Section 1: Research and Inspiration

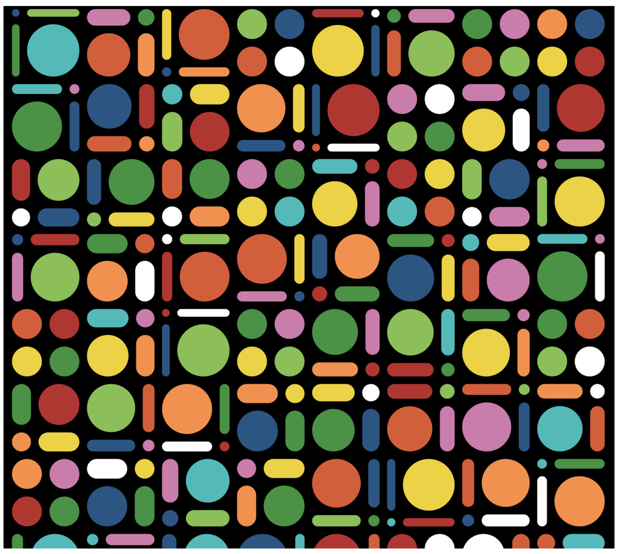
When imitating Anwar Jalal Shemza's The Apple Tree, we explored several design elements to shape the visuals and code structure:

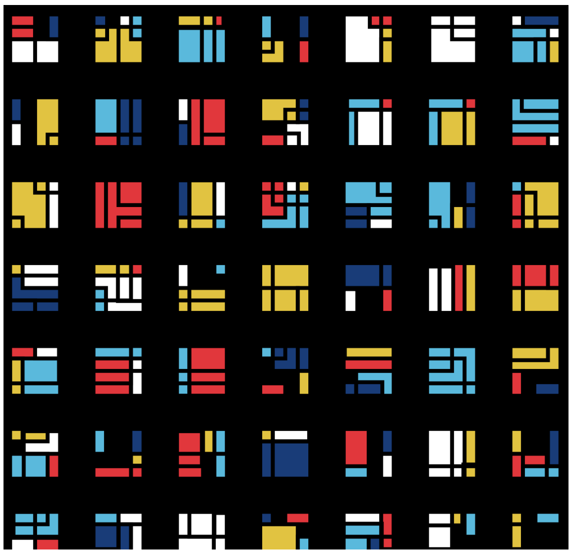
1. Geometric Abstract Art: The Apple Tree uses geometric shapes to convey complex ideas, emphasizing symmetry and layout. We replicate this by arranging simple shapes like squares, arcs, and rectangles in our code.

2. Okazz's Style: Inspired by Okazz's blend of geometric abstraction and soft curves, we incorporated:

•Static Growth: Using Perlin Noise, we generate natural curves, giving static images a dynamic flow.
•Layers and Space: Overlapping shapes with transparency adds depth, creating a complex, coordinated composition.
•Controlled Randomization: Random parameters balanced by symmetry ensure structured but varied visuals.
•Harmonious Color Gradients: Gradients enhance depth and cohesion, aligning with The Apple Tree's aesthetics.
•Orderly Randomness: Noise-controlled parameters produce balanced, natural structures, enriching the generative art.

The link of Okazz's works:





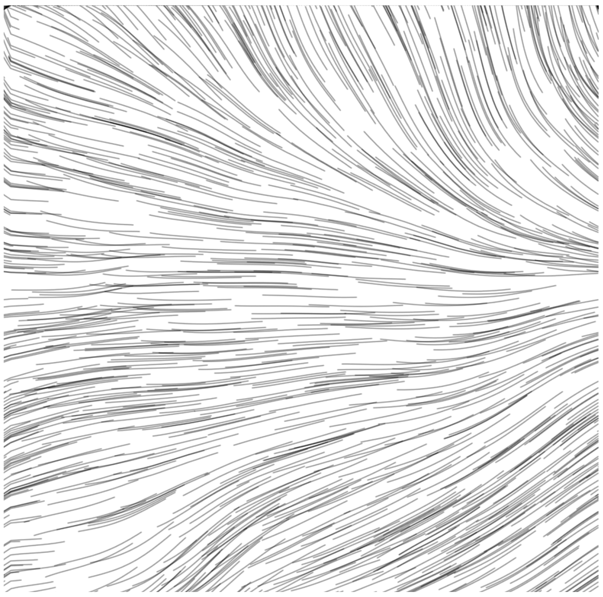
- **Perlin noise redux edit**

We're going to use Perlin noise to create a background intended to mimic the background texture of Anwar Jalal Shemza's Apple Tree to add depth and layering.

By generating a noise pattern with a certain regularity, it creates an effect similar to natural textures, making the background richer without being eye-catching.

- **The examples of Perlin noise redux edit**

The link of perlin noise redux edit 1 from Degox Art's work:
The link of perlin noise redux edit 2 from scott mayson's work:



Section 2: Technical planning

Four primary visual elements

Part 1 Background with a blue colour and thin yellow lines
Part 2 Scattered green blocks arranged horizontally
Part 3 The rectangular bases located at the bottom of the structure
Part 4 The main body structure of circles arranged in various patterns (horizontal, vertical, and side extensions)

Classes and Functions for Each Part

Part 1: Background Texture
We will use the drawBranch() function to create a textured background with randomly generated lines, maintaining the original colour tone (background('#1E3A5F')) in setup().

Part 2: Green Blocks
The drawGreenSquares() function draws green squares at the bottom of the canvas, with squareColor and strokeColor set globally for consistency.

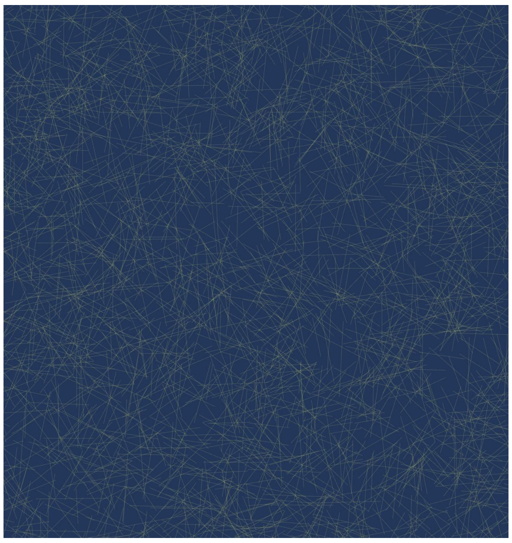
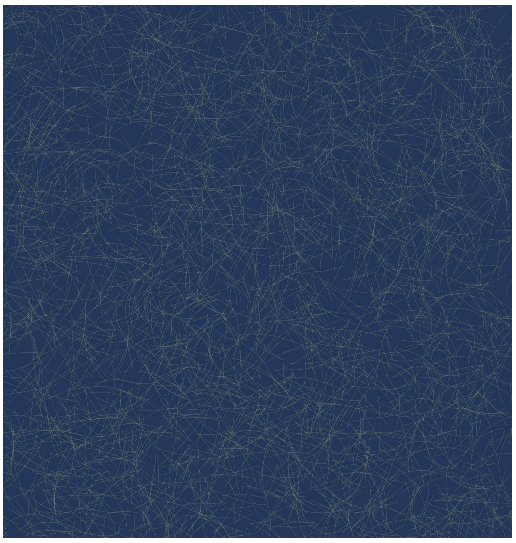
Part 3: Rectangular Base Units
The RectangleUnit class generates small rectangles, each with an optional semicircle at the bottom and random colours from shades of yellow, green, and red for visual detail.

Part 4: Main Body
The BicolorCircle class forms the main structure, with drawConnectedCircles() creating a balanced arrangement of circles. Five horizontal circles form the base, with vertical and side extensions.

Screen Adaptation
The windowResized() function adjusts the canvas to fit any screen size, recalculating layout in setup() for proportional scaling based on windowWidth and windowHeight.

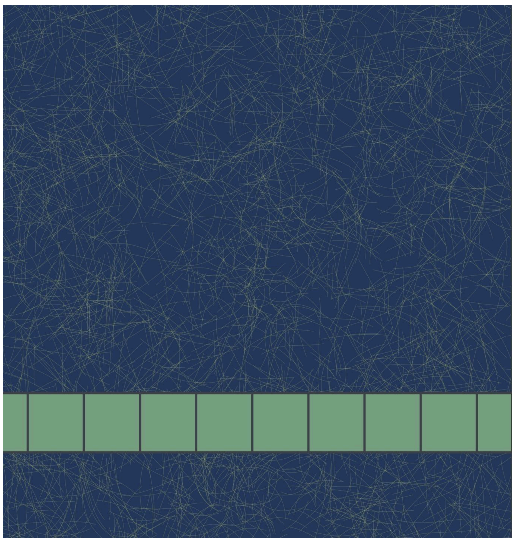
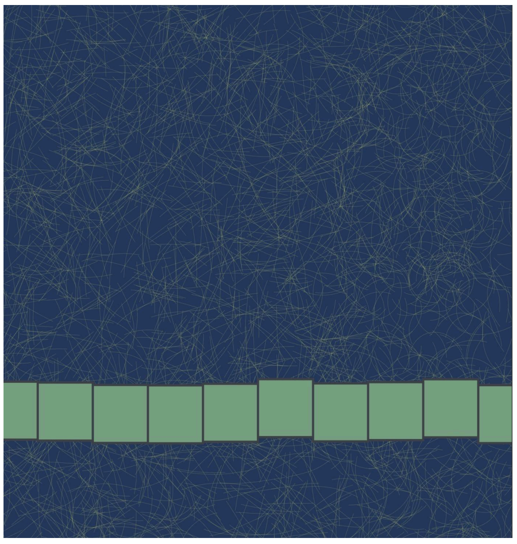
Section 3: Implementation

Iteration 1 - Background Texture

Initially, our background was created with straight lines. After iteration, we introduced Perlin noise to the drawBranch() function, allowing each line segment to vary slightly in direction. This created a more organic and textured effect, enhancing the background's complexity and adding depth to the composition.

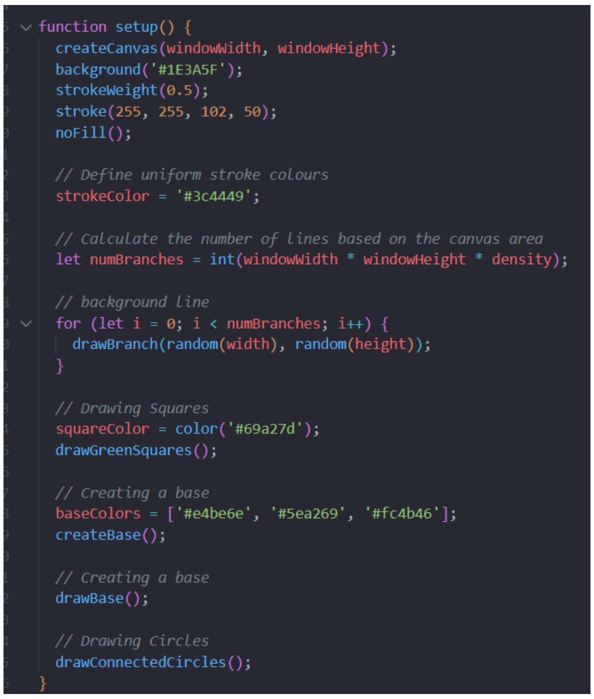
Iteration 2 - Green Squares
Green squares were initially drawn without any vertical offset, creating a rigid row at the bottom. After iteration, we introduced random yOffset values for each square within drawGreenSquares(), making the line of squares less static. This introduced a sense of randomness for each square, giving the artwork more visual flow and harmony with the textured background.

Section 4: Technical overview

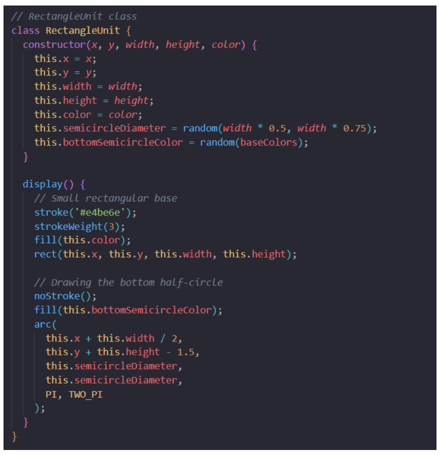
Setup Function

1. Set up the canvas and background, define global variables like strokeColor and squareColor, and call various helper functions to generate the artwork.
2. Background Lines: Random lines are drawn using drawBranch() to create a dense, organic background pattern.
3. Squares and Base: Green squares and a rectangular base with semicircles are drawn to add structural elements to the piece.

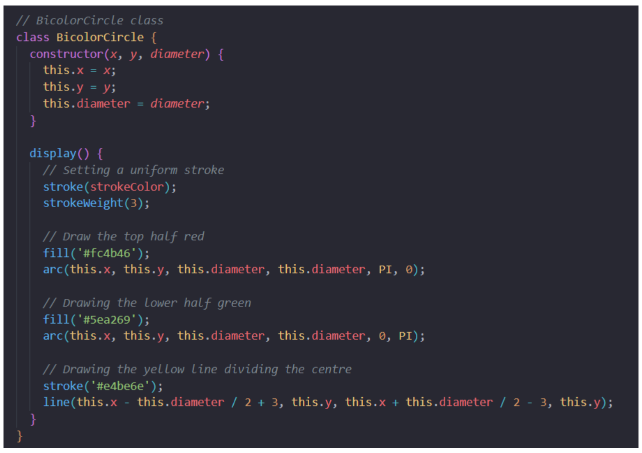


```
function setup() {
  createCanvas(windowWidth, windowHeight);
  background('#1E3A5F');
  strokeWeight(0.5);
  stroke(255, 255, 102, 50);
  noFill();

  // Define uniform stroke colours
  strokeColor = '#3c4449';

  // Calculate the number of lines based on the canvas area
  let numBranches = int(windowWidth * windowHeight * density);

  // background line
  for (let i = 0; i < numBranches; i++) {
    drawBranch(random(width), random(height));
  }

  // Drawing Squares
  squareColor = color('#69a27d');
  drawGreenSquares();

  // Creating a base
  baseColors = ['#e4be6e', '#5ea269', '#fc4b46'];
  createBase();

  // Creating a base
  drawBase();

  // Drawing Circles
  drawConnectedCircles();
}
```

RectangleUnit Class
Manages the rectangular base units that support the circles. Each rectangle has a randomly chosen color from baseColors and a semicircle at the bottom.
display Method: Draw the rectangle and a semicircle at its bottom edge.
BicolorCircle Class
The BicolorCircle class is used to create circles with two colors divided across the top and bottom halves, with a yellow line as a divider. The class constructor accepts x, y, and diameter to control the circle's position and size.
display Method: Draw the split circle using arc() and a line() for the dividing line.

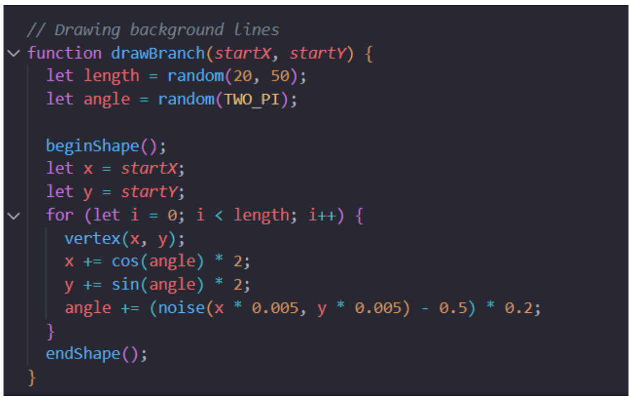


```javascript
// RectangleUnit class
class RectangleUnit {
  constructor(x, y, width, height, color) {
    this.x = x;
    this.y = y;
    this.width = width;
    this.height = height;
    this.color = color;
    this.semicircleDiameter = random(width * 0.5, width * 0.75);
    this.bottomSemicircleColor = random(baseColors);
  }

  display() {
    // Small rectangular base
    stroke('#e4be6e');
    strokeWeight(3);
    fill(this.color);
    rect(this.x, this.y, this.width, this.height);

    // Drawing the bottom half-circle
    noStroke();
    fill(this.bottomSemicircleColor);
    arc(
      this.x + this.width / 2,
      this.y + this.height - 1.5,
      this.semicircleDiameter,
      this.semicircleDiameter,
      PI, TWO_PI
    );
  }
}
```



```javascript
// BicolorCircle class
class BicolorCircle {
  constructor(x, y, diameter) {
    this.x = x;
    this.y = y;
    this.diameter = diameter;
  }

  display() {
    // Setting a uniform stroke
    stroke(strokeColor);
    strokeWeight(3);

    // Draw the top half red
    fill('#fc4b46');
    arc(this.x, this.y, this.diameter, this.diameter, PI, 0);

    // Drawing the lower half green
    fill('#5ea269');
    arc(this.x, this.y, this.diameter, this.diameter, 0, PI);

    // Drawing the yellow line dividing the centre
    stroke('#e4be6e');
    line(this.x - this.diameter / 2 + 3, this.y, this.x + this.diameter / 2 - 3, this.y);
  }
}
```

Generating background line
Draws a row of evenly spaced green squares at a fixed position with a small random vertical offset to add visual interest.
1. beginShape() and endShape(): These functions define the start and end of a custom shape, where each point is connected to form a line.
2. angle += (noise(x * 0.005, y * 0.005) - 0.5) * 0.2: Noise is used here to slightly alter the angle.



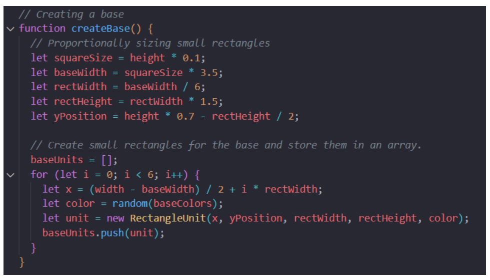
```javascript
// Drawing background lines
function drawBranch(startX, startY) {
  let length = random(20, 50);
  let angle = random(TWO_PI);

  beginShape();
  let x = startX;
  let y = startY;
  for (let i = 0; i < length; i++) {
    vertex(x, y);
    x += cos(angle) * 2;
    y += sin(angle) * 2;
    angle += (noise(x * 0.005, y * 0.005) - 0.5) * 0.2;
  }
  endShape();
}
```

Generating green squares
Draw a row of evenly spaced green squares at a fixed position with a small random vertical offset to add visual interest.

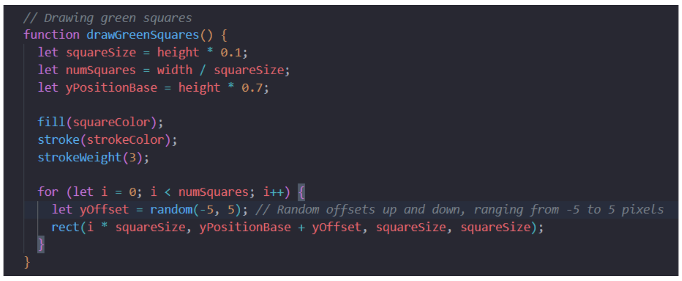
1. These variables define the size of each square, the number of squares that fit across the screen, and the baseline height for the row.
2. yOffset = random(-5, 5): Adds a slight vertical random offset for each square to create a non-uniform, lively arrangement.
3. rect(i * squareSize, yPositionBase + yOffset, squareSize, squareSize): Draws each square in the row, with the horizontal position determined by i * squareSize.

Create base

createBase() initializes six rectangular base units and stores them in the baseUnits array.
1. These variables define the size and position of each rectangle in the base, proportionally calculated based on screen height.
2. baseUnits = []: Initializes an array to store each rectangle's properties.
3. for (let i = 0; i < 6; i++): The loop generates six rectangles, each with calculated position and color.
4. x = (width - baseWidth) / 2 + i * rectWidth: Centers the rectangles horizontally and positions them side by side.
5. color = random(baseColors): Randomly selects a color for each rectangle from a predefined array baseColors.
6. RectangleUnit: Represents each rectangle as an object with specific attributes (x, y, width, height, and color), which is added to the baseUnits array for further processing or rendering.



```
// Creating a base
function createBase() {
  // Proportionally sizing small rectangles
  let squareSize = height * 0.1;
  let baseWidth = squareSize * 3.5;
  let rectWidth = baseWidth / 6;
  let rectHeight = rectWidth * 1.5;
  let yPosition = height * 0.7 - rectHeight / 2;

  // Create small rectangles for the base and store them in an array.
  baseUnits = [];
  for (let i = 0; i < 6; i++) {
    let x = (width - baseWidth) / 2 + i * rectWidth;
    let color = random(baseColors);
    let unit = new RectangleUnit(x, yPosition, rectWidth, rectHeight, color);
    baseUnits.push(unit);
  }
}
```

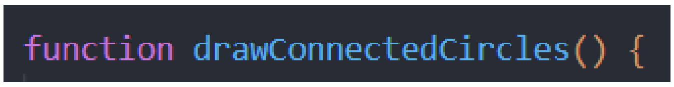


```
// Drawing green squares
function drawGreenSquares() {
  let squareSize = height * 0.1;
  let numSquares = width / squareSize;
  let yPositionBase = height * 0.7;

  fill(squareColor);
  stroke(strokeColor);
  strokeWeight(3);

  for (let i = 0; i < numSquares; i++) {
    let yOffset = random(-5, 5); // Random offsets up and down, ranging from -5 to 5 pixels
    rect(i * squareSize, yPositionBase + yOffset, squareSize, squareSize);
  }
}
```

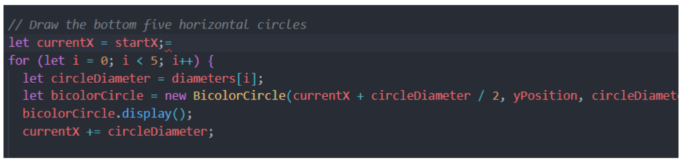
Generating Bicolor Circles

The primary function for placing and organizing circles. It starts by generating five horizontal circles, then add a series of six vertical circles in the middle.
1. A "for loop" iterates numCircles times to draw each circle and connect it to the previous one.
2. Calculate the position for every circle:
   1) "angle = random(TWO_PI)" generates a random angle in a specific range, which determines the direction of each new circle relative to the previous one.
   2) "distance = random(50, 100)" generates a random distance between 50 and 100 for each new circle.
   3) "x" and "y" calculate the new position of the circle based on the angle and distance:

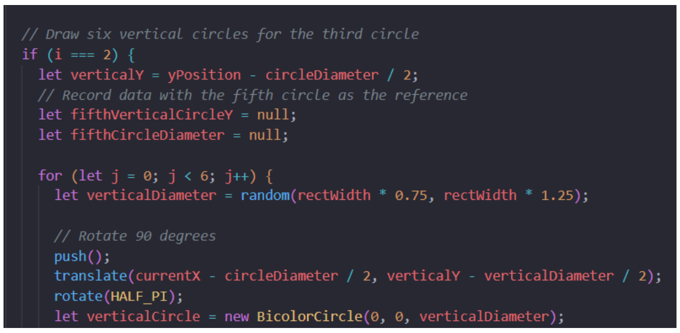
Additional circles are drawn on the sides of the vertical sequence, and a final trio of horizontal circles is placed at the top.



```
function drawConnectedCircles() {
```



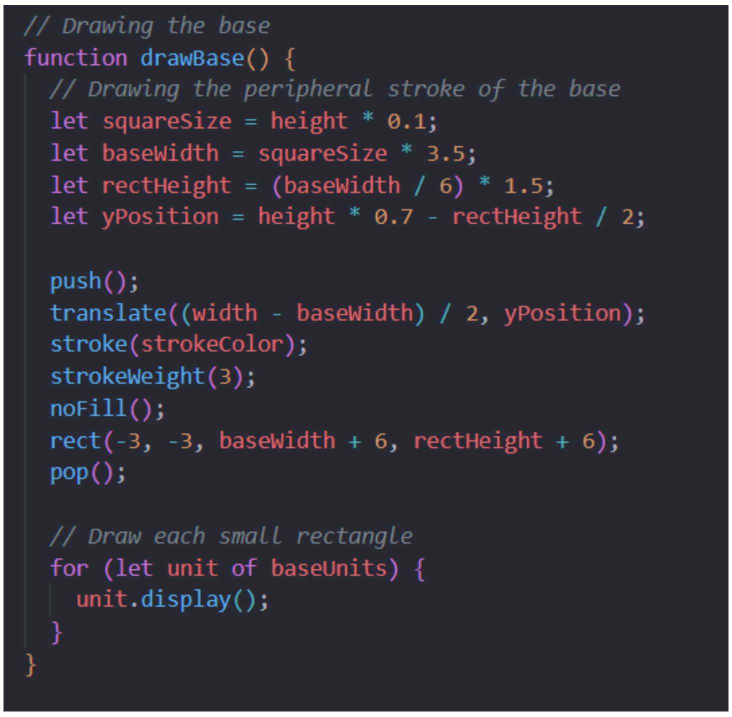
```
// Draw the bottom five horizontal circles
let currentX = startX;=
for (let i = 0; i < 5; i++) {
  let circleDiameter = diameters[i];
  let bicolorCircle = new BicolorCircle(currentX + circleDiameter / 2, yPosition, circleDiamet
  bicolorCircle.display();
  currentX += circleDiameter;
```



```
  // Draw six vertical circles for the third circle
  if (i === 2) {
    let verticalY = yPosition - circleDiameter / 2;
    // Record data with the fifth circle as the reference
    let fifthVerticalCircleY = null;
    let fifthCircleDiameter = null;

    for (let j = 0; j < 6; j++) {
      let verticalDiameter = random(rectWidth * 0.75, rectWidth * 1.25);

      // Rotate 90 degrees
      push();
      translate(currentX - circleDiameter / 2, verticalY - verticalDiameter / 2);
      rotate(HALF_PI);
      let verticalCircle = new BicolorCircle(0, 0, verticalDiameter);
```

draw base

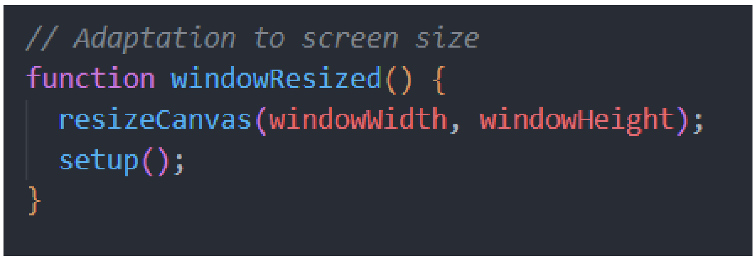
drawBase() outlines the base and calls display() on each RectangleUnit.



```
// Drawing the base
function drawBase() {
  // Drawing the peripheral stroke of the base
  let squareSize = height * 0.1;
  let baseWidth = squareSize * 3.5;
  let rectHeight = (baseWidth / 6) * 1.5;
  let yPosition = height * 0.7 - rectHeight / 2;

  push();
  translate((width - baseWidth) / 2, yPosition);
  stroke(strokeColor);
  strokeWeight(3);
  noFill();
  rect(-3, -3, baseWidth + 6, rectHeight + 6);
  pop();

  // Draw each small rectangle
  for (let unit of baseUnits) {
    unit.display();
  }
}
```

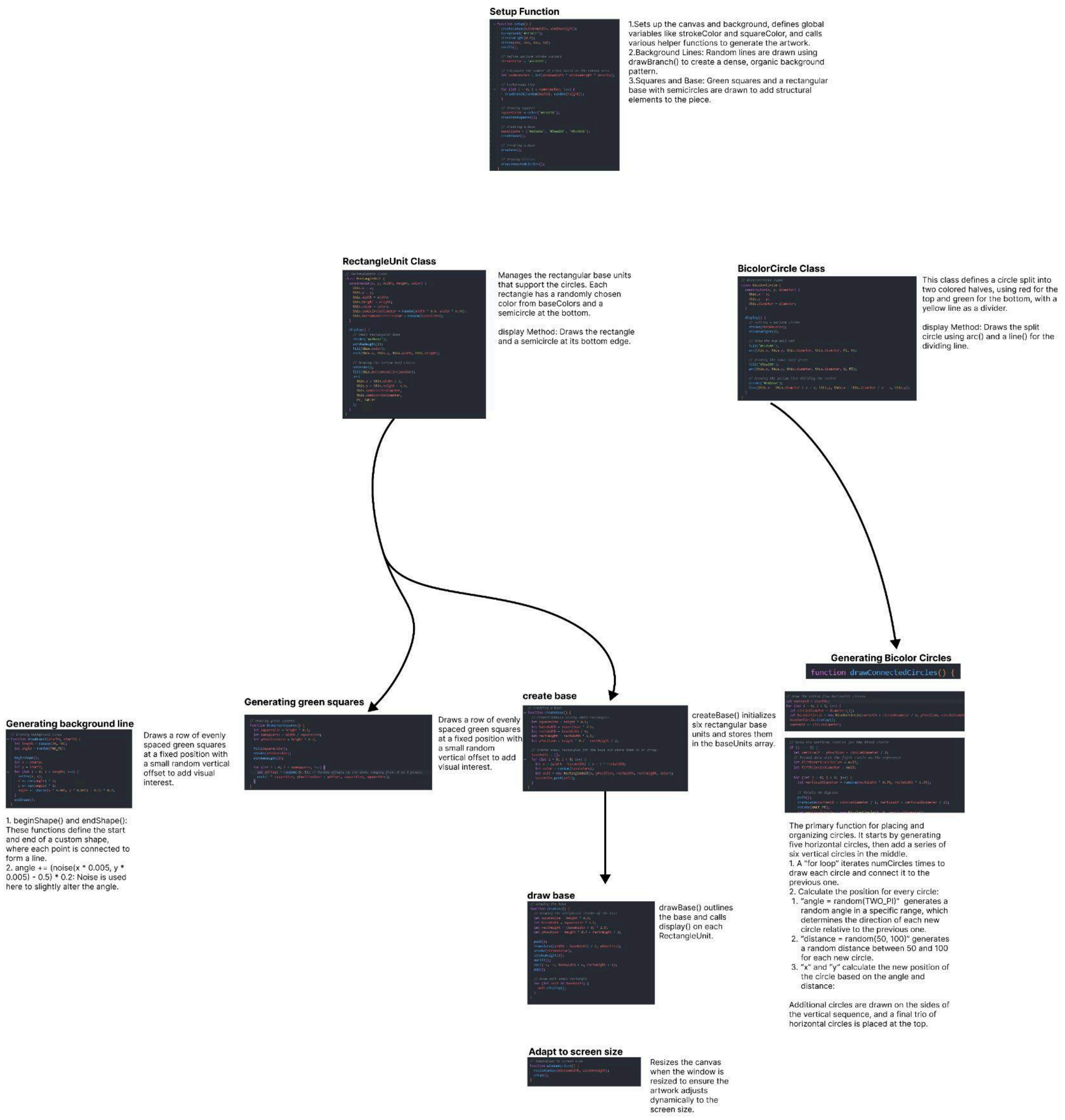
Adapt to screen size

Resize the canvas when the window is resized to ensure the artwork adjusts dynamically to the screen size.



```
// Adaptation to screen size
function windowResized() {
  resizeCanvas(windowWidth, windowHeight);
  setup();
}
```

visual map



Setup Function

1.Sets up the canvas and background, defines global variables like strokeColor and squareColor, and calls various helper functions to generate the artwork.
2.Background Lines: Random lines are drawn using drawBranch() to create a dense, organic background pattern.
3.Squares and Base: Green squares and a rectangular base with semicircles are drawn to add structural elements to the piece.

RectangleUnit Class

Manages the rectangular base units that support the circles. Each rectangle has a randomly chosen color from baseColors and a semicircle at the bottom.

display Method: Draws the rectangle and a semicircle at its bottom edge.

BicolorCircle Class

This class defines a circle split into two colored halves, using red for the top and green for the bottom, with a yellow line as a divider.

display Method: Draws the split circle using arc() and a line() for the dividing line.

Generating background line

Draws a row of evenly spaced green squares at a fixed position with a small random vertical offset to add visual interest.

1. beginShape() and endShape(): These functions define the start and end of a custom shape, where each point is connected to form a line.
2. angle += (noise(x * 0.005, y * 0.005) - 0.5) * 0.2: Noise is used here to slightly alter the angle.

Generating green squares

Draws a row of evenly spaced green squares at a fixed position with a small random vertical offset to add visual interest.

create base

createBase() initializes six rectangular base units and stores them in the baseUnits array.

Generating Bicolor Circles

```
function drawConnectedCircles() {
```

The primary function for placing and organizing circles. It starts by generating five horizontal circles, then add a series of six vertical circles in the middle.
1. A "for loop" iterates numCircles times to draw each circle and connect it to the previous one.
2. Calculate the position for every circle:
   1. "angle = random(TWO_PI)" generates a random angle in a specific range, which determines the direction of each new circle relative to the previous one.
   2. "distance = random(50, 100)" generates a random distance between 50 and 100 for each new circle.
   3. "x" and "y" calculate the new position of the circle based on the angle and distance:

Additional circles are drawn on the sides of the vertical sequence, and a final trio of horizontal circles is placed at the top.

draw base

drawBase() outlines the base and calls display() on each RectangleUnit.

Adapt to screen size

Resizes the canvas when the window is resized to ensure the artwork adjusts dynamically to the screen size.

Acknowledge: We acknowledge that we used ChatGPT to assist us in improving our written expression and correcting grammar errors in this assignment.