# Project Outline

Drawing app template.

## General practices

I found calling console.log() helpful in debugging.

Some code could be copy pasted or adapted into classes for reuse across different objects.

## Implementing a stamp tool

I neglected the frequency control from the video and opte for a fixed spacing slider instead. It took a simple implementation of HTML elements and JavaScript. Coding-wise, the Shift-key alternate mode for this tool was simple, only relying on an If conditional to detect the Shift-key. Logic-wise, creating the mathematical algorithm to distribute stamp images evenly over a distance was a long process.

## Implementing the scissors tool

I adapted the code shown in the video to be written in the same format as the rest of the tools. This was just a matter of copy and pasting previous formats.

## Implementing tool modifiers

## I worked HTML into my Javascript code to create tool modifiers, such as sliders and input fields for controlling the size of the stroke to a certain extent.

## Most implementations of the standard size modifier simply required me to set the p5js strokeWeight() to the object’s size property. The spray can tool and stamp tool however required more configuration in their algorithms as per their 'shape'.

## Implementing undoing and redoing tool stroke

Although the implementation of a stack is simple to me, my lack of understanding of the pixels system array hampered my progress. Discovering new functions related to p5.js Color got me a working result.

## Implementing shift key mode

## Users can hold down the shift key to toggle an alternative draw mode for all tools. This took the longest to design the algorithm for, as seen in the object method renderAlternate(). Shift-key is detected by an if-conditional at the start of draw() of each tool and requires startMouse positions to be stored as a variable first.

## Progress

I started off with brainstorming and penning down essential features for my drawing app:

Text

Description automatically generated

## Planning tool size sliders

## Diagram Description automatically generated

I may implement the slider, input fields and their respective HTML elements as classes since most of the tools share the same lines of code to implement them. Text, letter

Description automatically generated

## Undoing and redoing

I thought this would be easy as long as I had a stack keeping track of the drawing history so I tried to code it directly. This below was an initial version where the Undo function was limited to one time only, which was not the intended effect. saveState() is called after every mouse press on the canvas. This version also did not have stateFuture updaeted in saveState() as I was testing out undoing before moving on to redoing completely.

Text

Description automatically generated

## Improving stamp tool

I have plans to provide an array of default stamp pictures as seen with var stampImagesAr = Array(10);and if possible add an option to add stamp images from the user’s own storage. So far, I’ve added options to toggle fixed spacing.

## Shift key alternate mode on spray can and stamp tools

It was challenging trying to come up with the mathematical algorithm that would distribute the spray can stroke and the stamp images evenly over the starting mouse position and the ending mouse position. I create an additional renderAlternate() function that was easily editable to display the images after the calculations on their positions were done and passed to it.

Text

Description automatically generated

# Mirror draw for all tools

I’ve experimented with much failure, the main problem is getting the unique shapes of each tool to be represented on the reflected side. I may have to experiment with the set() method, when I still have time to do this extension. I’ve experimented with much failure, the main problem is getting the unique shapes of each tool to be represented on the reflected side. I may have to experiment with the set() method, when I still have time to do this extension. I worked on this for a long time, but it is now at the bottom of my priorities.

## Implementing 2x colour palette

I have also tried this with much failure. The main issue is not knowing how to detect right-clicks with event listeners. Will just have to try out more methods to detect right-clicking or simply add a button that sets stroke/fill colour selection mode. A work-in-progress that was retracted: Text

Description automatically generated

Generally, I thought that each extension/improvement to the code should only require up to a week, with as least overlaps as possible. Any more than that and it may be hard to remember what code I just added to the project.

I knew I had to spent a short amount of time on adapting the old code from drawApp-caseStudy to the new master base as well as touching up, as I had already started working on coding tool size sliders. Hence I though 3 days would be a safe duration.

I allowed myself to brainstorm and design algorithms for original extensions for a week.

Straightfoward extensions/improvements such as UI changes such as renewing drawing tool icons would require 3 days or less. Big edits such as implementing the shift-key alternate mode to all tool types took more time.

I’m not too sure of what more is expected of the project after the mid-term submissions so I’ve left it as debugging and general editing for the duration of a month for now. The Gantt chart will get updated and more details will be added as I go along.

How to remove right-click context menu: https://stackoverflow.com/questions/51958065/prevent-context-menu-from-opening-on-a-right-click-over-element

A tip from someone on the official BsC Computer Science Slack server on implementing undo and redo using a stack structure: [https://imgur.com/I92GM6x.png](https://imgur.com/I92GM6x.ong)

Fontawesome for some of the UI icons.