Draw App

Mariya Peeva (mpeev1)

Features

Features included in the draw app:

- rectangle / ellipse tool^{+}
    - with a fill or outline selector^{++}
- stamp tool^{+}
- eraser^{+}
- load button^{++}
- star trail^{++}
- scissors tool^{+++}
- spiragraph tool^{+++}
    - grid options 54, 36, 18 and 9
- flood fill^{+++}
    - colour replacement/fill
- background and foreground colours^{+++}
    - colour sampler

All features are included as constructor functions and added as new objects to the Toolbox() object. All constructors have properties this.icon, pointing to the file path of their icon, and this.name, which contains their name.

Rectangle Tool and Ellipse Tool

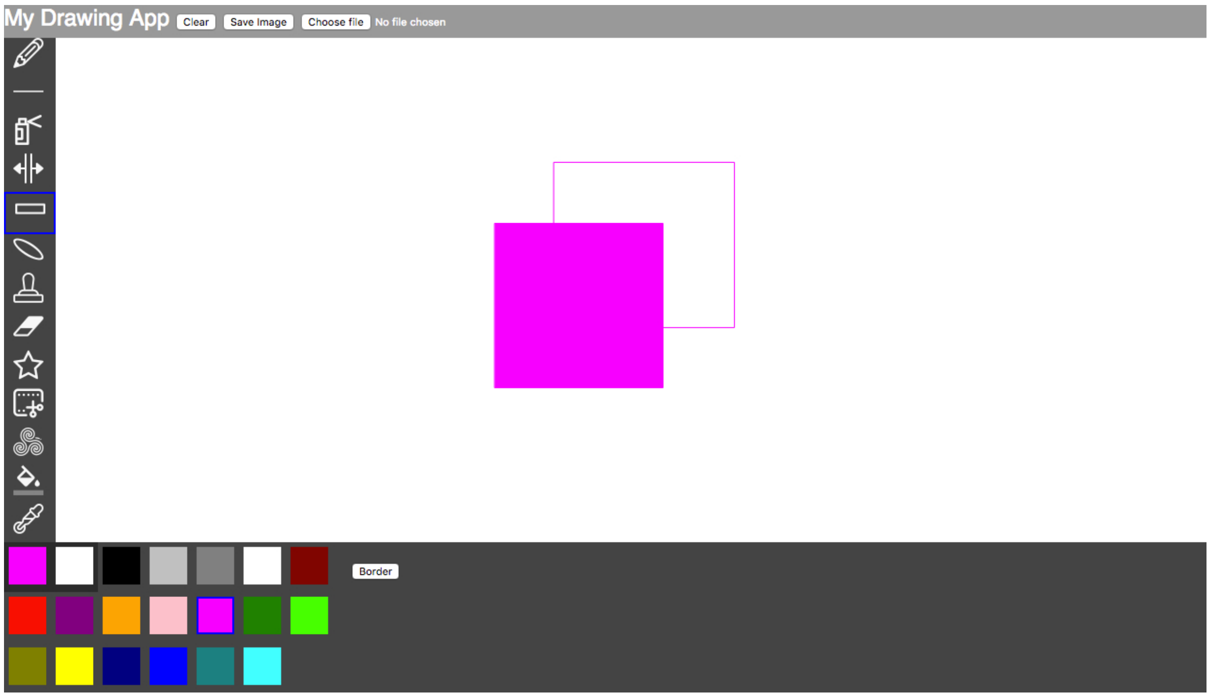


Figure 1 Rectangle Tool

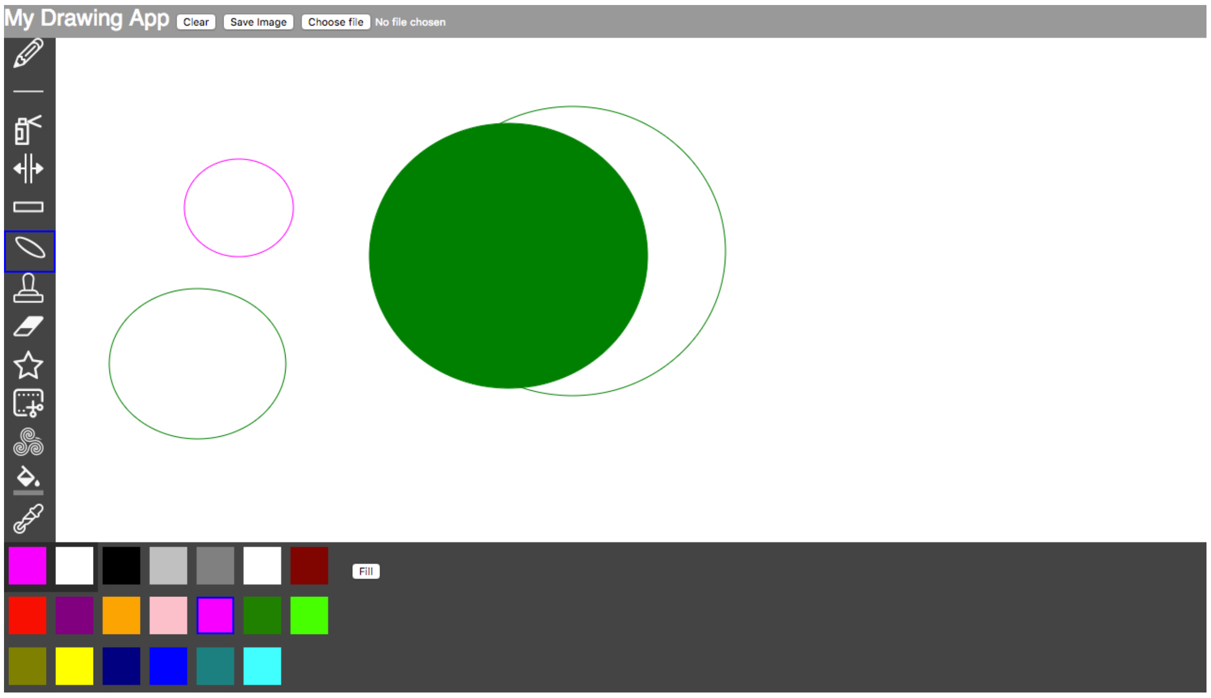


Figure 2 Ellipse Tool

The Rectangle (fig 1) and Ellipse (fig 2) Tools draw a rectangle or an ellipse respectively on the canvas. startMouseX and startMouseY are set to an off-canvas value. “this” assigned self for a closure. Both tools have a this.mode property, which can be set to “fill” or “border”. The draw() function features a “three-layered” structure for mouse press, drag and release events. On mouse press, startMouseX and startMouseY store the position of the mouse and drawing is set to true. A call to loadPixels() loads pixel data from the display window into the pixels[] array before drawing. Within the “mouse drag layer”, a call to updatePixels() loads new pixel data into pixels[]. A rectangle/ellipse is drawn within push() and pop(). If mode is set to “border”, the fill is set to noFill(). A rectangle is then drawn with x and y of startMouseX, startMouseY, width of (mouseX – startMouseX) and height of (mouseY – startMouseY), where mouseX and mouseY are the x and y of the mouse while dragging. If the mouse is not pressed, drawing is reset to false and startMouseX and startMouseY are reset to an off-canvas value.

Stamp Tool

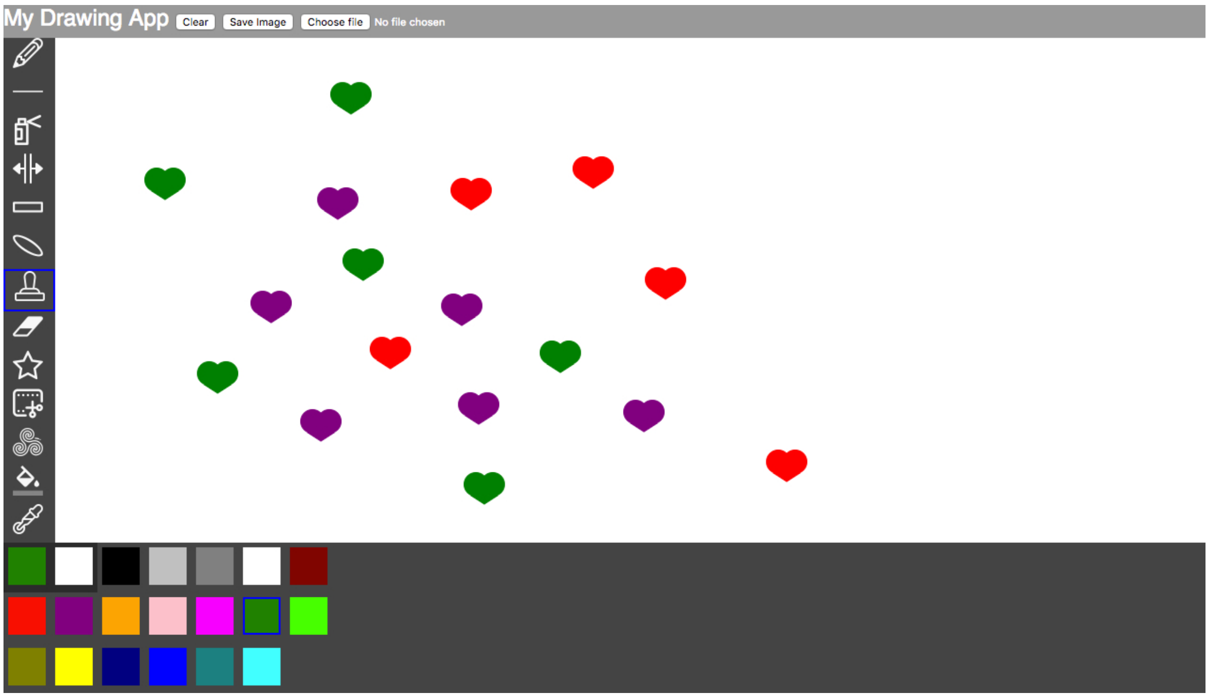


Figure 3 Stamp Tool

The Stamp (fig 3) Tool draws a heart stamp on the canvas. startMouseX and startMouseY are set to an off-canvas value and drawing is set to false. The draw() function features a “three-layered” structure for mouse press, drag and release events. On pressing the mouse, if the position of the mouse is off-canvas, startMouseX and startMouseY are set to the position of the mouse and drawing is set to true. Drawing occurs within the first layer. A heart stamp is drawn relative to startMouseX and startMouseY from three ellipses and a triangle. A call to loadPixels() loads pixel data into pixels[] after drawing. Within the “mouse drag layer”, updatePixels() loads new pixel data into the pixels[]. If the mouse is not pressed, drawing is reset to false to terminate the drawing process and variables startMouseX and startMouseY are reset to an off-canvas value.

Eraser Tool

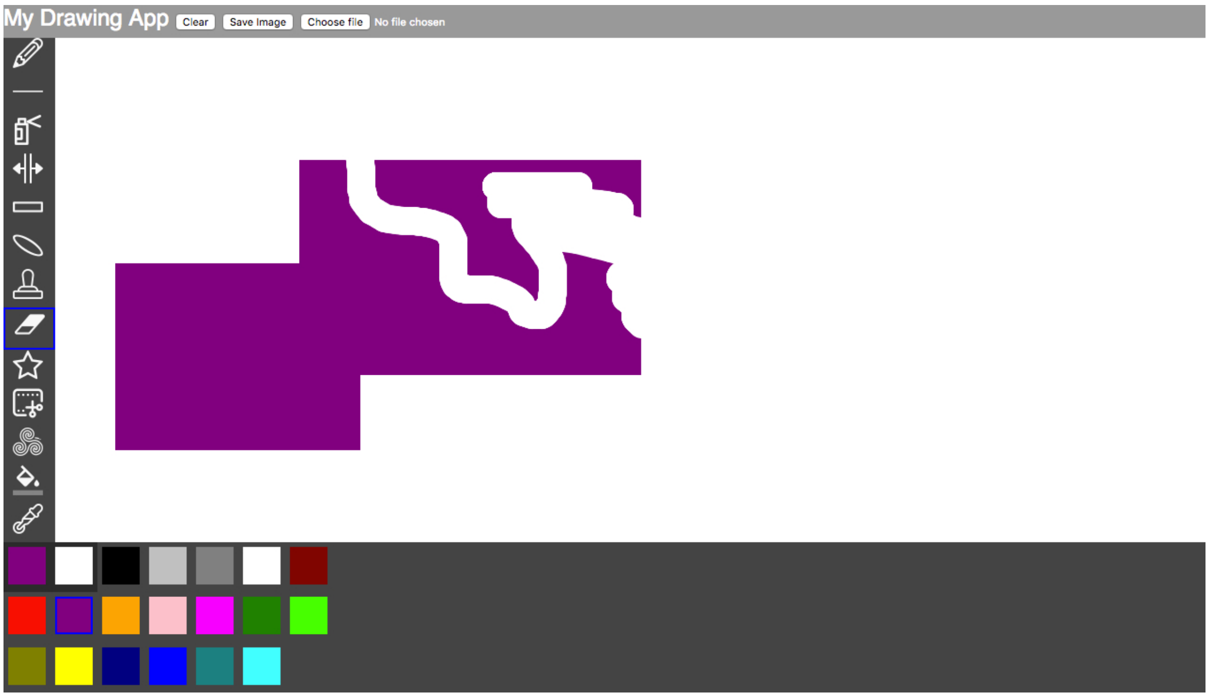
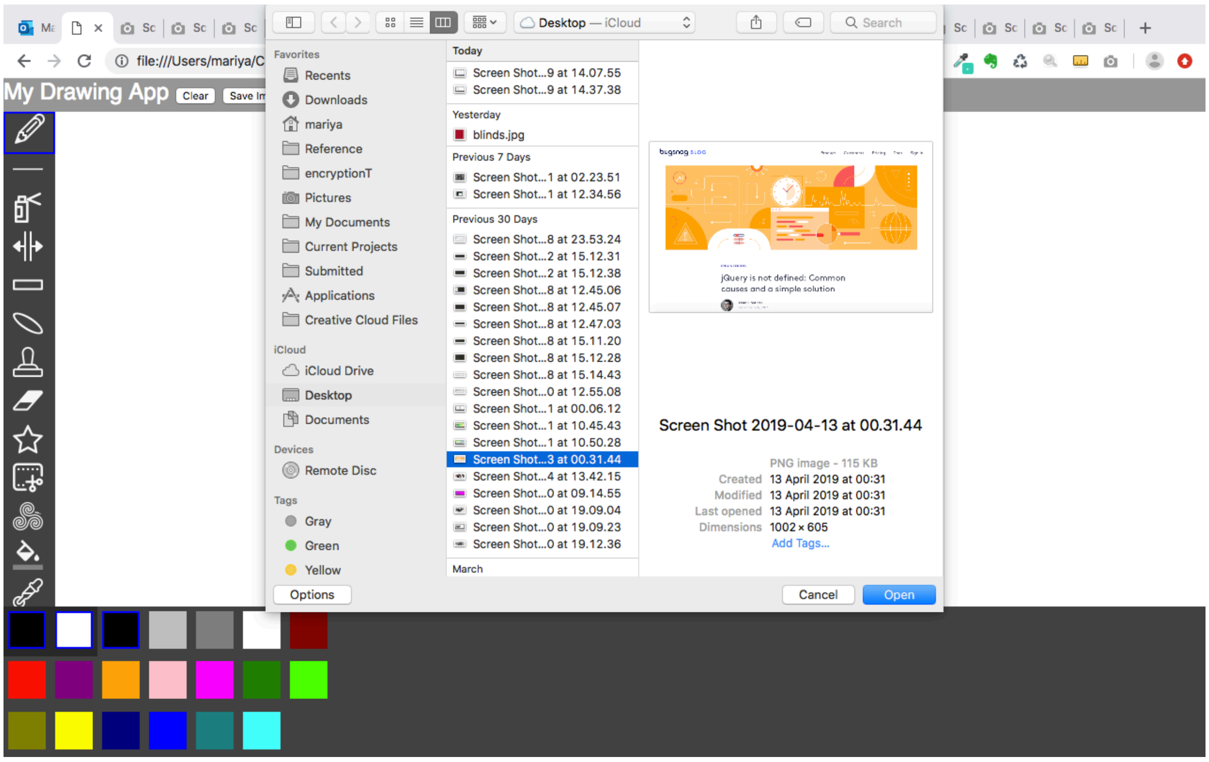


Figure 3 Eraser Tool

The eraser (fig 4) tool “erases” colour by drawing a white thick line on mouse drag. previousMouseX and previousMouseY are set to an off-canvas value. The draw() function features a “three-layered” structure. On pressing the mouse, if its position is off-canvas, previousMouseX and previousMouseY are set to store the mouseX and mouseY. Drawing occurs within the “mouse drag layer”. A line is drawn within push() and pop() with a white stroke and strokeWeight of 30 from previousMouseX and previousMouseY to mouseX and mouseY. previousMouseX and previousMouseY are then set to the new position of the mouse. If the mouse is not pressed, a third layer resets previousMouseX and previousMouseY back to their initial state.

Load File Button


file:///Users/mariya/C
My Drawing App
Clear
Save Im
Desktop — iCloud
Search
Favorites
Recents
Downloads
mariya
Reference
encryptionT
Pictures
My Documents
Current Projects
Submitted
Applications
Creative Cloud Files
iCloud
iCloud Drive
Desktop
Documents
Devices
Remote Disc
Tags
Gray
Green
Yellow
Options
Today
Screen Shot...9 at 14.07.55
Screen Shot...9 at 14.37.38
Yesterday
blinds.jpg
Previous 7 Days
Screen Shot...1 at 02.23.51
Screen Shot...1 at 12.34.56
Previous 30 Days
Screen Shot...8 at 23.53.24
Screen Shot...2 at 15.12.31
Screen Shot...2 at 15.12.38
Screen Shot...8 at 12.45.06
Screen Shot...8 at 12.45.07
Screen Shot...8 at 12.47.03
Screen Shot...8 at 15.11.20
Screen Shot...8 at 15.12.28
Screen Shot...8 at 15.14.43
Screen Shot...0 at 12.55.08
Screen Shot...1 at 00.06.12
Screen Shot...1 at 10.45.43
Screen Shot...1 at 10.50.28
Screen Shot...3 at 00.31.44
Screen Shot...4 at 13.42.15
Screen Shot...0 at 09.14.55
Screen Shot...0 at 19.09.04
Screen Shot...0 at 19.09.23
Screen Shot...0 at 19.12.36
March
jQuery is not defined: Common causes and a simple solution
Cancel
Open

Screen Shot 2019-04-13 at 00.31.44

PNG image - 115 KB

Created	13 April 2019 at 00:31
Modified	13 April 2019 at 00:31
Last opened	13 April 2019 at 00:31
Dimensions	1002 × 605

Add Tags...

A load image button is added to the HelperFunctions() object. An input element with id "loadButton" and type "file" is added as a child of ".header" in index.html. In helperFunctions.js, a jQuery on() "click" method is added to the element "#loadButton" with a handler function(). Within the function(), variable file is set to this.files[0]. this.files[0] is an object with an src property set to the output of the static method URL.createObjectURL(object), where object is this.files[0].
URL.createObjectURL(this.files[0]) creates a DOMString with the object's URL.
The p5.js method loadImage(path, [successCallback]) is then called with path file.src and [successCallback] of img, which loads the image img, positioning it at the top left corner of the canvas.

Star Trail Tool

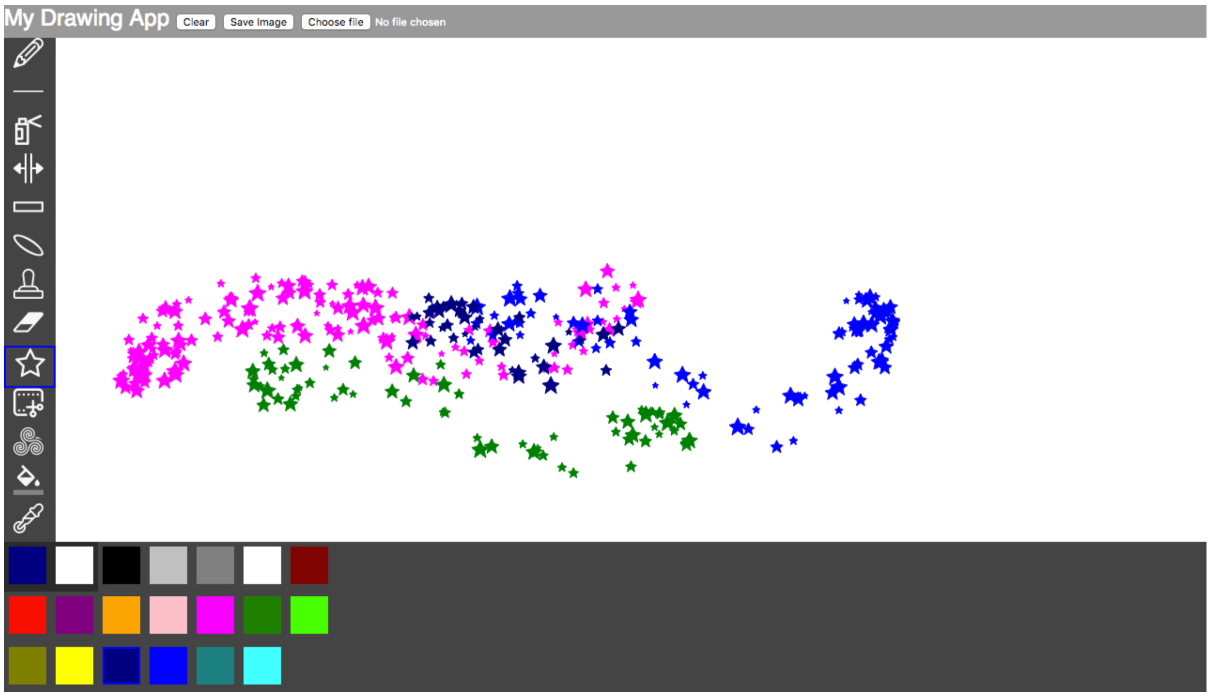
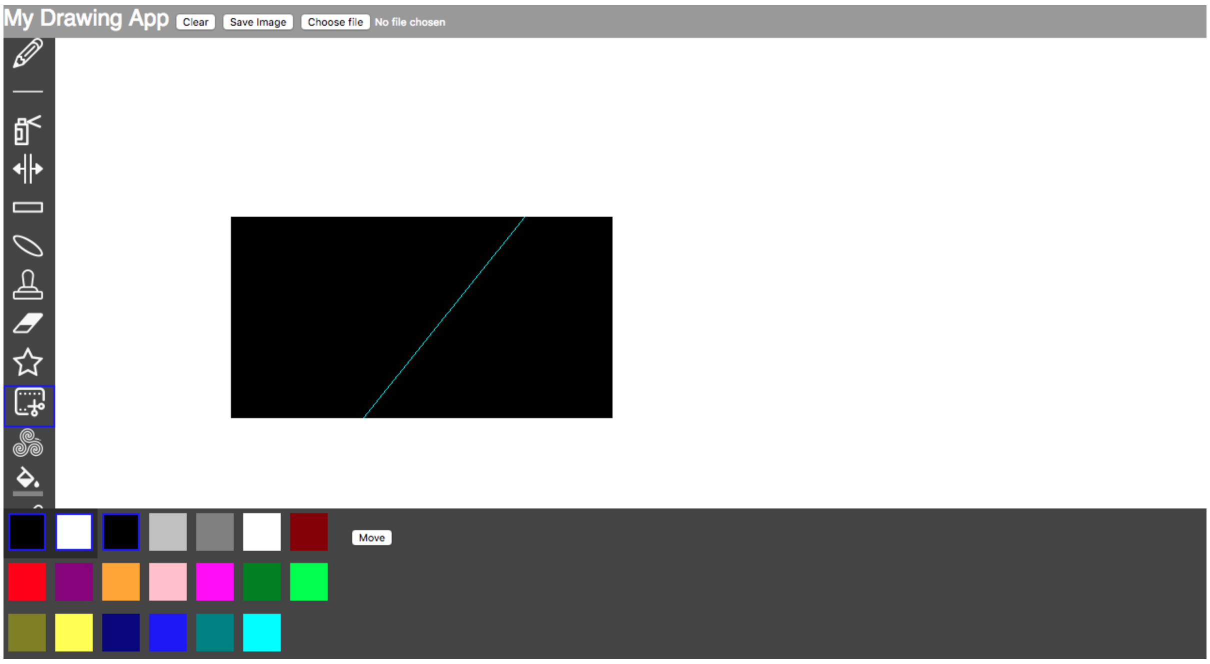
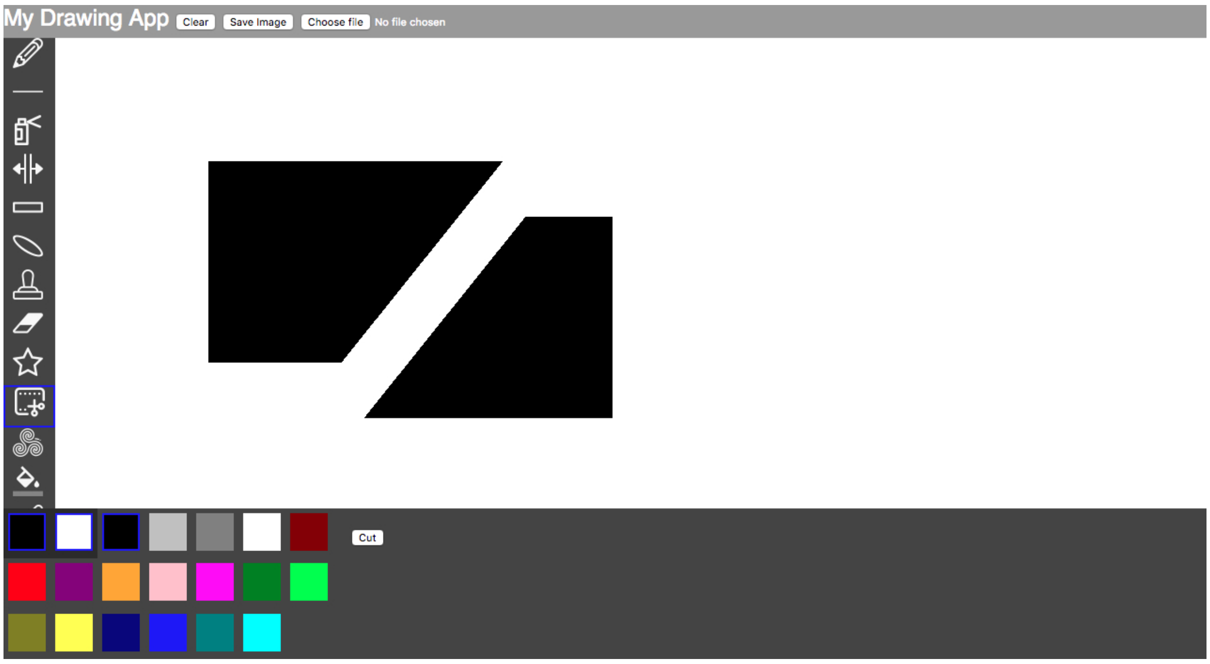


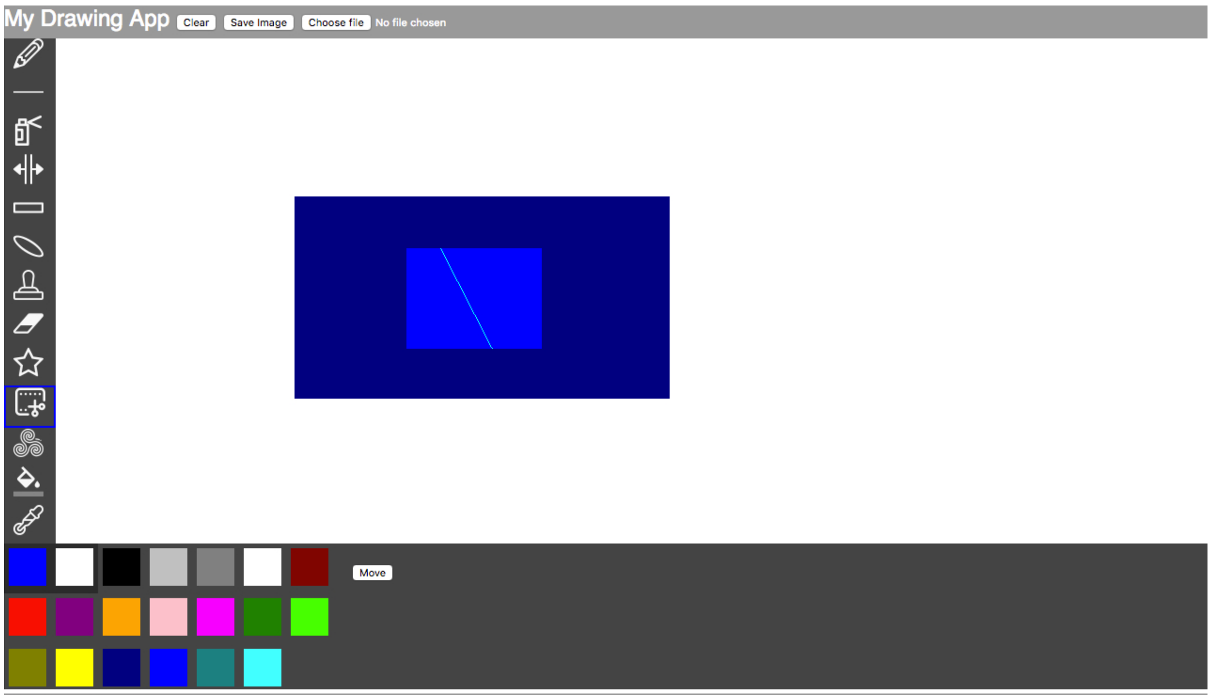
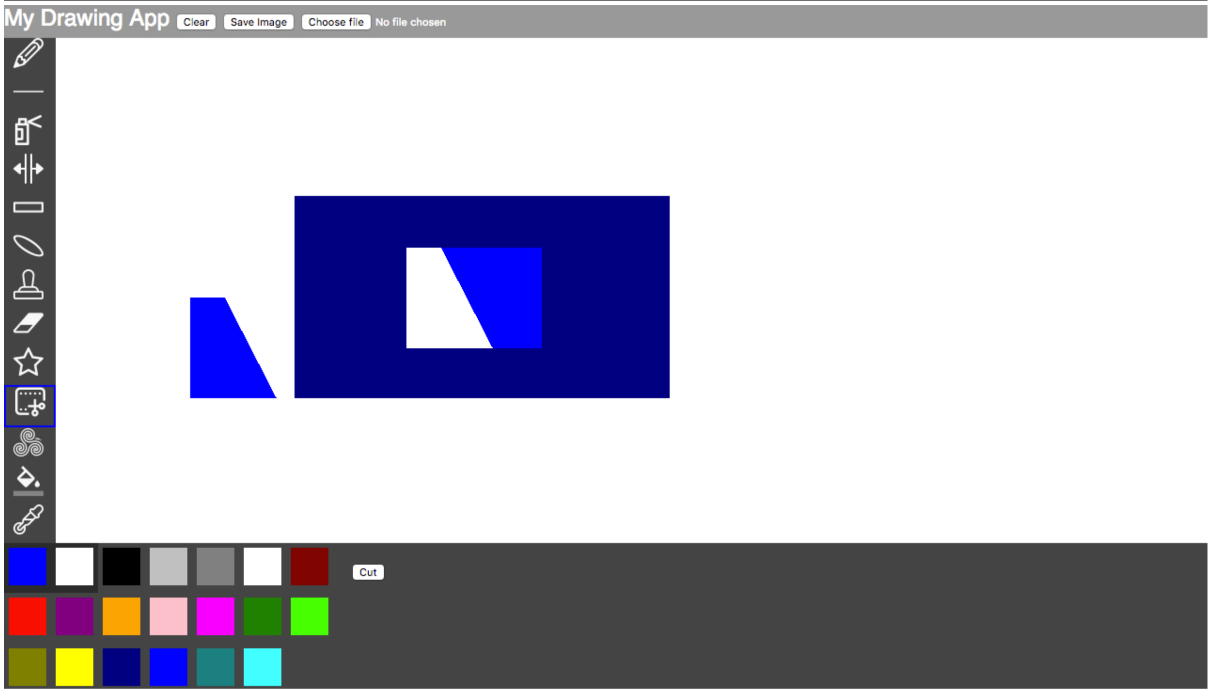
Figure 4 Star Trail

The Star Trail (fig 5) tool “sprays” stars on the canvas. Properties this.stars, this.spread and this.scale are set to the number of stars per frame on mouse press, the maximum distance from the mouse the stars will be positioned in and the minimum/maximum values for scaling. this.stars is set to 2, this.spread to 30 and this.scale is a min 0.1 and a maximum of 0.3. The draw() function features a “one-layer” structure to include events on mouse drag. If the mouse is pressed, 2 stars are drawn per frame within a for loop. For each star, starX and starY are set to random values within a spread of 30px and scale is set to a random value from this.scale[0] to this.scale[1]. A star shape is then drawn on the canvas using vertices.

Scissors Tool & Flood Fill Tool






My Drawing App
Clear
Save Image
Choose file
No file chosen
Move

My Drawing App
Clear
Save Image
Choose file
No file chosen
Cut

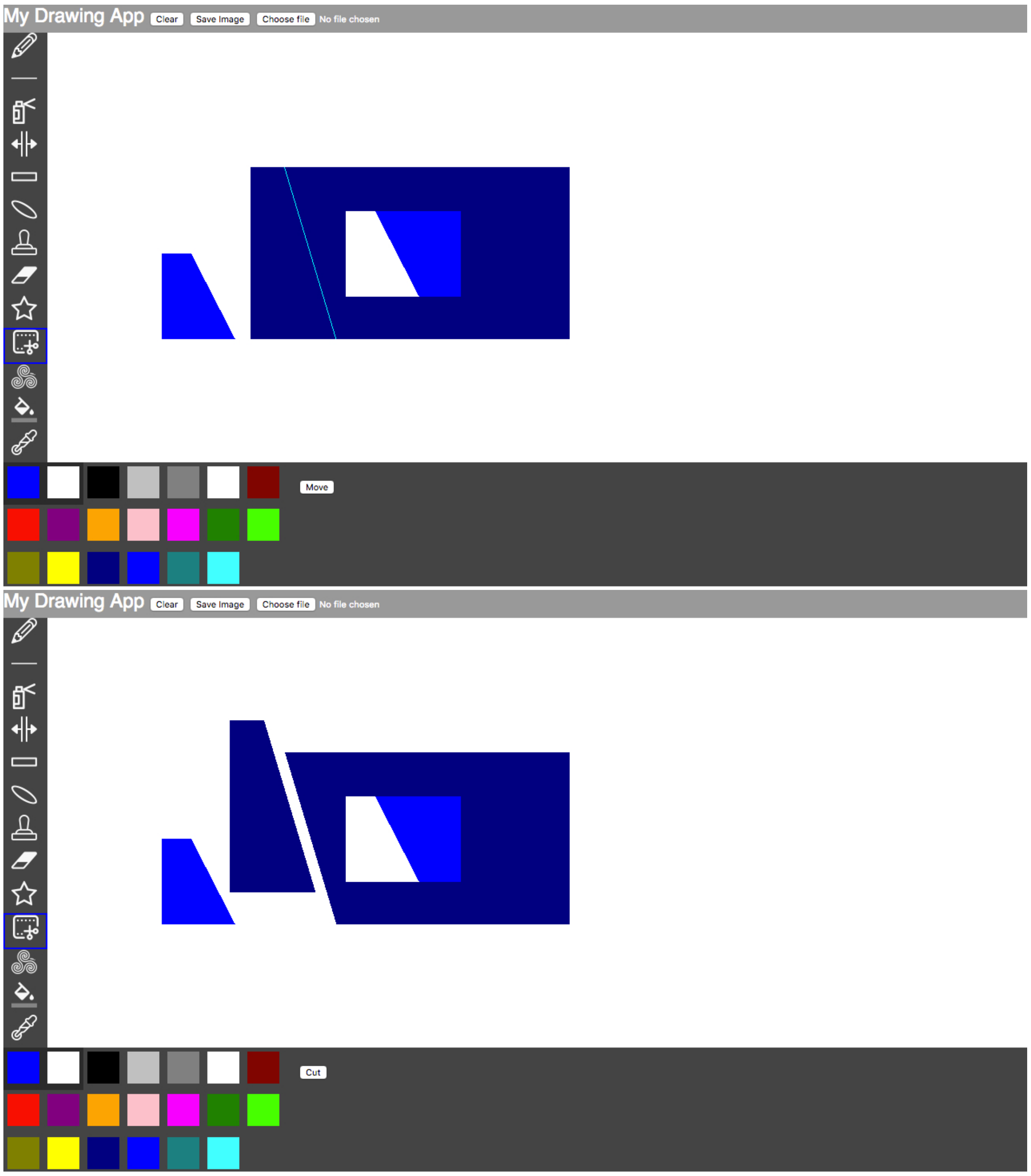
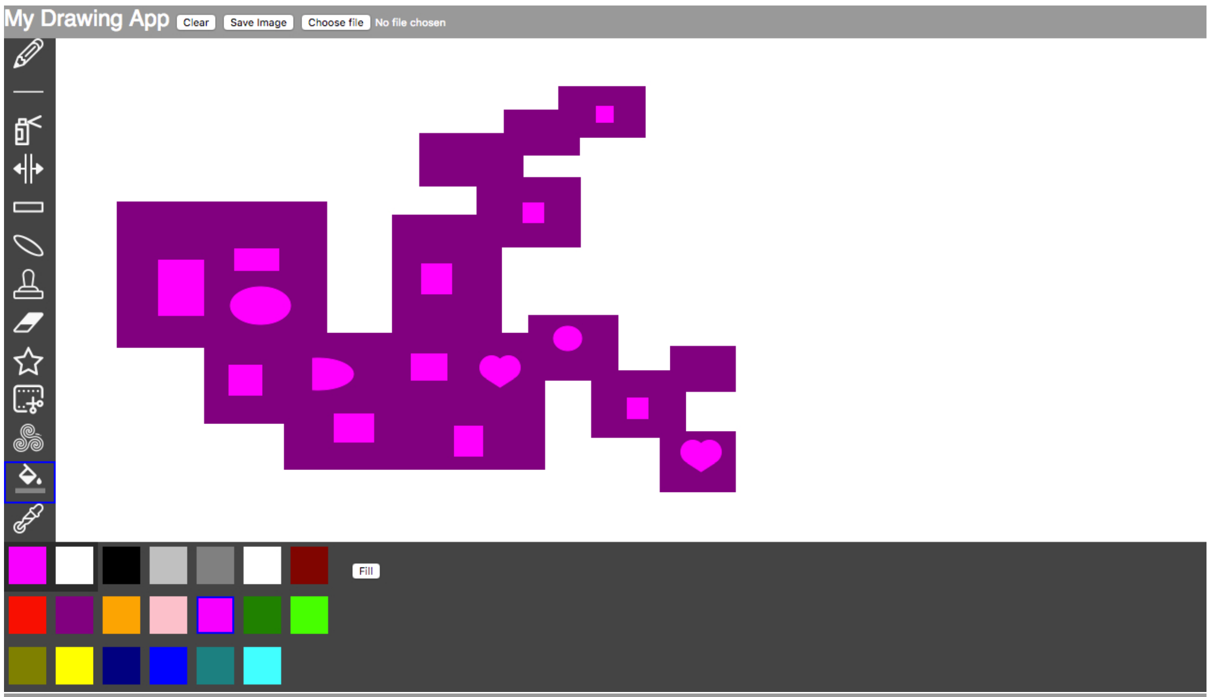
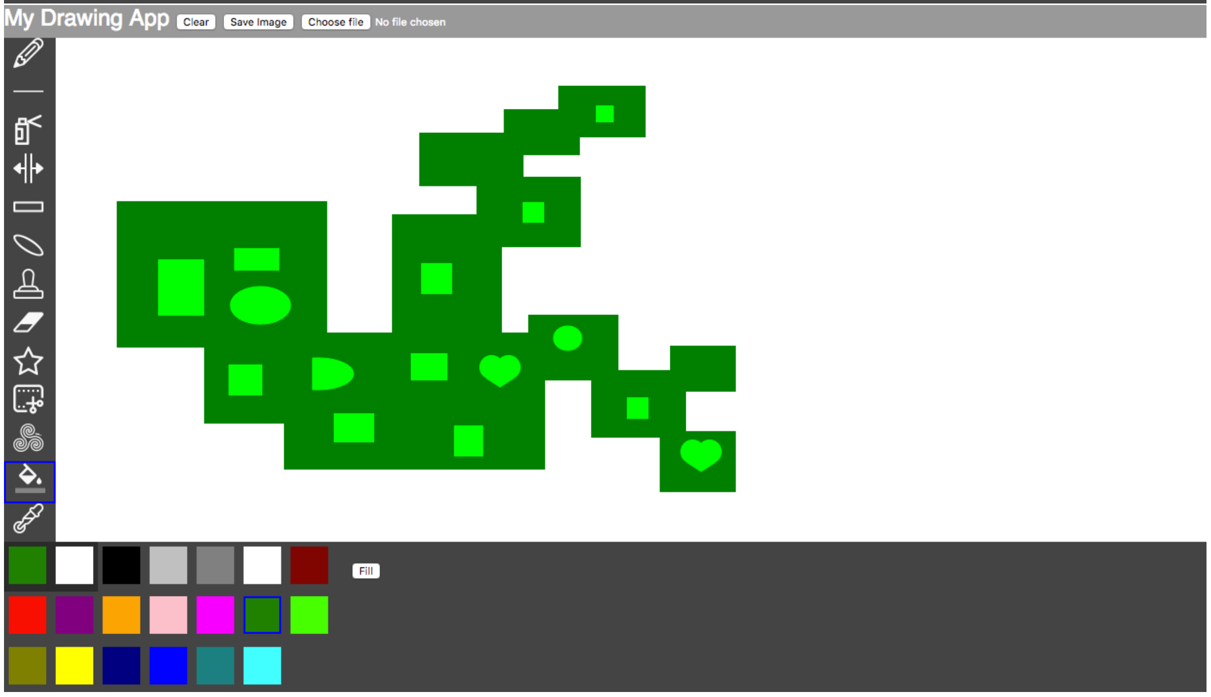


Figure 5 Scissors Tool


My Drawing App
Clear
Save Image
Choose file
No file chosen
Fill


My Drawing App
Clear
Save Image
Choose file
No file chosen
Fill

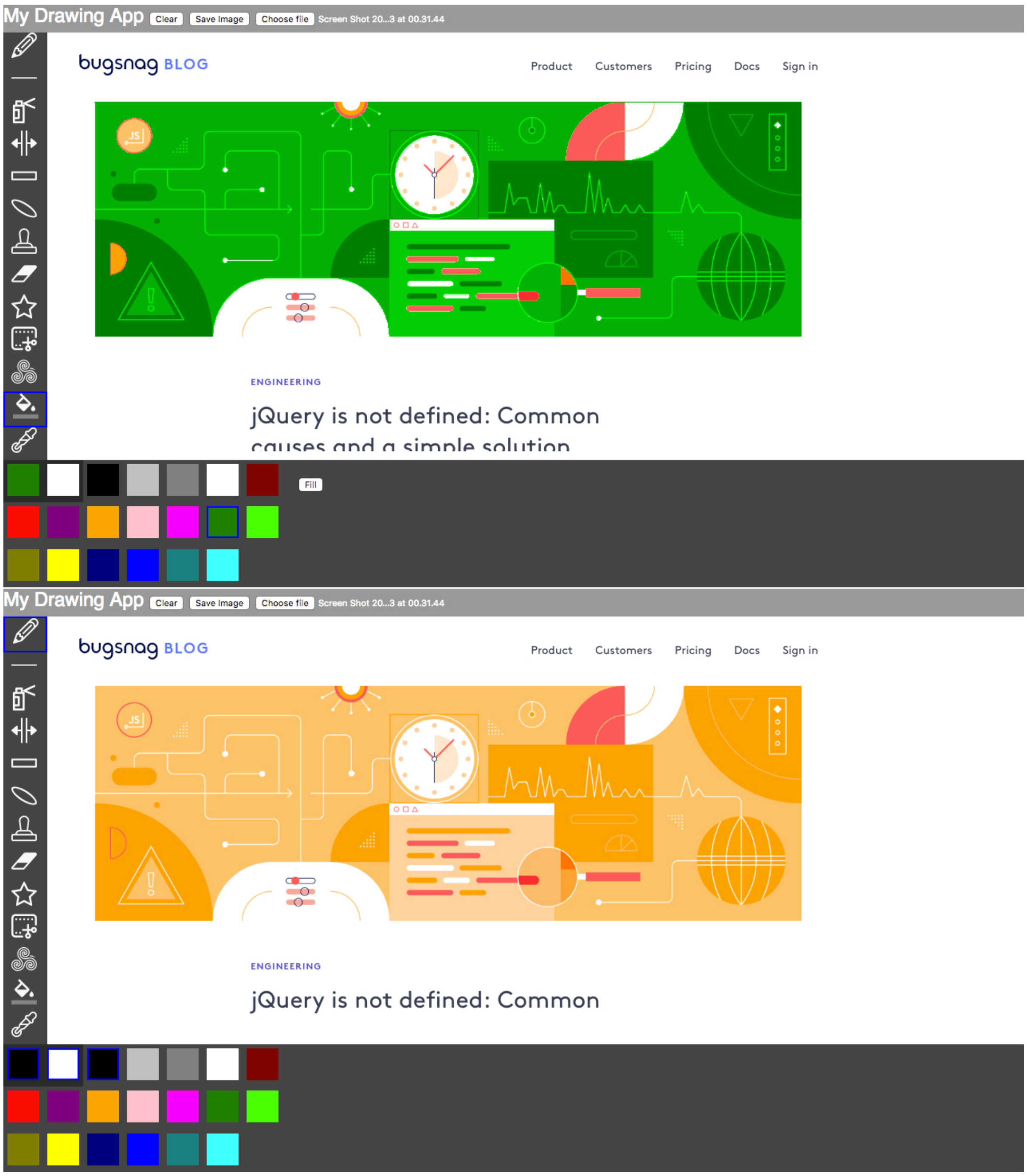


Figure 6. Flood Fill Tool – Colour Replacement

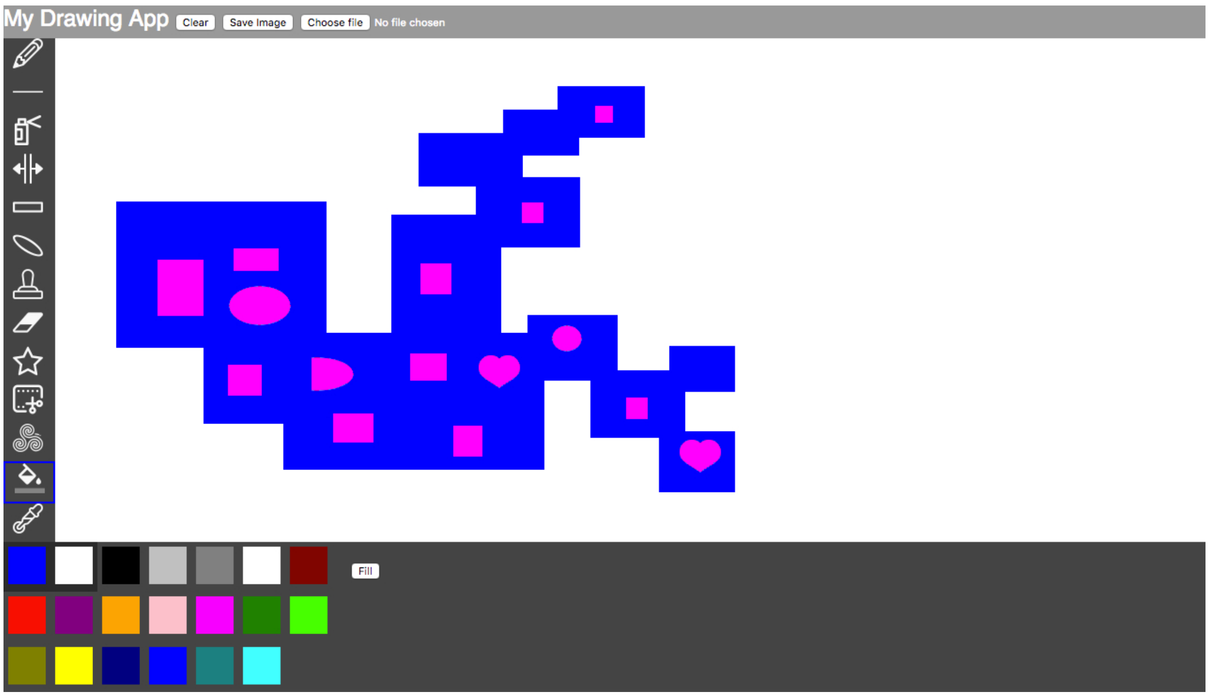


Figure 7. Flood Fill Tool - Fill

Methods included:

- whiteFix()
- getColour()
- compareColour()
- checkColour()
- selectEdge()
- selectColour()
- compareExactColour()
- genHSL()
- colourPHSL()
- rgbHSL()
- hslRGB()
- cutArray()

The Scissors and Flood Fill Tools (fig 6 and 7) have a similar structure. The Scissors Tool cuts a shape by splitting an array of pixels into pieces while the Flood Fill Tool replaces a selected colour. startMouseX and startMouseY are initialised off-canvas. Properties this.hRange, this.sRange, this.lRange and this.aRange hold the range of hues (+/- 10 degrees), saturation (100%), lightness (100%) and alpha(1) to be selected. Variable drawing is set to false and self is set to this (the ScissorsTool()/FloodFillTool() object) for a closure .

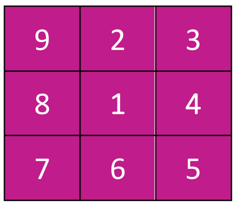
The Scissors Tool features options for cut (defaul) and move while the Flood Fill tool – for fill and replacement of a colour (default). Both tools select all pixels within an hsla range.
The draw() function of the Scissors Tool features a "three-layered" structure to provide support for mouse press, drag and release events while that of Flood Fill has a single layer to include events on mouse drag.

For the Scissors Tool, on pressing the mouse, if its position is off-canvas, startMouseX and startMouseY are set to the position of the mouse and drawing state is set to active. loadPixels() loads pixel data into the pixels[]. Within the “mouse drag layer”, a call to updatePixels() updates pixel data. This is where selection/cut/movement occurs. x1 and y1 are set to startMouseX and startMouseY , x2 and y2 are set to the position of the mouse while dragging and this.points is set to a new array – []. For the FloodFill Tool, on pressing the mouse, if its position is within the canvas, array this.points is set to empty and x and y are set to mouseX and mouseY.

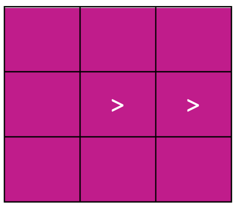
Both constructors set variable startColour, which stores the output of this.getColour(x,y), where x is x1 and y is x2. this.getColour(x,y) calculates the indices of red, green, blue and alpha values of the selected pixel within pixels[], gets the values and stores them as [r,g,b,a]. The array is then processed by this.whiteFix(colour), which converts white from [0,0,0,0] to [255,255,255,255] to allow differentiation between white and black based on hue rather than an alpha value. A call to the this.rgbHSL(rgba) method converts the array from RGBA to HSLA based on a conversion algorithm and returns the array [h, s, l, a].

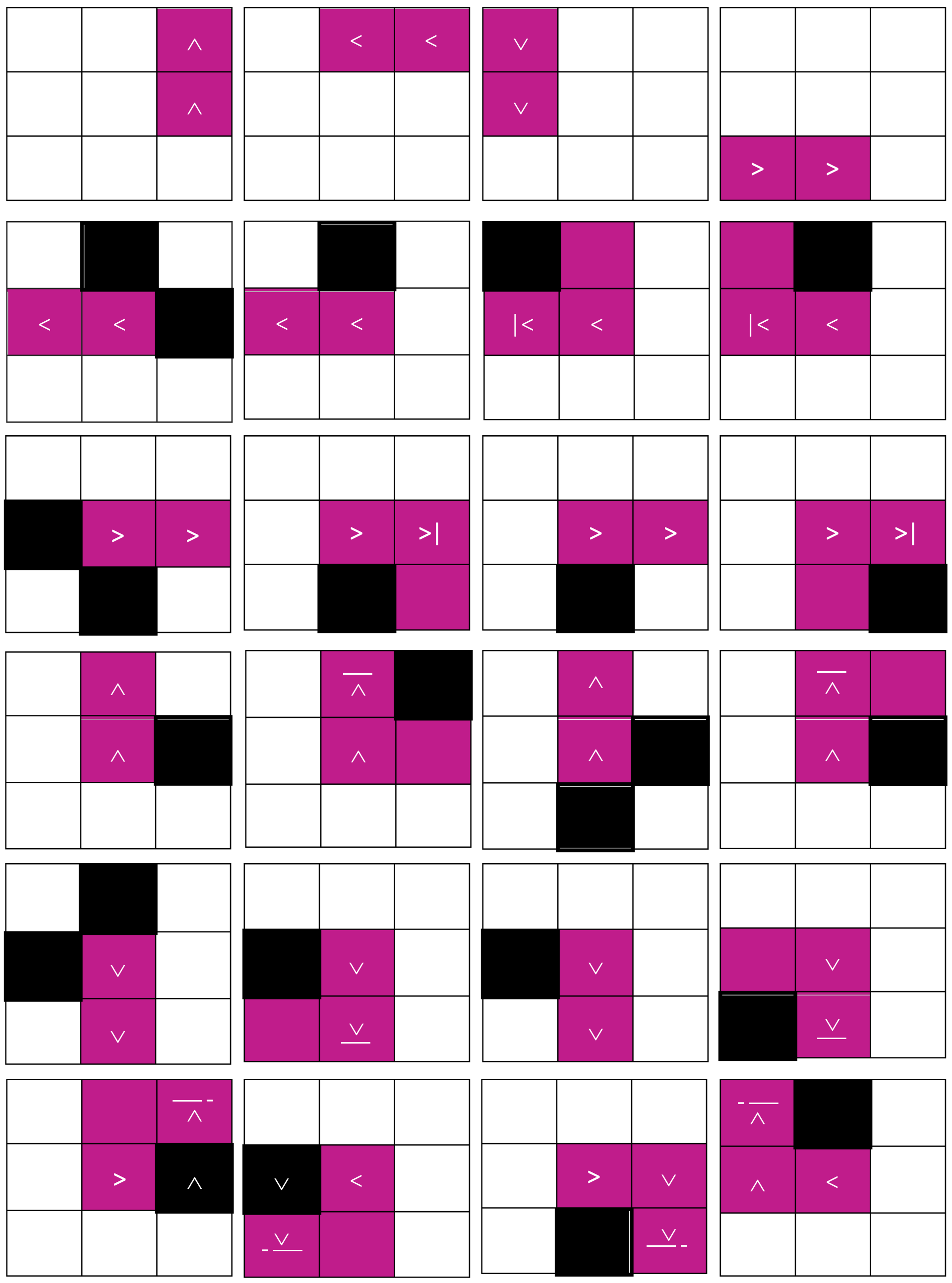
In draw(), variable colour is set to colourP.selectedColour, which refers to this.selectedColour of constructor ColourPalette() or the Background/Foreground Tool. If the colour is selected from the Foreground/Background tool, it is in the format [h,s,l,a]. If colour is selected from the palette, it is converted to an HSLA by this.colourPHSL(colour) and returned as [h,s,l,a]. Colour is selected by this.selectColour(array, startColour,colour,x,y,ind), where array is this.points, startColour is startColour, colour is colour, x is x1 and y is y1. No ind (index) is set. this.selectColour calls this.selectEdge(array,outline,startColour,colour,x,y,ind), which returns an outline of the shape. startX and newX are set to x, startY and newY – to y. If ind is not set, it is set to 0 and refers to the index of this.points, in which splicing of new points occurs. Variable start is set to 1 for indexing to reach an outline point. If the call to this.selectColour() originates from this.indexInner(), ind is set and points are added to a specific index. Since the outline is of an inner shape and point is at its top left corner, start is set to -1 and points are added from the first one.

Pixels along the edge are indexed in a while loop and their coordinates and colour are added to the array as [x,y,colour], where x is newX, y is newY and colour is the value of this.getColour(x,y). newX and newY variables increment/decrement to set the indexing direction based colour values of the selected and surrounding pixels.



1. newX, newY
2. newX, prevY
3. nextX, prevY
4. nextX, newY
5. nextX, nextY
6. newX, nextY
7. prevX, nextY
8. prevX, newY
9. prevX, prevY





🟪	compareColour() == true	>	continuous right
⬛	compareColour() == false	<	continuous left
⬜	N/A	^	continuous top
		v	continuous bottom

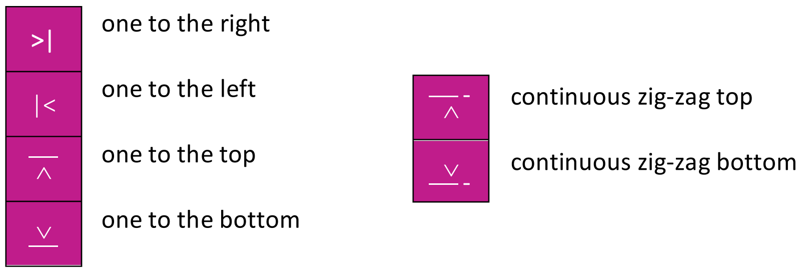


Fig 8. Indexing direction based on the surrounding pixels' colour

Addition of x, y and colour values occurs on change of y or colour (within a range).

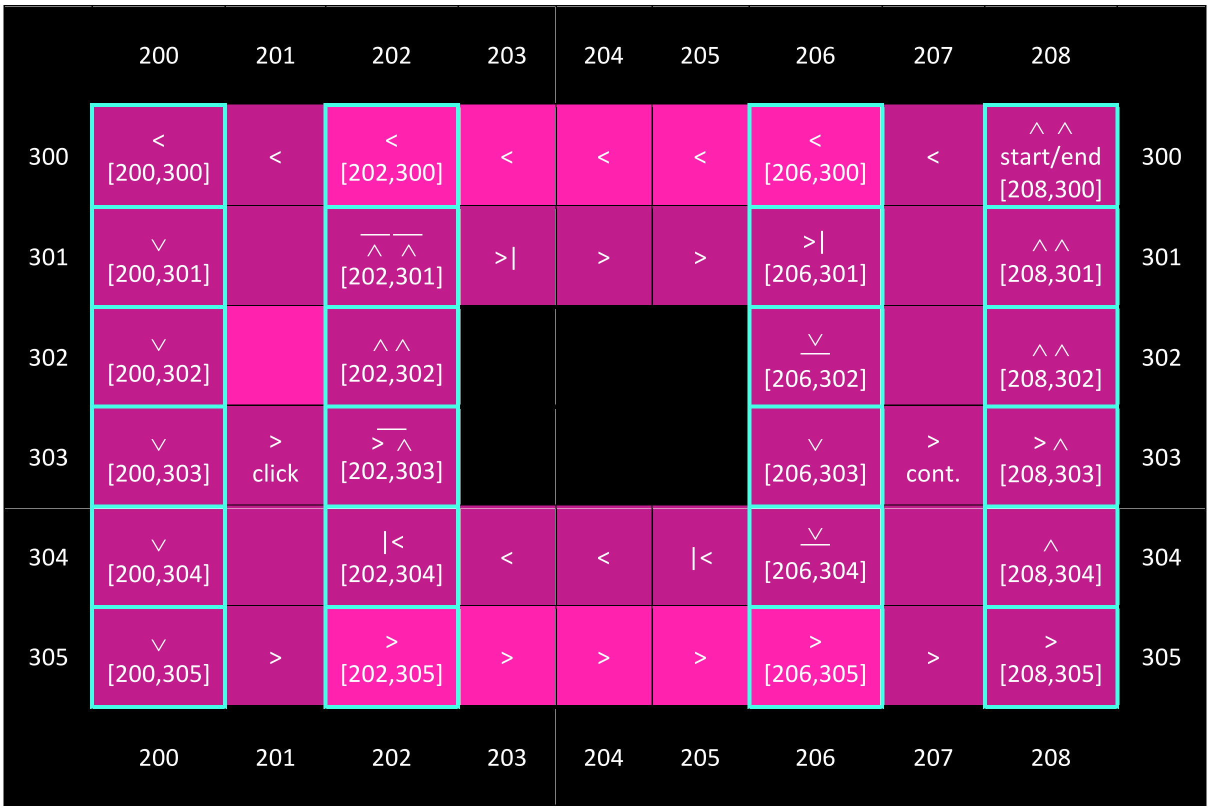


Figure 9 selectEdge() outline indexing example (1)

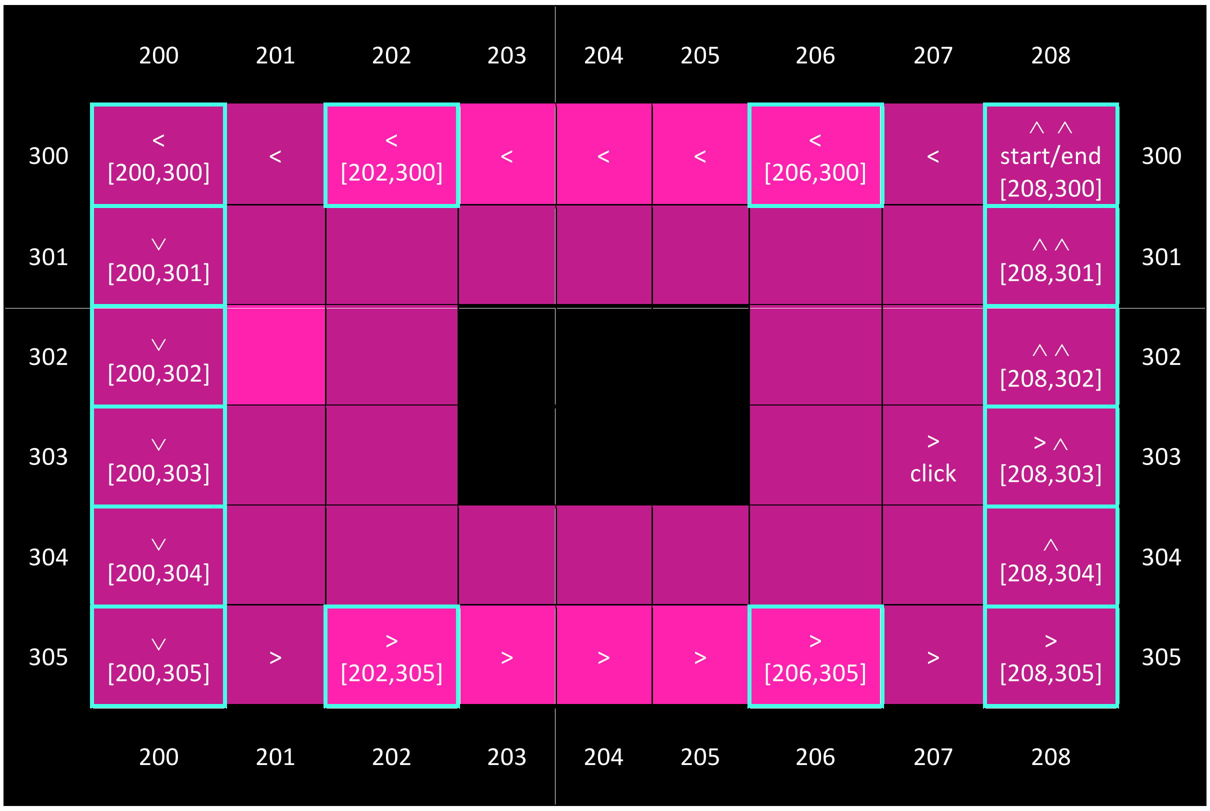


Figure 10 selectEdge() outline indexing example (2)

Variable edge is set to this.points. The outline is sorted by y and then by x by insertion sort. this.sortY(array,ind1,ind2), where array is edge, is an argument of this.sortX(array,ind1,ind2), where array is this.sortY(edge) (indices are not provided). this.sortY(edge) sets ind1 to 0 and ind2 to the length of the array – 1 and initialises currY to store y and its index iY. It traverses the array from ind1+1 to ind2 (incl.) within a for loop. If array[i-1][1] is not equal to currY, new values for currY and iY are set. If y of array[i-1] and array[i] are equal and x of array[i] is greater than the one of array[i-1], another for loop sets j to be equal to i and traverses the array backwards from i to iY+1. If the x value of array[j-1] is greater than the one of array[j], a call to swap(array,i1,i2), where i1 is j and i2 is j-1 is called to swap array[j-1] and array[j]. If array[j] is equal to array[j-1], a break exits the loop. this.sortX() then sets ind1 to 0 and ind2 to the length of the array – 1. currY and iY are initialised to store y and its index. A for loop traverses the array from ind1+1 to ind2. If y of array[i-1] and array[i] are not the same, new values for curry and iY are set – currY is set to array[i-1][1] and iY – to its index. If y at i-1 and i are equal and x of i-1 is less than x of i, another for loop sets j to i and traverses the array backwards from i to iY+1. If the x at j-1 is greater than x at j, this.swap() swaps array[j-1] for array[j] and array[j] for array[j-1]. The resulting arrays are shown below

Array 1 (if mouse pressed to the left of an inner shape):
```
[[200,300], [202,300], [206,300], [208,300],
  [200,301], [202,301], [206,301], [208,301],
  [200,302], [202,302], [206,302], [208,302],
  [200,303], [202,303], [206,303], [208,303],
  [200,304], [202,304], [206,304], [208,304],
  [200,305], [202,305], [206,305], [208,305]]
```

Array 2 (mouse pressed to the right of an inner shape):

```
[[200,300], [202,300], [206,300], [208,300],
 [200,301], [208,301],
 [200,302], [208,302],
 [200,303], [208,303],
 [200,304], [208,304],
 [200,305], [202,305], [206,305], [208,305]]
```

The array is passed as an argument to this.indexInner(array,startColour,colour) to index inner pixels. this.indexInner() detects colour changes and inner shapes. this.indexInner() traverses the this.points (array) in a for loop from i = 0 to the array length – 2. Points p1 and p2 are set to array[i] and array[i+1].

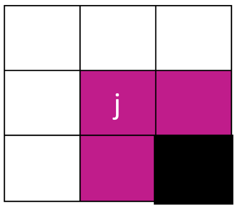


Figure 11. Shape check

If p1 is not an inner point as defined by the colour of the pixel to the right and no change of colour is detected, the current i is skipped. If it is, if there are more pixles than 1 between p2 and p1 and the their y coordinates are equal, a nested for loop indexes pixels from the pixel to the right of p1 to the pixel to the left of p2. An if statement checks whether p1 is the top left corner of an inner shape. If it is, initLen is set to store the length of the array and i1 is set to i+1 – the index at which outline points for the inner shape will be inserted. this.selectEdge(array, startColour,colour,x,y,ind), where x is x of a point between p1 and p2, y is y of p1 and ind is i1, indexes the outline of the inner shape and splices new points at i1 into the array (this.points). If the call to this.selectEdge() originates from this.indexInner(), at the end of indexing a for loop traverses the array searching for the point with rightmost x. The rightmost x is then assigned to maxX and its y to maxY. If the colour at maxX, maxY is within the specified range, indexing continues to the right of maxX, maxY by calling this.selectColour(array ,startColour,colour,x,y), where x is maxX and y is maxY. The new array length is assigned to newLen. The number of inserted points is newLen - initLen and is assigned to newPoints. i2 is set to the last new point + 1 (i1 + newPoints). endY is set to y of the first new point. A for loop traverses the array from i1 to i2 – 1 to find the maximum y of the inner shape by comparing each y to startY. k is set to i1 and the number of entries with a y of less than or equal to endY is counted in a while loop. this.sortX(array,ind1,ind2), where array is this.sortY(array,ind1,ind2), ind1 is i1 and ind2 is k (for both methods) returns a sorted array (this.points) following a partial insertion sort between i1 and k. A break exits the for loop and i increments by 1.

If p1 is not the top left corner of an inner shape, c1 and c2 are set to the colour of p1 and the colour of j between p1 and p2. The latter is the output of this.getColour(x,y), where x is j and y is y of p1. If c1 and c2 are not equal if compared with this.compareExactColour(c1,c2), x of j and y of p1[1] and its colour are spliced into the array at i+1 as [j,p1[1],c2].

The processed array is then returned by this.indexInner() to this.selectColour(), to be returned as an output.

The Scissors Tool and the Flood Fill Tool progress separately for more specific tasks.

Scissors Tool

The Scissors Tool proceeds with cutting the array into pieces based on a scissors line, stored in this.scissors. this.cutArray(array,scissors,x1,y1,x2,y2) receives the array as input, draws a scissors line, cuts the array and outputs a 2D array of pieces. this.cutArray() sets minY to the minimum of y1 and y2 as the top point of the scissors line. array1 and array2 are initialised to store the pieces. p1 and p2 are set within an if statement based on minY. p1 is the bottom of the scissors - if minY is equal to y1, p1 is set to [x2, y2], otherwise p1 is [x1,y1]. p2 is the top, i.e. if minY is y1, p2 is set to [x1,y1], otherwise p2 is [x2,y2]. The slope of the scissors line is calculated as (p2[0] - p1[0])/(p2[1] - p1[1]). sp2 and sp1 are set to the x and y of the lowest and the highest y. Each point between p1 and p2 is added to the scissors array as [x,y] in a for loop and a point with a cyan stroke is drawn between push() and pop().

array1 is set to a copy of the array. prevYi is set to 0. A for loop traverses scissors[] and a nested for loop traverses the array starting at i of 1. y of scissors[j] is matched with two sequential points array[i-1] with x less than or equal to that of scissors[j] and array[i] with x greater than or equal to that of scissors[j]. Two new points are spliced into the array as [scissors[j][0],scissors[j][1],array[i][2]], where scissors[j][0] and scissors[j][1] are the x and y of the point in scissors[] and array[i][2] is the colour value of the point at i. array[k] is added to array2 and removed from array1. array1 and array2 are then sorted by this.sortY() and this.sortX(), added to the 3D array pieces[] as [array1, array2] and returned by this.cutArray().

The "move" mode allows separation of the resulting pieces. Pressing the mouse within a piece calls this.move(pieces,scissors,colour,startColour,x1,y1,x2,y2). this.move() finds i, in which y1 is greater than y of the first point of a piece and lower than y of the last point and x1 is greater than x of the first point while lower than x of the last point, and assignes it to variable piece.

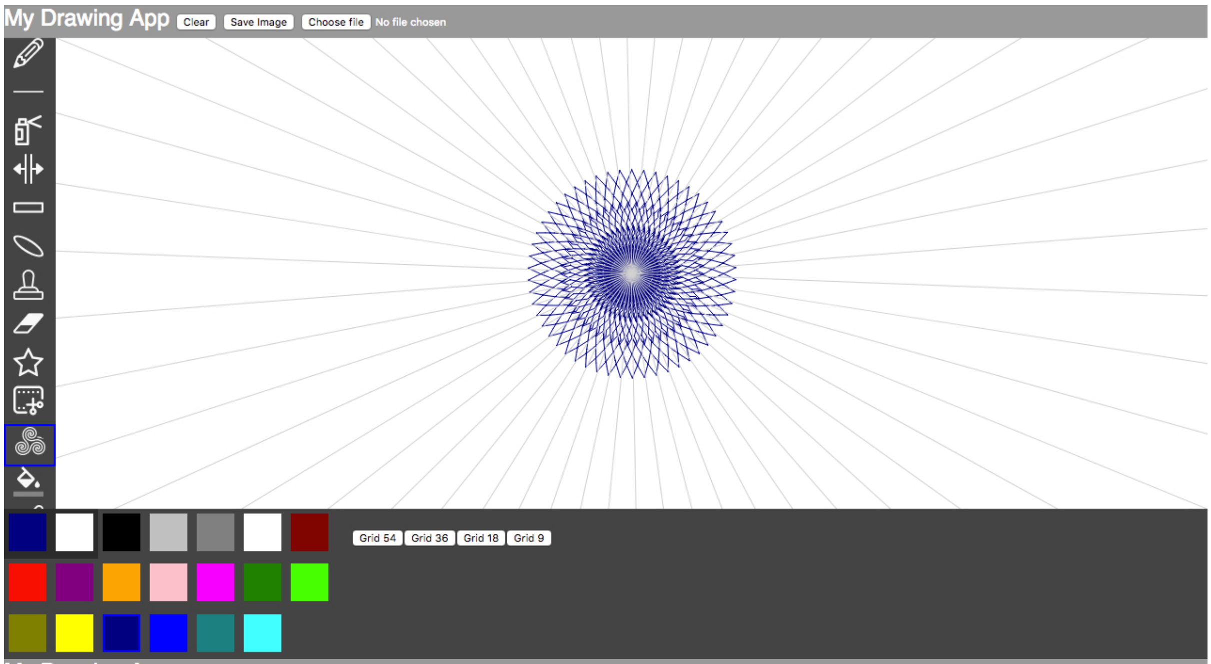
The initial colour of points along the scissors line is restored by drawing points at x of scissors[i][0] and y of scissors[i][1] with a stroke of scissors[i][2] within a for loop.

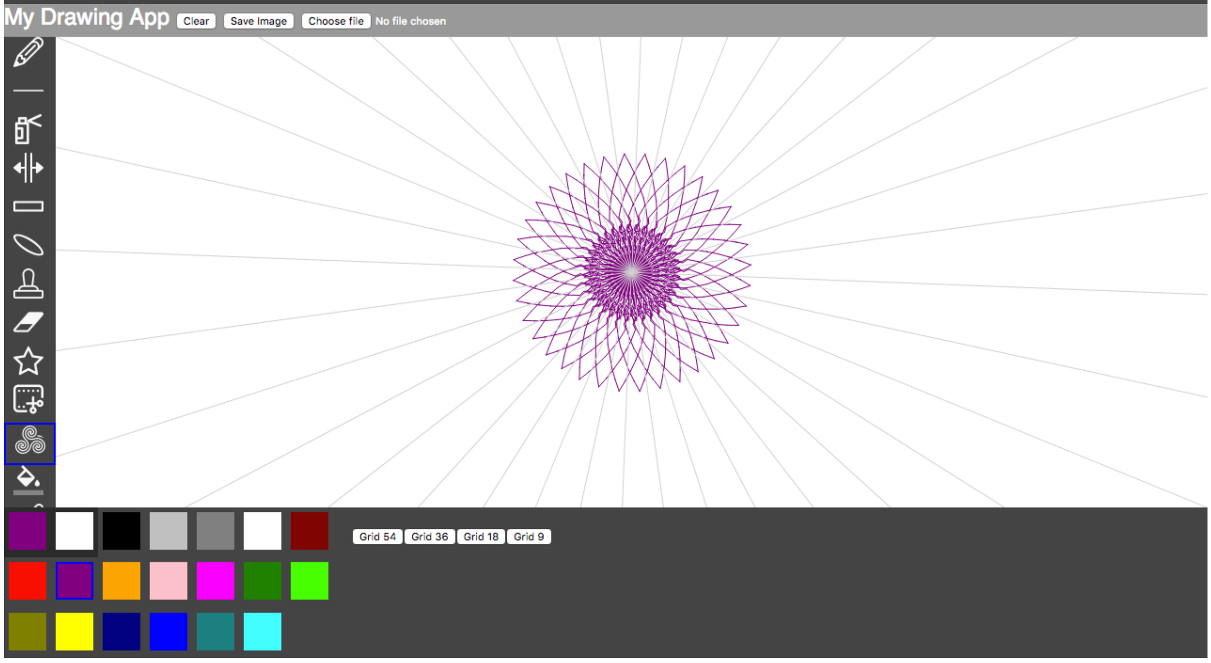
this.replaceColour(array,colour,startColour,x1,y1,x2,y2), where array is the selected piece and colour is startColour, draws the shape of the selected piece, which moves on mouse drag. this.replaceColour() traverses the array from i of 1 setting c1 and c2 to two sequential points array[i-1] and array[i]. The output of this.getColour(x,y), where x is x of the pixel to the right of c1 and y is y of c1, is assigned to nextXColour. The colour of c1 is assigned to c1Colour. this.calcColour(pointColour,colour,startColour), where pointColour is c1Colour and colour is startColour, calculates the colour of each point and assigns it to newColour. If nextXColour is equal to newColour and y of c1 is equal to that of c2 and if this.mode is set to "move", adjX and adjY are set to x2 - x1 and y2 - y1. If this.mode is set to "cut", adjX and adjY are set to 0. A line is drawn between c1 and the pixel to the left of c2 with a stroke of "white" to replace the selected piece. Another line is drawn between a point with x of c1[0]+AdjX and y of c1[1]+AdjY and a point with x of c2[0] – 1 + AdjX and y of c2[1]+AdjY, which are the new positions of c1 and c2 on mouse drag. If the colour of the pixel to the right of c1 is different from startColour, a point is drawn instead in "white" at c1 and another one at c1[0]+adjX, c1[1]+adjY in newColour. At the last i, points are drawn at c2 in "white" and c2[0] + adjX, c2[1] + adjY in newColour.

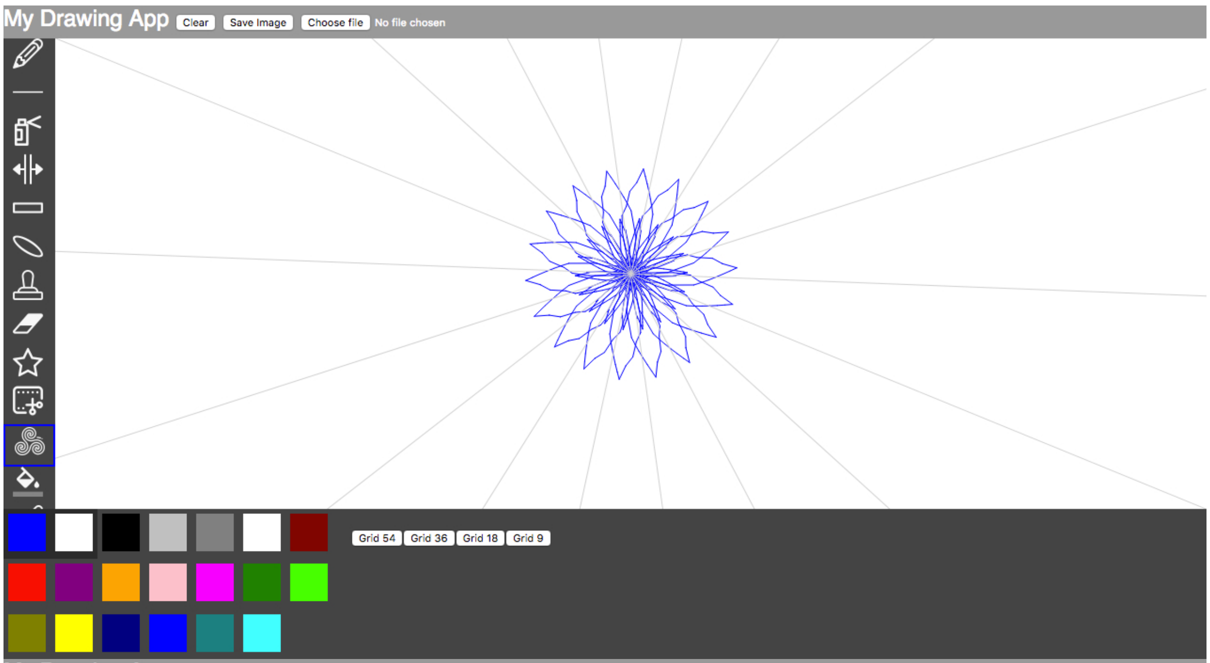
Flood Fill Tool

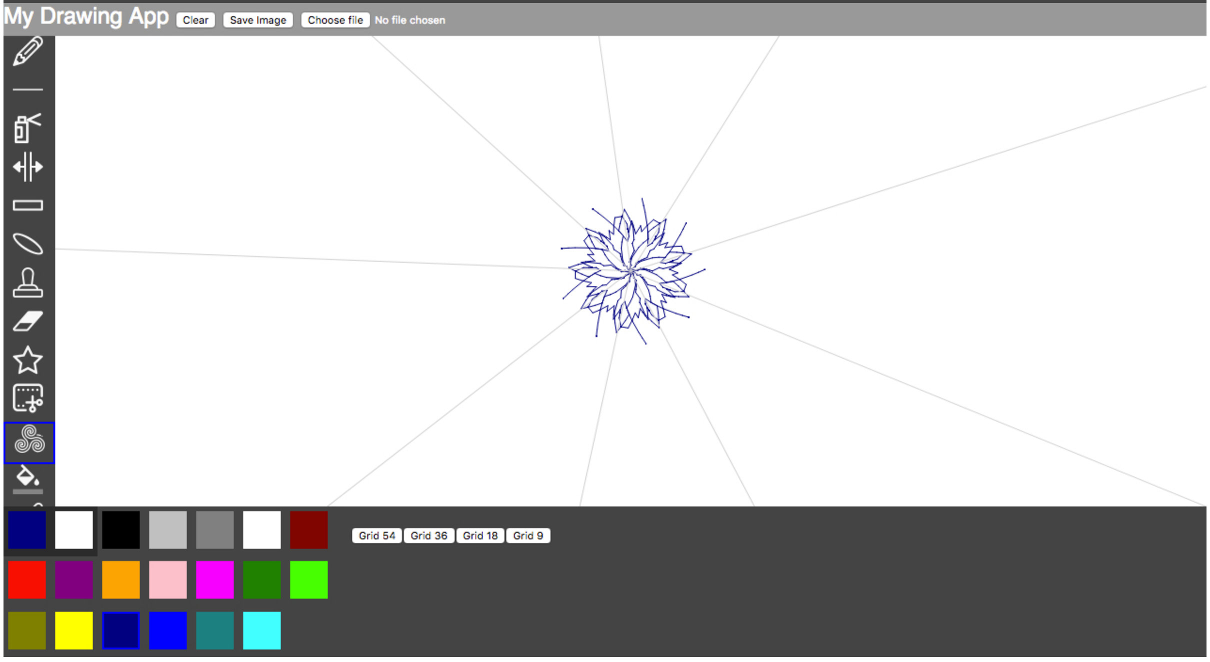
The Flood Fill Tool calls this.replaceColour(array,colour,startColour), where array is this.points, colour is the colour from the Colour Palette or the Foreground/Background Tool and startColour is the colour of the selected pixel on mouse press/drag. If the colour of the pixel to the right of c1 is the same as startColour, a line is drawn from c1 to c2[0]-1,c2[1]. If the colours are not the same, a point is drawn at c1 and at the last index of the array.

Spirograph Tool









The Spirograph Tool (fig 12) draws a spirograph on the canvas. this.grid is set to "on" to show the grid. this.symmetry is set to 54 grids by default. A mode selector property has options for 54, 36, 18 and 9 grids, included by this.populateOptions(), which inserts 4 html buttons with ids #grid54_Button, #grid36_Button, #grid18_Button and #grid9_Button with jQuery. A click event then sets this.symmetry to the selected grid. "this" is assigned to variable self for a closure. previousMouseX, previousMouseY and previousSymmetric are set to an off-canvas value of -1.

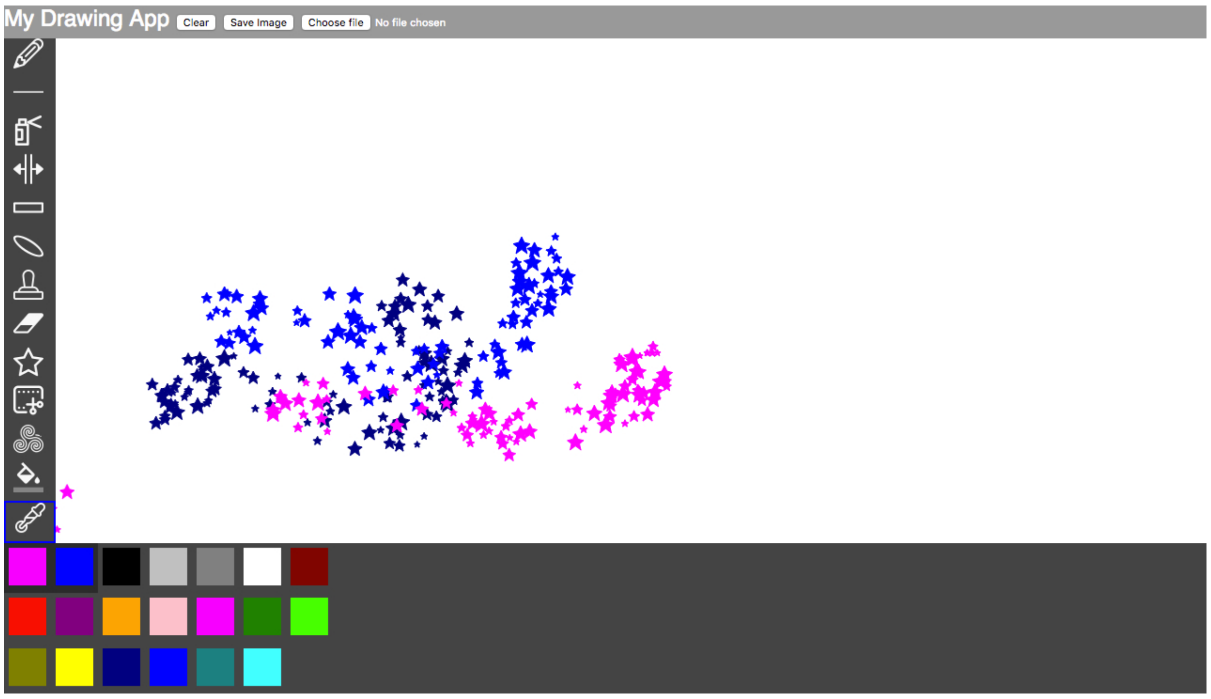
The draw() function has a "three-layered" structure for mouse press, drag and release events. On pressing the mouse, if the position of the mouse is off-canvas, previousMouseX and previousMouseY are set to the initial position of the mouse. this.calculateInitAngle(x,y), where x is previousMouseX and y is previousMouseY, is assigned to initAngle. this.calculateInitAngle(x,y) calculates the angle at which a point is drawn on the canvas relative to an x axis at height/2 and a y axis at width/2. Side a (a) is assigned to y-height/2, side b (b) – to x-width/2, the hypotenuse (r) – to sqrt(a**2 + b**2) and the angle at x and y of width/2, height/2 is calculated as the inverse cosine of b/r (angle), which is returned as an output. previousSymmetric is set to an empty array. Symmetric angles are calculated based on this.symmetry in a for loop. For each angle, symmetricAngle is set to initAngle + i*(TWO_PI/this.symmetry), i.e. a circle divided by the number of symmetric grids required. this.calculateSymmetric(x,y,angle), where x and y are the x coordinates of the initial point and angle is symmetricAngle returns the x and y of a symmetric point at its respective symmetricAngle. side a (a) is assigned to abs(y-height/2), side b (b) is assigned to abs(x-width/2) and the hypothenuse (r) is assigned to sqrt(a**2 * b**2). Then x and y are calculated as r*cos(angle) and r*sin(angle), which are assigned to x and y, returned as [x,y] and added to previousSymmetric.

Drawing occurs within the "mouse drag layer", a line is drawn from previousMouseX, previousMouseY to mouseX, mouseY and translated to the centre of the canvas width/2, height/2 within push() and pop(). This is followed by an updatePixels() statement to load new pixel data into pixels[]. previousMouseX and previousMouseY are set to the end point of the line. this.calculateInitAngle(x, y) with x and y of previousMouseX, previousMouseY is assigned to initAngle, symmetric is reset to a new array and the position of each drawn line is calculated in a for loop – symmetricAngle and pos are set for each symmetric point and pos is added to symmetric. A line is drawn from x and y of previousSymmetric[i][0] and previousSymmetric[i][1] to x and y of symmetric[i][0] and symmetric[i][1], which is translated to the center of the canvas within push() and pop() to cancel the translation and symmetric is then assigned to previousSymmetric.

If the mouse is not pressed, a third layer resets previousMouseX, previousMouseY and previousSymmetric back to an off canvas value.

loadPixels() ensures that symmetric lines will be removed from the canvas. Each line is drawn starting from top left corner of the canvas, rotated at an angle of TWO_PI/this.symmetry and translated to the centre of the canvas within push() and pop().

Background/Foreground Tool



The Background/Foreground Colour Tool (BFColourTool) (fig 13) stores background/foreground colours and provides a colour picker. this.selectedColour and this.foreground are set to “black” and this.background to "white". A mode property is set to either "foreground" and "background". Switching is by clicking on a foreground/background swatch. Two jQuery on() methods with click event handlers are added at the end of the constructor. Clicking on element with id #backgroundSwatch or #foregroundSwatch sets self.mode to “background” or “foreground”, resets the colourSwatches border back to 0 and adds a 2px solid blue border to the selected swatch. Another jQuery on() “click” method attached to ".colourPalette" with a selector ".colourSwatches" gets the background-color property of the selected swatch from the colour palette and sets self.foreground or self.background and the background-color of element with id #foregroundSwatch or #backgroundSwatch (based on self.mode) to the background-color of the colour palette swatch.

Progress Update

Week 1

01/04/2019	Ellipse Tool, Rectangle Tool
02/04/2019	Stamp Tool, Eraser Tool
03/04/2019	StarTrail Tool
04/04/2019 - 05/04/2019	Load Button
05/04/2019 - 07/04/2019	Flood Fill Tool

Week 2 - 3

08/04/2019 - 21/04/2019	Flood Fill Tool

Week 4

22/03/2019 - 25/04/2019	Spirograph Tool
26/03/2019 - 28/04/2019	Scissors Tool

Week 5 - 7

29/04/2019 - 10/05/2019	Scissors Tool

References

1. Niwa.nu. (2019). *Math behind colorspace conversions, RGB-HSL – Niwa*. [online] Available at: http://www.niwa.nu/2013/05/math-behind-colorspace-conversions-rgb-hsl/ [Accessed 10 May 2019].
2. Gist. (2019). *RGB, HSV, and HSL color conversion algorithms in JavaScript*. [online] Available at: https://gist.github.com/mjackson/5311256 [Accessed 10 May 2019].