Image Filter Project

Created by Luke Estes, Trent Pannkuk, and Isaiah Suarez
UT OnRamps Computer Science

The project and its documentation can also be accessed on GitHub at:
https://github.com/Kingofkode/image-filter-project/blob/master/README.md

Table of Contents

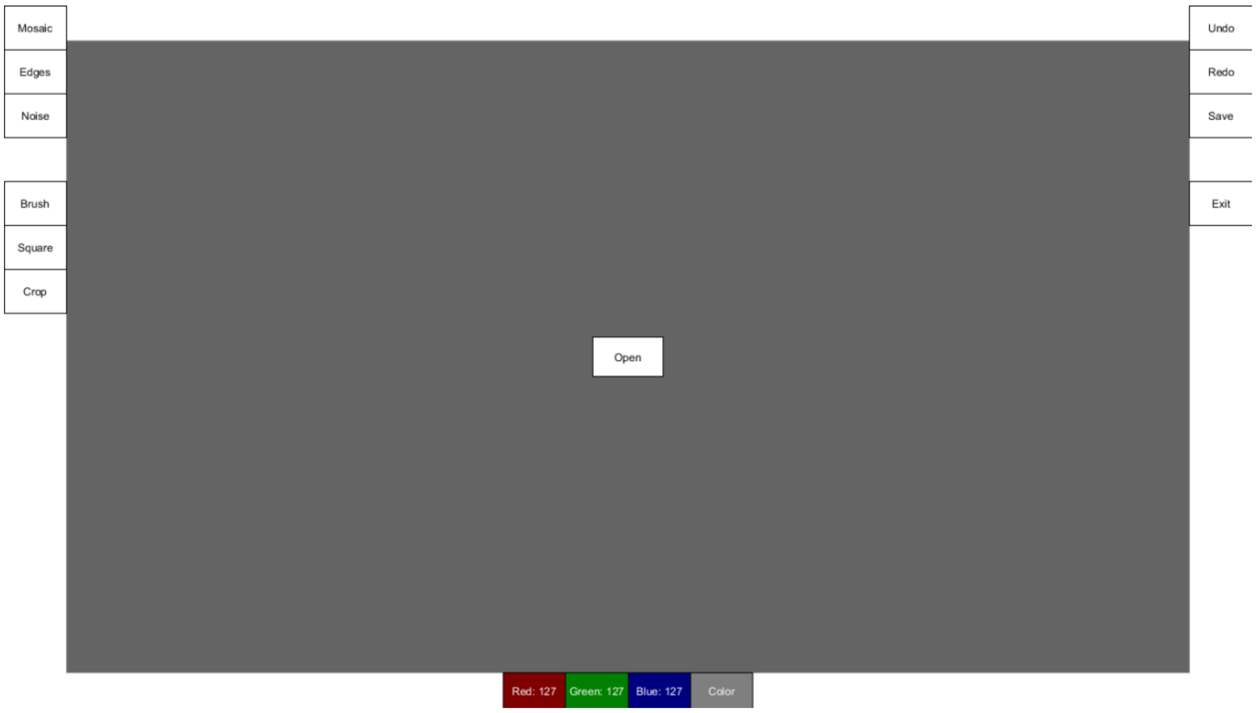
1. Installation 2
2. How to Use the Program 2
3. Getting Started with the UI Library 4
    i. Views 4
        a. Creating a View 4
    ii. Mouse Event Handling 5
    iii. Button 6
    iv. ImageView 7
        a. Applying Filters to ImageView 8
4. Tools 9
    i. Undo and Redo Buttons 9
    ii. Save Button 9
    iii. Open Button 10
    iv. Brush Toggle and Colors 10
    v. Square Button 10
    vi. Crop Button 10
5. Rubric Items Met 10
6. Reflection 12
7. Resources 13

Installation

1. Download the master branch
2. Open image-filter-project-master/Main/Main.pde

How to Use the Program

When the program is initially launched, the following window should appear:



To begin working, open an image and select an image file from the dialogue box that appears.

The left toolbar contain buttons for the three filters (see below for more). To apply any of the filters, simply click the appropriate button. The buttons for the brush, square fill, and crop tools are also on this side. To use the brush, click the button and drag the cursor across the screen. (Note: The line may be drawn from the center of the brush, not the tip.) To use the square fill or the crop tool, click the button, then select opposite corners of the rectangular region to be edited. Do NOT click and drag; this will not be enough by itself to complete the action.

The right toolbar contains general program functions, including the open button once an initial image has been selected. These functions are all applied by simply clicking the respective buttons.

The bottom toolbar contains a color selection tool for the brush and square tools. The three boxes representing the red, green, and blue values can be tweaked by clicking on them with a mouse. Right-click them to increase the values (up to 255), and left-click to decrease the values (down to 0). The last box mixes the RGB values to preview what the tools will use.

Getting Started with the UI Library

Views

Views are the fundamental building blocks the user interface. Because view objects are the main way the application interacts with the user, they have a number of responsibilities. Here are just a few:

- Views draw content in their rectangular area
- Layout and child view management
- Views may contain zero or more child views.
- Views can adjust the size and position of their child views.

Views can be nested inside other views to create view hierarchies, which offer a convenient way to organize related content. Nesting a view creates a parent-child relationship between the child view being nested and the parent. A parent view may contain any number of child views but each child view has only one parent view. When a child view’s visible area extends outside of the bounds of its parent view, no clipping of the child view's content occurs. The geometry of each view is defined by its `xPos`, `yPos`, `viewWidth`, and `viewHeight` properties.

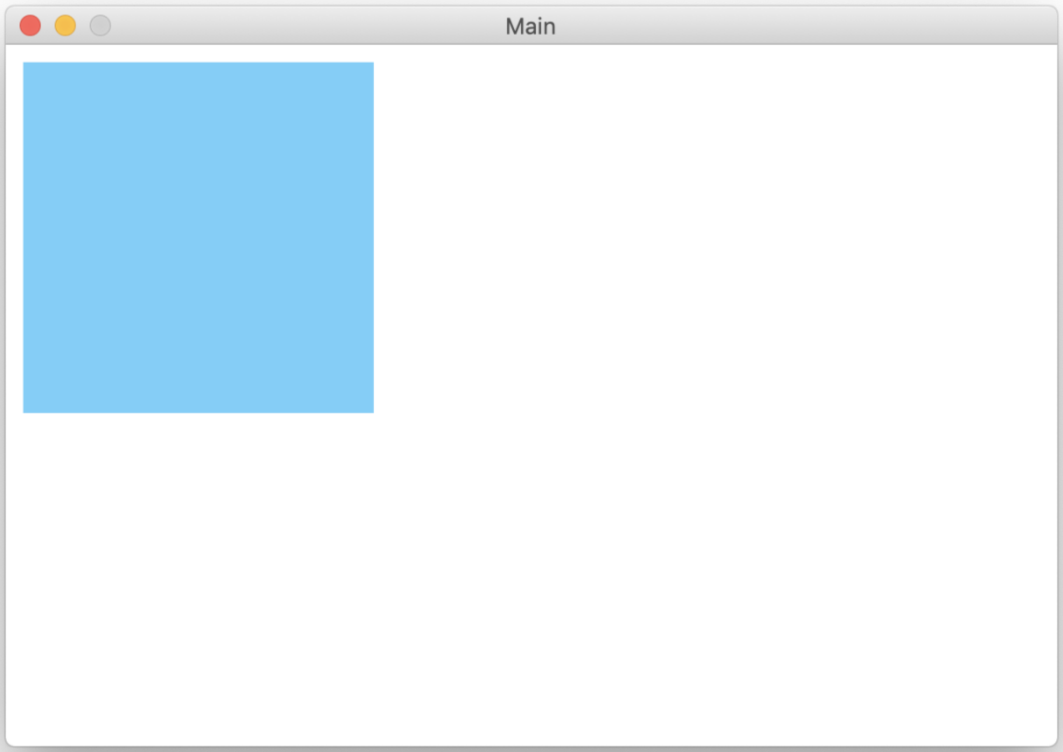
Creating a view

The following example creates a 200 x 200 blue view and places its top-left corner at the point (10, 10) in the parent view's coordinate system (once it is added to that parent view). The color was changed to blue in this example. The full list of View's customizable properties can be found here.

```
// Initializes new View object
final View blueView = new View(10, 10, 200, 200);

// Sets background color to blue
blueView.viewColor = color(133, 205, 246);

// Adds blueView to mainView so it is rendered
mainView.addChildView(blueView);
```

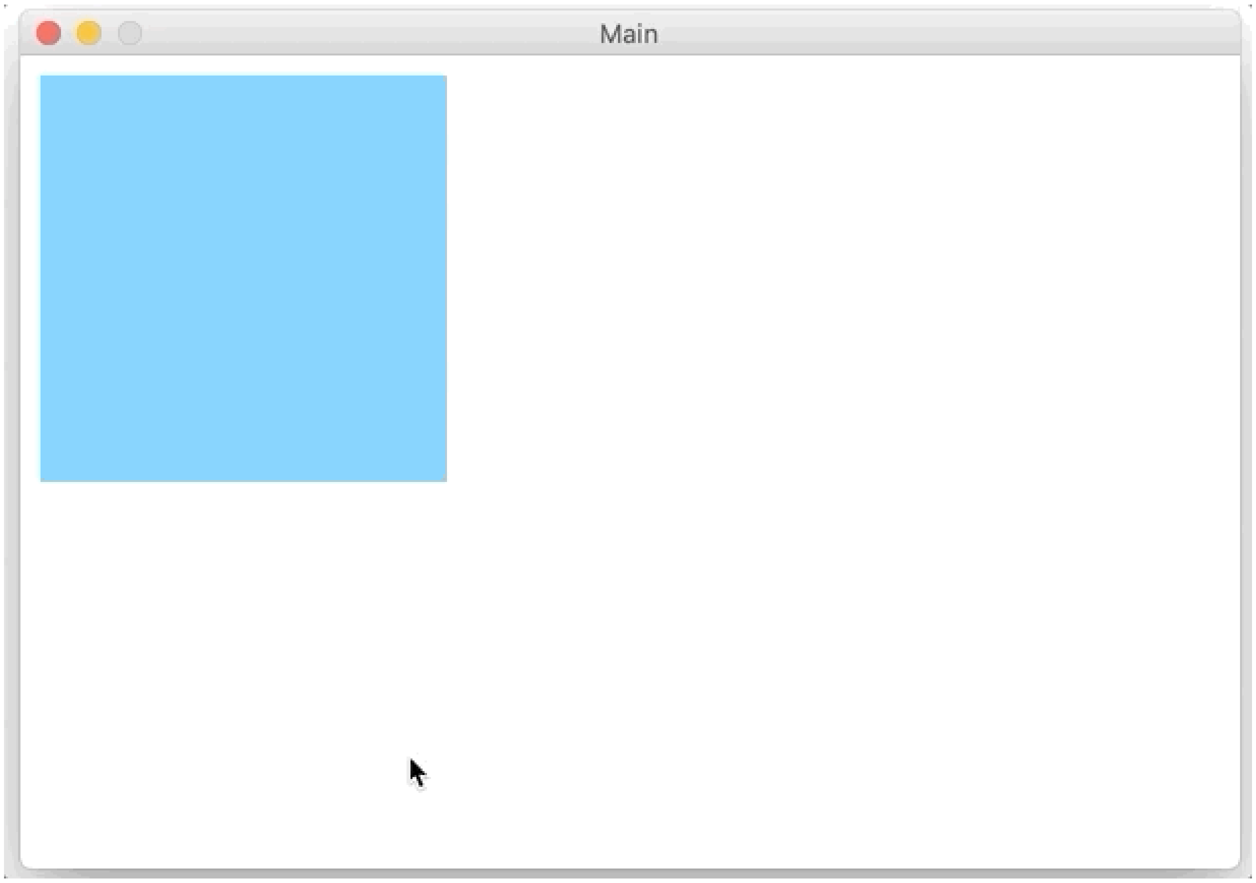


Mouse event handling

The following example modifies the view width when the view is clicked on.

```
blueView.responder = new MouseResponder() {
   public void isClicked() {
   // Increases view width to 400 when it is clicked on
     blueView.viewWidth = 400;
   }

   // All 3 methods must be present even if they are not used
   public void isHovering() {}
   public void buttonDown(Mouse button) {}
 };
```



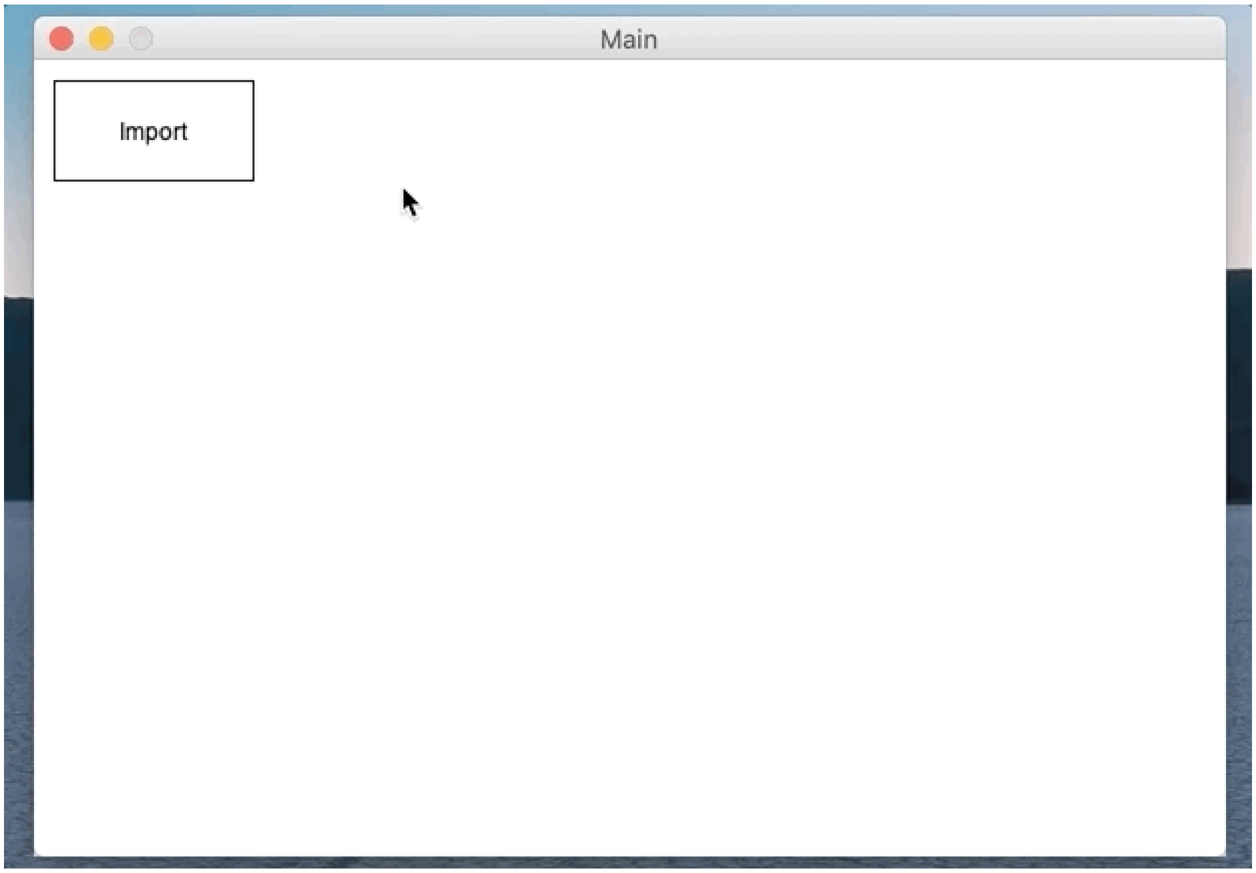
Button

A button inherits all of the properties of a view with some added functionality. For example, when you hover over it, the button highlights. The following example creates a 100 x 50 button with the title "Import" and places its top-left corner at the point (10, 10) in the parent view's coordinate system. The `highlightedViewColor` was changed to blue in this example. A full list of Button's customizable attributes can be found here.

```
// Initializes new Button object
final Button myButton = new Button("Import", 10, 10, 100, 50);

// Changes the highlight color to blue
myButton.highlightedViewColor = color(35, 130, 242);

// Adds it to mainView so it is rendered
mainView.addChildView(myButton);
```

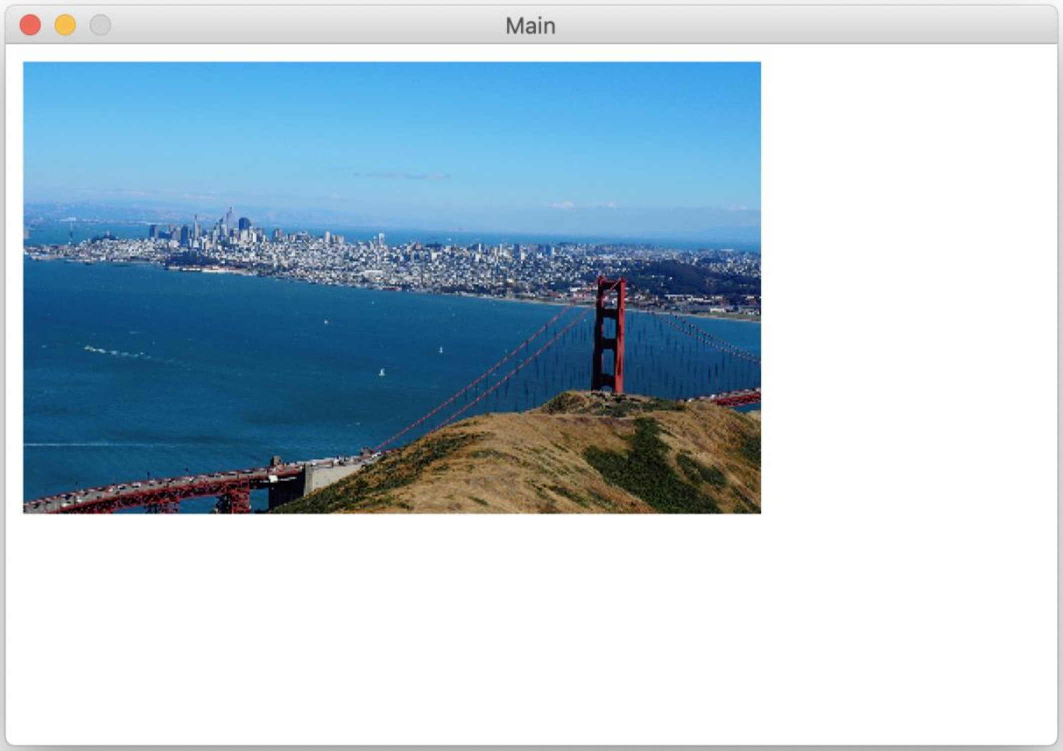


ImageView

Image views let you efficiently draw any image that can be specified using a `ImageView` object. For example, you can use the `ImageView` class to display the contents of many standard image files, such as JPEG and PNG files. The following example displays a 421 x 258 image of San Francisco loaded from the web and places its top-left corner at the point (10, 10) in the parent view's coordinate system.

```
// Initializes new ImageView object
final ImageView myImageView = new
ImageView("https://upload.wikimedia.org/wikipedia/commons/5/5c/San_Francisco%2C_California._June_2017_cropped.jpg", 10, 10, 421, 258);

// Adds it to mainView so it is rendered
mainView.addChildView(myImageView);
```

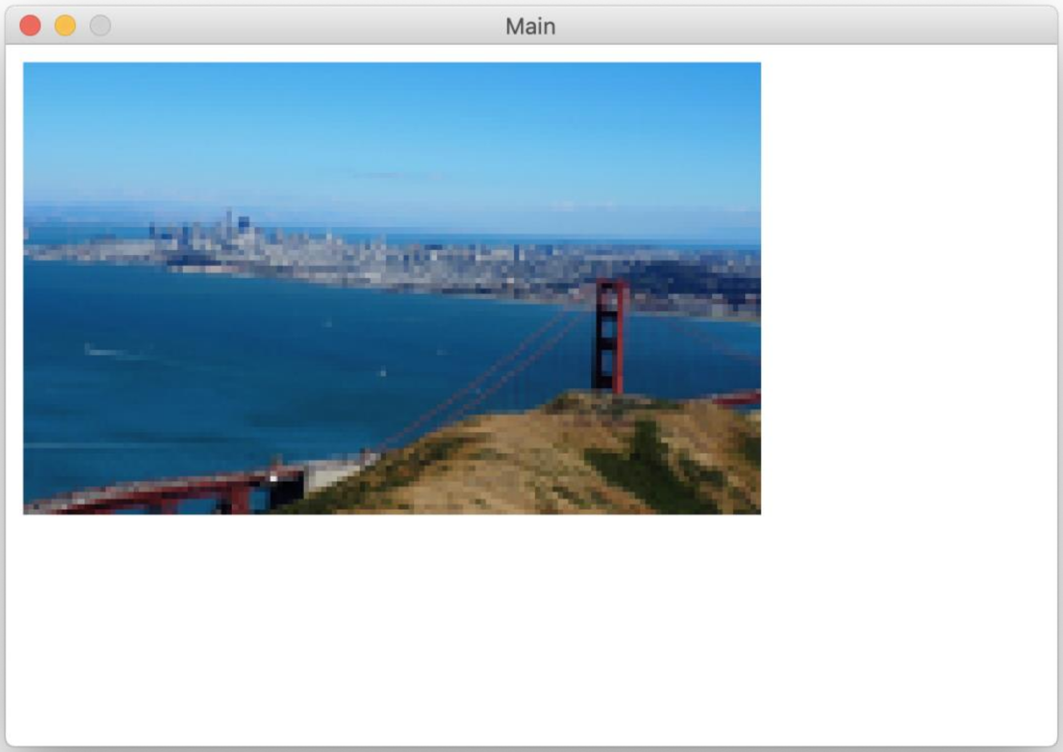


The picture presented in the `ImageView` can be changed by accessing its `photoPath` property. The `photoPath` property is a string that represents either an online URL to an image or the name of a locally stored image.

Applying filters to ImageView

The following example applies a mosaic filter to the aforementioned ImageView. A full list of available filters can be found here.

```
myImageView.applyFilter(MOSAIC);
```



Tools

In addition to the three filters, the following tools can be used to modify the images.

Undo and Redo Buttons

The undo button allows you to revert the changes made to the image by returning the images previously shown in the program. If so desired, the redo button allows for you to reinstate changes which were made at anytime prior to using the undo button.
Click here to read about the undo and redo functions.

Save Button

This button enables you to save a copy of the image which is currently displayed to a location on your computer. Click hereto read about how the program saves files.

Open Button

The open button is used to upload images onto the canvas so that they may be edited. Click here to read about how the program opens files.

Brush Toggle and Colors

The button labeled "Brush" when toggled creates a paint brush which moves with your cursor. By pressing down and dragging a line will be drawn. Additionally, the color selected may be changed by utilizing the color selector at the bottom of the canvas. Click here to read about how the brush edits images.

Square Button

This function allows you to click at two points which will become opposite corners of a rectangle created with the color which is selected at that time.
Click here to read about creating blocks of color.

Crop Button

This button allows you to select any two points on the image. These two points will become the opposite corners of a rectangle. All of the area of the photo outside of that rectangle will be removed, thus only saving the area which is contained. Click here to read about cropping the image.

Rubric Items Met

The following points from the rubric (as outlined by the project check-in) were met.

- Filters (part 1)
  - Makes use of the loadPixels() function (see line 13 in Filters.pde)
  - Utilizes a while() loop in order to modify the pixels[] array (see lines 15-16 and 21-22; although the loops are technically for loops, they function very similarly and are more concise)
  - Uses the color() function to reset color values within the pixels[] array (see lines 24-26)
  - Makes use of the red(), green() and blue() functions to manipulate channel colors within the pixels[] array (see lines 30 and 33)
  - Pushes programmatic changes back onto the screen using updatePixels() (see line 44)

- Filters (part 2)
    - Makes use of a temporary array in order to manipulate portions of the image (see line 49)
    - Makes use of the loadPixels() function (see line 50)
    - Utilizes a while() loop in order to modify the pixels[] array (see line 54)
    - Pushes programmatic changes back onto the screen using updatePixels() (see line 87)
    - The filter successfully demonstrates that the student is able to create a filter more advanced than simply shifting color values. (The edges filter considers other pixels in the area in order to determine whether or not to mark an edge.)
- Documentation: See this document
- Aesthetics: Ultimately, the visual quality of the outputs are up to the user to decide, although we believe they are quite nice.
- Loops
    - The author uses a while() loop within his/her code (see lines 154 and 219 of Main.pde and the examples listed above)
    - The author utilizes multiple loops in order to achieve more dynamic effect (see aforementioned examples)
- Syntax: This is apparent everywhere throughout our code. Every attempt was made to simplify the code as time permitted by using a custom library, and no objects are created by the program for no purpose.
- Conditionals
    - Program uses an if-then statement in one of the filters (see lines 55, 61, 66, and 72 in Filters.pde)
    - Program uses an if-then or if-then-else in more than one filter (see line 97)
- Color data type
    - Program uses the color(), red(), green(), and blue() functions effectively (see lines 24-26, 30, and 57-59)
- Correct functions
    - There are no redundant variables or code (see comments for syntax)
    - All three of the filters work and are able to be reset (use Undo and Redo)
    - Author utilizes commenting throughout the code to make his/her coding decisions clearer (see code)

The features detailed above meet the following points from the honors project rubric. (Note: Due to the fact that we realized certain tools were easier to implement after submitting the honors project application, some of the tools and filters may differ from what was listed initially.)

- Basic user interface
    - The program allows for an image to be loaded from file. (Open button)
    - The program allows for an image to be saved to file. (Save button)

    - There is a dynamic canvas that adjusts in size to the image being loaded. (ImageView changes size to correspond to the resolution and aspect ratio of the loaded image, even during crops.)
    - There are togglable buttons for tools and filters. (The tool buttons are all togglable. Because some of the filters can be applied multiple times to achieve different effects, and because the order in which the filters are applied makes a difference, the filter buttons were not made togglable.)
- Advanced user interface
    - There is an action history built in that allows at least one undo command. (Undo/redo buttons)
- Image manipulating tools
    - Brush (Brush tool)
    - Crop (Crop tool)
    - Color selection (at bottom of screen)
    - Draw basic shapes (Square tool)
- Advanced filters
    - Mosaic
    - Edges
    - Noise (all inspired by ideas in the honors project introduction)
- Program documentation: See this document

Reflection

To implement the features necessary for this project, we decided to first develop a small library to create the various features we would need, such as `View` and `Button`. This allowed the user interface to easily be made uniform, and it ensured the necessary features (such as recognizing when a mouse clicked a button) were always included. We then developed the filters in separate programs to test them on images in a streamlined environment; they were then imported into the program. The undo/redo functions were then added and checked to make sure they worked even when multiple filters were applied.

The tools were added once the filters were completed. The brush tool came first; when it was first implemented, the color was always set to white for simplicity. The square tool was implemented next (with the color always set to white as well); its method for determining which pixels had been selected was used for the crop tool as well. Finally, the color selector was added, and the color variable was inserted into the code for the brush and square tools.

During the course of this project, several major issues were encountered during the coding process. They included the following:

- To make the program look professional, we decided to make the program run at full screen. However, since different computers have different resolutions, no variables regarding position could be hard-coded; everything had to be set relative to the dimensions of the screen. While this was not necessarily difficult, it was tedious and made the code more complex.
- The undo/redo functionality relies upon an `ArrayList` storing different images to keep track of the various edits that have been made. To make this work, though, we had to figure out how to maneuver through the array, account for varying image sizes from the crop tool, and delete the correct images when a new filter or tool was applied to the image (otherwise, clicking 'Redo' may have advanced the program to another image when it should not have). The errors were particularly important to work out here because bugs in this code would lock up the program completely.
- The brush tool required a system for relating the mouse's position on the screen to the pixels in the image, which was particularly challenging for resized images. The variables `shrinkRatio1` and `shrinkRatio2` were ultimately introduced to keep track of the effect of the rescaling. However, to check if the pixels were close to the mouse, each pixel in the entire image had to be systematically checked. This system works, but it is inefficient and can cause the brush to lag, resulting in imperfect lines. This area would be a focus of any future development projects.

Resources

An invaluable resource during the course of this project was the Processing language reference page, located at https://processing.org/reference/.