Kevin Shin, Arianna Dart, Redi Kurt

COMP 123, Beth Ernst: Final Project

Photoshop-ish

**MANUAL AND REPORT**

**User’s Manual**

Our program, *Photoshop-ish*, allows the user to upload any image from their computer, perform a collection of edits on that photo, and then save the image back to their computer. If the user is interested in having a user-friendly, straightforward program to make basic edits on their photo, our program offers a simplified version of Photoshop which the user might find more intuitive and accessible than a complex photo editing program. This process begins by opening the Python file *Final Project Code.py* in any Python environment and running the program. *imageTools.py* must be in the same directory as the Python code. When the user first runs the program, a GUI interface will appear with a default image in place, and the user can press the button “Click Here to Begin” to upload an image file from any location on their computer. Once that is selected, the picture will be displayed on the interface, with a set of buttons on the right-hand side.

The available tools are as follows:

* *Crop*: The user clicks the button and then clicks twice on the image. The first click on the image designates the top left of the area to be cropped, and the second click designates the bottom right of the area to be cropped. The cropped image will then be displayed on the canvas.
* *Clone*: The user clicks the button, and then clicks twice on the image to define an area to clone. As with the crop function, the first click on the image designates the top left of the area to be cropped, and the second click designates the bottom right of the area to be cropped. Then, the user clicks a third time to clone that region onto the original image. The image with the cloned section is displayed on the canvas.
* *Healing Tool*: The user clicks the *Healing Tool* button, and then clicks a point on the image in order to blur a small region around the click. This converts the pixels in the region into same average RGB values. The image with the slight blur is displayed on the canvas.
* *Grayscale, Sepia, and Posterize*: The user clicks any of these buttons and then anywhere on the image, and the appropriate filter is applied to the entire image. The filtered image is displayed on the canvas.
* *Reset*: The user clicks on the *Reset* button, and all previous edits are cleared. The original uploaded image is displayed on the canvas.
* *Save*: The user clicks on the *Reset* button and can type in a string to designate the name of the file. The image with the latest edits is saved to the same directory where the Python file is located.

In order to quit the program entirely, the user can click the red button in the top left corner to close the window.

**Example**: Let’s say the user runs the program, and clicks the button to import an image. A main window appears which contains the imported image. The user then clicks on the *crop* button, and then clicks on the image twice to generate a smaller version. Then, the user selects the *sepia* button and clicks on the image in order to filter it. Finally, the user is satisfied with the edited image and selects save and enters their desired file name. The final image is saved as a jpg file.

**Report**

**Summary**

Our program, *Photoshop-ish*, only contains one file in which we have all of our code. However, in the zip folder there is also the image tools file, and the Sample Image folder which was provided in class. We provide blemish.jpg, a sample image on which we experimented with our functions. The user can import any photo of their choosing. Our program is structured as follows:

***Imports***: At the top of our python file we have our imports which are “tkinter” and “imageTools”.

***GUI Main and importPhoto***: Next up, we have our “GUIMain” function which will create a welcome window, with a text label, an image and an import button.  When the user clicks the import button our next function, “importPhoto”, will run. “ImportPhoto” opens another window which allows the user to select an image which they would like to manipulate. After the image is selected, the function closes the previous windows and opens the main window of our function, which contain the main canvas where the selected image is displayed, and all the buttons corresponding to our functions. The main canvas is set to the width of the imported photo. If, however, the height of the photo exceeds 900 pixels or width of the photo exceeds 700 pixels, the “scaleDown” function is called. This function scales the selected image down by half in order to ensure it can be displayed on a normal laptop screen. The canvas is then set to the scaled down image width.

***Click Functions***: Afterwards, we have our click functions. Their only job is to set our global variable “whichclick” to a certain value. These are the functions bound to the Buttons in the main window, so that when “whichclick” is set to a value, the coordinates function will only call one of our photo editing functions with the mouse event. We have six click functions: “cropClick”,”healClick”, “cloneClick”, “grayClick”, “sepiaClick” and “posterizeClick”, each corresponding to one of our main image manipulation functions.

***Coordinates***: Then, we have our “coordinates” function which controls the calls to the functions based on the value of the global variable “whichclick”.This function, depending on that value, takes in mouse events and calls a photo editing function. This function is the one which is bound to our canvas so is essentially the function that acts as the link between the user interface and our program.

***Functions***: Finally, we have all our individual photo editing functions. The function “cropImage”, crops the part of the image selected by the user, “healingPatch” masks the blemishes by blurring the area selected by the user and then the “cloneFunction” clones the area selected by the user and “copyPicInto” pastes that cloned area into a selected position by the user on the photo. Moreover, in this section we have the three filter functions: “grayscale”, “yellowSepia” and posturize which is divided into two functions “convertTo4” and “posterize64”. These functions apply the respective filter onto the photo in the canvas.

***Reset and Save***: Lastly, we have three more functions at the end: “reset”, “savePhoto” and “entryResponse”. “Reset” will undo all the manipulations the user has done on the image and set the picture of the canvas to the original image. “SavePhoto” opens a window that prompts the user to enter a name for their image. This function binds the return key to “entryResponse”, which in turn will take the image currently displayed on the canvas and save it under the name entered by the user.

**Implementation/Testing**

Initially, we started off by creating the main GUI for our program which included a canvas and the necessary buttons. When developing our photo editing functions, we confirmed that they worked visually; since our functions take in images and (x,y) values, we could simply ask the function to show the returned image created by the function. After we were confident that all of the functions did what we expected them to do, we started building the coordinates function and click functions which would set “whichclick” to a value we wanted it to hold in order to call the proper function.

We made sure that each of our functions worked individually, but we ran into some issues when trying to piece them all together. Using many print statements and running code line by line, we realized that the issue was getting the edited image to display on the canvas in the GUI. We realized that this was because tkinter has its own image type, and it is necessary for images to be in this format in order to display on a GUI canvas. However, in order to edit the image, it needed to be compatible with imageTools. In order to get around this, we made two global variables, “pic” and “tkpic”. “pic” was the imageTools compatible type, while “tkpic” was the tkinter compatible type. Throughout each function, we edited the “pic” image. We then saved all changes made to the “pic” image into the global variable “pic”. This way, when the next function was called to make edits, it would make edits to the most recent version of “pic”. In order to get this edited image to display on the canvas, we converted it to “tkpic” so as to be compatible with tkinter and allow us to display it on the canvas. In this way, our global variables are continually updated throughout the running of the code and this allowed us to switch at ease between editing functions without worrying about the sequence of edits made.

With each of our functions working correctly on their own, and the two global variables implemented to ensure smooth transition between all functions, our code ran smoothly and predictably.

Thus, we had the main window with the canvas, buttons that would control the value of “whichclick”, and the coordinates function which would call the photo editing functions themselves. We modified our photo editing functions to be responsible for setting the image on the canvas, so that when the function was called, it would handle both the image manipulation and replacing the photo on the canvas. We then moved on to creating a welcome window and binding our main window to the “Import Photo” button. In addition, we worked on our “Reset” and “Save” function to conclude the running of our program. One last edit we made was to resize the image that the user selected if it exceeded coordinates that we determined to be about the size of an average laptop screen. We had our canvas set to be the width of the imported image, which proved problematic if the image width was more than about 1,000 pixels. For these large images, the canvas ended up extending beyond the screen, making editing impossible. Therefore, we added a scaleDown function that is called if selected images exceeded screen dimensions. This scaleDown function resized the image, uses the new width to set the canvas width, and displays the scaled down image in the canvas for editing.

**Conclusion**

We started this project with an interest in photo editing and expanding our knowledge of what event-driven programming could do. What we expected was writing functions which would take care of every single portion of the photo-editing process, that is, taking in an image, an event, and setting the photo on the canvas. What we learned was that our code had a much more intuitive structure by allowing the coordinates function to control what function would be called with mouse events, and leaving the photo editing functions as they were, that is, taking in (x,y) values. This allowed us have a recipe for building a function in our interface; having developed a photo editing function which successfully implemented some edit, we could assign them a “whichclick” value, design a button and a corresponding click which would set “whichclick” to this value, and then add an if-clause to our coordinates function that would call the corresponding photo editing function only when “whichclick” was its value.

Having said this, if we were to design another event-driven program, we would split up the task between functions which take care of the event, and other functions to control the actual photo manipulation. That way, we could have a more systematic organization of which functions would be called.

Of course, the natural extension of our program is to create more options for photo manipulating. Functions like undo and redo in published photo editing software are incredibly useful, and allow the user much more range and flexibility in their editing. Furthermore, in order to enhance the efficiency of the user’s interaction, we would have liked to implement more options for the user. We had the idea of allowing the user to control the radius of the heal function so that they could control the area of the region that would be blurred. Nonetheless, our final product allows the user to effectively implement basic photo-editing functions, and we feel like our code is structured in a way to accommodate adding more functions.