BrushWork (Iteration #1)

Teaching Solution Design Decisions"

1. Tool Factory Class

We decided to use a factory class to manage all our tools. Each tool is inherited from a Tool class, and the only place these tools are explicitly included is the ToolFactory class. The ToolFactory even communicates the names of each tool to the BrushWork class for creating the UI. We chose this pattern in order to minimize the impact of tool change and the addition of new tools.

An alternative was to explicitly define the tool functionalities in BrushWork in specific tool methods and not have tool classes at all. We didn't like this because it would be unclear what would have to change in order to introduce new tools.

Another alternative was to have classes for each tool, but include all these tools in BrushWork. This is better than the previous alternative, but still requires BrushWork to know about each type of tool and be rebuilt every time a tool header changes.

The main advantage of our Factory pattern design as compared to the alternatives is that it hides the structure and addition of tools from BrushWork and only requires BrushWork to know about the factory. !

2. Abstracted Virtual Blending Math for Tools"

We observed that although tools have the common ability of applying themselves to a PixelBuffer (encapsulated in applyToBuffer() function), the method of blending the Tool's color with the canvasColor differed. We choose to abstract the blending logic out of the applyToBuffer() function into a seperate virtual function named blendColorMath(). The default, additive blending is defined in the Tool super class but is able to be overriden by any child class.

For example, the Eraser is an exception to the common additive blending strategy as it blends the canvasColor with the backgroundColor instead of with the toolColor. With our design we changed only the contents of blendColorMath() in the Eraser class to account for the Eraser-specific logic.

This design strategy was also utilized in the Highlighter and Chalk Tools.

An alternative was to have a check in the applyToBuffer method to see what Tool and its corresponsing color blend logic was to be used.

We choose to abstract a Tool's color blending math in this way in order to minimize the amount of added code and to keep the logic consistent between the Tools. We also found the blendColorMath() function useful for tools such as highlighter or our special tool, as we could experiment directly with the logic of pixel blending.