

Process & Decision Documentation

Entry Header

Name: Ksenia Li

Primary responsibility for this work: Side quest enhancement, add game emotion using week 5 example 5 zipped files.

Goal of Work Session

Briefly describe what you were trying to accomplish during this phase of the assignment.

- Create a game out of the initial week 5 example
- Add emotion to the camera during movement
- Add collectable objects for blob

Tools, Resources, or Inputs Used

- ChatGPT 5.2

GenAI Documentation

I used GenAI to help me generate code for the camera movement and objects

If GenAI was used (keep each response as brief as possible):

Date Used: February 23rd

Tool Disclosure: ChatGPT 5.2

Purpose of Use: Generate code for game

Summary of Interaction: Helped me enhance the existing game by generating code.

ChatGPT helped me code an interface I would not have known how to do based on my own knowledge

Appendix

Prompt: Can you help me generate code for my side quest week 5 here is what I need to enhance: Create a reflective or meditative camera experience that scrolls through a world larger than the screen. Use pacing and motion to evoke emotion. Bonus: Hide small interactive symbols or objects for the camera to “discover. this is the code I have: class BlobPlayer { constructor() { this.x = 0; this.y = 0; this.r = 26; this.vx = 0; this.vy = 0; this.accel = 0.55; this.maxRun = 4.0; this.gravity = 0.65; this.jumpV = -11.0; this.frictionAir = 0.995;

```

this.frictionGround = 0.88; this.onGround = false; // wobble visuals this.t = 0; this.tSpeed =
0.01; this.wobble = 7; this.points = 48; this.wobbleFreq = 0.9; } spawnFromLevel(level) {
this.x = level.start.x; this.y = level.start.y; this.r = level.start.r; this.vx = 0; this.vy = 0;
this.onGround = false; this.gravity = level.gravity; this.jumpV = level.jumpV; } tryJump() { if
(this.onGround) { this.vy = this.jumpV; this.onGround = false; } } update(level) { // input let
move = 0; if (keyIsDown(65) || keyIsDown(LEFT_ARROW)) move -= 1; if (keyIsDown(68) || keyIsDown(RIGHT_ARROW)) move += 1; this.vx += this.accel * move; this.vx *=
this.onGround ? this.frictionGround : this.frictionAir; this.vx = constrain(this.vx, -
this.maxRun, this.maxRun); this.vy += this.gravity; // collider box let box = { x: this.x - this.r,
y: this.y - this.r, w: this.r * 2, h: this.r * 2, }; // move X box.x += this.vx; for (const s of
level.platforms) { if (BlobPlayer.overlap(box, s)) { if (this.vx > 0) box.x = s.x - box.w; else if
(this.vx < 0) box.x = s.x + s.w; this.vx = 0; } } // move Y box.y += this.vy; this.onGround = false;
for (const s of level.platforms) { if (BlobPlayer.overlap(box, s)) { if (this.vy > 0) { box.y = s.y -
box.h; this.vy = 0; this.onGround = true; } else if (this.vy < 0) { box.y = s.y + s.h; this.vy = 0; } }
} // write back this.x = box.x + box.w / 2; this.y = box.y + box.h / 2; // keep inside world
horizontally, allow falling below world this.x = constrain(this.x, this.r, level.w - this.r); this.t
+= this.tSpeed; } draw(colHex) { fill(color(colHex)); noStroke(); beginShape(); for (let i = 0; i <
this.points; i++) { const a = (i / this.points) * TAU; const n = noise( cos(a) * this.wobbleFreq +
100, sin(a) * this.wobbleFreq + 100, this.t, ); const rr = this.r + map(n, 0, 1, -this.wobble,
this.wobble); vertex(this.x + cos(a) * rr, this.y + sin(a) * rr); } endShape(CLOSE); } static
overlap(a, b) { return ( a.x < b.x + b.w && a.x + a.w > b.x && a.y < b.y + b.h && a.y + a.h > b.y );
} class Camera2D { constructor(viewW, viewH) { this.viewW = viewW; this.viewH = viewH;
this.x = 0; this.y = 0; } followSideScrollerX(targetX, lerpAmt) { const desired = targetX -
this.viewW / 2; this.x = lerp(this.x, desired, lerpAmt); } clampToWorld(worldW, worldH) {
const maxX = max(0, worldW - this.viewW); const maxY = max(0, worldH - this.viewH);
this.x = constrain(this.x, 0, maxX); this.y = constrain(this.y, 0, maxY); } begin() { push();
translate(-this.x, -this.y); } end() { pop(); } } class LevelLoader { static
fromLevelsJson(allData, index) { return new WorldLevel(allData.levels[index]); } }{
"schemaVersion": 1, "levels": [ { "name": "Intro Steps", "theme": { "bg": "#F0F0F0",
"platform": "#C8C8C8", "blob": "#1478FF" }, "gravity": 0.65, "jumpV": -11.0, "camera": {
"lerp": 0.12 }, "world": { "w": 2400, "h": 360, "deathY": 560 }, "start": { "x": 80, "y": 220, "r": 26 },
"platforms": [ { "x": 0, "y": 424, "w": 640, "h": 36 }, { "x": 120, "y": 354, "w": 120, "h": 12 },
{ "x": 300, "y": 304, "w": 90, "h": 12 }, { "x": 440, "y": 244, "w": 130, "h": 12 }, { "x": 520, "y": 354,
"w": 90, "h": 12 }, { "x": 570, "y": 164, "w": 30, "h": 12 }, { "x": 720, "y": 424, "w": 420, "h": 36 },
{ "x": 760, "y": 364, "w": 110, "h": 12 }, { "x": 930, "y": 324, "w": 110, "h": 12 }, { "x": 1090, "y": 284,
"w": 110, "h": 12 }, { "x": 1200, "y": 424, "w": 520, "h": 36 }, { "x": 1400, "y": 344, "w": 140,
"h": 12 } ] } ] } class Platform { constructor(x, y, w, h) { this.x = x; this.y = y; this.w = w; this.h =
h; } } const VIEW_W = 800; const VIEW_H = 480; let allLevelsData; let levelIndex = 0; let

```

```

level; let player; let cam; function preload() { allLevelsData = loadJSON("levels.json"); // levels.json beside index.html [web:122] } function setup() { createCanvas(VIEW_W, VIEW_H); textFont("sans-serif"); textSize(14); cam = new Camera2D(width, height); loadLevel(levelIndex); } function loadLevel(i) { level = LevelLoader.fromLevelsJson(allLevelsData, i); player = new BlobPlayer(); player.spawnFromLevel(level); cam.x = player.x - width / 2; cam.y = 0; cam.clampToWorld(level.w, level.h); } function draw() { // --- game state --- player.update(level); // Fall death → respawn if (player.y - player.r > level.deathY) { loadLevel(levelIndex); return; } // --- view state (data-driven smoothing) --- cam.followSideScrollerX(player.x, level.camLerp); cam.y = 0; cam.clampToWorld(level.w, level.h); // --- draw --- cam.begin(); level.drawWorld(); player.draw(level.theme.blob); cam.end(); // HUD fill(0); noStroke(); text(level.name + " (Example 5)", 10, 18); text("A/D or ←/→ move • Space/W/↑ jump • Fall = respawn", 10, 36); text("camLerp(JSON): " + level.camLerp + " world.w: " + level.w, 10, 54); text("cam: " + cam.x + ", " + cam.y, 10, 90); const p0 = level.platforms[0]; text(p0: x=${p0.x} y=${p0.y} w=${p0.w} h=${p0.h}, 10, 108); text("platforms: " + level.platforms.length + " start: " + level.start.x + "," + level.start.y, 10, 72, ); } function keyPressed() { if (key === " " || key === "W" || key === "w" || keyCode === UP_ARROW) { player.tryJump(); } if (key === "r" || key === "R") loadLevel(levelIndex); } class WorldLevel { constructor(levelJson) { this.name = levelJson.name ?? "Level"; this.theme = Object.assign({ bg: "#F0F0F0", platform: "#C8C8C8", blob: "#1478FF" }, levelJson.theme ?? {}, ); // Physics knobs this.gravity = levelJson.gravity ?? 0.65; this.jumpV = levelJson.jumpV ?? -11.0; // Camera knob (data-driven view state) this.camLerp = levelJson.camera?.lerp ?? 0.12; // World size + death line this.w = levelJson.world?.w ?? 2400; this.h = levelJson.world?.h ?? 360; this.deathY = levelJson.world?.deathY ?? this.h + 200; // Start this.start = Object.assign({ x: 80, y: 220, r: 26 }, levelJson.start ?? {}); // Platforms this.platforms = (levelJson.platforms ?? []).map( (p) => new Platform(p.x, p.y, p.w, p.h), ); } drawWorld() { background(this.theme.bg); push(); rectMode(CORNER); // critical: undo any global rectMode(CENTER) [web:230] noStroke(); fill(this.theme.platform); for (const p of this.platforms) rect(p.x, p.y, p.w, p.h); // x,y = top-left [web:234] pop(); } }

```

Thought for 14s

Absolutely — you already have a scrolling world + smooth follow. To make it **reflective / meditative**, you want the camera to feel like it has *breath*, *weight*, and *attention* (pacing + small emotional motion), not just “track player.x”.

Below is a drop-in upgrade that adds:

- **Cinematic smoothing (spring-damper)** instead of basic lerp
- **Deadzone** (player can move a bit before camera reacts → calmer)

- **Look-ahead** based on movement direction (gentle anticipation)
 - **Idle drift + “breathing” micro-motion** (meditative)
 - **Soft zoom** that reacts to speed (slower = closer / calmer)
 - **Bonus “discoverables”**: hidden symbols that reveal when the *camera* sees them, and optionally trigger a tiny “linger / focus” moment
-

1) Replace your `Camera2D` with this (meditative camera)

```
class Camera2D {
  constructor(viewW, viewH) {
    this.viewW = viewW;
    this.viewH = viewH;

    // top-left world coordinate of the view
    this.x = 0;
    this.y = 0;

    // internal camera "center" (for nicer math)
    this.cx = viewW / 2;
    this.cy = viewH / 2;

    // velocity for spring motion
    this.vx = 0;
    this.vy = 0;

    // presentation
    this.zoom = 1.0;
    this.zoomV = 0;

    // meditative knobs (tune)
    this.deadzoneX = 90;    // px around center before cam reacts
    this.deadzoneY = 70;
    this.lookAhead = 140;   // px
    this.maxLookAhead = 200;

    this.spring = 0.09;      // "pull" strength
    this.damping = 0.78;     // "calm" / friction

    // subtle drift/breathing
    this.t = random(1000);
    this.breatheAmp = 6;     // px
    this.breatheSpeed = 0.008;

    // moments: temporary focus/linger
    this.focusTimer = 0;
    this.focusX = 0;
    this.focusY = 0;
    this.focusZoom = 1.0;
  }

  // Call this when you want the camera to "notice" something
  cueFocus(x, y, holdFrames = 60, zoom = 1.06) {
    this.focusTimer = max(this.focusTimer, holdFrames);
  }
}
```

```

        this.focusX = x;
        this.focusY = y;
        this.focusZoom = zoom;
    }

    // Critically: update camera using player + level (called every frame)
    update(player, level) {
        // Desired camera center
        let desiredCx = player.x;
        let desiredCy = player.y;

        // Look-ahead based on horizontal velocity
        const la = constrain(player.vx * 28, -this.maxLookAhead,
        this.maxLookAhead);
        desiredCx += la;

        // Deadzone: only move if player exits calm region
        const dx = desiredCx - this.cx;
        if (abs(dx) < this.deadzoneX) desiredCx = this.cx;

        const dy = desiredCy - this.cy;
        if (abs(dy) < this.deadzoneY) desiredCy = this.cy;

        // If focusing on a discovered symbol, blend target toward it
        if (this.focusTimer > 0) {
            const k = 0.55; // how strongly we bias attention to the focus point
            desiredCx = lerp(desiredCx, this.focusX, k);
            desiredCy = lerp(desiredCy, this.focusY, k);
            this.focusTimer--;
        }

        // Add breathing drift (very subtle)
        this.t += this.breatheSpeed;
        const driftX = (noise(this.t) - 0.5) * 2 * this.breatheAmp;
        const driftY = (noise(this.t + 999) - 0.5) * 2 * (this.breatheAmp * 0.7);

        desiredCx += driftX;
        desiredCy += driftY;

        // Spring-damper toward desired center
        const ax = (desiredCx - this.cx) * this.spring;
        const ay = (desiredCy - this.cy) * this.spring;

        this.vx = (this.vx + ax) * this.damping;
        this.vy = (this.vy + ay) * this.damping;

        this.cx += this.vx;
        this.cy += this.vy;

        // Zoom: calmer when slow, slightly wider when fast
        const speed = abs(player.vx) + abs(player.vy) * 0.2;
        let desiredZoom = map(constrain(speed, 0, 6), 0, 6, 1.08, 0.98);
        if (this.focusTimer > 0) desiredZoom = lerp(desiredZoom, this.focusZoom,
        0.6);

        // Smooth zoom
        const zA = (desiredZoom - this.zoom) * 0.10;
    }
}

```

```

this.zoomV = (this.zoomV + zA) * 0.80;
this.zoom += this.zoomV;

// Convert center -> top-left (x,y)
this.x = this.cx - this.viewW / 2;
this.y = this.cy - this.viewH / 2;

// Clamp to world
this.clampToWorld(level.w, level.h);
}

clampToWorld(worldW, worldH) {
    // With zoom, visible area changes
    const visW = this.viewW / this.zoom;
    const visH = this.viewH / this.zoom;

    const maxX = max(0, worldW - visW);
    const maxY = max(0, worldH - visH);

    this.x = constrain(this.x, 0, maxX);
    this.y = constrain(this.y, 0, maxY);

    // keep center consistent with clamped x,y
    this.cx = this.x + visW / 2;
    this.cy = this.y + visH / 2;
}

// Use zoom + translation around the viewport center
begin() {
    push();
    translate(this.viewW / 2, this.viewH / 2);
    scale(this.zoom);
    translate(-(this.x + (this.viewW / 2) / this.zoom), -(this.y +
(this.viewH / 2) / this.zoom));
}

end() {
    pop();
}

// Helpful: is a world point inside the current view?
worldPointInView(px, py, margin = 0) {
    const visW = this.viewW / this.zoom;
    const visH = this.viewH / this.zoom;
    return (
        px >= this.x - margin &&
        px <= this.x + visW + margin &&
        py >= this.y - margin &&
        py <= this.y + visH + margin
    );
}
}

```

2) Add hidden “discoverables” (symbols the camera discovers)

A) Add this class

```
class HiddenSymbol {
    constructor(x, y, r = 10) {
        this.x = x;
        this.y = y;
        this.r = r;

        this.found = false;      // revealed when camera sees it
        this.collected = false; // optional: when player interacts
        this.pulseT = random(1000);
    }

    // Reveal when camera sees it
    update(cam, player) {
        if (!this.found && cam.worldPointInView(this.x, this.y, 30)) {
            this.found = true;

            // tiny “attention” moment
            cam.cueFocus(this.x, this.y, 45, 1.07);
        }

        // Optional: collect when close + press E
        const d = dist(player.x, player.y, this.x, this.y);
        if (this.found && !this.collected && d < this.r + player.r + 10) {
            if (keyIsDown(69)) { // E key
                this.collected = true;
                cam.cueFocus(this.x, this.y, 25, 1.09);
            }
        }

        this.pulseT += 0.02;
    }

    draw() {
        if (!this.found) return;

        push();
        noStroke();

        // soft pulse
        const pulse = 0.6 + 0.4 * sin(this.pulseT);
        const rr = this.r + pulse * 2;

        // subtle glow ring
        fill(0, 0, 0, this.collected ? 30 : 55);
        ellipse(this.x, this.y, rr * 2.6, rr * 2.6);

        // core
        fill(0, 0, 0, this.collected ? 90 : 140);
        ellipse(this.x, this.y, rr * 1.4, rr * 1.4);

        // tiny mark
        stroke(255, 220);
    }
}
```

```

        strokeWeight(2);
        line(this.x - 4, this.y, this.x + 4, this.y);
        line(this.x, this.y - 4, this.x, this.y + 4);

        pop();
    }
}

```

3) Wire symbols into `WorldLevel` (data-driven, like your platforms)

A) Update your JSON to include symbols (example)

Add this inside your level JSON object:

```

"symbols": [
  { "x": 585, "y": 140, "r": 9 },
  { "x": 980, "y": 290, "r": 10 },
  { "x": 1500, "y": 320, "r": 10 }
]

```

B) Update `WorldLevel` to load and draw them

```

class WorldLevel {
  constructor(levelJson) {
    this.name = levelJson.name ?? "Level";

    this.theme = Object.assign(
      { bg: "#F0F0F0", platform: "#C8C8C8", blob: "#1478FF" },
      levelJson.theme ?? {}
    );

    this.gravity = levelJson.gravity ?? 0.65;
    this.jumpV = levelJson.jumpV ?? -11.0;

    this.camLerp = levelJson.camera?.lerp ?? 0.12; // you can keep this, but
    we won't use it now

    this.w = levelJson.world?.w ?? 2400;
    this.h = levelJson.world?.h ?? 360;
    this.deathY = levelJson.world?.deathY ?? this.h + 200;

    this.start = Object.assign({ x: 80, y: 220, r: 26 }, levelJson.start ?? {});

    this.platforms = (levelJson.platforms ?? []).map(
      (p) => new Platform(p.x, p.y, p.w, p.h),
    );

    // NEW: hidden symbols
    this.symbols = (levelJson.symbols ?? []).map(
      (s) => new HiddenSymbol(s.x, s.y, s.r ?? 10),
    );
  }
}

```

```

drawWorld() {
    background(this.theme.bg);
    push();
    rectMode(CORNER);
    noStroke();
    fill(this.theme.platform);
    for (const p of this.platforms) rect(p.x, p.y, p.w, p.h);
    pop();
}

updateSymbols(cam, player) {
    for (const s of this.symbols) s.update(cam, player);
}

drawSymbols() {
    for (const s of this.symbols) s.draw();
}

collectedCount() {
    let n = 0;
    for (const s of this.symbols) if (s.collected) n++;
    return n;
}
}

```

4) Update your main `draw()` loop (camera pacing + symbols)

Replace your camera lines with this:

```

function draw() {
    // --- game state ---
    player.update(level);

    if (player.y - player.r > level.deathY) {
        loadLevel(levelIndex);
        return;
    }

    // NEW: meditative camera update
    cam.update(player, level);

    // NEW: symbols update (camera "discovers" them)
    level.updateSymbols(cam, player);

    // --- draw ---
    cam.begin();
    level.drawWorld();
    level.drawSymbols();           // draw symbols behind or above player (your
choice)
    player.draw(level.theme.blob);
    cam.end();

    // HUD
    fill(0);
}

```

```

noStroke();
text(level.name + " (Week 5 - Meditative Camera)", 10, 18);
text("A/D or ←→ move • Space/W/↑ jump • E collect symbol", 10, 36);
text("Symbols collected: " + level.collectedCount() + "/" +
level.symbols.length, 10, 54);
}

```

And in `loadLevel(i)` you can initialize the camera center cleanly:

```

function loadLevel(i) {
    level = LevelLoader.fromLevelsJson(allLevelsData, i);

    player = new BlobPlayer();
    player.spawnFromLevel(level);

    cam = new Camera2D(width, height);
    cam.cx = player.x;
    cam.cy = player.y;
    cam.update(player, level); // one update to clamp nicely
}

```

5) Tuning (the “emotion” sliders)

In `Camera2D`, tweak these first:

- **More calm:** increase `deadzoneX/Y`, `lower spring`, `increase damping`
- **More dreamy drift:** increase `breatheAmp`, slightly decrease `breatheSpeed`
- **More cinematic:** increase `lookAhead`, slightly increase zoom range (1.10 to 0.96)
- **More “attention moments”:** increase `cueFocus hold frames` (45 → 70)

Suggested calm defaults:

- `deadzoneX = 110, spring = 0.075, damping = 0.82, breatheAmp = 7`
-

If you want, I can also add **parallax background layers** (super effective for “meditative world larger than the screen”) using 2–3 slow-moving noise gradients or silhouette shapes that move at $0.2\times / 0.5\times$ camera speed.

Prompt: where do integrate this : "symbols": [{ "x": 585, "y": 140, "r": 9 }, { "x": 980, "y": 290, "r": 10 }, { "x": 1500, "y": 320, "r": 10 }] and this function draw() { // --- game state ---
`player.update(level); if (player.y - player.r > level.deathY) { loadLevel(levelIndex); return; } //`
`NEW: meditative camera update cam.update(player, level); // NEW: symbols update`
`(camera "discovers" them) level.updateSymbols(cam, player); // --- draw --- cam.begin();`
`level.drawWorld(); level.drawSymbols(); // draw symbols behind or above player (your`

```

choice) player.draw(level.theme.blob); cam.end(); // HUD fill(0); noStroke(); text(level.name
+ " (Week 5 — Meditative Camera)", 10, 18); text("A/D or ←/→ move • Space/W/↑ jump • E
collect symbol", 10, 36); text("Symbols collected: " + level.collectedCount() + "/" +
level.symbols.length, 10, 54); } and this function loadLevel(i) { level =
LevelLoader.fromLevelsJson(allLevelsData, i); player = new BlobPlayer();
player.spawnFromLevel(level); cam = new Camera2D(width, height); cam.cx = player.x;
cam.cy = player.y; cam.update(player, level); // one update to clamp nicely }

```

1) Where to put "symbols": [...]

Put it **inside the level object in your `levels.json`**, at the same level as "platforms", "start", "world", etc.

So in your `levels.json`, edit your existing level like this (I'm showing only the important part):

```

{
  "schemaVersion": 1,
  "levels": [
    {
      "name": "Intro Steps",
      "theme": {
        "bg": "#F0F0F0",
        "platform": "#C8C8C8",
        "blob": "#1478FF"
      },
      "gravity": 0.65,
      "jumpV": -11.0,
      "camera": { "lerp": 0.12 },
      "world": { "w": 2400, "h": 360, "deathY": 560 },
      "start": { "x": 80, "y": 220, "r": 26 },

      "symbols": [
        { "x": 585, "y": 140, "r": 9 },
        { "x": 980, "y": 290, "r": 10 },
        { "x": 1500, "y": 320, "r": 10 }
      ],
      "platforms": [
        { "x": 0, "y": 424, "w": 640, "h": 36 }
        // ...
      ]
    }
  ]
}

```

That's it for the JSON part.

2) Where to put the new `draw()` function

You already have a function `draw()` in your main sketch file (the one with `setup()`, `preload()`, etc.). You will **replace your current `draw()`** with the new one.

So in your main JS file, find:

```
function draw() {  
    // your current draw code...  
}
```

...and replace the whole thing with the updated `draw()` you pasted.

Key changes vs your current draw:

- you'll call `cam.update(player, level)` instead of `cam.followSideScrollerX(...)`
- you'll call `level.updateSymbols(cam, player)`
- you'll draw `level.drawSymbols()` between world and player

3) Where to put the new `loadLevel(i)` function

Same idea: you already have function `loadLevel(i)` in the main sketch file.

You will **replace your current `loadLevel(i)`** with the new one.

So find:

```
function loadLevel(i) {  
    // your current loadLevel code...  
}
```

Replace it with:

```
function loadLevel(i) {  
    level = LevelLoader.fromLevelsJson(allLevelsData, i);  
  
    player = new BlobPlayer();  
    player.spawnFromLevel(level);  
  
    cam = new Camera2D(width, height);  
    cam.cx = player.x;  
    cam.cy = player.y;  
    cam.update(player, level); // one update to clamp nicely  
}
```

Important: this version **recreates the camera each time you reload the level** (so it doesn't keep old velocity / drift). That's usually what you want for a clean respawn.

4) Don't forget the required supporting pieces (otherwise it'll error)

To make the above work, you need these in your project (once each):

1. **HiddenSymbol class** somewhere in your JS (same file is fine)
2. **Upgraded Camera2D class** (the meditative one)
3. In `WorldLevel`:
 - o load `this.symbols = ...` in the constructor
 - o add `updateSymbols(cam, player), drawSymbols(), collectedCount()`

If any of those are missing, you'll get errors like:

- `level.updateSymbols` is not a function
 - `HiddenSymbol` is not defined
 - `cam.update` is not a function
-

5) Practical “exact placement” checklist (fast)

Do this order:

A) `levels.json`

- Add `"symbols": [...]` inside the `level` object (right before `"platforms"` is perfect)

B) `main sketch JS`

- Replace `Camera2D` class with the new one
- Paste `HiddenSymbol` class (anywhere above where it's used)
- Replace `loadLevel(i)`
- Replace `draw()`

C) `WorldLevel JS`

- Add the symbols loading + methods
-

If you want, paste your current `WorldLevel` class again (the exact code you're running), and I'll give you a single edited version you can copy-paste with **only the necessary changes** (no extra).

Prompt: You mean level loaer?

Good question — **no, not LevelLoader**.

You do **NOT** change `LevelLoader`.

You modify:

- `levels.json`
 - `WorldLevel` class
 - `draw()` in your main sketch
 - `loadLevel(i)` in your main sketch
-

🔍 Why not LevelLoader?

Your `LevelLoader` only does this:

```
class LevelLoader {  
    static fromLevelsJson(allData, index) {  
        return new WorldLevel(allData.levels[index]);  
    }  
}
```

It simply passes the JSON data into `WorldLevel`.

So when you add "symbols" to the JSON, `WorldLevel` will receive it automatically.

You do **not** touch `LevelLoader`.

✓ What Actually Changes

1 Change `levels.json`

Inside your level object (same place as "platforms"):

```
"symbols": [
  { "x": 585, "y": 140, "r": 9 },
  { "x": 980, "y": 290, "r": 10 },
  { "x": 1500, "y": 320, "r": 10 }
],
```

That's it for JSON.

2 Modify `WorldLevel` (THIS is where symbols are integrated)

Inside the constructor, add:

```
this.symbols = (levelJson.symbols ?? []).map(
  (s) => new HiddenSymbol(s.x, s.y, s.r ?? 10),
);
```

Then add these methods inside `WorldLevel`:

```
updateSymbols(cam, player) {
  for (const s of this.symbols) s.update(cam, player);
}

drawSymbols() {
  for (const s of this.symbols) s.draw();
}

collectedCount() {
  let n = 0;
  for (const s of this.symbols) if (s.collected) n++;
  return n;
}
```

3 Replace `draw()` in your main sketch file

Not in `WorldLevel`.

Not in `LevelLoader`.

In the main file where `setup()` is.

4 Replace `loadLevel(i)` in your main sketch file

Again — not inside `WorldLevel`.

Architecture Overview (so this clicks)

```
levels.json → LevelLoader → WorldLevel → draw()
          ↓
          creates WorldLevel
          ↓
          WorldLevel stores symbols
          ↓
          draw() calls level.updateSymbols()
```

LevelLoader just passes data along.

WorldLevel is where the new feature actually lives.

If you'd like, paste your exact `WorldLevel` class and I'll give you the **exact edited version** ready to paste without breaking anything.

Prompt: i dont see anything showing up, review the code:

```
class BlobPlayer {
  constructor() { this.x = 0; this.y = 0; this.r = 26; this.vx = 0; this.vy = 0; this.accel = 0.55;
    this.maxRun = 4.0; this.gravity = 0.65; this.jumpV = -11.0; this.frictionAir = 0.995;
    this.frictionGround = 0.88; this.onGround = false; // wobble visuals this.t = 0;
    this.tSpeed = 0.01; this.wobble = 7; this.points = 48; this.wobbleFreq = 0.9; }
  spawnFromLevel(level) { this.x = level.start.x; this.y = level.start.y; this.r = level.start.r;
    this.vx = 0; this.vy = 0; this.onGround = false; this.gravity = level.gravity; this.jumpV =
    level.jumpV; } tryJump() { if (this.onGround) { this.vy = this.jumpV; this.onGround =
    false; } } update(level) { // input let move = 0; if (keyIsDown(65) ||
    keyIsDown(LEFT_ARROW)) move -= 1; if (keyIsDown(68) || keyIsDown(RIGHT_ARROW))
    move += 1; this.vx += this.accel * move; this.vx *= this.onGround ? this.frictionGround :
    this.frictionAir; this.vx = constrain(this.vx, -this.maxRun, this.maxRun); this.vy +=
    this.gravity; // collider box let box = { x: this.x - this.r, y: this.y - this.r, w: this.r * 2, h:
    this.r * 2, }; // move X box.x += this.vx; for (const s of level.platforms) { if
    (BlobPlayer.overlap(box, s)) { if (this.vx > 0) box.x = s.x - box.w; else if (this.vx < 0) box.x =
    s.x + s.w; this.vx = 0; } } // move Y box.y += this.vy; this.onGround = false; for (const s of
    level.platforms) { if (BlobPlayer.overlap(box, s)) { if (this.vy > 0) { box.y = s.y - box.h;
    this.vy = 0; this.onGround = true; } else if (this.vy < 0) { box.y = s.y + s.h; this.vy = 0; } } } } // write back
    this.x = box.x + box.w / 2; this.y = box.y + box.h / 2; // keep inside world
    horizontally, allow falling below world this.x = constrain(this.x, this.r, level.w - this.r);
    this.t += this.tSpeed; } draw(colHex) { fill(color(colHex)); noStroke(); beginShape(); for
    (let i = 0; i < this.points; i++) { const a = (i / this.points) * TAU; const n = noise( cos(a) *
    this.wobbleFreq + 100, sin(a) * this.wobbleFreq + 100, this.t, ); const rr = this.r + map(n,
```

```

0, 1, -this.wobble, this.wobble); vertex(this.x + cos(a) * rr, this.y + sin(a) * rr); }
endShape(CLOSE); } static overlap(a, b) { return ( a.x < b.x + b.w && a.x + a.w > b.x &&
a.y < b.y + b.h && a.y + a.h > b.y ); } } class Camera2D { constructor(viewW, viewH) {
this.viewW = viewW; this.viewH = viewH; // top-left world coordinate of the view this.x =
0; this.y = 0; // internal camera "center" (for nicer math) this.cx = viewW / 2; this.cy =
viewH / 2; // velocity for spring motion this.vx = 0; this.vy = 0; // presentation this.zoom =
1.0; this.zoomV = 0; // meditative knobs (tune) this.deadzoneX = 90; // px around
center before cam reacts this.deadzoneY = 70; this.lookAhead = 140; // px
this.maxLookAhead = 200; this.spring = 0.09; // "pull" strength this.damping = 0.78; //
"calm" / friction // subtle drift/breathing this.t = random(1000); this.breatheAmp = 6; //
px this.breatheSpeed = 0.008; // moments: temporary focus/linger this.focusTimer = 0;
this.focusX = 0; this.focusY = 0; this.focusZoom = 1.0; } // Call this when you want the
camera to "notice" something cueFocus(x, y, holdFrames = 60, zoom = 1.06) {
this.focusTimer = max(this.focusTimer, holdFrames); this.focusX = x; this.focusY = y;
this.focusZoom = zoom; } // Critically: update camera using player + level (called every
frame) update(player, level) { // Desired camera center let desiredCx = player.x; let
desiredCy = player.y; // Look-ahead based on horizontal velocity const la =
constrain(player.vx * 28, -this.maxLookAhead, this.maxLookAhead); desiredCx += la; //
Deadzone: only move if player exits calm region const dx = desiredCx - this.cx; if
(abs(dx) < this.deadzoneX) desiredCx = this.cx; const dy = desiredCy - this.cy; if
(abs(dy) < this.deadzoneY) desiredCy = this.cy; // If focusing on a discovered symbol,
blend target toward it if (this.focusTimer > 0) { const k = 0.55; // how strongly we bias
attention to the focus point desiredCx = lerp(desiredCx, this.focusX, k); desiredCy =
lerp(desiredCy, this.focusY, k); this.focusTimer--; } // Add breathing drift (very subtle)
this.t += this.breatheSpeed; const driftX = (noise(this.t) - 0.5) * 2 * this.breatheAmp;
const driftY = (noise(this.t + 999) - 0.5) * 2 * (this.breatheAmp * 0.7); desiredCx +=
driftX; desiredCy += driftY; // Spring-damper toward desired center const ax =
(desiredCx - this.cx) * this.spring; const ay = (desiredCy - this.cy) * this.spring; this.vx =
(this.vx + ax) * this.damping; this.vy = (this.vy + ay) * this.damping; this.cx += this.vx;
this.cy += this.vy; // Zoom: calmer when slow, slightly wider when fast const speed =
abs(player.vx) + abs(player.vy) * 0.2; let desiredZoom = map(constrain(speed, 0, 6), 0, 6,
1.08, 0.98); if (this.focusTimer > 0) desiredZoom = lerp(desiredZoom, this.focusZoom,
0.6); // Smooth zoom const zA = (desiredZoom - this.zoom) * 0.10; this.zoomV =
(this.zoomV + zA) * 0.80; this.zoom += this.zoomV; // Convert center -> top-left (x,y)
this.x = this.cx - this.viewW / 2; this.y = this.cy - this.viewH / 2; // Clamp to world
this.clampToWorld(level.w, level.h); } clampToWorld(worldW, worldH) { // With zoom,
visible area changes const visW = this.viewW / this.zoom; const visH = this.viewH /
this.zoom; const maxX = max(0, worldW - visW); const maxY = max(0, worldH - visH);

```

```

this.x = constrain(this.x, 0, maxX); this.y = constrain(this.y, 0, maxY); // keep center
consistent with clamped x,y this.cx = this.x + visW / 2; this.cy = this.y + visH / 2; } // Use
zoom + translation around the viewport center begin() { push(); translate(this.viewW /
2, this.viewH / 2); scale(this.zoom); translate(-(this.x + (this.viewW / 2) / this.zoom), -
(this.y + (this.viewH / 2) / this.zoom)); } end() { pop(); } // Helpful: is a world point inside
the current view? worldPointInView(px, py, margin = 0) { const visW = this.viewW /
this.zoom; const visH = this.viewH / this.zoom; return ( px >= this.x - margin && px <=
this.x + visW + margin && py >= this.y - margin && py <= this.y + visH + margin ); } }{
"include": [ "*.js", "libraries/*.js",
"c:\\\\Users\\\\david\\\\vscode\\\\extensions\\\\wmcicompsci.cs30-p5-
1.9.3\\\\p5types\\\\global.d.ts" ] } class LevelLoader { static fromLevelsJson(allData,
index) { return new WorldLevel(allData.levels[index]); } } { "schemaVersion": 1,
"levels": [ { "name": "Intro Steps", "theme": { "bg": "#F0F0F0", "platform": "#C8C8C8",
"blob": "#1478FF" }, "gravity": 0.65, "jumpV": -11.0, "camera": { "lerp": 0.12 }, "world": {
"w": 2400, "h": 360, "deathY": 560 }, "start": { "x": 80, "y": 220, "r": 26 }, "symbols": [ {
"x": 585, "y": 140, "r": 9 }, { "x": 980, "y": 290, "r": 10 }, { "x": 1500, "y": 320, "r": 10 } ],
"platforms": [ { "x": 0, "y": 424, "w": 640, "h": 36 } ] } } class Platform { constructor(x, y,
w, h) { this.x = x; this.y = y; this.w = w; this.h = h; } } const VIEW_W = 800; const VIEW_H =
480; let allLevelsData; let levelIndex = 0; let level; let player; let cam; function
preload() { allLevelsData = loadJSON("levels.json"); // levels.json beside index.html
[web:122] } function setup() { createCanvas(VIEW_W, VIEW_H); textFont("sans-serif");
textSize(14); cam = new Camera2D(width, height); loadLevel(levelIndex); } function
loadLevel(i) { level = LevelLoader.fromLevelsJson(allLevelsData, i); player = new
BlobPlayer(); player.spawnFromLevel(level); cam = new Camera2D(width, height);
cam.cx = player.x; cam.cy = player.y; cam.update(player, level); // one update to clamp
nicely } function draw() { // --- game state --- player.update(level); if (player.y - player.r >
level.deathY) { loadLevel(levelIndex); return; } // NEW: meditative camera update
cam.update(player, level); // NEW: symbols update (camera "discovers" them)
level.updateSymbols(cam, player); // --- draw --- cam.begin(); level.drawWorld();
level.drawSymbols(); // draw symbols behind or above player (your choice)
player.draw(level.theme.blob); cam.end(); // HUD fill(0); noStroke(); text(level.name + "
(Week 5 — Meditative Camera)", 10, 18); text("A/D or ←/→ move • Space/W/↑ jump • E
collect symbol", 10, 36); text("Symbols collected: " + level.collectedCount() + "/" +
level.symbols.length, 10, 54); } function keyPressed() { if (key === " " || key === "W" ||
key === "w" || keyCode === UP_ARROW) { player.tryJump(); } if (key === "r" || key === "R")
loadLevel(levelIndex); } class WorldLevel { constructor(levelJson) { this.name =
levelJson.name ?? "Level"; this.theme = Object.assign( { bg: "#F0F0F0", platform:
"#C8C8C8", blob: "#1478FF" }, levelJson.theme ?? {}, ); this.gravity = levelJson.gravity
}

```

```

?? 0.65; this.jumpV = levelJson.jumpV ?? -11.0; this.camLerp = levelJson.camera?.lerp
?? 0.12; // you can keep this, but we won't use it now this.w = levelJson.world?.w ??
2400; this.h = levelJson.world?.h ?? 360; this.deathY = levelJson.world?.deathY ??
this.h + 200; this.start = Object.assign({ x: 80, y: 220, r: 26 }, levelJson.start ?? {});
this.platforms = (levelJson.platforms ?? []).map( (p) => new Platform(p.x, p.y, p.w, p.h),
); // NEW: hidden symbols this.symbols = (levelJson.symbols ?? []).map( (s) => new
HiddenSymbol(s.x, s.y, s.r ?? 10), ); } drawWorld() { background(this.theme.bg); push();
rectMode(CORNER); noStroke(); fill(this.theme.platform); for (const p of
this.platforms) rect(p.x, p.y, p.w, p.h); pop(); } updateSymbols(cam, player) { for (const
s of this.symbols) s.update(cam, player); } drawSymbols() { for (const s of
this.symbols) s.draw(); } collectedCount() { let n = 0; for (const s of this.symbols) if
(s.collected) n++; return n; } }

```

I see the issue immediately from what you pasted:

 Your project is missing the `HiddenSymbol` class

In your code dump, you **never define**:

```
class HiddenSymbol { ... }
```

...but your `WorldLevel` constructor does this:

```
this.symbols = (levelJson.symbols ?? []).map(
  (s) => new HiddenSymbol(s.x, s.y, s.r ?? 10),
);
```

So one of two things is happening:

- **Best case:** your browser console is throwing `ReferenceError: HiddenSymbol is not defined` and the sketch stops / symbols never update.
- **Worst case:** it silently fails depending on environment, but either way: nothing will show.

 Fix: **add the `HiddenSymbol` class** (exact code below) into your sketch file.

1) Add this class (paste it ABOVE `class WorldLevel`)

Put it after `class Platform` or right before `class WorldLevel`:

```
class HiddenSymbol {
  constructor(x, y, r = 10) {
```

```

this.x = x;
this.y = y;
this.r = r;

this.found = false;      // revealed when camera sees it
this.collected = false; // optional: when player interacts
this.pulseT = random(1000);
}

update(cam, player) {
  // reveal when inside camera view
  if (!this.found && cam.worldPointInView(this.x, this.y, 30)) {
    this.found = true;

    // small "linger" moment
    cam.cueFocus(this.x, this.y, 45, 1.07);
  }

  // optional collect (press E when near)
  const d = dist(player.x, player.y, this.x, this.y);
  if (this.found && !this.collected && d < this.r + player.r + 10) {
    if (keyIsDown(69)) { // E
      this.collected = true;
      cam.cueFocus(this.x, this.y, 25, 1.09);
    }
  }

  this.pulseT += 0.02;
}

draw() {
  if (!this.found) return;

  push();
  noStroke();

  const pulse = 0.6 + 0.4 * sin(this.pulseT);
  const rr = this.r + pulse * 2;

  // glow
  fill(0, 0, 0, this.collected ? 30 : 55);
  ellipse(this.x, this.y, rr * 2.6, rr * 2.6);

  // core
  fill(0, 0, 0, this.collected ? 90 : 140);
  ellipse(this.x, this.y, rr * 1.4, rr * 1.4);

  // cross mark
  stroke(255, 220);
  strokeWeight(2);
  line(this.x - 4, this.y, this.x + 4, this.y);
  line(this.x, this.y - 4, this.x, this.y + 4);

  pop();
}
}

```

That should make symbols appear once the camera scrolls over them.

2) Second issue: your symbol Y positions are probably off-screen

Your world is:

```
"h": 360
```

But your canvas is:

```
VIEW_H = 480
```

And your only platform is at y: 424, which is **below a 360px world**. That mismatch can make camera clamping + visibility behave weirdly.

Also your symbols are at y: 140, 290, 320. Those are “in the upper-middle” of a 360 world, but your player starts at y: 220 and gravity pulls them down — so you might never naturally “see” the symbol at y=140 unless the camera follows Y (y), which it currently does.

However: your camera *does* follow `player.y` (`desiredCy = player.y`), so symbols can be seen if the player moves upward on platforms— but right now you only have **one platform** in JSON:

```
{ "x": 0, "y": 424, "w": 640, "h": 36 }
```

So there’s nothing to jump on to reach y=140 area; player just falls / stands near bottom.

 Quick fix for visibility while testing:

Put one symbol near the ground where the player actually is, e.g.:

```
"symbols": [
    { "x": 200, "y": 380, "r": 10 }
],
```

But better fix is:

Make world height match your ground (recommended)

Either:

- Change JSON world height to at least **480** (or 560), since your death line is 560 and ground is 424:

```
"world": { "w": 2400, "h": 560, "deathY": 560 }
```

OR

- Move ground up into the world (e.g., y=324) if you want h=360.

Right now your numbers contradict each other.

3) Third issue: you pasted non-JS files into the same JS blob

I see this inside your JS:

```
{  
  "include": [  
    "*.js",  
    "libraries/*.js",  
    ...  
  ]  
}
```

That is **tsconfig / vscode config JSON**, not JavaScript code. If that's literally in your `.js` file, it will break parsing and nothing after it runs.

Same with the big JSON level block you pasted — that must live in **levels.json**, not inside your JS file.

 Make sure:

- `levels.json` contains ONLY JSON
- your JS contains ONLY JS

If you accidentally pasted JSON into the JS file, your sketch will fail.

Do this debugging step (takes 10 seconds)

Open browser devtools console.

If you see any of these errors, it confirms the above:

- `HiddenSymbol` is not defined
 - Unexpected token `'.'` (means JSON is inside JS)
 - Unexpected token `'{'` near that `include` block
-

Minimal “it must show something” test

After you add `HiddenSymbol`, temporarily force symbols to draw even if not found:

Change:

```
if (!this.found) return;
```

to:

```
// if (!this.found) return;
```

Now you should see them immediately (somewhere in the world). Once confirmed, turn it back on.

If you paste **your browser console errors** (just copy the red lines), I can tell you exactly which of the 3 issues is blocking you in your case.

Prompt: how do i locate everything closer? the game has everything to spread apart making it impossible to play

Good — this is now a **world scale problem**, not a code bug.

Right now your world is:

```
"world": { "w": 2400, "h": 360 }
```

That means:

- The level is **2400px wide**
- Your canvas is only **800px wide**
- So everything feels very far apart

You have three clean options.

OPTION 1 (Recommended for Week 5): Shrink the World

Go into `levels.json` and reduce world width.

Change this:

```
"world": { "w": 2400, "h": 360, "deathY": 560 }
```

To something like:

```
"world": { "w": 1200, "h": 480, "deathY": 560 }
```

That instantly halves the scroll distance.

If it still feels too wide:

```
"world": { "w": 1000, "h": 480, "deathY": 560 }
```

✓ OPTION 2: Move Platforms + Symbols Closer

Right now your platforms are likely spaced far apart.

Example:

```
{ "x": 980, "y": 290 }
{ "x": 1500, "y": 320 }
```

Those are 500px apart.

Instead compress them:

```
{ "x": 580, "y": 290 }
{ "x": 800, "y": 320 }
```

Think of your world like a film set — bring your props closer.

✓ OPTION 3 (Cinematic Fix): Increase Camera Zoom

Right now zoom varies between:

1.08 → 0.98

If things feel too far apart visually, make the camera slightly more zoomed-in.

Inside `Camera2D.update()`, change:

```
let desiredZoom = map(constrain(speed, 0, 6), 0, 6, 1.08, 0.98);
```

To:

```
let desiredZoom = map(constrain(speed, 0, 6), 0, 6, 1.15, 1.02);
```

That makes the world feel tighter and more intimate.



Best Quick Fix (Do This First)

Change your world to:

```
"world": { "w": 1200, "h": 480, "deathY": 560 }
```

Then reload.

That alone will make it playable.



Also Important

Your current mismatch:

- $h = 360$
- Ground platform at $y = 424$
- Canvas height = 480

That's inconsistent.

Your world height should be at least:

```
"h": 480
```

or your camera clamping will behave strangely.



Design Advice (for your assignment)

For a “meditative camera experience,” you actually want:

- Smaller world
- Slower movement
- Denser discoverables
- Less dead space

A huge empty 2400px level is more “Mario speedrun” than reflective.

If you want, tell me:

- What width feels good?
- Do you want it slow and intimate or wide and exploratory?

I'll give you exact tuned values for your emotional goal.

```
Prompt: { "schemaVersion": 1, "levels": [ { "name": "Intro Steps", "theme": { "bg": "#F0F0F0", "platform": "#C8C8C8", "blob": "#1478FF" }, "gravity": 0.65, "jumpV": -11.0, "camera": { "lerp": 0.12 }, "world": { "w": 1200, "h": 480, "deathY": 560 }, "start": { "x": 80, "y": 220, "r": 26 }, "symbols": [ { "x": 585, "y": 140, "r": 9 }, { "x": 980, "y": 290, "r": 10 }, { "x": 1500, "y": 320, "r": 10 } ], "platforms": [ { "x": 0, "y": 424, "w": 640, "h": 36 } ] } }  
//the platforms are still too far
```

Yep — because **your platform list only has ONE platform** right now, and your symbols still include an `x: 1500` which is **outside** your new world width (`w: 1200`). So the “play space” is basically empty + spread.

What you want is a compressed layout pass:

- bring symbols into 0...1200
- add a handful of platforms with **short gaps** (120–220px) so it feels playable
- optionally add a “ground strip” across most of the level so you’re not falling

Below is a **copy-paste replacement** for your `platforms` and `symbols` that will feel **close + playable** in a 1200px world.

✓ Replace your `symbols + platforms` with this

```
"symbols": [  
  { "x": 220, "y": 320, "r": 9 },  
  { "x": 520, "y": 260, "r": 10 },  
  { "x": 860, "y": 300, "r": 10 }  
,
```

```

"platforms": [
  // Ground: continuous floor so it's playable
  { "x": 0,      "y": 424, "w": 1200, "h": 56 },

  // Small "steps" close together (easy rhythm)
  { "x": 140,    "y": 360, "w": 120,   "h": 14 },
  { "x": 300,    "y": 320, "w": 110,   "h": 14 },
  { "x": 460,    "y": 280, "w": 120,   "h": 14 },
  { "x": 640,    "y": 320, "w": 120,   "h": 14 },
  { "x": 820,    "y": 290, "w": 120,   "h": 14 },
  { "x": 980,    "y": 340, "w": 140,   "h": 14 },

  // A tiny "perch" near the end
  { "x": 1080,   "y": 260, "w": 60,    "h": 14 }
]

```

Why this fixes it

- Your world is 1200 wide → everything stays inside $x \leq 1200$
 - Ground is full width → no dead empty stretches
 - Platforms are spaced ~160–180px apart → reachable with your jump
-

Two quick rules so it stays playable

Rule A: keep everything inside world bounds

If "world": { "w": 1200 }, then:

- platforms must satisfy $x + w \leq 1200$
- symbols must satisfy $x \leq 1200$

Your current symbol at $x: 1500$ can never be reached/seen.

Rule B: “close spacing” numbers

For your physics (`jumpV: -11, gravity: 0.65`), good beginner spacing:

- **horizontal gaps:** 120–220px
 - **vertical changes:** 30–80px
-

If you want it even tighter (super “meditative”)

Change the world width to **900** and compress again. But first get it playable with the block above.

If you paste your **full current levels.json** (after changes), I can give you an even better “meditative path” layout with:

- a slow rising arc
- 3 discovery moments
- a calm ending “rest platform” where the camera lingers

Prompt: now there are no holes to fall through for the blob as in the original code. Also the blob is now collecting the pieces

Got it — two separate things to fix:

1. **You want holes again** (places to fall through), instead of a continuous floor.
2. **You don't want auto-collect;** you want a deliberate “discover / interact” moment.

Let's fix both cleanly.

1) Bring back holes (split the ground into segments)

Right now you have:

```
{ "x": 0, "y": 424, "w": 1200, "h": 56 }
```

That's one solid strip → no gaps.

Replace your ground with multiple pieces (gaps between them)

Use this `platforms` block (copy/paste). It keeps the close spacing, but adds 3 holes:

```
"platforms": [
  // Ground segments (gaps = holes)
  { "x": 0, "y": 424, "w": 260, "h": 56 },
  { "x": 330, "y": 424, "w": 240, "h": 56 },
  { "x": 640, "y": 424, "w": 210, "h": 56 },
  { "x": 920, "y": 424, "w": 280, "h": 56 },

  // Steps (keep them close)
```

```

{ "x": 140, "y": 360, "w": 120, "h": 14 },
{ "x": 300, "y": 320, "w": 110, "h": 14 },
{ "x": 460, "y": 280, "w": 120, "h": 14 },
{ "x": 640, "y": 320, "w": 120, "h": 14 },
{ "x": 820, "y": 290, "w": 120, "h": 14 },
{ "x": 980, "y": 340, "w": 140, "h": 14 },
{ "x": 1080, "y": 260, "w": 60, "h": 14 }
]

```

Holes are the “missing” x ranges:

- 260→330
- 570→640
- 850→920

If you want *bigger* holes, increase those gaps.

2) Stop auto-collect: require a single press of E (edge-trigger)

Right now your `HiddenSymbol.update()` uses:

```
if (keyIsDown(69)) { ... }
```

That can feel like “auto-collect” because:

- if E is held even briefly, it counts
- depending on your keyboard repeat / focus, it can register more than you expect

Fix: collect only when E is *pressed once*, not held

In `HiddenSymbol`, remove the `keyIsDown(69)` check and instead call a method `tryCollect(player)` from `keyPressed()`.

A) Update `HiddenSymbol` class (replace ONLY collection logic)

Add this method inside `HiddenSymbol`:

```
tryCollect(cam, player) {
    if (!this.found || this.collected) return false;

    const d = dist(player.x, player.y, this.x, this.y);
    if (d < this.r + player.r + 10) {
        this.collected = true;
        if (cam?.cueFocus) cam.cueFocus(this.x, this.y, 25, 1.09);
        return true;
    }
}
```

```
    return false;  
}
```

Then in `update(cam, player)`, **delete** the whole “press E when near” block (so it does NOT collect in `update()` anymore). Your `update()` should only reveal + pulse.

So your `update()` becomes basically:

```
update(cam, player) {  
    if (!this.found && cam.worldPointInView(this.x, this.y, 30)) {  
        this.found = true;  
        if (cam.cueFocus) cam.cueFocus(this.x, this.y, 45, 1.07);  
    }  
    this.pulseT += 0.02;  
}
```

B) In sketch.js, update keyPressed() to trigger collection

Add this inside your `keyPressed()`:

```
function keyPressed() {  
    if (key === " " || key === "w" || key === "W" || keyCode === UP_ARROW) {  
        player.tryJump();  
    }  
  
    // Press E to collect (once)  
    if (key === "e" || key === "E") {  
        for (const s of level.symbols) {  
            if (s.tryCollect(cam, player)) break; // collect at most one per press  
        }  
    }  
  
    if (key === "r" || key === "R") loadLevel(levelIndex);  
}
```

Now it will **never** auto-collect:

- Symbols reveal when camera sees them
- Collection only happens on a single **E press**, only if you’re close

Quick sanity checks

- Make sure all symbol `x` values are **≤ 1200** (your world width).
 - If you fall and never respawn, ensure `deathY` is below the ground (you set 560 — that’s good for ground at 424).
-

If you paste your current `HiddenSymbol` class (or tell me which file it's in: `WorldLevel.js` or `discoverables.js`), I'll give you the exact final version with no "replace this block" ambiguity.

```
Prompt: class HiddenSymbol { constructor(x, y, r = 10) { this.x = x; this.y = y; this.r = r;
this.found = false; // revealed when camera sees it this.collected = false; // optional:
when player interacts this.pulseT = random(1000); } tryCollect(cam, player) { if
(!this.found || this.collected) return false; const d = dist(player.x, player.y, this.x,
this.y); if (d < this.r + player.r + 10) { this.collected = true; if (cam?.cueFocus)
cam.cueFocus(this.x, this.y, 25, 1.09); return true; } return false; } update(cam, player) {
if (!this.found && cam.worldPointInView(this.x, this.y, 30)) { this.found = true; if
(cam.cueFocus) cam.cueFocus(this.x, this.y, 45, 1.07); } this.pulseT += 0.02; } //
Optional: collect when close + press E const d = dist(player.x, player.y, this.x, this.y); if
(this.found && !this.collected && d < this.r + player.r + 10) { if (keyIsDown(69)) { // E key
this.collected = true; cam.cueFocus(this.x, this.y, 25, 1.09); } } this.pulseT += 0.02; }
draw() { if (!this.found) return; push(); noStroke(); // soft pulse const pulse = 0.6 + 0.4 *
sin(this.pulseT); const rr = this.r + pulse * 2; // subtle glow ring fill(0, 0, 0, this.collected
? 30 : 55); ellipse(this.x, this.y, rr * 2.6, rr * 2.6); // core fill(0, 0, 0, this.collected ? 90 :
140); ellipse(this.x, this.y, rr * 1.4, rr * 1.4); // tiny mark stroke(255, 220);
strokeWeight(2); line(this.x - 4, this.y, this.x + 4, this.y); line(this.x, this.y - 4, this.x,
this.y + 4); pop(); } } //why are there mistakes
```

Because your class has **duplicate + mis-nested code**. After your `update(cam, player)` function, you accidentally left the *old* "Optional: collect when close + press E" block **outside of any method**, plus you have an extra `this.pulseT += 0.02;` and mismatched braces.

In JavaScript, everything inside a `class { ... }` must be **methods** (or fields). Random statements like:

```
const d = dist(...)
if (...) { ... }
```

cannot just sit inside the class body — that causes syntax errors and/or weird behavior.

✓ Copy-paste: fixed `HiddenSymbol` class (no auto-collect)

Replace your entire class with this:

```
class HiddenSymbol {
constructor(x, y, r = 10) {
  this.x = x;
  this.y = y;
```

```

this.r = r;

this.found = false;      // revealed when camera sees it
this.collected = false; // set true only when player presses E nearby
this.pulseT = random(1000);
}

// Called once when player presses E (from keyPressed)
tryCollect(cam, player) {
    if (!this.found || this.collected) return false;

    const d = dist(player.x, player.y, this.x, this.y);
    if (d < this.r + player.r + 10) {
        this.collected = true;
        if (cam && cam.cueFocus) cam.cueFocus(this.x, this.y, 25, 1.09);
        return true;
    }
    return false;
}

// Called every frame
update(cam, player) {
    // reveal when camera sees it
    if (!this.found && cam.worldPointInView(this.x, this.y, 30)) {
        this.found = true;
        if (cam && cam.cueFocus) cam.cueFocus(this.x, this.y, 45, 1.07);
    }

    this.pulseT += 0.02;
}

draw() {
    if (!this.found) return;

    push();
    noStroke();

    const pulse = 0.6 + 0.4 * sin(this.pulseT);
    const rr = this.r + pulse * 2;

    fill(0, 0, 0, this.collected ? 30 : 55);
    ellipse(this.x, this.y, rr * 2.6, rr * 2.6);

    fill(0, 0, 0, this.collected ? 90 : 140);
    ellipse(this.x, this.y, rr * 1.4, rr * 1.4);

    stroke(255, 220);
    strokeWeight(2);
    line(this.x - 4, this.y, this.x + 4, this.y);
    line(this.x, this.y - 4, this.x, this.y + 4);

    pop();
}
}

```

What I removed (the “mistakes”)

- The old `keyIsDown(69)` collection block (that caused auto-collect / hold-to-collect)
 - The duplicated `this.pulseT += 0.02;`
 - The stray code that was sitting outside methods
-

Required: update `keyPressed()` so E collects (once)

In `sketch.js`, add this inside your existing `keyPressed()`:

```
if (key === "e" || key === "E") {  
    for (const s of level.symbols) {  
        if (s.tryCollect(cam, player)) break; // collect max 1 per press  
    }  
}
```

Now:

- camera “discovers” symbols when they enter view
 - player collects **only** when pressing E near them
-

If you want, paste your current `keyPressed()` and I’ll rewrite the whole function cleanly so it includes jump + reset + E collect without conflicts.