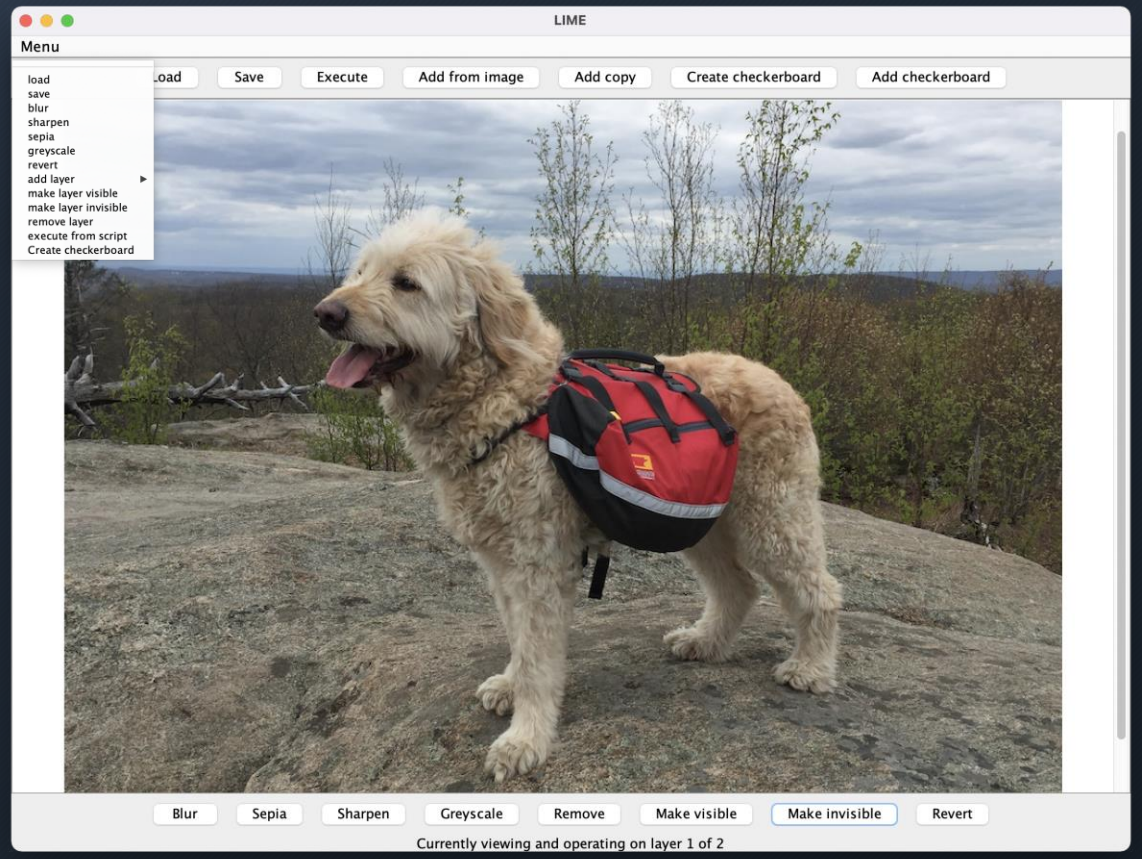
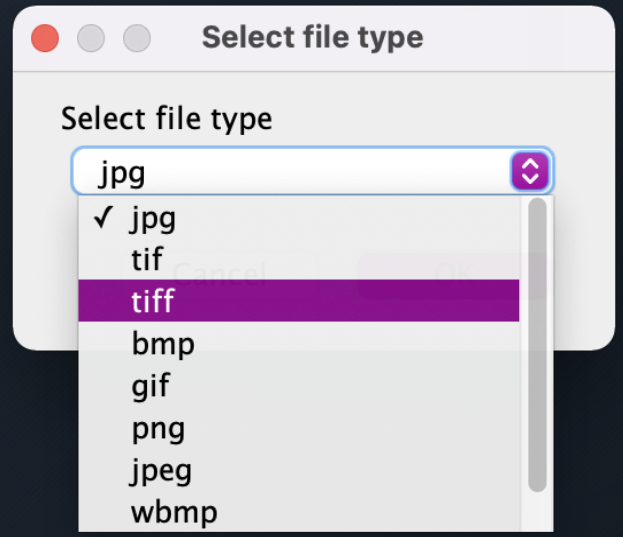
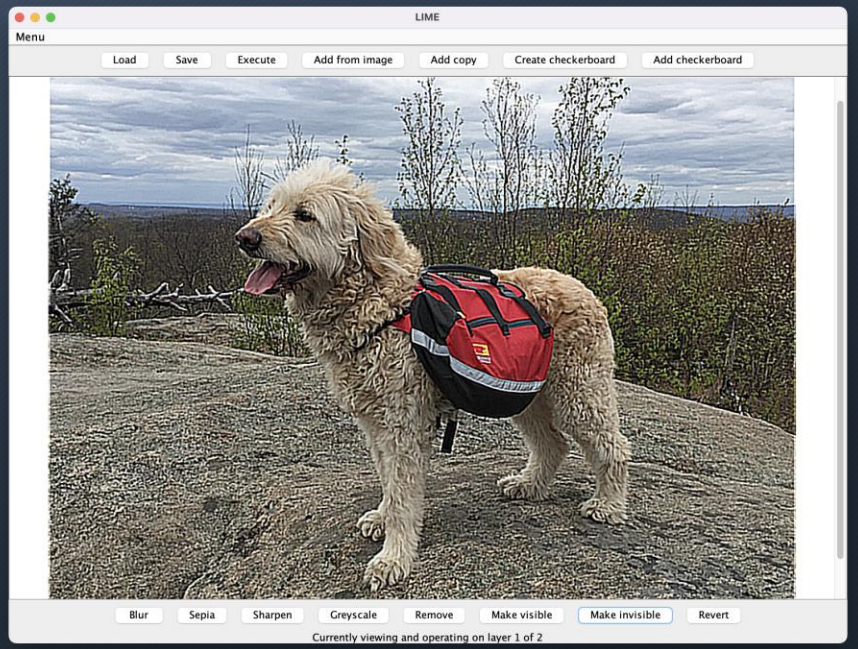
Layered Image Manipulator & Enhancer

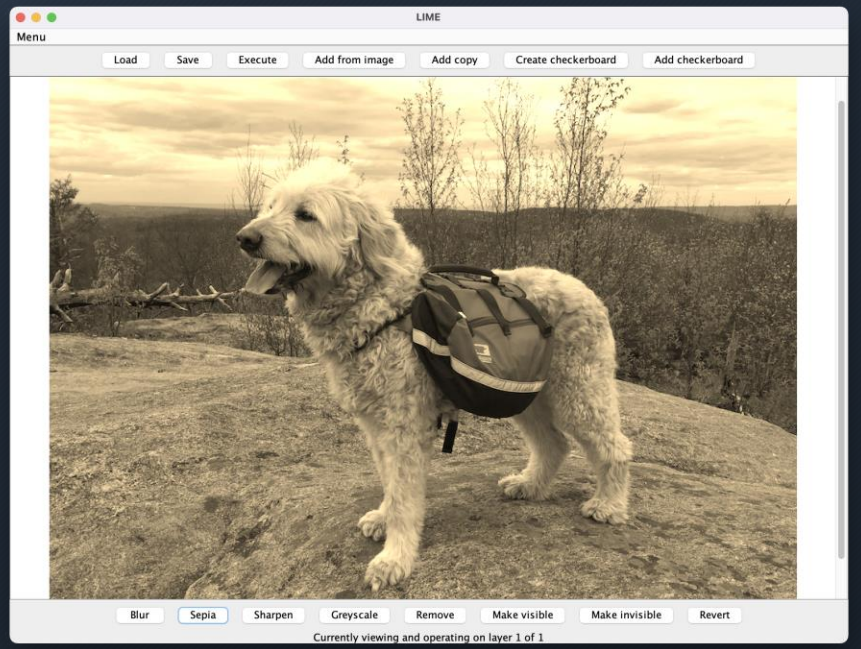
Zachary Rippas

Project Overview

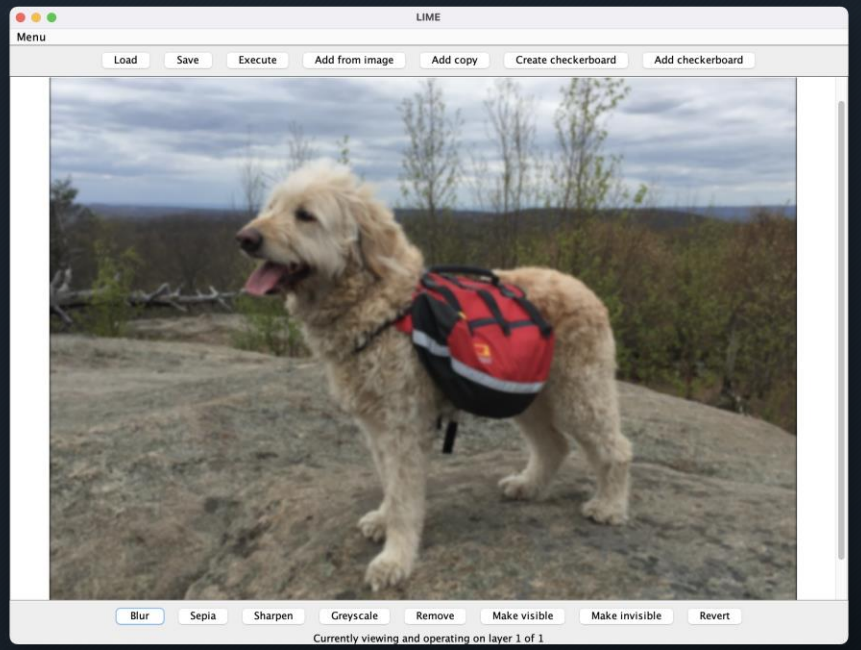


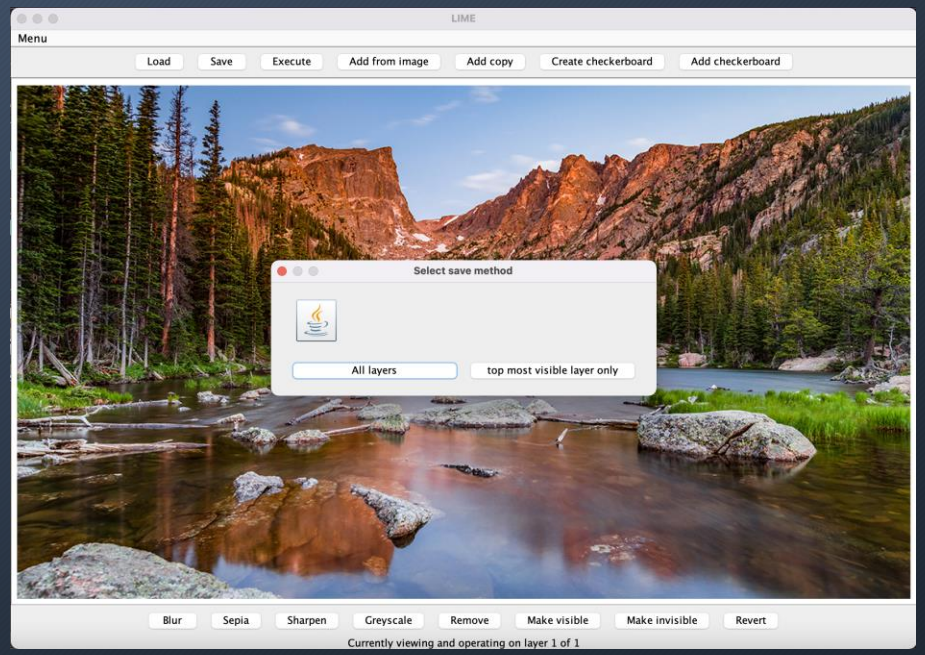


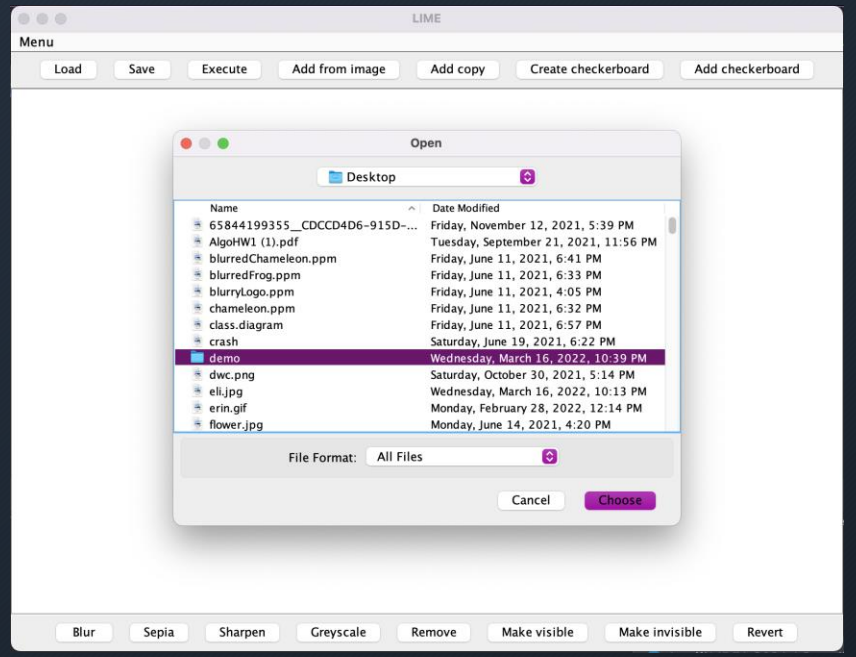

LIME
Menu
Load
Save
Execute
Add from image
Add copy
Create checkerboard
Add checkerboard
Blur
Sepia
Sharpen
Greyscale
Remove
Make visible
Make invisible
Revert
Currently viewing and operating on layer 1 of 2

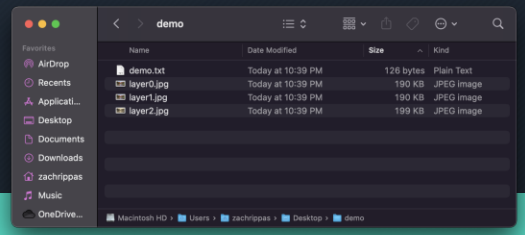

LIME
Menu
Load
Save
Execute
Add from image
Add copy
Create checkerboard
Add checkerboard
Blur
Sepia
Sharpen
Greyscale
Remove
Make visible
Make invisible
Revert
Currently viewing and operating on layer 1 of 1


LIME
Menu
Load
Save
Execute
Add from image
Add copy
Create checkerboard
Add checkerboard
Blur
Sepia
Sharpen
Greyscale
Remove
Make visible
Make invisible
Revert
Currently viewing and operating on layer 2 of 2


LIME
Menu
Load
Save
Execute
Add from image
Add copy
Create checkerboard
Add checkerboard
Blur
Sepia
Sharpen
Greyscale
Remove
Make visible
Make invisible
Revert
Currently viewing and operating on layer 1 of 1







Download/Upload Functionality

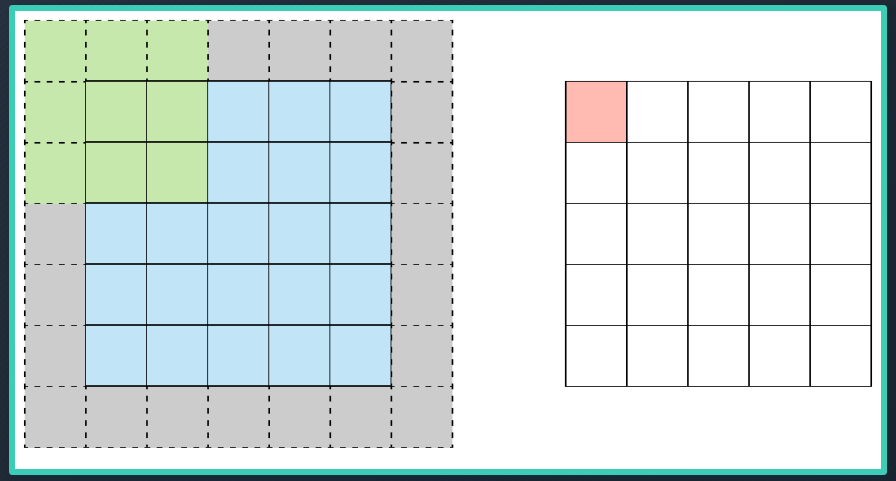
Design Overview

- Strong emphasis on OO principles → Encapsulation, Abstraction, Inheritance, and Polymorphism
- Followed MVC design strategy
- Implemented Command Design Pattern and Factory Pattern
- Designed with future extensibility in mind

Model View Controller

Algorithms Used

Filtering: $O(nm)$



$$\begin{bmatrix} r' \\ g' \\ b' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} * \begin{bmatrix} r \\ g \\ b \end{bmatrix}$$

Color Transformation: $O(n)$

Code Samples

```java
@Override
public void greyscale() {
  versions.push(new Greyscale().apply(versions.peek()));
}
```

```java
@Override
public void revert() throws IllegalStateException {
  if (versions.size() == 1) {
    throw new IllegalStateException("Nothing to revert to");
  }
  versions.pop();
}
```

```java
Pixel[][] pixels = new Pixel[image.getHeight()][image.getWidth()];

for (int i = 0; i < image.getHeight(); i++) {
  for (int j = 0; j < image.getWidth(); j++) {
    int pixel = image.getRGB(j, i);
    Color color = new Color(pixel);
    int red = color.getRed();
    int green = color.getGreen();
    int blue = color.getBlue();

    pixels[i][j] = new Pixel(new ClampedColor(red, green, blue));
  }
}
```

```java
@Override
public File chooseFile() {
  final JFileChooser fchooser = new JFileChooser( currentDirectoryPath: ".");
  fchooser.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
  int retvalue = fchooser.showOpenDialog( parent: GraphicalView.this);
  if (retvalue == JFileChooser.APPROVE_OPTION) {
    return fchooser.getSelectedFile();
  }
  return null;
}
```

```java
@Override
public void export(String fileName) {
  File outFile = new File(fileName);
  try {
    outFile.createNewFile();
    String format = FileUtil.getFileExtension(fileName);
    FileUtil.checkNotSupported(format);
    ImageIO.write(makeBufferedImage(), format, outFile);
  } catch (IOException e) {
    throw new IllegalArgumentException("Writing to the given file failed: " + e.getMessage());
  }
}
```

```java
@Test
public void testConstructorClamps() {
  c = new ClampedColor( red: -100,  green: 50,  blue: 300);
  assertEquals( expected: 0, c.getRed());
  assertEquals( expected: 50, c.getGreen());
  assertEquals( expected: 255, c.getBlue());
}
```

```java
@Test
public void testCreateWidthHeight() {
  assertEquals( expected: 1024, new ImageFromPPM().create("Koala.ppm").getWidth());
  assertEquals( expected: 768, new ImageFromPPM().create("Koala.ppm").getHeight());
}
```

```java
for (int i = 0; i < expected.length; i ++) {
  for (int j = 0; j < expected[0].length; j ++) {
    assertEquals(expectedImage.getPixelAt(i,j).getColor().getRed(),
        blurred.getPixelAt(i, j).getColor().getRed());
    assertEquals(expectedImage.getPixelAt(i,j).getColor().getGreen(),
        blurred.getPixelAt(i, j).getColor().getGreen());
    assertEquals(expectedImage.getPixelAt(i,j).getColor().getBlue(),
        blurred.getPixelAt(i, j).getColor().getBlue());

  }
}
```

```java
@Test(expected = IllegalArgumentException.class)
public void testNullImage() {
  ImageOperation blur = new ImageBlurrer();
  blur.apply( image: null);
}
```

```java
@Test
public void testSaveWiring() {
  this.mockView.fireSaveEvent();
  assertEquals( expected: "save initiated", this.vout.toString());
  assertEquals( expected: "save handled", this.cout.toString());
}
```

Quality Assurance

github.com/zatchet/LIME