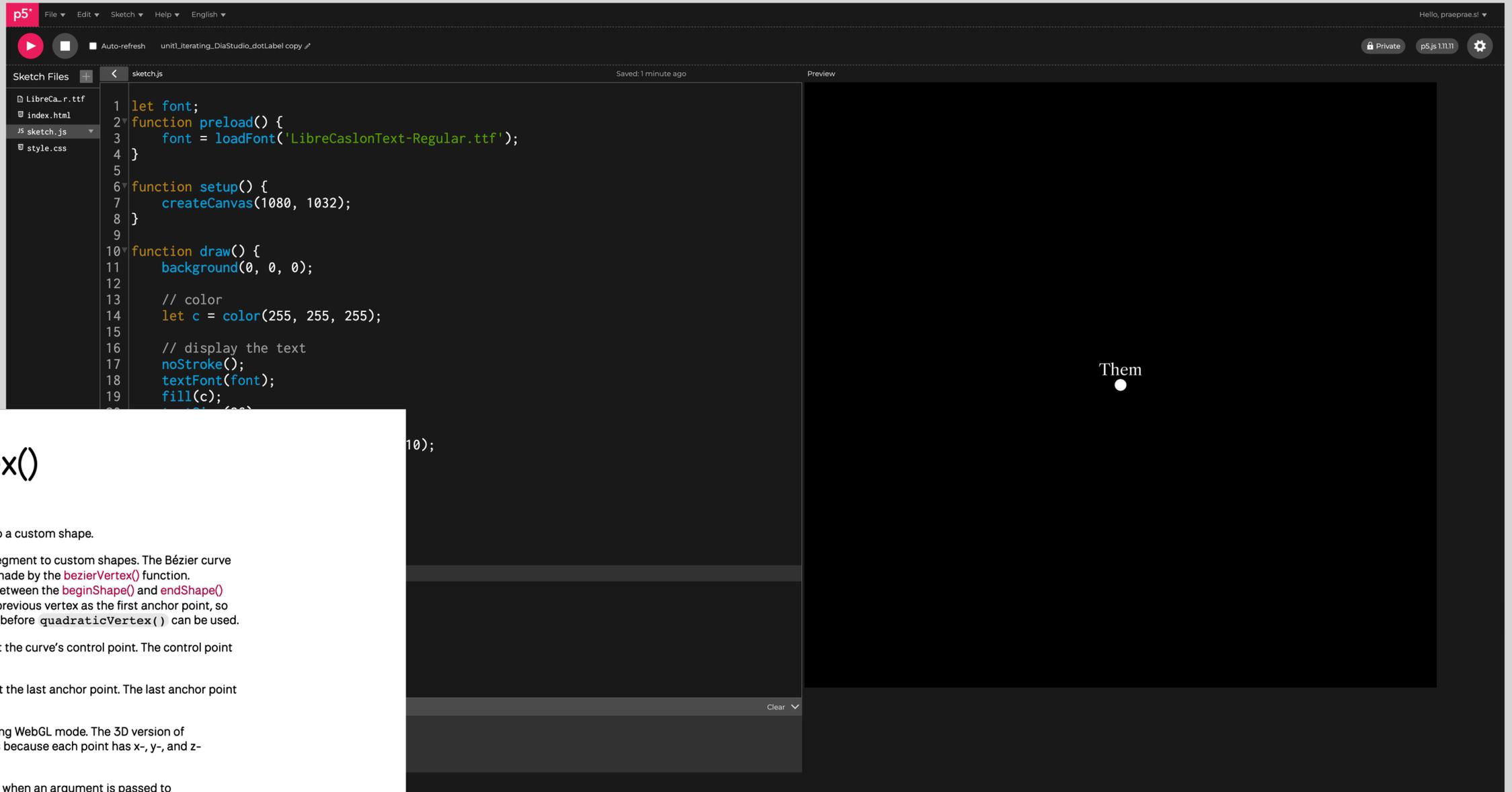


Unit 1

Methods of Iterating week 2

Selected Tool

p5.js



Reference > quadraticVertex()

quadraticVertex()

Adds a quadratic Bézier curve segment to a custom shape.

`quadraticVertex()` adds a curved segment to custom shapes. The Bézier curve segments it creates are similar to those made by the `bezierVertex()` function. `quadraticVertex()` must be called between the `beginShape()` and `endShape()` functions. The curved segment uses the previous vertex as the first anchor point, so there must be at least one call to `vertex()` before `quadraticVertex()` can be used.

The first two parameters, `cx` and `cy`, set the curve's control point. The control point "pulls" the curve towards its.

The last two parameters, `x3`, and `y3`, set the last anchor point. The last anchor point is where the curve ends.

Bézier curves can also be drawn in 3D using WebGL mode. The 3D version of `bezierVertex()` has eight arguments because each point has x-, y-, and z-coordinates.

Note: `quadraticVertex()` won't work when an argument is passed to `beginShape()`.

Examples



```
function setup() {
  createCanvas(100, 100);

  background(200);

  // Style the curve.
  noFill();

  // Draw the curve.
  beginShape();
  vertex(20, 20);
  quadraticVertex(80, 20, 50, 50);
  endShape();

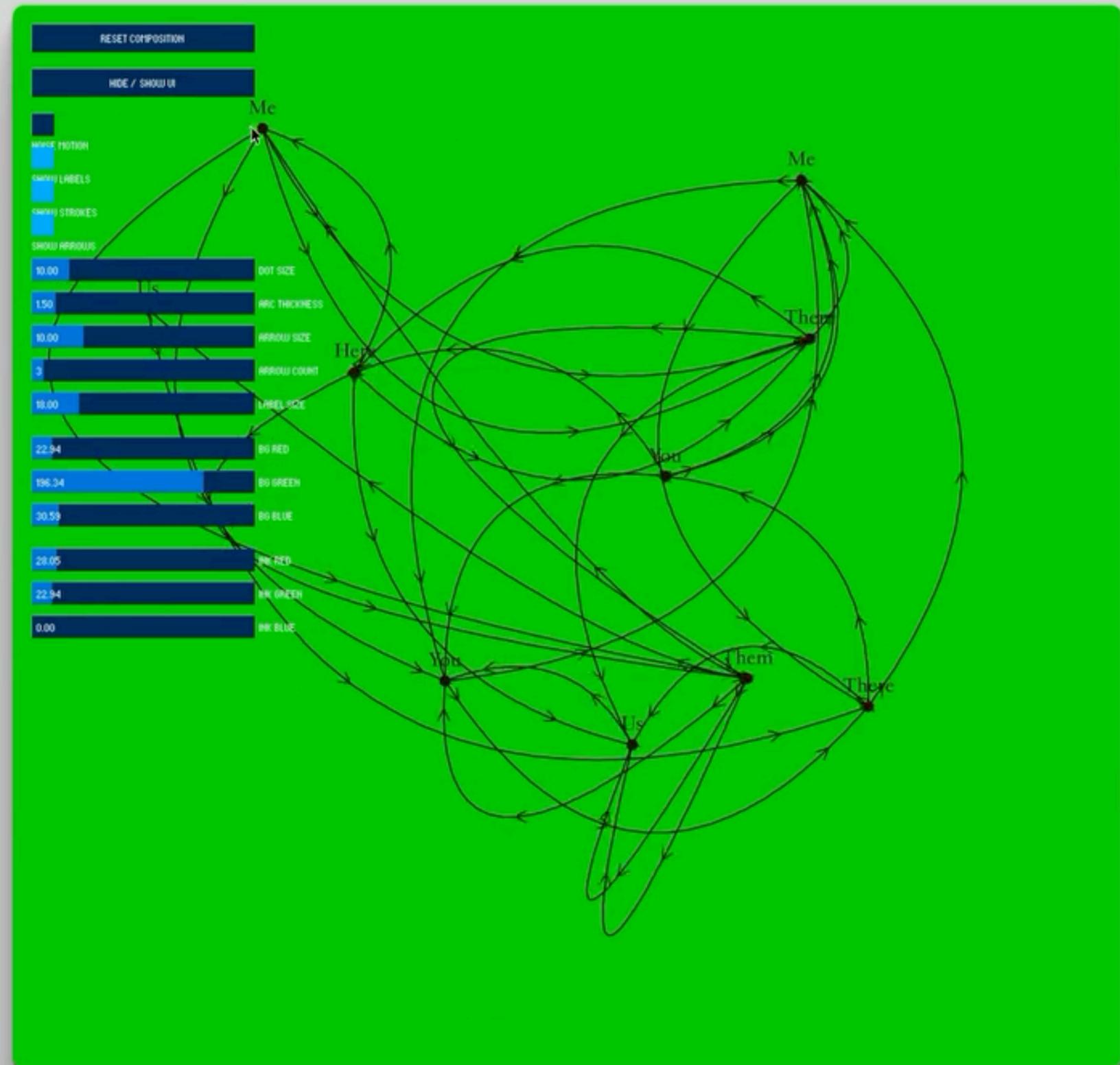
  describe('A black curve drawn on a gray square. The curve starts at the top-left corner and ends at the center.');
```

p5.js is an open-source JavaScript library and web editor that simplifies coding, making it easy to learn through its visual documentation and community example projects.

Selected Project

Us,
Me,
You,
Them,
Here,
There

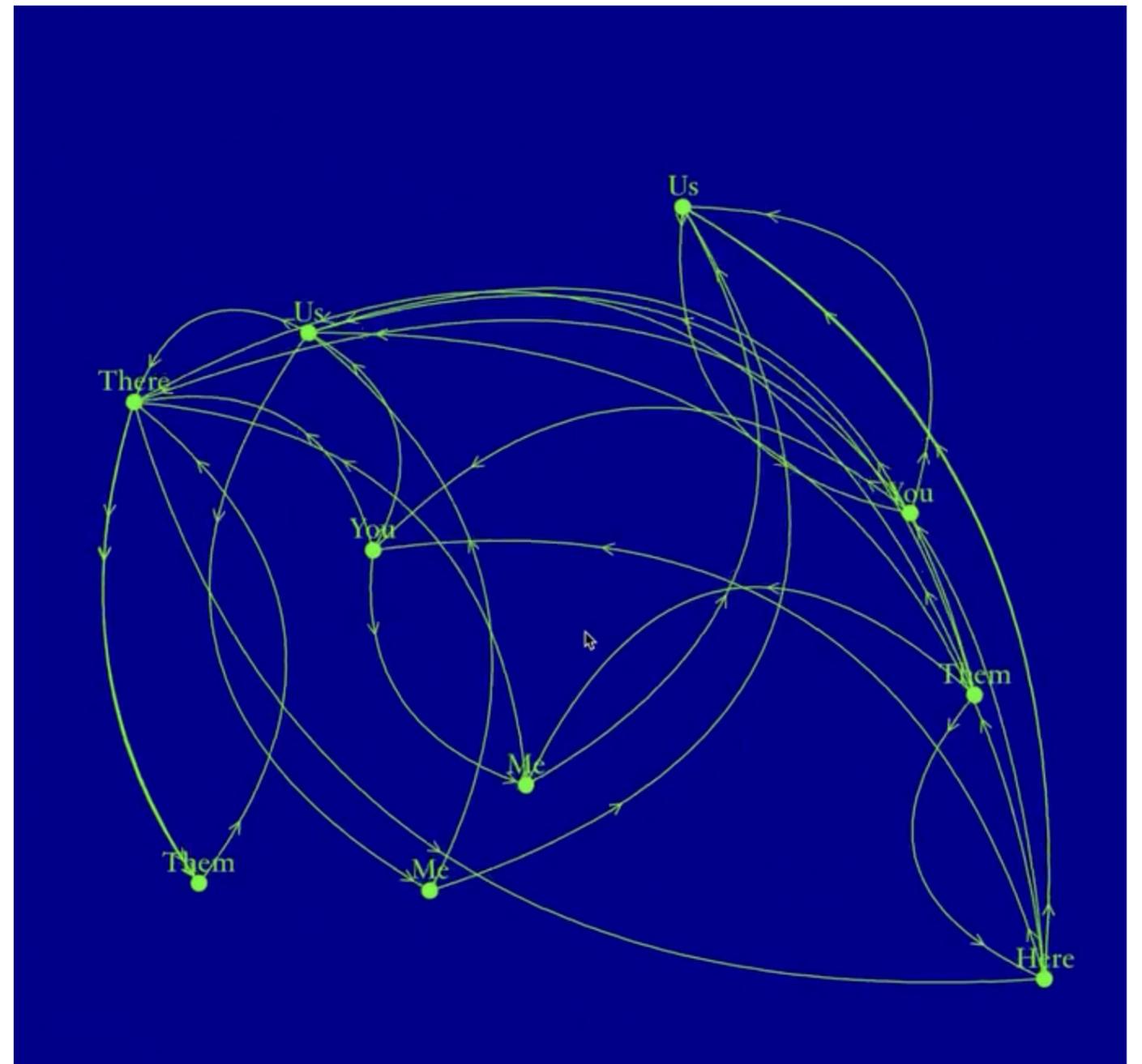
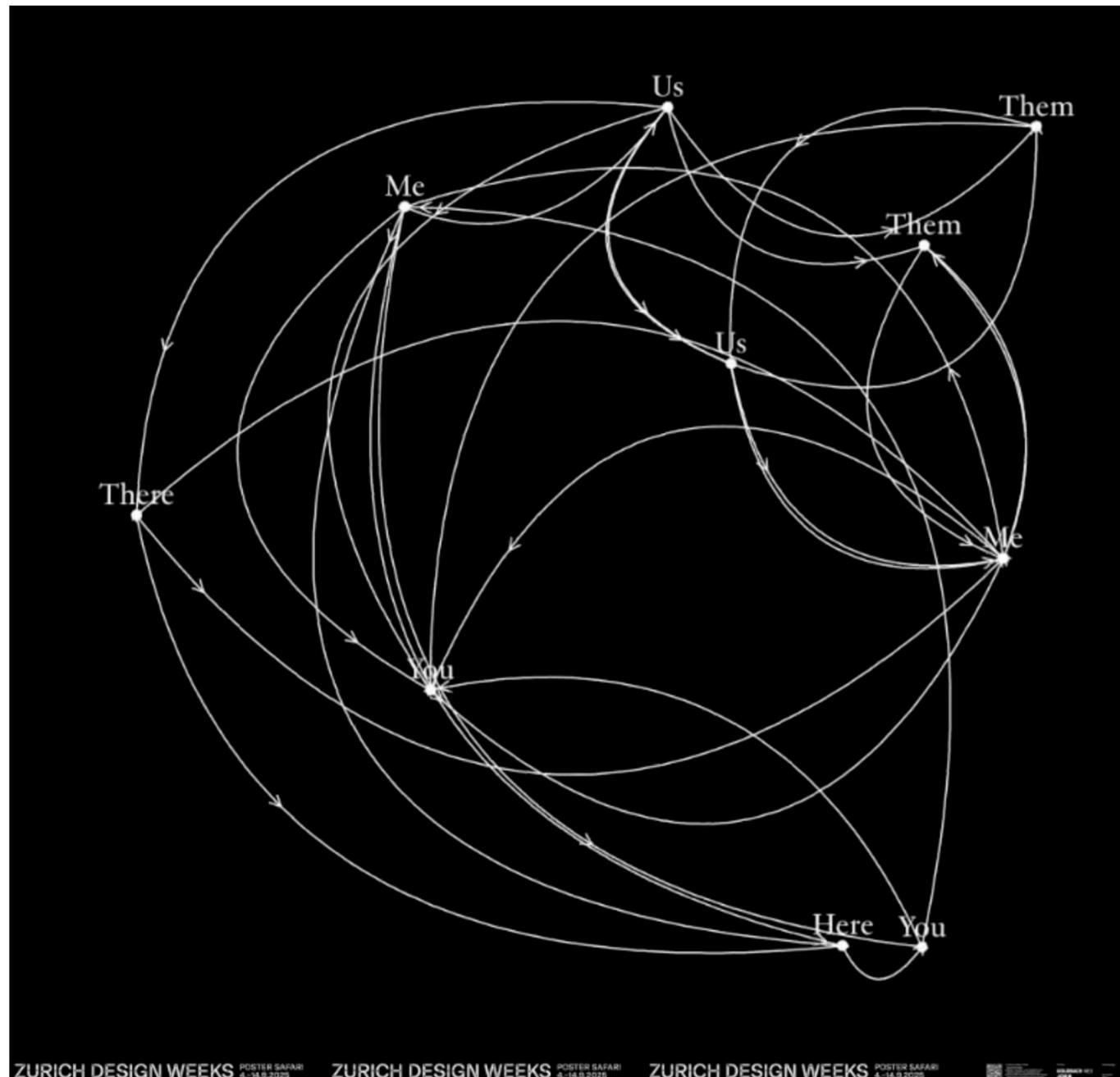
A generative design application for a poster featuring the theme 'TEAM UP' at Zurich Design Week by DIA Studio.



describe ('replicatingProcess');

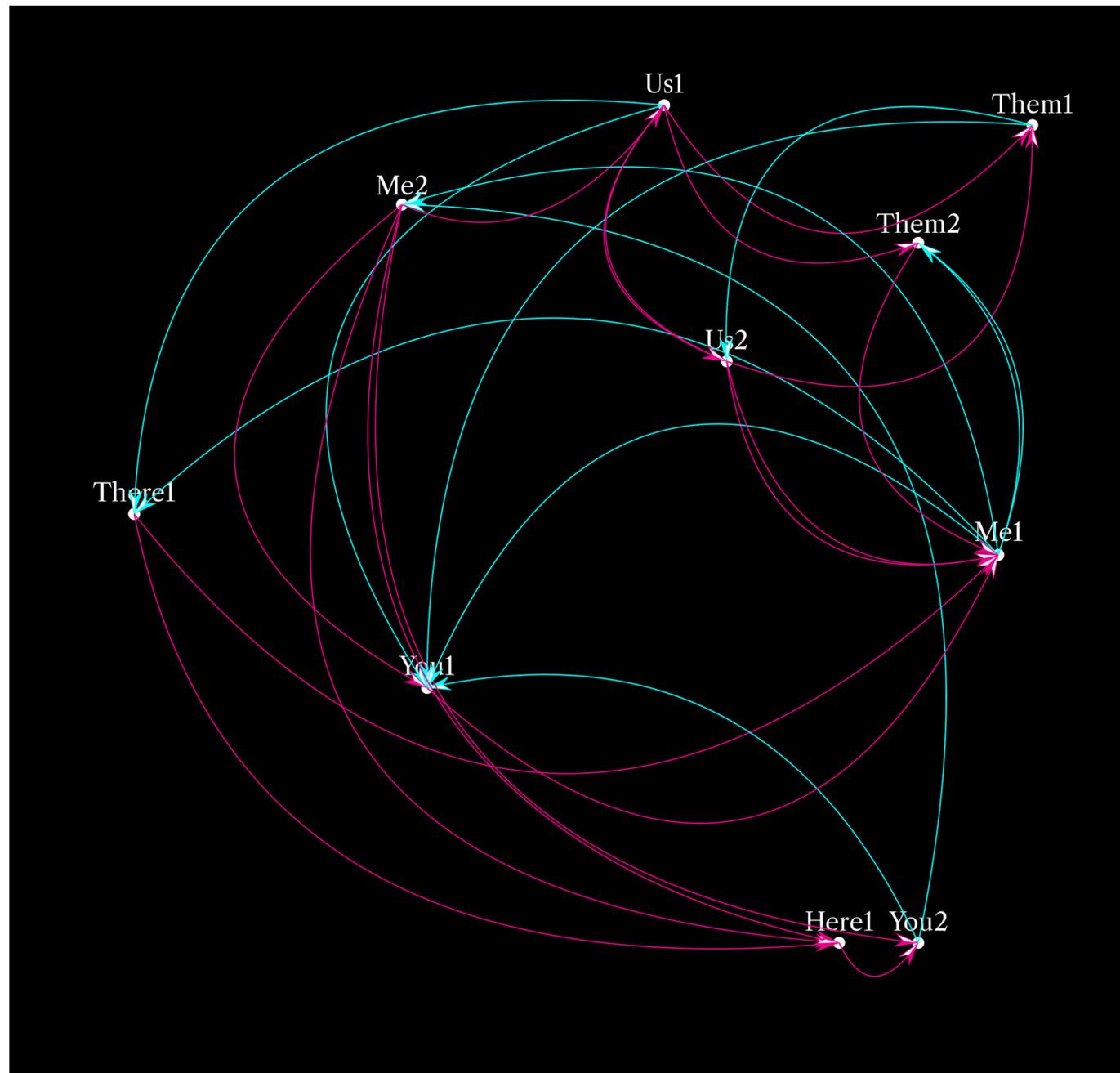
describe ('replicatingProcess');
// Create drafts in Illustrator to analyse the logic behind visual outputs

Screenshots from the chosen project

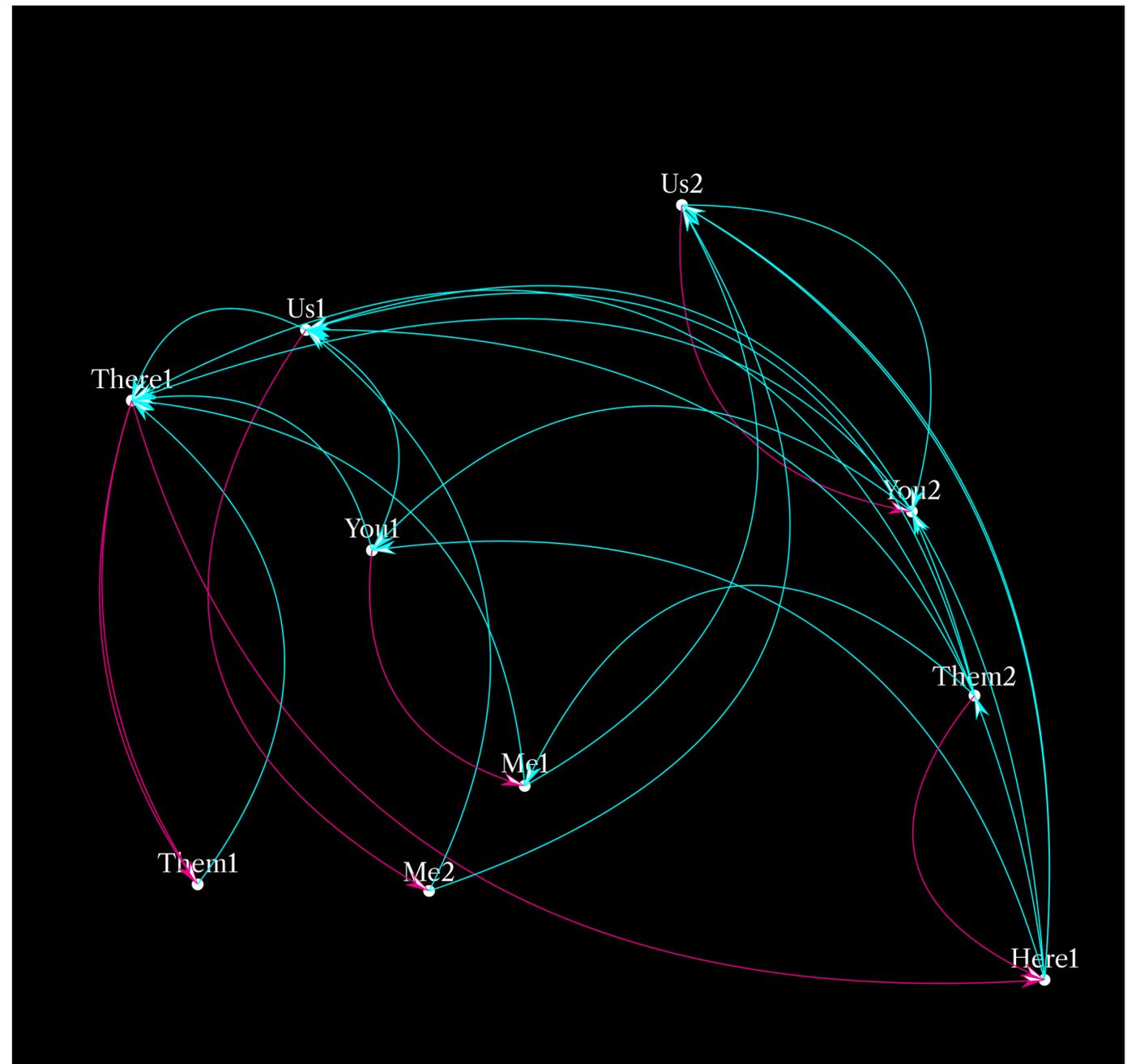


describe ('replicatingProcess');
// Create drafts in Illustrator to analyse the logic behind visual outputs

Draft 1



Draft 2

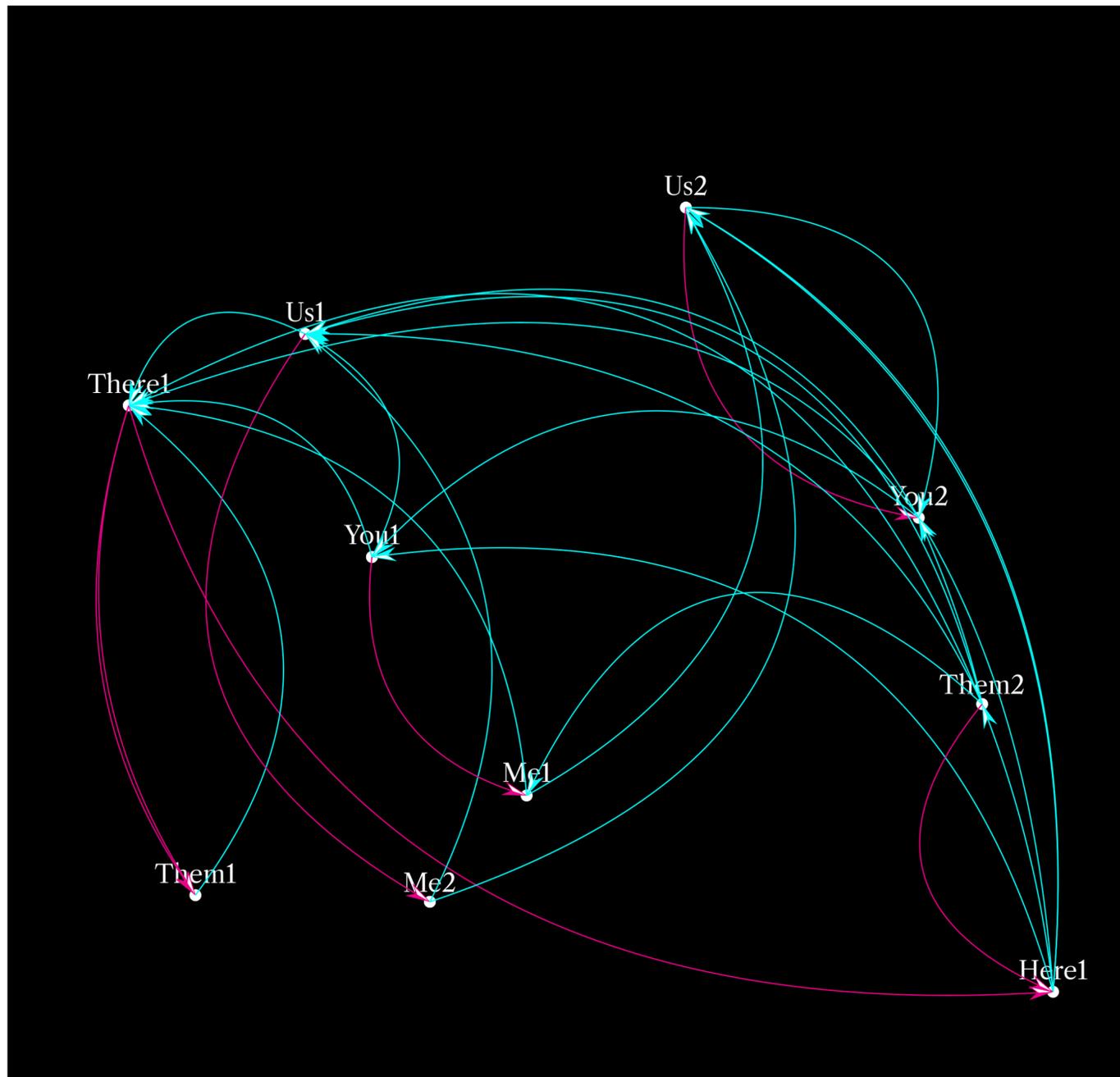


describe ('replicatingProcess');
// Analyse data, logic, and rules

Dataset

6 Words> 10 Dots/Labels> 28 Connections
 3-10 connections per dot

Us,
 Me,
 You,
 Them,
 Here,
 There,
 Us,
 Me,
 You,
 Them



Rules

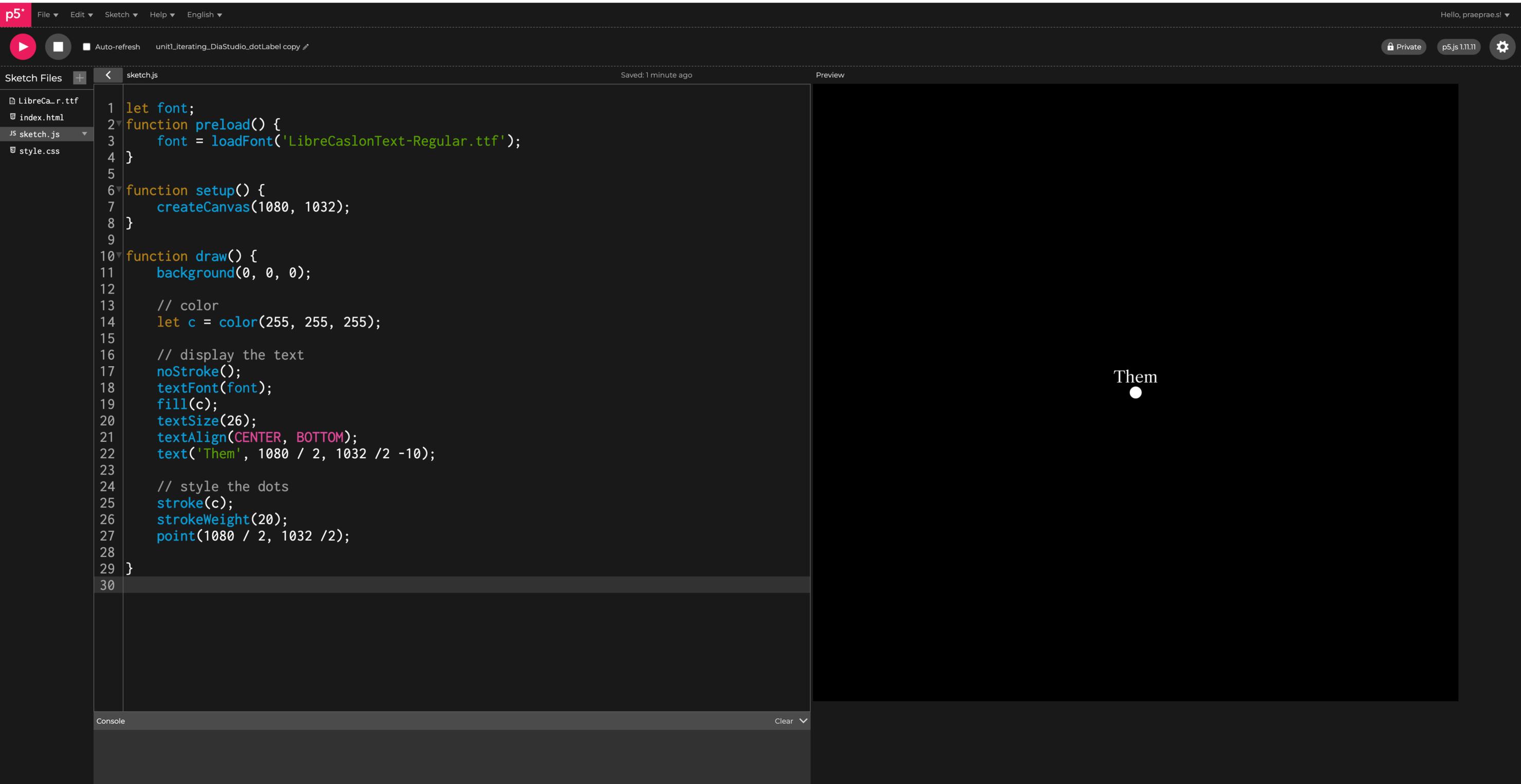
1. Dots are randomly placed but stay inside a margin
2. Labels repeat in a cycle (not random)
3. A dot cannot connect to itself
4. Each connection is a curve with varying depth

	A	B		A	B
1	From	to	1	From	to
2	Us1	There1	2	Us1	There1
3	Us1	You1	3	Us1	Me2
4	Us1	Them1	4	Us2	You2
5	Us1	Them1	5	Us2	You2
6	Us1	Us2	6	Me1	Us2
7	Us1	Us2	7	Me1	There1
8	Us2	Me1	8	Me2	Us1
9	Us2	Me1	9	Me2	Us2
10	Us2	Them1	10	You1	There1
11	Me1	Them2	11	You1	Us1
12	Me1	Them2	12	You1	Me1
13	Me1	You1	13	You2	Us1
14	Me1	There1	14	You2	There1
15	Me1	Me2	15	You2	You1
16	Me2	You1	16	Them1	There1
17	Me2	Us1	17	Them2	Me1
18	Me2	Here1	18	Them2	Us1
19	Me2	Here1	19	Them2	Us1
20	Me2	You2	20	Them2	There1
21	You1	Me1	21	Them2	You2
22	You2	You1	22	Here1	You1
23	You2	Me2	23	Here1	Them2
24	Them1	Us2	24	Here1	You2
25	Them1	You1	25	Here1	Us2
26	Them2	Me1	26	Here1	Us2
27	Here1	You2	27	There1	Them1
28	There1	Here1	28	There1	Them1
29	There1	Me1	29	There1	Here1

describe ('replicatingProcess'); // Set up the canvas and create the most basic elements

Main Sketch 1

This is the easiest part of my process because the syntax in the p5.js library is well-documented. I found that some syntax is simple for designers to understand since it uses familiar vocabulary, such as draw(), arc(), and text(). However, even when creating the simplest elements, I had to write many lines of code.



The screenshot shows the p5.js IDE interface. The left sidebar displays the file explorer with the following files: LibreCa... r.ttf, index.html, JS sketch.js, and style.css. The main editor area shows the code for sketch.js, which is as follows:

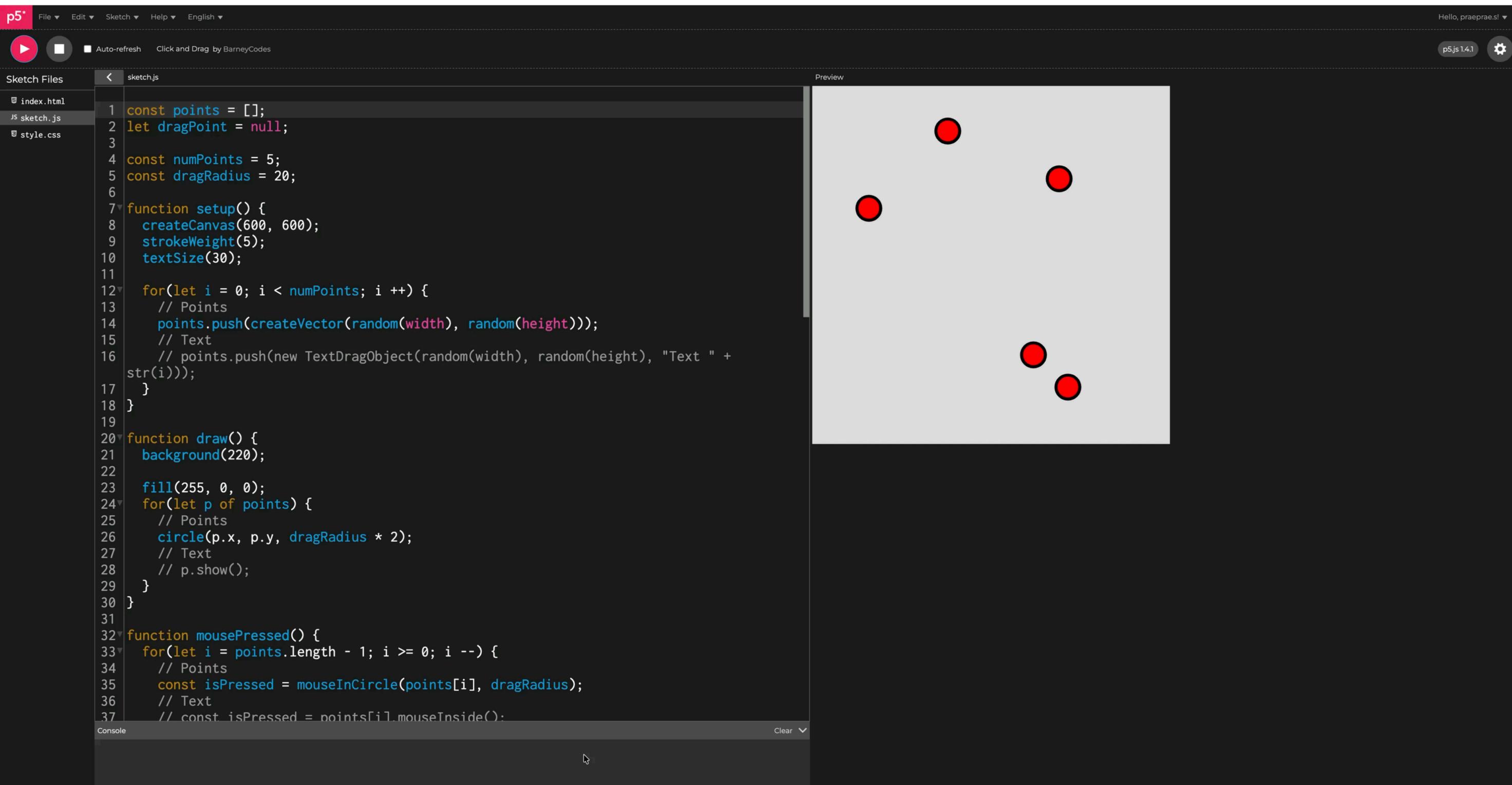
```
1 let font;
2 function preload() {
3   font = loadFont('LibreCaslonText-Regular.ttf');
4 }
5
6 function setup() {
7   createCanvas(1080, 1032);
8 }
9
10 function draw() {
11   background(0, 0, 0);
12
13   // color
14   let c = color(255, 255, 255);
15
16   // display the text
17   noStroke();
18   textFont(font);
19   fill(c);
20   textSize(26);
21   textAlign(CENTER, BOTTOM);
22   text('Them', 1080 / 2, 1032 / 2 - 10);
23
24   // style the dots
25   stroke(c);
26   strokeWeight(20);
27   point(1080 / 2, 1032 / 2);
28
29 }
30
```

The right side of the IDE shows a preview of the sketch. The output is a black canvas with the word "Them" centered horizontally and vertically. Below the text, there is a small white dot, which is the result of the point() function call in the code.

```
describe ('replicatingProcess');  
// Create randomised labels > Finding an example
```

However, I began to find it challenging to formulate code as the rules became more complex. Therefore, most of my learning revolves around adapting, reusing, and patching existing resources instead of creating everything from the ground up.

Example Project by BarneyCodes



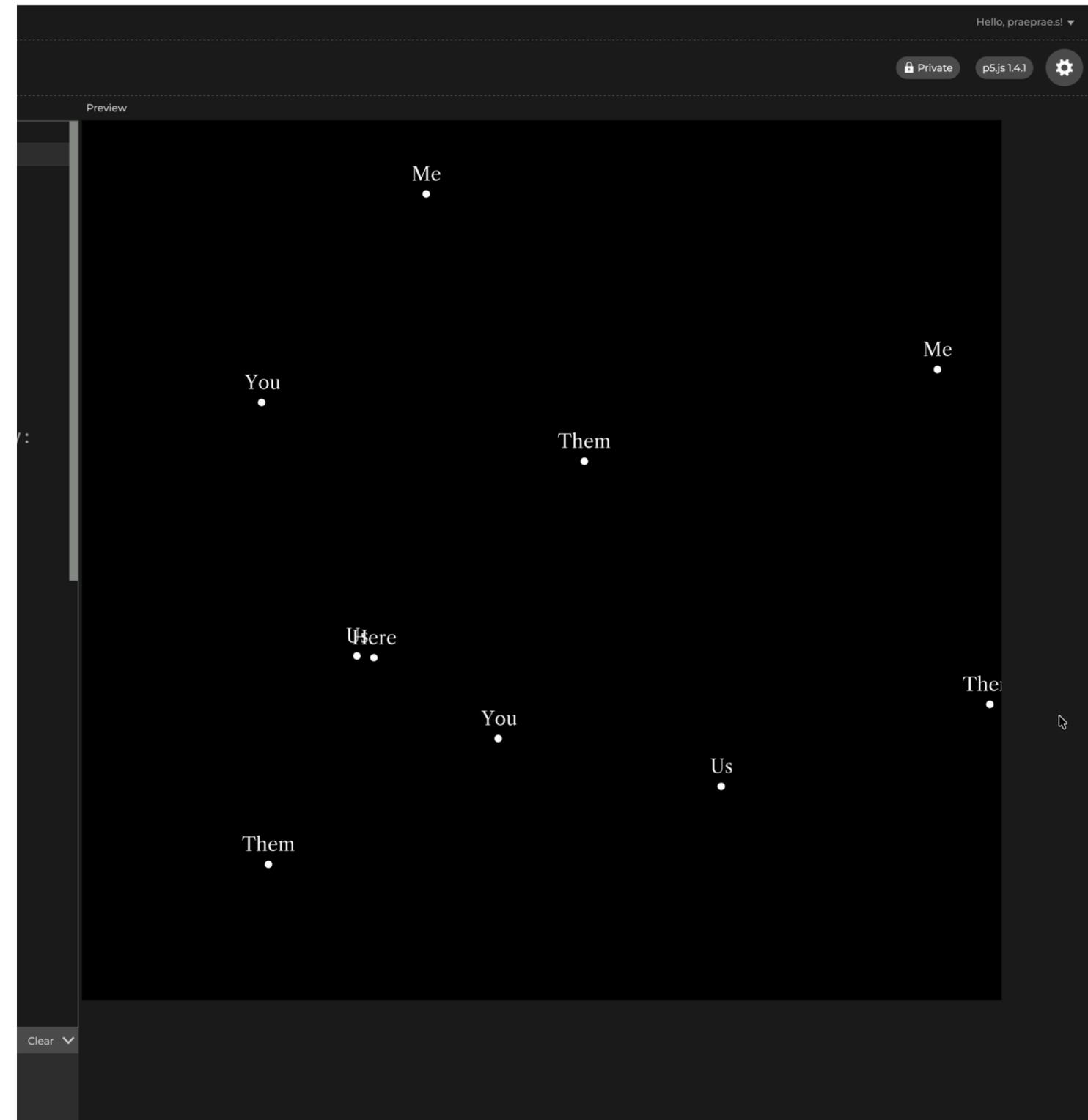
The screenshot shows the p5.js IDE interface. The left sidebar displays the file explorer with 'index.html', 'sketch.js', and 'style.css'. The main editor area shows the following code:

```
1 const points = [];  
2 let dragPoint = null;  
3  
4 const numPoints = 5;  
5 const dragRadius = 20;  
6  
7 function setup() {  
8   createCanvas(600, 600);  
9   strokeWeight(5);  
10  textSize(30);  
11  
12  for(let i = 0; i < numPoints; i++) {  
13    // Points  
14    points.push(createVector(random(width), random(height)));  
15    // Text  
16    // points.push(new TextDragObject(random(width), random(height), "Text " +  
str(i)));  
17  }  
18 }  
19  
20 function draw() {  
21   background(220);  
22  
23   fill(255, 0, 0);  
24   for(let p of points) {  
25     // Points  
26     circle(p.x, p.y, dragRadius * 2);  
27     // Text  
28     // p.show();  
29   }  
30 }  
31  
32 function mousePressed() {  
33   for(let i = points.length - 1; i >= 0; i--) {  
34     // Points  
35     const isPressed = mouseInCircle(points[i], dragRadius);  
36     // Text  
37     // const isPressed = points[i].mouseInside();
```

The right side of the IDE shows a preview window with a light gray background. Five red circles with black outlines are scattered across the canvas, representing the points generated by the code. The top of the IDE shows the p5.js logo, menu options (File, Edit, Sketch, Help, English), and a 'Hello, praeprae!' message. The version 'p5.js 1.4.1' is also visible in the top right corner.

describe ('replicatingProcess');
// Create randomised labels > Implement to my project

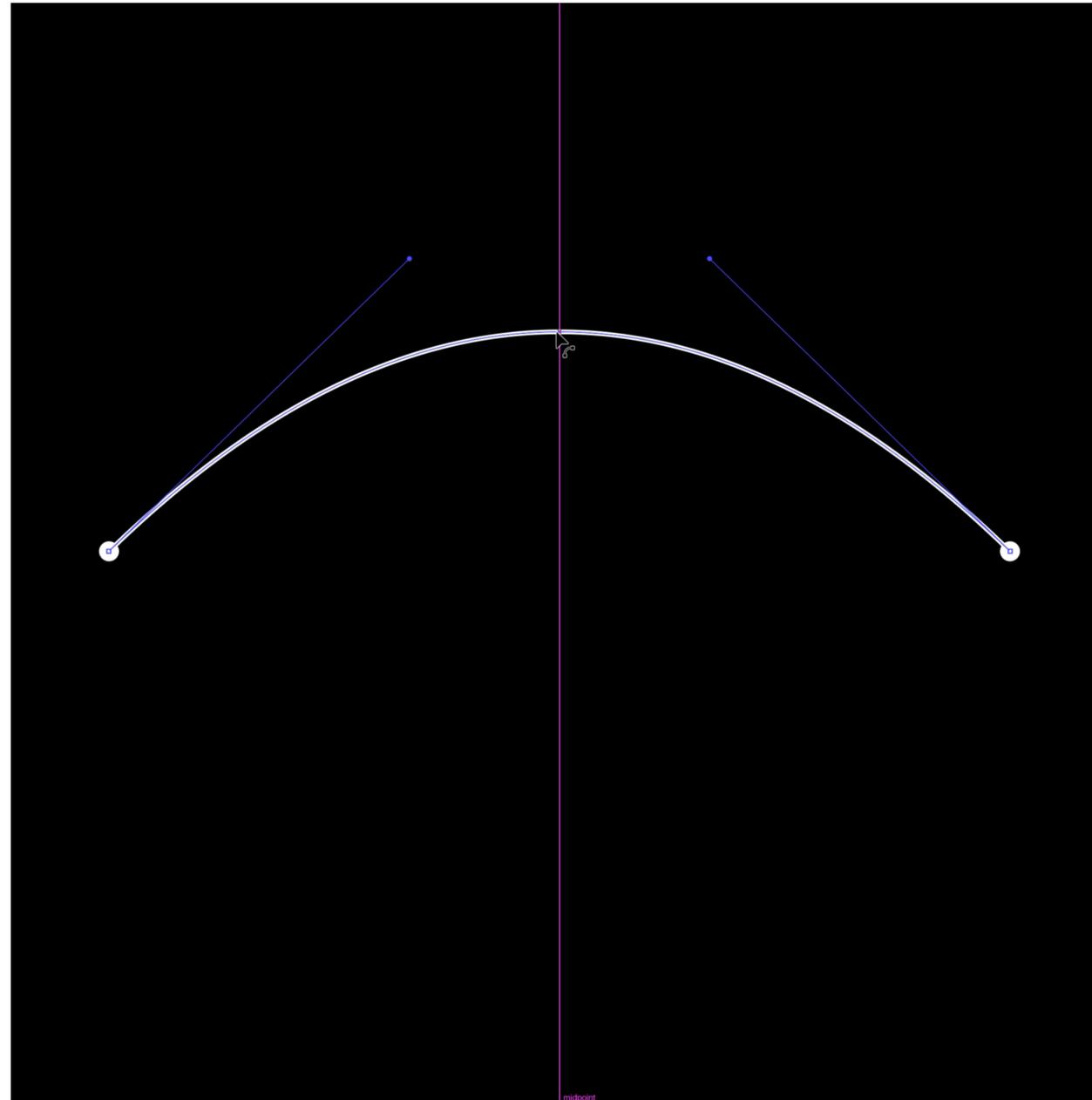
Main Sketch 1



describe ('replicatingProcess');
// How a curve is created in coding

The next step is to find a way to create a curve in coding. While a curve is one of the simplest elements for graphic designers, typically just dragging a line, it's a completely different language when it comes to programming.

Drawing a curve in Illustrator



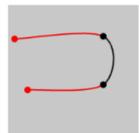
describe ('replicatingProcess'); // How a curve is created in coding

There are four types of syntax that can generate a curve, and the library simplifies maths for the user, so we don't have to calculate everything from scratch. In this case, I chose bezierVertex() to begin with.

Drawing a curve in p5.js

curve()

Examples

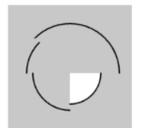


```
function setup() {
  createCanvas(100, 100);

  background(200);

  // Draw a black spline curve.
  noFill();
  strokeWeight(1);
  stroke(0);
  curve(5, 26, 73, 24, 73, 61, 15, 65);
}
```

arc()



```
function setup() {
  createCanvas(100, 100);

  background(200);

  // Bottom-right.
  arc(50, 55, 50, 50, 0, HALF_PI);

  noFill();

  // Bottom-left.
  arc(50, 55, 60, 60, HALF_PI, PI);

  // Top-left.
  arc(50, 55, 70, 70, PI, PI + QUARTER_PI);
}
```

quadraticVertex()

Examples



```
function setup() {
  createCanvas(100, 100);

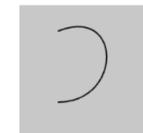
  background(200);

  // Style the curve.
  noFill();

  // Draw the curve.
  beginShape();
  vertex(20, 20);
  quadraticVertex(80, 20, 50, 50);
  endShape();
}
```

bezierVertex()

Examples



```
function setup() {
  createCanvas(100, 100);

  background(200);

  // Style the shape.
  noFill();

  // Start drawing the shape.
  beginShape();

  // Add the first anchor point.
  vertex(30, 20);

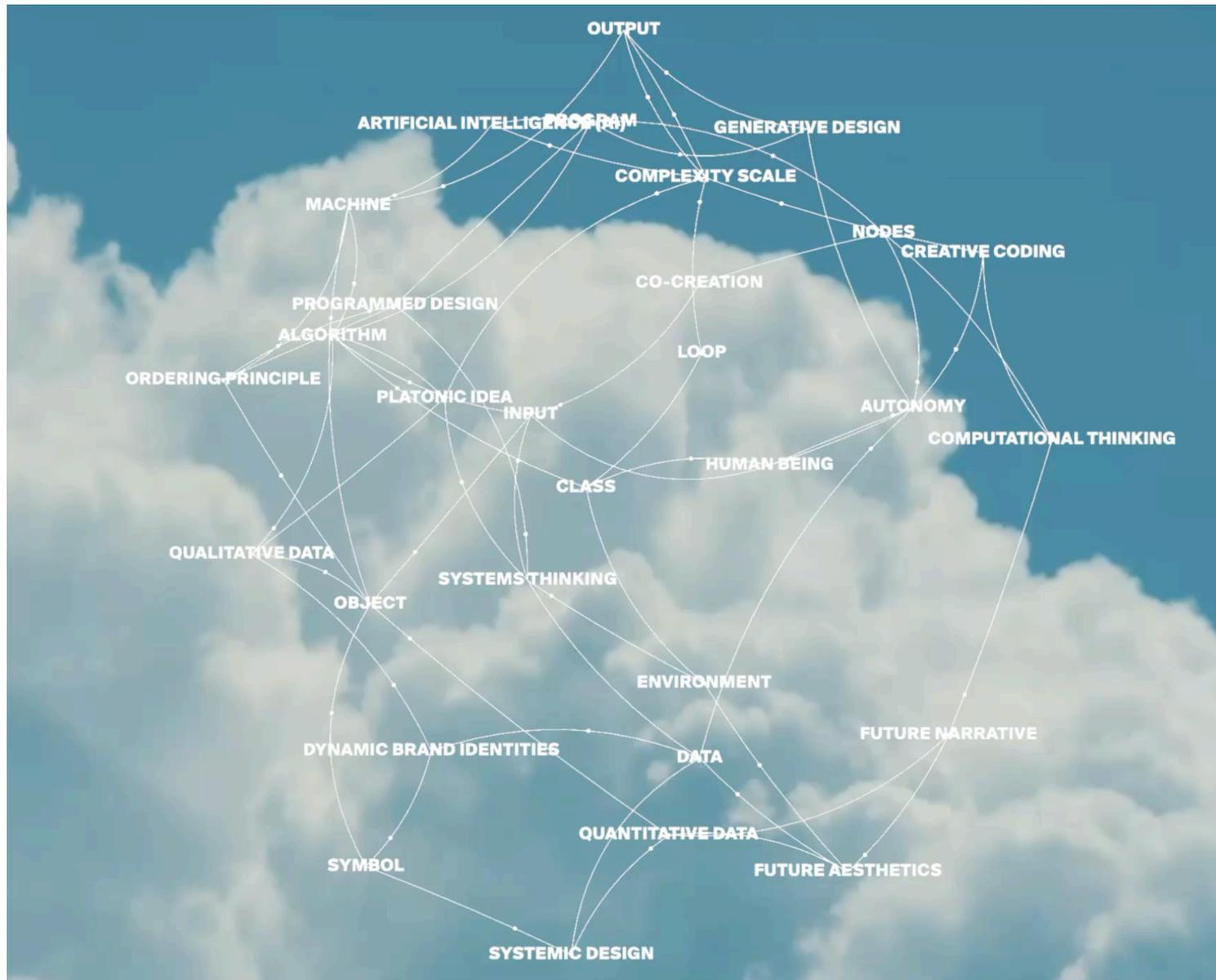
  // Add the Bézier vertex.
  bezierVertex(80, 0, 80, 75, 30, 75);

  // Stop drawing the shape.
  endShape();
}
```

describe ('replicatingProcess');
// Find a case study on how dots are connected

I don't know how to link these curves to random dots, so I had to find a case study, examine their code, and then adapt it into my own code as part of my hacking process.

Data design dictionary by Patrick Hubner



<https://www.patrik-huebner.com/datadesigndictionary/>

```

// Build nodes
let nodes = [];
for (let index in terms) {
  nodes.push({
    id: terms[index].term,
    url: terms[index].url
  })
}

// Build links
let links = [];
for (let i = 0; i < terms.length; i++) {
  for (let j = 0; j < 2; j++) {
    let randomConnection = Math.floor(Math.random()*terms.length);
    if (randomConnection == i) {
      if (i < terms.length-1) randomConnection = i+1; else randomCo
    };
    let linkData = {
      source: terms[i].term,
      target: terms[randomConnection].term,
      curvature: Math.random()*0.3,
    }
    links.push(linkData)
  }
}

// Build data
const gData = {
  nodes: nodes,
  links: links
}

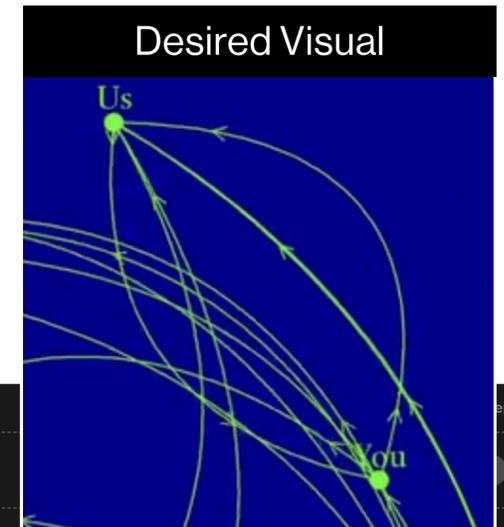
// Create graph
const Graph = ForceGraph()
(document.getElementById('graph'))
  .linkDirectionalParticles(1)
  .linkCurvature('curvature')
  .linkDirectionalParticleSpeed((link)=>{return link.index%4*0.001})
  .linkDirectionalParticleSpeed(0.001)
  .linkDirectionalParticleWidth(2)
  .maxZoom(10)

```

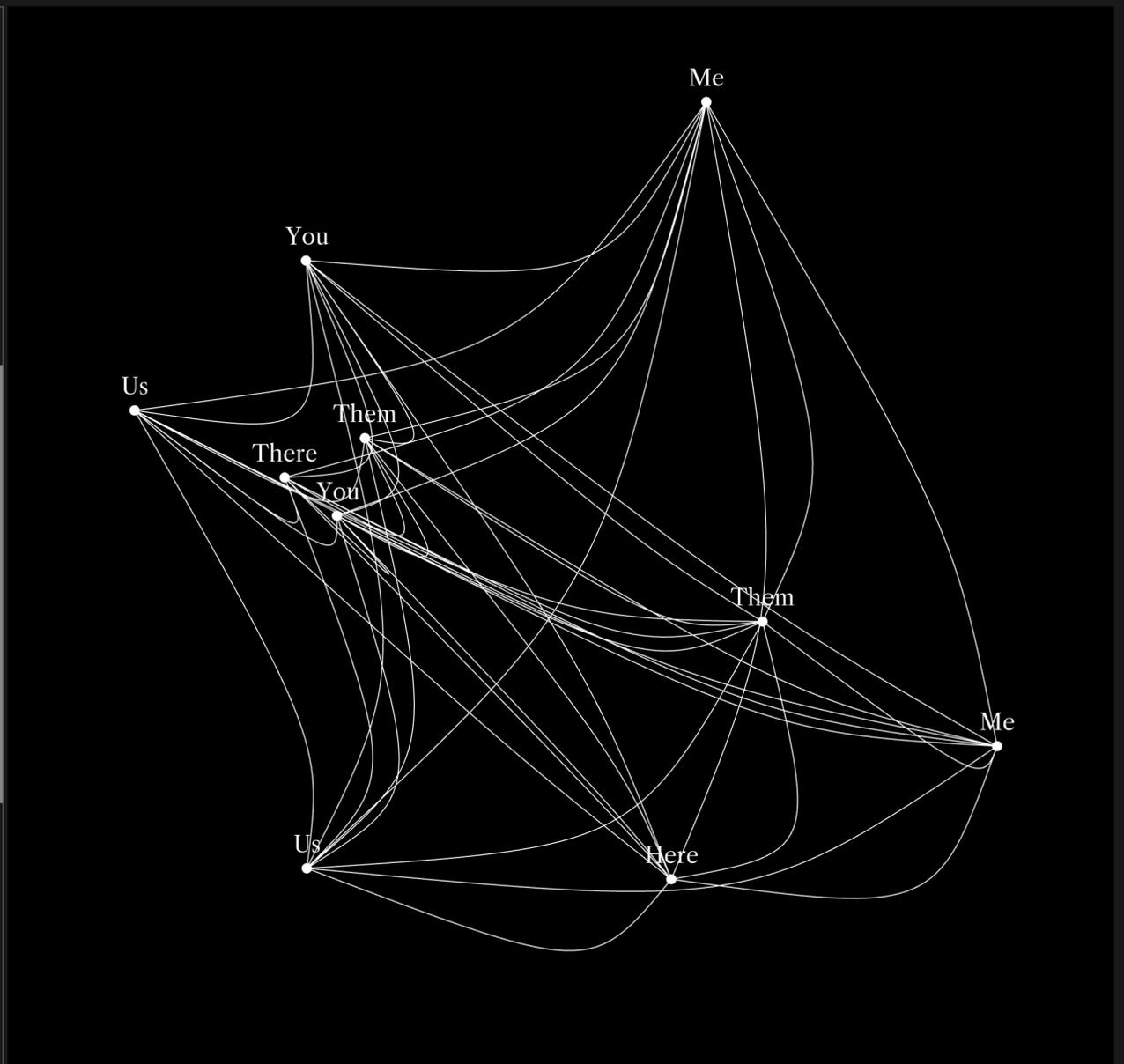
I can't use this as it is syntax from a different library

describe ('replicatingProcess');
// Create curves by bezierVertex()
// Set up rules for creating connections

Main Sketch 2 The outcome was not align with my desired visual



```
p5* File Edit Sketch Help English
Auto-refresh unit_iterating_DiaStudio_connecting_02
Sketch Files sketch.js Saved: 1 day ago Preview
assets
  Libr... r.ttf
  index.html
  JS sketch.js
  style.css
31 }
32 }
33
34 function draw() {
35   background(0, 0, 0);
36   let c = color(255, 255, 255);
37
38   // draw curves and connect dots
39   noFill();
40   stroke(c);
41   strokeWeight(1);
42
43   for(let i = 0; i < points.length; i++) {
44     for(let j = i + 1; j < points.length; j++) {
45       let p1 = points[i].pos;
46       let p2 = points[j].pos;
47
48       // control point is halfway between, but offset up/down
49       let cx = (p1.x + p2.x) / 2 + 100;
50       let cy = (p1.y + p2.y) / 2 + 100; // -50 makes it curve upward
51
52       // draw bezier curve
53       bezier(p1.x, p1.y, cx, cy, cx, cy, p2.x, p2.y);
54     }
55   }
56 }
57
58 // draw dots and text
59 for(let pt of points) {
60   fill(c);
61   noStroke();
62   circle(pt.pos.x, pt.pos.y, dragRadius * 2);
63
64   fill(c);
65   text(pt.label, pt.pos.x, pt.pos.y - dragRadius - 5);
66 }
67 }
68
```



```
describe ('replicatingProcess');
// Create curves by quadraticVertex()
```

Snippet Sketch: Curve

p5* File Edit Sketch Help English

Auto-refresh unit1_iterating_DiaStudio_connecting_01(learn how to draw curve)

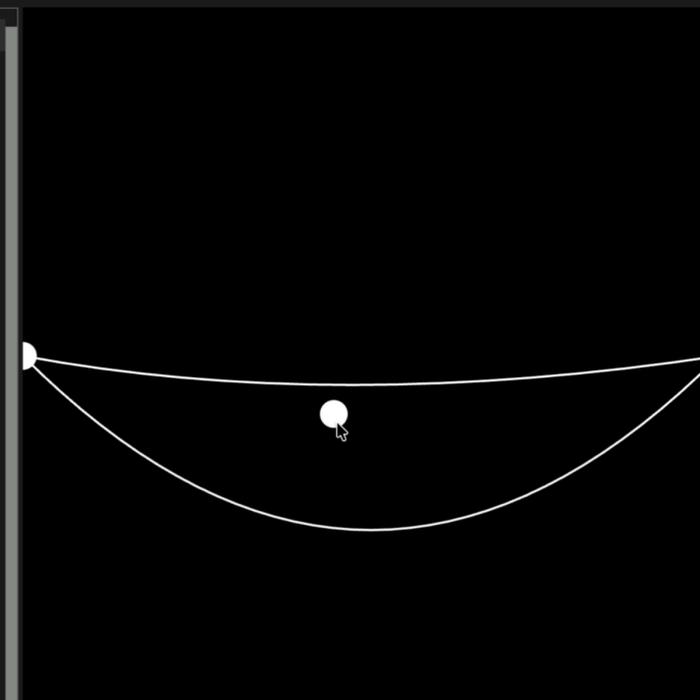
Sketch Files sketch.js Saved: 1 day ago Preview

```

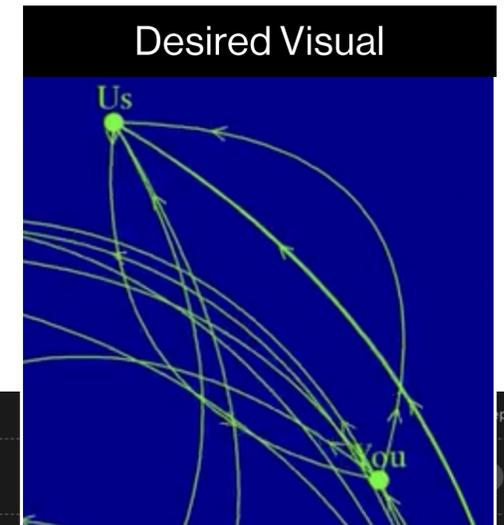
1
2 // Bezier (bezierVertex)
3 // The Coding Train / Daniel Shiffman
4 // https://thecodingtrain.com/CodingChallenges/163-bezier.html
5 // https://youtu.be/zUgsYaUQNBk
6
7 // Basic: https://editor.p5js.org/codingtrain/sketches/Z53a719cQ
8 // Editor by Simon Tiger: https://editor.p5js.org/codingtrain/sketches/_R7RgtGfA
9 // bezierVertex: https://editor.p5js.org/codingtrain/sketches/03_cLi0aw
10 // Time Table Cardioid with Bezier:
11 // https://editor.p5js.org/codingtrain/sketches/kZ8dpk1Qg
12 // Quadratic: https://editor.p5js.org/codingtrain/sketches/fJIMDmHcE
13 // Cubic: https://editor.p5js.org/codingtrain/sketches/S1Pt8lol1
14 // Bezier with Formula: https://editor.p5js.org/codingtrain/sketches/0XOLNHbvC
15 function setup() {
16   createCanvas(600, 600);
17 }
18
19 function draw() {
20   background(0);
21
22   // draw dot
23   stroke(255);
24   strokeWeight(24);
25   point(0, 300); // at the starting
26   point(mouseX, mouseY);
27   point(600, 300); //at the ending
28
29   strokeWeight(2);
30   noFill();
31   beginShape();
32   vertex(0, 300); //x,y position at the starting
33   quadraticVertex(mouseX, mouseY, 600, 300); // last two values are x,y at the end
34   quadraticVertex(300, 600, 0, 300); // first two values are
35   endShape();
36 }
37

```

Console Clear

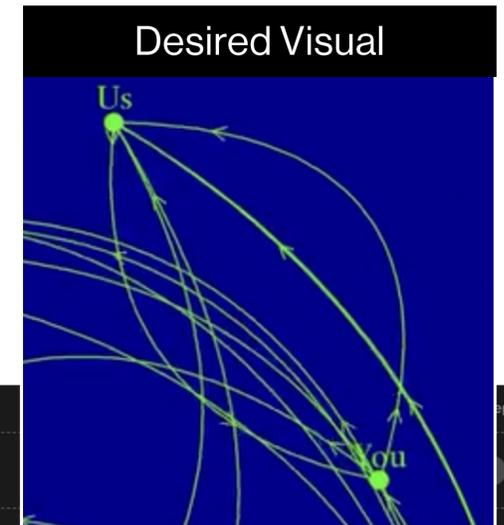


I modified the syntax for creating the curve to `quadraticVertex()` because it better matched my desired visual. I also realised that adapting the code within the main sketch was quite complex, so I separated it into smaller parts. This approach helped me understand how to create the curve and allowed me to focus on the process more effectively.



describe ('replicatingProcess');
// Integret snippet into the main sketch

Main Sketch 3 The outcome matched my desired visual.



```

58
59 // crawl curves and connect dots
60 noFill();
61 stroke(c);
62 strokeWeight(1);
63
64 //loop through all connection
65 for(let conn of connections) {
66   let p1 = points[conn.source].pos; //position of first dot
67   let p2 = points[conn.target].pos; //position of second dot
68
69   // Midpoint of curve (math: combine length and divide with 2 to find the middle
70   // point)
71   let cx = (p1.x + p2.x) / 2; // Horizontal midpoint
72   let cy = (p1.y + p2.y) / 2; // Vertical midpoint
73
74   // Calculate perpendicular offset (90 degree)
75   let dx = p2.x - p1.x;
76   let dy = p2.y - p1.y;
77   let perpX = -dy;
78   let perpY = dx;
79   let length = sqrt(perpX * perpX + perpY * perpY);
80
81   if(length > 0) {
82     // Use the STORED curveOffset instead of random + more number is more curvy
83     perpX = (perpX / length) * conn.curveOffset * 300;
84     perpY = (perpY / length) * conn.curveOffset * 300;
85   }
86
87   cx += perpX;
88   cy += perpY;
89
90   // draw quadratic bezier curve: https://p5js.org/reference/p5/quadraticVertex/
91   beginShape();
92   vertex(p1.x, p1.y);
93   quadraticVertex(cx, cy, p2.x, p2.y);
94   endShape();
95

```

describe ('replicatingProcess'); // Language challenges during integreting snippet into the main sketch

During this process, integrating the snippet took longer because randomness changed the output each run, complicating debugging. Abstract variable names and abbreviations made the logic harder to follow, and I kept referring to the code library and tutorials like a dictionary. It felt less like designing and more like learning a new language of maths and logic, quite different from my usual approach.

Snippet Sketch: Curve

Main Sketch 4

```

1 // Bezier (bezierVertex)
2 // The Coding Train / Daniel Shiffman
3 // https://thecodingtrain.com/CodingChallenges/163-bezier.html
4 // https://youtu.be/zUgsYaUQNBk
5
6
7 // Basic: https://editor.p5js.org/codingtrain/sketches/Z53a719cQ
8 // Editor by Simon Tiger: https://editor.p5js.org/codingtrain/sketches/_R7RgtGfA
9 // bezierVertex: https://editor.p5js.org/codingtrain/sketches/03_cLiOaw
10 // Time Table Cardioid with Bezier:
11 // Quadratic: https://editor.p5js.org/codingtrain/sketches/fJIMDmHcF
12 // Cubic: https://editor.p5js.org/codingtrain/sketches/SiPt3l0lI
13 // Bezier with Formula: https://editor.p5js.org/codingtrain/sketches/030U6bC
14
15 function setup() {
16   createCanvas(600, 600);
17 }
18
19 function draw() {
20   background(0);
21
22   // draw dot
23   stroke(255);
24
25   // The equation for drawing a single quadratic curve.
26   // Most of the variables are numbers
27
28   strokeWeight(2);
29   noFill();
30   beginShape();
31   vertex(0, 300); //x,y position at the starting
32   quadraticVertex(mouseX, mouseY, 600, 300); // last two values are x,y at the end
33   quadraticVertex(300, 600, 0, 300); // first two values are
34   endShape();
35 }
36
37

```

```

58
59
60
61
62
63
64 //loop through all connection
65 for(let conn of connections) {
66   let p1 = points[conn.source].pos; //position of first dot
67   let p2 = points[conn.target].pos; //position of second dot
68
69   // Midpoint of curve (math: combine length and divide with 2 to find the middle point)
70   let cx = (p1.x + p2.x) / 2; // Horizontal midpoint
71   let cy = (p1.y + p2.y) / 2; // Vertical midpoint
72
73   // Calculate perpendicular offset (90 degree)
74   let dx = p2.x - p1.x;
75   let dy = p2.y - p1.y;
76   let perpX = -dy;
77   let perpY = dx;
78   let length = sqrt(perpX * perpX + perpY * perpY);
79
80   if(length > 0) {
81     // Use the STORED curveOffset instead of random + more number is more curvy
82     perpX = (perpX / length) * conn.curveOffset * 300;
83     perpY = (perpY / length) * conn.curveOffset * 300;
84   }
85
86   cx += perpX;
87   cy += perpY;
88
89   // draw quadratic bezier curve: https://p5js.org/reference/p5/quadraticVertex/
90   beginShape();
91   vertex(p1.x, p1.y);
92   quadraticVertex(cx, cy, p2.x, p2.y);
93   endShape();
94 }
95

```

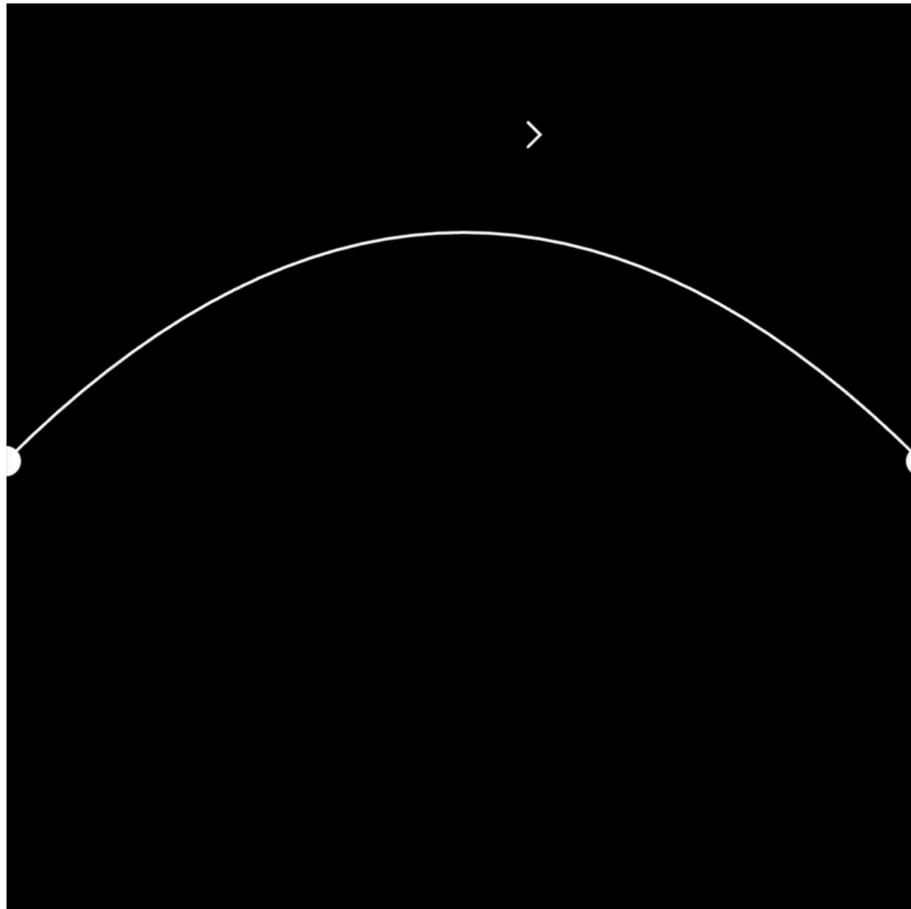
Variables need to be calculated and defined before using them in the main equation.

The equation for drawing a single quadratic curve. Most of the variables are numbers

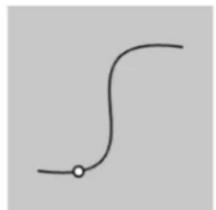
The equation for drawing multiple quadratic curves

describe ('replicatingProcess');
// Create an arrow and make it move on a quadratic curve!

Snippet Sketch: Moving Arrow
1. bezierPoint()



p5.js only has a syntax for bezierPoint



Syntax

```
bezierPoint(a, b, c, d, t)
```

Another challenge I encountered was implementing the mathematics, as the p5.js library lacked functions for calculating points on a quadratic curve. To make my arrow follow the curve, I had to find an appropriate mathematical equation for my code.

2. Calculating quadratic points

Quadratic Bézier Curve: Calculate Points

Asked 14 years, 9 months ago Modified 6 years, 6 months ago Viewed 60k times

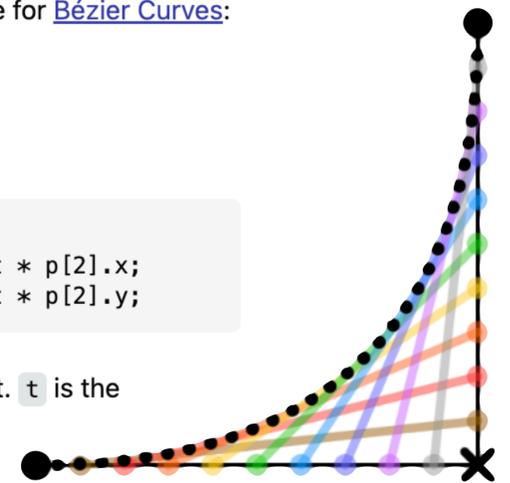
▲ Use the quadratic Bézier formula, found, for instance, on the Wikipedia page for [Bézier Curves](#):

135
$$B(t) = (1 - t)^2P_0 + 2(1 - t)tP_1 + t^2P_2, t \in [0, 1].$$

▼ In pseudo-code, that's

```
t = 0.5; // given example value
x = (1 - t) * (1 - t) * p[0].x + 2 * (1 - t) * t * p[1].x + t * t * p[2].x;
y = (1 - t) * (1 - t) * p[0].y + 2 * (1 - t) * t * p[1].y + t * t * p[2].y;
```

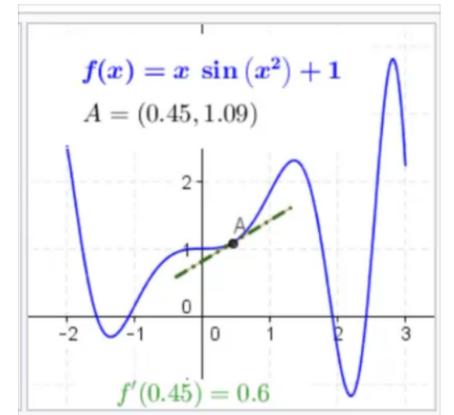
🔖
 ✓
 🔄 $p[0]$ is the start point, $p[1]$ is the control point, and $p[2]$ is the end point. t is the parameter, which goes from 0 to 1.



3. Calculating the arrow's direction

Which immediately gives the derivative of the Bézier curve with respect to t :

$$B'(t) = 2(1 - t)(P_1 - P_0) + 2t(P_2 - P_1),$$

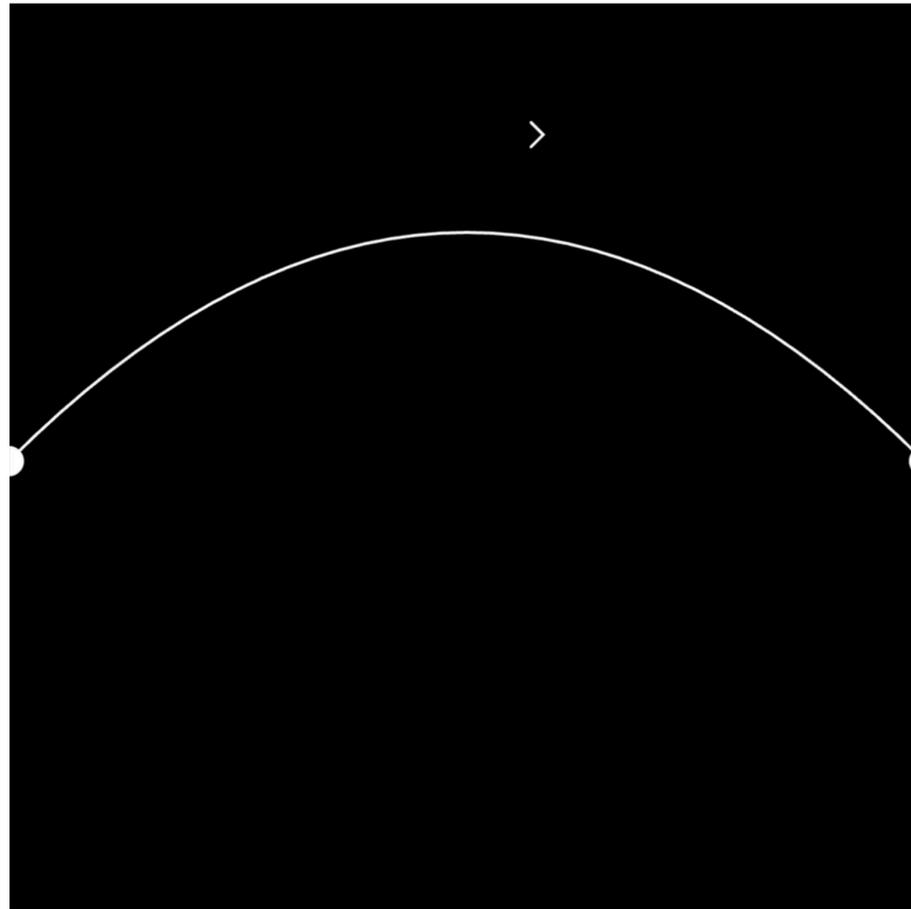


describe ('replicatingProcess');

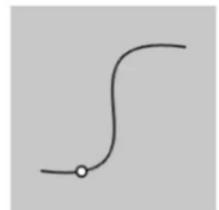
// Create an arrow and make it move on a quadratic curve!

Snippet Sketch: Moving Arrow

1. bezierPoint()



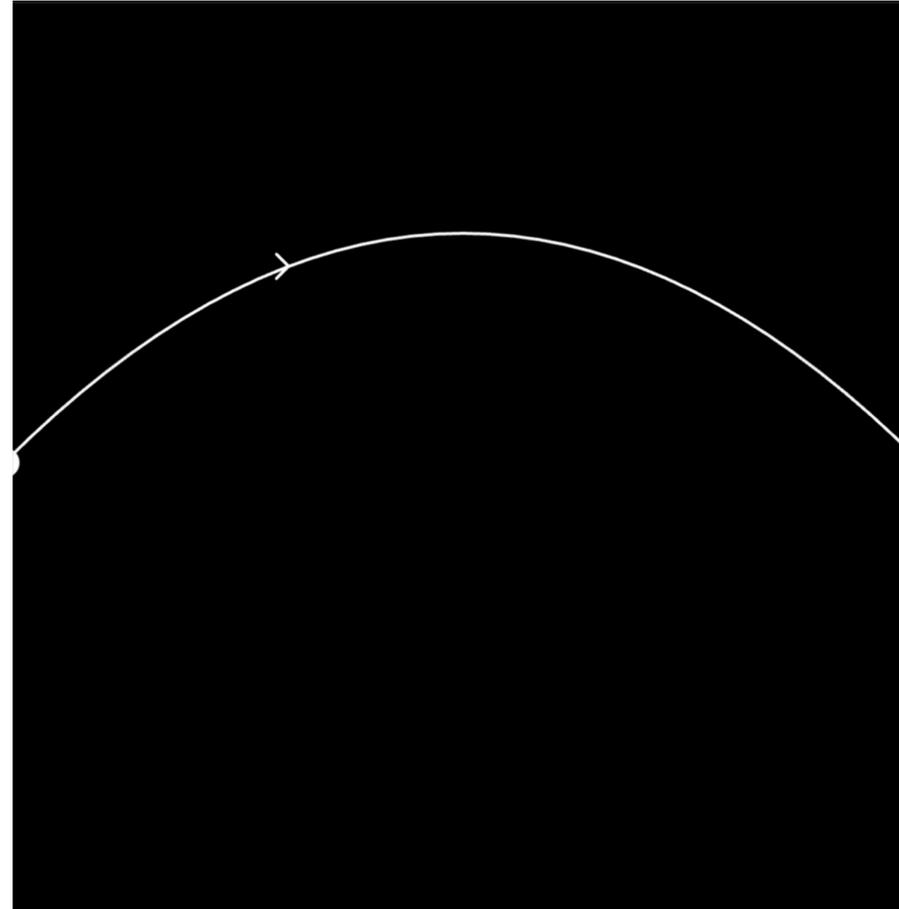
p5.js only has a syntax for bezierPoint



Syntax

```
bezierPoint(a, b, c, d, t)
```

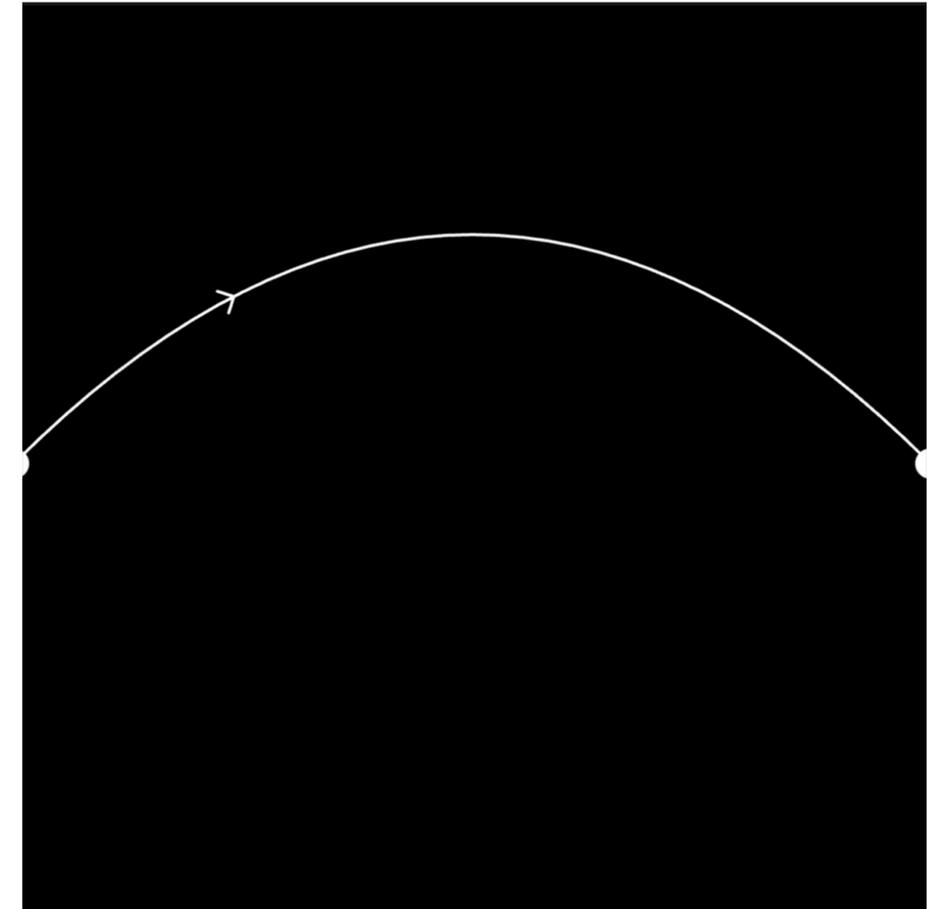
2. Calculating quadratic points



$B(t) = (1 - t)^2\mathbf{P}_0 + 2(1 - t)t\mathbf{P}_1 + t^2\mathbf{P}_2, t \in [0, 1].$

```
x = (1 - t) * (1 - t) * p0.x + 2 * (1 - t) * t * p1.x + t * t * p2.x;
y = (1 - t) * (1 - t) * p0.y + 2 * (1 - t) * t * p1.y + t * t * p2.y;
```

3. Calculating the arrow's direction

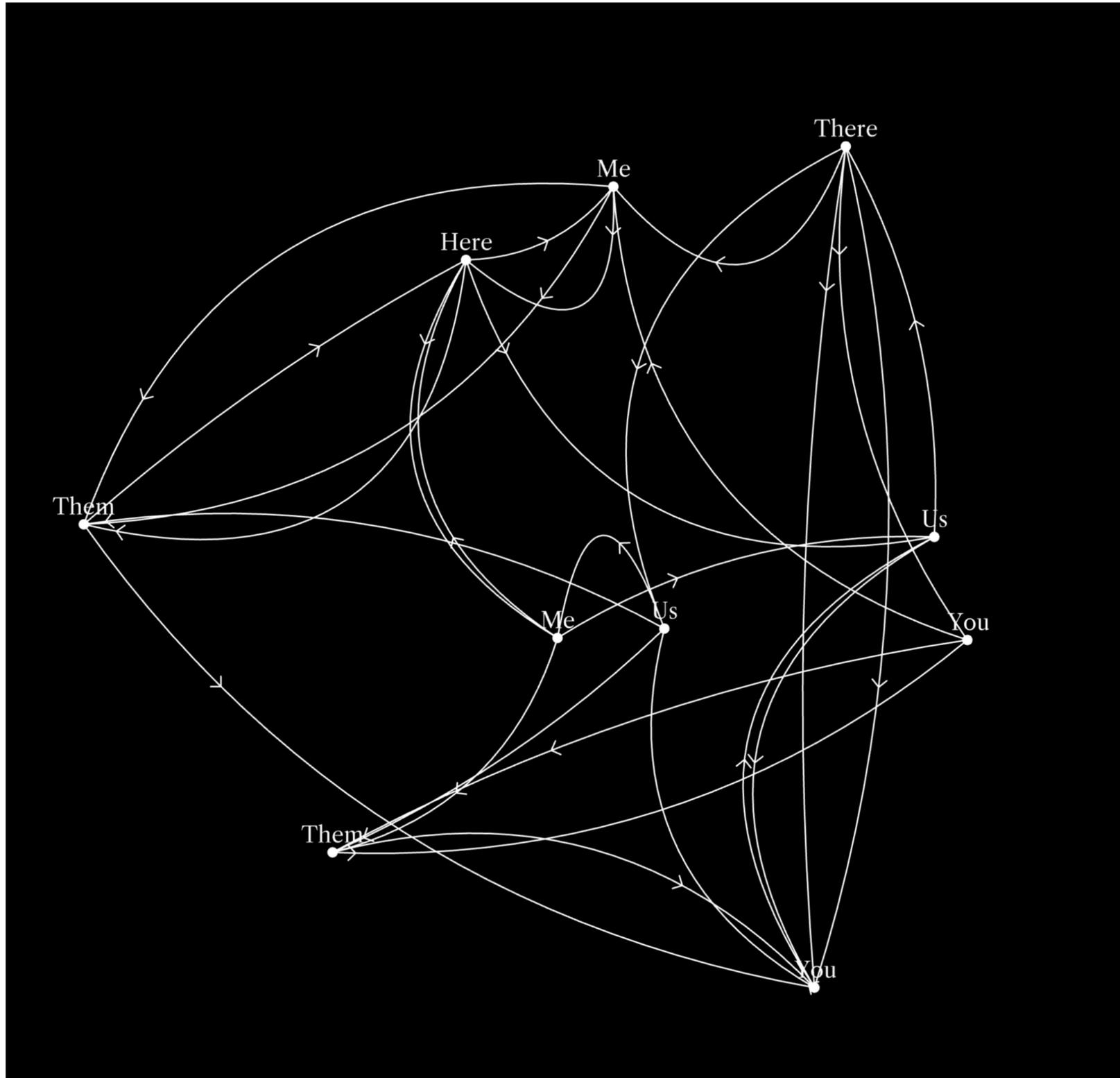


$B'(t) = 2(1 - t)(\mathbf{P}_1 - \mathbf{P}_0) + 2t(\mathbf{P}_2 - \mathbf{P}_1),$

```
let tx = 2 * (1 - t) * (p1.x - p0.x) + 2 * t * (p2.x - p1.x);
let ty = 2 * (1 - t) * (p1.y - p0.y) + 2 * t * (p2.y - p1.y);
let angle = atan2(ty, tx);
```

describe ('replicatingProcess');

// The final output



```
1 // ===== DOCUMENTATION =====
2 // REFERENCES & RESOURCES
3 // Selected project: https://www.instagram.com/p/DOvY-zEnZw/?hlm=en&index=
4 // Array of texts: https://lmsrdenbroeker.de/courses/basic-datastructures/my-five-favorite-cities/
5 // Random point drag tutorial: https://www.youtube.com/watch?v=kjq_u0n87I
6 // Documentation: https://www.alschools.com https://developer.mozilla.org/en-US/
7 // Dot movement: https://www.youtube.com/watch?v=CH90ad1w
8 // Force directed graph: https://www.youtube.com/watch?v=VxU_gJpCk // no use
9 // Click and drag: https://www.youtube.com/watch?v=kjq_u0n87I
10 // Network case study: https://www.patrik-huebner.com/datasigndictionary/
11
12 let font;
13 function preload() {
14   font = loadFont('assets/libreCaslonText-Regular.ttf');
15 }
16
17 // Canvas setting
18 const margin = 70;
19
20 // Dot settings
21 const numDots = 10;
22 const dotRadius = 10;
23 const labelOffset = 4;
24
25 // Arrow settings
26 const arrowSize = 7;
27 const arrowSpeed = 0.009;
28
29 // Curve settings
30 const thickness = 1.5;
31 const bendMin = 20;
32 const bendMax = 250;
33
34 // Data arrays
35 const dots = []; // assign array
36 const connections = [];
37 const labels = ['Me', 'Here', 'Them', 'Us', 'You', 'Here', 'There'];
38
39 function setup() {
40   createCanvas(1000, 1000);
41   textFont(font);
42   textSize(24);
43   textAlign(CENTER, BOTTOM);
44
45   // ===== 1. setup dots =====
46   // .push() method adds one or more elements to the end of an array and returns the new length.
47   // dots is the number used to find a dot in the array.
48   for (let i = 0; i < numDots; i++) {
49     dots.push({
50       position: createVector(
51         random(margin, width - margin),
52         random(margin, height - margin)
53       ),
54       label: labels[i % labels.length], // assign array
55     });
56   }
57
58   // ===== 2. setup connection counts =====
59   // 20 base + 8 extra * 25
60   // each dot starts with 2 outgoing connections
61   let counts = new Array(numDots).fill(2);
62   for (let i = 0; i < 8; i++) counts[i] = counts[i] + 25;
63
64   // ===== 3. build connections =====
65   // REF: https://www.patrik-huebner.com/datasigndictionary/
66   // REF: https://p5js.org/reference/p5.Vector
67   // REF: https://www.youtube.com/watch?v=WHf0pJ_g8t=845 (bezier)
68   // REF: https://javascript.info/bezier-curve
69   for (let startIndex = 0; startIndex < numDots; startIndex++) {
70     for (let i = 0; i < counts[startIndex]; i++) {
71       // pick random target
72       let targetIndex = floor(random(numDots));
73       // prevent self-connection (to next dot)
74       if (targetIndex == startIndex) targetIndex = (startIndex + 1) % numDots;
75
76       // get positions of starting and ending dots
77       let startPos = dots[startIndex].position;
78       let endPos = dots[targetIndex].position;
79
80       // get control point (mid point) using linear interpolation (0.5 = 50%)
81       let controlPoint = p5.Vector.lerp(startPos, endPos, 0.5);
82
83       // calculate the bend direction vector
84       let bend = p5.Vector.sub(endPos, startPos);
85       // create a symmetry curve by perpendicular (rotate 90 degrees)
86       bend.rotate(MUL_7_2);
87
88       // set how far it bends out
89       let bendSize = random(0, 0.5 * 5 * 7 - 1); // go both side 50/50 upward/downward
90       let bendDepth = random(bendMin, bendMax); // pick depth randomly
91       bend.setMag(bendSize * bendDepth);
92
93       // add the offset to the midpoint
94       controlPoint.add(bend);
95
96       // compute control point once and keep it, so curves don't reshape every frame > use this name in draw function
97       connection = {
98         startIndex,
99         controlPoint,
100         targetIndex,
101         arrowPos: random(0, 1),
102       };
103     }
104   }
105
106   function draw() {
107     // set background color
108     // background('#000000'); //black
109     // background('#000000'); //green
110     // background('#000000'); //blue
111
112     // let elementColor = color('FFFFFF'); //white
113     // let elementColor = color('000000'); //black
114     // let elementColor = color('#420058'); //green
115     // let elementColor = color('#000000'); //blue
116
117     // ===== 4. draw connections and moving arrows in loop =====
118     for (let conn of connections) {
119       let p1 = dots[conn.startIndex].position; // start point
120       let p2 = dots[conn.targetIndex].position; // end point
121
122       // draw the curved line (only once per connection)
123       noFill();
124       stroke(elementColor);
125       strokeWeight(thickness);
126
127       beginShape();
128       vertex(p1.x, p1.y);
129       quadraticBezierTo(p1.x, p1.y, p2.x, p2.y);
130       endShape();
131
132       // move the arrow
133       let t = (frameCount * arrowSpeed + conn.arrowPos) % 1;
134
135       // plotting the arrow position (quad point)
136       let tx = quadLerp(p1.x, p1.x, p2.x, t);
137       let ty = quadLerp(p1.y, p1.y, p2.y, t);
138
139       // plotting the arrow direction (derivative tangent)
140       let tx_t = quadTangent(p1.x, p1.x, p2.x, t);
141       let ty_t = quadTangent(p1.y, p1.y, p2.y, t);
142       let angle = atan2(ty_t, tx_t);
143
144       // draw arrow head
145       noFill();
146       stroke(elementColor);
147       strokeWeight(thickness);
148       noStroke();
149
150       beginShape();
151       vertex(-arrowSize, -arrowSize);
152       vertex(0, 0);
153       vertex(arrowSize, arrowSize);
154       endShape();
155     }
156
157     // ===== 5. draw all dots and text =====
158     for (let dot of dots) {
159       fill(elementColor);
160       noStroke();
161       size(dot.position.x, dot.position.y, dotRadius);
162       text(dot.label, dot.position.x, dot.position.y - labelOffset);
163     }
164   }
165
166   // Formula functions
167   function quadLerp(p1, p2, t) {
168     return (1 - t) * (1 - t) * p1 + 2 * (1 - t) * t * p1 + t * t * p2;
169   }
170
171   function quadTangent(p1, p2, t) {
172     return 2 * (1 - t) * (p1 - p2) + 2 * t * (p2 - p1);
173   }
174
175   // REF: Click and drag by BarneyCodes
176   // I can't use his splice as it will bring up a new array and replace with other text
177
178   function mousePressed() {
179     for (let i = dots.length - 1; i >= 0; i--) {
180       let distance = dist(mouseX, mouseY, dots[i].position.x, dots[i].position.y);
181       // added a small padding for easier clicking
182       if (distance < dotRadius * 10) {
183         dragDotIndex = i;
184         break;
185       }
186     }
187   }
188
189   function mouseDragged() {
190     if (dragDotIndex != null) {
191       // limit the dragging so dots stay inside the margin
192       let targetx = constrain(mouseX, margin, width - margin);
193       let targety = constrain(mouseY, margin, height - margin);
194       dots[dragDotIndex].position.set(targetx, targety);
195     }
196   }
197
198   function mouseReleased() {
199     dragDotIndex = null;
200   }
201 }
202
```

describe ('replicatingProcess'); // Reorganise the code's layout

The unexpected shift was realising that organising code became part of my creative process. As the sketch grew complex, my focus shifted from visuals to code, which I constantly restructured, renamed, relayouted, and grouped, not just for maintenance but to improve clarity and revisitability.

Before

```
// 1.Create dots > use .push to begin/contain > REF: BarneyCodes
for(let i = 0; i < numDots; i++) {
  dots.push({
    position: createVector(random(margin, width - margin), random(margin, height - margin)),
    label: labels[i % labels.length] // assign array
  });
}

// 2.Setup connection counts (20 base + 8 extra = 28)
let counts = new Array(numDots).fill(2); // each dot connects to 2 random other dots
for(let i = 0; i < 8; i++) counts[floor(random(numDots))]++;

// 3.Build connections
// REF: https://www.patrik-huebner.com/datadesigndictionary/
// REF: https://p5js.org/reference/p5/p5.Vector/
// REF: https://www.youtube.com/watch?v=enNfb6p3j_g&t=845s (bezier)
// REF: https://javascript.info/bezier-curve
for (let i = 0; i < numDots; i++) {
  for (let j = 0; j < counts[i]; j++) {
    let endingDot = floor(random(numDots)); // Pick random target dot
    // prevent self-connection (to next dot)
    if (endingDot === i) endingDot = (i + 1) % numDots;

    // get positions of starting and ending dots
    let starting = dots[i].position;
    let ending = dots[endingDot].position;

    // get midpoint using linear interpolation (0.5 = 50%)
    let controlPoint = p5.Vector.lerp(starting, ending, 0.5); //define value of control point
    // get direction vector
    let direction = p5.Vector.sub(ending, starting);
    // rotate 90 degrees(perpendicular) > create a symmetry curve
    direction.rotate(HALF_PI);
    // curve go both direction + variable magnitude
    let curveDirection = random() < 0.5 ? -1 : 1; // Pick side
    let curveMagnitude = random(curveMin, curveMax); // Pick depth
    direction.setMag(curveDirection * curveMagnitude); // bend strength

    // Add the offset to the midpoint
    controlPoint.add(direction);

    // contain connection with fixed control point > use this name in draw function
    connections.push({
      startingDot: i,
      endingDot,
      controlPoint
    });
  }
}
```

After

```
// ===== 1.Setup dots =====
// .push() method adds one or more elements to the end of an array and returns the new length.
// dotIndex=the number used to find a dot in the array
for (let i = 0; i < numDots; i++) {
  dots.push({
    position: createVector(
      random(margin, width - margin),
      random(margin, height - margin)
    ),
    label: labels[i % labels.length], // assign array
  });
}

// ===== 2.Setup connection counts =====
// 20 base + 8 extra = 28
// each dot starts with 2 outgoing connections

let counts = new Array(numDots).fill(2);
// add 8 extra randomly
for (let i = 0; i < 8; i++) counts[floor(random(numDots))]++;

// ===== 3.Build connections =====
// REF: https://www.patrik-huebner.com/datadesigndictionary/
// REF: https://p5js.org/reference/p5/p5.Vector/
// REF: https://www.youtube.com/watch?v=enNfb6p3j_g&t=845s (bezier)
// REF: https://javascript.info/bezier-curve

for (let startIndex = 0; startIndex < numDots; startIndex++) {
  for (let i = 0; i < counts[startIndex]; i++) {
    // pick random target
    let targetIndex = floor(random(numDots));
    // prevent self-connection (to next dot)
    if (targetIndex === startIndex) targetIndex = (startIndex + 1) % numDots;

    // get positions of starting and ending dots
    let startPos = dots[startIndex].position;
    let endPos = dots[targetIndex].position;

    // get controlPoint(mid point) using linear interpolation (0.5 = 50%)
    let controlPoint = p5.Vector.lerp(startPos, endPos, 0.5);

    // calculate the bend direction vector
    let bend = p5.Vector.sub(endPos, startPos);
    // create a symmetry curve by drawing it perpendicular (rotate 90 degrees)
    bend.rotate(HALF_PI);

    // set how far it bends out
    let bendSide = random() < 0.5 ? -1 : 1; // go both side (upward/downward)
    let bendDepth = random(bendMin, bendMax); // pick depth randomly
    bend.setMag(bendSide * bendDepth);

    // add the offset to the midpoint
    controlPoint.add(bend);
  }
}
```

describe ('replicatingProcess');
// Rename words and variables

I renamed variables to be more explicit and easier to read later. For example, startIndex replaced misleading startDot, which was an index for a dot's position, not the dot itself. I also made the naming more consistent by replacing mixed names like controlPoint and e for "end" with math-friendly names and added comments for clarity.

Before

startingDot

```
for(let conn of connections) { //loop for single array
  let s = dots[conn.startingDot].position;
  let e = dots[conn.endingDot].position;
  beginShape();
  vertex(s.x, s.y); //insert a randomised starting position
  quadraticVertex(conn.controlPoint.x, conn.controlPoint.y, e.x, e.y);
  endShape();
}
```

After

startIndex

```
// ===== 4.draw connections and moving arrows in loop =====
for (let conn of connections) {
  let p0 = dots[conn.startIndex].position; // start point
  let p1 = conn.controlPoint; // control point
  let p2 = dots[conn.targetIndex].position; // end point

  beginShape();
  vertex(p0.x, p0.y);
  quadraticVertex(p1.x, p1.y, p2.x, p2.y);
  endShape();
}
```

$$\mathbf{B}(t) = (1-t)^2 \mathbf{P}_0 + 2(1-t)t \mathbf{P}_1 + t^2 \mathbf{P}_2, t \in [0, 1].$$

describe ('replicatingProcess');

// The difference between readability for humans vs computers

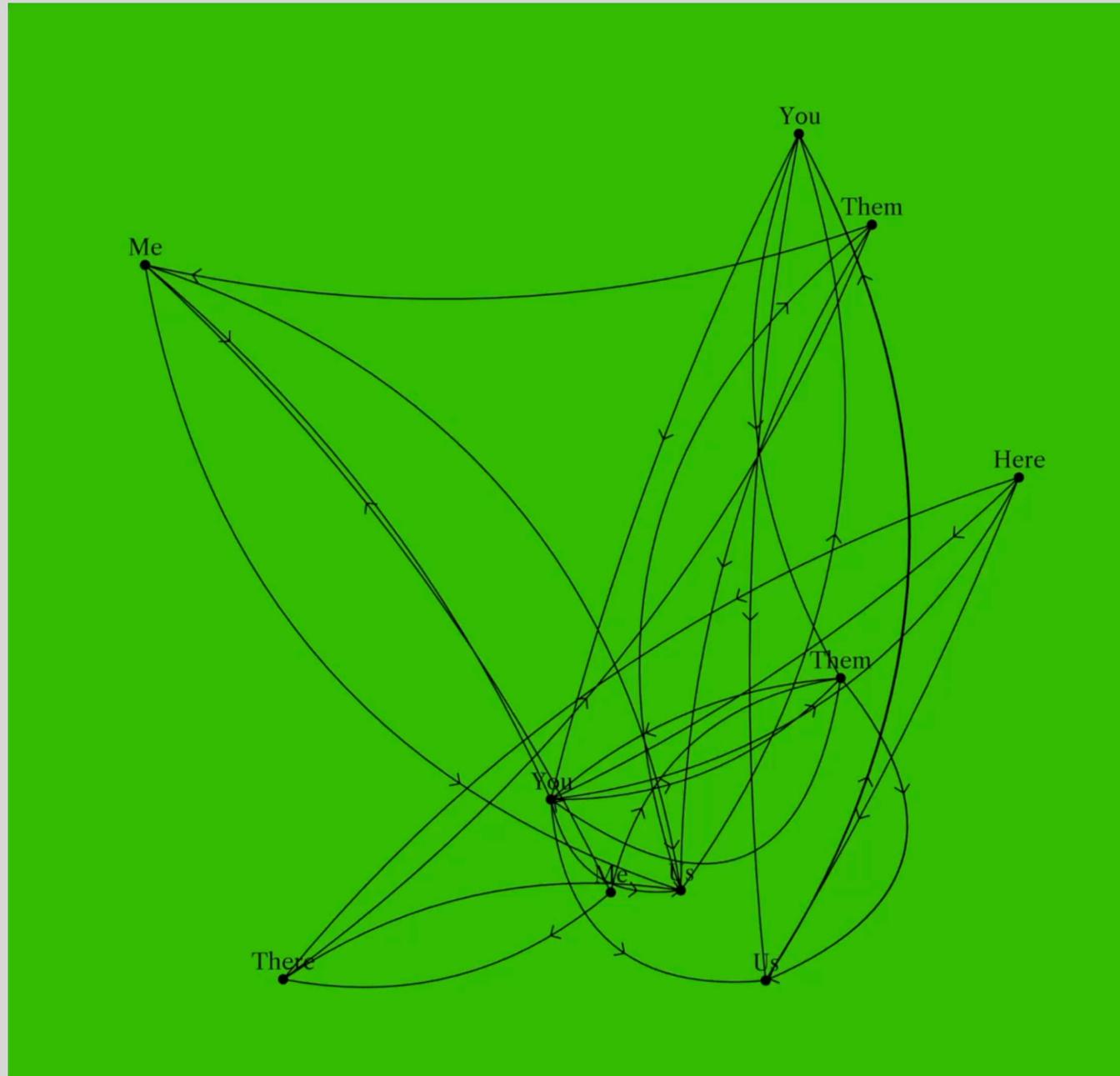
Minification is a process of removing unnecessary characters, making code smaller and lighter for faster load times and improved web performance.

The image shows a side-by-side comparison of HTML code in an Emacs editor window titled 'emacs@scarone'. The left pane shows the original, unminified HTML code, which is significantly longer and contains many unnecessary characters like spaces, tabs, and line breaks. The right pane shows the same HTML code after minification, where all these unnecessary characters have been removed, resulting in a much shorter and more compact code block. The minified code is color-coded, with tags in blue, attributes in green, and values in red.

```

!doctype html>
<html lang="en" itemscope itemtype='http://schema.org/CollectionPage'>
  <head>
    <meta name="generator" content="Hugo 0.46" />
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <meta property="description" content='Kids Games has family games that are fun to play. Play exciting games for the ages of 2 to 6 years, online and for free! A safe place to play the best free games! Games for boys and girls that can be played on any device (mobile, tablets and computers).' />
    <meta name="author" content="Marcelo Canina">
    <link rel="icon" href="https://kidsgames.world/favicon.ico">
    <link rel="canonical" href="https://kidsgames.world/">
    <title>Play Fun, Free Games for Boys and Girls Online | Free Games For Kids</title>
    <link href="https://kidsgames.world/css/style.css" rel="stylesheet" type="text/css">
    <link href="https://kidsgames.world/css/custom.css" rel="stylesheet" type="text/css">
    <meta property="og:title" content="Play Fun, Free Games for Boys and Girls Online" />
    <meta property="og:description" content="Kids Games has family games that are fun to play. Play exciting games for the ages of 2 to 6 years, online and for free! A safe place to play the best free games! Games for boys and girls that can be played on any device (mobile, tablets and computers)." />
    <meta property="og:type" content="website" />
    <meta property="og:url" content="https://kidsgames.world/" />
    <meta property="og:updated_time" content="2017-11-19T16:06:44-03:00"/>
!doctype html><html lang=en itemscope itemtype=http://schema.org/CollectionPage>
<head><meta name=generator content="Hugo 0.46"><meta charset=utf-8><meta name=viewport content="width=device-width,initial-scale=1,shrink-to-fit=no"><meta property=description content="Kids Games has family games that are fun to play. Play exciting games for the ages of 2 to 6 years, online and for free! A safe place to play the best free games! Games for boys and girls that can be played on any device (mobile, tablets and computers)."><meta name=author content="Marcelo Canina"><link rel=icon href=https://kidsgames.world/favicon.ico><link rel=canonical href=https://kidsgames.world/><title>Play Fun, Free Games for Boys and Girls Online | Free Games For Kids</title><link href=https://kidsgames.world/css/style.css rel=stylesheet><link href=https://kidsgames.world/css/custom.css rel=stylesheet><meta property=og:title content="Play Fun, Free Games for Boys and Girls Online"><meta property=og:description content="Kids Games has family games that are fun to play. Play exciting games for the ages of 2 to 6 years, online and for free! A safe place to play the best free games! Games for boys and girls that can be played on any device (mobile, tablets and computers)."><meta property=og:type content=website><meta property=og:url content=https://kidsgames.world/><meta property=og:updated_time content=2017-11-19T16:06:44-03:00><meta property=og:site_name content="Free Games For Kids"><meta itemprop=name content="Play Fun, Free Games for Boys and Girls Online"><meta itemprop=description content="Kids Games has family games that are fun to play. Play exciting games for the ages of 2 to 6 years, online and for free! A safe place to play the best free games! Games for boys and girls that can be played on any device (mobile, tablets and computers)."><meta name=twitter:card content=summary><meta name=twitter:title content="Play Fun, Free Games for Boys and Girls Online"><meta name=twitter:description content="Kids Games has family games that are fun to play. Play exciting games for the ages of 2 to 6 years, online and for free! A safe place to play the best free games! Games for boys and girls that can be played on any device (mobile, tablets and computers)."><meta name=twitter:site content=@allgamesforkids><meta name=google-site-verification content=BJqwEJl00m40MZS905kMNmQb7wjxzEd54eqqNFRK07U><meta name=msvalidate.01 content=4253BB779294406F41BD202E3E24FFE55><meta name=yandex-verification content=94dfdda772c13d7b><script async src=//pagead2.googlesyndication.com/pagead/js/adsbygoogle.js></script><script>(adsbygoogle=window.adsbygoogle||[]).push({google_ad_client:"ca-pub-1234567890",enable_page_level_ads:true});</script><link rel=alternate type=application/rss+xml href=https://kidsgames.world/index.xml title="Free Games For Kids"><script>var doNotTrack=false;if(!doNotTrack){window.ga=window.ga||function(){(ga.q=ga.q||[]).push(arguments)};ga.l+=new Date;ga('create','UA-31899481-6','auto');ga('send','pageview');}</script><script async src=https://www.google-analytics.com/analytics.js></script></head><body><nav class="navbar navbar-expand-lg navbar-light bg-light"><a class="navbar-brand href=https://kidsgames.world/>@ Kids Ga
  
```

My Prototype



The Chosen Project

