# EasyArt Implementation

Katie Jordan and Manisha Karani

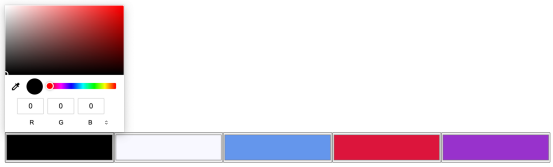
## Implementations

**Drawing Canvas - Katie**

This feature includes the app setup. The most basic functionality of the app is to be able to draw on a canvas. I used 2 canvas elements and javascript to accomplish this functionality. The first canvas is where all of their permanent strokes are drawn. For example, a simple pen drawing would appear on the first canvas. The second canvas is directly over the first canvas. This canvas is where temporary strokes are drawn. Examples of this include the shape snapping feature. The initial drawing is temporary because it will be snapped into a shape. Also, the color selecting tool that appears on click and hold is drawn on the second canvas. The second canvas is necessary because if we draw on one canvas but we want certain strokes to disappear, then we would lose whatever is drawn under that temporary stroke.

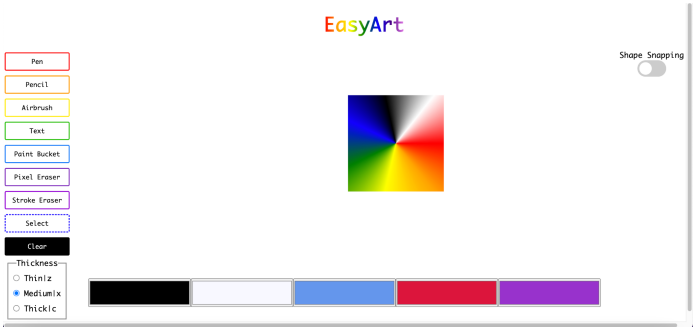
This feature also includes the layout of the application. I chose to have the tool buttons on the left side because it keeps the buttons mostly out of the way but accessible enough. More importantly, the keyboard strokes are essential to making this application much more efficient than the typical art application. The keyboard strokes are on the left side of the keyboard so that the user can draw with the mouse and select the tool with the left hand. This also mirrors the visual layout on the screen.

**Color Toolbar - Katie**



The color toolbar is located at the bottom-center of the screen. Clicking on one of the colors opens up this color selector. This makes selecting colors much more efficient because once a color is selected, it is saved to that color selector so that the user can easily access that color at any point by clicking on the color selector. The color selectors are also large targets near the edge of the screen. This allows for the user to easily select them because they are large and use the screen edges.

**Click & Hold Color Selector - Katie**




The color selector also helps the user to choose a color more quickly and easily. When the user clicks and holds anywhere on the screen, this color wheel appears on the canvas. The user can then move their cursor to anywhere on the color wheel and the place they release is the new color that is selected. This color gets saved in the first color selector on the bottom row. This allows the user to quickly choose a new color without moving their mouse to the bottom of the screen.

**Pen, Pencil, Airbrush - Katie**

The pen, pencil, and airbrush each have different appearances to allow the user to choose what type of shape and texture they want to draw with. The airbrush is lighter and gets darker when the user slows down in an area. This is intuitive because this models real life airbrushes. The pen is a circular dark stroke, and the pencil has straight, slightly distressed strokes typical of pencil lead drawing.

**Select feature - Katie**

The select feature allows the user to click and drag a rectangle over certain drawings and clicking ‘q’ deletes the drawing. I chose ‘q’ because the user is going to be using the left side of the keyboard for all interactions and q correlates with quit which can be translated to clear. There are also tool tips to indicate this feature.

**Clear feature - Katie**

Clear simply clears the entire drawing canvas. The button is fully black to indicate that the interaction is more serious than the other buttons. It is also the last button so it is not likely to be accidentally hit. A further improvement for this interaction would be to implement a pop up that asks if the user is sure before clearing the canvas to account for accidents. This button also has no “quick” shortcut like the other buttons. This was intentional to prevent accidental clearing of the canvas with accidental keystroke hits.

**Thickness Selection - Katie**

Because the keyboard is essential for quick use of the application, I created 3 options for thickness so that they could be selected through the keyboard as well. They are thin, medium, and thick. This is a group of radio buttons, and the user begins with the medium thickness selection. Thickness selection also changes the thickness of the eraser.

**Keyboard Shortcuts - Katie**

All of the tool functionality has a keyboard shortcut that is intuitive based on the screen location of the tools, but there are also tooltips to demonstrate these shortcuts. 1, 2, 3 can be used to select the pen, pencil, and airbrush respectively. This makes sense because they are the top three tools. Paint bucket can also be switched to through typing 4. However, this tool would be useful through a clicking interaction, so triple clicking will fill the clicked area as a shortcut. This way, the user can quickly fill an area without fully switching to the paint bucket. Selecting ‘s’ switches to stroke erase and ‘e’ selects pixel erase. These are easily accessible with the left hand. And, the letters correspond with e for erase and s for stroke erase. There is no shortcut for clear because the user should not be easily able to click clear. Lastly, ‘z’, ‘x’, ‘c’ can be used to select thin, medium, and thick respectively. These are on the bottom row and thickness is at the bottom of the tool selection. There is also an indication on the screen to show which keys correspond to which so that the user does not have to hover in order to see the key shortcut. The other shortcuts are easy to remember because they are intuitive. This shortcut seems less intuitive so the visual without hovering should be helpful in case the user forgets.

**Tooltips & Overall Appearance - Katie**

Tool tips are created for each of the tools so that the user is aware that these shortcuts exist. When the user hovers over these tools, they are able to see what shortcuts there are for that tool. The buttons fill with their color upon hovering to show which button the user is about to select. This helps give visual feedback for the consequences of the user's actions. Clear changes to a white background to mirror what would happen to the canvas when it is clicked.

**Shape Snapping - Katie**

In shape snapping mode, the 1$ recognition algorithm coded by University of Washington is used to detect what shape is drawn and snap to that shape. This code is strictly used to recognize the stroke; I implemented the shape snapping. The user draws on the second canvas and the shape is created on the original canvas after being identified by the algorithm. The app can recognize and snap to triangle, x, rectangle, circle, check, caret, left square bracket, right square bracket, and v. Color, color transparency, and thickness is also accounted for. If the thin, purple airbrush is being used to draw, the snapped shape will appear more transparent, have a thin line, and purple.

**Cursor Images - Manisha**

We replaced the cursor with images to indicate which tool the user is currently using. Each tool has a specific image to represent it. The red x over stroke erase is especially helpful to indicate that it is a click interaction and not a click and drag interaction. One challenge with this feature was that with the new cursor image, it wouldn't draw where you'd expect the icon to interact with the canvas. To solve this, we added an iconOffset x and y value to offset the interaction to the coordinate on the cursor image that makes sense (i.e. the pen icon draws at the tip of the pen instead of at the left corner of the image). To manage different offsets for different image icons, we added state management that keeps track of which tool is being used and adjusts the iconOffset accordingly.

**Pixel Erase (Double Click) - Manisha**

There is a pixel eraser that can erase based on the pixel through turning the stroke white. Thickness also affects the eraser stroke size. The user can double click to switch between pen and pixel erase. This interaction helps when a user makes a mistake in their drawing and they'd like to quickly erase their mistake and then quickly resume drawing again.

**Stroke Erase - Manisha**

Stroke erase deletes an entire line or stroke when the user clicks on any part of the stroke. This allows for easier erasing instead of needing to swipe through every pixel to erase.

**Paint Bucket (Triple Click) - Manisha**

The paint bucket uses the flood fill algorithm to fill any area. To summarize, the flood fill algorithm searches for pixels that are the same color as the pixel that was clicked, adds them to a stack and paints them all the same color as the color that is currently selected. The user can also select the color of the paint bucket to change what color gets painted. Right clicking any area also prompts the paint bucket. Since the paint bucket cannot have a direct keyboard shortcut

because the keyboard does not have a way to indicate location on the screen, having a mouse shortcut is more logical.

## Development Environment

This app is developed in Javascript, HTML, and CSS and is run using a live server. To start the application in VSCode, you need the live server extension. Then, right click on the index.html and click open with the live server. From there, the application should run on the browser (chrome is preferred since we developed on chrome).

## 3rd Party Libraries

**Shape Snapping:** http://depts.washington.edu/acelab/proj/dollar/index.html

Used University of Washington's javascript version of the dollar recognizer algorithm to implement stroke recognition. The result of this recognition is used to snap the shape based on the recognition.

**Flood Fill & Stroke Erase:**

- **Flood fill - Wikiwand**
- **Flood fill algorithm in javascript - LearnersBucket**
- **javascript - Paint bucket getting "Maximum Call Stack Size Exceeded" error - Stack Overflow**

Inspired by these algorithms. We coded the stack implementation based on the pseudocode and we used the skeleton code provided by the stackoverflow post. We coded all of the methods called within the skeleton code based on our understanding of the algorithm from the pseudocode/explanation in the other two links.

**RGB to HEX and HEX to RGB:** https://stackoverflow.com/questions/5623838/rgb-to-hex-and-hex-to-rgb

Code copied that converts RGB values to HEX and the other way as well

**Other:**

We used Javascript, HTML, and CSS documentation, especially w3wschools.com, for all other implementations.