**Lebanese American University**



**COE Application**

**COE 593 – Section 31**

*Dr. Maria Abi Saad*

**Project 1 – Painter Using JavaFX**

*Edwin Odeimi 201601044*

*November 11th, 2020*

Contents

[Figures 3](#_Toc56025376)

[Introduction 4](#_Toc56025377)

[Scene Graph 4](#_Toc56025378)

[Code Explanation 5](#_Toc56025379)

[Constants & Variables Declarations 5](#_Toc56025380)

[Initialization 5](#_Toc56025381)

[UI Interaction Events 5](#_Toc56025382)

[IO Input Events (Mouse/Keyboard) 6](#_Toc56025383)

[Helper Class and Method 6](#_Toc56025384)

[Conclusion 6](#_Toc56025385)

# Figures

[Figure 1 - Scene Graph (Made in MS-Paint) 4](file:///C:\Users\edwin\Desktop\COE%20Application\Project\COE%20Application%20-%20Project%231_Painter.docx#_Toc56025386)

[Figure 2 - Window visualization 4](file:///C:\Users\edwin\Desktop\COE%20Application\Project\COE%20Application%20-%20Project%231_Painter.docx#_Toc56025387)

[Figure 3 - Painter Demo (The black hollow circle in the canvas is the cursor) 6](file:///C:\Users\edwin\Desktop\COE%20Application\Project\COE%20Application%20-%20Project%231_Painter.docx#_Toc56025388)

# Introduction

In this project, I am required to develop a simple application that presents writing tools and a canvas for a quick writing or drawing session. It should be developed using JavaFX and optionally SceneBuilder. I decided to go with SceneBuilder instead of writing my own FXML file and constantly running and debugging, due to the semester load (I am taking 18 credits), and I wouldn’t of had time to accomplish everything I did in time without this tool.

# Scene Graph

The first thing I did was to split the stage using a SplitPane (with a light gray background color) as a root to my graph, and manually setting the divider to allow a width of 200 exactly for the tools panel on the left. As for the right AnchorPane, I appended a simple empty Pane (with a white background color) which will be used as a canvas for the painter project. This Pane will be resized programmatically following the user input in the left panel. As for the left tools panel, a wrapping GridPane (only rows; I didn’t use VBox to allow independent resizing flexibility in SceneBuilder) is used with its children being two TiledPanes for “Resolution” and “Tools Settings”, and a VBox containing both the “Undo” and “Clear” buttons. All these layouts and nodes’ widths and heights are fixed, except for an empty row in the GridPane which helps with resizing the window (it works without the empty row, but it is nicer that way):

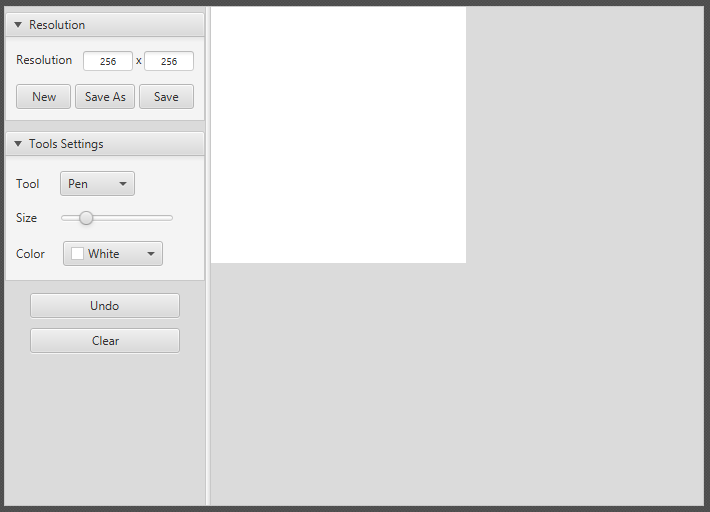
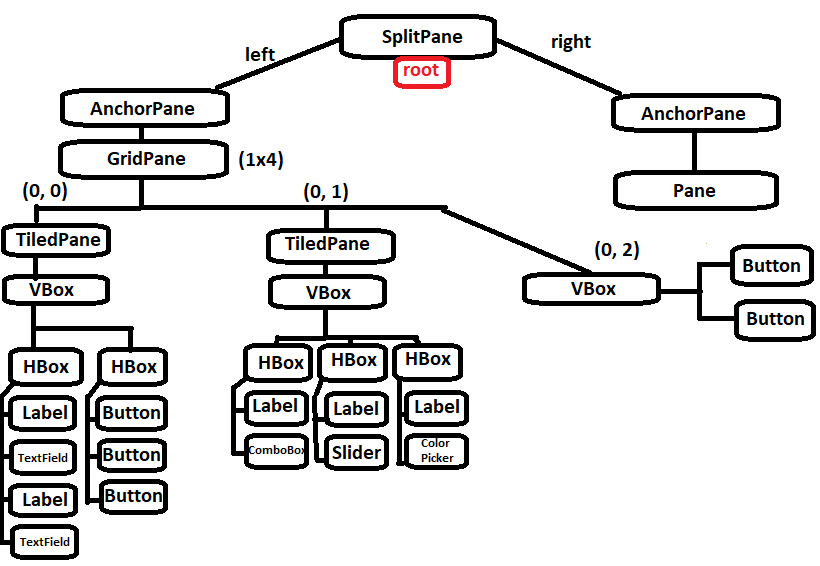
* As for the top TiledPane, labeled “Resolution”, I added a VBox to have an HBox each. The top HBox contains the following nodes: Label (value = “Resolution”) TextField (widthDefaultValue=256) Label (value=”x”) Texfield (heightDefaultValue=256). The bottom HBox contains three buttons: “New” button which creates a blank canvas with the specified resolution, “Save As”, and “Save” buttons (allowing “jpg/jpeg” and “png”).
* The bottom TiledPane, labeled “Tools Settings”, contains (with a similar layout with respect to labels and input fields) a ComboBox allowing a “Pen” or “Eraser” to be selected (the tool can also be quickly chosen using “P” key for “Pen”, and “E” key for “Eraser”), a slider for the tool size (the tool size can also be changed dynamically with the scroll wheel, while the mouse is inside the canvas), and ColorPicker to select a color for the “Pen” tool.
* Finally, the last, non-empty GridPane row contains the “Undo” and the “Clear” buttons.

Figure 1 - Scene Graph (Made in MS-Paint)

Figure 2 - Window visualization

# Code Explanation

The code can be divided into five sections:

## Constants & Variables Declarations

Defined and instantiated all the useful variables, as well as the constants, e.g.:

* All nodes needed from the FXML file.
* Constant boundaries and limits.
* Initial variable conditions.

## Initialization

* Filled and initialized the ComboBox.
* Set TextFields default values.
* Initialize ColorPicker and Slider.
* Initialize canvas:
  + Set initial default dimensions.
  + Create and apply rect mask to avoid spillage.
  + Initialize undoList.

## UI Interaction Events

* New
  + Validate TextFields.
  + Resize canvas.
  + Re-initialize canvas.
* SaveAs
  + FileChooser to get Path.
  + Take canvas snapshot.
  + Save snapshot.
  + Update path in memory.
* Save
  + Check if SaveAs needed.
  + Take canvas snapshot.
  + Save snapshot in path saved in memory.
* SelectTool
* Undo
  + Check if undo or redo (re-add after erase) needed.
  + Undo/Redo.
  + Update undoList.
* Clear
  + Re-initialize TextFields
  + Call “New”
* UseTool
  + Check tool.
  + If Draw, create Line shape as needed and update undoList.
  + If Erase, check intersecting shapes with mouse and update all lists accordingly.

## IO Input Events (Mouse/Keyboard)

* QuickSelectTool
  + If Key pressed “E”, select Eraser tool.
  + If Key pressed “P”, select Pen tool.
* QuickResizeTool
  + Check if mouse is over the canvas.
  + Calculate scroll wheel delta and evaluate the sign to increment/decrement the tool size.
* InitialMouseCanvasClick
  + Add initial Circle shape for precision.
  + Set-up undoList for next move.
* FinalMouseCanvasClick
* CanvasHover
  + Add hollow Circle as cursor when in the canvas (radius = tool size).
* CanvasHoverEnter
* CanvasHoverExit
  + Remove cursor.

## Helper Class and Method

* isInsideCanvas is a helper method to check if (x, y) point is inside the canvas (xcanvas, ycanvas, width, height).
* UndoElement is a helper class that holds the ArrayList of shapes from the move, and a boolean indicating if the move was a draw or erase.

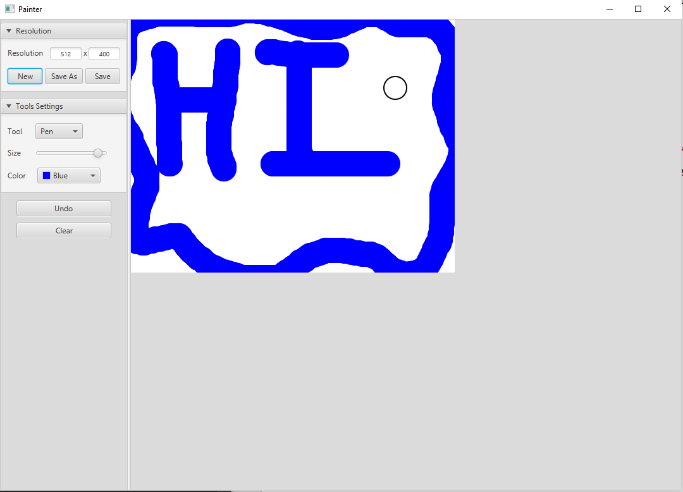


Figure 3 - Painter Demo (The black hollow circle in the canvas is the cursor)

# Conclusion

Throughout this project I learned a lot about SceneBuilder, and how much it helps make our lives easier, but my main take-away is surprisingly the design. I have always been bad at design, but I learned a lot while developing the Painter application, one example is the effective use of cascaded layouts.