***Developer’s Guide***

Image Editor

Khizzer Ahmed, Abdullah Syed, Roland Vargas

## **Introduction:** This document serves as a guide for developers who will maintain and possibly extend the ImageEditor.py software. It is intended to provide all necessary information on the software's design, implementation, and testing. Additionally, the guide will assist developers in understanding and modifying the codebase effectively.

## **Software Overview:** ImageEditor.py is a Python-based GUI application developed using the Tkinter and PIL (Pillow) libraries. It enables users to perform various image manipulation tasks such as drawing, applying filters, creating a new blank image, and modifying image attributes (e.g., colors, size). The software is structured to facilitate easy maintenance and scalability by following modular programming principles.

## **System Requirements**

* Python 3.4
* Tkinter
* Pillow

## 

## **Key Features**

**Basic Image Operations:** Load, save, and create new images with specified dimensions and colors.

**Drawing Tools:** Freehand drawing, line drawing, and color filling.

**Image Filters:** Includes grayscale, invert colors, black & white, horizontal and vertical flip, blur, sharpen, and quantize.

**Undo/Redo:** Supports undo and redo operations for all manipulative actions on the image.

## 

## 

## 

## **Code Structure**

### **Main Components**

**ImageEditorApp Class:** This is the core class that initializes the application GUI, binds event handlers, and manages user interactions.

**Toolbar:** Contains buttons for performing image operations and applying filters.

**Canvas:** Displays the current image and handles drawing operations.

### **Utilities**

**Color Management:** Functions to convert between RGB and HSV color models and apply color changes.

**Image Filters:** Separate methods for each filter, allowing for modularity and easy updates or additions.

**Undo/Redo Stack Management:** Methods to manage undo and redo operations, limiting the stack to prevent excessive memory use.

## **Implementation Details**

### **Initializing the GUI**

The application window and its components are initialized in the constructor of ImageEditor.py. This includes setting up the main canvas, toolbar, and binding mouse events for drawing.

### **Drawing on the Canvas**

Drawing operations are managed through mouse event handlers that track mouse movement and apply drawing or painting effects directly on the image displayed on the canvas.

### **Applying Filters**

Each filter is implemented as a method within the ImageEditorApp class. Filters modify the image data based on the algorithm defined in the method (e.g., averaging pixel values for blur).

## **Error Handling**

The application includes basic error handling for file operations and user inputs. This ensures robust performance by notifying users of issues like invalid inputs or file access errors through dialog boxes.

## **Coding Logic** This section describes how parts of the program logically function.

Import statements: These allow the program to access functions from the tkinter and Pillow libraries.

Class ImageEditorApp: This is the main class definition for the Image Editor program as an object. It contains most of the non-library function definitions and variables.

Variables:

* self.root.title: Sets the name of the program window.
* self.root.geometry: Sets the size of the program window.
* self.oldx and self.oldy: Store the previous cursor position while drawing to the image. Used to draw lines from the old position to the new position every frame.
* self.undoImageStack: Stores previous versions of the image which can replace the current image with the Undo button.
* self.redoImageStack: Stores undone versions of the image which can replace the current image with the Redo button.
* self.imageStackLimit: An arbitrary limit to the number of image versions the undoImageStack and redoImageStack variables are allowed to maintain. This is so no unintended performance issues arise after performing many actions.
* self.imgpath: Stores the file path for an image to be loaded. Can be utilized later to set the destination of save operations by default to the location loaded from or last saved to.
* self.img: Stores the image data using the Pillow library’s Image object. Set to a blank, white, full-opacity, 500x500 pixel image when the program launches.  
  Parameters in Image.new(): Type (such as “RGB” or “RGBA”), Dimensions (x, y), Color ((Red, Green, Blue, Alpha) for type “RGBA”)
* self.imgdata: Stores the image data in a form converted to work as a tkinter interface PhotoImage object using the Pillow library’s ImageTk adapter.
* self.brushSize: Not used, as brush size customization is not yet implemented.
* self.brushColor: Stores the brush color. Applied when drawing to the image.

ImageEditorApp function calls:

self.createMainCanvas(): Creates the main Image Editor window.

self.createToolbar(): Creates the toolbar with buttons at the bottom of the main Image Editor window.

self.openColorChangeWindow(): Creates the small secondary window for brush customization settings. Currently only has a color selector button which opens up the color selector.

self.canvas.bind: Sets functions to be triggered by specific inputs. Pressing the left mouse button calls self.paintStart, moving the mouse with the left mouse button held calls self.paint, and releasing the left mouse button calls self.paintFinish.

ImageEditorApp functions:

createMainCanvas(): Creates the main Image Editor program window with a light-gray background (Color: F0F0F0). Uses self.canvas.create\_image() to display the initial blank image.

createToolbar(): Defines the toolbar and the buttons on it. Matches the buttons with their proper functions. NOTE: I just realized the redo button is named undo internally, and the new image button is named load image internally here, but the variable reassignment has not caused any errors.

openColorChangeWindow(): Creates the brush color customization window with a button to open the color chooser. Future features which can be implemented: Can display a box of the current color, the integer RGBA values, and the color hex code in this window.

refreshImage(): Clears the image loaded in the main window and redraws it. This is used to update the screen after the internal image data has been changed.

newImage(): Calls the Pillow library’s new image function, converts it to tkinter’s PhotoImage object time, and updates the displayed image.

newImageChoice(): Uses tkinter’s dialog windows to ask the user for new image dimensions. The dimensions are then used to call the newImage() function to create a blank, white, full-opacity image.

loadImage(): Uses tkinter to open a file browser to get a user-input filepath. It then attempts to load the image using the Pillow library and converts it to the RGBA type if it isn’t already in that form. It is then stored in self.img and converted to tkinter’s PhotoImage object type with the Pillow library’s ImageTk adapter. Then the display is refreshed with self.refreshImage().

paint(): Draws a line from the old cursor location to the current one, if at least one location is on the image, and there is an old position. Updates the old cursor position as the current position for the next pass. NOTE: This could cause a bug where quickly drawing over the corner of the image does not draw anything, as both endpoints could be off-image.

paintStart(): If the cursor is on the image, draws to the initial cursor pixel position and sets this position as the next old cursor position.

paintFinish(): Un-assigns the old cursor position variable.

drawLine(): This draws a line from one pixel location to another, checking each location to make sure it is still on the image, in case one endpoint is out-of-bounds. This code is rather complicated and could take time to fully grasp.

checkPlace(): Checks if a pixel position is inside the bounds of the current image and draws there if it is.

storeImageUndo(): Stores the current version of the image data in the undo stack. Drops the oldest version from the stack if it has reached the arbitrary stack size limit. The function name includes “Image” because a possible later feature is to have separate undo/redo function and button sets for layer changes and brush changes.

imageUndo(): Pops an image version from the undo stack to replace the current image. Before doing so, it places the current image into the redo stack.

storeImageRedo(): Stores the current version of the image data to the redo stack. Drops the furthest version from the stack if it has reached the arbitrary stack size limit.

imageRedo(): Places the current image into the undo stack, then replaces the current image with the version popped from the redo stack.

addColorFilter(): Asks for user-input values using tkinter’s dialog box windows. Gets Red, Green, and Blue values and adds them to each pixel in the image. Input negative values to subtract color.

invertColorFilter(): Inverts the image by taking the RGB value of each pixel and setting it to 255 minus the initial value.

grayscaleFilter(): Removes the hues from the image by averaging the RGB values per pixel.

bwFilter(): Converts the image to only full black and full white by taking the average of the RGB values per pixel and comparing it to 50% brightness gray. If it is above, it is replaced by white. If it is below, it is replaced by black.

hFlip(): Makes a copy of the image, and copies it back in the reverse order of column positions to horizontally flip the image.

vFlip(): Makes a copy of the image, and copies it back in the reverse order of row positions to vertically flip the image.

aaBlurFilter(): Makes a copy of the image. For each pixel of the copy, it takes the average of the adjacent pixels and the current pixel and sets the original image’s pixel to that average.

aaSharpenFilter(): May not currently produce the intended output. Makes a copy of the image and gets the average of a pixel and its adjacent pixels for each pixel. Each pixel of the original image is changed to be further from the average.

quantize(): Takes a user-input “step” value with tkinter’s dialog input window. Each RGB pixel value in the image is rounded down to the previous multiple of the step value. This effectively reduces the amount of unique colors in the image, moreso for higher step values.

invertValue(): For each pixel in the image, converts the RGB values to HSV values, inverts the V value (the brightness), and then converts the HSV values back to RGB and sets the pixel to it. This effectively inverts the brightness without changing the hue the way invertColorFilter() does. NOTE: This does seem to cause some strange effects with initially very dark / near black colors. This may be due to pre-existing compression in the images used, as applying this filter a second time did not leave any noticeable degradation.

General Functions (outside of the ImageEditorApp class)

byteLimit(): Ensures a value fits in an unsigned byte. Invalid values are replaced by the nearest valid value and returned.

byteChangeLimit(): Functions similar to the byteLimit() function, but allows negative values. Does not need to include -256 in the range, because the maximum value is 255, and the most subtraction from that maximum should result in 0.

rgb2hsv(): Converts RGB values to HSV. For more information on this, look up online resources.

hsv2rgb(): Converts HSV values to RGB. For more information on this, look up online resources.

The [if \_\_name\_\_ == “\_\_main\_\_”:] code block is the active code start when the program is run. This initializes the program using tkinter, creating the window from the class definition.

**Change Log**

Version 0.1test:

* Main window added
* Load image from filepath implemented
* Preset image loaded from hardcoded filepath
* Brush color window added
* Brush color selection functionality implemented
* Brush size window added
* Drawing to image implemented (only applies to position of cursor each frame)
* Draw with brush color implemented

Version 0.2test:

* Add/Subtract color filter implemented
* Invert color filter implemented
* Grayscale filter implemented

Version 0.3test:

* Black and white filter implemented
* Horizontal flip filter implemented
* Vertical flip filter implemented
* Adjacent average blur implemented

Version 0.4test:

* Line drawing implemented (Draws a complete line between cursor positions)
* Adjacent average sharpen filter implemented. May not work as intended.

Version 0.5test:

* Altered main window height (from 800 to 750)
* Added toolbar and buttons for filters (No means to input values for Add/Subtract color filter yet)
* Load image with file browser functionality implemented
* Undo/Redo functionality implemented

Version 0.6test:

* Added “Load Image” to undo-able and redo-able actions
* Added button for “New Image” (No means to input dimensions yet)

Version 0.7:

* Changed initial image to be blank instead of preset image from hard-coded filepath
* Changed main window background color from white to light gray so that the blank image can be differentiated from the background
* Fixed line drawing so that one endpoint can be off-image. Line draw function now checks each position as well.
* Fixed bug in line drawing (fixed with >= 0 instead of > 0) which prevented drawing to pixel positions with an x or y value of 0

Version 0.8:

* Added dialog windows which ask for input values for the New Image and Add/Subtract Color buttons (Could be improved. Currently asks for a single input per window.).
* Quantize colors filter implemented with input dialog window for input value

Version 0.9:

* Increased main window width from 1000 to 12000
* Added rgb2hsv and hsv2rgb functions
* Invert value filter implemented
* Fixed bug with input dialog windows where both “0” and “None” (null) where treated the same, meaning inputting 0 would cancel the operation as if the cancel button was pressed
* Added error message windows for errors
* Added notice message windows then attempting to undo/redo when the corresponding stack is empty
* Increased arbitrary undo/redo stack size from 3 to 5

Version 1.0:

* Disabled the unused Brush Size Customization window

## **Testing**

The software has been tested across various scenarios to ensure functionality and stability. Test cases include drawing operations, filter applications, and undo/redo actions.

Test cases:

Test “New Image” with dimensions input  
 -Test with negative (-1x-1) invalid input  
 -Test with 0x0 invalid input  
 -Test with 1x1 valid input  
 -Test with 1000x1000 valid input  
 -Test with unrecommended (5000x5000) large but valid input

Test draw to image  
 -Try to draw outside the image bounds  
 -Draw single pixel to image  
 -Draw a line on the image by clicking and dragging  
 -Draw a line by dragging the mouse off the edge of the image and back onto it

Test color selector  
 -Choose a valid color from the “Choose Color” window  
 -Draw and see if the selected color is applied correctly  
 -Attempt to input invalid values  
 -If you successfully trigger a handled error:  
 Try drawing to make sure the last valid color is applied

Test “Load Image”  
 -Select “cancel” in file browser to ensure the operation is cancelled without causing unintended effects  
 -Select a valid PNG file and test drawing and filters  
 -Select a valid image file and test drawing and filters to see if transparency data was properly imported/inserted  
 -Select a non-image file to test error handling  
 -Input a filename for a non-existent file to test error handling  
 -Input an empty filename to test error handling

Undo/Redo  
 -Try to undo when no actions have been performed  
 -Try to redo when no actions have been performed or undone  
 -Try to redo when actions have been performed but not undone  
 -Try to undo actions  
 -Try to redo undone actions  
 -Try to undo more actions than have been performed  
 -Try to redo more actions than have been undone  
  
Filters without inputs  
 -Apply filter on blank image  
 -Apply filter on test images that have a wide range of colors  
 -Apply filters on a 1x1 pixel image  
 -Apply filters on a 5000x5000 image  
 -Apply the same filter multiple times in a row

Filters with inputs  
 -Perform all tests for filters without inputs using valid input values  
 -Apply filters with lowest valid input values  
 -Apply filters with intermediate valid input values  
 -Apply filters with highest valid input values  
 -Apply filters with invalid input values just below the valid range  
 -Apply filters with invalid input values significantly below the valid range  
 -Apply filters with invalid input values just above the valid range  
 -Apply filters with invalid input values significantly above the valid range

## **Future Enhancements**

**Scrapped initial features:** Brush size customization, basic image saving (to PNG), and image saving for alternate file types.

**Layer Support:** Adding support for multiple layers to enhance image editing capabilities.

**Advanced Drawing Tools:** More sophisticated drawing tools like bezier curves, fill tool, selection tool, color dropper (selects color from image), and text insertion.

**Performance Optimization:** Utilizing more efficient algorithms or Python libraries for faster processing and smoother user experience.

**Layer-Preserving Project File Saving:** Designing a file output type which keeps working project image layers separated so they can be loaded as separate layers later.