The Drawing Program

Filip Eller, 1004225

Bachelor's Programme of Science and Technology, 1st year

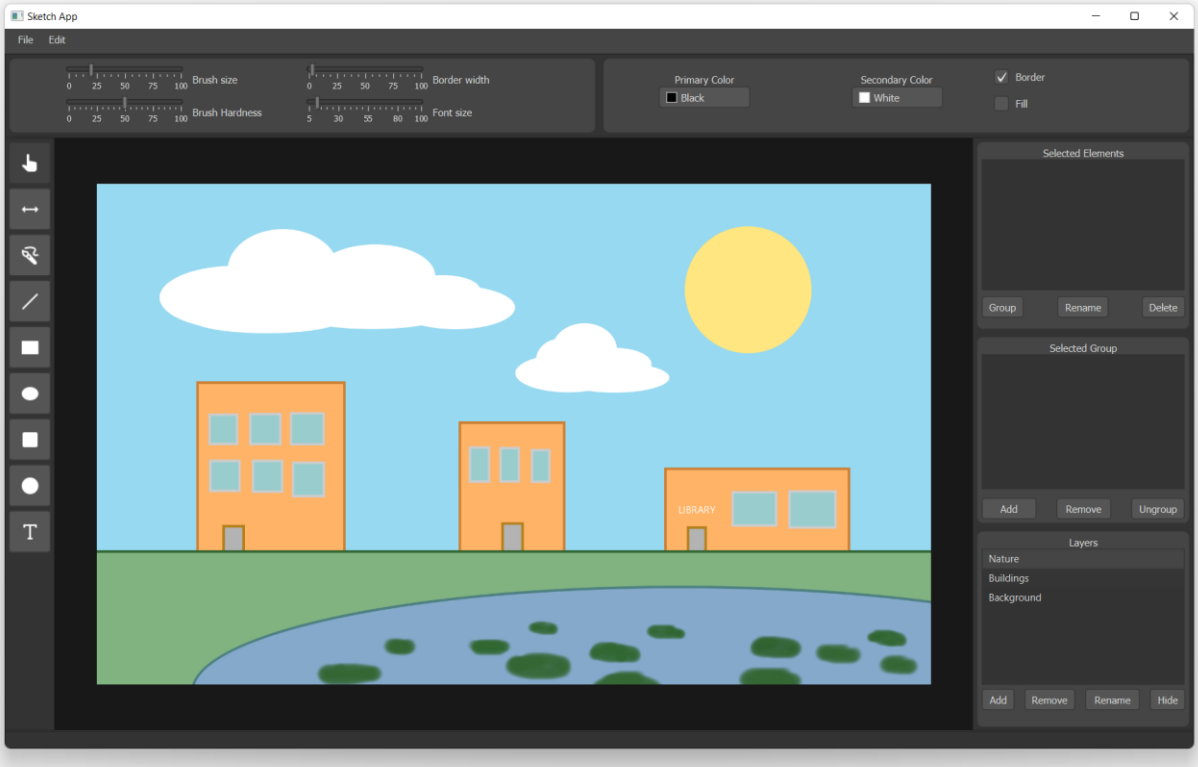
24.4.2022

General description

I have made a drawing program at demanding difficulty. The user of the program can draw free brush strokes, straight lines, rectangles, ellipses, squares, and circles. Text can also be added to the drawing. The user can configure the colours, brush size and hardness, border width and font size of the elements they are going to draw. These properties of a drawn element can also be modified later. The user can also move drawn elements, but not scale or rotate them unlike I had planned. These actions were not, however, required anyway.

Any actions can be undone while the drawing remains open. The action history is discarded when the program is closed or the user switches to another drawing. The user can save their drawings to files and load existing drawings from files. A new blank drawing can also be created.

There are two ways for the user to structure their work: groups and layers. Selected elements as well as grouped elements can be modified together. Elements can be added and removed from a given group, and a group can be broken up. All elements belong to some layer. There is only one empty layer at first, but the user can add as many layers as they like. The new layers are created as transparent. Layers are displayed on top of each other so that elements on upper layers cover elements in lower layers.

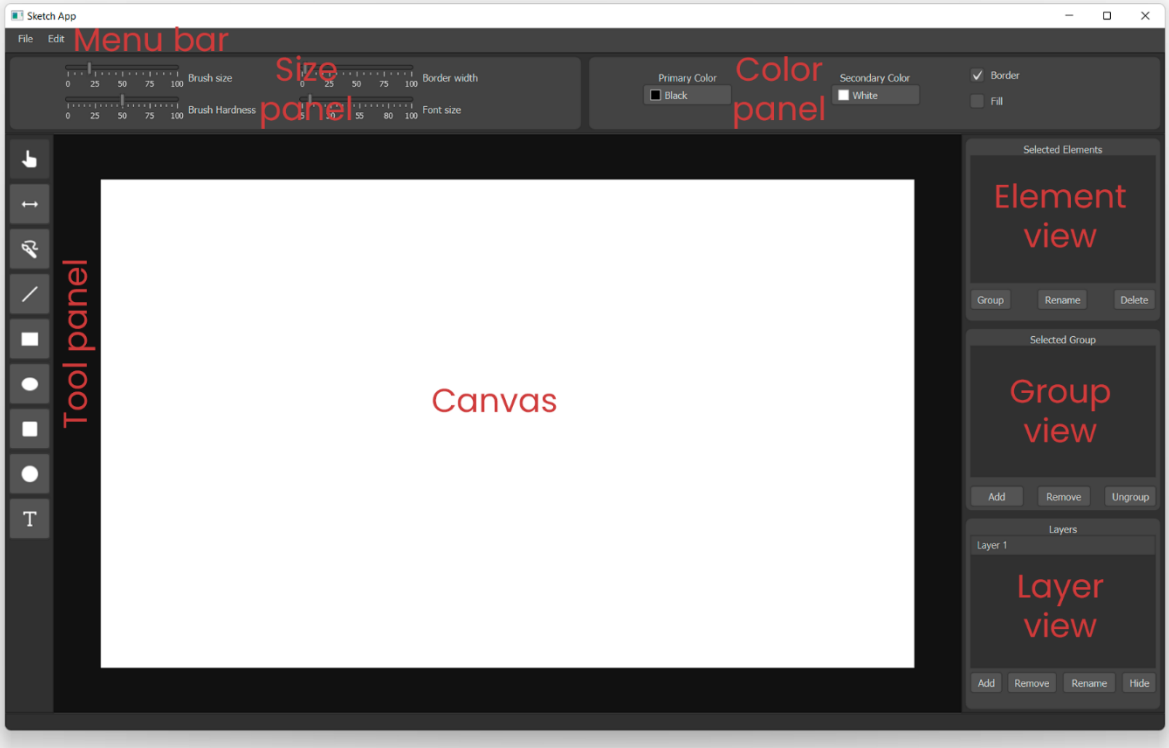


User interface

The program is started by clicking at the play button next to the Main object's first actual line inside src/main/scala/gui/Main.scala:

```
16
17    object Main extends JFXApp {
18
```

The program has a graphical user interface, which I made with Scene Builder, FXML and CSS:



Tool panel

Here the user can select which tool to use on the canvas. The name of a tool is shown in a tooltip after hovering on the tool's button for a short while. This also tells the corresponding hotkey for switching to that tool. There are nine tools in total: Selection, Move, Brush, Line, Rectangle, Ellipse, Square, Circle, and Text.

Canvas

Here the user can draw their artwork. The selected tool is used when the user clicks and drags on the canvas.

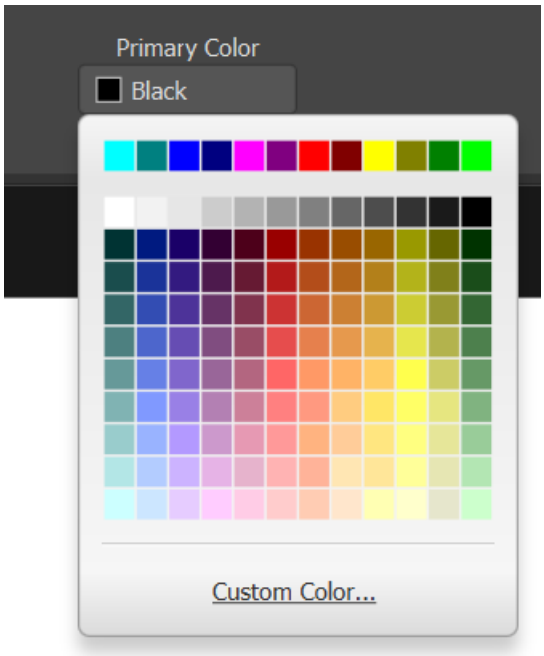
Size panel

Here the user can make some configurations for using the tool, namely changing the size and hardness of the brush strokes made with Brush and Line tools, changing the border width used by the tools for drawing shapes, and changing the font size used by the Text tool.

This panel also serves another purpose: if the user has selected some elements, the same controls are used for changing the corresponding properties of those elements.

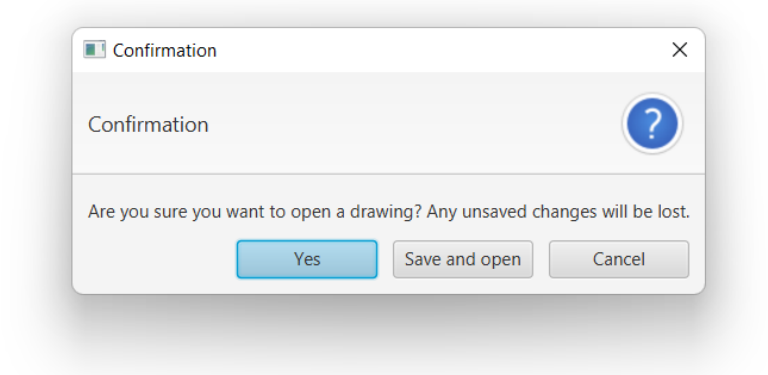
Colour panel

Like the Size panel, but this one controls the primary and secondary colours used by the tools. Primary colour is used for brush strokes, the borders of shapes, and text. Secondary colour in turn is only used as the fill colour of shapes. Clicking on the primary or the secondary colour opens a colour palette for choosing the new colour. If none of the given colours is suitable, the user can choose any 8-bit colour by clicking on "Custom color…". The Colour panel, too, can change the properties of selected elements if there are any.



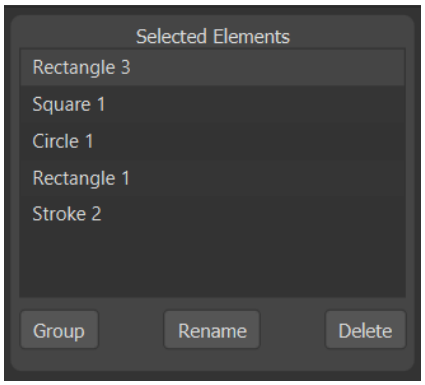
Menu bar

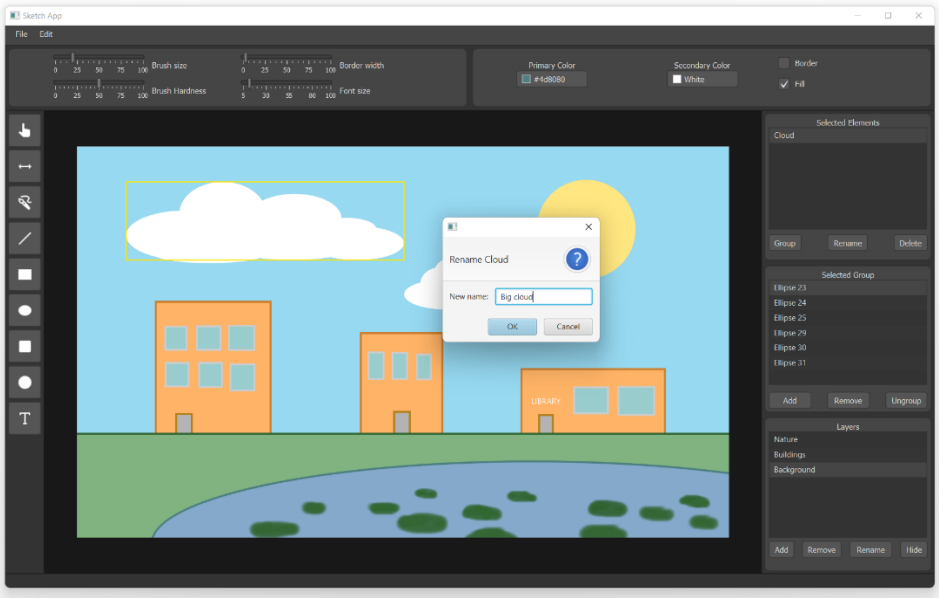
File management is found in the File menu. Here the user can create a new drawing, save the current drawing to a file, or open an existing drawing from a file. If the user has a non-empty Drawing open, opening or creating a Drawing must be confirmed through a dialog. The Edit menu contains some useful actions, namely selecting all elements, deselecting all selected elements, deleting selected and undoing previous actions. The shortcuts for these actions are also listed.



Element view

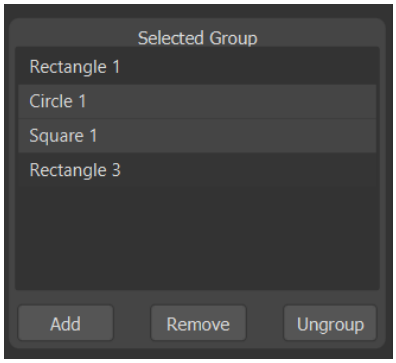
All the selected elements are displayed here. If there are plenty of them, scrolling the view up and down might be needed to see all of them. The “Group” button forms a group of all the selected elements. “Rename” opens a dialog for changing the name of the element that is focused on the list of selected elements. Focusing an element in this view requires clicking on the element’s name. The “Delete” button deletes all selected elements.





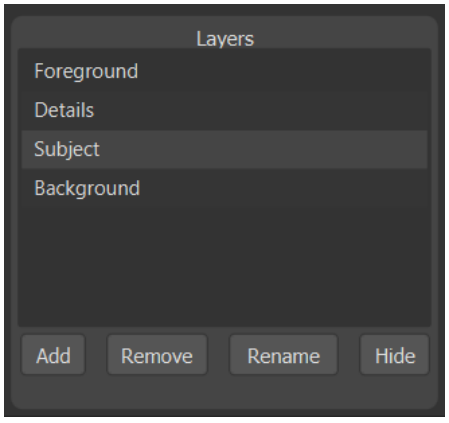
Group view

This view shows all the elements of the selected group. The most recently selected element that is a group is considered the selected group. The “Add” button adds all other selected elements to this group. The “Remove” button removes from this group those elements that are focused in the group view. Again, focusing happens by clicking on an element’s name, but here the user can focus multiple elements at a time by holding shift or control keys while clicking. “Ungroup” disassembles the group whereupon elements inside it become individual elements again.



Layer view

Here the user can find all the layers that make up their artwork. The different layers are rendered on top of each other on the canvas in the same order as they appear in this view. The topmost layer on the list appears on the top of the drawing and is not covered by other layers. The “Add” button adds a new layer to the drawing. “Remove” deletes the focused layer. “Hide” hides or unhides the focused layer. A hidden layer is not rendered on the canvas and cannot be drawn on.



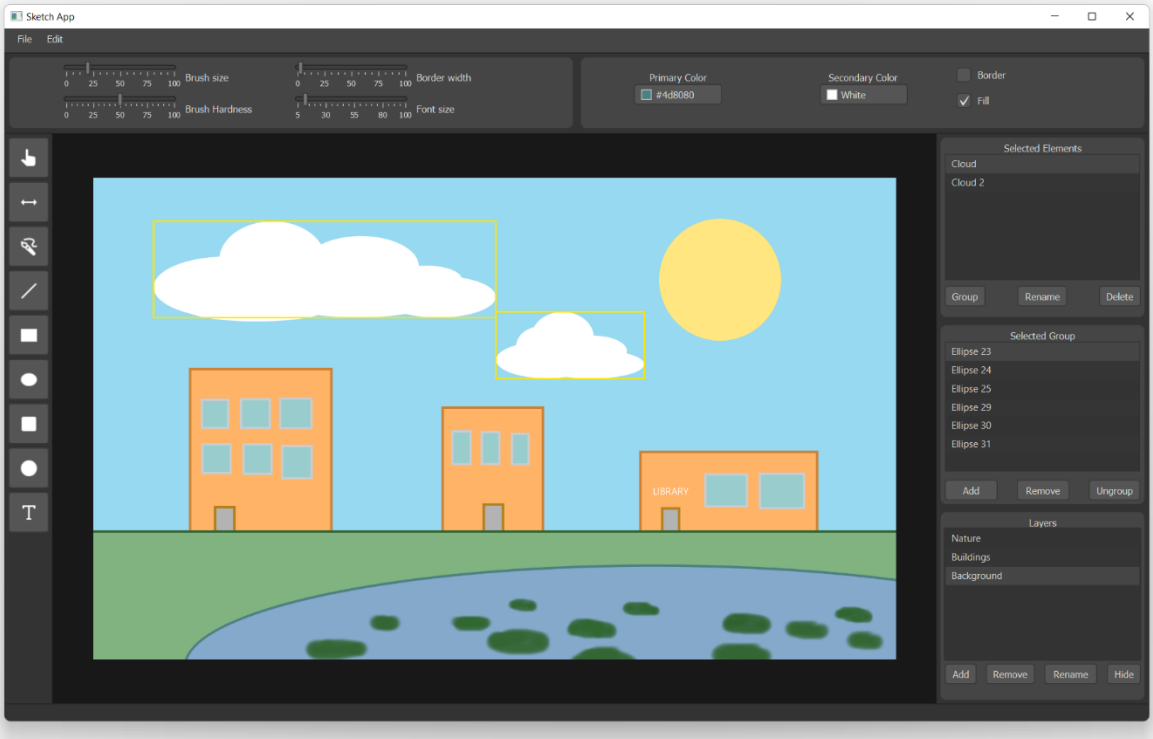
Tools

As was mentioned in the Tool panel section, the user has nine tools to choose from. Let use go over each of them.

Selection

With this tool the user can click an element to select it or click and drag to select all elements that are dragged over. Holding shift while clicking and/or dragging allows adding elements to the selection while

holding control allows removing elements from selection. A yellow rectangle is painted around the selected Elements.



Move

With this tool the user can move the selected elements by clicking on one of them and the dragging to move all of them. The moving is finished when the user releases the mouse.

Brush

With this tool the user can draw freely by clicking and dragging. The configured brush size determines how thick the brush stroke is while the brush hardness determines how hard or soft the stroke looks like.

Line

With this tool the user can draw straight lines by clicking at the line's starting point and then dragging to the end point and releasing. Otherwise, it works just like the brush tool.

Rectangle

With this tool the user can draw rectangles. The user clicks at a point to anchor one of the rectangle's corners, then drags and releases to place the opposing corner.

Ellipse

This tool works like the rectangle tool but draws ellipses. The click and release points correspond to the corners of the ellipse's bounding box.

Square

With this tool the user can draw square. The user clicks at a point to anchor one of the square's corners, then drags and releases to place one of the opposing sides.

Circle

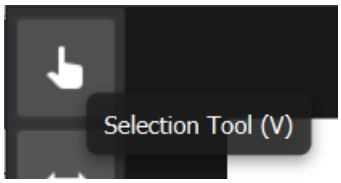
This tool works like the square tool but draws circles. The click and release points correspond to the circle's bounding box.

Text

With this tool the user can add text to their drawing. The user adds a text box the same way they would draw a rectangle. Any selected text box can be written in by typing on the keyboard. Writing does not require the text tool, only adding new text boxes does. If multiple text boxes are selected, text will be added to all of them. When a new text box is added, it is automatically selected so the user can start writing on it right away.

Shortcuts

The user can speed up their work by utilising the available shortcuts. Shortcuts can be found in the program through the menus and in the tooltips that are shown after hovering over a related button for a short while. The tools have two possible shortcuts because the ones with just a letter cannot be used when a text box is selected.



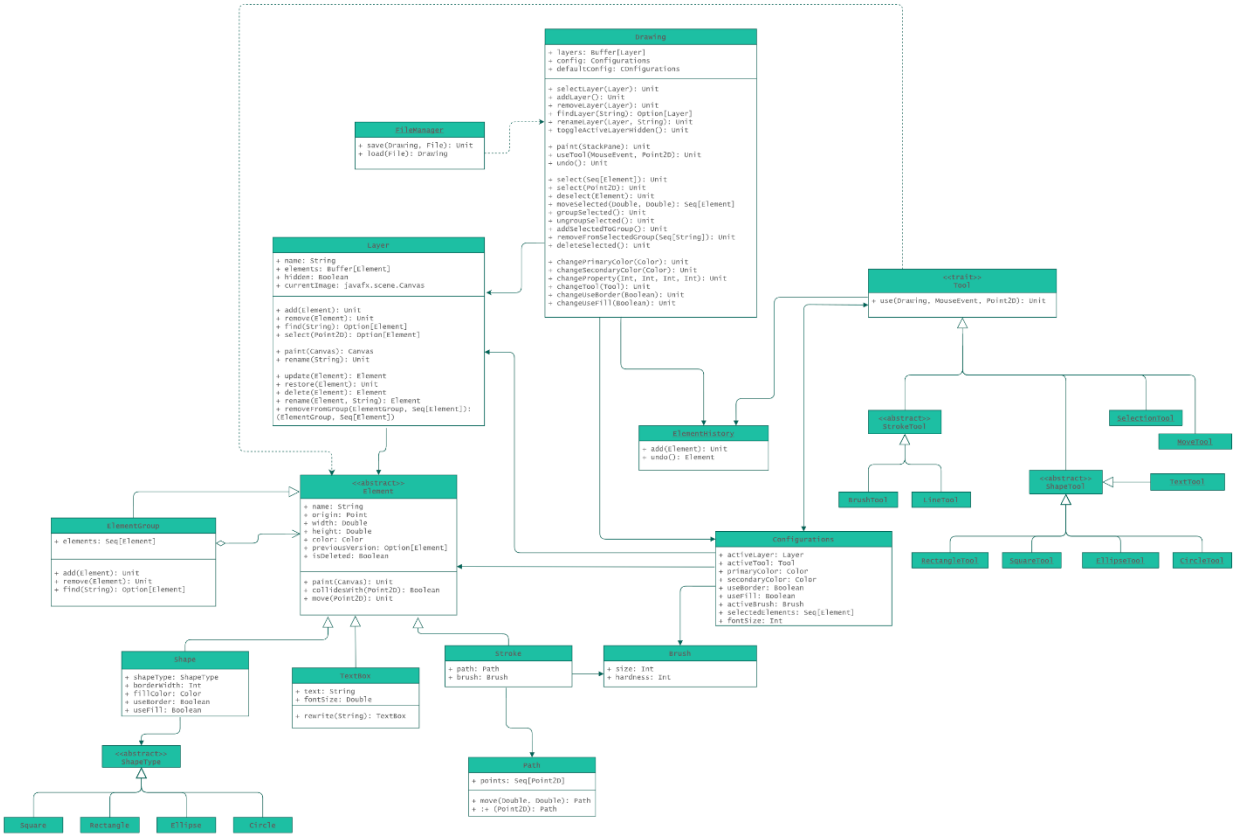
Selection tool	V / Alt+V
Move tool	M / Alt+M
Brush tool	B / Alt+B
Line tool	L / Alt+L
Rectangle tool	R / Alt+R
Ellipse tool	E / Alt+E
Square tool	S / Alt+S
Circle tool	C / Alt+C
Text tool	T / Alt+T
Undo	CTRL+Z
New drawing	CTRL+N

Open drawing	CTRL+O
Save drawing	CTRL+S
Select all	CTRL+A
Deselect all	CTRL+D
Delete selected	CTRL+X
Group selected	CTRL+G
Ungroup selected	CTRL+U
Hide/unhide layer	CTRL+H
Toggle border	CTRL+B
Toggle fill	CTRL+F

Program structure

The main sub-parts of the program are the packages “logic” and “gui”. The logic package contains the actual logic of the program while the gui package controls the graphical user interface. The visuals of the GUI are, however, not in the package gui but in the FXML and CSS files of the src/main/resources directory.

The class structure of the logic package looks as follows (reading the methods might require a bit of zooming in):



Drawing is the most important class. It keeps track of all the Layers that make up the user’s artwork. For this it has multiple methods for controlling the layers, such as for adding and removing them. The layers are stored in a private mutable Buffer which can be read through Drawing.layers.

The Drawing’s paint method is what the GUI calls to get an image of the artwork. That method calls each of Drawing’s layers’ paint methods which in turn call each of their element’s paint methods. The user’s active

tool is used through the Drawing's useTool method, and the Drawing also has multiple methods for manipulating selected elements. The undo method is the one that is called when the user wants to undo a previous action.

The Drawing also has a reference to an instance of the Configurations class, which keeps track of the settings that the user has access to in the panels of the GUI. The class is immutable and does not have any methods apart from the Scala case class's copy method. The Configurations only save data, and all the manipulation of that data is done by other objects.

One of the things that Configurations saves is the elements the user has currently selected. These are mainly controlled by the Drawing class, which has many methods for them. The Drawing can select and deselect elements, make groups of the selected elements and control those group, and delete the selected elements. These actions are done through the similarly named methods.

The methods of Drawing that start with change… serve a double duty. First, they are used to change the data saved in Configurations. Second, if the user has any selected elements, those methods change the properties of those elements instead of changing Configurations.

The Element class describes different kinds of graphical components, such as rectangles and lines. Elements are immutable. They have a few methods: paint, collidesWith and move. Paint renders the Element on a canvas. CollidesWith determines whether the Element encloses a point. Move returns a new version of the element where the origin point has changed according to the parameter.

Element is an abstract class and is inherited by Shape, TextBox and Stroke. Shapes describe various kinds of geometrical shapes which are recognized by their instance of ShapeType. ShapeType is an abstract class that is inherited by Rectangle, Square, Ellipse and Circle. TextBoxes are Elements that can be written in.

Strokes are used for free brush strokes and straight lines. A Stroke has a reference to an instance of Brush which contains information about how the Stroke should be rendered. This is saved in the size and hardness values of the Brush. Each Stroke also references a Path. Path is a group of Point2D objects that inform which points of the canvas the Stroke goes through. Path takes care of connecting separate points to make a continuous line.

Elements can be grouped. An ElementGroup is also an Element itself, so actions can be performed on groups just as well as on individual elements. Specifically, a Layer saves its ElementGroups among all the other Elements. ElementGroups are immutable too, so their methods for adding and removing Elements from the group return a new group instead of changing itself. The find method returns a Some[Element] of an Element with the name given as parameter and None if such an Element is not a part of that group.

Each Layer has a collection of Elements. The Layer class has methods for adding and removing Elements in it. Among other things these methods are used by the more advanced methods related to manipulating Elements of the Layer, namely the methods update, restore, delete, and rename. The update method removes the previous version of an element and adds the new version. Restore does the reverse, that is removes the newest version and goes back to the previous one. Delete updates an element so that its value isDeleted is changed to true, which causes the Element not to be rendered on the screen. Rename updates an element and changes its name. Layer's other rename method changes the name of the Layer itself. Many of the Layer's methods are overloaded with small variations to the parameters to make calling them easier from Layer's other methods as well as other objects' methods.

The Layer's RemoveFromGroup is called by Drawing's method removeFromSelectedGroup and takes care of the specific adding and removing of elements related to the action of removing some Elements from and ElementGroup.

Objects that inherit the Tool trait are used to make new Elements (or modify existing ones in the case of Move Tool). The Tools are named based on what type of an Element they create. Their functionality is accessed through the use method of which each Tool implements its own version. The use method is given the current Drawing as a parameter, so they can read data from the Drawing's Configurations instance as well as call the Drawing's methods.

ElementHistory keeps track of elements that have been changed/created/deleted. The Tools' use methods as well as some of the Drawing's methods add the relevant elements to the history, while calling the Drawing's undo method removes the most recent collection of elements in history.

Algorithms

The only used algorithm with a name is Bresenham's line algorithm. This is used by the Path class to determine points on a straight line between two points, which is needed for drawing brush strokes. The algorithm takes as input the coordinates of the two endpoints.

Let's go over the Bresenham algorithm. Suppose we have two points with pixel coordinates (x_0, y_0) and (x_1, y_1). We want to find all pixels that fall on the straight line that goes through both points. We only move by integer steps of one pixel on either or both axes.

The differences between the starting and ending coordinates are

$\Delta x = |x_1 - x_0|$ and $\Delta y = |y_1 - y_0|$.

These tell us the total number of steps we must take in each direction. If $\Delta x > \Delta y$ then naturally we must move in the x direction more often than in the y direction. The signs of each difference Δx and Δy tell us which direction on the respective axes we must move. We name these the signs of differences x and y

$if\ (x_1 - x_0 > 0)\ sx = 1, else\ sx = -1$ and

$if\ (y_1 - y_0 > 0)\ sy = 1, else\ sy = -1$.

In the algorithm, we balance an error to help us determine which axis to move on at each point. For using the error, we name Δy as the negative of the difference in the y-coordinates: $\Delta y = -|y_1 - y_0|$. We start at the point (x, y) = (x_0, y_0) and with the error = Δx + Δy which we update on each step. We have three different options for the next point: $(x + sx, y)$, $(x, y + sy)$ and $(x + sx, y + sy)$.

To determine the next point (x, y) we compare the error with Δy and Δx. If $2 \cdot error \geq \Delta y$, we should move in the x direction and thus $x_{next} = x + sx$. Otherwise, we stay at the same x coordinate and $x_{next} = x$. If $2 \cdot error \leq \Delta x$, we should move in the y direction and get $y_{next} = y + sy$. Otherwise, $y_{next} = y$. If we moved in the x direction, we make moving in that direction at the next point less likely by adding Δy to the error. Since Δy is negative, error decreases and the condition for moving in the x direction $2 \cdot error \geq \Delta y$ becomes less likely. We do the same with y direction: if we moved in the y direction, we add Δx to the error thus making the condition $2 \cdot error \leq \Delta x$ less likely to be true. We then move to the next point (x_{next}, y_{next}) and determine the

next step similarly. Adding the oppositely signed Δy or Δx to the error whenever we move on one axis balances the error in such a way that the number of times we move on each axis corresponds to the ratio Δy/Δx which is also the slope of the line between the starting and end points. Thus, we always step in the direction that best resembles the line between the starting and end points. We have reached the end of the algorithm when (x, y) = (x_1, y_1). All the points (x, y) we have visited then make up the set of pixels on the line between (x_0, y_0) and (x_1, y_1).

Algorithms related to undoing and selecting elements might also be worth pointing out.

The program offers the user the ability to undo as many previous actions as needed. To accomplish this, each element holds a reference to the previous version of itself. The whole history of a given element is thus saved in a recursive manner in a chain of its versions. When an action is undone, the corresponding element is removed from the drawing and replaced by its previous version. That version in turn remembers its own previous version so actions can be undone indefinitely. The first version of an element does not have a previous version. At that point, undoing the creation of that element means removing that element completely.

The user can select elements by clicking at a point on the drawing. We find the element that the user clicked on by taking the topmost element that encloses the point the user clicked on. Given a coordinate point, an element use one of the following equations to determine whether it encloses that point. In the following equations, the origin point of an element is the upper left corner of that element, and coordinates increase rightwards and downwards. A point (x, y) is inside a rectangle (or a square) with the origin point (x_0, y_0) if the following equation is true:

$$x \geq x_0 \ \wedge \ x \leq x_0 + width \ \wedge \ y \geq y_0 \ \wedge \ y < y_0 + height$$

In the case of an ellipse, the equation looks like this:

$$\left(\frac{x - x_0}{\frac{1}{2}width}\right)^2 + \left(\frac{y - y_0}{\frac{1}{2}height}\right)^2 \leq 1$$

And for a circle the following must hold:

$$\sqrt{(x - x_0)^2 + (y - y_0)^2} \leq \frac{1}{2}width$$

In the case of brush strokes and text, simply a bounding box of the element is used for determining if the element encloses a point. Thus, the used equation is that of a rectangle. A group of elements is considered to enclose a point if one or more of its elements enclose it.

Data structures

The program uses Scala's native data structures. A Scala Seq is used for most collections. Options are used in methods such as Layer find(Element) where something may or may not exist. The immutability of Seqs and Options helps reasoning about the program. In fact, Element class and its subclasses such as Shape and ElementGroup are completely immutable.

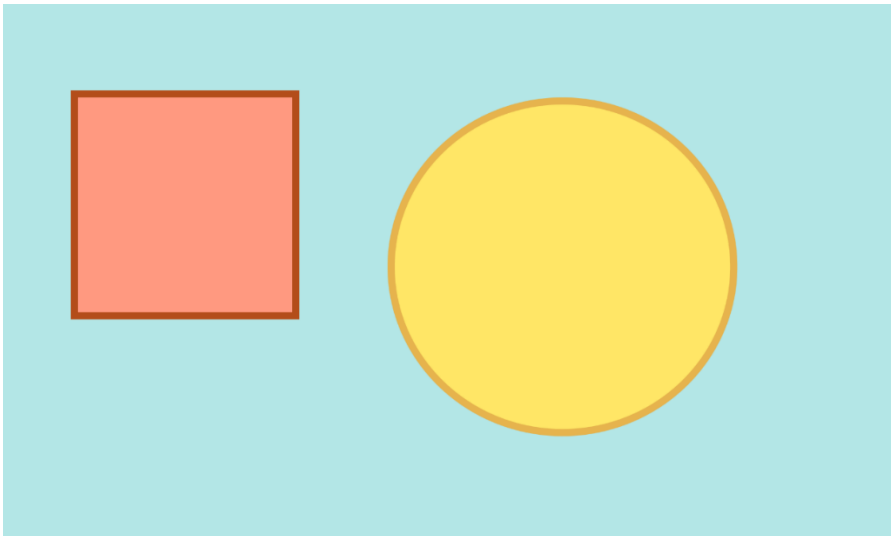
The Drawing keeps track of a varying number of Layers and each Layer in turn keeps track of multiple Elements. Both the Layers and the Elements must be kept in order so that they can be rendered correctly to the screen. Scala's Buffers were used for both collections as their mutability makes adding and removing layers/elements easy. Both Buffers are private to make sure they are only manipulated through the classes' methods.

The ElementHistory object uses a mutable Scala Stack[Seq[Element]] to save the recently created/changed/deleted elements. This is because undoing actions always starts from the very last one and the "last in, first out" principle of stacks matches this need. When an action is performed on an Element, it is pushed onto the stack. If the user wants to undo their last action, the relevant Element is popped from the top of the stack. Some methods modify multiple elements at once, so the elements are actually saved in sequences, and the full sequence handled with a single undo.

Files

The user can save a drawing to a JSON file and a drawing can be loaded from an existing JSON file. A simple example file created with the program looks like this:

```
{
  "width": 1000,
  "height": 600,
  "layers": [
    {
      "name": "Subject",
      "hidden": false,
      "elements": [
        {
          "type": "shape",
          "shapeType": "ellipse",
          "width": 385.6000061035156,
          "height": 372,
          "borderWidth": 8,
          "color": [
            230,
            179,
            77,
            1
          ],
          "fillColor": [
            255,
            230,
            102,
            1
          ],
          "useBorder": true,
          "useFill": true,
          "origin": [
            437.6000061035156,
            109.20001220703125
          ],
          "name": "Ellipse 17"
        },
        {
          "type": "shape",
          "shapeType": "square",
          "width": 249.5999755859375,
          "height": 249.5999755859375,
          "borderWidth": 8,
          "color": [
            179,
            77,
            26,
            1
          ],
          "fillColor": [
            255,
            153,
            128,
            1
          ],
          "useBorder": true,
          "useFill": true,
          "origin": [
            81.5999984741211,
            101.20001220703125
          ],
          "name": "Square 4"
        }
      ]
    },
    {
      "name": "Background",
      "hidden": false,
      "elements": [
        {
          "type": "shape",
          "shapeType": "rectangle",
          "width": 1047.200074672699,
          "height": 632.7999725341797,
          "borderWidth": 3,
          "color": [
            179,
            230,
            230,
            1
          ],
          "fillColor": [
            179,
            230,
            230,
            1
          ],
          "useBorder": true,
          "useFill": true,
          "origin": [
            -25.599977016448975,
            -11.599960327148438
          ],
          "name": "Sky"
        }
      ]
    }
  ]
}
```

More example files are provided with the source code in src/test/resources.

At the root of the file is an object which corresponds to the user's drawing. The object has the properties width, height, and layers which is a list of objects. Each object on the list corresponds to a layer in the user's drawing. A layer object has the properties name, hidden, and elements which in turn is a list of objects. Each object corresponds to an element of the drawing. The property type is used to determine whether the element is a shape, a stroke, a text box, or a group. The other properties determine what the element looks like and where it is positioned. Colours are encoded as a list of the red, green, blue values from 0–255 and the opacity value 0–1. Coordinate points are encoded as a list of numbers where the first number is the x coordinate and the second the y coordinate. An object with a type of group has itself a list of element objects.

The previousVersion property of an Element is not saved to a file and neither is the ElementHistory or any Elements that have an isDeleted value of true. This means that the user cannot undo actions that were done in a previous session.

Testing

The program was mainly tested through the GUI as I had planned. Whenever I added a new feature, I tested it in the same way a user might use that feature. I also tried to cover rarer edge cases. Whenever I found new bugs, I wrote them down or fixed them immediately. For example, when making the Rectangle Tool, I verified that all these actions worked as I wanted:

1. Drawing a rectangle by clicking, dragging towards the down-right corner, and releasing. This should draw a rectangle with one corner where the canvas was clicked and the opposite corner where the mouse was released.
2. Drawing a rectangle by dragging towards other corners. A rectangle should be drawn.
3. First dragging one way, then another and releasing. Again, a rectangle should be drawn.
4. Just clicking once on the canvas without dragging. Nothing should be drawn.
5. Dragging over the border of the canvas and releasing. A rectangle should be drawn and by moving the rectangle it should be seen that the corner opposite to the clicking point was indeed where the mouse was released.
6. Clicking, dragging and releasing outside the canvas. Nothing should be drawn.
7. Clicking outside the canvas, dragging onto the canvas and releasing. Nothing should be drawn.
8. Drawing a rectangle on top of another rectangle. A rectangle should be drawn so that the new rectangle covers the older one.
9. Drawing a rectangle on an upper layer on top of another rectangle on a lower layer. A rectangle should be drawn so that the new rectangle covers the older one.
10. Drawing a rectangle on a lower layer under another rectangle on an upper layer. A rectangle should be drawn so that the new rectangle is covered by the older one.

I tested file management by saving and loading various Drawing files. If the Drawing looks the same after loading it as look like before saving it, the file was written and read correctly. That has been the case with the drawings I have tested. I tested saving and loading drawings with different combinations of Elements of multiple types in varying layer and group structures. The program uses JSON for saving drawings, so the files produced by the program are, in principle, human-readable and human-writeable. However, the files tend to get very long rather quickly and writing them from scratch would be tedious. The format still enabled me to read the files and see that everything looked the way I had planned. I tried corrupting the files by hand and loading them as well as loading completely unrelated files to see what kind of exceptions are thrown with invalid files. I added error handling for such cases.

At the end of the project, I made tests for most of Layer's methods as well as Drawing's methods related to its Layers collection. Writing these tests helped find a few more edge cases I had not thought of before and the tests would be of help if the program was to be developed further. Admittedly, the tests would have been more useful had I written them a bit earlier. The reason I only wrote them at the end was because I first wanted to make sure I could complete all the requirements before using my time on a bunch of testing. Of course, using automated tests throughout the project might have saved some time but perhaps not quite as much time as writing them took me.

I made the tests with the ScalaTest library, and they can be found in src/test/scala. The tests are quite simple, so no Stub or Mock classes were used. The LayerTest tests most of Layer's methods and as those

methods use the Element class, so do the tests. The tests still have minimal dependence on the Element class. The only Element's method that is called in the tests is the case class copy method, and, as the Element class is immutable, it is easy to use it in a controlled manner. The DrawingTest tests those Drawing's methods that are related to controlling the Drawing's layers. The only Layer's method any of those Drawing's methods call is the Layer.rename used to change the Layer's name so there is not much dependence on Layer and no substitute class is needed. As I wrote the tests, I came up with a few special cases for the methods that I had not thought of before. I then updated the Layer and Drawing classes to pass all the tests, making the classes more fool proof.

Known bugs and missing features

There are quite a few bugs and missing features left.

Removing and adding layers cannot be undone. Fixing this would need quite a bit of refactoring as currently ElementHistory can only save changed Elements inside it. One possible solution would be to make another Stack in the history object where changed Layers are saved and a third one to determine whether an Element or a Layer was changed at each stage so that the correct Element or Layer could be fetched from the correct stack.

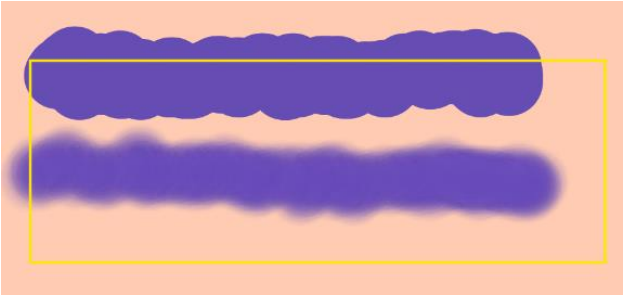
Brush strokes are selected based on their bounding box and not their visual boundaries. This is not the case with shapes that can only be selected by clicking inside their actual borders. Implementing a proper collidesWith for strokes would require a good bit of mathematics. The method would have to determine the smallest distance from the input point to any of the points on the Stroke's Path and compare that distance with the radius of the used brush. Calling this method for each stroke on the active layer whenever the Selection tool is used might prove itself rather too computationally intensive so further optimizations might still be required.

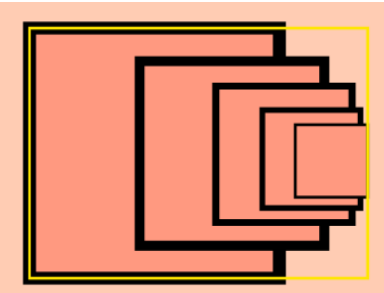
The user cannot configure the size of the drawing. This could be implemented by prompting the user through a dialog when making a new drawing.

There is no visual cue for whether the current layer is hidden or not. To only way for the user to know this is by toggling the hide button or trying to draw on the layer. A simple way to fix this would be to change the hide button's text to hide or unhide based on the visibility of the current layer.

The only visual cue for the active tool is the focus on the corresponding button which is seen as a darker background. However, this focus goes away if any other button of the interface, for example the group button, is clicked. The focus also does not change when the user changes the tool via a shortcut.

The bounding box of a selected group is not calculated quite correctly. With groups with strokes, the group does not consider how strokes are rendered with a small offset towards the upper left corner from its stored origin. With groups with shapes, the group does not take into account the width of the shapes' borders. These things are taken care of when selecting individual elements, however. Fixing this would require adding such considerations to groups, too.

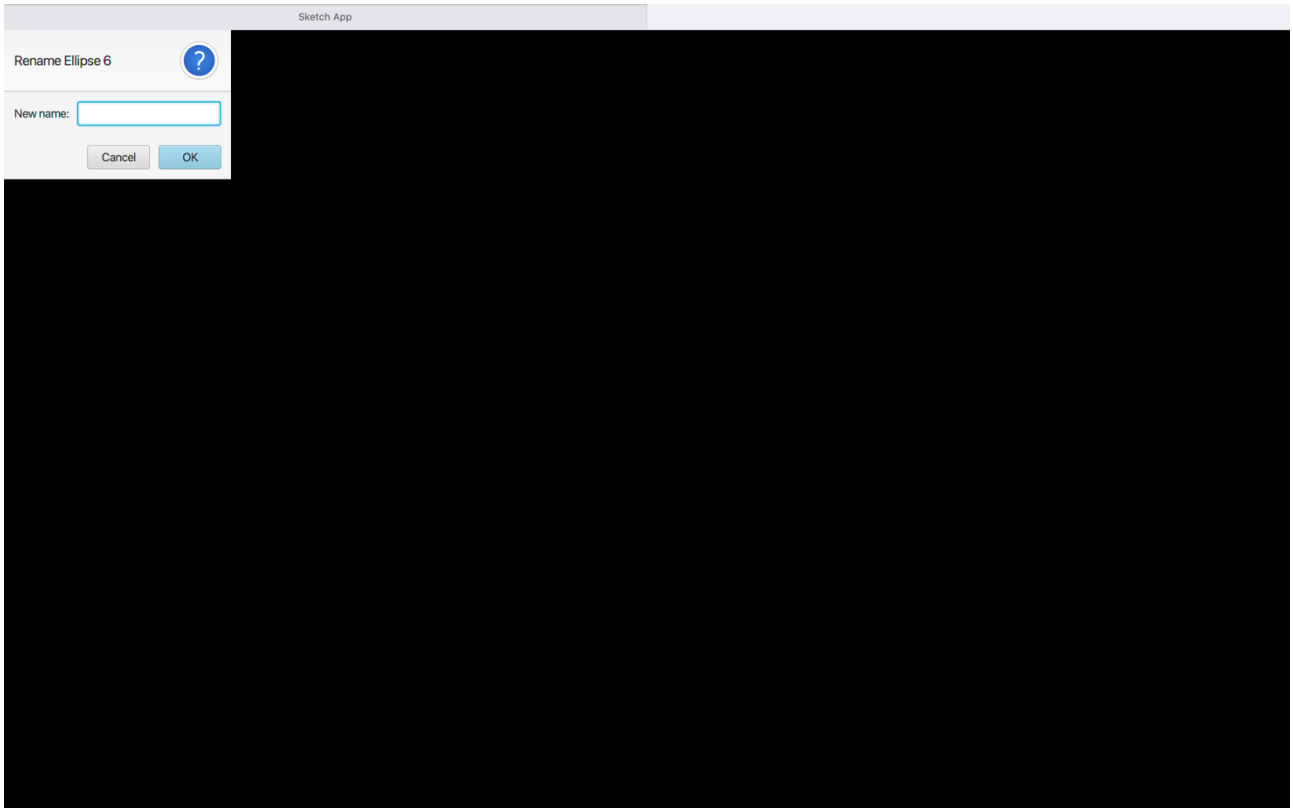




Duplicate Element names are not completely taken care of. Layer.rename ensures that an Element will not be given a name that already exists in that layer. However, it does not check if other layers of the Drawing contain that name. This might cause problems, for example, if a layer is searched for and found by checking if it contains an Element with a specific name (though this is probably not done anywhere in the program currently). The companion objects of Element's subclasses name the new elements as the Element's type and an index, e. g. "Rectangle 1", "Rectangle 2". The companion objects only decide the index based on how many such Elements they have created bu does not check if the user has named some other Element similarly. The indices also only consider this running session and are reset when the program is restarted. Again, this might cause problems with finding the right Element if it is searched for by its name. Perhaps it would be best not to even use names to uniquely identify Element but instead give each Element a unique ID that is never changed unlike the name.

The user can move an Element completely out of the canvas. This by itself could be considered a feature, not a bug, but moving the Element back on the canvas is clunky. The only way to do this is to select that Element and some other Element, then move them both by dragging that other Element. To select an Element outside the canvas, the user either must use Select All or click somewhere on the canvas and drag over the Element outside the canvas. Elements outside the canvas are not rendered, which also makes this harder.

On Mac OS in full screen mode, the dialogs are not displayed on top of the main window but as an ugly full screen tab:

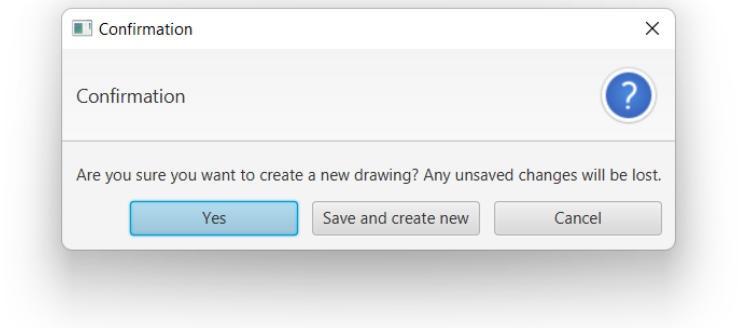


This does not happen on Windows.

Undoing changing the colours of multiple selected elements requires pressing undo multiple times even if the change was done through a single action. This would be rather easily fixed by saving the changed elements to ElementHistory together instead of separately as is done with grouping and ungrouping elements.

3 best sides and 3 weaknesses

The user interface is one of the program's strengths. Using the program is smooth and intuitive and the shortcuts reduce the amount of clicking needed. Dialogs are used for user confirmation with file management to ensure no unsaved work is lost. Graphically the user interface looks pleasing but also minimalistic enough so as not to draw the user's focus away from their artwork.



Another strength is the variety of features. The user can draw in seven different ways using the seven different tools that draw something as well as modify previously drawn things with the help of the other

two tools, Selection and Move. The user can also structure their drawing in two different ways: using groups and layers. Groups make modifying multiple elements at once easy, while layers allow the user to control the order and visibility of the drawing's elements.

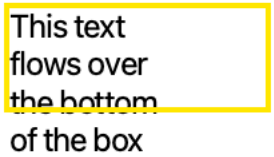
The third best side of the program is its configurability. The user can change the used brush size, brush hardness, border width, font size, and colours as well as whether to use borders and fill with shapes. Moreover, these properties can be changed on existing elements. Configuring is very easy to do with the sliders, check boxes and colour menus of the graphical user interface.

One of the weaknesses of the program is the brush strokes. The visual look of brush strokes with colours of small opacities. The brush stroke just so happens to be one of the program's most important features so it would be nice for it to look as good as possible. A stroke with a colour of small opacity looks graduated. Such colours can be selected in the custom colour menu. This is because the stroke with an opacity of say 30 % actually uses an opacity of less than 2 % when the stroke is rendered. The stroke is rendered by drawing a lot of circles with a radial gradient right next to each other. At any point multiple circles lay on top of each other and, thus, the opacity used with a single circle must be lower than the configured opacity. At very small opacities and gradients that go from that opacity to transparent, the limitations of 8-bit colours arise as there simple are not enough available opacities to make a smooth transition. These problems become increasingly apparent as strokes are drawn on top of each other.

Another weakness of the program relates to the way the Element class and its subclasses are coded. Element is an abstract class and Shape, Stroke, TextBox and ElementGroup are case classes. A case class has the very useful copy method which is used at most places where Elements are manipulated. However, copy is not a method of the Element class because it is not a case class. Therefore, we cannot use the method when we only know an object to be an Element event though all its subclasses have the method. Making Element a case class does not help since it is an abstract class and does not get a copy method because of that. We still want Element to be an abstract class though, since different subclasses implement paint and collidesWith in completely different ways. Because of these factors, we sometimes have to make separate cases for each of the Element's subclasses even if the implementations were identical. This can make understanding the code harder and coding the project more prone to error. It also makes adding new subclasses of Element more time-consuming.

```
def delete(element: Element): Element = {
  val deleted = element match {
    case e: Shape => e.copy(isDeleted = true, previousVersion = Some(e))
    case e: Stroke => e.copy(isDeleted = true, previousVersion = Some(e))
    case e: TextBox => e.copy(isDeleted = true, previousVersion = Some(e))
    case e: ElementGroup => e.copy(isDeleted = true, previousVersion = Some(e))
    case e: Element => e
  }
  this.update(deleted)
}
```

The third weakness of the program is TextBoxes. When making a text box, the user draws a rectangle to determine where the text should go. However, this box does not actually limit the text inside it very well. Horizontally, the text is squeezed to fit the box instead of, say, starting a new line near the right side of the box, adjusting the width of the box as needed or simply hiding the overflowing text. Vertically, text is not limited in any way to the boundaries of the box, but the text can go an unlimited number of lines over the bottom of the box. The box drawn by the user is still used for selecting the text box even if its text flows over it. All in all, the text boxes work rather unintuitively.



Deviations from the plan, realized process and schedule

I tried to keep track of my time usage quite accurately. According to my calculations, I used some 85 hours on the project, not including writing the plans, the checkpoints nor this document. In my technical plan I had planned it to use only 50 hours on programming and writing the document, so the hours did not really match at all. I did just about stay on my weekly schedule about when each feature should be implemented.

Pretty much all parts of the program took more time to implement than I had planned. Especially slow was programming JavaFX, which required plenty of googling at all stages of the process. The features of Element Groups also took quite much longer than I had planned. There just was a lot of small things that had to be taken care of such as keeping the Elements in the same order when they are grouped and ungrouped. Still, all the other parts of the program, too, required much more tinkering and going back and forth than I had thought it would. I do admit that I used a bit more time than necessary on some parts of the program, that is on the appearance of the GUI and the Brush Tool. Those were fun to make so I took my time. The visuals were tough work, though, as there was much to learn about FXML / Scene Builder and writing CSS for FXML was much clunkier than writing it for HTML.

I did not implement some of the methods that I had planned on making, namely scaling and rotation elements. These were not required features but would have been nice because they unlock new possibilities for drawing. Leaving them out was first and foremost a question of time. There were also some changes to the class structure along the way. Some of the larger ones include adding inheritance structure to Tools by introducing Shape and Stroke tools as super classes and storing direct references to Elements in the Element History instead of just names of those Elements to make undoing actions easier. Drawing and Configurations were made full classes instead of single objects and plenty of new methods were added, specially so to Drawing and Layer. I also made ElementGroup immutable along with the other subclasses of Element. I seem not to have planned that initially as I wrote Unit return values to ElementGroup's methods in the technical plan, though I probably did not give immutability much thought back then.

I implemented the program's functionalities more or less in the order I had planned, but there was also a lot of going back and forth between the different parts. This was often because of something old not working the way I intended with new features or because I discovered old bugs or defects while testing a new feature. Other times, specially towards the end of the project, I got ideas for tweaks and improvements to older features while developing new features. At the end, it seemed like many of the features, old and new, still had several little improvements that could be made, but making them all would hours of additional work.

Final evaluation

I implemented all the required features for demanding level, and they work quite well. The feature for working with multiple layers was not required, but I wanted to add it in anyway. There are some bugs left but they are not so major that they would ruin the user experience. Now, if I was to start building the program afresh or develop it further, there are some things I would seek to improve.

First, the class structure could be improved a little. Specifically, the Drawing class has grown rather large and multifunctional, so it might be best to broken up to smaller classes with narrower roles. For example, there could be a class that only controls the set of layers the Drawing contains.

The program's computational needs could be optimized. Currently, whenever something new is drawn, the whole drawing is rendered again. A lot of computation power could be saved by only re-rendering the newly drawn part and the parts on top of it.

Extending the program by making new tools would be rather easy, since filling the interface of the Tool trait only requires implementing the use method. Making new kinds of Element, in turn, might be rather cumbersome, since each subclass of Element requires its own case implementations whenever the case class copy method is used. Perhaps the first focus of further development should be fixing this issue. This could be done by making the program's own version of the copy method in the Element class instead of relying on the one that comes with case classes.

There could also be more throwing and catching exceptions. Currently exceptions are avoided with branching, e. g. by checking that the active layer contains the Element that a method should work on. In some cases, failing such a condition is a sign of something not working properly, and an exception should be thrown. Currently the program just ignores such cases.

In my view, the good aspects of the program outweigh the bad ones. First, the program has good capabilities for basic drawing with great configurability. The program is easy to use despite the variety of features, and the graphical user interface looks nice. Saving and opening drawings is quick and easy. Attention to detail is shown in multiple ways ranging from providing several shortcuts to using dialogs for user confirmation. I also invested a lot of time into the plans and this document. All in all, I think I did a good job with the project.

References

Oracle Help Center

https://docs.oracle.com/javase/8/javafx/api

ScalaFX

https://www.scalafx.org/

JavaFX Java GUI Tutorial – 31 – Introduction to FXML. thenewboston. Youtube.
https://www.youtube.com/watch?v=K7BOH-Ll8_g

JavaFX Java GUI Tutorial – 32 – Controllers in FXML. thenewboston. Youtube.

https://www.youtube.com/watch?v=LMdjhuYSrqg

1. How to create Modern GUI Using IntelliJ JavaFX 16 and SceneBuilder. KeepToo. Youtube

https://www.youtube.com/watch?v=VOiFmZyGAps

JavaFX Scene Builder Tutorial for Beginners. Genuine Code. Youtube.

https://www.youtube.com/watch?v=Z1W4E2d4Yxo

4.1 GridPane - Introduction [learn JavaFX]. Humiliator B. Youtube.

https://www.youtube.com/watch?v=9X4R9v2Y51s

4.2 GridPane - Exploring GridPane [learn JavaFX]. Humiliator B. Youtube.

https://www.youtube.com/watch?v=0G1J-4S5vcM

JavaFX Mouse Events Tutorial For Beginners. Kensoft PH. Youtube.

https://www.youtube.com/watch?v=-0k6tQNdViw

JavaFX and Scene Builder Beginner Course - IntelliJ #7: Add and remove items from ListView. Random code. Youtube.

https://www.youtube.com/watch?v=TDyIRhqdNqs

How to CSS a Slider in JavaFX | Java 16. Raviel. Youtube.

https://www.youtube.com/watch?v=wK1eiX4LCCw

2 easy ways to check which button got clicked – JavaFX. Ed Eden-Rump. Edencoding.

https://edencoding.com/check-whats-been-clicked/

How to use the FXMLLoader and tips for debugging it. Ed Eden-Rump. Edencoding.

https://edencoding.com/fxmlloader/

Common fonts to all versions of Windows & Mac equivalents. Alberto Martinez Peres. AMPsoft.

http://ampsoft.net/webdesign-l/WindowsMacFonts.html

Check if a point is inside, outside or on the ellipse. IshwarGupta. GeeksforGeeks.

https://www.geeksforgeeks.org/check-if-a-point-is-inside-outside-or-on-the-ellipse/

JavaFX | Alert with examples. andrew1234. GeeksforGeeks.

https://www.geeksforgeeks.org/javafx-alert-with-examples/?ref=lbp

JavaFX | TextInputDialog. andrew1234. GeeksforGeeks.

https://www.geeksforgeeks.org/javafx-textinputdialog/

Bresenham's line algorithm. Wikipedia.

https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm

Intellij .gitignore. gitignore.io, Toptal.

https://www.toptal.com/developers/gitignore/api/intellij

A sample .gitignore file for Scala/SBT projects. Alvin Alexander.

https://alvinalexander.com/source-code/scala/sample-gitignore-file-scala-sbt-intellij-eclipse/

uJson: fast, flexible and intuitive JSON for Scala. Haoyi's Programming Blog.

https://www.lihaoyi.com/post/uJsonfastflexibleandintuitiveJSONforScala.html

How to work with Files in Scala. Haoyi's Programming Blog.

https://www.lihaoyi.com/post/HowtoworkwithFilesinScala.html

µPickle 1.6.0. com-lihaoyi. GitHub.

https://com-lihaoyi.github.io/upickle/#uJson

OS-Lib. com-lihaoyi. GitHub.

https://github.com/com-lihaoyi/os-lib

Reading and writing JSON with Scala. mrpowers. MungingData.

https://mungingdata.com/scala/read-write-json/

Scala Filesystem Operations (paths, move, copy, list, delete). mrpower. MungingData.

https://mungingdata.com/scala/filesystem-paths-move-copy-list-delete-folders/

Stack Overflow (various threads by various users.)

https://stackoverflow.com/questions/60333783/button-hover-and-pressed-effect-css-javafx

https://stackoverflow.com/questions/29707882/javafx-hbox-alignment

https://stackoverflow.com/questions/31927757/paintbrush-stroke-in-javafx

https://stackoverflow.com/questions/42434769/how-to-set-default-color-for-colorpicker-in-fxml

https://stackoverflow.com/questions/11088612/javafx-select-item-in-listview

https://stackoverflow.com/questions/16977100/how-do-i-add-margin-to-a-javafx-element-using-css

https://stackoverflow.com/questions/43557722/javafx-border-radius-background-color

https://stackoverflow.com/questions/24702542/how-to-change-the-color-of-text-in-javafx-textfield

https://stackoverflow.com/questions/29962395/how-to-write-a-keylistener-for-javafx

https://stackoverflow.com/questions/37648222/how-can-i-detect-the-space-keyevent-anywhere-in-my-javafx-app

https://stackoverflow.com/questions/13726824/javafx-event-triggered-when-selecting-a-check-box

https://stackoverflow.com/questions/44061828/javafx-slider-track-length-tick-label-color

https://stackoverflow.com/questions/24158394/javafx-listview-multiple-selection

https://stackoverflow.com/questions/41449309/setting-tooltip-for-tablecolumn-in-fxml

https://stackoverflow.com/questions/32420960/javafx-how-to-make-tooltip-bigger

https://stackoverflow.com/questions/45226652/how-do-i-change-the-color-of-a-slider-track-track-in-javafx-using-css

https://stackoverflow.com/questions/63545407/javafx-remove-default-css-shadow-on-buttons

https://stackoverflow.com/questions/33454279/listview-styling-in-javafx

https://stackoverflow.com/questions/34040400/adding-children-to-javafx-control-parent-classes

https://stackoverflow.com/questions/27066484/remove-all-children-from-a-group-without-knowing-the-containing-nodes

https://stackoverflow.com/questions/43566587/styling-javafx-checkbox

https://stackoverflow.com/questions/65534317/how-do-i-make-a-fxml-controller-with-scala

https://stackoverflow.com/questions/34533539/case-class-default-apply-method

https://stackoverflow.com/questions/15641478/javafx-css-styling-listview

https://stackoverflow.com/questions/60101703/how-do-you-css-style-a-scroll-bar-in-javafx

https://stackoverflow.com/questions/30643146/javafx-dialog-doesnt-show-all-content-text

https://stackoverflow.com/questions/8309981/how-to-create-and-show-common-dialog-error-warning-confirmation-in-javafx-2