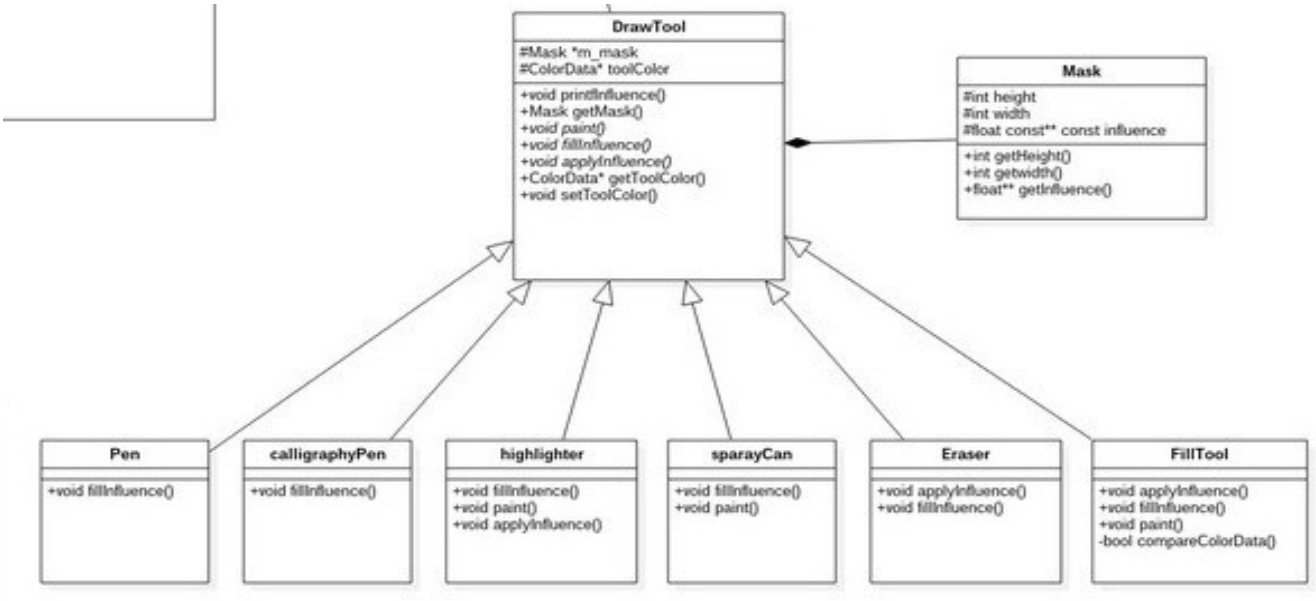
Jacob Reynolds
Juhwan Park
Zhexuan Zachary Yang

git: TwelfthNight

We believe we have fully adhered to the standards given in the handout document and our application should be fully functioning.



The main focus of our design was the DrawTool interface. There was a large amount of discussion between us about whether every tool should inherit from the interface, some tools should sub-inherit from other tools, or if we should not even use an interface. After much deliberation we decided to use a DrawTool interface that every Tool would inherit from. These Tools would also have a mask property This would make it easy to specify height and width for every tool, as well as have a very clear inheritance structure.

Our BrushWorkApp will have one DrawTool active at all times. This DrawTool has a member variable m_mask which will point to it's sub-class. These sub-classes are all of our masks. DrawTool has a paint function that is called when the mouse button is clicked. This will then call the applyInfluence function which applies the strength of every pixel of the mask to the PixelBuffer. The reason for paint calling applyInfluence is because we have properties in the paint function that will fill the gap between two drag events. It calculates the euclidean distance and fills in the gap between them.

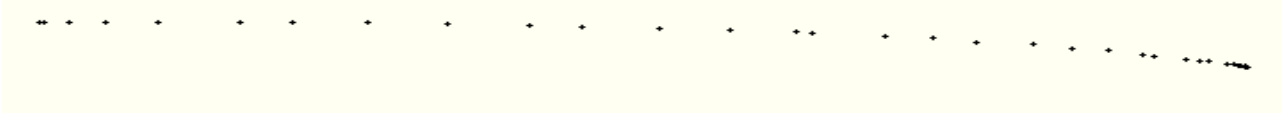
As stated earlier we had some discussion on how to implement this tool structure. The first alternative was to have all rectangle tools inherit from a RectangleTool mask. This would make it easy for things like the highlighter, calligraphyPen, and any future tools that would require rectangle shaped masks. Our main issue with this structure was having inheritance 3 levels deep. The highlighter would inherit from the RectangleTool which would inherit from the DrawTool. We saw no strong reasoning behind complicating the structure this much, and decided each tool will be it's own class that inherits directly from the DrawTool.

Our other option was to have the BrushWorkApp instantiate a Mask property. Then the pen, highlighter, etc.. would inherit from the MaskClass. Our debate behind this centered around the fact that the mask shouldn't be applying itself. The mask should just have its own properties. This is how we abstracted out the DrawTool class. This allows us to define functions that apply the mask to the PixelBuffer. This was an initial design that helped us reach the final stage of our iteration 1 design.

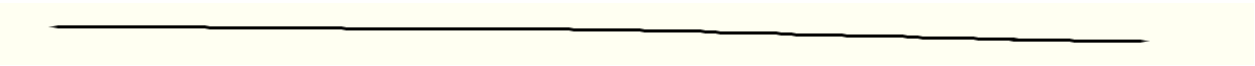
The main advantage of our design compared to alternative #1 is clarity. Our design has a simple to follow inheritance structure. This allows us to remove duplicate code and have expected member variables of every DrawTool. This was clear to everyone in the group after some discussion and it worked favorably when we decided to add a special tool.

The main advantage of our design compared to alternative #2 is concise coding practices. With alternative #2 we would have had to rewrite all of the same functions in each Tool. With our interface we can create one function and all tools will now have that function. It allows us to remove duplicate code and have expected, consistent functionality.

Our second most important design decision would have been the way we fill empty spaces in between lines. The default function for mouseDragged was not triggered often enough. It would cause painting to look like this.



We decided we wanted to make these lines smoother so that the application would feel very responsive. The way we designed this in our application is a very simple and efficient way. When ever the mouseDrag function is called we send the paint function our x,y coordinates, along with a previously stored coordinate pair called prevX, prevY which is the last position called. We then calculate the euclidean distance of these, and calculate the stepping distance. This allows us to quickly and efficiently apply the mask in a for loop for all spots missed between the previous and current coordinates. Doing so gives us lines that looks like this.



This was a much more favorable result for us. We went through a couple ideas while designing this though. Our first implementation was through program efficiency. We thought maybe the lagging was coming from inefficiency in our program. We optimized the draw cycle so that the least amount of function calls possible were happening and we made sure to clear any memory usage that was not needed. The problem still persisted though. We eventually discovered it was due to the base application code not being called rapidly enough. This lead to our second implementation.

In our second implementation we tried to do exactly what we are doing now, except we wanted to do it for every tool. This lead to our highlighter and spray can tools not working well. Since we were applying the line so many times in between points, the spray can always appeared very dark, as well as the highlighter. We tried various ways to only apply the fill functions in larger intervals so the mask wouldn't double apply over the same areas on these tools. None of these ideas worked out very well because they ended up hurting other tools' performance.

We ended up fixing the above issue in our final design by disabling the filling for the highlighter and spray can tools. We overrode the virtual paint functions in these classes so that the filling code in DrawTool.cpp would not execute. We think the main advantage of our final design over the first attempt is the fact that our final design worked. We were incorrect as to the reason causing the spacing issue and that is why we did not choose the first implementation. The second implementation was much closer put caused breaking changes in some of our tools. We chose our final design because it enhanced the problem areas of some tools, while still allowing us to just disable it for others without hurting their performance. We are very satisfied with the implementation of this tool for this iteration. In future iterations we hope to clean up this function a little bit since we currently have some gaps when doing diagonal lines. This should be cleared up in future code reviews.