10 days ago)
* ee35027 Baseline from turtle graphics demo (10 days ago)
```

Exam #2 is Tuesday, October 17

- Review the study sheet and all Quick Review solutions
- Do and review the Practice Exam