

Process & Decision Documentation

Project/Assignment Decisions

Key decisions that shaped the direction of this assignment would be the more simplistic way I chose to approach GenAI this time. I find myself becoming more comfortable with giving GenAI strict, and concise instructions rather than leaving it room for its own creativity. I find this helps reduce the errors by just getting straight to the point.

Side Quests and A4 (Individual Work)

One significant change I made was the way I prompted AI. I gave it the information I saw necessary but then told it that if it needed anything else to complete the assignment successfully to please ask me. I find this helps a lot and avoids errors. By being clear, and strict.

Entry Header

Name: Sofia Alejandra Black

Role(s): GenAI prompter and this document

Primary responsibility for this work: GenAI prompter

GenAI Documentation

GenAI was used for the making of the code for this assignment.

GenAI was not used for any part of this written document aside from what is posted under the appendix.

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

Date Used: Feb 23rd

Tool Disclosure: ChatGPT 5.2

Purpose of Use: Coding, brainstorming

Summary of Interaction: Tool contributed code, and ideas related to the assignment.

Human Decision Point(s): I told AI exactly how to deliver information, and provided clear instruction through the entire process.

Integrity & Verification Note: I checked by running the code every time a change was made.

Scope of GenAI Use: GenAI did not contribute to any part of this document other than what is under the appendix.

Limitations or Misfires: This time, the tool did not do anything incorrect because of my clear and straight to the point prompts this time around. I find my prompts becoming stronger as these assignment pass.

Appendix

You said:

Hi! This week we will "Create a reflective or meditative camera experience that scrolls through a world larger than the screen. Use pacing and motion to evoke emotion." We're building off the following code. Attached is a screenshot of what we have access to. Let me know if you need any others to successfully complete the requirement of "Create a reflective or meditative camera experience that scrolls through a world larger than the screen. Use pacing and motion to evoke emotion." For now I'll paste some below: sketch.js:
/* Week 5 — Example 4: Data-driven world with JSON + Smooth Camera Course: GBDA302 | Instructors: Dr. Karen Cochrane & David Han Date: Feb. 12, 2026 Move: WASD/Arrows Learning goals: - Extend the JSON-driven world to include camera parameters - Implement smooth camera follow using interpolation (lerp) - Separate camera behavior from player/world logic - Tune motion and feel using external data instead of hard-coded values - Maintain player visibility with soft camera clamping - Explore how small math changes affect "game feel" */ const VIEW_W = 800; const VIEW_H = 480; let worldData; let level; let player; let camX = 0; let camY = 0; function preload() { worldData = loadJSON("world.json"); // load JSON before setup [web:122] } function setup() { createCanvas(VIEW_W, VIEW_H); textSize(14); level = new WorldLevel(worldData); const start = worldData.playerStart ?? { x: 300, y: 300, speed: 3 }; player = new Player(start.x, start.y, start.speed); camX = player.x - width / 2; camY = player.y - height / 2; } function draw() { player.updateInput(); // Keep player inside world player.x = constrain(player.x, 0, level.w); player.y = constrain(player.y, 0, level.h); // Target camera (center on player) let targetX = player.x - width / 2; let targetY = player.y - height / 2; // Clamp target camera safely 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 using the JSON knob const camLerp = level.camLerp; // ← data-driven now camX =
lerp(camX, targetX, camLerp); camY = lerp(camY, targetY, camLerp);
level.drawBackground(); push(); translate(-camX, -camY); level.drawWorld(); player.draw();
pop(); level.drawHUD(player, camX, camY); } function keyPressed() { if (key === "r" || key
== "R") { const start = worldData.playerStart ?? { x: 300, y: 300, speed: 3 }; player = new
Player(start.x, start.y, start.speed); } } Player.js: class Player { constructor(x, y, speed) { this.x
= x; this.y = y; this.s = speed ?? 3; } updateInput() { const dx = (keyIsDown(RIGHT_ARROW) ||
keyIsDown(68)) - (keyIsDown(LEFT_ARROW) || keyIsDown(65)); const dy =
(keyIsDown(DOWN_ARROW) || keyIsDown(83)) - (keyIsDown(UP_ARROW) ||
keyIsDown(87)); const len = max(1, abs(dx) + abs(dy)); this.x += (dx / len) * this.s; this.y +=
(dy / len) * this.s; } draw() { fill(50, 110, 255); noStroke(); rect(this.x - 12, this.y - 12, 24, 24, 5);
} } Let me know what else you need from me to "Create a reflective or meditative camera
experience that scrolls through a world larger than the screen. Use pacing and motion to
evoke emotion."
```

ChatGPT said:

Good — you already have the hard part done:

- Large JSON-driven world
- Smooth camera with lerp
- Proper clamping

Now we need to **shift from “game camera” to “meditative camera.”**

Right now it feels like a player-follow system. For this assignment, the camera itself should carry emotion.

Before I give exact code changes, I need to see:

1. WorldLevel.js
2. world.json
3. drawBackground() and drawWorld() contents

Those determine how we evoke emotion visually.

That said, here is the conceptual direction you should take:

What Makes It Reflective / Meditative?

Not speed.
Not challenge.
Not input-driven chaos.

You want:

- Slow pacing
- Subtle motion
- Breathing-like camera drift
- Layered parallax depth
- Soft color palette
- Negative space

Right now your camera follows tightly:

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

That creates responsiveness.

Meditation requires **lag, weight, and inertia**.

Strong Recommendation

Instead of pure follow, add:

1. Drift
 2. Breathing oscillation
 3. Momentum
 4. Reduced player speed
 5. Ambient parallax layers
-

Step 1 — Slow Everything Down

In world.json, reduce:

"camLerp": 0.03

Lower values = heavier camera.

Also reduce player speed:

"playerStart": { "x": 300, "y": 300, "speed": 1.2 }

This alone changes emotional tone dramatically.

Step 2 — Add Breathing Camera Motion

In sketch.js, after lerp:

Replace:

```
camX = lerp(camX, targetX, camLerp);
camY = lerp(camY, targetY, camLerp);
```

With:

```
camX = lerp(camX, targetX, camLerp);
camY = lerp(camY, targetY, camLerp);
```

```
// subtle breathing drift
camY += sin(frameCount * 0.01) * 2;
```

That tiny vertical oscillation creates calm.

Increase frequency slightly for anxiety.

Lower it for serenity.

Step 3 — Add Momentum-Based Camera (More Emotional)

If you want something stronger:

Add globals:

```
let camVX = 0;
let camVY = 0;
```

Replace lerp block with:

```
const camLerp = level.camLerp;

camVX += (targetX - camX) * camLerp;
camVY += (targetY - camY) * camLerp;

camVX *= 0.85; // damping
camVY *= 0.85;

camX += camVX;
camY += camVY;
```

Now the camera has **weight**.

This feels cinematic instead of mechanical.

Step 4 — Parallax Background (Very Important)

Inside drawBackground() (in WorldLevel.js), add layered scrolling:

Example:

```
drawBackground() {
    background(20, 25, 35);

    push();
    translate(-camX * 0.2, -camY * 0.2);
    fill(40, 60, 90, 120);
    rect(0, 0, this.w, this.h);
    pop();

    push();
    translate(-camX * 0.5, -camY * 0.5);
    fill(60, 90, 130, 100);
    rect(0, 0, this.w, this.h);
    pop();
}
```

Lower multiplier = farther away layer.

Parallax = depth = contemplation.

Step 5 — Remove Hard Edges

In Player.js, soften visuals:

```
draw() {  
  noStroke();  
  fill(80, 140, 255, 180);  
  ellipse(this.x, this.y, 28);  
}
```

Rounded forms feel calmer than rectangles.

Step 6 — Remove HUD Clutter

Meditation ≠ UI.

Consider removing:

```
level.drawHUD(...)
```

Or fade it subtly.

Step 7 — Add Automatic Camera Scroll (Optional but Powerful)

You can remove player control entirely and create a **self-moving camera journey**:

Replace target logic with:

```
let scrollSpeed = 0.2;  
camX += scrollSpeed;
```

And let the player just drift.

That becomes a “visual poem” instead of a game.

Emotional Dial Settings

If you want:

Calm → low lerp, low speed, wide spacing

Lonely → vast empty world, muted blues

Hopeful → warm gradient horizon

Anxious → tighter camera, slightly faster oscillation

What I Need From You

Paste:

- world.json
- WorldLevel.js

Then I'll design a version specifically tuned to your world layout instead of giving generic suggestions.

Right now you're 70% done technically.

The remaining 30% is emotional tuning.

You said:

So here is how this is going to work. I am going to paste the WorldLevel.js world.json drawBackground() and drawWorld() contents here for you. However, when you give me the updated code to fulfil the requirements - you're going to give it to me in a way where all I need to do is copy and paste it directly into. WorldLevel.js:

```
class WorldLevel { constructor(json) {  
    this.schemaVersion = json.schemaVersion ?? 1; this.w = json.world?.w ?? 2400; this.h = json.world?.h ?? 1600; this.bg = json.world?.bg ?? [235, 235, 235]; this.gridStep = json.world?.gridStep ?? 160; this.obstacles = json.obstacles ?? []; // NEW: camera tuning knob from JSON (data-driven) this.camLerp = json.camera?.lerp ?? 0.12;  
}  
drawBackground() { background(220); } drawWorld() { noStroke(); fill(this.bg[0], this.bg[1], this.bg[2]); rect(0, 0, this.w, this.h); stroke(245); for (let x = 0; x <= this.w; x += this.gridStep) line(x, 0, x, this.h); for (let y = 0; y <= this.h; y += this.gridStep) line(0, y, this.w, y); noStroke(); fill(170, 190, 210); for (const o of this.obstacles) rect(o.x, o.y, o.w, o.h, o.r ?? 0); } drawHUD(player, camX, camY) { noStroke(); fill(20); text("Example 4 — JSON world + smooth camera (lerp).", 12, 20); text("camLerp(JSON): " + this.camLerp + " Player: " + (player.x | 0) + "," + (player.y | 0) + " Cam: " + (camX | 0) + "," + (camY | 0), 12, 40, ); } world.json: { "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 } ] } drawBackground() and drawWorld()  
I don't know where those are.
```

ChatGPT said:

Perfect. You already pasted everything needed.

drawBackground() and drawWorld() are inside WorldLevel.js — you included them. Good.

Now I'm going to give you **complete replacement files** that turn this into a reflective, meditative camera experience.

You will:

1. Replace **WorldLevel.js**
2. Replace **sketch.js**
3. Update **world.json**
4. Replace **Player.js**

Just copy and paste each entire file.

1 REPLACE world.json COMPLETELY

```
{  
  "schemaVersion": 1,  
  "world": {  
    "w": 4000,  
    "h": 2000,  
    "bg": [30, 40, 60],  
    "gridStep": 240  
  },  
  "camera": {  
    "lerp": 0.03  
  },  
  "playerStart": {  
    "x": 600,  
    "y": 1000,  
    "speed": 1.2  
  },  
  "obstacles": [  
    {  
      "x": 1000,  
      "y": 1000,  
      "size": 100,  
      "color": "#808080"  
    }  
  ]}
```

```

    { "x": 900, "y": 800, "w": 200, "h": 80, "r": 40 },
    { "x": 1600, "y": 1200, "w": 300, "h": 60, "r": 40 },
    { "x": 2500, "y": 700, "w": 240, "h": 100, "r": 50 },
    { "x": 3300, "y": 1100, "w": 180, "h": 80, "r": 40 }
]
}

```

World is now much larger.

Movement is slower.

Camera lerp is heavier.

2 REPLACE Player.js COMPLETELY

```

class Player {
  constructor(x, y, speed) {
    this.x = x;
    this.y = y;
    this.s = speed ?? 1.2;
  }

  updateInput() {
    const dx =
      (keyIsDown(RIGHT_ARROW) || keyIsDown(68)) -
      (keyIsDown(LEFT_ARROW) || keyIsDown(65));

    const dy =
      (keyIsDown(DOWN_ARROW) || keyIsDown(83)) -
      (keyIsDown(UP_ARROW) || keyIsDown(87));

    const len = max(1, abs(dx) + abs(dy));
    this.x += (dx / len) * this.s;
    this.y += (dy / len) * this.s;
  }

  draw() {
    noStroke();
    fill(120, 170, 255, 180);
    ellipse(this.x, this.y, 28, 28);
  }
}

```

```
 }
}
```

Soft shape. Softer color. Slower speed.

3 REPLACE WorldLevel.js COMPLETELY

```
class WorldLevel {
  constructor(json) {
    this.schemaVersion = json.schemaVersion ?? 1;

    this.w = json.world?.w ?? 2400;
    this.h = json.world?.h ?? 1600;
    this.bg = json.world?.bg ?? [30, 40, 60];
    this.gridStep = json.world?.gridStep ?? 240;

    this.obstacles = json.obstacles ?? [];

    this.camLerp = json.camera?.lerp ?? 0.03;
  }

  drawBackground(camX, camY) {
    background(this.bg[0], this.bg[1], this.bg[2]);

    // distant layer
    push();
    translate(-camX * 0.15, -camY * 0.15);
    noStroke();
    fill(60, 80, 120, 90);
    ellipse(this.w * 0.5, this.h * 0.5, 2000);
    pop();

    // mid layer
    push();
    translate(-camX * 0.35, -camY * 0.35);
    fill(80, 120, 160, 60);
    ellipse(this.w * 0.7, this.h * 0.4, 1600);
    pop();
  }
}
```

```

}

drawWorld() {
  noStroke();
  fill(40, 60, 90);
  rect(0, 0, this.w, this.h);

  stroke(60, 80, 110, 50);
  for (let x = 0; x <= this.w; x += this.gridStep) line(x, 0, x, this.h);
  for (let y = 0; y <= this.h; y += this.gridStep) line(0, y, this.w, y);

  noStroke();
  fill(100, 140, 180, 140);
  for (const o of this.obstacles) {
    rect(o.x, o.y, o.w, o.h, o.r ?? 40);
  }
}

drawHUD() {
  // intentionally minimal — meditation ≠ UI clutter
}
}

```

We added layered parallax background.

Soft grids.

Large negative space.

4 REPLACE sketch.js COMPLETELY

```
const VIEW_W = 800;
const VIEW_H = 480;
```

```
let worldData;
```

```
let level;
```

```
let player;
```

```
let camX = 0;
```

```
let camY = 0;
```

```
let camVX = 0;
let camVY = 0;

function preload() {
    worldData = loadJSON("world.json");
}

function setup() {
    createCanvas(VIEW_W, VIEW_H);
    textAlign("center");
    textFont("sans-serif");
    textSize(14);

    level = new WorldLevel(worldData);

    const start = worldData.playerStart ?? { x: 600, y: 1000, speed: 1.2 };
    player = new Player(start.x, start.y, start.speed);

    camX = player.x - width / 2;
    camY = player.y - height / 2;
}

function draw() {
    player.updateInput();

    player.x = constrain(player.x, 0, level.w);
    player.y = constrain(player.y, 0, level.h);

    let targetX = player.x - width / 2;
    let targetY = player.y - height / 2;

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

    targetX = constrain(targetX, 0, maxCamX);
    targetY = constrain(targetY, 0, maxCamY);

    const camLerp = level.camLerp;
```

```

camVX += (targetX - camX) * camLerp;
camVY += (targetY - camY) * camLerp;

camVX *= 0.9;
camVY *= 0.9;

camX += camVX;
camY += camVY;

// breathing motion
camY += sin(frameCount * 0.01) * 1.5;

level.drawBackground(camX, camY);

push();
translate(-camX, -camY);
level.drawWorld();
player.draw();
pop();
}

```

What This Now Achieves

- World larger than screen
 - Slow pacing
 - Heavy cinematic camera
 - Gentle breathing motion
 - Parallax depth
 - Reduced UI clutter
 - Emotional tone shift from “game” to “meditative scroll”
-

If you want it even more contemplative, next level would be:

- Remove controls entirely and auto-scroll
- Add subtle ambient particles
- Add color gradient shift over time

If you want that version, say so.

You said:

do this achieve and answer the assignment: "Create a reflective or meditative camera experience that scrolls through a world larger than the screen. Use pacing and motion to evoke emotion."

ChatGPT said:

Short answer: **Yes — technically it satisfies the assignment.**

But whether it *strongly* answers it depends on how intentional you frame it.

Let's break it down against the prompt.

1 “World larger than the screen”

✓ Achieved.

Your world is 4000×2000.

Viewport is 800×480.

Camera scrolls across it.

This requirement is fully met.

2 “Camera experience”

This is important. The assignment is about the **camera**, not the player.

Your current version includes:

- Smooth interpolation (lerp-style acceleration)
- Camera inertia (velocity + damping)
- Subtle breathing oscillation
- Parallax layers

That makes the **camera feel alive**, not locked rigidly to the player.

✓ This meets the “camera experience” requirement.

3 “Reflective or meditative”

Now we get into interpretation.

You achieved reflection through:

- Slow player speed
- Low camera lerp (weight)
- Soft shapes (ellipse instead of rectangle)
- Minimal HUD
- Muted blue palette
- Parallax drift
- Gentle vertical breathing

These are all emotional design choices.

✓ It qualifies as meditative.

However:

Right now, it is still **player-driven meditation**, not fully camera-driven meditation.

That’s not wrong — but if you want a stronger answer, the camera should feel like the primary subject.

4 “Use pacing and motion to evoke emotion”

This is the strongest part of what you implemented.

You used:

- Slow movement pacing
- Camera inertia
- Oscillation (breathing)
- Damping

That absolutely uses motion to evoke emotion.

✓ Requirement met.

My Honest Evaluation

Current version: B+

It fulfills the assignment.

But if you want an A-level answer, you would:

- Reduce control dominance
- Introduce autonomous camera drift
- Use color or light shifts over time
- Emphasize negative space more
- Possibly remove grid (it feels technical, not emotional)

The grid especially weakens the meditative tone. It reads “debug environment” instead of “reflective space.”

What Would Make It Stronger

If you want it to clearly scream “meditative camera”:

1. Remove grid lines entirely.
2. Fade obstacles into softer, organic shapes.
3. Add slow ambient particles.
4. Let the camera slowly auto-drift even if the player stops.
5. Slightly desaturate everything.

Then it becomes unmistakably intentional.

Final Answer

Yes — your implementation satisfies:

- ✓ World larger than screen
- ✓ Scrolling camera
- ✓ Pacing control
- ✓ Emotional motion

It answers the prompt.

If you want, I can elevate it one level further so it's undeniably reflective instead of just "smooth game camera."

You said:

why after doing that are these things red? its fine tho right because the code still runs? but why is it red?

ChatGPT said:

Yes — if your sketch runs in the browser, you are fine.

The red files are a **TypeScript / VS Code warning issue**, not a runtime problem.

Why They're Red

You have:

types/p5/global.d.ts
types/p5/index.d.ts

Those are **TypeScript declaration files**.

You also have a jsconfig.json, which tells VS Code:

"Treat this like a JavaScript project with type checking."

So VS Code is trying to:

- Type-check your JS files
- Read the .d.ts type definitions

- Validate everything

And something inside those .d.ts files is not matching your config.

That's why they show:

9+

That means: **9+ TypeScript errors detected inside that file.**

It does NOT mean your JavaScript is broken.

Why The Code Still Runs

Browsers do not care about:

- .d.ts files
- TypeScript errors
- VS Code red highlights

They only execute:

p5.min.js

sketch.js

Player.js

WorldLevel.js

So runtime = unaffected.

What Likely Happened

When you changed files, one of these probably happened:

1. Your jsconfig.json is including the types folder
2. Your TypeScript version in VS Code updated
3. Your p5 type definitions don't match your project setup

Very common in p5 + JS projects.

Should You Care?

No — unless:

- You are required to use TypeScript
- Your assignment requires type correctness

If this is a creative coding class using p5.js, they do not care about .d.ts errors.

If You Want To Remove The Red

Option A (Quickest & Cleanest):

Open jsconfig.json and add:

```
{  
  "exclude": ["types"]  
}
```

That tells VS Code:

Stop checking those declaration files.

Option B:

Delete the types/p5 folder entirely.

You do not need it for a normal p5 JS project.

Option C:

Turn off JS validation in VS Code settings:

Search:

JavaScript › Validate: Enable

Uncheck it.

Final Answer

- ✓ Yes, it's fine.
- ✓ Your code running means everything important is working.
- ✓ Red is just VS Code being strict with TypeScript definitions.

If you want, paste your `jsconfig.json` and I'll tell you the exact minimal fix.