

PaintFX - Custom Paint App in JavaFX

Georgio Yammine

Abstract — This report focuses on implementing a paint alike app that allows drawing using Java and JavaFX. The application offers tools such as pencil, brush, eraser, color picker, and shapes such as rectangle, round rectangle, and ellipse while allowing the user to change the color and shape. The tool also supports history tracking, canvas zoom, canvas resize, open image from a file, and save the image to a file.



CONTENTS

Introduction and background.....	1
Tools Used.....	1
Application Design	2
Drawing on Canvas	4
Point Augmentation Method	5
Canvas Zoom and Scroll.....	6
Open and Save Image	6
History Tracking.....	7
Conclusion.....	8

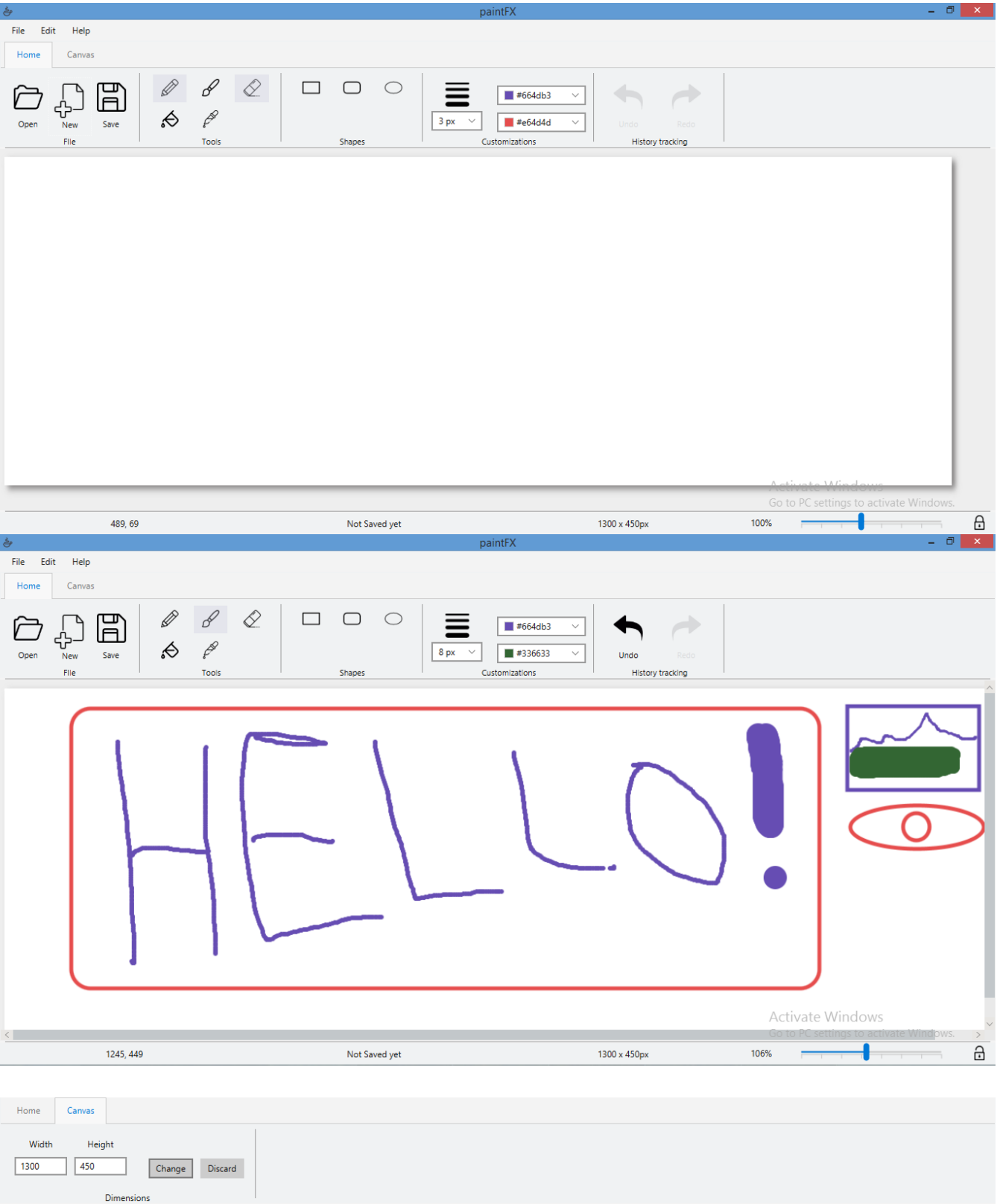
Introduction and background

JavaFx is a well know tool for building GUIs and applications. In this report, we implement a Paint alike app - PaintFX - that allows users to draw and edit images. The application offers tools such as pencil, brush, eraser, color picker, and shapes such as rectangle, round rectangle, and ellipse while allowing the user to change the color and shape. The tool also supports history tracking, canvas zoom, canvas resize, open image from a file, and save the image to a file. The application follows a fluent design while being simple and user-friendly design while being

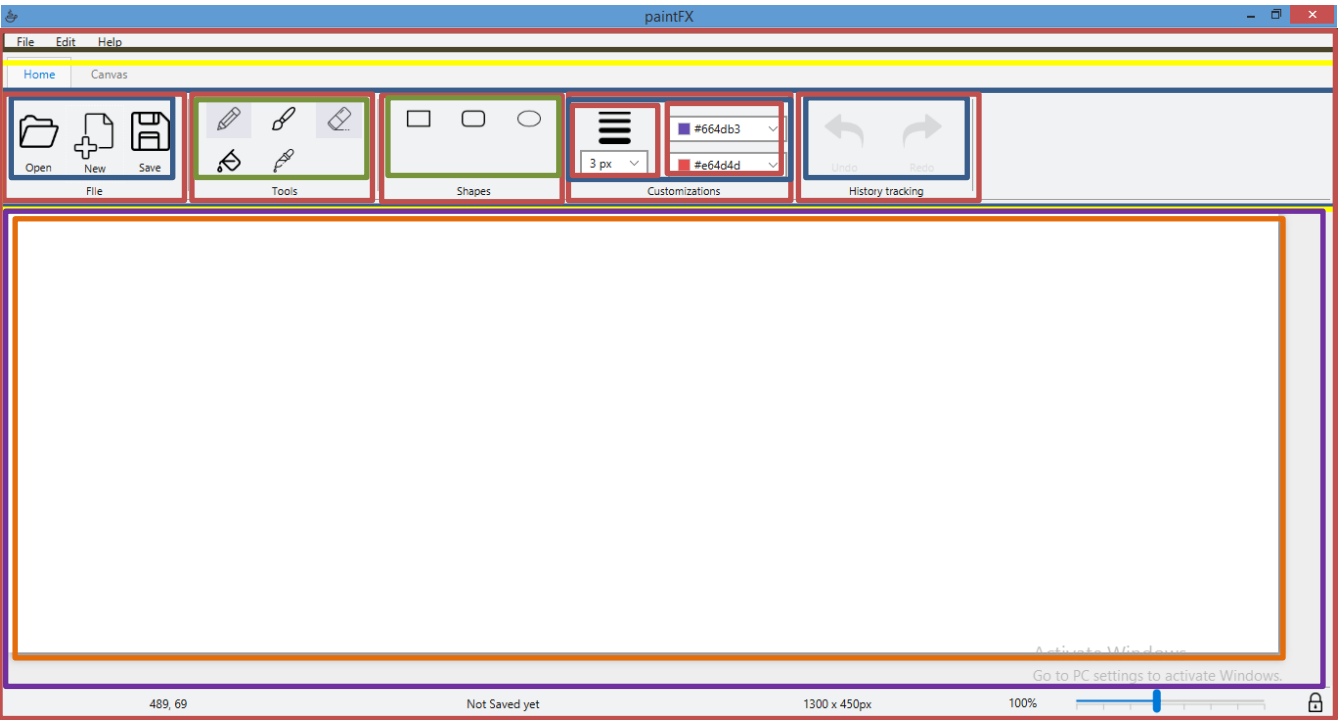
Tools Used








To implement our project we used Java, JavaFX with FXML - SceneBuilder, and Jmetro Library.

Application Design



The Application structure is highlighted in the image below



Mouse X and Y	Image Name	Image Dimensions	Zoom %, slider, snap/unsap to ticks
	HBox		
	VBox		
	GridPane		
	TabPane		
	MenuBar		
	ScollPane		
	group <- group <- [canvas and Anchor pane]		

Drawing on Canvas

Tools:

- Pen
- Brush
- Eraser
- Color Fill
- Color Picker

Shapes:

- Rectangle
- Rounded Rectangle
- Ellipse

Shape Size

Color Selector:

- Color 1
- Color 2

All the drawing logic works almost the same regardless of the tool/shape used.

1. On mouse pressed event the X and Y coordinates are stored in a prevX and prevY.
An if statement is used to check if it is a right-click or left-click to assign color 1 or color 2. A switch statement is used to detect the tool used.
Some tools have specialized extra calls such as:
 - Pencil: draw a point of width selected at X,Y
 - Brush: draw a circle of radius width at point X,Y
 - Eraser: draw a width by width white square
 - Color Picker: get color at pixel X,Y
2. On mouse dragged each tool has its own logic:
 - Pencil: draw a line of width between prevX, prevY and X,Y
 - Brush: draw circle at X,Y with radius width *along a point augmentation method*
 - Eraser: draw a width by width white square at X,Y *along a point augmentation method*
 - Rectangle/Round Rect/Ellipse: draw to the snapshot and then draw a rectangle from prevX, prevY to X,Y.
3. On mouse release save the snapshot to the undo stack.

Point Augmentation Method

For the Brush and Eraser method, we used the mouse dragged event to draw circles and squares at every point of the mouse path. However, dragging the mouse fast will result in a discrete path in which points are marginally separated and thus will not form a continuous line.

To solve this issue, we had to come up with a points augmentation method which will draw extra circles/squares at extra points to ensure that the line is continuous.

The issue and fix can be seen in the images below

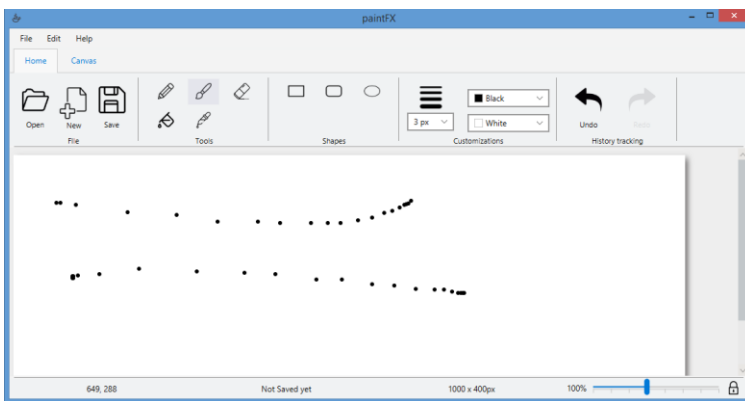


Figure 1: Without Points Augmentation

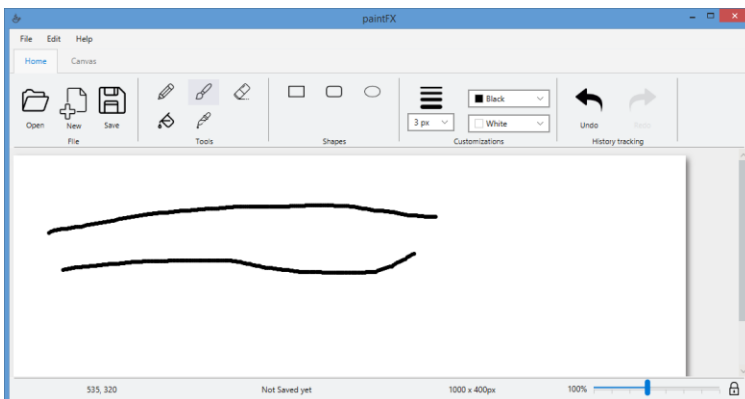


Figure 2: With Points Augmentation

The pseudocode is below:

```

1. private void PointAugmentation(prevX, prevY, x, y) {
2.     if (abs(x - prevX) > ε || abs(y - prevY) > ε )
3.         midx = (int) ((x + prevX) / 2);
4.         midy = (int) ((y + prevY) / 2);
5.
6.         drawAtPoint(midx, midY);
7.
8.         PointAugmentation(prevX, prevY, midx, midy);
9.         PointAugmentation(midx, midy, x, y);
10    }
11.}

```

Canvas Zoom and Scroll

To achieve Scroll we used a scroll bar as the parent container. The scroll bar will contain the anchor pane and the canvas. However, for the scroll bar to automatically resize to the canvas, we had to put the canvas and anchor pane inside a group. However, while applying the zoom on both the canvas and white background anchor pane, the zoomed panes did not scale accordingly, and each moved in a different direction. To solve this issue, we placed both the canvas and anchorpane in another group and scaled the group instead.

Open and Save Image

To save and open an image we used ImageIO and SwingFXUtils. In saving an image as a jpeg format, the ImageIO library had a bug that misinterprets colors and thus results in a wrongly colored image. To fix the issue, we first stored the Image in a Graphics2D object and then save it using ImageIO to fix the issue.

History Tracking

History Tracking has been implemented in many applications. The traditional way of implementing History tracking is using stacks and keeping track of the changes in the current path. However, any new move will discard previously possible redo operations. Advanced history tracking systems keep track of the event in a Tree format and thus allow the user to navigate the history in any possible way and does not disregard an old path.

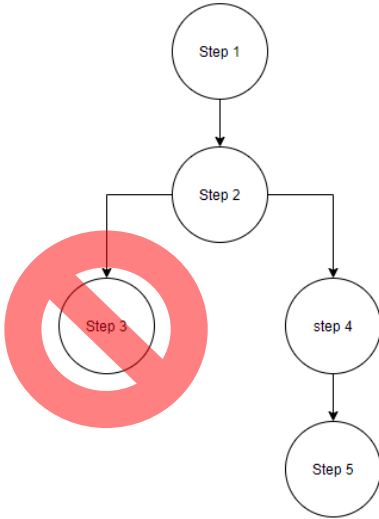


Figure 3: Traditional Simple Undo Redo System

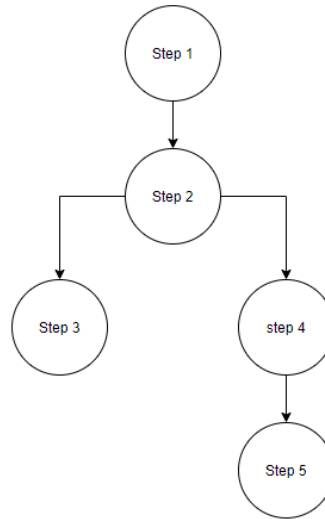


Figure 4: Advanced Tree Undo Redo System

In our app, we followed the simplistic version where we only keep track of 1 path and thus resetting the Redo Stack after doing a new operation.

Our implementation keeps track of 2 stacks:

1. Undo Stack
2. Redo Stack

The stack will store snapshots of the canvas at different times. These will be used to go back to the previous state by drawing the image to the canvas using the Graphics Context.

To save a state, we checked on mouse released event and checked if there was a change to the canvas, and if so pushed that state to the stack and clear the Redo Stack.

On Undo, we popped and pushed the current state to the Redo Stack and drew the previous snapshot to the canvas.

On Redo, we popped the last snapshot from the stack and drew it to the canvas while adding it to the Undo Stack.

Conclusion

In this project, we demonstrated the power of JavaFX to build desktop apps. We built a very powerful drawing and editing app even rivaling MS Paint. We implemented a fluent design, lots of drawing tools such as a pencil, brush, and shapes, and a history management system.