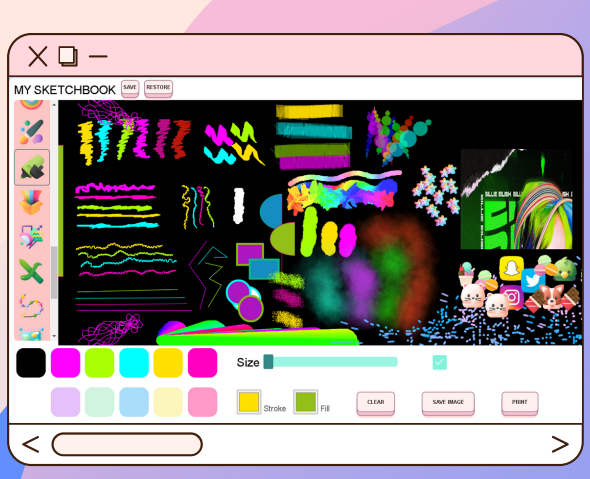
INTERIM PROJECT PROGRESS [001]

Outline your project and which extensions you are building on to the template.
Your answer should include:
Which of the templates you are extending and why have you chosen to do it.
What extensions have you chosen to do. You should include in your answer: any complex coding techniques you will need to use, (such as arrays of objects, constructor functions, nested looping); the complexity of the extension; and any expected challenges you will have implementing it.

CHOICE: DRAWING APP



Untitled - TextEdit

File Edit View Help

There is an extremely small gap or divide between traditional and digital art. Digital art is a relatively new and developing form of expression considered one of the evolutions of today's world. Since, unlike traditional art, it is not subject to the limitations of the material world. You can get an infinite number of colors seen daily by the human eye, change the intensity, brightness or erase everything without leaving a trace.All this has awakened an interest in me to be able to extend this project and develop different tools that could be necessary when carrying out a digital artistic work as if it were a traditional one.

The following functions are included as constructor functions, so they are added as new objects to the toolbar: TOOLBOX(). Also, all constructors have two main properties, this.name = name of the tool and this.icon = path to the icon of the tool. In all tools, the unselectTool() function was implemented. This feature allows changes made within the HTML to be temporary. That is, each time the tool is deselected, the changes made will be removed so as not to disturb the next tool used.

DONE

BACKGROUND
- Change Background color

PAINT BRUSHES
- Freehand Tool (MORE)
- Crayon Tool
- Angle Brush Tool
- Arcs Brush Tool
- Hatching Brush Tool

TOOLS
- LineTo Tool (MORE)
- Editable shape
- Eraser

SHAPES
- Circle Tool
- Rectangle Tool
- MidMoon Tool

SPRAY
- SprayCan Tool
- Spray Pincel Tool

DECORATION
- Mandala Tool
- Radial Rainbow Tool
- WaterColour Tool
- Multicolor Line Tool
- Rainbow Brush Tool
- HIghLight Marker
- AutoRainbow Tool
- Shape Brush Tool
- Blades Tool
- Points Tool

EXTRAS
- Add Image
- Star Sticker Tool
- Stickers Tool

HELPER FUNCTIONS

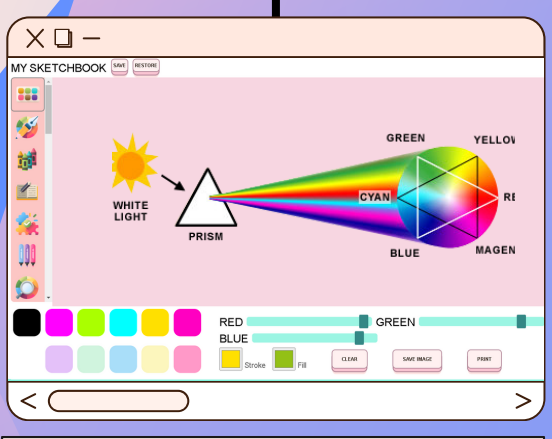
IMPLEMENTATIONS

BACKGROUND, ERASER, AUTORAINBOW, FREEHAND, CIRCLE, SHAPEBRUSH, RECTANGLE, CRAYON, BLADES, MIDMOON, ANGLEBRUSH, POINTS, SPRAY 1, ARCS, INPUTIMAGE, SPRAY 2, HATCHING, STARSTICKER, MANDALA, STICKERS, MIRRORDRAW, RADIAL, CLONE, LINE, WATERCOLOUR, EDITABLE, HIGH LIGHTER

FUTURE IMPLEMENTATIONS

BACKGROUND
Change Background color:
-Colors (RGB, HSB, HSL, ColorPicker)
PAINT BRUSHES
Freehand Tool (MORE):
- Opacity of the pencil
-Cursor
Angle Brush Tool:
-Line thickness
-Line size
-Cursor
Arcs Brush Tool:
-Line thickness
-Color of the line (colorPicker)
-Cursor
Bloom Brush Tool:
-Line thickness
-Color of the line (colorPicker)
-Cursor
TOOLS
LineToTool:
-Opacity of the line
Editable shape:
-Opacity of the line
SHAPES
Circle Tool
Rectangle Tool
MidMoon Tool
(noFill option)
SPRAY
Spray Brush Tool:
-Line color (colorPicker)
DECORATION
Mandala Tool:
-More types
Radial Rainbow Tool:
-More types
Shape Brush Tool:
-More shapes (rectangles, etc)
Blades Tool:
-Line color (colorPicker)
Point Tool:
-Change color mode
-Line color (colorPicker)
EXTRAS
Stickers Tool:
-Rotate sticker
-Change the size
ADDITIONAL FEATURES
New tools:
-cloning
New features:
-Ambiental music
Note:-Fix bugs

BACKGROUND



FILE NAME: BACKGROUNDCOLOR.JS

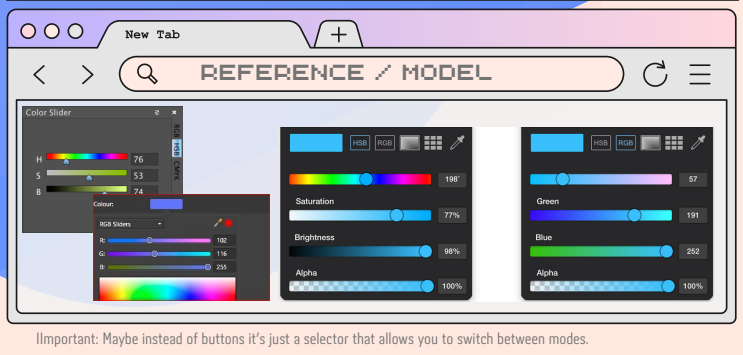
File Edit View Help

BackColorTool() allows you to change the color of the background of the canvas. To do this, I created three sliders within the populateOptions() function that independently simulate the intensity of each color composition in terms of the intensity of the light's primary colors: RGB (Red-Green-Blue). In this way, the first slider simulates the variation of R, the second of G, and the third of B. Thus, each independent value is updated in the draw() function so that the changes are reflected in the background color: background(r,g,b). On the other hand, when the tool is not selected, the unselectTool() function allows you to remove the changes made to the HTML so that they do not get in the way when using another tool.

FILE NAME: BACKGROUNDCOLOR.JS

----FUTURE IMPLEMENTATIONS/CHANGES----

For now there are three sliders to control the RGB values for the color of the canvas, but four buttons will be implemented that allow switching between modes of changing the color of the canvas (RGB,HSL,HSB,ColorPicker). In this way, if we press the first button we can change the colors of the canvas by controlling the values for R-G-B, if we press the second we can change the colors of the canvas by controlling the values for H-S-L and so on. Also, if we press the last button, it will allow us to choose a color for the canvas through a colorPicker.For all this, a button will be created for each mode and each time it is pressed their respective sliders appear to make the changes. If the button is pressed to choose a color through a colorPicker, the colorPicker will appear.



!Important: Maybe instead of buttons it's just a selector that allows you to switch between modes.

CATEGORY: PAINT BRUSHES

FREEHAND

FILE NAME: FREEHANDTOOL.JS

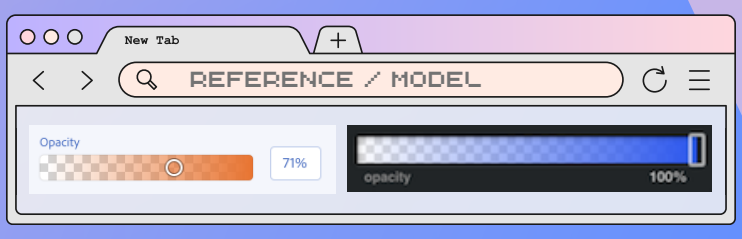
File Edit View Help

I decided to implement several functionalities for this tool. 1. Inside the populateOptions() function we created a slider and a label. The label reflected in the canvas allows us to know what functionality the slider has. On the other hand, the value (x) obtained from the slider is reflected in the draw() function. Thus, the value for the line thickness strokeWeight(x) is the value obtained from the slider.2. The color of the line is congruent to the color that is chosen in the ColorPicker that has a "Stroke" label next to it.3. Implementation of a new cursor model, its size will change as the line thickness value increases.

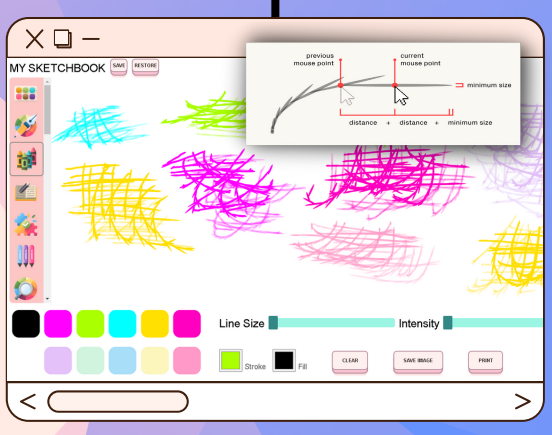
FILE NAME: FREEHANDTOOL.JS

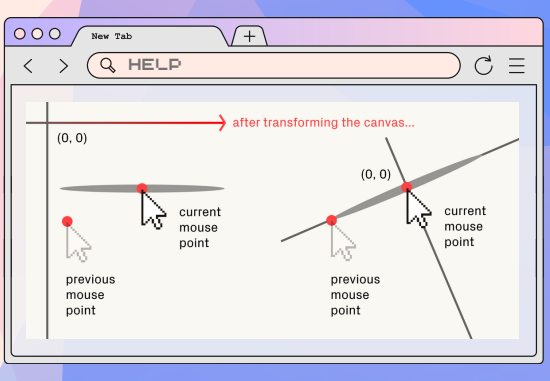
----FUTURE IMPLEMENTATIONS/CHANGES----

A slider will be implemented that allows changing the opacity of the color for the line.



CRAYON





FILE NAME: CRAYONTOOL.JS

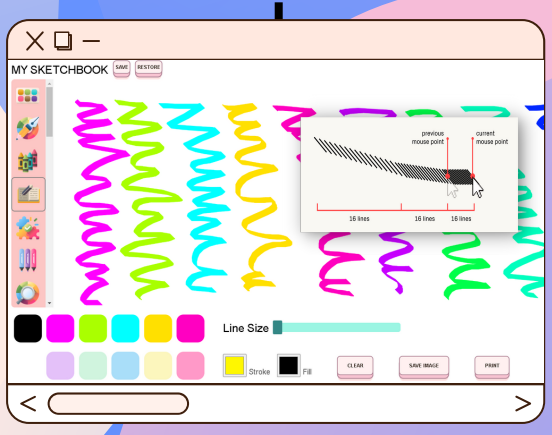
File Edit View Help

CrayonTool() allows you to draw strokes similar to a traditional crayon, which are strictly together to the curved path of the mouse. To do this, ellipses with great width and little height will be created, which will be rotated according to the direction of the mouse. The model of the cursor was changed.
Inside the crayon() function:

1. The origin of the canvas is by default at the (0,0) coordinate and will be moved to the current coordinate of the mouse.
2. The canvas will be rotated with respect to the current and previous mouse position. To do this, we will find the value to rotate with the help of the Math.atan2(x) function and then we will rotate the canvas with the help of the rotate(x) function.
3. The ellipse() function will allow us to draw an ellipse with a very long width and a very short height, this function will be composed of four different parameters ellipse(a,b,c,d), where a and b will be the initial position of the ellipse , since the origin and rotation of the canvas is set with respect to the current mouse position, we can leave it at (0,0), the width of the ellipse (c) will be the distance between the current and previous mouse position. Finally, we will add a height (d) for the ellipse.

- Within the **populateOptions()** function, we will create two sliders, one that will return a value to modify the height of the ellipses (minimunSize) and another to modify the Opacity of the color of the ellipses.
- Inside the **draw()** function, the crayon() function will be reflected and will allow us to draw this simulation of a crayon on the canvas, in addition the slider values will also be reflected.

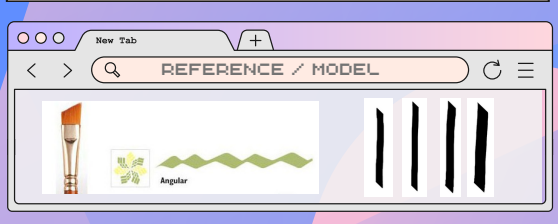
ANGLEBRUSH



FILE NAME: ANGLEBRUSHTOOL.JS

----FUTURE IMPLEMENTATIONS/CHANGES----

Just as the width of the line can be modified, a slider will be implemented that allows us to modify the general size of the line. That is, it will allow us to modify the value of "withPlume" found inside the plume() function. In addition, another slider will be implemented for the opacity of the line



FILE NAME: ANGLEBRUSHTOOL.JS

File Edit View Help

AngleBrushTool() allows us to draw strokes simulating that it is a traditional angle brush. To do this, we will use linear interpolation. This is a method in mathematics to find a number between two numbers lerp() at a specified increment, often called lerping in shorthand.
Inside the **plume()** function:
1.
withPlume = 5
line(mouseX - withPlume = 5, mouseY - withPlume = 5, mouseX + withPlume = 5, mouseY + withPlume = 5)
This code will allow us to draw slanted lines but since the frequency is limited, it will leave large gaps between lines. Therefore, we will repeat that slanted line several times with the lerp() function.
First of all, we will make a loop where the number of times we want it to repeat is 40:
for (let i = 0; i < 40; i++){} inside this loop we will set the two lerp() functions, one for the x coordinate and another for the y coordinate.
So:

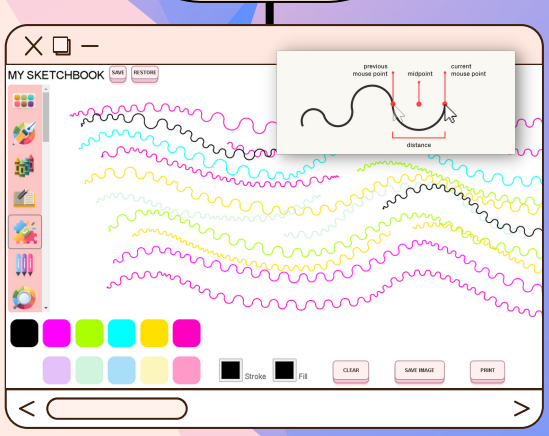
```
withPlume = 5
for (let i = 0; a < lerps; i++)
{
   var x = lerp(mouseX, prevmouseX, a / lerps)
   var y = lerp(mouseY, prevmouseY, a / lerps)
 line(x - withPlume, y - withPlume, x + withPlume, y + withPlume)
}
```

To find the "x" coordinate we can use the current and previous coordinates, the same for the "y" coordinate. Finally, to create equal spacing between each drawn line, the loop index is divided by the total number of loop iterations. All this allows us to repeatedly draw the inclined line at equal intervals between the current and previous mouse coordinates.

Inside the **populateOptions()** function:A slider was created to modify the width of the line and a label to indicate what the slider's functionality is.

Inside the **draw()** function:The plume() function and the slider's value are reflected so they can be used on the canvas.

ARCS



FILE NAME: ARCSTOOL.JS

----FUTURE IMPLEMENTATIONS/CHANGES----

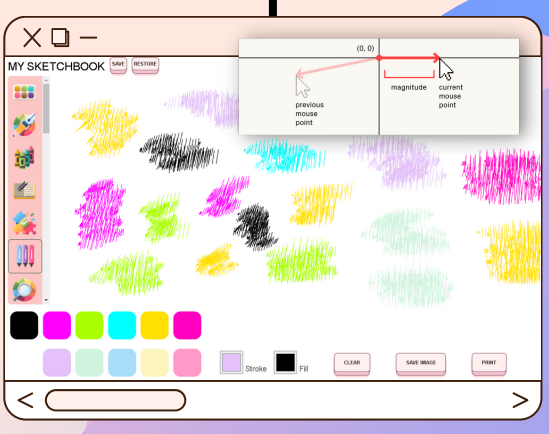
Two sliders will be implemented, one for the width of the line and one for the opacity of the line. On the other hand, to make the color of the line equal to the color selected in the colorPicker, it will return the hexadecimal value of the color selected in the colorPicker that has a "Stroke" tag next to it. Thus, the stroke() will contain the value returned

FILE NAME: ARCSTOOL.JS

File Edit View Help

ArcsTool() allows us to draw wavy paths. This tool is a path of semicircles that alternate as they are drawn. To do this, we use a mid-coordinate between the current and previous mouse coordinates. In addition, we'll also need the dist() function, the current mouse position, and "frameCount", which is the number of frames.
Inside the **wiggleArcs()** function:
Find the distance **"theDist"** between the current and previous mouse coordinates, then find the midpoint between the current and previous mouse coordinates for "x" = "**middleX**" and "y" = "**middleY**". Then find the "angle" of the direction the mouse is moving with the help of the Math.atan2() function. So, Math.atan2(mouseY - pmouseY, mouseX - pmouseX). On the other hand, it is necessary to find in which way to **"flip"** the bow. To do this, we will use (frameCount mod2) * PI. So, (frameCount % 2) will return two values, 0 or 1, which will be multiplied by PI, if it comes out 0 then the result will remain at 0, if it comes out 1 the result will be PI. This value will be included in the parameters of the initial and final angle of the arc. This will flip all arcs accordingly. Finally, we will add all this data to the parameters of the arc() function. So, **arc(middleX, middleY, theDist, theDist, angle + flip, angle + PI + flip).**
Inside the **draw()** function:
New cursor model. When the mouse is pressed, the wiggleArcs() function is reflected and allows the wavy strokes to be made.

HATCHING



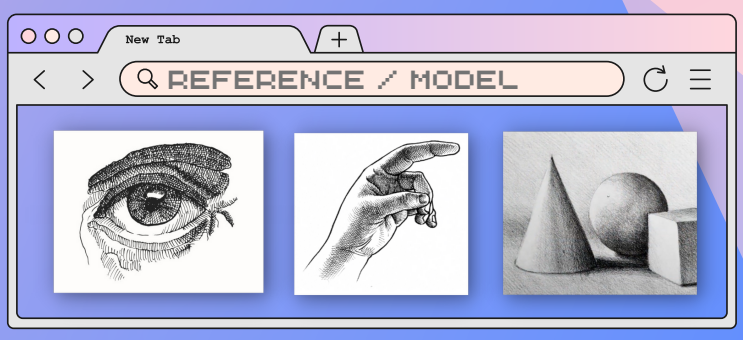
FILE NAME: HATCHINGTOOL.JS

File Edit View Help

HatchingTool() allows you to draw lines of different sizes to simulate a drawing technique for hatching areas in a drawing called **HATCHING**.
Inside the **hatching()** function:
First find the speed of the mouse, this value will be responsible for the size of the lines. Every time the mouse moves faster, the lines will get bigger. Then, to get the relationship between the current mouse point and the previous mouse point, do a "vector" **createVector()** reversing the previous and current values of the mouse coordinates. So, **createVector(mouseY - pmouseY, mouseX - pmouseX)**.
Then, set the magnitude of the vector (the length of the line) based on the speed of the mouse: **vector.setMag(speed / 2)**. Finally we will use the lerp() function to make the strokes smoother and denser.
Inside the **draw()** function:
When the mouse is pressed, the **hatching()** function reflects back and allows us to draw lines that simulate the **HATCHING** effect.

FILE NAME: HATCHINGTOOL.JS

----FUTURE IMPLEMENTATIONS/CHANGES----
Two sliders will be implemented, one for the width of the line and one for the opacity of the line. On the other hand, to make the color of the line equal to the color selected in the colorPicker, it will return the hexadecimal value of the color selected in the colorPicker that has a "Stroke" tag next to it. Thus, the stroke() will contain the value returned by the colorPicker.



GATEGORY: TOOLS

LINE

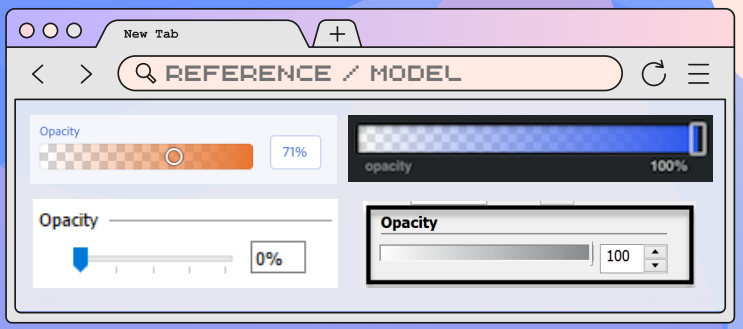
FILE NAME: LINETOTOOL.JS

File Edit View Help

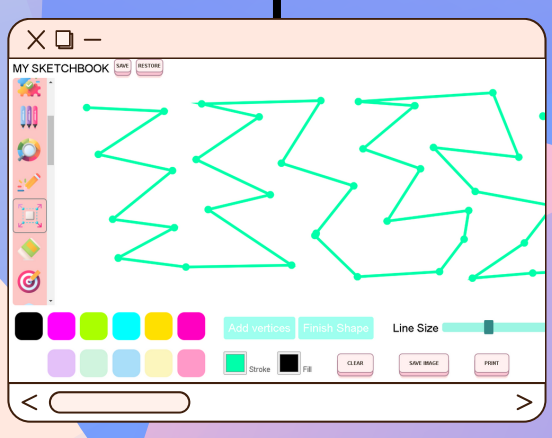
Inside **LineToTool():** I decided to implement inside the populateOptions() function a slider, which will allow to modify the width of the line. In the **draw()** function, the value returned by the "x" slider set will be within the parameter required for the strokeWeight(x) function. Likewise, we will call the colorPicker by means of its ID. In this way, the color of the line will depend on the color that is selected in the colorPicker labeled "Stroke" next to it.

FILE NAME: LINETOTOOL.JS

----FUTURE IMPLEMENTATIONS/CHANGES----
A slider will be implemented that allows changing the opacity of the color for the line.



EDITABLE



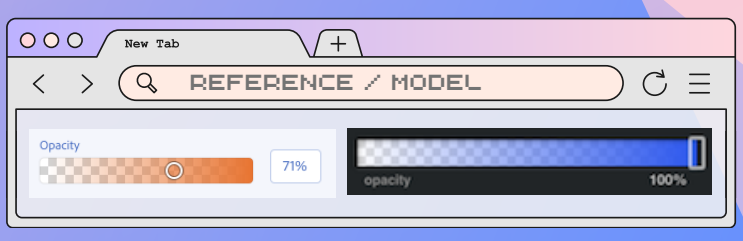
FILE NAME: EDITABLESHAPETOOL.JS

File Edit View Help

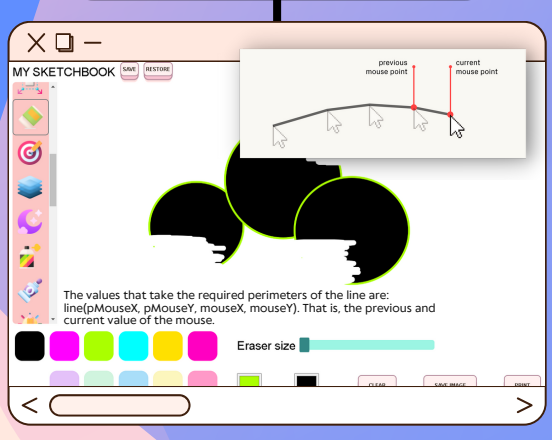
This implemented function allows drawing free paths in which each vertex can be modified. To do this, an array was created that stored all the previous x,y coordinates that were drawn on the canvas when the mouse was pressed. A loop has been created that draws all the verties on the canvas as they are added to the array. When the edit mode is activated, a loop is created with the length of the array that will allow to identify the current distance of the mouse with each vertex in the figure drawn, at a certain distance the coordinate of the mouse will be equal to that of the chosen vertex. In this way, the vertex will be able to move and change position.
I decided to implement within the **populateOptions()** function, a button that would activate the edit mode and another button that would allow the finalization of the figure drawn, I also added a slider which will return a value that will later be returned to the **draw( )** is reflected and allows you to modify the width of the line. Also, the color of the line is congruent with the color that is selected in the ColorPicker called by its ID in the **draw()** function.

FILE NAME: EDITABLESHAPETOOL.JS

----FUTURE IMPLEMENTATIONS/CHANGES----
A slider will be implemented that allows changing the opacity of the color for the line.



ERASER



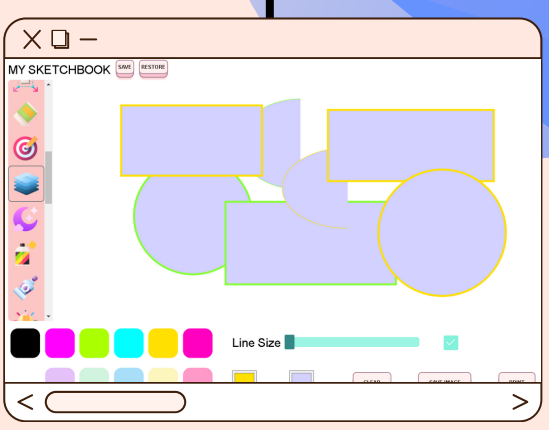
FILE NAME: ERASERTOOL.JS

File Edit View Help

MyEraserTool() allows you to remove errors made by simulating a traditional eraser. To do this, with the help of the line() function, a white line was created, which is drawn every time the mouse is pressed on the canvas. On the other hand, within the **populateOptions()** function, a slider was created which will not allow the width of the line to be modified. Then inside the draw() function we will obtain the value that the slider returns. In addition, the model of the cursor has been changed, its size will increase depending on the value of the line width.

GATEGORY: SHAPES

SHAPES



----FUTURE IMPLEMENTATIONS/CHANGES----Add a button that allows you to turn fill() or noFill() on and off. In this way, the figures will have filling or not.

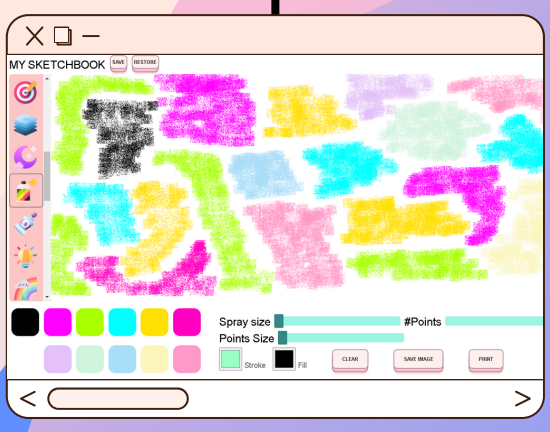
FILE NAME: MIDMOONTOOL.JS CIRCLECANTOOL.JS RECTANGLETOOL.JS

File Edit View Help

CircleCanTool() allows you to draw ellipses.
RectangleTool() allows you to draw rectangles.
MidMoonTool() allows you to draw half moons.
For all three functions:
Within the **populateOptions()** function, a checkbox was implemented to enable and disable the noStroke mode. Inside the **draw()** function: if the noStroke mode is on then the figure will have no Stroke, if it does, the color will be congruent to the color selected from the ColorPicker with a "Stroke" tag next to it. On the other hand, the fill color of the shape will be congruent to the color selected from the ColorPicker with the "fill" label.

GATEGORY: SPRAY

S P R A Y 1

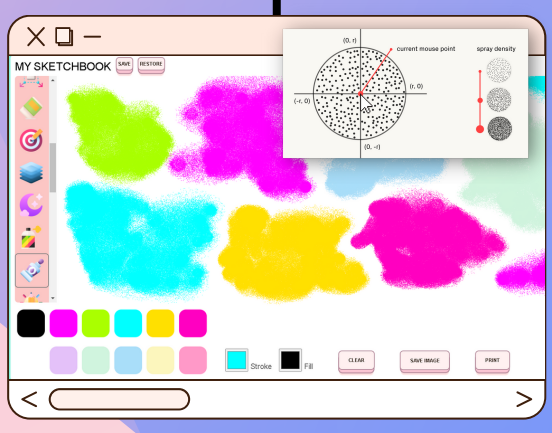


FILE NAME: SPRAYCANTOOL.JS

File Edit View Help

SprayCanTool() is the tool that was implemented to simulate a traditional aerosol. In the **populateOptions()** function, three sliders and three labels were created, each of them indicating the functionality of the slider on its right side. The first slider will return a new value for the size of the spray, the second slider will return a new value for the number of dots created along the stroke, and the third slider will return a new value for the width of the dots border. The **draw()** function will call the values returned by the sliders. Which are responsible for setting new values for the operation of the spray when the mous is pressed inside the canvas. Finally, the color of the points will be congruent to the color selected in the colorPicker labeled "Stroke".

S P R A Y 2



FILE NAME: SPRAYPINCELTOOL.JS

File Edit View Help

SprayBrushTool() allows you to paint the canvas simulating the appearance of a traditional spray. To do this, random points will be drawn inside a circle around the mouse position, the range of this circle will depend on how fast the mouse moves across the canvas.

Inside the **pintura()** function: We identify the "speed" at which the mouse moves. Which will be the sum of integers between the subtraction of the previous and current "x" and "y" coordinates of the mouse: **speed = abs(mouseX - pmouseX) + abs(mouseY - pmouseY)**. Then we identify the radius of our circle, which will be the speed at which the mouse moves plus a certain value. Finally, the diameter will be the radius squared. We create a loop, which allows to find random coordinates in the interval of the diameter around the mouse position. However, the points found are still far apart from each other and we want to achieve something denser and smoother, so this loop will be a nested loop. To do this, we use the lerp() function. This outer loop will allow you to create denser strokes.

Inside the **draw()** function:

Every time we press and move the mouse over the canvas, the functionality of the pintura() function will be reflected on the canvas. In this way, a drawing similar to that of a traditional aerosol will be traced.

FILE NAME: SPRAYPINCELTOOL.JS

----FUTURE IMPLEMENTATIONS/CHANGES----

Implement two sliders; one for the color density and one for the size of the radius. On the other hand, to make the color of the line equal to the color selected in the colorPicker, it will return the hexadecimal value of the color selected in the colorPicker that has a "Stroke" tag next to it. Thus, the stroke() will contain the value returned by the colorPicker

GATEGORY: DECORATION

W A T E R C O L O R

FILE NAME: WATERCOLOURTOOL.JS

File Edit View Help

WatercolorTool() allows you to draw paths that simulate a traditional watercolor. In the draw() function, a loop was created that is repeated three times, inside this loop we will use the translate() function and we will translate the origin, which by default is (0,0), to the current coordinate of the mouse. Next, we will create another loop, in this case nested to the previous one. This loop will draw vertices around the origin. In this way, we can obtain a figure in the shape of a circle or the closest thing to one. Then, in the first loop we will use the function rotate(random(PI*2)), so that each of the three figures created is rotated randomly and creates a watercolor effect. Well, there are two options, that this path has the appearance that it was diluted in water or that it is completely filled and dense. For the first appearance the color will be congruent to the selected color of a new colorPicker, each selected color will be modified, which will create a soft and slightly transparent effect. If we want the appearance to be more dense and filled, then we will use a button to deactivate the "smooth" mode and automatically activate the "full fill". So, the fill and border color will be congruent to the colorPicker which has a "Stroke" label next to it.

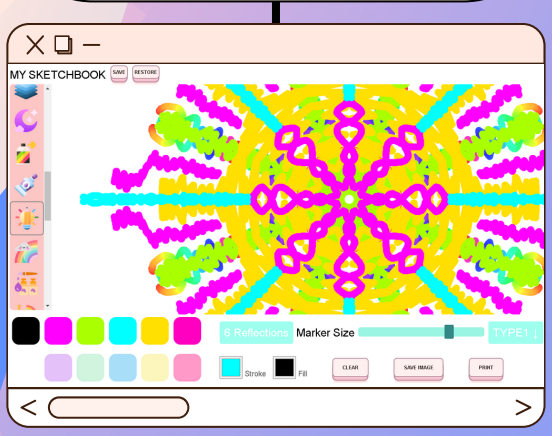
New Tab

IMPORTANT

FILE NAME: MULTICOLORLINE.JS

MulicolorLineTool() allows you to draw multicolor paths.
MulicolorLineTool() has the same implementations as WatercolorTool(), except for the checkbox that allows "smooth" mode. Also, unlike WatercolorTool(), MulicolorLineTool() has another color mode that allows you to create a multicolor appearance. To do this, we use the colorMode(HSB) function and set stroke((5*frameCount) % 360, 80, 140).

M A N D A L A



FILE NAME: MANDALA.JS

File Edit View Help

Mandala() allows you to draw reflective lines together that simulate a kaleidoscope or mandala. The reflection point is the center of the screen. Within the mandala() function, a loop was created that has the function of drawing two symmetrical and reflective lines between them for each number of reflections, the total number of reflections by default is 8. In the populateOptions() function, three buttons were created and a slider. The slider will return a new value that will modify the width of the lines. The first button will return a value that will change the number of reflections from 8 to 6 or vice versa. The second and third buttons will allow you to change the color mode of the lines. Then, in the draw() function, all of these values are called by their IDs, so they can modify the width and color mode of the lines. In addition, the mandala() function is also called so that every time the mouse is clicked and moved around the canvas it works and allows us to draw. Finally, the color of the lines will be the value returned by the colorPicker that has a "Stroke" label next to it via the ID assigned to it.

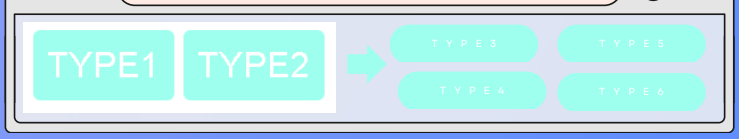
FILE NAME: MANDALA.JS

----FUTURE IMPLEMENTATIONS/CHANGES----

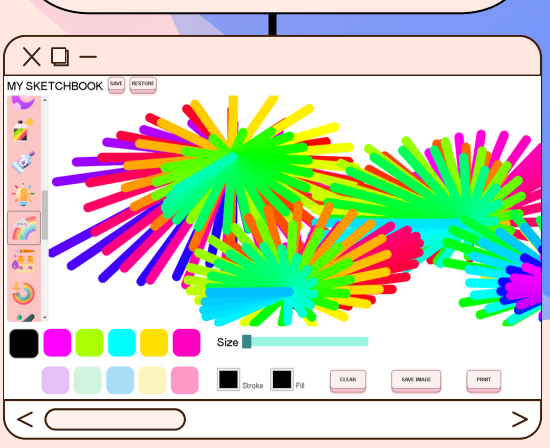
At the moment there are only two styles, but more styles will be added.

New Tab

REFERENCE / MODEL



R A D I A L



FILE NAME: RAINBOWRADIAL.JS

File Edit View Help

RainbowRadialTool() allows us to draw multicolored lines that simulate a rainbow. To do this, we'll need the line() function. So, we set line(start_X, start_Y, mouseX, mouseY). Where the start coordinate will always be the same, to give a more abstract detail to the figure. Also, the color mode for the line will be changed to HSB (hue, saturation, and brightness). This creates a rainbow effect. A rainbow gradient from left to right. In the populateOptions() function we will create a slider that will return an "x" value. This "x" value will be called in the draw() function and then put in strokeWeight(x) to modify the width of the line. Finally, when we press on the canvas we can draw this abstract rainbow.

FILE NAME: RAINBOWRADIAL.JS

----FUTURE IMPLEMENTATIONS/CHANGES----

A selector will be created to change the style of the line.

New Tab

REFERENCE / MODEL

RAINBOW

FILE NAME: RAINBOWBRUSHTOOL.JS

File Edit View Help

RainbowBrushTool() has the same behavior as freehandTool(). But unlike freehandTool(), RainbowBrushTool() has another color mode: colorMode(HSB).Where HSB(Hue,Saturation,Brightness): H = number between 0 and 360. S = number between 0 and 100. B = number between 0 and 100. So, stroke((5*frameCount) % 360,40, 100) . Also, we'll use "frameCount" which tells us how many frames have passed so far, or in other words, the total number of calls to . If we divide by 360, then the remainder will always be some number from 0 to 360.

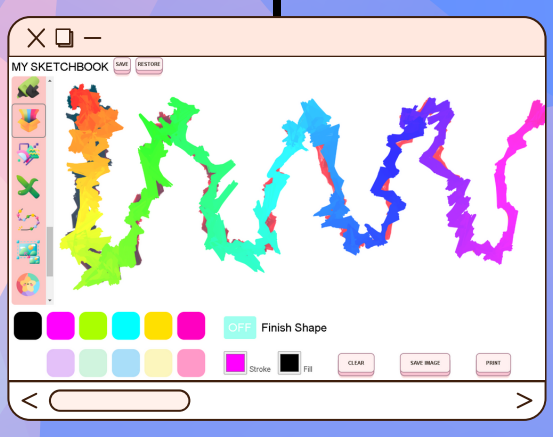
HIGHLIGHTER

FILE NAME: HIGHLIGHTERTOOL.JS

File Edit View Help

HighLightTool() allows you to draw paths that have the appearance of a traditional highlighter. We use the line() function to draw basic lines. Next, we will use the strokeCap() function to define the style of the caps of the lines. So, we will define strokeCap(SQUARE). This way, the beginning of the line and again at the end will be square, giving us a highlighter look. The default color of the line will have a low intensity, to have a high intensity we will create a checkbox in the populateOptions() function. that allows to deactivate "smoothMode" and raise the intensity. Likewise, in the function populateOptions() a slider was created and from the function draw() we will call the value obtained from the slider and we will modify the width of the line according to the value of the slider.

AUTORAINBOW



FILE NAME: AUTORAINBOWTOOL.JS

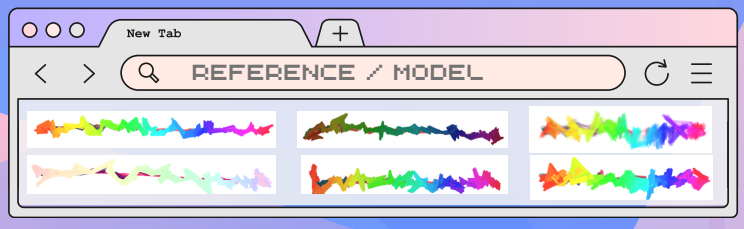
File Edit View Help

AutoRainbowTool() allows you to draw irregular multicolor vibrant lines. To do this, we will change the color mode to HSB. We will make a matrix where as the mouse moves across the canvas, the coordinates will be added. Then a rainbow() function where we will create a loop to go through the points of the matrix. The tone of the lines will depend on how far away the line is. Well, this loop will allow us to make overlapping lines randomly between the previous and current coordinates. These lines will be drawn automatically so if we want them to stop drawing we will use an "OFF" button which will allow all the coordinates you draw on the canvas to remain and you can start from scratch another shape or path.

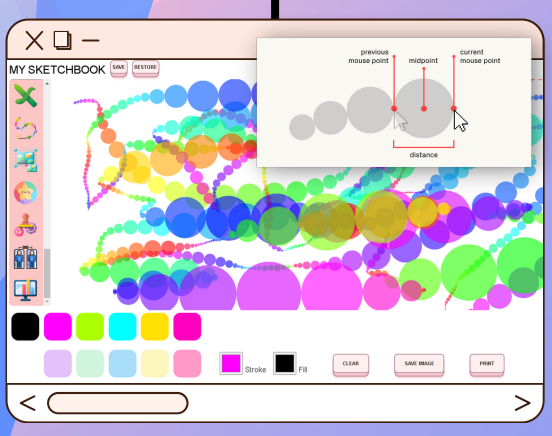
FILE NAME: AUTORAINBOWTOOL.JS

----FUTURE IMPLEMENTATIONS/CHANGES----

Create three sliders; one to modify the opacity of the color and another to modify the intensity and the last one to modify the softness



SHAPEBRUSH



FILE NAME: SHAPEBRUSHTOOL.JS

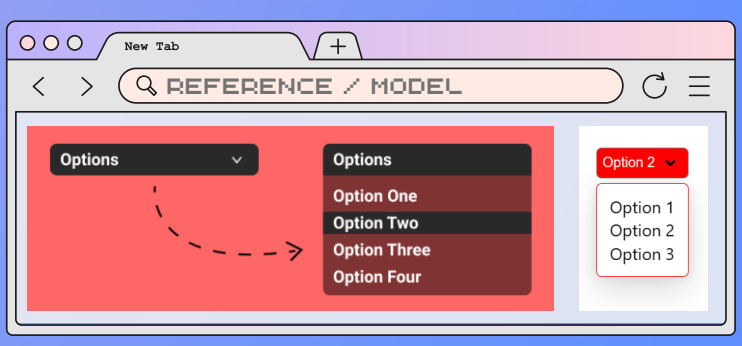
File Edit View Help

ShapeBrushTool() allows you to create paths with multicolored circles together. So, in the rainbowShapes() function:We need to find the distance between the current and previous coordinates of the mouse. Next, we'll find the midpoint between the current and previous mouse points. And finally, we'll use these values to draw a circle: circle(midX, midY, distance). On the other hand, we will change the hue of the circles to give it a multicolored appearance. To do this, we will set the color mode to HSB. Then stroke(frameCount * 10) % 360 which tells us how many frames have passed so far, or in other words, the total number of calls to . If we divide by 360, then the remainder will always be a number from 0 to 360. After all that, in the draw() function we will call the rainbowShapes() function and it will be ready to use when the mouse is clicked on the canvas.

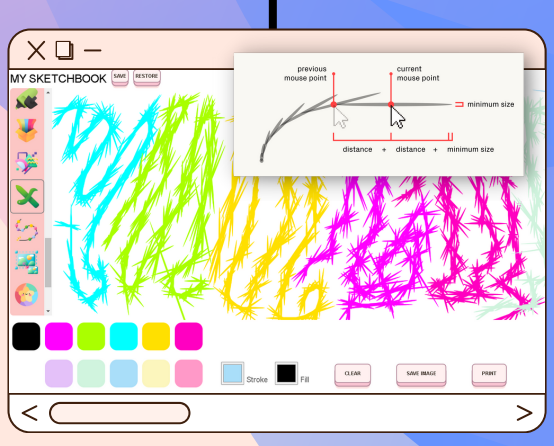
FILE NAME: SHAPEBRUSHTOOL.JS

----FUTURE IMPLEMENTATIONS/CHANGES----

At the moment the figures drawn on the canvas can only be circles. Therefore, I will implement the option that rectangles can also be made. To do this, I will create a selector with two options:Rectangles and Circles. In addition, so that the figures are only one color and not multicolored there will be a button. With which, the fill and border color will be equal to the color selected in the colorPicker.



BLADES



FILE NAME: BLADESTOOL.JS

File Edit View Help

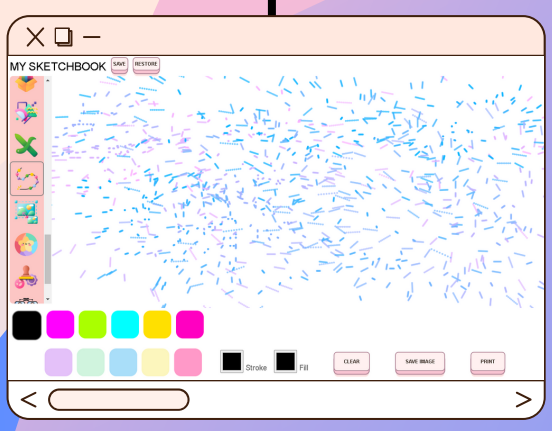
BladesTool() allows you to create jagged-looking paths that simulate blades. This function has the same implementations as the CrayonTool() function mentioned above, but unlike CrayonTool(), BladesTool() has the width of the ellipse more extended, allowing it to simulate the shape of a blade.

FILE NAME: BLADESTOOL.JS

----FUTURE IMPLEMENTATIONS/CHANGES----

Add two sliders; one to modify the thickness of the line and another for the opacity of the color of the line. On the other hand, to make the color of the line equal to the color selected in the colorPicker, it will return the hexadecimal value of the color selected in the colorPicker that has a "Stroke" tag next to it. Thus, the stroke() will contain the value returned by the colorPicker

POINTS



FILE NAME: POINTSTOOL.JS

File Edit View Help

PointsTool() simulates paint splatters. To do this we create a loop and with the help of the lerp() function we find coordinates between the current and previous mouse position. Finally, in these coordinates points will be drawn. For now, I set the color mode to HSB. To create multi-colored dots as the mouse moves across the canvas.

FILE NAME: POINTSTOOL.JS

----FUTURE IMPLEMENTATIONS/CHANGES----

A slider will be added that allows modifying the opacity of the line. Also, two buttons will be added; one that will allow you to change the color style of the line and one that will allow all points drawn to be only one color. To do this, when this last button is pressed, the color of the points will be equal to the color selected in the colorPicker.

CATEGORY: EXTRAS

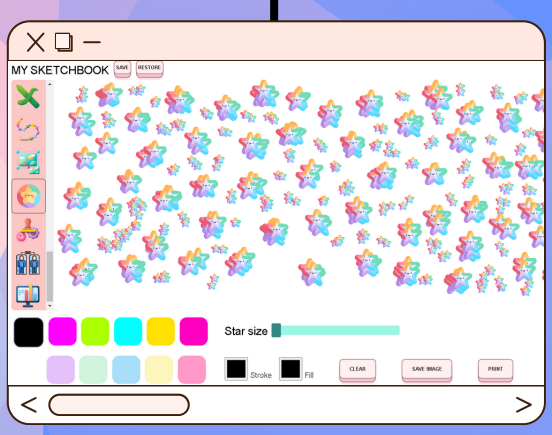
ADDIMAGE

FILE NAME: ADDIMAGETOOL.JS

File Edit View Help

AddImageTool() allows you to add images to the canvas. To do this, a graphics buffer was first created with the createGraphics() function. Next in the populateOptions() function: we'll use createFileInput() to select local files, in this case images, so they can be viewed on the canvas. The handleFile() function will recognize if the added document is an image, if so an image will be created. Then, in the draw() function with the help of the image() function we will add the image to the canvas. We will also create two buttons, the first to remove the images completely from the canvas and the second to save the added image on the canvas and add a new one. In order to save the image already added to the canvas, inside the draw() function we add all the pixels in an array. Which will remain on the canvas when the "Finish Shape" button is pressed.

STARS



FILE NAME: STARTOOL.JS

File Edit View Help

StarTool() allows you to add a star as if it were a stamp on the canvas. To do this, we will add an image and in the draw() function we will randomly center its position around the mouse as it moves on the canvas. Also, in the populateOptions() function we will create a slider. Then in the draw() function we will call the value obtained by the slider and with this we will modify the size of the stamp.

STICKERS

FILE NAME: STICKERTOOL.JS

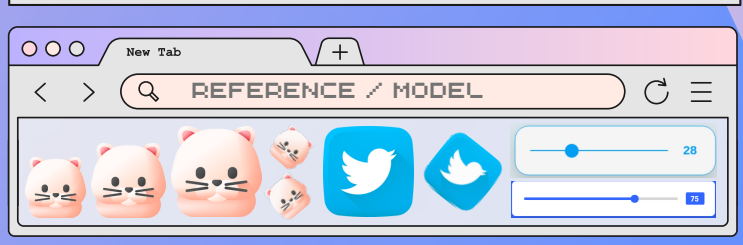
File Edit View Help

StickerTool() allows you to select images and then add them to the canvas, simulating that they are stickers. To do this, first an array will be created where all the names of the stickers will be added and one of the stickers will be selected by default, calling it from its local path, this sticker will be the main sticker by default. Afterwards, a loop will be created that will go through all the names inside the array. Inside this loop we will get each image from its local path, so what we will do is join the main folder "assets/stickers/" with each name inside the array + ".png".... Then, when we click on any of these images, the image will be loaded and it will be possible to add it to the canvas. Finally, in the draw() function an image() image will be created with the selected sticker and it will be added to the canvas, its position will depend on the current mouse position.

FILE NAME: STICKERTOOL.JS

----FUTURE IMPLEMENTATIONS/CHANGES----

Two sliders will be added; the first will allow to change the size of the sticker and the second will allow to rotate the sticker.



HELPER FUNCTIONS / OTHERS

FILE NAME: HELPERFUNCTIONS.JS / SKETCH.JS / ETC

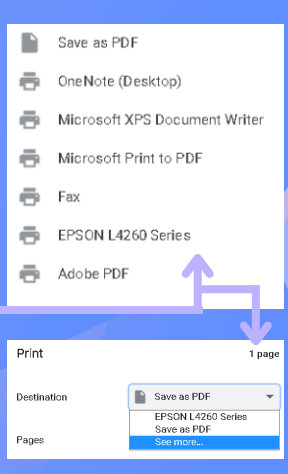
File Edit View Help

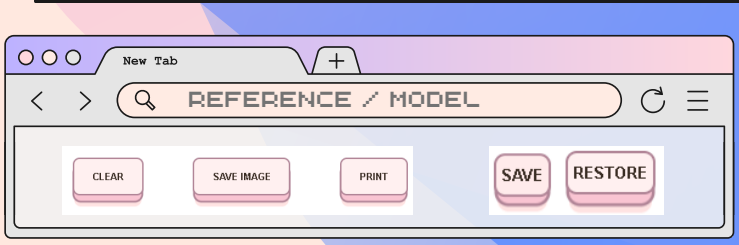
CHANGES MADE

- Added p5.pdf.js inside the lib folder. This file is based on the browser's print to pdf function. When the PRINT button is pressed, we can:
- All the buttons and the interface were stylized.
- In the setup() function within sketch.js, two buttons were added:

SAVE is used to copy all the current pixels of the canvas to a graphic **BUFFER**. After that, if any changes are made to the canvas and we are not satisfied with it, we can press **RESTORE** before press **SAVE** and automatically all the pixels previously copied from the canvas to the BUFFER will be copied back to the canvas. This will allow the last changes made that were not saved, with which we were not satisfied, to be removed. Pressing RESTORE is equivalent to pressing **Ctrl+Z** on the keyboard.

Important: Any bugs present regarding the SAVE and RESTORE buttons will be fixed in the coming weeks.





OTHER IMPLEMENTATIONS

FILE NAME: CLONESHAPETOOL.JS

A new tool CloneShapeTool() will be implemented. Which will simulate a clone of a region of the canvas to a new area. To do this, we will use the copy() method. Which, copies a region of pixels from one part of the canvas to another. In this way, when shift is pressed and the mouse is on the canvas, a rectangle will be drawn around the mouse position, this rectangle will represent the initial position to be cloned. In such a way, that when you stop pressing shift and just move the mouse over the canvas, the pixels will begin to be copied to a new area of our choice, from that initial position that we chose.

MoonDark