

# Process & Decision Documentation

In this Side Quest 5, I modified the Example 4 code from Week 5. The original game featured a standard lerp camera. To transform it into a meditative camera experience, I added new features such as a “breathing camera” motion to create a calm and reflective atmosphere. Additional shapes were placed outside the camera’s default view for players to discover as they explore the map.

## Role-Based Process Evidence

The screenshot shows the GitHub Changes interface for a repository named 'e76huang\_sidequest\_W5'. The current branch is 'main'. A warning message at the top indicates a change in line endings from 'LF' to 'CRLF'. The changes are listed for four files:

- Player.js:** Changes line 39 from `const start = worldData.playerStart ?? { x: 300, y: 300, speed: 3 };` to `const start = worldData.playerStart ?? { x: 300, y: 300, speed: 0.001 };` and adds a new line `player = new Player(start.x, start.y, start.speed);` below it.
- sketch.js:** Changes line 54 from `let targetX = player.x - width / 2;` and line 55 from `let targetY = player.y - height / 2;` to `// --- Camera target (player-centered) ---` followed by `let targetX = player.x - width / 2;` and `let targetY = player.y - height / 2 + 80;`.
- world.json:** No visible changes.
- WorldLevel.js:** No visible changes.

A summary box on the left indicates 4 changed files and a required summary. A commit button at the bottom says 'Commit 4 files to main'.

Screenshot of GitHub changes and commits.

```
File Edit Selection View Go ... < > Q e76huang_sidequest_W5 JS Player.js JS WorldLevel.js X world.json JS sketch.js # style.css

EXPLORER E76HUANG SIDEQUEST_W5
> libraries
< types\p5
  TS global.d.ts
  TS index.d.ts
  .gitattributes
  index.html
  {} jsonconfig.json
  JS Player.js
  JS sketch.js
  # style.css
  {} world.json
  JS WorldLevel.js

JS WorldLevel.js > WorldLevel > drawHUD
1   class WorldLevel {
27  drawWorld() {
39    // --- Floating / scattered shapes ---
40    // Light blue ellipse
41    fill(173, 216, 230);
42    ellipse(1200, 500, 100, 60);
43
44    // Pink ellipse
45    fill(255, 182, 193);
46    ellipse(1500, 1200, 80, 50);
47
48    // Light green ellipse
49    fill(144, 238, 144);
50    ellipse(1800, 300, 70, 70);
51
52    // Light yellow ellipse
53    fill(255, 255, 224);
54    ellipse(2000, 900, 100, 40);
55
56    // Light purple ellipse
57    fill(216, 191, 216);
58    ellipse(500, 1300, 60, 60);
59
60    // Light magenta rounded rectangle
61    fill(255, 182, 255);
62    rect(2200, 1100, 100, 50, 20);
63
64    // Indigo rounded rectangle (soft)
65    fill(138, 43, 226, 180); // indigo with alpha for softness
66    rect(2100, 700, 120, 60, 25);
67
68
69
```

Screenshot of adding hidden shapes around the map in VScode.

```
1  {
2    "schemaVersion": 1,
3    "world": {
4      "w": 2400,
5      "h": 1600,
6      "bg": [140, 255, 219],
7      "gridStep": 160
8    },
9    "camera": {
10
11      "lerp": 0.05,
12      "breathSpeed": 0.01,
13      "noiseAmp": 3,
14      "noiseSpeed": 0.002,
15    }
}
```

Screenshot of adding new meditative camera characteristics, such as breathing and noise, in VS Code.

#### *GenAI Documentation*

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

**Date Used:** February 23, 2026

**Tool Disclosure:** ChatGPT 5.2

**Purpose of Use:** I used GenAI for brainstorming debugging, code generation and understanding/learning.

**Summary of Interaction:** I used GenAI to understand parts of the original example code. I also used GenAI to brainstorm characteristics of a meditative camera and to help me generate example codes on how to implement it.

**Human Decision Point(s):** I made sure to understand and learn the original example code and what a meditative camera is, as well as the code that GenAI generated. Then I was able to make my own judgment about which code to include. I made my own manual adjustments to control values. I also made decisions about the design and placement of shapes scattered around the map.

**Integrity & Verification Note:** I checked GenAI's output for accuracy by comparing it with the original example code and course materials. I also ensured the suggestions were appropriate, unbiased, and fit with the course concepts before integrating them into my project. I also play-tested the example code that GenAI provided on a live server.

**Scope of GenAI Use:** GenAI was not used to write all the new code through mindless pasting. GenAI did not replace the creative design process or the final judgments about which code was included in the final code.

**Limitations or Misfires:** Certain recommendations and example code generated by GenAI did not always work. Some parts needed to be omitted to prevent errors, and unnecessary code was removed because it did not align with the project requirements or the side quest, or it did not perform as intended.

#### *Summary of Process (Human + Tool)*

Describe what you did, focusing on process rather than outcome. This may include:

- Iterated on meditative camera code with breathing, noise, and gentle zoom.
- Tested and adjusted lerp, breathing amplitude, and camera offsets.
- Debugged and removed GenAI code that caused errors.

- Placed pastel shapes outside default camera view for exploration.

#### *Decision Points & Trade-offs*

Options considered: I considered using only breathing, combining it with subtle noise, adding a deadpan effect, breathing amplitude, different shape types, and gentle zoom.

What changed: I added slow breathing with minimal noise, removed the deadpan and removed amplitude due to jittery motion, adjusted lerp values, and added scattered shapes

Why the choice was made: To create a calm, meditative camera while keeping the code functional and aligned with Side Quest requirements.

#### *Verification & Judgement*

- Play-tested camera and shape placement outside the default camera view.
- Compared GenAI suggestions with the original example code.
- Tested newly added meditative camera code on a live server.

#### *Limitations, Dead Ends, or Open Questions*

- Some GenAI code caused errors and had to be removed.
- Some camera effects simplified to maintain calm, meditative feel.
- Shape placement and motion could be further refined.

## Appendix

### **Prompt #1:**

How to add a meditative camera experience to p5.js code? What are some criteria to follow?

### **ChatGPT Response #1:**

If you want to create a meditative camera experience in p5.js, you're basically designing how the viewer moves and feels inside the space — not just how things look.

Since you're in Digital Arts and care about composition and mood, think of this as designing visual rhythm + emotional pacing, not just camera math.

Here's how to approach it:

#### What Makes a “Meditative” Camera?

A meditative experience usually feels:

- Slow
- Smooth

- Predictable but organic
- Minimal and uncluttered
- Responsive but not chaotic
- Breathing / floating / drifting

### Core Criteria to Follow

#### **1** Slow, Continuous Movement (No Jumps)

Avoid:

```
camera(x, y, z);
```

with sudden position changes.

Instead use interpolation:

```
let targetX = mouseX * 0.1;
```

```
camX = lerp(camX, targetX, 0.02);
```

Rule: Movement should ease in and ease out — never snap.

#### **2** Low Frequency Motion

Meditative motion feels like: Floating, Breathing, Orbiting, Slow drifting

Use sine waves:

```
let camX = sin(frameCount * 0.005) * 200;
```

```
let camY = cos(frameCount * 0.003) * 100;
```

Small numbers = calm, Large multipliers = chaotic

#### **3** Minimal Camera Rotation

Too much rotation = disorientation.

Good: Slow orbital rotation, Gentle tilt, Subtle zoom in/out

Avoid: Rapid spins, Constant shaking, Sudden FOV changes

#### **4** Depth & Parallax (Very Subtle)

Add layered elements moving at different speeds.

Example:

```
translate(0, 0, sin(frameCount * 0.002) * 50);
```

Small depth shifts feel immersive.

## 5 Soft Visual Language

Meditative camera works best with:

- Soft color palettes
- Low contrast
- Gradients
- Transparency
- Fewer objects
- Repetition

Since you're strong in color theory, use: Analogous colors, Desaturated tones, Gradual transitions

## 🧘 Design Principles (More Important Than Code)

Think about:

- Rhythm: Does the camera movement feel like breathing?

You can literally simulate breathing:

```
let breath = sin(frameCount * 0.01) * 20;
```

- Stillness: Meditative ≠ constant movement.  
Include moments where the camera slows down, pauses, barely moves.

## 🧠 Technical Tools in p5.js

If using WEBGL:

```
createCanvas(windowWidth, windowHeight, WEBGL);
```

Camera controls:

- camera()
- orbitControl() (avoid default — too interactive for meditative feel)
- lerp()
- sin() / cos()
- noise() (VERY good for organic motion)

Example using Perlin noise:

```
let offset = frameCount * 0.005;  
let camX = noise(offset) * 200 - 100;
```

Noise feels more natural than sine.

### Example: Calm Floating Camera

```
let t = 0;
```

```
function setup() {  
  createCanvas(windowWidth, windowHeight, WEBGL);  
}  
  
function draw() {
```

```
  background(20, 30, 40);  
  
  t += 0.003;
```

```
  let camX = sin(t) * 200;  
  let camY = cos(t * 0.8) * 100;  
  let camZ = 500;
```

```
  camera(camX, camY, camZ, 0, 0, 0, 0, 1, 0);
```

```
  ambientLight(80);  
  directionalLight(150, 150, 200, 0.5, 1, 0);  
  
  sphere(100);
```

}

This creates: Gentle orbit, Soft lighting, Central focus, No chaos

### ✿ Artistic Criteria Checklist

When evaluating your meditative camera:

- Does it feel slow enough?
- Are transitions smooth?
- Is there visual breathing space?
- Does movement feel intentional?
- Would someone feel calm watching it for 2 minutes?

If yes → you're succeeding.

### ✳ Advanced Ideas (Since You're Into Experimental Media)

You could:

- Tie camera movement to microphone input (slow ambient sound)
- Use very subtle mouse influence (not direct control)
- Add grain or blur for softness
- Slowly shift color temperature over time
- Create depth-of-field effect

If you want, show me your current p5.js sketch and I'll help you redesign the camera to feel more meditative and intentional (composition-wise, not just technically).

---

### **Prompt #2:**

Where can I add those attributes to the code?

### **ChatGPT Response #2:**

This is already very cleanly structured 🤝 — your camera is nicely separated from world + player logic.

To make it more meditative, you don't want to randomly insert things. You want to enhance these specific layers:

- Camera smoothing behavior

- Camera “breathing” / drift
- Soft zone (don’t instantly center on player)
- Optional noise-based motion

Since your system is JSON-driven, the correct place to add attributes is:

- 👉 world.json
- 👉 then read them inside WorldLevel
- 👉 then apply them inside the camera section in draw()

#### 👉 Step 1 — Add Camera Attributes to world.json

```
{
  "camLerp": 0.06,
  "camBreathAmp": 10,
  "camBreathSpeed": 0.01,
  "camNoiseAmp": 6,
  "camNoiseSpeed": 0.002,
  "camDeadZone": 80
}
```

These are your meditative knobs.

#### 👉 Step 2 — Store Them in WorldLevel

```
class WorldLevel {
  constructor(data) {
    this.w = data.worldWidth;
    this.h = data.worldHeight;
    this.camLerp = data.camLerp ?? 0.08;
  }
}
```

// NEW

```
this.camBreathAmp = data.camBreathAmp ?? 0;  
this.camBreathSpeed = data.camBreathSpeed ?? 0.01;  
this.camNoiseAmp = data.camNoiseAmp ?? 0;  
this.camNoiseSpeed = data.camNoiseSpeed ?? 0.002;  
this.camDeadZone = data.camDeadZone ?? 0;  
}  
}
```

### 💡 Step 3 — Modify Camera Section in draw()

Replace your current camera section with this upgraded version:

```
// Target camera (center on player)  
  
let targetX = player.x - width / 2;  
  
let targetY = player.y - height / 2;  
  
// --- DEAD ZONE (soft center area) ---  
  
if (level.camDeadZone > 0) {  
  
    const dx = player.x - (camX + width / 2);  
    const dy = player.y - (camY + height / 2);  
  
    if (abs(dx) < level.camDeadZone) {  
        targetX = camX;  
    }  
    if (abs(dy) < level.camDeadZone) {  
        targetY = camY;  
    }  
}
```

```

// Clamp

const maxCamX = max(0, level.w - width);
const maxCamY = max(0, level.h - height);
targetX = constrain(targetX, 0, maxCamX);
targetY = constrain(targetY, 0, maxCamY);

// Smooth follow

camX = lerp(camX, targetX, level.camLerp);
camY = lerp(camY, targetY, level.camLerp);

// --- BREATHING EFFECT ---

const breathX = sin(frameCount * level.camBreathSpeed) * level.camBreathAmp;
const breathY = cos(frameCount * level.camBreathSpeed) * level.camBreathAmp;

// --- NOISE DRIFT ---

const noiseOffset = frameCount * level.camNoiseSpeed;
const noiseX = (noise(noiseOffset) - 0.5) * level.camNoiseAmp;
const noiseY = (noise(noiseOffset + 1000) - 0.5) * level.camNoiseAmp;

// Apply meditative offsets

camX += breathX + noiseX;
camY += breathY + noiseY;

```

### ✿ What Each Attribute Does (Game Feel Logic)

camLerp: Lower = floaty, dreamy. Higher = tight, responsive  
 camDeadZone: Prevents camera from constantly adjusting.  
 camBreathAmp: Adds gentle oscillation.  
 camNoiseAmp: Adds organic drift.

### Why This Works (Design Perspective)

Transforms camera from “Tracking device” → “Emotional lens.” Player feels grounded; world feels alive; movement feels intentional.

### Advanced Optional: Ease Camera Speed When Player Stops

```
let moving = keyIsPressed;  
  
let camLerp = moving ? level.camLerp : level.camLerp * 0.5;  
  
camX = lerp(camX, targetX, camLerp);  
  
camY = lerp(camY, targetY, camLerp);
```

### Conceptual framing for GBDA critique:

“The camera is treated not as a reactive mechanic, but as a breathing presence within the world, introducing subtle organic drift to soften mechanical navigation.”

---

### **Prompt #3:**

There are some lines of code that I didn’t add. Should I adjust the world JSON like this?

```
{  
  
  "schemaVersion": 1,  
  
  "world": {  
  
    "w": 2400,  
  
    "h": 1600,  
  
    "bg": [235, 235, 235],  
  
    "gridStep": 160  
  
  },  
  
  "camera": {  
  
    "lerp": 0.12  
  
  },  
  
  "playerStart": {  
  
    "x": 300,
```

```
"y": 300,  
"speed": 3  
},  
"obstacles": [{ "x": 40, "y": 40, "w": 80, "h": 80, "r": 10 }]  
}
```

### ChatGPT Response #3:

Yes — your JSON is already structured nicely, but for a meditative camera, you can extend the camera object with the extra attributes we talked about. You don't need to touch the world or playerStart sections — just the camera section.

Here's an updated version with meditative knobs added:

```
{  
  "schemaVersion": 1,  
  "world": {  
    "w": 2400,  
    "h": 1600,  
    "bg": [235, 235, 235],  
    "gridStep": 160  
  },  
  "camera": {  
    "lerp": 0.12,  
    "breathAmp": 10,  
    "breathSpeed": 0.01,  
    "noiseAmp": 6,  
    "noiseSpeed": 0.002,  
    "deadZone": 80  
  },  
  "playerStart": {
```

```
"x": 300,  
"y": 300,  
"speed": 3  
,  
"obstacles": [  
{ "x": 40, "y": 40, "w": 80, "h": 80, "r": 10 }  
]  
}
```

 What changed:

- Added breathAmp & breathSpeed for gentle oscillation.
- Added noiseAmp & noiseSpeed for subtle organic drift.
- Added deadZone so the camera won't constantly snap when the player moves slightly.

Then in your p5.js code, read these values in WorldLevel (like I showed earlier) and apply them in the camera calculation.

If you want, I can rewrite the camera part of your draw() function fully using this updated JSON so it's ready to drop in — it will make the movement floaty and meditative. Do you want me to do that?