BRUSHWORK APP ITERATION #1

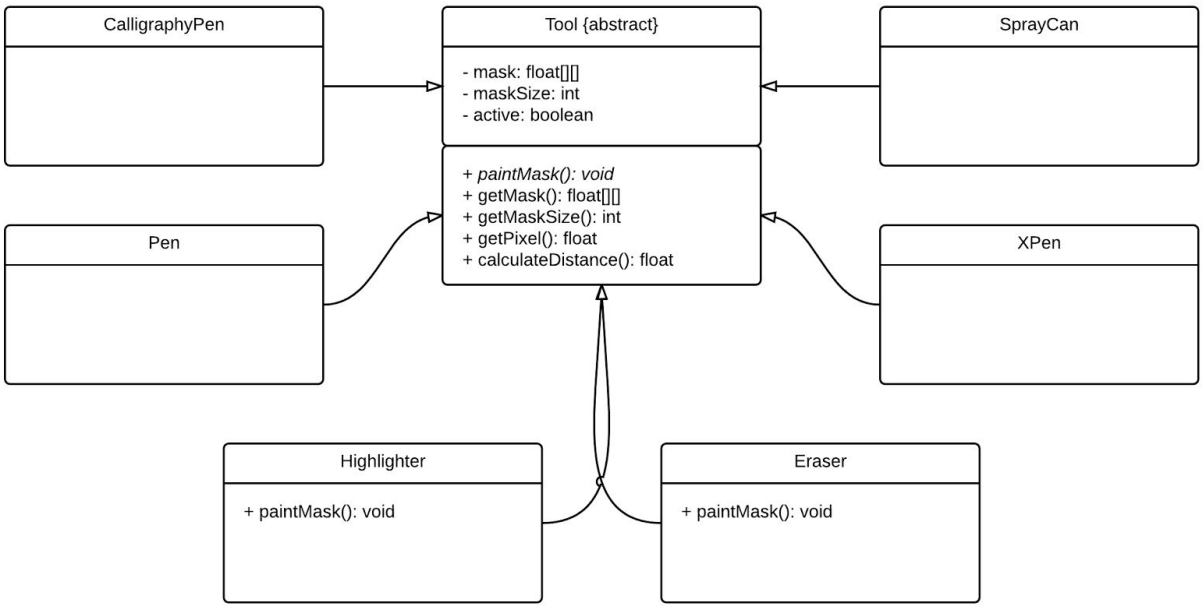
Jonathan Meyer
Steven Frisbie
Jacob Grafenstein

Repository: repo-group-bandcamp

The BrushWork App Iteration #1 is complete with all the specifications outlined in the Official Software Requirements, and each tool works correctly and applies continuously to the canvas.

The most important design decision that was implemented in our solution was how to implement the toolset. We decided to use the Template Method of design so that we could keep a list of the tools in our main BrushWorkApp.cpp file. The Tool.h superclass provided us the flexibility to define generic methods to apply tools to the canvas that could then be overwritten by the children of the Tool.h superclass. A perfect example of this methodology occurs in the paintMask method of the Tool.h file. The Pen.h, SprayCan.h, CalligraphyPen.h, and XPen.h classes all utilize the generic method inherited from Tool.h, whereas the Eraser.h and HighLighter.h classes redefine the paintMask function. The Eraser.h paintMask function modified the method to take in the background color instead of the colors defined by the GUI. The Highlighter.h paintMask utilizes a temporary ColorData pixel to store the modified pixel, and then it compares the colors of the original pixel with the colors of the modified pixel to determine which colors should be adjusted in the main pixelBuffer. Using the Template Method of design, the paintMask function is defined as “virtual” meaning that it can be redefined within its children sub-classes.

We came to this design decision through intense group discussion. We had several other design ideas that were eventually discarded. First, before we fully understood how the support code worked, we did not consider using a Tool superclass and instead just defined each Tool as its own separate entity. However, when we discovered that it would be best to keep all the tools in an array for the GUI to work correctly, we decided the easiest way to do that was to have each tool inherit from a Tool superclass. The UML diagram below provides a good visualization of our final implementation for our tool class structure.



The second design implementation we discussed was similar to the Duck example we saw in class. After we had defined a Tool class, we thought that each paintMask function was likely to be different because each Tool had a different mask size (i.e. the pen tool only had a mask seven pixels wide, whereas the the spray can tool had a mask size of forty-one pixels). So we considered defining a set of Tool behaviors for applying the mask to the canvas. Each behavior would be different depending on the size of the mask, whether or not it took in ColorData or used the background color, and whether it needed to use both (i.e. the highlighter tool). However, we realized that we could make a function that worked with a variable mask size

and that we could create a generic function to handle applying tools to the canvas, and redefine those functions in the separate classes if needed. Thus, we landed at our final decision to use the Template method of design. This implementation can also be observed in the UML diagram above.

To make the code as readable and organized as possible, we tried to change the BrushWorkApp class as little as possible, and added most of the functionality in the Tool class and its subclasses. We arrived at this decision to try to keep any graphics work in BrushWorkApp (which was already provided to us), and provide other functionalities in additional classes (Tool and its subclasses), and use them in the BrushWorkApp class as cleanly as possible.

Initially, we were able to follow this strategy very closely. We were able to implement the basic functionality of the class by only modifying the BrushWorkApp class to have a Tool array attribute and calling paintMask when the mouse is pressed and dragged. This allowed us to implement the basic functionality of the app in an organized and clean manner.

As a group, however, we wanted to implement more than just the basic functionality required, and we had to deviate somewhat from this strategy to make the app cleaner. Our original version, when the mouse was moved quickly, did not refresh quickly enough to draw continuous lines, so we came up with a solution that applied the mask along a line between two points that were dragged. This solution involved storing the coordinates of the previous mouse event (dragged or pressed), adding a method to apply the mask along a line between the last mouse event and the current one and store the location of the current call, and resetting the coordinates to an initial or 'null' (in our case, -1) value to indicate that there is no previous point yet.

Following the strategy we had chosen, we would have implemented all of this in the Tool class. However, implementing this functionality in the Tool class would have caused two issues: First, that most of the necessary information for the functionality (the current and previous coordinates as well as the pixelBuffer) were all contained in the BrushWorkApp class and would have been needed to be passed to the Tool method; and Second, that implementing these in the Tool class would have meant that there would be one instance of each of these attributes and methods for each tool; the current version of the app has 6 tools, and while it would have been easy to determine which method to call to draw the line (just use the m_curTool), it would have been a bit more complex to keep track of the previous points, as it would have needed to keep track of six pairs of previous coordinates.

In the end, we discussed the involved modifications for both methods of implementation: in the BrushWorkApp or Tool class; and in the end chose to break with the overall strategy in and implement the functionality to fill in missing portions of lines in the BrushWorkApp class. This allowed us to more cleanly and efficiently implement the intended functionality by adding the extra previous coordinate attributes to BrushWorkApp only once, and adding the extra methods only once and put all of the additions in one consistent location. Although this went against our earlier intentions, it was the easiest way for us to add the functionality to the app.