# CMPT 381 Assignment 3: 2D Graphics, MVC, Multiple Views

Due: **Sunday**, Nov. 14, 11:59pm

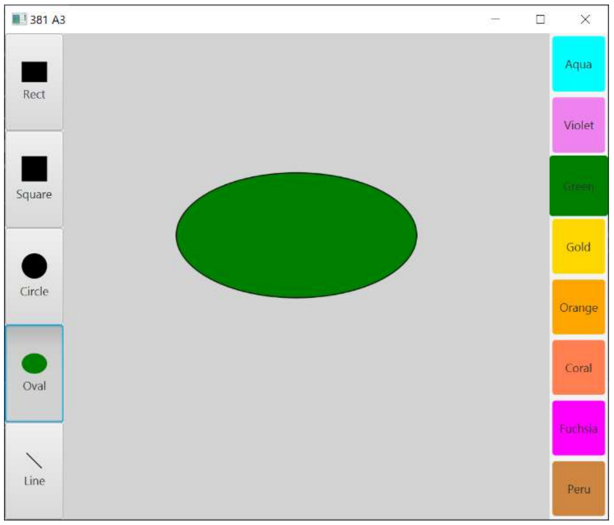
## Overview

In this assignment you will build a JavaFX drawing program that demonstrates your skills with immediate-mode graphics, interaction with graphics, more complex model-view-controller architectures, and multiple synchronized visual representations of the model. Part 1 covers the basic system and interactive creation of drawing shapes, Part 2 covers multiple views, Part 3 covers user interactions with the system, and Part 4 covers view movement and synchronization.

The drawing application allows users to select a shape tool from a tool palette and a colour from a colour palette, and interactively draw shapes on the drawing surface by dragging the mouse. There is a main surface and a miniature view (which shows the entire document), and both can be used for drawing, selection, and navigation (although they show the drawing in slightly different ways). The user can select shapes by clicking on them with the mouse, and selected shapes show their bounding box and a single resize handle at the bottom-right corner of the bounding box. Users can click and drag on a shape to move it, and can drag the resize handle to resize the shape. If the user presses the Delete key and there is a selected shape, the shape is deleted. The user can pan (i.e., scroll) the main drawing surface by pressing the right mouse button and dragging. The user can also pan by dragging the viewfinder rectangle in the miniature view (this rectangle shows the location and extents of the main drawing surface within the overall document).

You will develop this application incrementally through the four parts described below.

## Part 1: A Basic Drawing Application



Using a full MVC architecture, develop a JavaFX application that implements basic drawing capability as shown in the picture →. The app has a shape toolbar on the left side of the window, a drawing surface in the middle, and a colour toolbar on the right.

Write the following classes (you may also write other classes as needed):

- Application class
  - DrawingApp: the main application class
- Model classes
  - DrawingModel: the model that stores all elements of the drawing
  - XShape: the abstract supertype of all drawing shapes
  - XRectangle, XSquare, XOval, XCircle, XLine: classes to represent different shapes
- InteractionModel classes
  - InteractionModel: the interaction model that stores all elements related to view state
- View classes
  - MainUI: a view that contains and lays out the shape toolbar, the drawing surface, and the colour toolbar
  - ShapeToolbar: a view that contains buttons for showing and selecting a shape tool for drawing
  - ColourToolbar: a view that contains buttons for showing and selecting a colour for drawing
  - DrawingView: a view that contains a canvas to show the drawing and allow user interaction
- Controller classes
  - DrawingController: the controller to handle events from the view classes

*Interaction requirements:*

- A default shape tool and colour are selected on system startup

- The user can click on any of the shape tools or colours to select them for drawing; selected buttons in both toolbars show the selection using visual means (e.g., shading, outline, or colouring)
- The current colour is also shown in the selected shape button
- When the user presses the left mouse button and drags on the drawing surface, the chosen shape is created in the chosen colour; as the user drags, the shape is resized until the user releases the mouse button
- The initial location of the user's press is fixed during creation, but the other point (at the user's cursor location) can move anywhere, and the shape should respond appropriately (see video demo)
- Shapes are filled with the chosen colour and have a black outline (note that once a shape is created its colour does not change)

*Software requirements:*

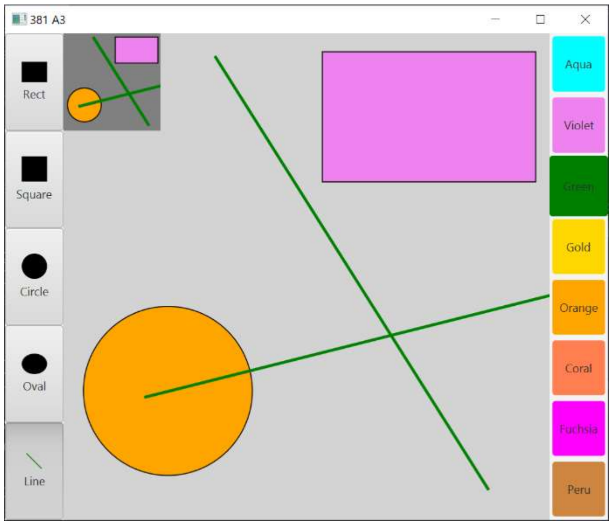
- Your model class must create and manage objects of type XShape (and subclasses) based on the user's actions on the drawing surface
- Your shape classes cannot use any existing JavaFX shape or geometry classes (e.g., you cannot use a Rectangle to implement an XRectangle)
- Your shape classes should store all position information as normalized coordinates as discussed in lectures
- Your drawing view should have an initial size of 500x500, and should take extra space when the window resizes (the two toolbars should fill their vertical space)
- Your system must use an MVC architecture with publish-subscribe communication between models and views
- Your controller should implement methods for handling user events from all of your views
- Your controller must use a state machine (as described in lectures) to handle mouse events from the drawing view
- Your interaction model must keep track of the current shape and the current colour

Resources:

- The BoxDemo code presented in lectures
- The JavaFX API documents for Canvas and GraphicsContext (https://openjfx.io/javadoc/17/javafx.graphics/javafx/scene/canvas/package-summary.html)

## Part 2: Adding a Second View

Create a second view that shows the drawing in miniature, and place the mini view in the top-left corner of your main drawing view. The miniature should show all elements of the drawing and should support the same interactions that are possible with the main drawing view. The app with the miniature should look like →



*Interaction requirements:*

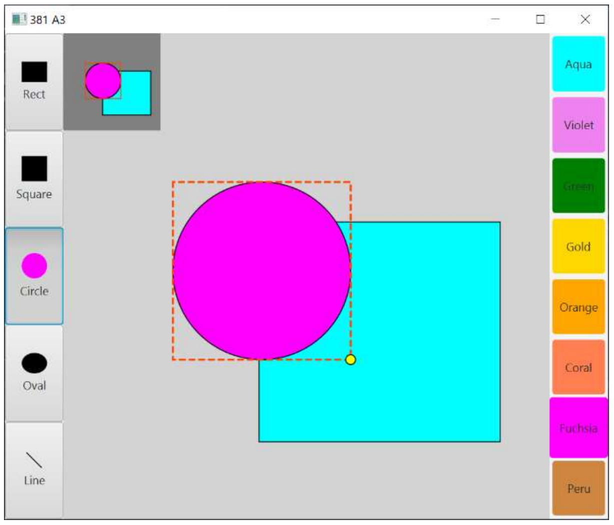
- The user should be able to create shapes in the miniature view with the same interactions that are used for the main drawing view (i.e., left-press and drag to interactively create the selected shape with the selected colour)
- The miniature should have a different background colour so that it can be clearly seen on the main drawing view

*Software requirements:*

- Create a subclass of your DrawingView class called MiniDrawingView to implement the miniature. Re-use as much code as possible from the parent class
- The miniature view should be 100x100 in size, and should show the entire extents of the drawing
- The miniature should be placed at the top-left corner of the main drawing view
    - You can use a StackPane (or a class that extends it) to put one view on top of another; see the *setAlignment()* method of StackPane to control where views are stacked

- You can use the same controller to handle mouse events from the miniature view: if you have normalized the coordinates that you pass to the controller, the same code should work regardless of the size of the view

## Part 3: User Interactions



Add support for several types of interaction with the shapes on the drawing surface: selection, moving, resizing, and deletion.

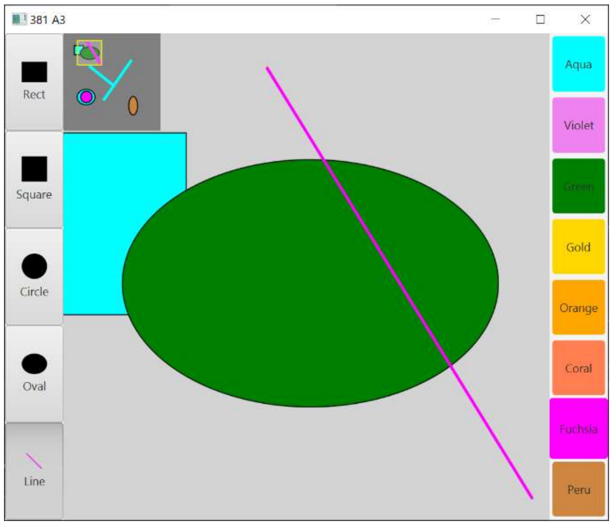
*Interaction requirements:*

- *Selection:*
- When the user presses the left mouse button with the cursor on a shape (in either the main or miniature view), the shape moves above any other shapes and becomes selected
- In the main drawing view, a selected rectangle, square, circle, or oval shows its bounding box as a dashed red rectangle; for lines, the system draws a dashed red line overtop the line
- In the miniature, selected objects are shown in the same way but with solid lines rather than dashed lines
- Pressing the mouse on a different shape selects that shape and unselects any previous selection
- Pressing and releasing the mouse on the background (without moving) unselects any previous selection
- *Moving:*
- If the user clicks on a shape and drags, the shape moves along with the mouse cursor until the user releases the mouse button
- *Resizing:*
- In the main drawing view (but *not* the miniature), the selected shape also shows a yellow resize handle at the bottom-right corner of the bounding box (or for lines, at the end of the line). Dragging this handle with the mouse resizes the shape, similarly to how interactive creation worked above. Note that the resize handle is always shown at bottom right of the bounding box regardless of where the user drags.
- *Deletion:*
- If a shape is selected and the user presses the Delete key, the shape is deleted.

*Software requirements:*

- Add code to your controller to implement selection, moving, resizing, and deletion. You will need new states in your state machine, and you will need to add an onKeyPressed event handler to your Scene object in the application class
- Add code to your model to implement *contains()* methods that indicate whether a shape contains a given point
- Add code to your model to implement Z-order (to bring selected objects to the top) as discussed in lectures
- Add code to your interaction model to store a reference to the selected shape
- Add code to your view classes to show the bounding box and resize handle for the selected shape

## Part 4: View Panning and Synchronization



Increase the size of the 'document' that your drawing app looks onto; this means that the main drawing view is now a viewport into this larger world. The miniature view still shows the entire document, but now shows a 'viewfinder' rectangle that shows the size and extent of the main drawing view in the context of the whole document. Add panning capability in the main view (dragging using the right mouse button) and in the miniature view (by dragging the viewfinder rectangle). The finished system should look something like this (note the yellow viewfinder rectangle in the miniature view) →

*Interaction requirements:*

- The size of the document (the 'world') should be 2000x2000; the initial size of the main drawing view is still 500x500, and the size of the miniature view is still 100x100
- The miniature view shows the entire 2000x2000 document
- The miniature shows a yellow rectangle indicating the size and extents of the main drawing view within the document
- Right-clicking and dragging in the main drawing view pans the view (the document should move in the same direction as the cursor, similar to touch-screen scrolling); the viewfinder rectangle in the miniature updates as the main drawing view pans
- The viewport cannot go past the extents of the document
- Left-clicking and dragging the viewfinder rectangle in the miniature view also pans the view (this means that the user is not able to select objects underneath the viewfinder rectangle)

*Software requirements:*

- Change your view constructors to take in and record the document size in the view
- Change the normalization of your mouse coordinates in the view to use the document size instead of the canvas size
- Create a subclass of DrawingController for your miniature view, called MiniDrawingController (with the viewfinder interaction, the miniature view now needs a separate controller). Again, reuse as much code as possible from the parent.
- Add code to MiniDrawingController to handle dragging of the viewfinder
- Store information about the view location (i.e., viewLeft and viewTop) in your interaction model, and use this information as an offset to correctly locate mouse locations in the document (see lectures for details)

## What to hand in (each student will hand in an assignment)

- Create a zip file of your JavaFX/IDEA project zip (File → Export → Project to Zip file…).
- Add a readme.txt file to the zip that describes the organization of your handin and provides any comments to the markers about what to look at in your code (as always, systems for 381 should never require the marker to install external libraries). If parts of the system don't work, but you want us to look at the code for these parts, please explain in your readme.txt

## Where to hand in

Hand in your zip file to the Assignment 3 link on the course Canvas site.

## Evaluation

- Part 1: all classes are implemented as specified, all interaction requirements are working, and all software requirements are implemented as specified
- Part 2: the miniature view is implemented as specified, and satisfied all interaction and software requirements
- Part 3: all required interactions are implemented as specified and work as expected for the user
- Part 4: view panning and synchronization are correctly implemented, including both the interaction and software requirements
- Overall, your system should run correctly without errors, and your code should clearly demonstrate that you have satisfied the software requirements stated in the assignment description. (Document your code to assist the markers understand how you are satisfying the software requirements)

If parts of your system have only partial implementations (e.g., a feature does not work but has been partially developed), clearly indicate this in your readme.txt file. Code should be appropriately documented and tested (although documentation will not be explicitly marked). No late assignments will be allowed, and no extensions will be given, without medical reasons.