Generative Design Colour Experiment Report

Hannah Devlin

Creative Computing (DL836)

Fourth Year

Introduction

My concept for the generative design colour experiment was to utilize P5 JS to explore and experiment with image pixel arrays by altering the size and colour of them. This involved taking an image as an input, taking the pixel array, altering the size of these pixels and outputting an image with different sized pixels.

Process

The development of the project included several processes of experimentation with both the code and images the code would use to output a result.

Early Stage

To begin the project, I used the generative-gestaltung website [1] to assist me. The original exercise depicted an image portrayed as a series of pixels; whenever the mouse was moved, the pixels grew which made the image less clear. This was due to the tile count on the x axis being reduced, therefore the pixels had more room and grew.

After gaining an understanding of how the original code worked, I commented each line with a description of what it did and broke it up into steps. I took a similar approach with my project by doing it step by step.

Step One

I began with setting up global variables, the canvas and the setup function. As seen below, I ensured to comment nearly every line so that I had a clear understanding of the code. This didn't output anything into the page, but when using the inspect tool the canvas outline can be seen.

```
// allows you to place a program, or a function, in a "strict" operating context.
// use strict also prevents certain actions from being taken and throws more exceptions.
'use strict';

// declare global variable
var img;
// declare colors array which will store all image pixels
var colorsArray = [];
var sortMode = null;

// preloads image using path in the brackets
function preload() {
  // img variable (declared above) loads image using given path
  img = loadImage('data/pic2.jpg');
}

// sets up canvas, its width and height
function setup() {
  createCanvas(1240, 1748);
  // no stroke around each pixel
  noStroke();
}
```

Step Two

This step was about incorporating a draw function where the pixels would be drawn as tiles across the x and y axis. The pixels are loaded into an array, loaded into variables (px and py) and converted to integers. The colours array uses the push function towards the end of step two which saves the drawing style settings currently being used. Like step one, nothing is displayed to the webpage but the canvas outline can be clearly seen using the inspect tool.

```
function draw() {
  // floor() calculates the closest int value that is less than or equal to the value of the parameter
  var tileCount = floor(width/10);
  // set the siz of the rectangle (tile size)
  var tileSize = width / tileCount;
  // loads pixels from image into colors array below
  img.loadPixels();
  colorsArray = [];

  // for loop printing out tiles into canvas accross x and y axis
  for (var gridY = 0; gridY < tileCount; gridY++) {
    for (var gridX = 0; gridX < tileCount; gridX++) {
      // pixels position of x axis
      // int() converts the parameter(s) to an int
      var px = int(gridX * tileSize);
      // pixels position on y axis
      var py = int(gridY * tileSize);
      var i = (py * img.width + px) * 4;
      var c = color(img.pixels[i], img.pixels[i + 1], img.pixels[i + 2], img.pixels[i + 3]);
      //  function that saves the current drawing style settings and transformations
      colorsArray.push(c);
    }
  }
}
```

Step Three

Unlike step one and two, this step produces a visible output into the canvas. The for-loop choses random values for the x and y positions as well as the pixel height and width using a variable that selects a random value between 1 – 100.

```
// generative design library = gd
gd.sortColors(colorsArray, sortMode);

var i = 0;
// for loop draws tiles accross x and y axis
for (var gridY = 0; gridY < tileCount; gridY++) {
  for (var gridX = 0; gridX < tileCount; gridX++) {
    // loop through colours in color array and fill the shape
    fill(colorsArray[i]);
    // c is equal to a random colour between 0 and 100
    var c = random(100);
    // blendmode is set to overlay - Uses a combination of Screen blend mode and Multiply blend mode
    blendMode(OVERLAY);
    // rectangle (tile) x position, y position, width, height
    // these will be random values as it uses the c variable from above
    rect(gridX * tileSize +c, gridY * tileSize+c, tileSize+c, tileSize+c);
    // counter
    i++;
  }
}
// stops it from looping
noLoop();
```

Step Four

The final step of code integration is enabling you to download the image as either a PNG or ASE file (Adobe Swatch File). This is done by pressing a particular key.

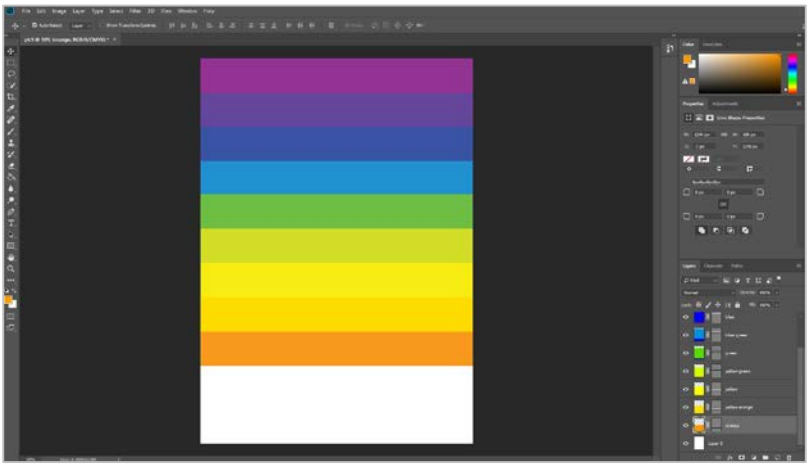
```
// function with actions and their corresponding keys
function keyReleased() {
  // if c is pressed, write a new file and
  // save image as ase (adobe swatch exchange file) and
  // uses timestamp from generative design library to name the image
  if (key == 'c' || key == 'C') writeFile([gd.ase.encode(colorsArray)], gd.timestamp(), 'ase');
  // if s is pressed, save canvas
  // uses timestamp from generative design library to name the image
  // save it as PNG
  if (key == 's' || key == 'S') saveCanvas(gd.timestamp(), 'png');
}

//function keyPressed() {
//    if (key == 'l'){
//        bm = "LIGHTEST";
//    }
//
//    if(key == 'm'){
//        bm = "MULTIPLY";
//    }
//}
```

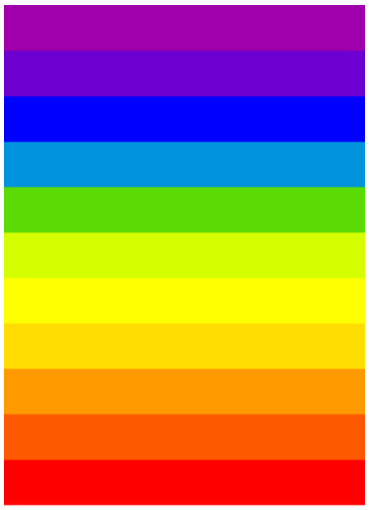
As seen above I also attempted to add another function called **keyPressed()** which would change the blend mode according to a particular key pressed. Unfortunately, I couldn't get this to work.

Middle Stage

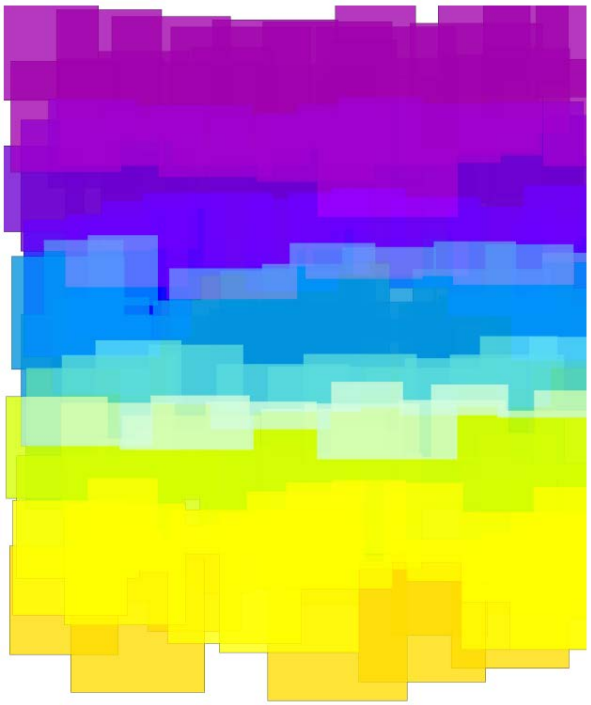
The next stage of the process involved deciding on an image.

Firstly, I attempted to create my own image using Adobe Photoshop with the aim being to output an image full of colour. I therefore created a row of colours in order from cold colours to warm colours.





After experimenting with the tile count, several blend modes and different alphas, I got the following output.



Alpha	Tile Count	Blend Mode
200	Width / 100 pixels	Lighter

I was not satisfied with this output as I did not feel that it demonstrated enough experiment with colour and shapes. This was mainly due to the rows of colour and that, in the original picture, there was only contrast in colour between each row and nowhere else.

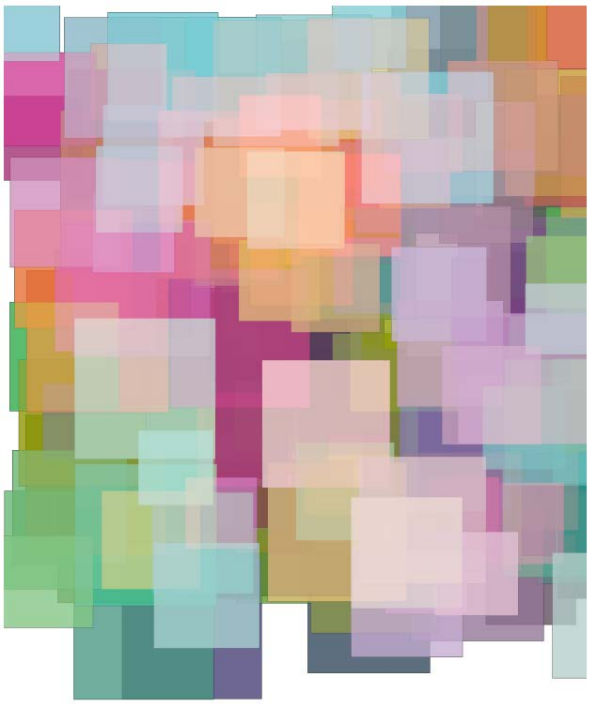
I moved onto researching for pieces of artwork in an attempt to generate an output with more contrast, interesting shapes and colours. I came across the following piece that was full of colour.

Final Stage

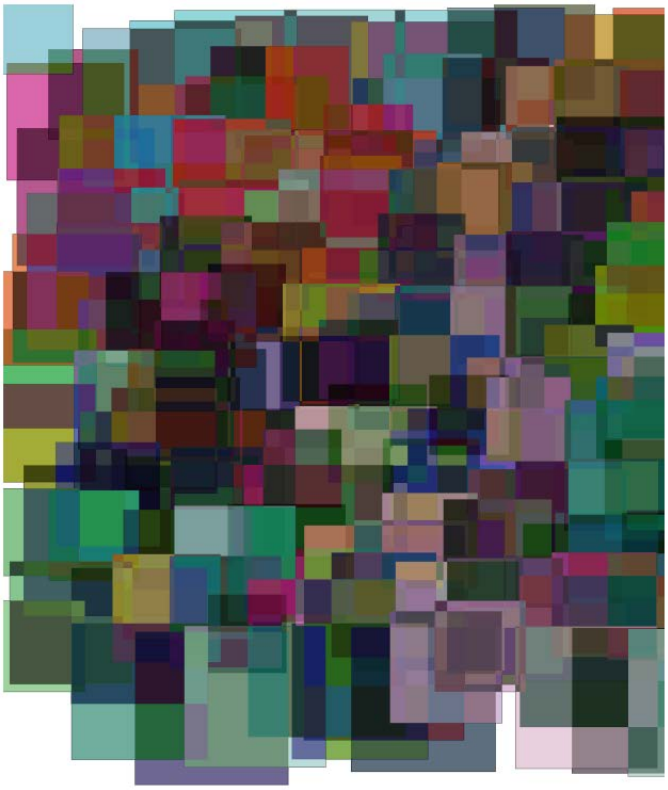
With the foundation of the code created and an image selected, I decided to progress onto incorporating some features to enhance my project. One of these features was **blendMode()** which I employed this into the second for loop inside the draw function. I researched this method using P5 JS references [3] which showed a wide range of options to pass as a parameter:

Multiply: multiplies the colours with the result always being darker.

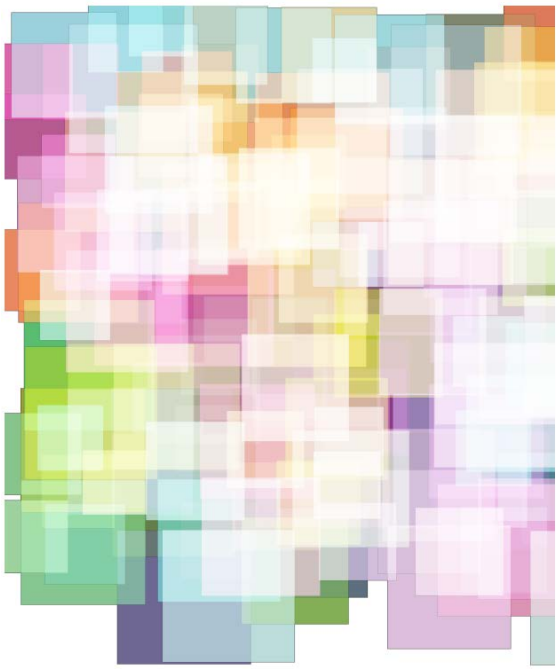
Lightest: whatever the lightest colours are prominent and will stand out more.



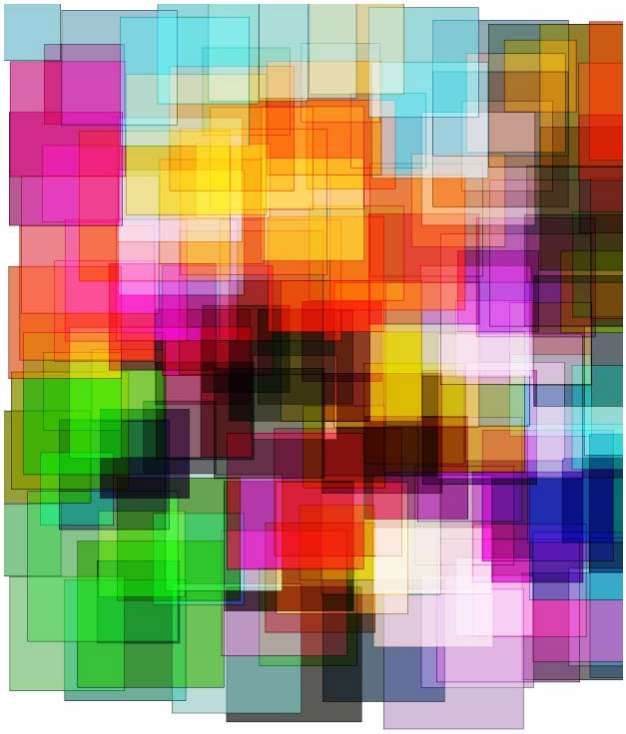
Difference: subtracts colours from the underlying image.



Screen: similar method to lighten blend mode but it is brighter and removes darker pixels.



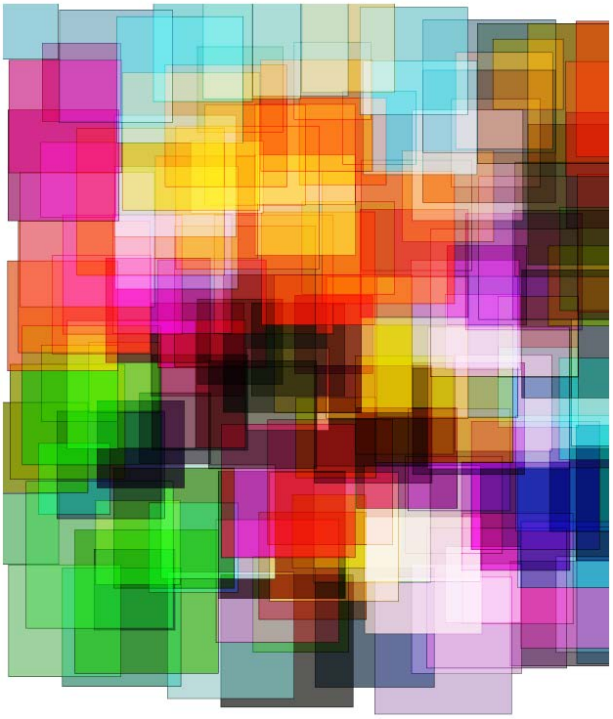
Overlay: Uses a combination of the Screen blend mode on the lighter pixels, and the Multiply blend mode on the darker pixels.



After experimenting with several different blend mode examples and researching how they worked, I concluded that the Overlay version was my preferred output.

Conclusion

Overall, I was very pleased with the outcome which resulted in an image with distorted pixels bigger than each and over lapping each other which creates a very interesting output full of colour an contrast.



References

[1] Generative Gestaltung

http://www.generative-gestaltung.de/2/

[2] Generative Gestaltung Exercise Five

http://www.generative-gestaltung.de/2/sketches/?01_P/P_1_2_2_01

[3] blendMode() P5 JS reference

https://p5js.org/reference/#/p5/blendMode

[4] Photoshop Blend Modes Explained

https://photoblogstop.com/photoshop/photoshop-blend-modes-explained