Unit 1

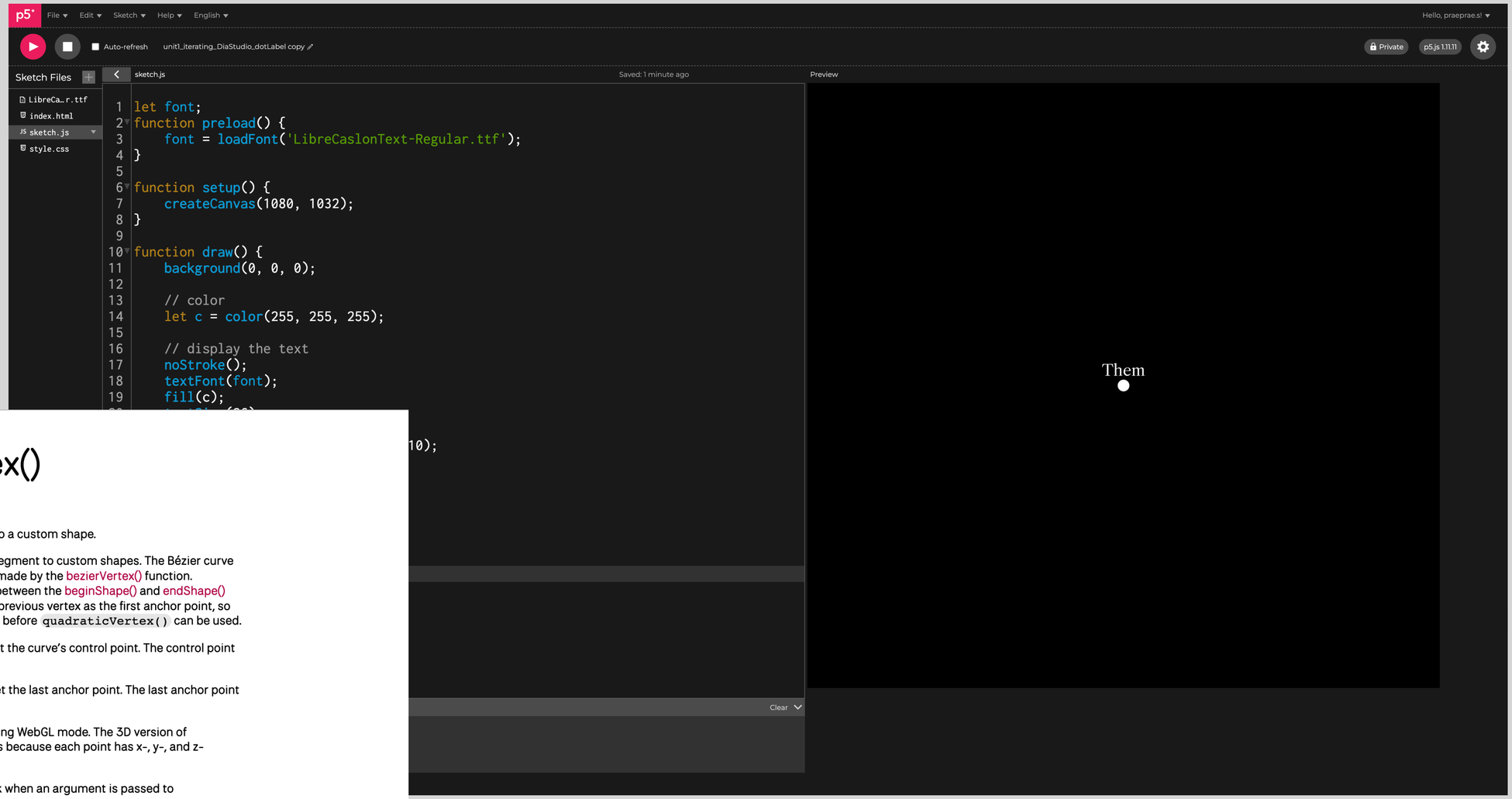
Methods of Iterating week 2

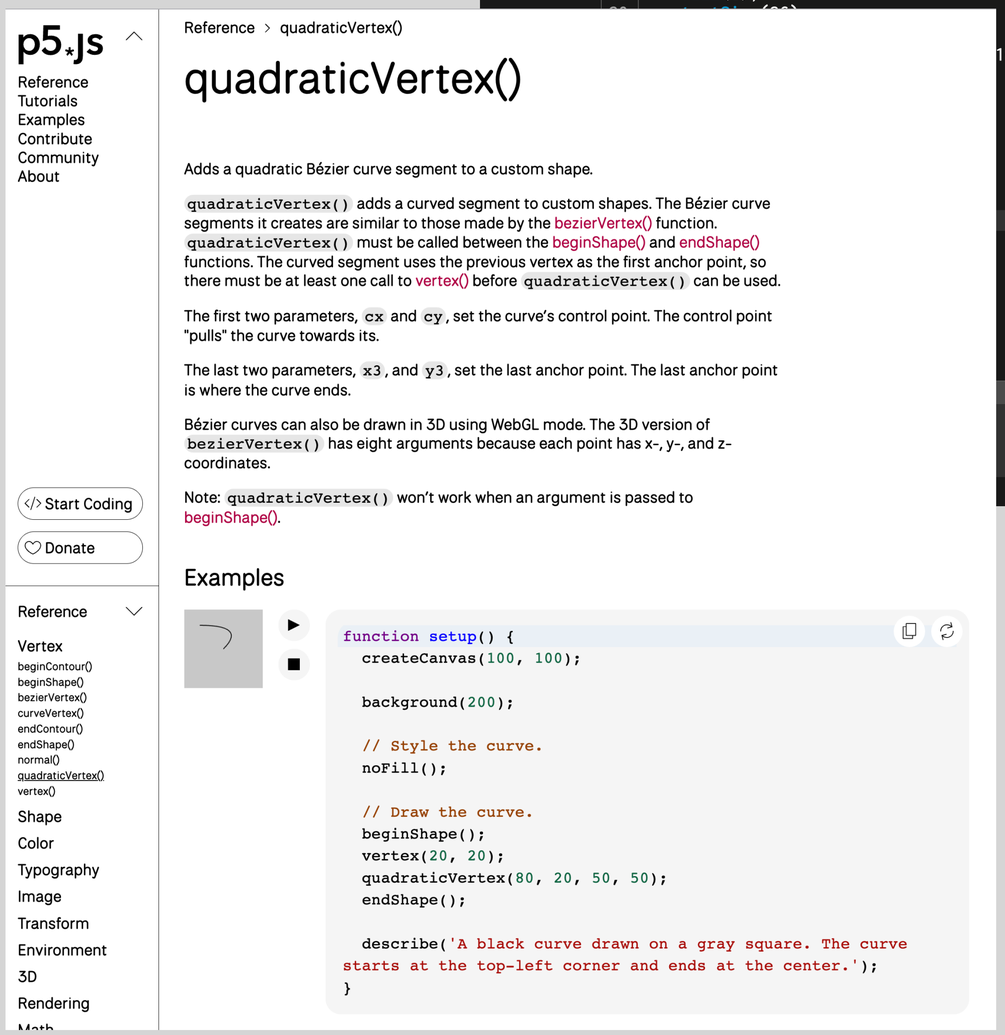
MA GCD | Y1 Prae Sunsermsook

23 January 2026

Selected Tool

p5.js





p5.js
Reference
Tutorials
Examples
Contribute
Community
About

Start Coding
Donate

Reference
Vertex
beginContour()
beginShape()
bezierVertex()
curveVertex()
endContour()
endShape()
normal()
quadraticVertex()
vertex()
Shape
Color
Typography
Image
Transform
Environment
3D
Rendering

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

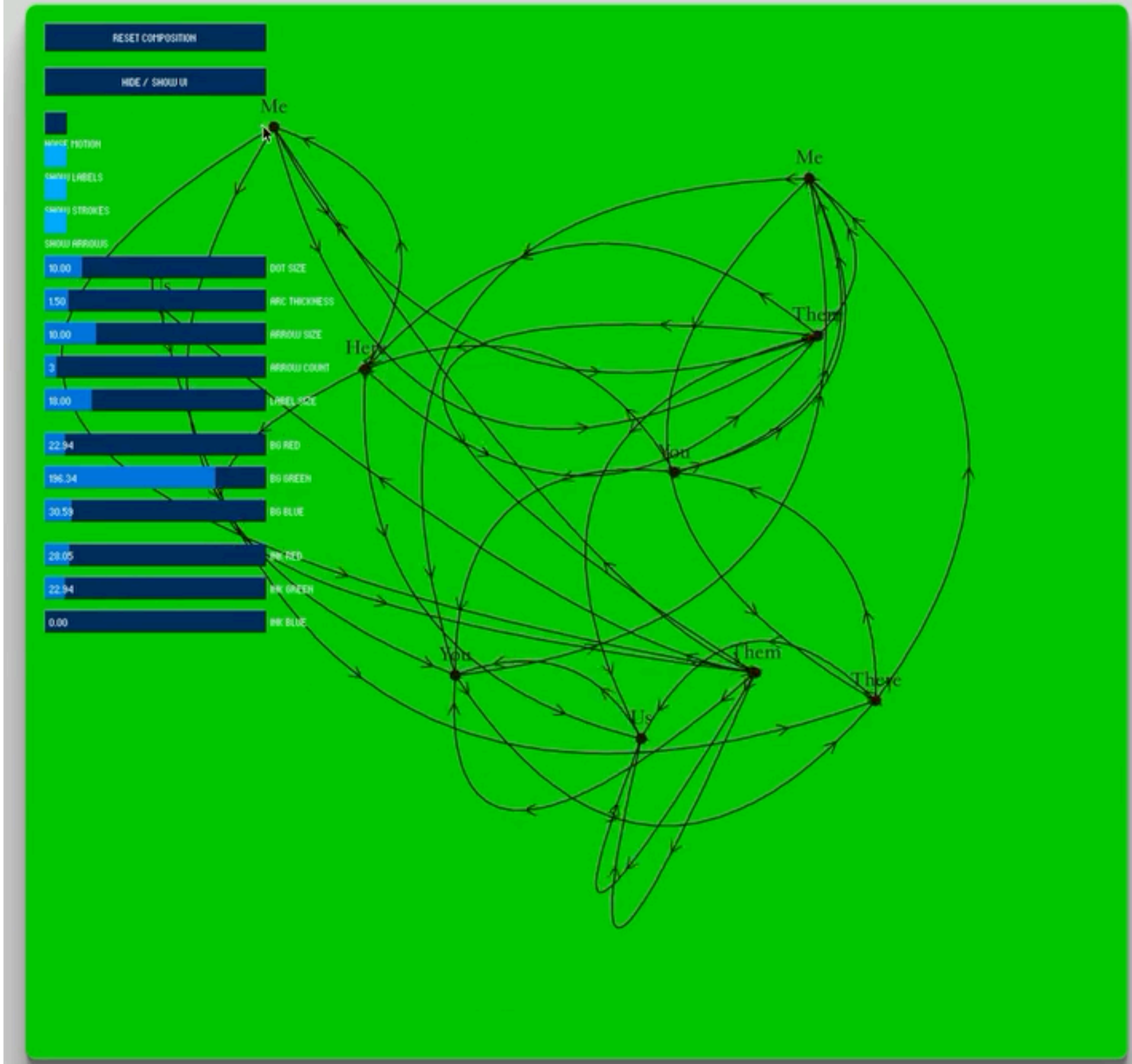
```
function setup() {
  createCanvas(100, 100);

  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

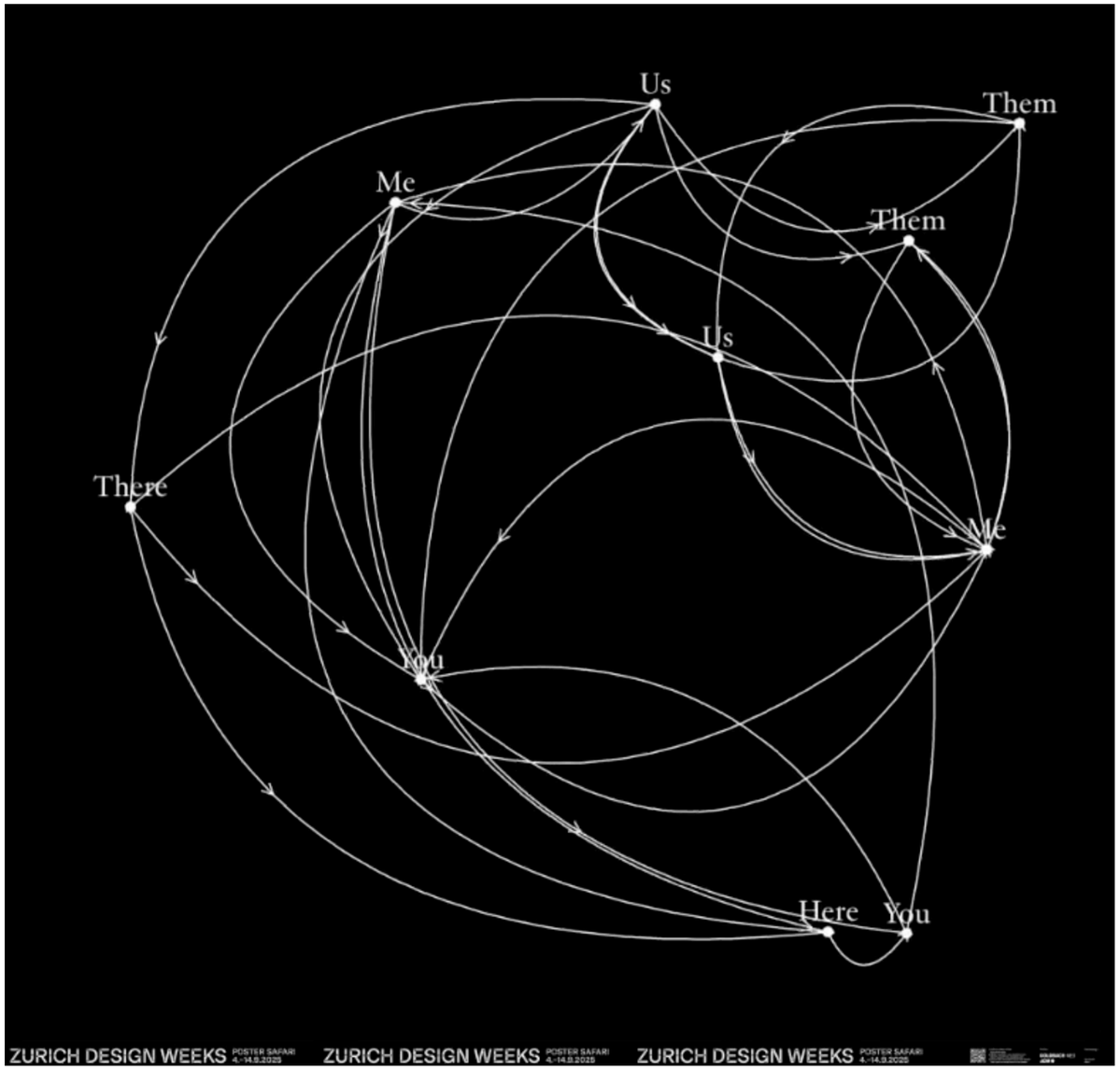
A generative design application for a poster featuring the theme ‘TEAM UP’ at Zurich Design Week by DIA Studio.

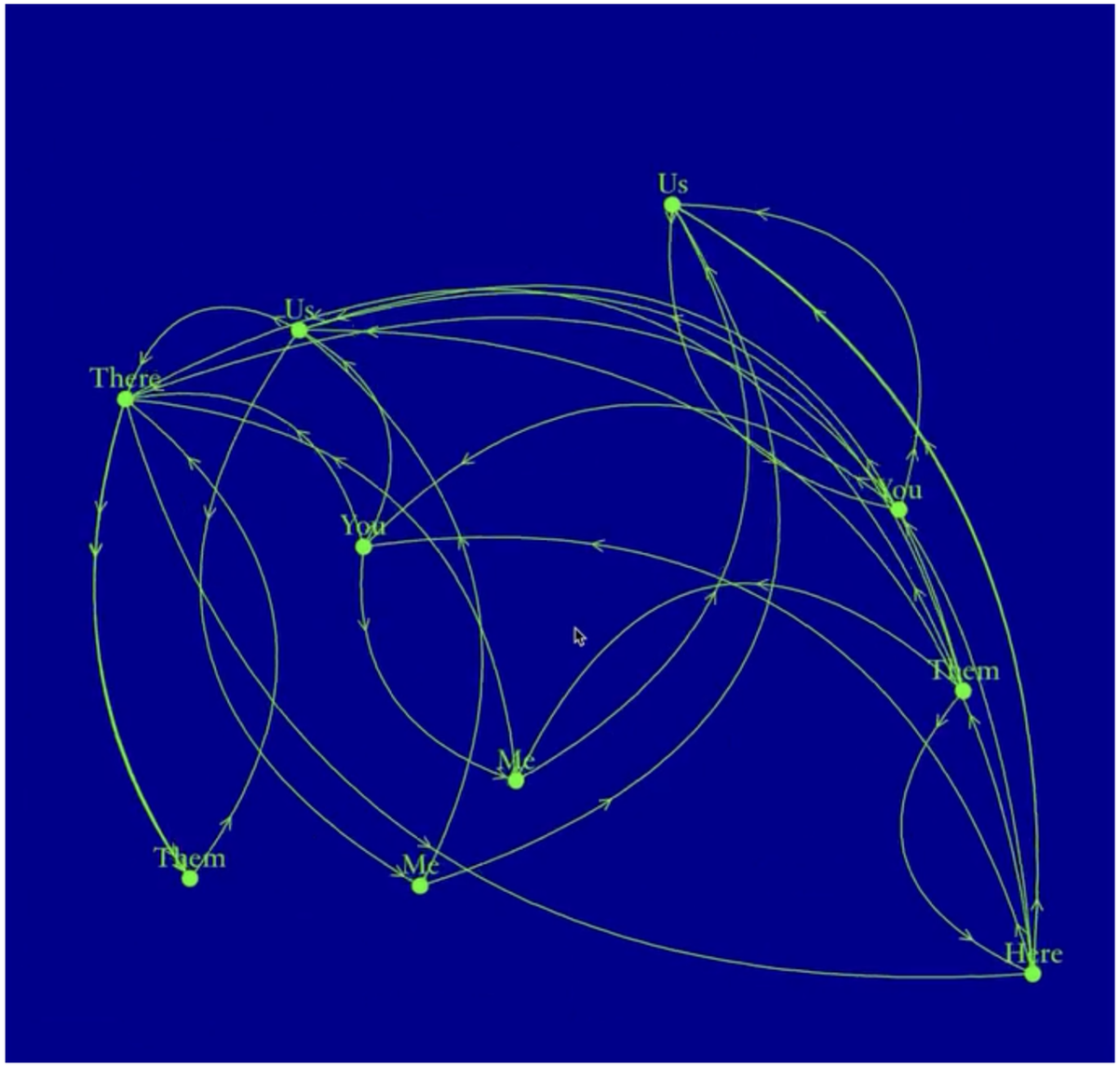


describe (‘replicatingProcess’);

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

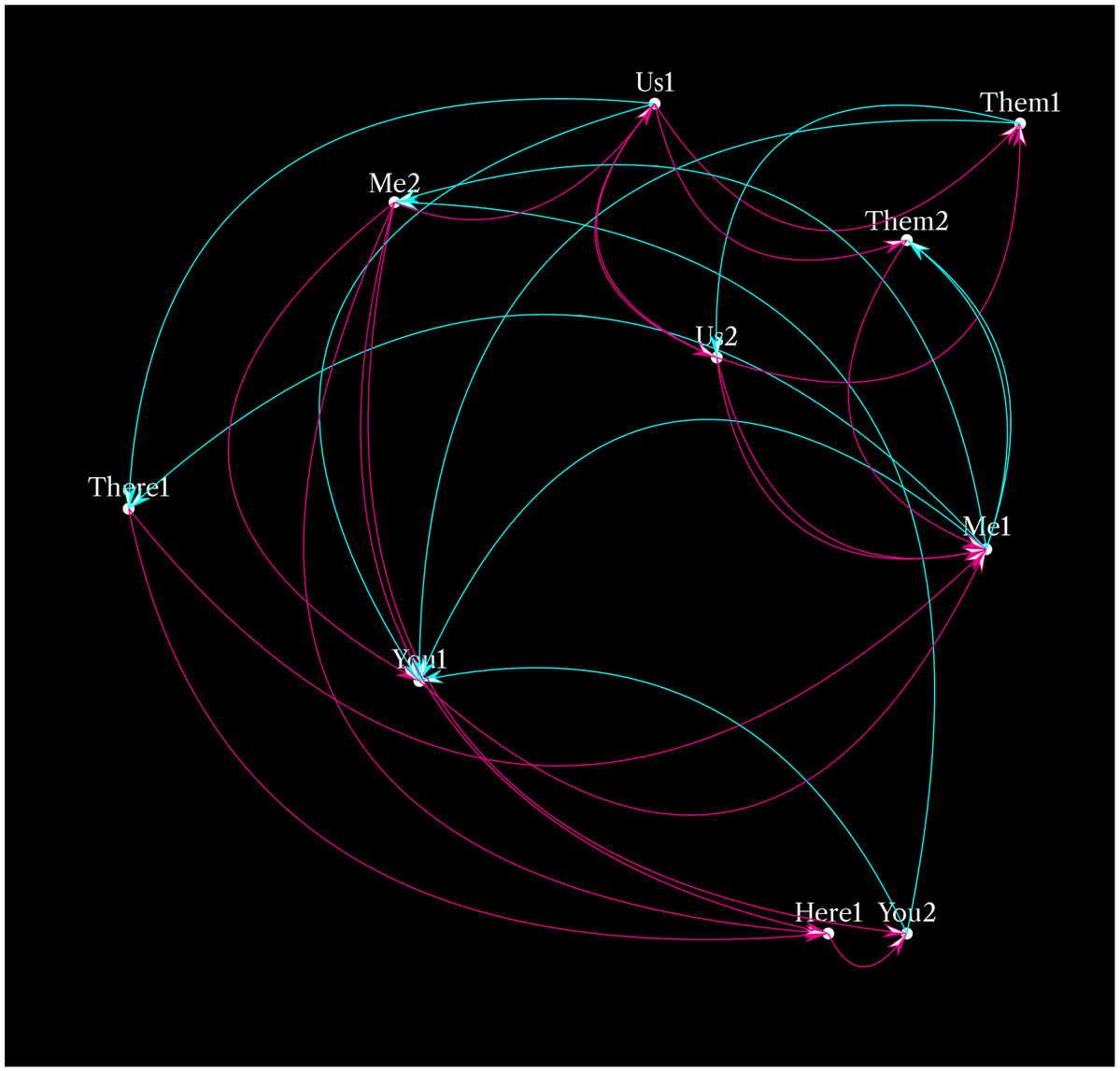
Screenshots from the chosen project



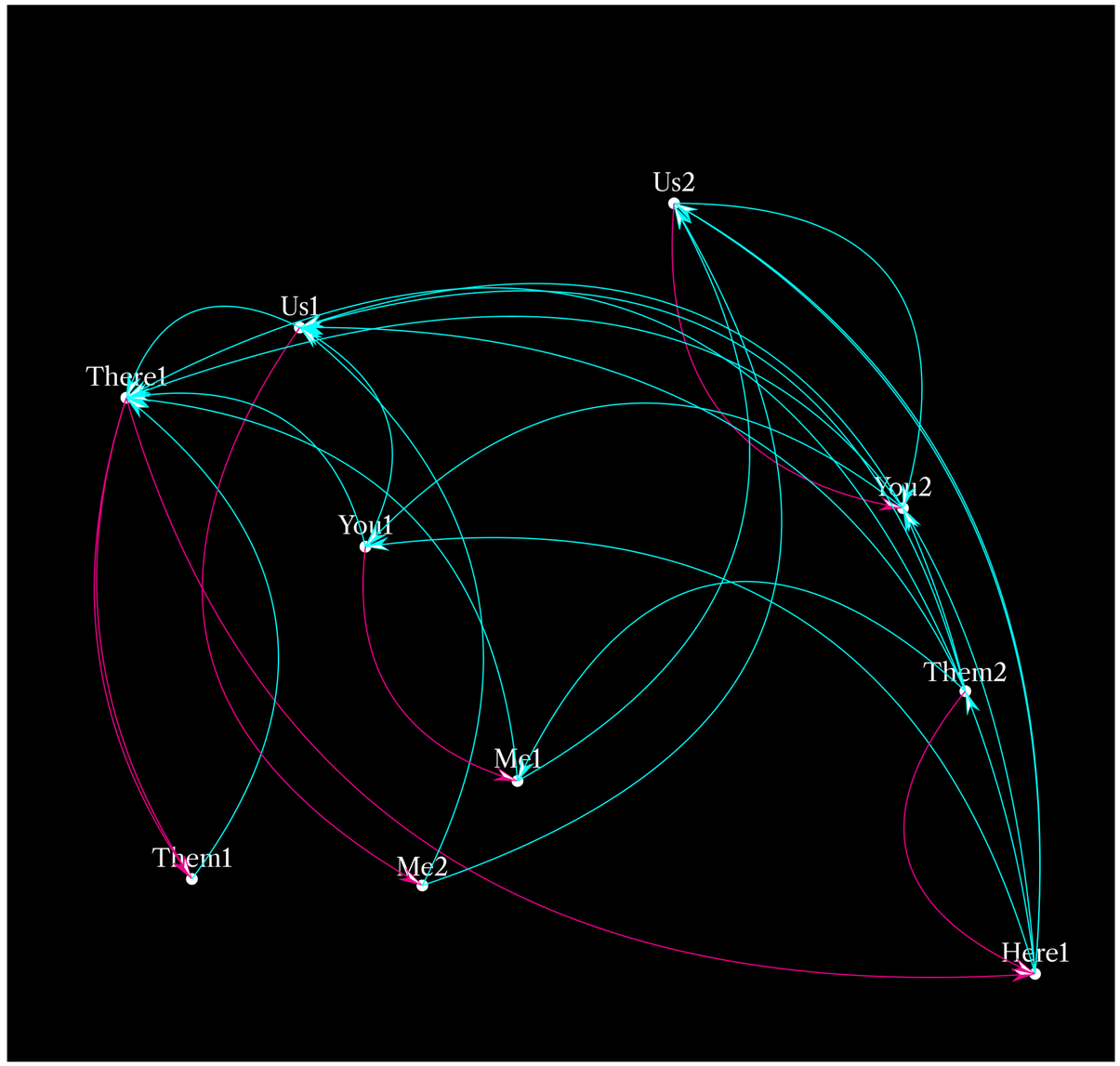


describe (‘replicatingProcess’);
// Create drafts in Illustrator to analyse the logic behind visual outputs

Draft 1



Draft 2



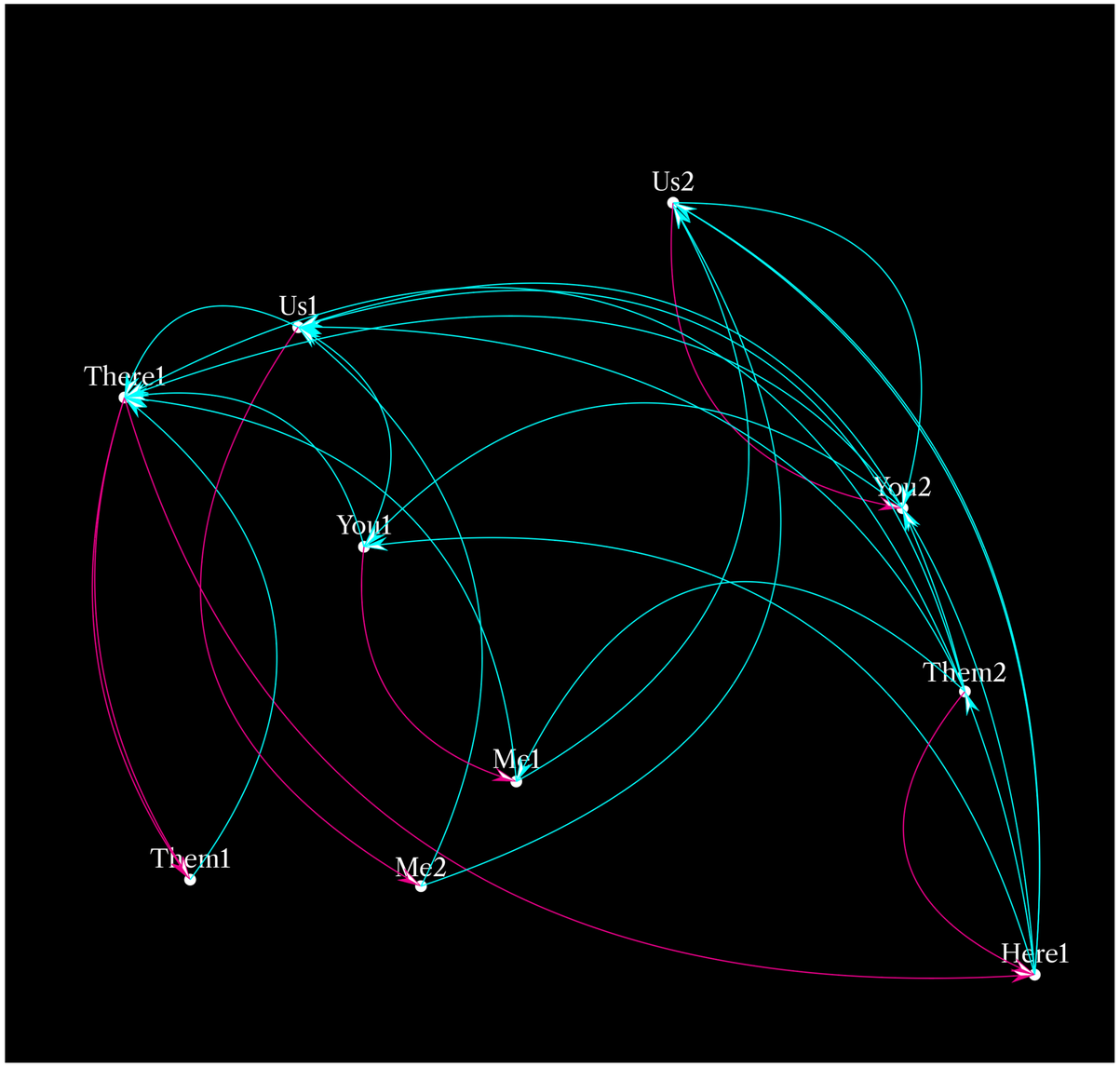
describe (‘replicatingProcess’);
// Analyse data, logic, and rules

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

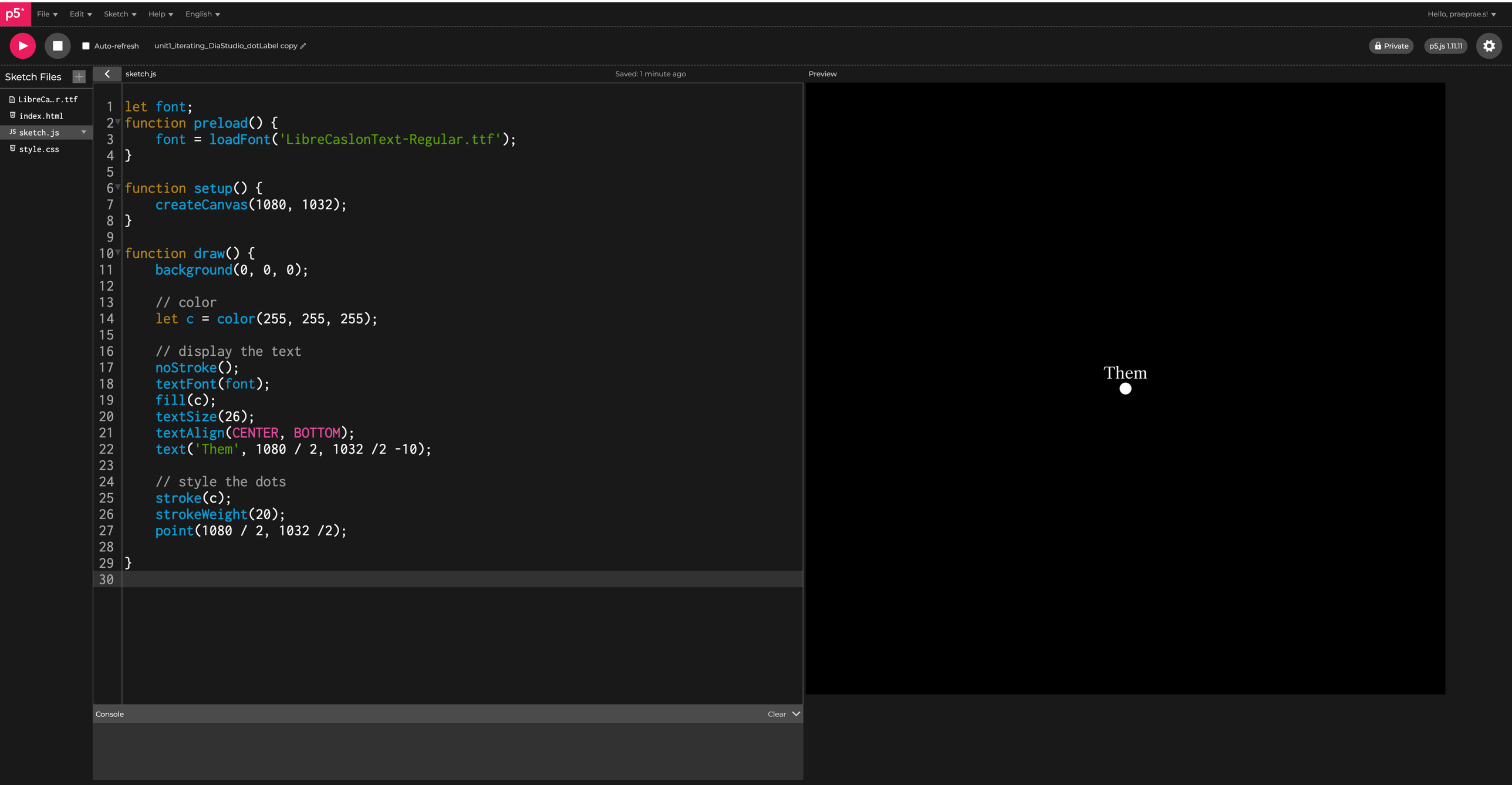
	A	B
1	**From**	**to**
2	Us1	There1
3	Us1	You1
4	Us1	Them1
5	Us1	Them1
6	Us1	Us2
7	Us1	Us2
8	Us2	Me1
9	Us2	Me1
10	Us2	Them1
11	Me1	Them2
12	Me1	Them2
13	Me1	You1
14	Me1	There1
15	Me1	Me2
16	Me2	You1
17	Me2	Us1
18	Me2	Here1
19	Me2	Here1
20	Me2	You2
21	You1	Me1
22	You2	You1
23	You2	Me2
24	Them1	Us2
25	Them1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

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

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

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.

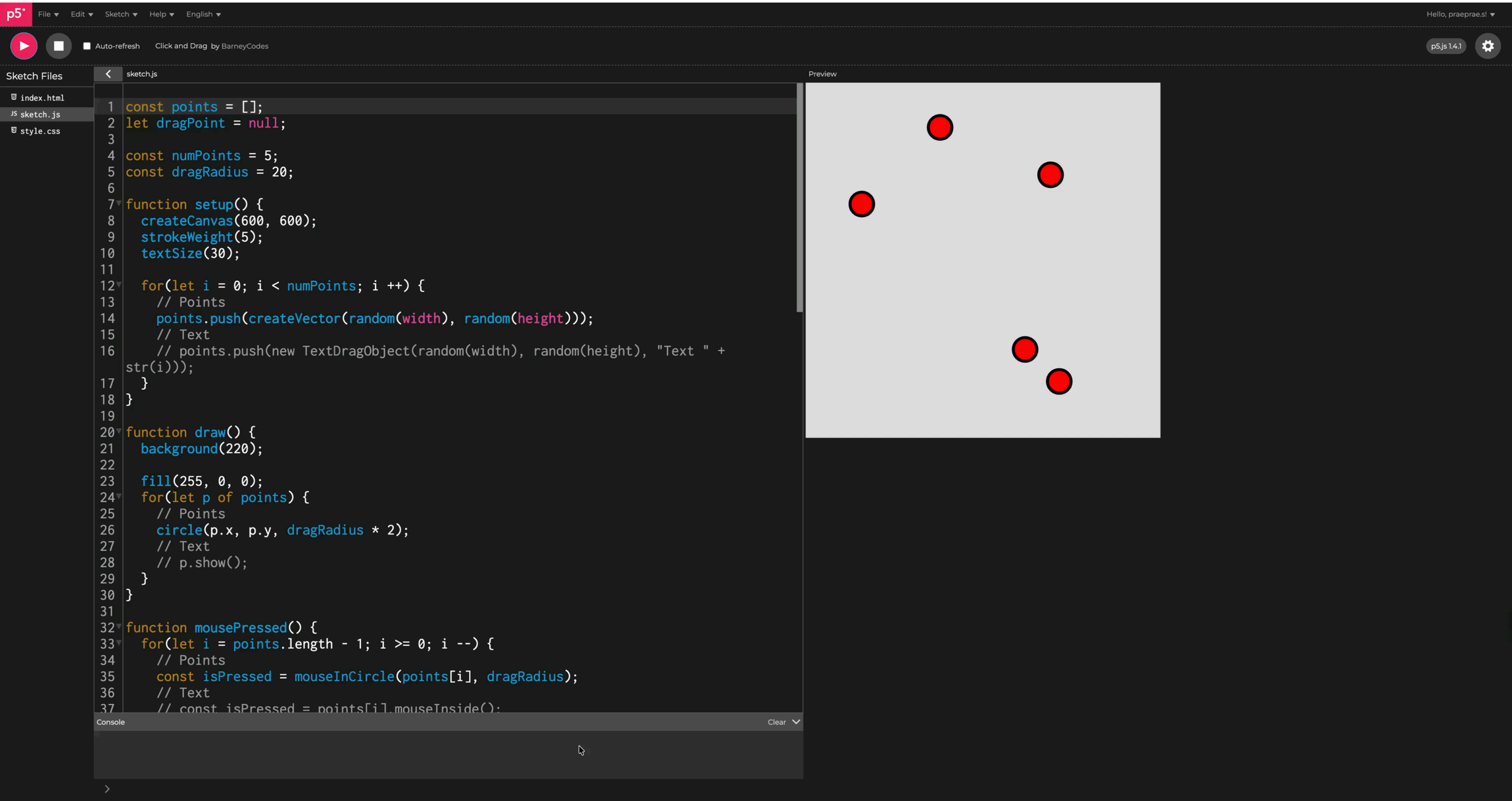
Main Sketch 1



describe (‘replicatingProcess’);
// Create randomised labels > Finding an example

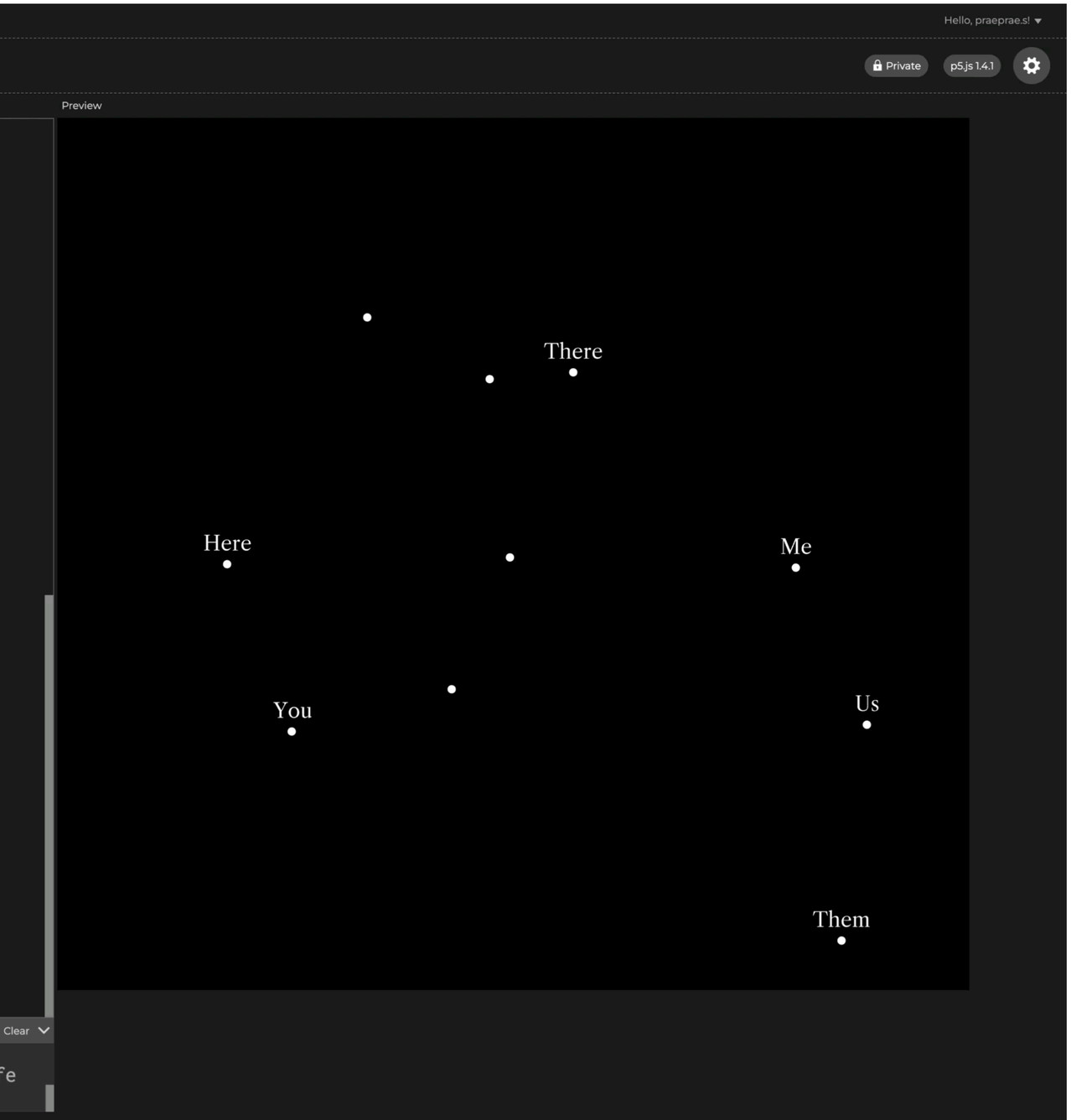
However, I began to find it challenging to formulate code as the rules became more complex. Therefore, most of my learning revolves around adapting, reusing, and patching existing resources instead of creating everything from the ground up.

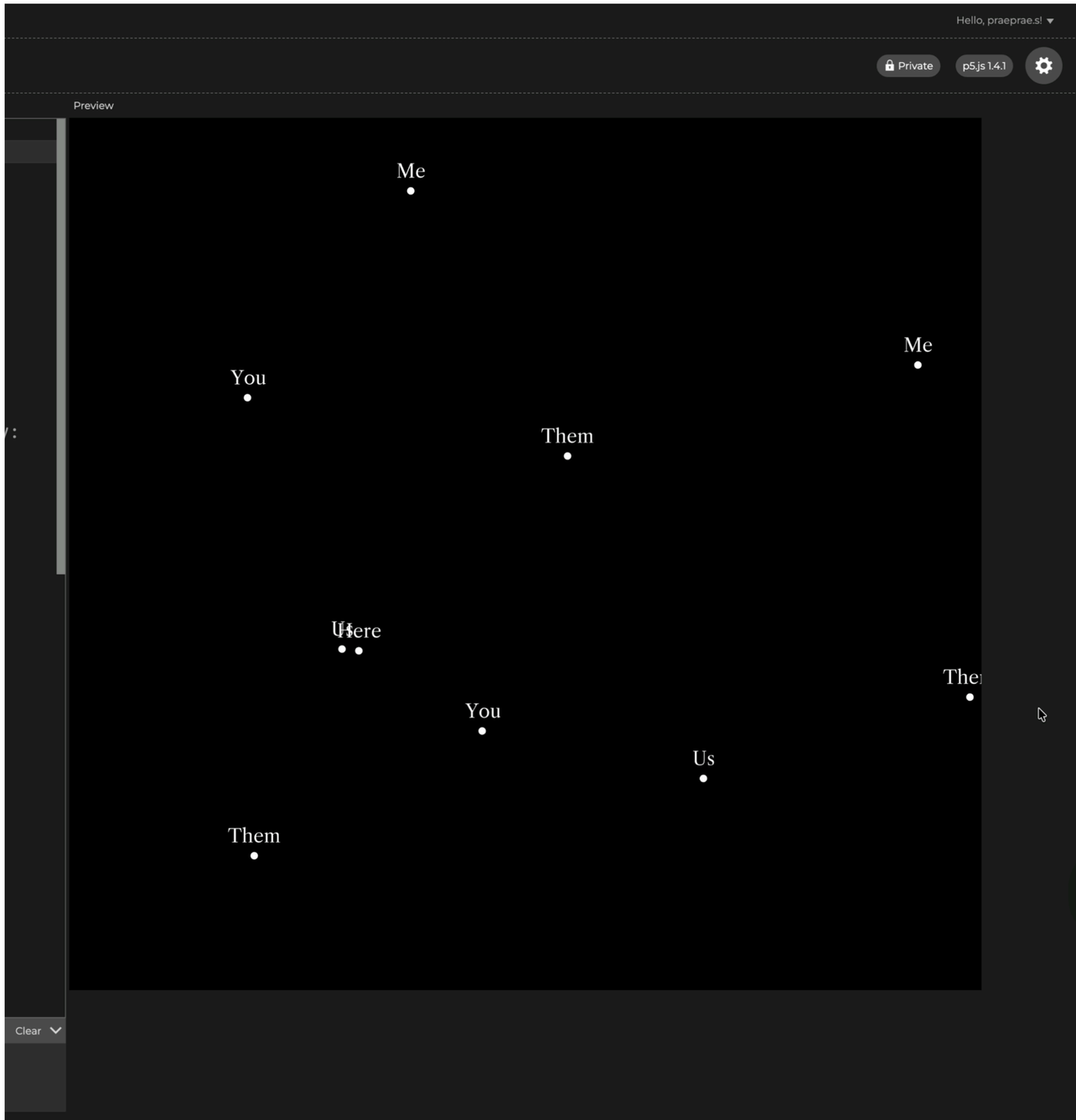
Example Project by BarneyCodes



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

Main Sketch 1

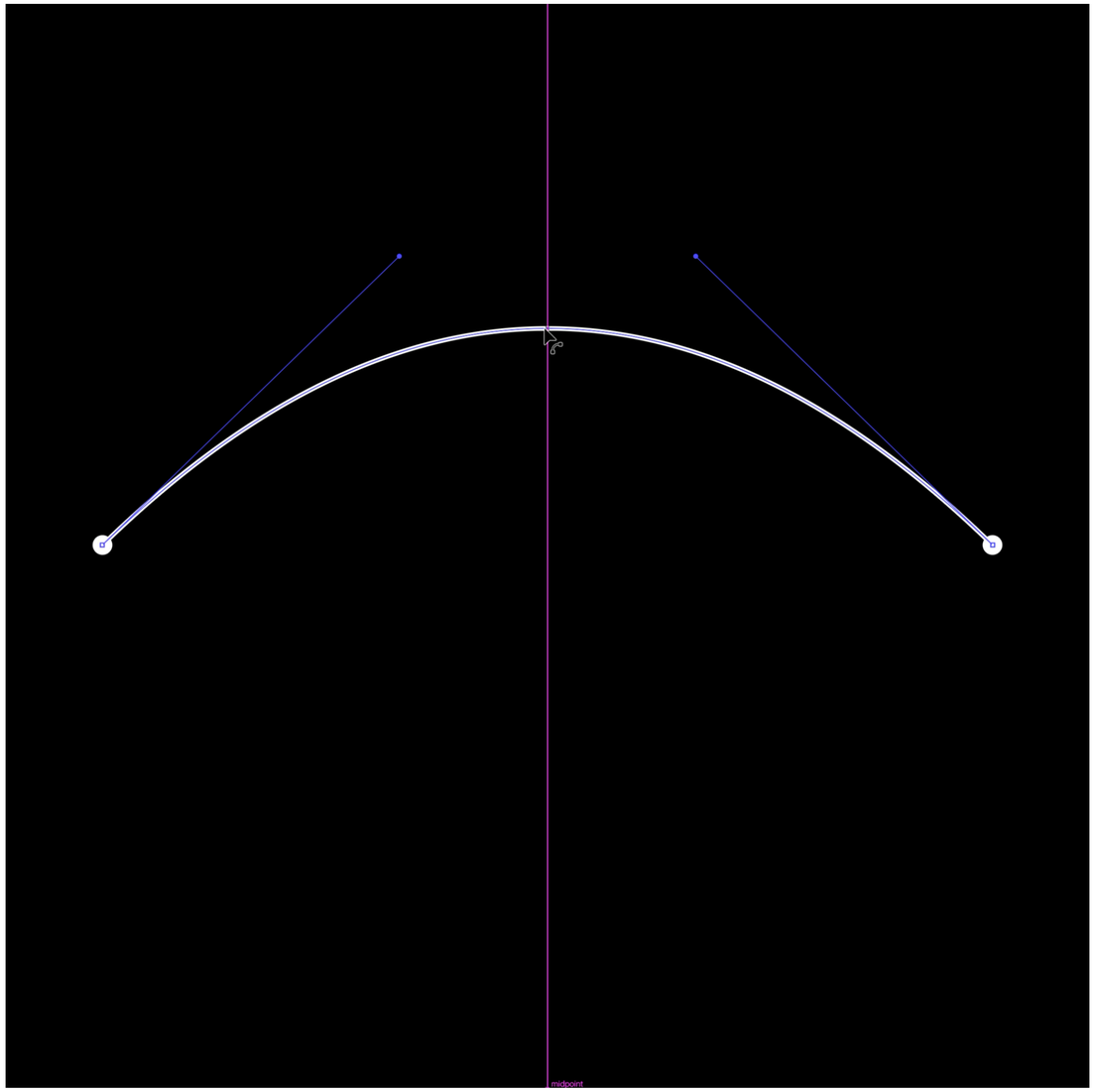




describe (‘replicatingProcess’);
// How a curve is created in coding

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



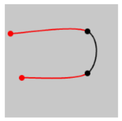
describe (‘replicatingProcess’);
// How a curve is created in coding

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

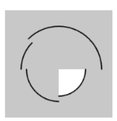
curve()

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()

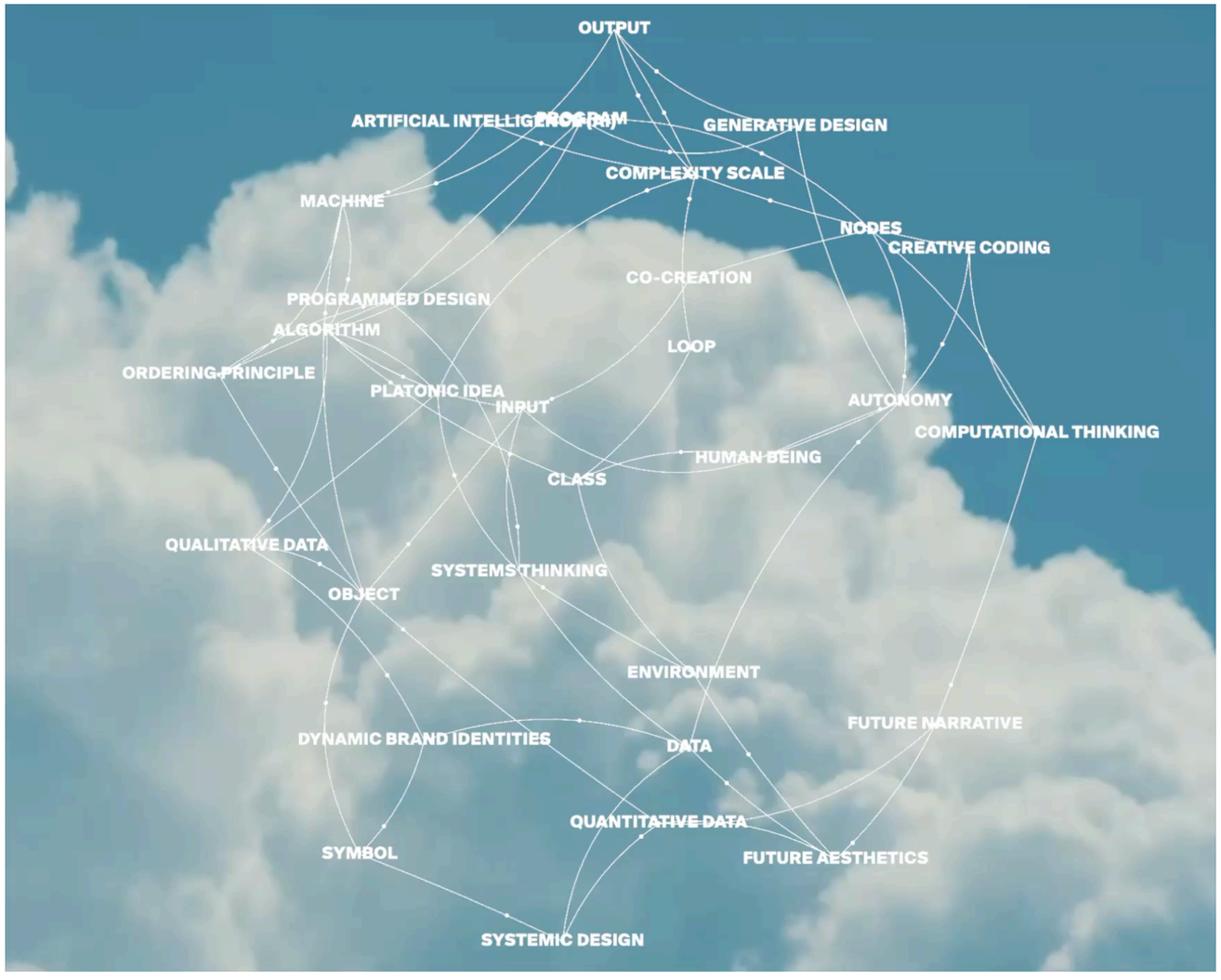
Examples

```
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();
```

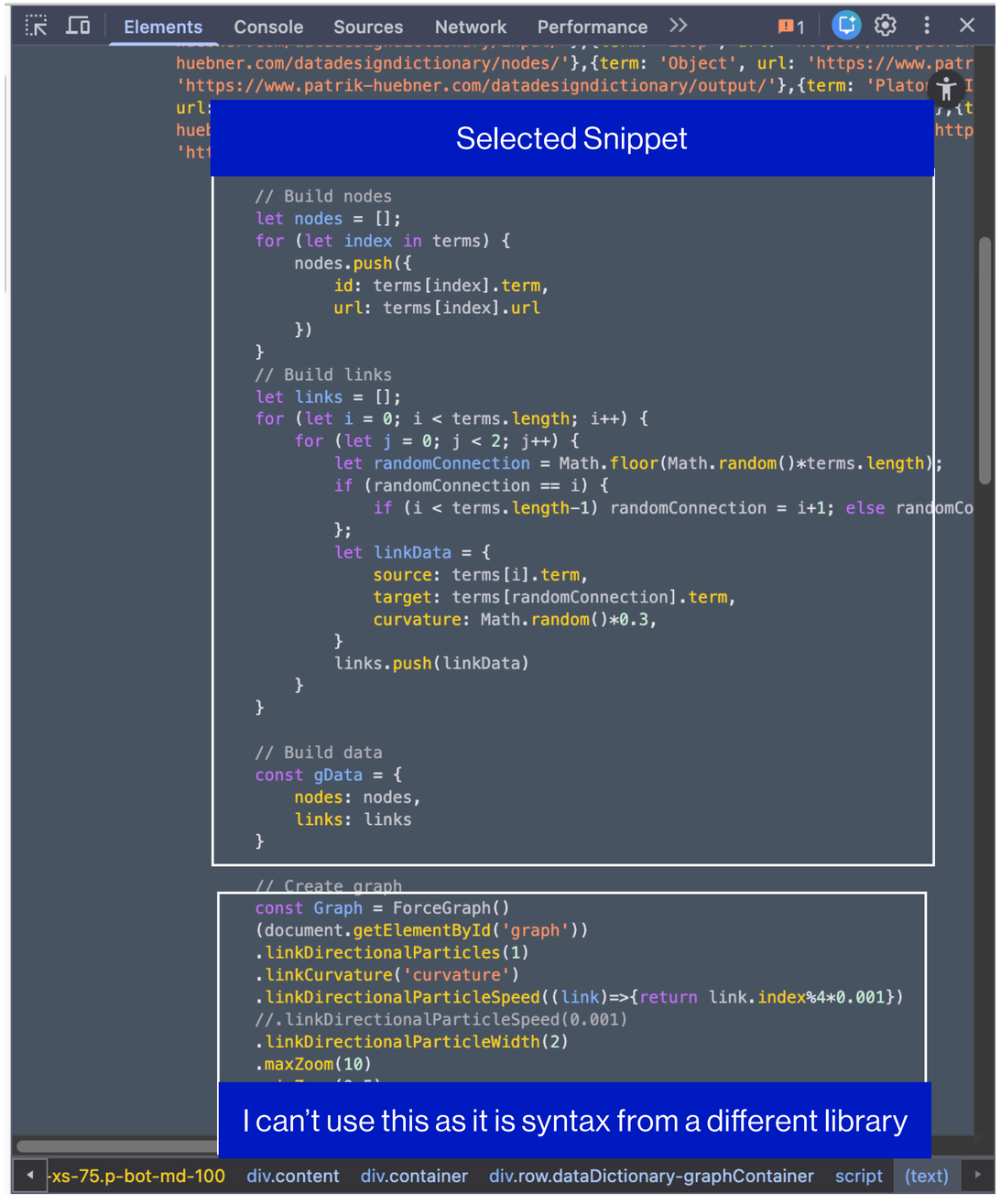
describe (‘replicatingProcess’);
// Find a case study on how dots are connected

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/

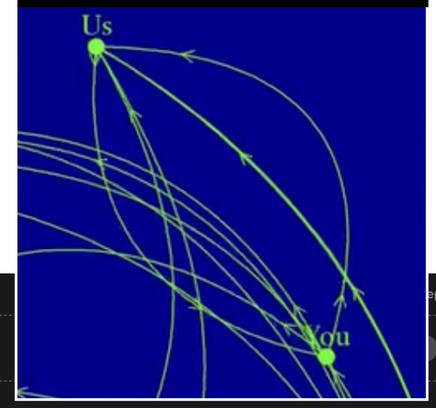


describe (‘replicatingProcess’);
// Create curves by bezierVertex()
// Set up rules for creating connections

Main Sketch 2

The outcome was not align with my desired visual

Desired Visual



p5* File Edit Sketch Help English

Auto-refresh unit1_iterating_DiaStudio_connecting_02

Sketch Files

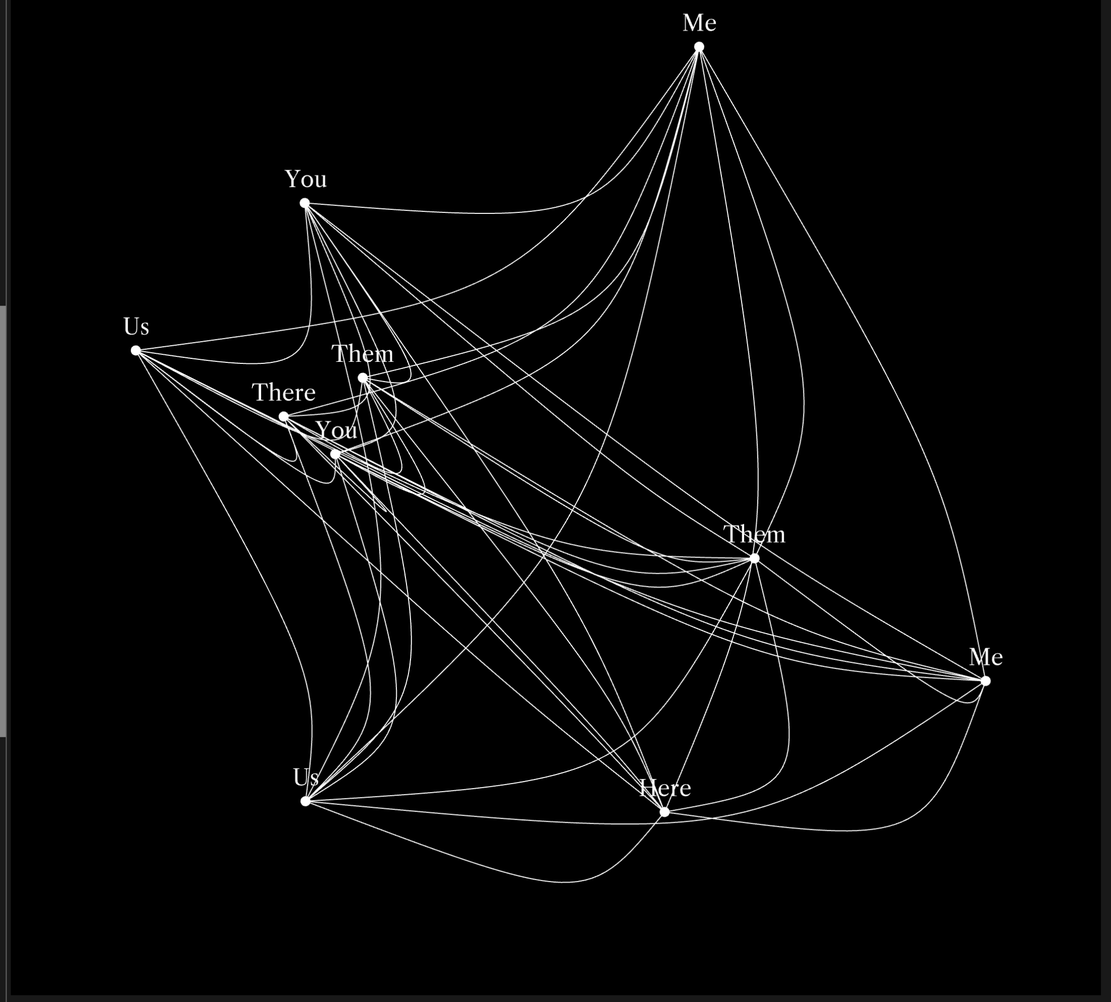
- assets
  - Libr… r.ttf
- index.html
- sketch.js
- style.css

sketch.js Saved: 1 day ago

```
  }
}

function draw() {
  background(0, 0, 0);
  let c = color(255, 255, 255);

  // draw curves and connect dots
  noFill();
  stroke(c);
  strokeWeight(1);

  for(let i = 0; i < points.length; i++) {
    for(let j = i + 1; j < points.length; j++) {
      let p1 = points[i].pos;
      let p2 = points[j].pos;

      // control point is halfway between, but offset up/down
      let cx = (p1.x + p2.x) / 2 + 100;
      let cy = (p1.y + p2.y) / 2 + 100; // -50 makes it curve upward

      // draw bezier curve
      bezier(p1.x, p1.y, cx, cy, cx, cy, p2.x, p2.y);

    }
  }

  // draw dots and text
  for(let pt of points) {
    fill(c);
    noStroke();
    circle(pt.pos.x, pt.pos.y, dragRadius * 2);

    fill(c);
    text(pt.label, pt.pos.x, pt.pos.y - dragRadius - 5);
  }
}
```

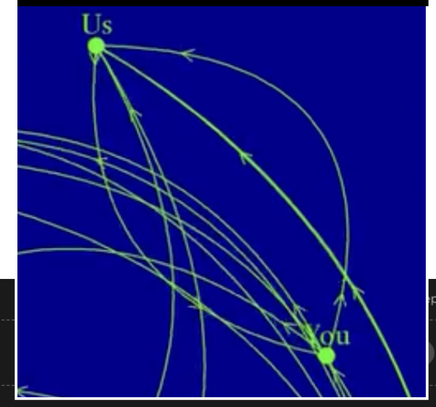
Console Clear

Preview



describe (‘replicatingProcess’);
// Create curves by quadraticVertex()

Desired Visual



Snippet Sketch: Curve

p5* File Edit Sketch Help English

Auto-refresh unit1_iterating_DiaStudio_connecting_01(learn how to draw curve))

Sketch Files

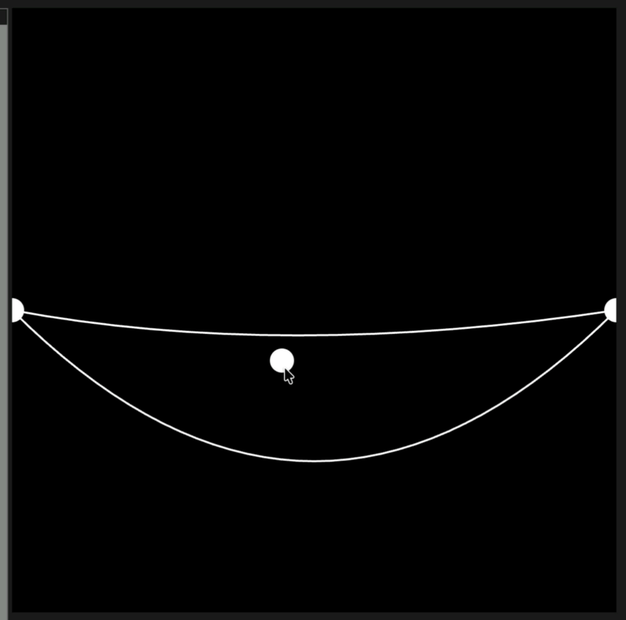
- assets
  - Libr… r.ttf
- index.html
- sketch.js
- style.css

sketch.js Saved: 1 day ago

```
// Bezier (bezierVertex)
// The Coding Train / Daniel Shiffman
// https://thecodingtrain.com/CodingChallenges/163-bezier.html
// https://youtu.be/zUgsYaUQNBk

// Basic: https://editor.p5js.org/codingtrain/sketches/Z53a719cQ
// Editor by Simon Tiger: https://editor.p5js.org/codingtrain/sketches/_R7RgtGfA
// bezierVertex: https://editor.p5js.org/codingtrain/sketches/O3_cLiOaw
// Time Table Cardioid with Bezier:
https://editor.p5js.org/codingtrain/sketches/kZ8dpklQg
// Quadratic: https://editor.p5js.org/codingtrain/sketches/fJIMDmHcE
// Cubic: https://editor.p5js.org/codingtrain/sketches/S1Pt8lol1
// Bezier with Formula: https://editor.p5js.org/codingtrain/sketches/0XOLNHbvC

function setup() {
  createCanvas(600, 600);
}

function draw() {
  background(0);

  // draw dot
  stroke(255);
  strokeWeight(24);
  point(0, 300); // at the starting
  point(mouseX, mouseY);
  point(600, 300); //at the ending

  strokeWeight(2);
  noFill();
  beginShape();
  vertex(0, 300); //x,y position at the starting
  quadraticVertex(mouseX, mouseY, 600, 300); // last two values are x,y at the end
  quadraticVertex(300, 600, 0, 300); // first two values are
  endShape();
}
```

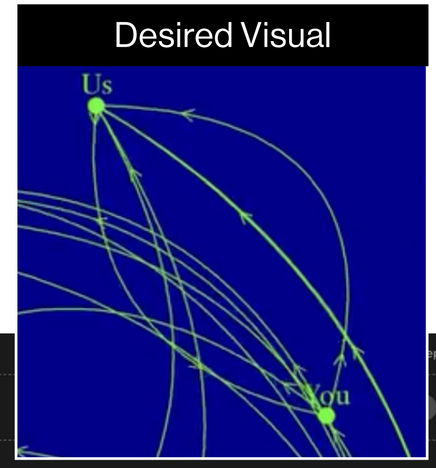
Console Clear

Preview



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.

p5* File Edit Sketch Help English

Auto-refresh unit1_iterating_DiaStudio_connecting_03

Sketch Files

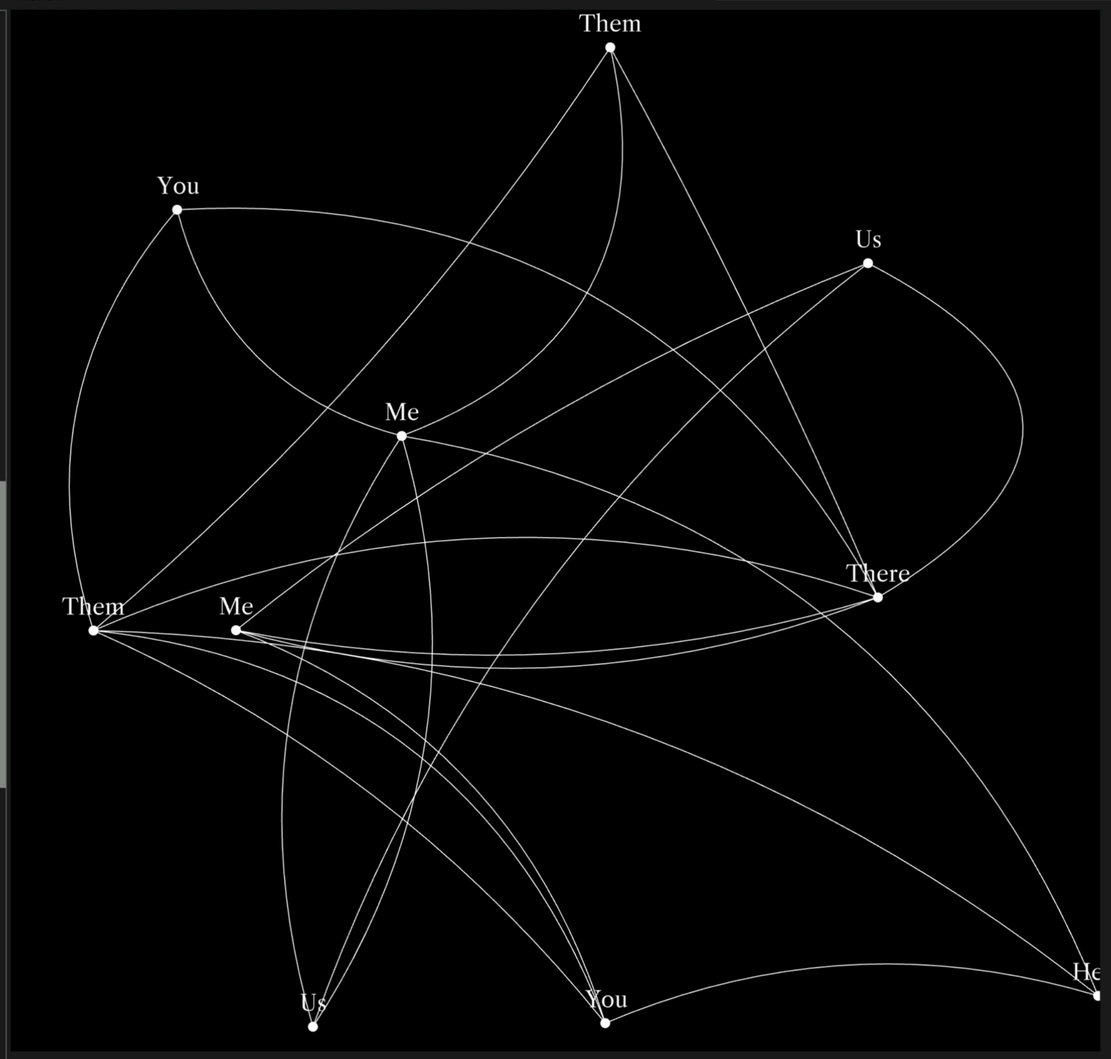
- assets
  - Libr… r.ttf
- index.html
- sketch.js
- style.css

sketch.js Saved: 1 day ago

```
58
59  // craw 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 divine 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 qaudratic 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
```

Console Clear

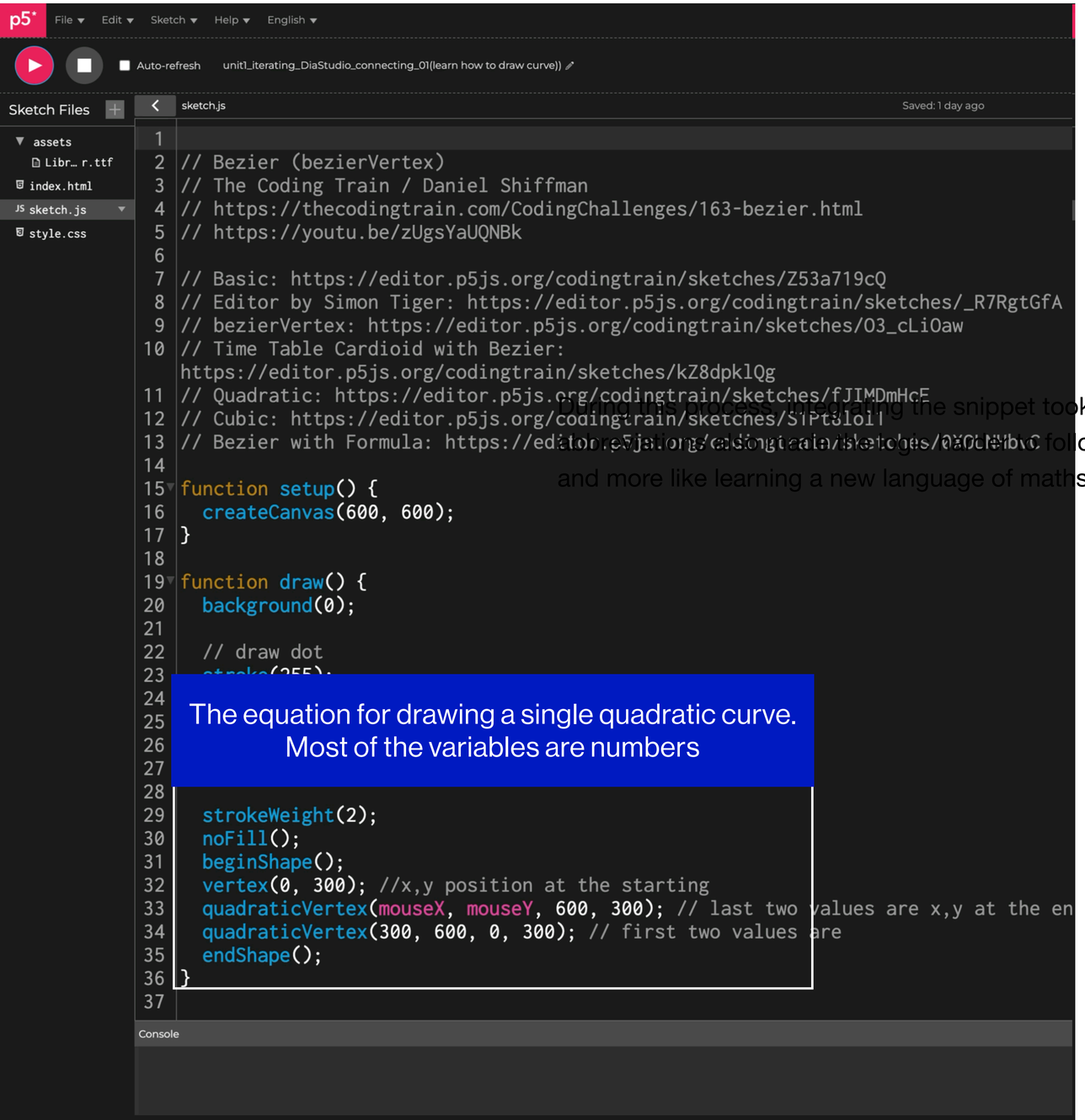
Preview



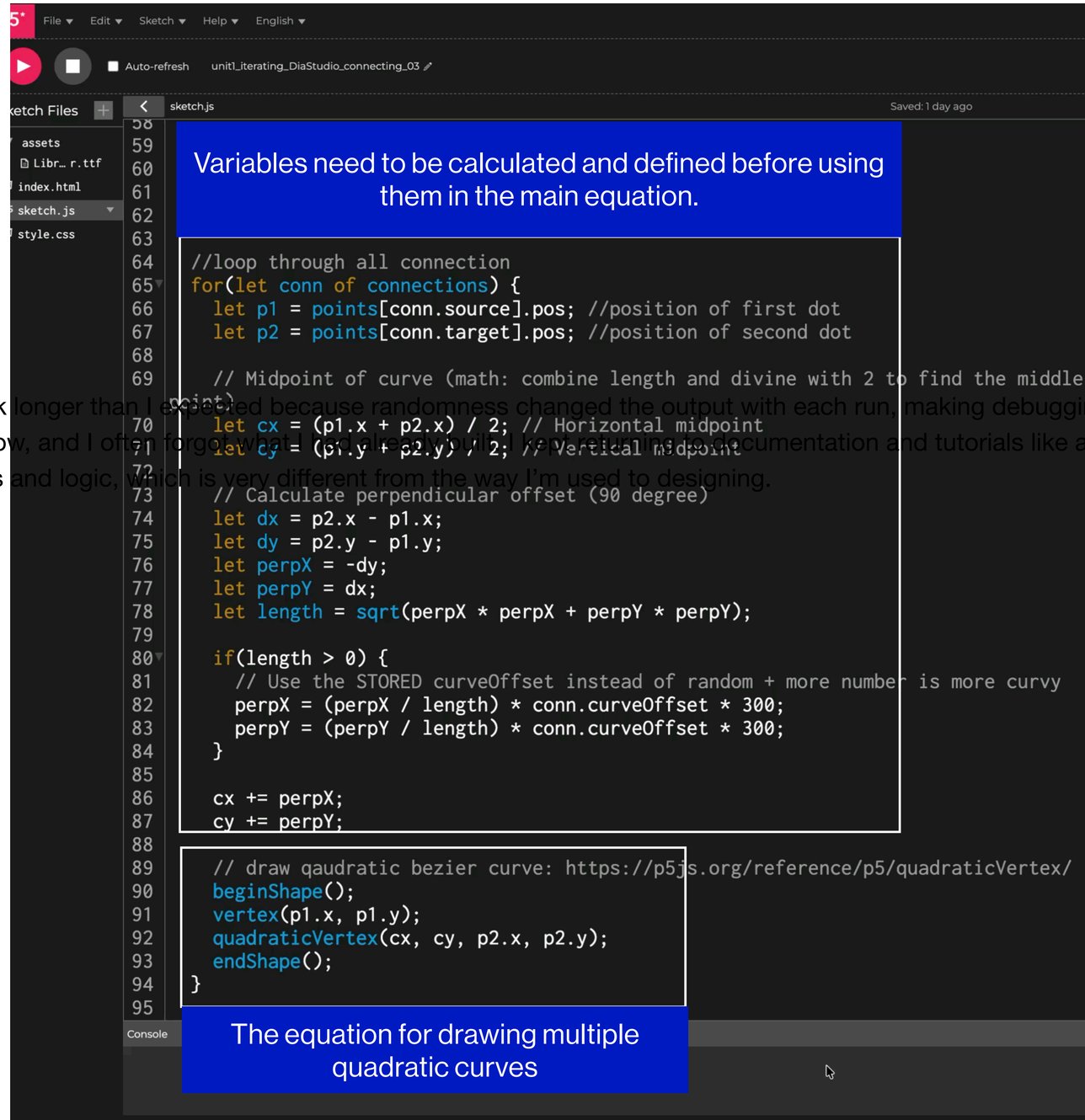
describe (‘replicatingProcess’);
// Language challenges during integreting snippet into the main sketch

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



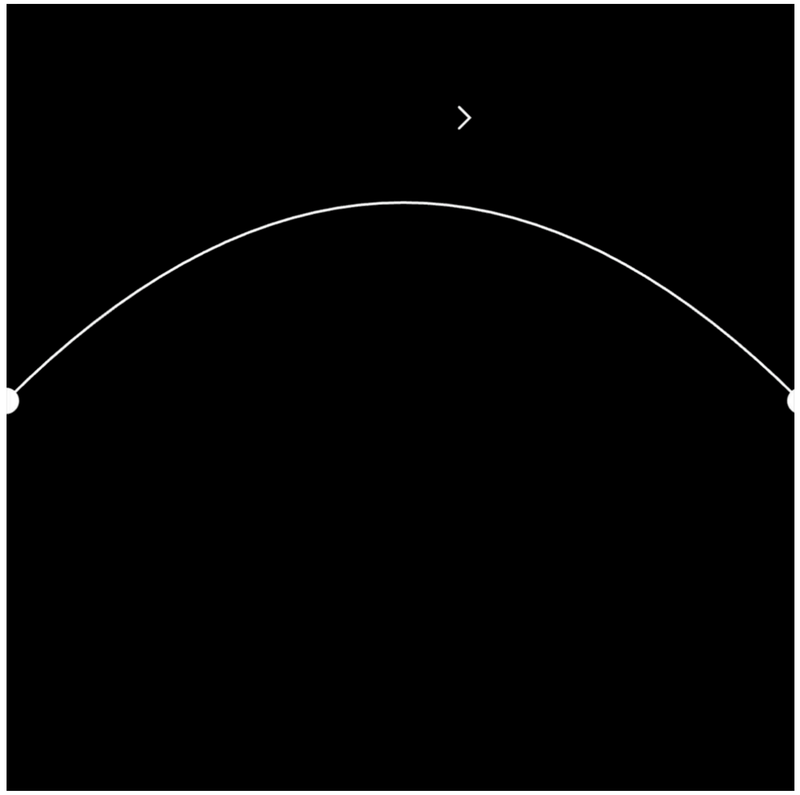
Main Sketch 4



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

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.

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

Quadratic Bézier Curve: Calculate Points

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

135

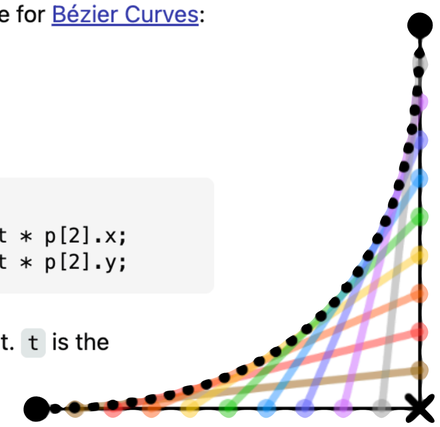
Use the quadratic Bézier formula, found, for instance, on the Wikipedia page for Bézier Curves:

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

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;
```

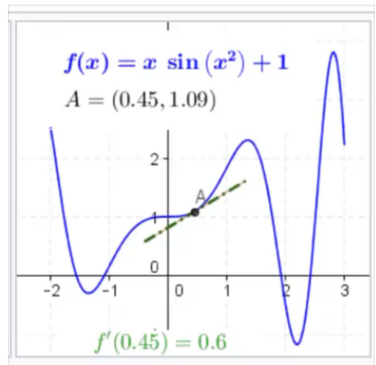
`p[0]` is the start point, `p[1]` is the control point, and `p[2]` is the end point. `t` is the parameter, which goes from 0 to 1.



3. Calculating the arrow’s direction

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

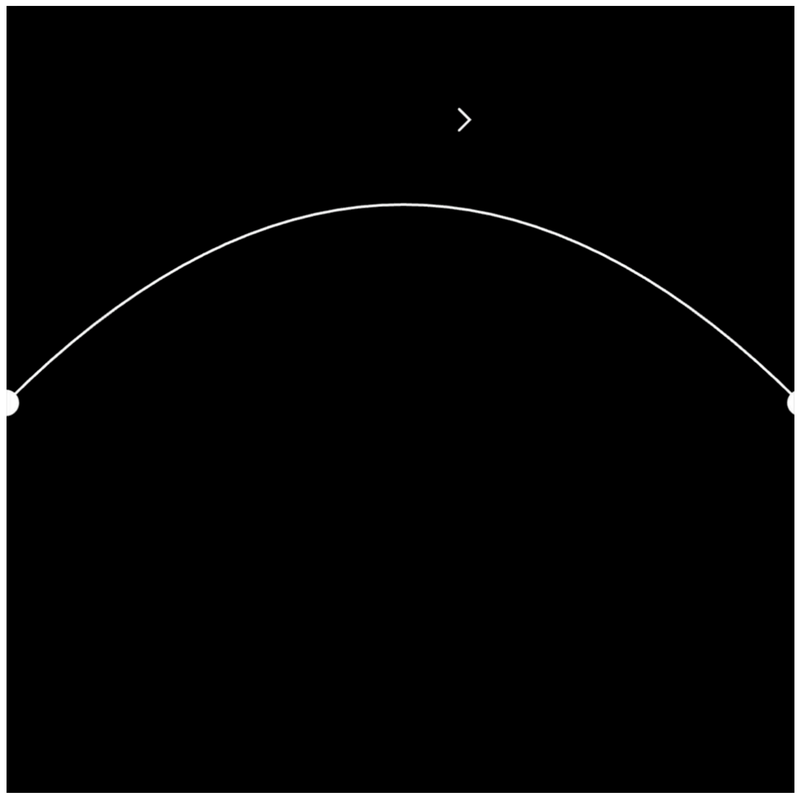
$$\mathbf{B}'(t) = 2(1-t)(\mathbf{P}_1 - \mathbf{P}_0) + 2t(\mathbf{P}_2 - \mathbf{P}_1),$$



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

Snippet Sketch: Moving Arrow

1. bezierPoint()

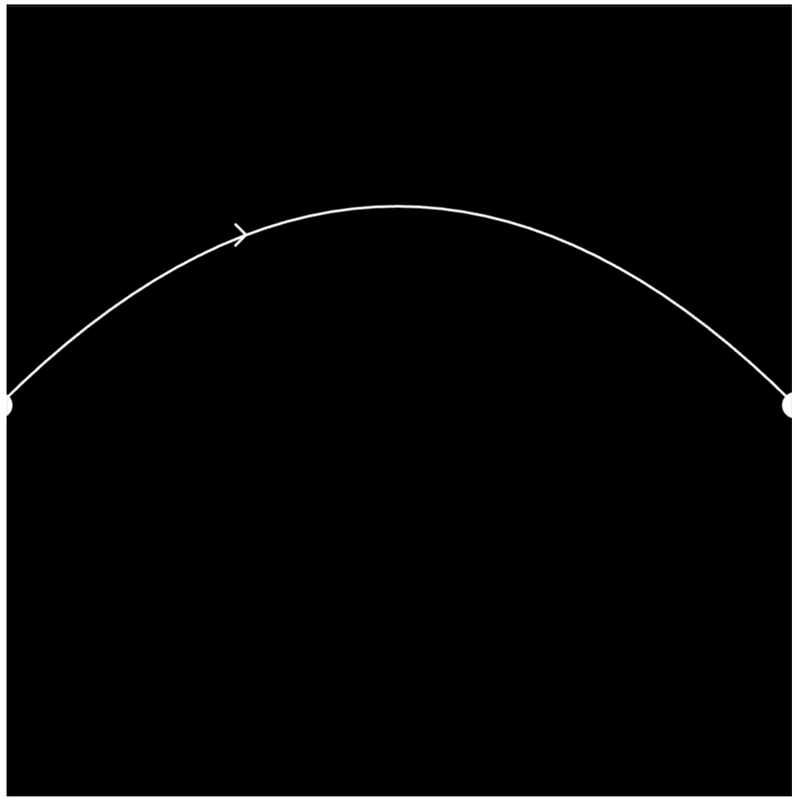


p5.js only has a syntax for bezierPoint

Syntax

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

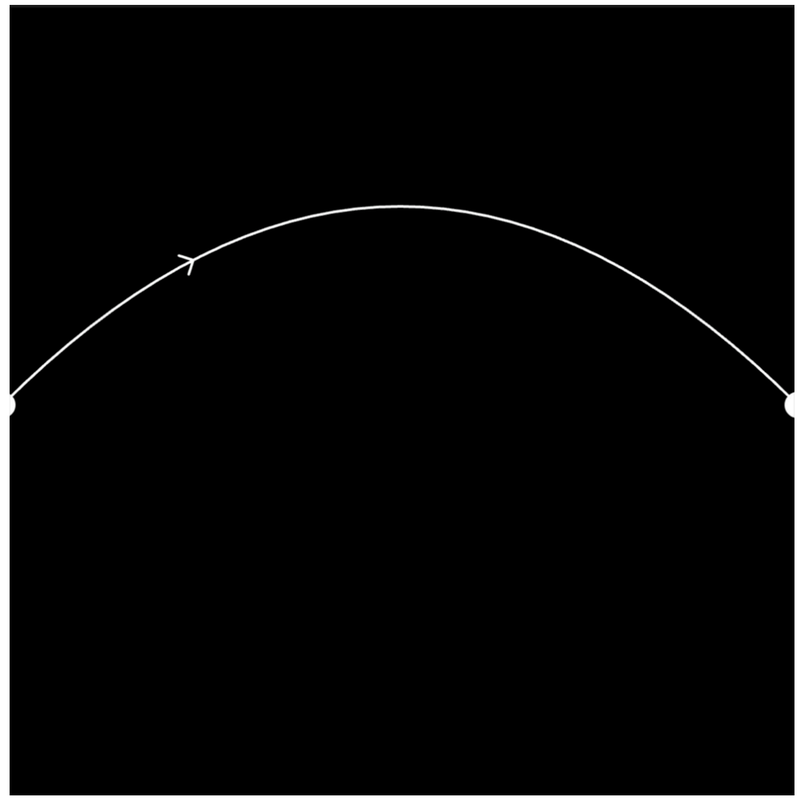
2. Calculating quadratic points



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

```
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

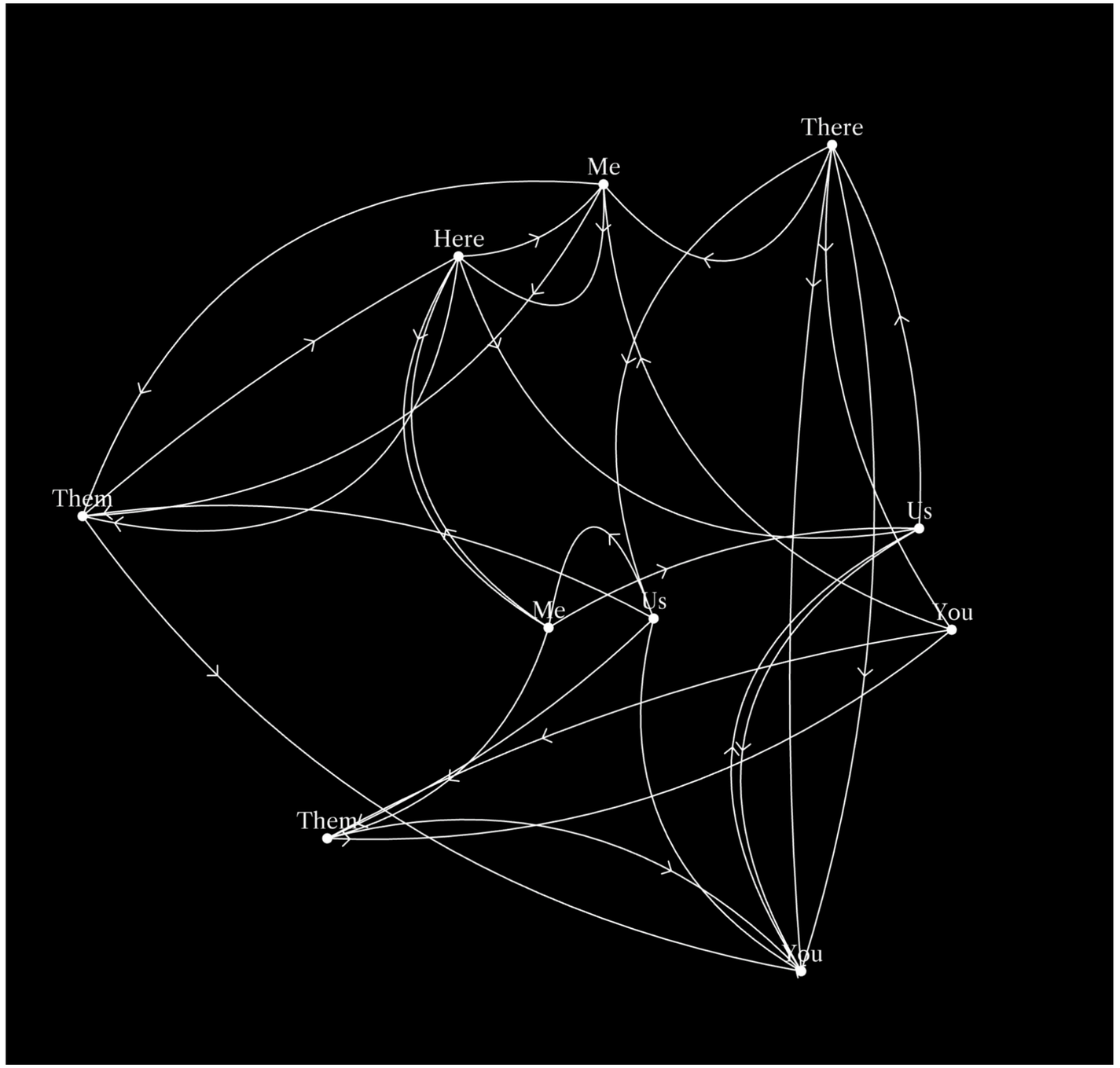


$$\mathbf{B}'(t) = 2(1-t)(\mathbf{P}_1 - \mathbf{P}_0) + 2t(\mathbf{P}_2 - \mathbf{P}_1),$$

```
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);
```

Unit 1 Methods of Iterating

describe (‘replicatingProcess’);
// The final output




```
sketch.js

// ===== DOCUMENTATION =====
// REFERENCES & RESOURCES
// Selected project: https://www.instagram.com/p/DOeV-szEnZm/?hl=en&img_index=1
// Array of texts: https://timrodenbroeker.de/courses/basic-datastructures/my-five-favorite-cities/
// Random point drag tutorial: https://www.youtube.com/watch?v=kjqo_uXn87I
// Documentation: https://www.w3schools.com/, https://developer.mozilla.org/en-US/
// Dot movement: https://www.youtube.com/watch?v=cH900adi1w Y
// Force directed graph: https://www.youtube.com/watch?v=YskU_gIpc0c // no use
// Click and drag: https://www.youtube.com/watch?v=kjqo_uXn87I
// Network case study: https://www.patrik-huebner.com/datadesigndictionary/

let font;
function preload() {
  font = loadFont('assets/LibreCaslonText-Regular.ttf');
}

// Canvas setting
const margin = 70;

// Dot settings
const numDots = 10;
const dotRadius = 10;
const labelOffset = 4;

// Arrow settings
const arrowSize = 7;
const arrowSpeed = 0.009;

// Curve settings
const thickness = 1.5
const bendMin = 20;
const bendMax = 250;

// Data arrays
const dots = []; //assign array
const connections = [];
const labels = ["Us", "Me", "Them", "You", "Here", "There"];

function setup() {
  createCanvas(1080, 1032);
  textFont(font);
  textSize(21);
  textAlign(CENTER, BOTTOM);

  // ========= 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=enNfb6pJj_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 perpendicular (rotate 90 degrees)
      bend.rotate(HALF_PI);

      // set how far it bends out
      let bendSide = random() < 0.5 ? -1 : 1; // go both side 50/50 upward/downward
      let bendDepth = random(bendMin, bendMax); // pick depth randomly
      bend.setMag(bendSide * bendDepth);

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

      // compute controlPoint once and keep it, so curves don't reshape every frame > use this name in draw function
      connections.push({
        startIndex,
        controlPoint,
        targetIndex,
        arrowPos: random(0,1),
      });
    }
  }
}

function draw() {

  // background('#000000'); //black
  background( '#34B5B8'); //green
  // background('#090992'); //blue

  // let elementColor = color('#FFFFFF'); //white
  let elementColor = color( '#000000'); //black
  // let elementColor = color('#34D508'); //green
  // let elementColor = color('#090992'); //blue

  // ========= 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

    // draw the curved line (only once per connection)
    noFill();
    stroke(elementColor);
    strokeWeight(thickness);

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

    // move the arrow
    let t = (frameCount * arrowSpeed + conn.arrowPos) % 1;

    // plotting the arrow position (quad point)
    let lx = quadLerp(p0.x, p1.x, p2.x, t);
    let ly = quadLerp(p0.y, p1.y, p2.y, t);

    // plotting the arrow direction (derivative>tangent)
    let tx = quadTangent(p0.x, p1.x, p2.x, t);
    let ty = quadTangent(p0.y, p1.y, p2.y, t);
    let angle = atan2(ty, tx);

    // draw arrow head
    push();
    translate(lx, ly);
    rotate(angle);
    stroke(elementColor);
    strokeWeight(thickness);
    noFill();

    beginShape();
    vertex(-arrowSize, -arrowSize);
    vertex(0, 0);
    vertex(-arrowSize, arrowSize);
    endShape();
    pop();
  }

  // ========= 5. draw all dots and text =========
  for (let dot of dots) {
    fill(elementColor);
    noStroke();
    circle(dot.position.x, dot.position.y, dotRadius);
    text(dot.label, dot.position.x, dot.position.y - labelOffset);
  }
}

// Formula functions
function quadLerp(p0, p1, p2, t) {
  return (1 - t) * (1 - t) * p0 + 2 * (1 - t) * t * p1 + t * t * p2;
}

function quadTangent(p0, p1, p2, t) {
  return 2 * (1 - t) * (p1 - p0) + 2 * t * (p2 - p1);
}

// REF: Click and drag by BarneyCodes
// I can't use his splice as it will bring up a new array and replace with other text

function mousePressed() {
  for (let i = dots.length - 1; i >= 0; i--) {
    let distance = dist(mouseX, mouseY, dots[i].position.x, dots[i].position.y);
    // added a small padding for easier clicking
    if (distance < dotRadius + 10) {
      dragDotIndex = i;
      break;
    }
  }
}

function mouseDragged() {
  if (dragDotIndex !== null) {
    // limit the dragging so dots stay inside the margin
    let targetX = constrain(mouseX, margin, width - margin);
    let targetY = constrain(mouseY, margin, height - margin);
    dots[dragDotIndex].position.set(targetX, targetY);
  }
}

function mouseReleased() {
  dragDotIndex = null;
}
```

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 valiables

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

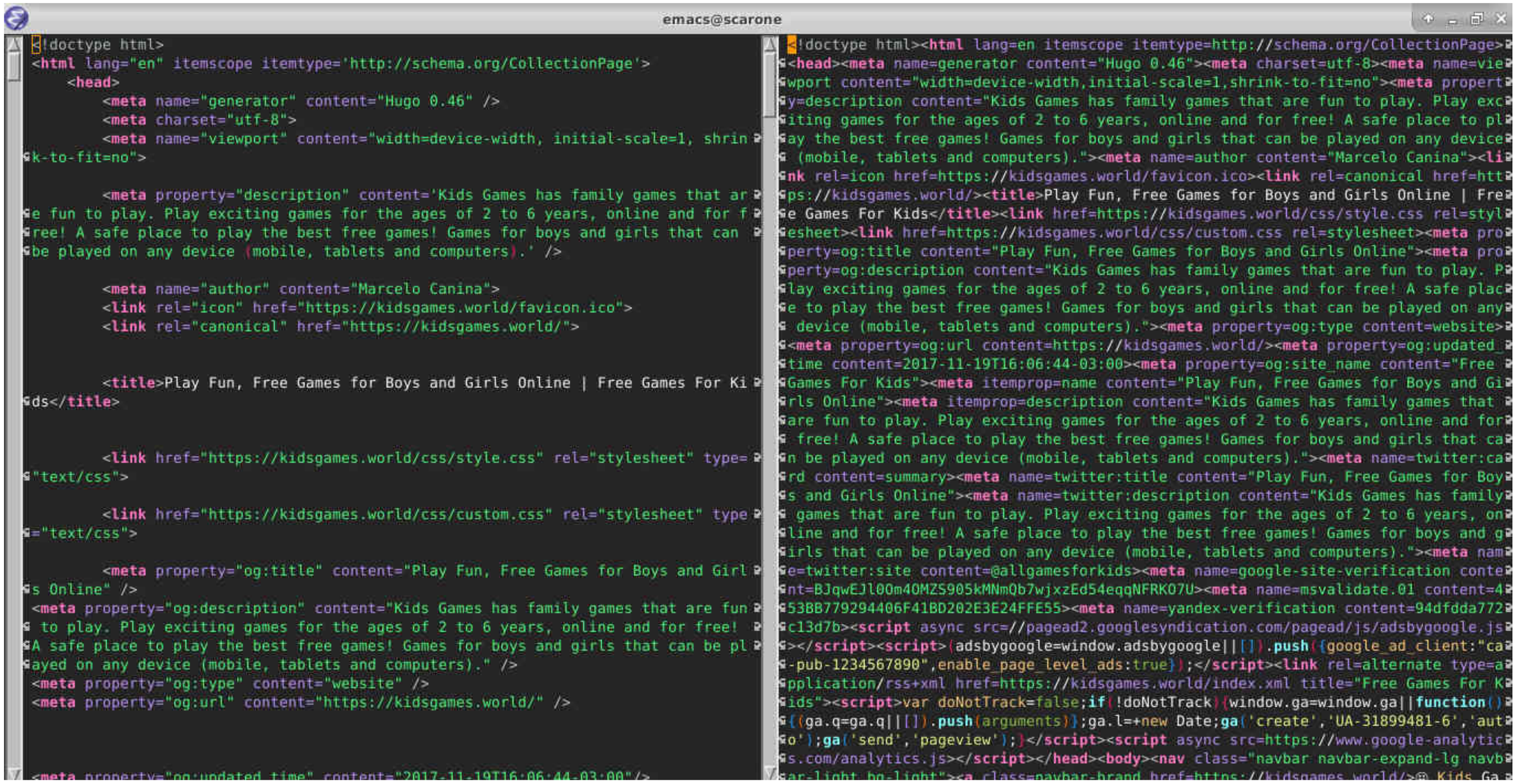
startIndex

```
// ========== 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();
```

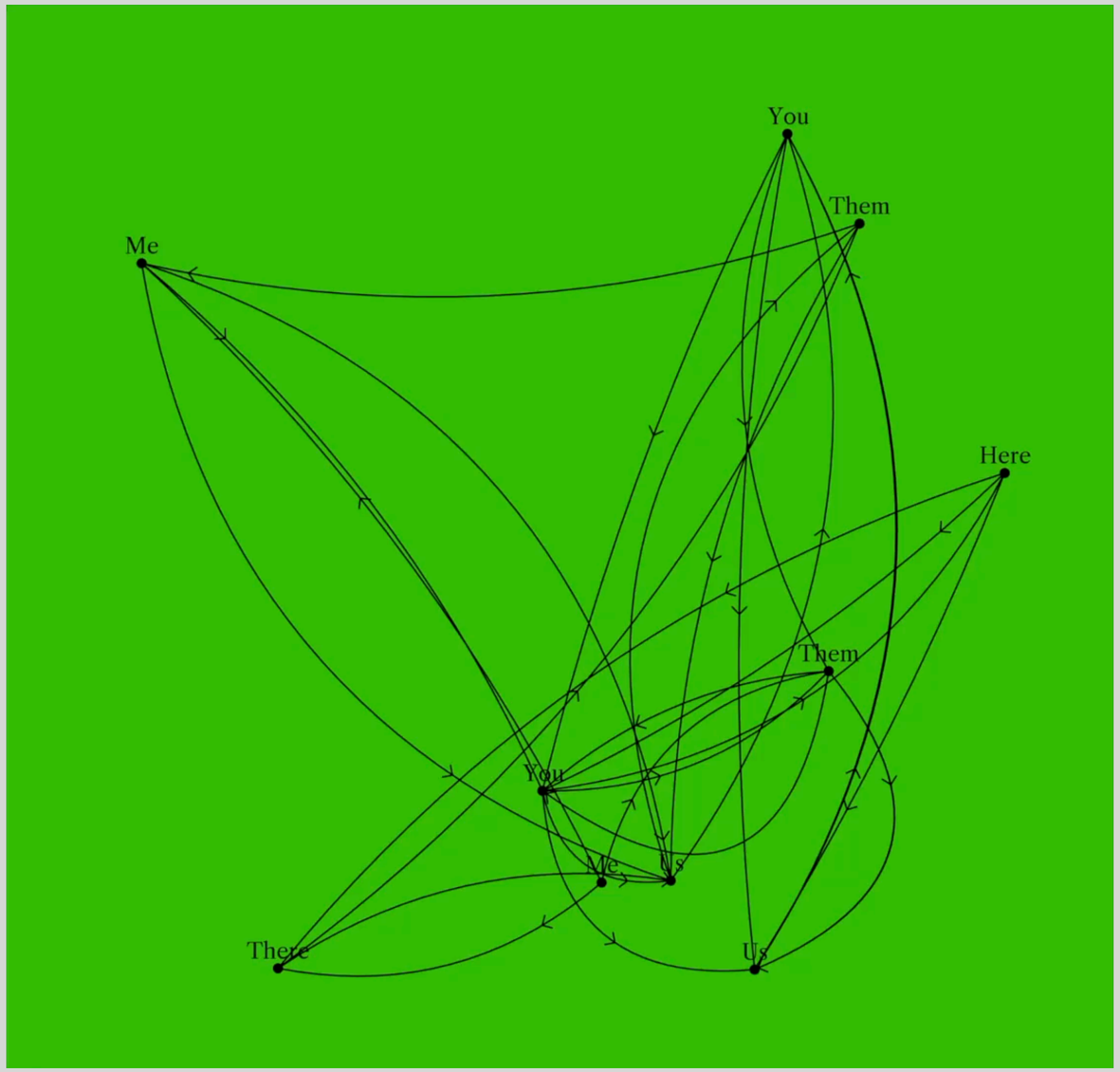
$$\mathbf{B}(t) = (1-t)^2\mathbf{P}_0 + 2(1-t)t\mathbf{P}_1 + t^2\mathbf{P}_2, \; t \in [0,1].$$

describe (‘replicatingProcess’);
// The difference between readability for humans vs computers

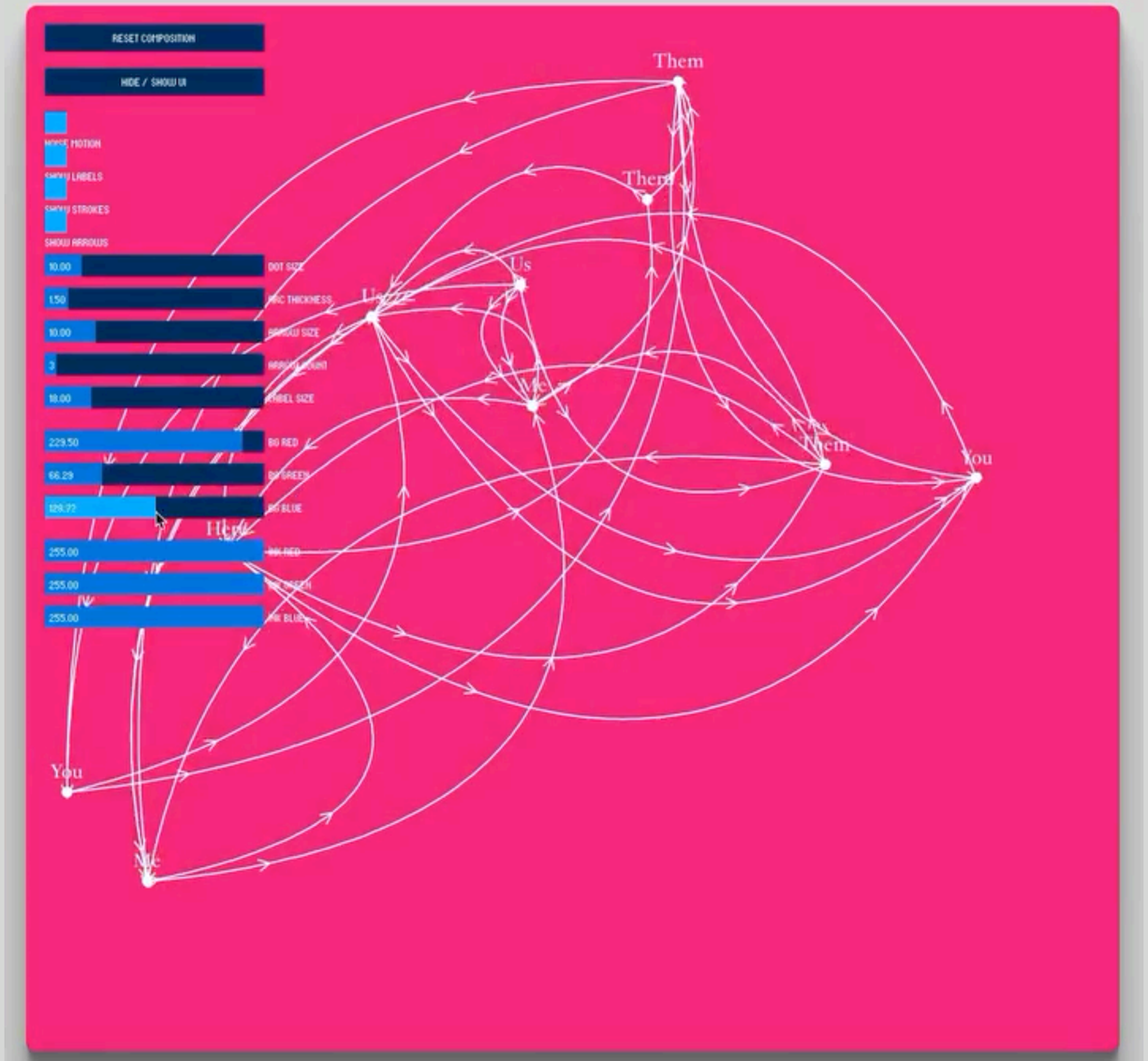
Minification is a process of removing unnecessary characters, making code smaller and lighter for faster load times and improved web performance.

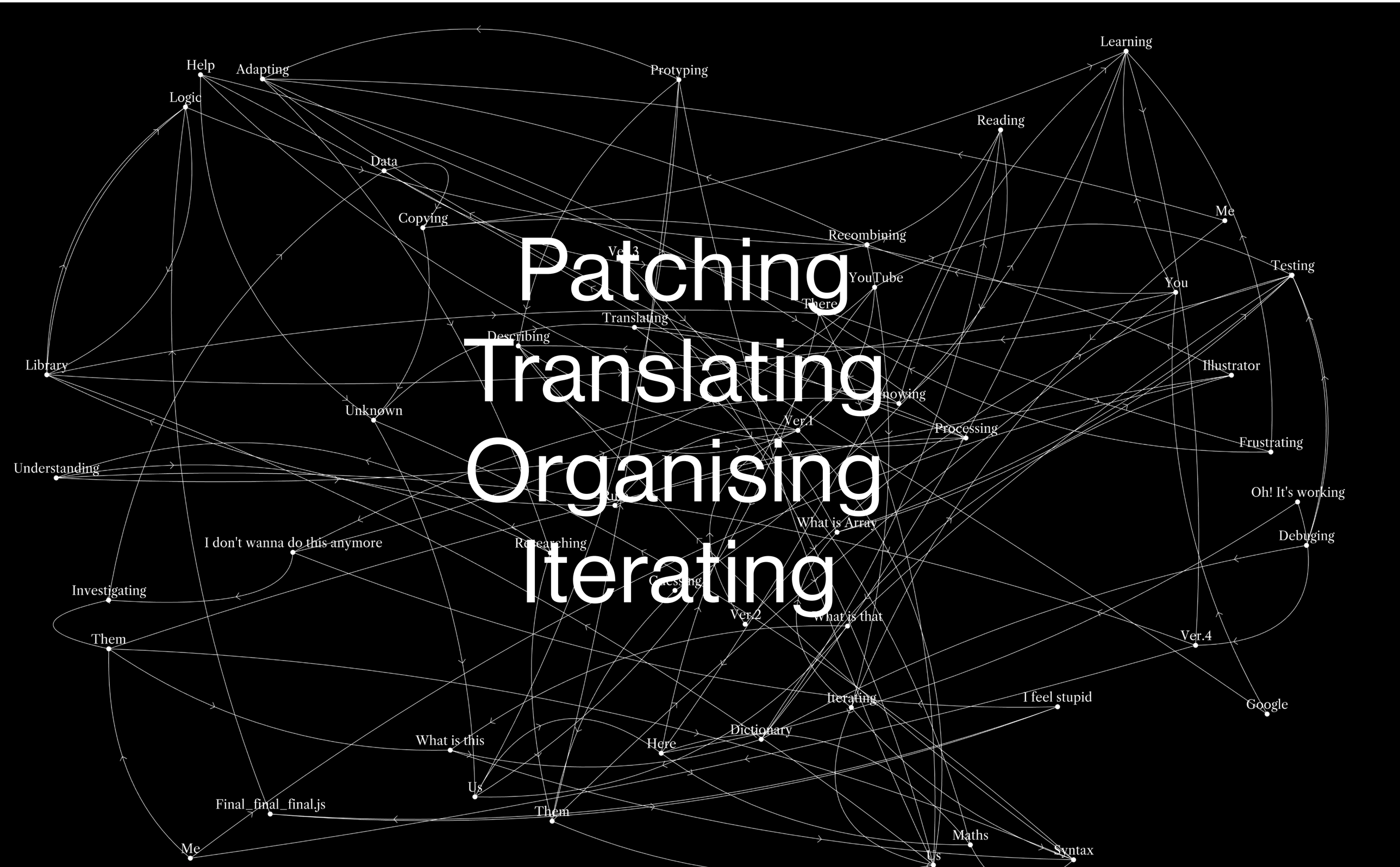


My Prototype



The Chosen Project




Patching
Translating
Organising
Iterating
Learning
Help
Adapting
Protyping
Logic
Reading
Data
Copying
Me
Recombining
Ver.3
Testing
YouTube
You
There
Translating
Describing
Library
Illustrator
Unknown
Knowing
Ver.1
Processing
Frustrating
Understanding
Oh! It's working
What is Array
Debuging
I don't wanna do this anymore
Researching
Investigating
Guessing
Ver.2
What is that
Them
Ver.4
Iterating
I feel stupid
Google
Dictionary
What is this
Here
Us
Final_final_final.js
Them
Maths
Me
Us
Syntax